
DataRobot Python API Documentation

Release 2.22.2

DataRobot, Inc.

Mar 31, 2021

Contents

1	Getting Started	1
1.1	Installation	1
1.2	Configuration	2
1.3	Common Issues	3
2	Configuration	5
2.1	Configurable Variables	5
2.2	Proxy support	6
3	QuickStart	7
4	Datasets	9
4.1	Creating A Dataset	9
4.2	Getting Information From A Dataset	10
4.3	Retrieving Datasets	11
4.4	Managing Datasets	12
4.5	Managing Dataset Featurelists	12
5	Projects	15
5.1	Create a Project	15
5.2	Select Modeling Parameters	16
5.3	Start Modeling	17
5.4	Interact with a Project	18
5.5	Further reading	21
6	Datetime Partitioned Projects	23
6.1	Setting Up a Datetime Partitioned Project	23
6.2	Modeling with a Datetime Partitioned Project	25
6.3	Dates, Datetimes, and Durations	25
7	Time Series Projects	27
7.1	Setting Up A Time Series Project	27
7.2	Modeling Data and Time Series Features	30
7.3	Making Predictions	30
7.4	Calendars	31
7.5	Prediction Intervals	32

8	Visual AI Projects	35
8.1	Create a Visual AI Project	35
8.2	Start Modeling	36
8.3	Interact with a Visual AI Project	36
9	Unsupervised Projects (Anomaly Detection)	41
9.1	Creating Unsupervised Projects	41
9.2	Creating Time Series Unsupervised Projects	41
9.3	Unsupervised Project Metrics	42
9.4	Assessing Unsupervised Anomaly Detection Models on External Test Set	42
9.5	Requesting External Scores and Insights (Time Series)	42
9.6	Requesting External Scores and Insights for AutoML models	43
9.7	Retrieving External Scores and Insights	43
10	Blueprints	45
10.1	Quick Reference	45
10.2	List Blueprints	45
10.3	Get a blueprint	46
10.4	Get a blueprint chart	46
10.5	Get a blueprint documentation	46
10.6	Blueprint Attributes	46
10.7	Create a Model from a Blueprint	47
11	Models	49
11.1	Start Training a Model	49
11.2	List Finished Models	49
11.3	Retrieve a Known Model	50
11.4	Train a Model on a Different Sample Size	51
11.5	Cross-Validating a Model	51
11.6	Find the Features Used	51
11.7	Feature Impact	52
11.8	Predict new data	52
11.9	Model IDs Vs. Blueprint IDs	52
11.10	Model parameters	52
11.11	Create a Blender	53
11.12	Lift chart retrieval	53
11.13	ROC curve retrieval	53
11.14	Residuals chart retrieval	53
11.15	Word Cloud	54
11.16	Scoring Code	54
11.17	Get a model blueprint chart	54
11.18	Get a model missing values report	54
11.19	Get a blueprint documentation	56
11.20	Request training predictions	56
11.21	Advanced Tuning	56
11.22	SHAP Impact	57
11.23	Number of Iterations Trained	57
12	Jobs	59
12.1	Checking the Contents of the Queue	59
12.2	Cancelling a Job	59
12.3	Retrieving Results From a Job	60
13	ModelJobs	61
13.1	Get an existing ModelJob	61

13.2	Get created model	62
13.3	wait_for_async_model_creation function	62
14	Predictions	63
14.1	Starting predictions generation	63
14.2	Listing Predictions	64
14.3	Get an existing PredictJob	64
14.4	Get generated predictions	65
14.5	Wait for and Retrieve results	65
14.6	Get previously generated predictions	65
15	Prediction Explanations	67
15.1	Quick Reference	67
15.2	List Prediction Explanations	68
15.3	Initialize Prediction Explanations	68
15.4	Compute Prediction Explanations	68
15.5	Retrieving Prediction Explanations	69
15.6	Adjusted Predictions In Prediction Explanations	70
15.7	Deprecated Reason Codes Interface	70
15.8	SHAP based prediction explanations	70
16	Batch Predictions	73
16.1	Scoring local CSV files	73
16.2	Scoring from and to S3	74
16.3	Wiring a Batch Prediction Job manually	74
16.4	Copying a previously submitted job	78
17	DataRobot Prime	79
17.1	Approximate a Model	79
17.2	Prime Models vs. Models	79
17.3	Retrieving Code from a PrimeModel	80
18	Rating Table	81
18.1	Download A Rating Table	81
18.2	Uploading A Rating Table	82
19	Training Predictions	83
19.1	Quick reference	83
20	Monotonic Constraints	85
20.1	Creating featurelists	85
20.2	Specify default monotonic constraints for a project	86
20.3	Retrieve models and blueprints using monotonic constraints	86
20.4	Train a model with specific monotonic constraints	87
21	Database Connectivity	89
21.1	Creating Drivers	90
21.2	Creating DataStores	90
21.3	Creating DataSources	90
21.4	Creating Projects	91
21.5	Creating Predictions	91
22	Model Recommendation	93
22.1	Retrieve all recommendations	94
22.2	Retrieve a default recommendation	94

22.3	Retrieve a specific recommendation	94
22.4	Get recommended model	94
23	Sharing	95
23.1	Access Levels	95
23.2	Examples	96
24	Deployments	99
24.1	Manage Deployments	99
24.2	Model Replacement	101
24.3	Monitoring	102
24.4	Settings	105
25	Custom Models	107
25.1	Manage Execution Environments	107
25.2	Manage Custom Models	110
25.3	Manage Custom Model Versions	112
25.4	Preparing a Custom Model Version for Use	114
25.5	Manage Custom Model Tests	115
25.6	Manage Custom Inference Images	117
26	Compliance Documentation	121
26.1	Generate and Download	121
27	Compliance Documentation Template	123
27.1	Creating a Custom Template	125
28	Credentials	127
28.1	List credentials	127
28.2	Basic credentials	127
28.3	S3 credentials	128
28.4	OAuth credentials	128
29	External Testset	129
29.1	Requesting External Scores and Insights	129
29.2	Retrieving External Metric Scores and Insights	129
29.3	Retrieving External Metric Scores	130
29.4	Retrieving External Lift Chart	130
29.5	Retrieving External Multiclass Lift Chart	130
29.6	Retrieving External ROC Curve	130
29.7	Retrieving Multiclass Confusion Matrix	130
29.8	Retrieving Residuals Chart	131
30	Feature Discovery	133
30.1	Register Primary Dataset to start Project	133
30.2	Register Secondary Dataset(s) in AI Catalog	133
30.3	Create Relationships Configuration	133
30.4	Create Feature Discovery Project	134
31	Relationships Configuration	137
31.1	Create Relationships Configuration	137
31.2	Retrieving Relationships Configuration	138
31.3	Updating details of Relationships Configuration	140
31.4	Delete Relationships Configuration	140

32	API Reference	143
32.1	Advanced Options	143
32.2	Batch Predictions	145
32.3	Blueprint	151
32.4	Calendar File	153
32.5	Compliance Documentation Templates	157
32.6	Compliance Documentation	159
32.7	Confusion Chart	160
32.8	Credentials	161
32.9	Custom Models	164
32.10	Database Connectivity	180
32.11	Datasets	190
32.12	Deployment	197
32.13	External Scores and Insights	214
32.14	Feature	217
32.15	Feature Engineering	224
32.16	Feature List	229
32.17	Job	233
32.18	Lift Chart	239
32.19	Missing Values Report	240
32.20	Models	241
32.21	ModelJob	342
32.22	Pareto Front	345
32.23	Partitioning	346
32.24	PayoffMatrix	361
32.25	PredictJob	364
32.26	Prediction Dataset	367
32.27	Prediction Explanations	369
32.28	Predictions	377
32.29	PredictionServer	379
32.30	Ruleset	380
32.31	PrimeFile	380
32.32	Project	381
32.33	VisualAI	406
32.34	Feature Association	409
32.35	Feature Association Matrix Details	409
32.36	Feature Association Featurelists	409
32.37	Rating Table	410
32.38	Reason Codes (Deprecated)	411
32.39	Recommended Models	417
32.40	ROC Curve	418
32.41	SharingAccess	418
32.42	Training Predictions	419
32.43	Word Cloud	423
32.44	Feature Discovery / Safer	424
32.45	SHAP	429
33	Examples	431
34	Changelog	433
34.1	2.22.2	433
34.2	2.22.1	433
34.3	2.21.0	435
34.4	2.20.0	438

34.5	2.19.0	441
34.6	2.18.0	442
34.7	2.17.0	443
34.8	2.16.0	444
34.9	2.15.1	445
34.10	2.15.0	446
34.11	2.14.2	447
34.12	2.14.1	447
34.13	2.14.0	447
34.14	2.13.0	449
34.15	2.12.0	450
34.16	2.11.0	450
34.17	2.9.0	452
34.18	2.8.1	452
34.19	2.8.0	453
34.20	2.7.2	454
34.21	2.7.1	454
34.22	2.7.0	454
34.23	2.6.1	455
34.24	2.6.0	455
34.25	2.5.1	456
34.26	2.5.0	457
34.27	2.4.0	457
34.28	2.3.0	458
34.29	2.2.33	460
34.30	2.2.32	461
34.31	2.1.31	462
34.32	2.1.30	462
34.33	2.1.29	462
34.34	2.1.28	462
34.35	2.0.27	464
34.36	0.2.26	464
34.37	0.2.25	464
34.38	0.2.24	465
34.39	0.1.24	465
34.40	0.1.23	465
34.41	0.1.22	465
34.42	0.1.21	465
34.43	0.1.20	465
34.44	0.1.19	465
34.45	0.1.18	466
34.46	0.1.17	466
34.47	0.1.16	466
34.48	0.1.15	466
34.49	0.1.14	466
34.50	0.1.13	466
34.51	0.1.12	467
34.52	0.1.11	467
34.53	0.1.10	467
34.54	0.1.9	467
35	Indices and tables	469
	Python Module Index	471

1.1 Installation

You will need the following

- Python 2.7 or 3.4+
- DataRobot account
- pip

1.1.1 Installing for Cloud DataRobot

If you are using the cloud version of DataRobot, the easiest way to get the latest version of the package is:

```
pip install datarobot
```

Note: If you are not running in a Python `virtualenv`, you probably want to use `pip install --user datarobot`.

1.1.2 Installing for an On-Site Deploy

If you are using an on-site deploy of DataRobot, the latest version of the package is not the most appropriate for you. Contact your CFDS for guidance on the appropriate version range.

```
pip install "datarobot>=${MIN_VERSION},<${EXCLUDE_VERSION}"
```

For some particular installation of DataRobot, the correct value of `$(MIN_VERSION)` could be 2.0 with an `$(EXCLUDE_VERSION)` of 2.3. This ensures that all the features the client expects to be present on the backend will always be correct.

Note: If you are not running in a Python `virtualenv`, you probably want to use `pip install --user "datarobot>=$(MIN_VERSION) , <$(MAX_VERSION)"`.

1.2 Configuration

Each authentication method will specify credentials for DataRobot, as well as the location of the DataRobot deployment. We currently support configuration using a configuration file, by setting environment variables, or within the code itself.

1.2.1 Credentials

You will have to specify an API token and an endpoint in order to use the client. You can manage your API tokens in the DataRobot webapp, in your profile. This section describes how to use these options. Their order of precedence is as follows, noting that the first available option will be used:

1. Setting endpoint and token in code using *`datarobot.Client`*
2. Configuring from a config file as specified directly using *`datarobot.Client`*
3. Configuring from a config file as specified by the environment variable *`DATAROBOT_CONFIG_FILE`*
4. Configuring from the environment variables *`DATAROBOT_ENDPOINT`* and *`DATAROBOT_API_TOKEN`*
5. Searching for a config file in the home directory of the current user, at *`~/.config/datarobot/drconfig.yaml`*

Note: If you access the DataRobot webapp at *`https://app.datarobot.com`*, then the correct endpoint to specify would be *`https://app.datarobot.com/api/v2`*. If you have a local installation, update the endpoint accordingly to point at the installation of DataRobot available on your local network.

1.2.2 Set Credentials Explicitly in Code

Explicitly set credentials in code:

```
import datarobot as dr
dr.Client(token='your_token', endpoint='https://app.datarobot.com/api/v2')
```

You can also point to a YAML config file to use:

```
import datarobot as dr
dr.Client(config_path='/home/user/my_datarobot_config.yaml')
```

1.2.3 Use a Configuration File

You can use a configuration file to specify the client setup.

The following is an example configuration file that should be saved as *`~/.config/datarobot/drconfig.yaml`*:

```
token: yourtoken
endpoint: https://app.datarobot.com/api/v2
```

You can specify a different location for the DataRobot configuration file by setting the `DATAROBOT_CONFIG_FILE` environment variable. Note that if you specify a filepath, you should use an absolute path so that the API client will work when run from any location.

1.2.4 Set Credentials Using Environment Variables

Set up an endpoint by setting environment variables in the UNIX shell:

```
export DATAROBOT_ENDPOINT='https://app.datarobot.com/api/v2'
export DATAROBOT_API_TOKEN=your_token
```

1.3 Common Issues

This section has examples of cases that can cause issues with using the DataRobot client, as well as known fixes.

1.3.1 InsecurePlatformWarning

On versions of Python earlier than 2.7.9 you might have `InsecurePlatformWarning` in your output. To prevent this without updating your Python version you should install `pyOpenSSL` package:

```
pip install pyopenssl ndg-httpsclient pyasn1
```

1.3.2 AttributeError: 'EntryPoint' object has no attribute 'resolve'

Some earlier versions of `setuptools` will cause an error on importing DataRobot. The recommended fix is upgrading `setuptools`. If you are unable to upgrade `setuptools`, pinning `trafaret` to version `<=7.4` will correct this issue.

```
>>> import datarobot as dr
...
File "/home/clark/.local/lib/python2.7/site-packages/trafaret/__init__.py", line 1550,
  ↪ in load_contrib
    trafaret_class = entrypoint.resolve()
AttributeError: 'EntryPoint' object has no attribute 'resolve'
```

To prevent this upgrade your `setuptools`:

```
pip install --upgrade setuptools
```

1.3.3 Connection Errors

<configuration.rst> describes how to configure the DataRobot client with the `max_retries` parameter to fine tune behaviors like the number of times it attempts to retry failed connections.

1.3.4 ConnectTimeout

If you have a slow connection to your DataRobot installation, you may see a traceback like

```
ConnectTimeout: HTTPSConnectionPool(host='my-datarobot.com', port=443): Max
retries exceeded with url: /api/v2/projects/
(Caused by ConnectTimeoutError(<requests.packages.urllib3.connection.
↪VerifiedHTTPSConnection object at 0x7f130fc76150>,
'Connection to my-datarobot.com timed out. (connect timeout=6.05)'))
```

You can configure a larger connect timeout (the amount of time to wait on each request attempting to connect to the DataRobot server before giving up) using a *connect_timeout* value in either a configuration file or via a direct call to `datarobot.Client`.

1.3.5 project.open_leaderboard_browser

Calling the `project.open_leaderboard_browser` may block if ran with a text-mode browser or running on a server that doesn't have an ability to open a browser.

This section describes all of the settings that can be configured in the DataRobot configuration file. This file is by default looked for inside the user's home directory at `~/.config/datarobot/drconfig.yaml`, but the default location can be overridden by specifying an environment variable `DATAROBOT_CONFIG_FILE`, or within the code by setting the global client with `dr.Client (config_path='/path/to/config.yaml')`.

2.1 Configurable Variables

These are the variables available for configuration for the DataRobot client:

endpoint This parameter is required. It is the URL of the DataRobot endpoint. For example, the default endpoint on the cloud installation of DataRobot is `https://app.datarobot.com/api/v2`

token This parameter is required. It is the token of your DataRobot account. This can be found in the user settings page of DataRobot

connect_timeout This parameter is optional. It specifies the number of seconds that the client should be willing to wait to establish a connection to the remote server. Users with poor connections may need to increase this value. By default DataRobot uses the value `6.05`.

ssl_verify This parameter is optional. It controls the SSL certificate verification of the DataRobot client. DataRobot is built with the python `requests` library, and this variable is used as the `verify` parameter in that library. More information can be found in their [documentation](#). The default value is `true`, which means that `requests` will use your computer's set of trusted certificate chains by default.

max_retries This parameter is optional. It controls the number of retries to attempt for each connection. More information can be found in the [requests documentation](#). By default, the client will attempt 10 retries (the default provided by `Retry`) with an exponential backoff between attempts. It will retry after connection errors, read errors, and 413, 429, and 503 HTTP responses, and will respect the `Retry-After` header, as in: `Retry (backoff_factor=0.1, respect_retry_after_header=True)` More granular control can be acquired by passing a `Retry` object from `urllib3` into a direct instantiation of `dr.Client`.

```
import datarobot as dr
dr.Client(endpoint='https://app.datarobot.com/api/v2', token='this-is-a-fake-token
→',
          max_retries=Retry(connect=3, read=3))
```

2.2 Proxy support

DataRobot API can work behind a non-transparent HTTP proxy server. Please set environment variable `HTTP_PROXY` containing proxy URL to route all the DataRobot traffic through that proxy server, e.g. `HTTP_PROXY="http://my-proxy.local:3128" python my_datarobot_script.py`.

CHAPTER 3

QuickStart

Note: You must set up credentials in order to access the DataRobot API. For more information, see [Credentials](#)

All of the modeling within DataRobot happens within a project. Each project has one dataset that is used as the source from which to train models.

There are three steps required to begin modeling:

1. Create an empty project.
2. Upload a data file to model.
3. Select parameters and start training models with the autopilot.

The following command includes these three steps. It is equivalent to choosing all of the default settings recommended by DataRobot.

```
import datarobot as dr
project = dr.Project.start(project_name='My new project',
                           sourcedata='/home/user/data/last_week_data.csv',
                           target='ItemsPurchased')
```

Where:

- `name` is the name of the new DataRobot project.
- `sourcedata` is the path to the dataset.
- `target` is the name of the target feature column in the dataset.

You can also pass additional optional parameters:

- `worker_count` - int, sets number of workers used for modeling.
- `metric` - str, name of metric to use.
- `autopilot_on` - boolean, defaults to `True`; set whether or not to begin modeling automatically.
- `blueprint_threshold` - int, number of hours the model is permitted to run. Minimum 1.

- `response_cap` – float, Quantile of the response distribution to use for response capping. Must be in range 0.5..1.0
- `partitioning_method` – `PartitioningMethod` object.
- `positive_class` – str, float, or int; Specifies a level of the target column that should be treated as the positive class for binary classification. May only be specified for binary classification targets.
- `target_type` – str, override the automatically selected `target_type`. An example usage would be setting the `target_type=TARGET_TYPE.MULTICLASS` when you want to perform a multiclass classification task on a numeric column that has a low cardinality.

Before training any models or creating any projects, you need to upload your data into a Dataset.

4.1 Creating A Dataset

There are several ways to create a Dataset. `Dataset.create_from_file` can take either a path to a local file or any stream-able file object.

```
>>> import datarobot as dr
>>> dataset = dr.Dataset.create_from_file(file_path='data_dir/my_data.csv')
>>> with open('data_dir/my_data.csv', 'rb') as f:
...     other_dataset = dr.Dataset.create_from_file(filelike=f)
```

`Dataset.create_from_in_memory_data` can take either a `pandas.DataFrame` or a list of dictionaries representing rows of data. Note that the dictionaries representing the rows of data must contain the same keys.

```
>>> import pandas as pd
>>> data_frame = pd.read_csv('data_dir/my_data.csv')

# do things to my data_frame
>>> pandas_dataset = dr.Dataset.create_from_in_memory_data(data_frame=data_frame)

>>> in_memory_data = [{'key1': 'value', 'key2': 'other_value', ...},
...                   {'key1': 'new_value', 'key2': 'other_new_value', ...}, ...]
>>> in_memory_dataset = dr.Dataset.create_from_in_memory_data(records=other_data)
```

`Dataset.create_from_url` takes csv data from a URL. If you have not set `ENABLE_CREATE_SNAPSHOT_DATASOURCE`, you must set `do_snapshot=False`.

```
>>> url_dataset = dr.Dataset.create_from_url('https://s3.amazonaws.com/my_data/my_
↳ dataset.csv',
...                                         do_snapshot=False)
```

`Dataset.create_from_url` takes data from a data source. If you have not set `ENABLE_CREATE_SNAPSHOT_DATASOURCE`, you must set `do_snapshot=False`.

```
>>> data_source_dataset = dr.Dataset.create_from_data_source(data_source.id, do_
↳ snapshot=False)
```

or

```
>>> data_source_dataset = data_source.create_dataset(do_snapshot=False)
```

4.1.1 Using Datasets

Once a Dataset is created, you can create *Projects* from it and then begin training on the projects. (You can also combine project creation and uploading Dataset in a single step in *Project.create*. However, this means the data is only accessible to the project which created it.)

```
>>> project = dataset.create_project(project_name='New Project')
>>> project.set_target('some target')
Project(New Project)
```

4.2 Getting Information From A Dataset

The dataset object contains some basic information:

```
>>> dataset.id
u'5e31cdac39782d0f65842518'
>>> dataset.name
u'my_data.csv'
>>> dataset.categories
["TRAINING", "PREDICTION"]
>>> dataset.created_at
datetime.datetime(2020, 2, 7, 16, 51, 10, 311000, tzinfo=tzutc())
```

There are several methods to get details from a Dataset.

```
# Details
>>> details = dataset.get_details()
>>> details.last_modification_date
datetime.datetime(2020, 2, 7, 16, 51, 10, 311000, tzinfo=tzutc())
>>> details.feature_count_by_type
[FeatureTypeCount(count=1, feature_type=u'Text'),
 FeatureTypeCount(count=1, feature_type=u'Boolean'),
 FeatureTypeCount(count=16, feature_type=u'Numeric'),
 FeatureTypeCount(count=3, feature_type=u'Categorical')]
>>> details.to_dataset().id == details.dataset_id
True

# Projects
>>> dr.Project.create_from_dataset(dataset.id, project_name='Project One')
Project(Project One)
>>> dr.Project.create_from_dataset(dataset.id, project_name='Project Two')
Project(Project Two)
>>> dataset.get_projects()
```

(continues on next page)

(continued from previous page)

```
[ProjectLocation(url=u'https://app.datarobot.com/api/v2/projects/
↳5e3c94aff86f2d10692497b5/', id=u'5e3c94aff86f2d10692497b5'),
 ProjectLocation(url=u'https://app.datarobot.com/api/v2/projects/
↳5e3c94eb9525d010a9918ec1/', id=u'5e3c94eb9525d010a9918ec1')]
>>> first_id = dataset.get_projects()[0].id
>>> dr.Project.get(first_id).project_name
'Project One'

# Features
>>> all_features = dataset.get_all_features()
>>> feature = next(dataset.iterate_all_features(offset=2, limit=1))
>>> feature.name == all_features[2].name
True
>>> print(feature.name, feature.feature_type, feature.dataset_id)
(u'Partition', u'Numeric', u'5e31cdac39782d0f65842518')
>>> feature.get_histogram().plot
[{'count': 3522, 'target': None, 'label': u'0.0'},
 {'count': 3521, 'target': None, 'label': u'1.0'}, ... ]

# The raw data
>>> with open('myfile.csv', 'wb') as f:
...     dataset.get_file(filelike=f)
```

4.3 Retrieving Datasets

You can retrieve either specific datasets, the list of all datasets or an iterator that can get all or some of the datasets.

```
>>> dataset_id = '5e387c501a438646ed7bf0f2'
>>> dataset = dr.Dataset.get(dataset_id)
>>> dataset.id == dataset_id
True
# a blocking call that returns all datasets
>>> dr.Dataset.list()
[Dataset(name=u'Untitled Dataset', id=u'5e3c51e0f86f2d1087249728'),
 Dataset(name=u'my_data.csv', id=u'5e3c2028162e6a5fe9a0d678'), ...]

# avoid listing Datasets that failed to properly upload
>>> dr.Dataset.list(filter_failed=True)
[Dataset(name=u'my_data.csv', id=u'5e3c2028162e6a5fe9a0d678'),
 Dataset(name=u'my_other_data.csv', id=u'3efc2428g62eaa5f39a6dg7a'), ...]

# an iterator that lazily retrieves from the server page-by-page
>>> from itertools import islice
>>> iterator = dr.Dataset.iterate(offset=2)
>>> for element in islice(iterator, 3):
...     print(element)
Dataset(name='some_data.csv', id='5e8df2f21a438656e7a23d12')
Dataset(name='other_data.csv', id='5e8df2e31a438656e7a23d0b')
Dataset(name='Untitled Dataset', id='5e6127681a438666cc73c2b0')
```

4.4 Managing Datasets

You can modify, delete and un_delete datasets. Note that you need the dataset's ID in order to un_delete it and if you do not keep track of this it will be gone. If your deleted dataset had been used to create a project, that project can still access it, but you will not be able to create new projects using that dataset.

```
>>> dataset.modify(name='A Better Name')
>>> dataset.name
'A Better Name'

>>> new_project = dr.Project.create_from_dataset(dataset.id)
>>> stored_id = dataset.id
>>> dr.Dataset.delete(dataset.id)

# new_project is still ok
>>> dr.Project.create_from_dataset(stored_id)
Traceback (most recent call last):
...
datarobot.errors.ClientError: 410 client error: {u'message': u'Requested Dataset_
↳5e31cdac39782d0f65842518 was previously deleted.'}

>>> dr.Dataset.un_delete(stored_id)
>>> dr.Project.create_from_dataset(stored_id, project_name='Successful')
Project (Successful)
```

4.5 Managing Dataset Featurelists

You can create, modify, and delete custom featurelists on a given dataset. Some featurelists are automatically created by DataRobot and can not be modified or deleted. There is no option to un_delete a deleted featurelist.

```
>>> dataset.get_featurelists()
[DatasetFeaturelist(Raw Features),
 DatasetFeaturelist(universe),
 DatasetFeaturelist(Informative Features)]

>>> dataset_features = [feature.name for feature in dataset.get_all_features()]
>>> custom_featurelist = dataset.create_featurelist('Custom Features', dataset_
↳features[:5])
>>> custom_featurelist
DatasetFeaturelist(Custom Features)

>>> dataset.get_featurelists()
[DatasetFeaturelist(Raw Features),
 DatasetFeaturelist(universe),
 DatasetFeaturelist(Informative Features),
 DatasetFeaturelist(Custom Features)]

>>> custom_featurelist.update('New Name')
>>> custom_featurelist.name
'New Name'

>>> custom_featurelist.delete()
>>> dataset.get_featurelists()
[DatasetFeaturelist(Raw Features),
```

(continues on next page)

(continued from previous page)

```
DatasetFeaturelist(universe),  
DatasetFeaturelist(Informative Features)]
```


All of the modeling within DataRobot happens within a project. Each project has one dataset that is used as the source from which to train models.

5.1 Create a Project

You can create a project from previously created *Datasets* or directly from a data source.

```
import datarobot as dr
dataset = Dataset.create_from_file(file_path='/home/user/data/last_week_data.csv')
project = dr.Project.create_from_dataset(dataset.id, project_name='New Project')
```

The following command creates a new project directly from a data source. You must specify a path to data file, file object URL (starting with `http://`, `https://`, `file://`, or `s3://`), raw file contents, or a `pandas.DataFrame` object when creating a new project. Path to file can be either a path to a local file or a publicly accessible URL.

```
import datarobot as dr
project = dr.Project.create('/home/user/data/last_week_data.csv',
                           project_name='New Project')
```

You can use the following commands to view the project ID and name:

```
project.id
>>> u'5506fcd38bd88f5953219da0'
project.project_name
>>> u'New Project'
```

5.2 Select Modeling Parameters

The final information needed to begin modeling includes the target feature, the queue mode, the metric for comparing models, and the optional parameters such as weights, offset, exposure and downsampling.

5.2.1 Target

The target must be the name of one of the columns of data uploaded to the project.

5.2.2 Metric

The optimization metric used to compare models is an important factor in building accurate models. If a metric is not specified, the default metric recommended by DataRobot will be used. You can use the following code to view a list of valid metrics for a specified target:

```
target_name = 'ItemsPurchased'
project.get_metrics(target_name)
>>> {'available_metrics': [
    'Gini Norm',
    'Weighted Gini Norm',
    'Weighted R Squared',
    'Weighted RMSLE',
    'Weighted MAPE',
    'Weighted Gamma Deviance',
    'Gamma Deviance',
    'RMSE',
    'Weighted MAD',
    'Tweedie Deviance',
    'MAD',
    'RMSLE',
    'Weighted Tweedie Deviance',
    'Weighted RMSE',
    'MAPE',
    'Weighted Poisson Deviance',
    'R Squared',
    'Poisson Deviance'],
    'feature_name': 'SalePrice'}
```

5.2.3 Partitioning Method

DataRobot projects always have a *holdout* set used for final model validation. We use two different approaches for testing prior to the holdout set:

- split the remaining data into *training* and *validation* sets
- cross-validation, in which the remaining data is split into a number of folds; each fold serves as a validation set, with models trained on the other folds and evaluated on that fold.

There are several other options you can control. To specify a partition method, create an instance of one of the *Partition Classes*, and pass it as the `partitioning_method` argument in your call to `project.set_target` or `project.start`. See [here](#) for more information on using datetime partitioning.

Several partitioning methods include parameters for `validation_pct` and `holdout_pct`, specifying desired percentages for the validation and holdout sets. Note that there may be constraints that prevent the actual percentages used from exactly (or some cases, even closely) matching the requested percentages.

5.2.4 Queue Mode

You can use the API to set the DataRobot modeling process to run in either automatic or manual mode.

Autopilot mode means that the modeling process will proceed completely automatically, including running recommended models, running at different sample sizes, and blending.

Manual mode means that DataRobot will populate a list of recommended models, but will not insert any of them into the queue. Manual mode lets you select which models to execute before starting the modeling process.

Quick mode means that a smaller set of Blueprints is used, so autopilot finishes faster.

5.2.5 Weights

DataRobot also supports using a weight parameter. A full discussion of the use of weights in data science is not within the scope of this document, but weights are often used to help compensate for rare events in data. You can specify a column name in the project dataset to be used as a weight column.

5.2.6 Offsets

Starting with version v2.6 DataRobot also supports using an offset parameter. Offsets are commonly used in insurance modeling to include effects that are outside of the training data due to regulatory compliance or constraints. You can specify the names of several columns in the project dataset to be used as the offset columns.

5.2.7 Exposure

Starting with version v2.6 DataRobot also supports using an exposure parameter. Exposure is often used to model insurance premiums where strict proportionality of premiums to duration is required. You can specify the name of the column in the project dataset to be used as an exposure column.

5.3 Start Modeling

Once you have selected modeling parameters, you can use the following code structure to specify parameters and start the modeling process.

```
import datarobot as dr
project.set_target(target='ItemsPurchased',
                  metric='Tweedie Deviance',
                  mode=dr.AUTOPILOT_MODE.FULL_AUTO)
```

You can also pass additional optional parameters to `project.set_target` to change parameters of the modeling process. Some of those parameters include:

- `worker_count` – int, sets number of workers used for modeling.
- `partitioning_method` – `PartitioningMethod` object.

- `positive_class` – str, float, or int; Specifies a level of the target column that should be treated as the positive class for binary classification. May only be specified for binary classification targets.
- `advanced_options` – *AdvancedOptions* object, used to set advanced options of modeling process.
- `target_type` – str, override the automatically selected `target_type`. An example usage would be setting the `target_type=TARGET_TYPE.MULTICLASS` when you want to perform a multiclass classification task on a numeric column that has a low cardinality.

For a full reference of available parameters, see `Project.set_target`.

You can run with different autopilot modes with the `mode` parameter. `AUTOPILOT_MODE.FULL_AUTO` is the default, which will trigger modeling with no further actions necessary. Other accepted modes include `AUTOPILOT_MODE.MANUAL` for manual mode (choose your own models to run rather than use the DataRobot autopilot) and `AUTOPILOT_MODE.QUICK` for quickrun (run on a more limited set of models to get insights more quickly).

5.3.1 Clone a Project

Once a project has been successfully created, you may clone it using the following code structure:

```
new_project = project.clone_project(new_project_name='This is my new project')
new_project.name
>> 'This is my new project'
new_project.id != project.id
>> True
```

The `new_project_name` attribute is optional. If it is omitted, the default new project name will be “Copy of <project.name>”.

5.4 Interact with a Project

The following commands can be used to manage DataRobot projects.

5.4.1 List Projects

Returns a list of projects associated with current API user.

```
import datarobot as dr
dr.Project.list()
>>> [Project(Project One), Project(Two)]

dr.Project.list(search_params={'project_name': 'One'})
>>> [Project(One)]
```

You can pass following parameters to change result:

- `search_params` – dict, used to filter returned projects. Currently you can query projects only by `project_name`

5.4.2 Get an existing project

Rather than querying the full list of projects every time you need to interact with a project, you can retrieve its `id` value and use that to reference the project.

```
import datarobot as dr
project = dr.Project.get(project_id='5506fcd38bd88f5953219da0')
project.id
>>> '5506fcd38bd88f5953219da0'
project.project_name
>>> 'Churn Projection'
```

5.4.3 Get feature association statistics for an existing project

Get either feature association or correlation statistics and metadata on informative features for a given project

```
import datarobot as dr
project = dr.Project.get(project_id='5506fcd38bd88f5953219da0')
association_data = project.get_associations(assoc_type='association', metric=
↪ 'mutualInfo')
association_data.keys()
>>> ['strengths', 'features']
```

5.4.4 Get whether your featurelists have association statistics

Get whether an association matrix job has been run on each of your featurelists

```
import datarobot as dr
project = dr.Project.get(project_id='5506fcd38bd88f5953219da0')
featurelists = project.get_association_featurelists()
featurelists['featurelists'][0]
>>> {"featurelistId": "54e510ef8bd88f5aeb02a3ed", "hasFam": True, "title":
↪ "Informative Features"}
```

5.4.5 Get values for a pair of features in an existing project

Get a sample of the exact values used in the feature association matrix plotting

```
import datarobot as dr
project = dr.Project.get(project_id='5506fcd38bd88f5953219da0')
feature_values = project.get_association_matrix_details(feature1='foo', feature2='bar'
↪)
feature_values.keys()
>>> ['features', 'types', 'values']
```

5.4.6 Update a project

You can update various attributes of a project.

To update the name of the project:

```
project.rename(new_name)
```

To update the number of workers used by your project (this will fail if you request more workers than you have available; the special value *-1* will request your maximum number):

```
project.set_worker_count(num_workers)
```

To unlock the holdout set, allowing holdout scores to be shown and models to be trained on more data:

```
project.unlock_holdout()
```

To add or change the project description:

```
project.set_project_description(project_description)
```

5.4.7 Delete a project

Use the following command to delete a project:

```
project.delete()
```

5.4.8 Wait for Autopilot to Finish

Once the modeling autopilot is started, in some cases you will want to wait for autopilot to finish:

```
project.wait_for_autopilot()
```

5.4.9 Play/Pause the autopilot

If your project is running in autopilot mode, it will continually use available workers, subject to the number of workers allocated to the project and the total number of simultaneous workers allowed according to the user permissions.

To pause a project running in autopilot mode:

```
project.pause_autopilot()
```

To resume running a paused project:

```
project.unpause_autopilot()
```

5.4.10 Start autopilot on another Featurelist

You can start autopilot on an existing featurelist.

```
import datarobot as dr

featurelist = project.create_featurelist('test', ['feature 1', 'feature 2'])
project.start_autopilot(featurelist.id)
>>> True

# Starting autopilot that is already running on the provided featurelist
project.start_autopilot(featurelist.id)
>>> dr.errors.AppPlatformError
```

Note: This method should be used on a project where the target has already been set. An error will be raised if autopilot is currently running on or has already finished running on the provided featurelist.

5.5 Further reading

The Blueprints and Models sections of this document will describe how to create new models based on the Blueprints recommended by DataRobot.

Datetime Partitioned Projects

If your dataset is modeling events taking place over time, datetime partitioning may be appropriate. Datetime partitioning ensures that when partitioning the dataset for training and validation, rows are ordered according to the value of the date partition feature.

6.1 Setting Up a Datetime Partitioned Project

After creating a project and before setting the target, create a *DatetimePartitioningSpecification* to define how the project should be partitioned. By passing the specification into `DatetimePartitioning.generate`, the full partitioning can be previewed before finalizing the partitioning. After verifying that the partitioning is correct for the project dataset, pass the specification into `Project.set_target` via the `partitioning_method` argument. Once modeling begins, the project can be used as normal.

The following code block shows the basic workflow for creating datetime partitioned projects.

```
import datarobot as dr

project = dr.Project.create('some_data.csv')
spec = dr.DatetimePartitioningSpecification('my_date_column')
# can customize the spec as needed

partitioning_preview = dr.DatetimePartitioning.generate(project.id, spec)
# the preview generated is based on the project's data

print(partitioning_preview.to_dataframe())
# hmm ... I want more backtests
spec.number_of_backtests = 5
partitioning_preview = dr.DatetimePartitioning.generate(project.id, spec)
print(partitioning_preview.to_dataframe())
# looks good

project.set_target('target_column', partitioning_method=spec)
```

(continues on next page)

(continued from previous page)

```
# I can retrieve the partitioning settings after the target has been set too
partitioning = dr.DatetimePartitioning.get(project.id)
```

6.1.1 Configuring Backtests

Backtests are configurable using one of two methods:

Method 1:

- `index` (int): The index from zero of this backtest.
- `gap_duration` (str): A duration string such as those returned by the `partitioning_methods.construct_duration_string` helper method. This represents the gap between training and validation scoring data for this backtest.
- `validation_start_date` (datetime.datetime): Represents the start date of the validation scoring data for this backtest.
- `validation_duration` (str): A duration string such as those returned by the `partitioning_methods.construct_duration_string` helper method. This represents the desired duration of the validation scoring data for this backtest.

```
import datarobot as dr

partitioning_spec = dr.DatetimePartitioningSpecification(
    backtests=[
        # modify the first backtest using option 1
        dr.BacktestSpecification(
            index=0,
            gap_duration=dr.partitioning_methods.construct_duration_string(),
            validation_start_date=datetime(year=2010, month=1, day=1),
            validation_duration=dr.partitioning_methods.construct_duration_
↪string(years=1),
        )
    ],
    # other partitioning settings...
)
```

Method 2 (New in version v2.20):

- `validation_start_date` (datetime.datetime): Represents the start date of the validation scoring data for this backtest.
- `validation_end_date` (datetime.datetime): Represents the end date of the validation scoring data for this backtest.
- `primary_training_start_date` (datetime.datetime): Represents the desired start date of the training partition for this backtest.
- `primary_training_end_date` (datetime.datetime): Represents the desired end date of the training partition for this backtest.

```
import datarobot as dr

partitioning_spec = dr.DatetimePartitioningSpecification(
    backtests=[
        # modify the first backtest using option 2
        dr.BacktestSpecification(
```

(continues on next page)

(continued from previous page)

```

        index=0,
        primary_training_start_date=datetime(year=2005, month=1, day=1),
        primary_training_end_date=datetime(year=2010, month=1, day=1),
        validation_start_date=datetime(year=2010, month=1, day=1),
        validation_end_date=datetime(year=2011, month=1, day=1),
    )
    ],
    # other partitioning settings...
)

```

Note that Method 2 allows you to directly configure the start and end dates of each partition, including the training partition. The gap partition is calculated as the time between `primary_training_end_date` and `validation_start_date`. Using the same date for both `primary_training_end_date` and `validation_start_date` will result in no gap being created.

After configuring backtests, you can set `use_project_settings` to `True` in calls to `Model.train_datetime`. This will create models that are trained and validated using your custom backtest training partition start and end dates.

6.2 Modeling with a Datetime Partitioned Project

While `Model` objects can still be used to interact with the project, `DatetimeModel` objects, which are only retrievable from datetime partitioned projects, provide more information including which date ranges and how many rows are used in training and scoring the model as well as scores and statuses for individual backtests.

The autopilot workflow is the same as for other projects, but to manually train a model, `Project.train_datetime` and `Model.train_datetime` should be used in the place of `Project.train` and `Model.train`. To create frozen models, `Model.request_frozen_datetime_model` should be used in place of `DatetimeModel.request_frozen_datetime_model`. Unlike other projects, to trigger computation of scores for all backtests use `DatetimeModel.score_backtests` instead of using the `scoring_type` argument in the `train` methods.

6.3 Dates, Datetimes, and Durations

When specifying a date or datetime for datetime partitioning, the client expects to receive and will return a `datetime`. Timezones may be specified, and will be assumed to be UTC if left unspecified. All dates returned from DataRobot are in UTC with a timezone specified.

Datetimes may include a time, or specify only a date; however, they may have a non-zero time component only if the partition column included a time component in its date format. If the partition column included only dates like `2015-03-24T00:00:00`, then the time component of any datetimes, if present, must be zero.

When date ranges are specified with a start and an end date, the end date is exclusive, so only dates earlier than the end date are included, but the start date is inclusive, so dates equal to or later than the start date are included. If the start and end date are the same, then no dates are included in the range.

Durations are specified using a subset of ISO8601. Durations will be of the form `PnYnMnDTnHnMnS` where each `n` may be replaced with an integer value. Within the duration string,

- `nY` represents the number of years
- the `nM` following the `P` represents the number of months
- `nD` represents the number of days

- nH represents the number of hours
- the nM following the `âĀĪJTâĀĪ` represents the number of minutes
- nS represents the number of seconds

and `âĀĪJPâĀĪ` is used to indicate that the string represents a period and `âĀĪJTâĀĪ` indicates the beginning of the time component of the string. Any section with a value of 0 may be excluded. As with datetimes, if the partition column did not include a time component in its date format, the time component of any duration must be either unspecified or consist only of zeros.

Example Durations:

- `âĀĪJP3Y6MâĀĪ` (three years, six months)
- `âĀĪJP1Y0M0DT0H0M0SâĀĪ` (one year)
- `âĀĪJP1Y5DT10HâĀĪ` (one year, 5 days, 10 hours)

`datarobot.helpers.partitioning_methods.construct_duration_string` is a helper method that can be used to construct appropriate duration strings.

Time Series Projects

Time series projects, like OTV projects, use *datetime partitioning*, and all the workflow changes that apply to other datetime partitioned projects also apply to them. Unlike other projects, time series projects produce different types of models which forecast multiple future predictions instead of an individual prediction for each row.

DataRobot uses a general time series framework to configure how time series features are created and what future values the models will output. This framework consists of a Forecast Point (defining a time a prediction is being made), a Feature Derivation Window (a rolling window used to create features), and a Forecast Window (a rolling window of future values to predict). These components are described in more detail below.

Time series projects will automatically transform the dataset provided in order to apply this framework. During the transformation, DataRobot uses the Feature Derivation Window to derive time series features (such as lags and rolling statistics), and uses the Forecast Window to provide examples of forecasting different distances in the future (such as time shifts). After project creation, a new dataset and a new feature list are generated and used to train the models. This process is reapplied automatically at prediction time as well in order to generate future predictions based on the original data features.

The `time_unit` and `time_step` used to define the Feature Derivation and Forecast Windows are taken from the datetime partition column, and can be retrieved for a given column in the input data by looking at the corresponding attributes on the `datarobot.models.Feature` object. If `windows_basis_unit` is set to ROW, then Feature Derivation and Forecast Windows will be defined using number of the rows.

7.1 Setting Up A Time Series Project

To set up a time series project, follow the standard *datetime partitioning* workflow and use the six new time series specific parameters on the `datarobot.DatetimePartitioningSpecification` object:

use_time_series bool, set this to True to enable time series for the project.

default_to_known_in_advance bool, set this to True to default to treating all features as known in advance, or a priori, features. Otherwise, they will not be handled as known in advance features. Individual features can be set to a value different than the default by using the `featureSettings` parameter. See *the prediction documentation* for more information.

default_to_do_not_derive bool, set this to True to default to excluding all features from feature derivation. Otherwise, they will not be excluded and will be included in the feature derivation process. Individual features can be set to a value different than the default by using the `featureSettings` parameter.

feature_derivation_window_start int, specifies how many units of the `windows_basis_unit` from the forecast point into the past is the start of the feature derivation window

feature_derivation_window_end int, specifies how many units of the `windows_basis_unit` from the forecast point into the past is the end of the feature derivation window

forecast_window_start int, specifies how many units of the `windows_basis_unit` from the forecast point into the future is the start of the forecast window

forecast_window_end int, specifies how many units of the `windows_basis_unit` from the forecast point into the future is the end of the forecast window

windows_basis_unit string, set this to `ROW` to define feature derivation and forecast windows in terms of the rows, rather than time units. If omitted, will default to the detected time unit (one of the `datarobot.enums.TIME_UNITS`).

feature_settings list of `FeatureSettings` specifying per feature settings, can be left unspecified

7.1.1 Feature Derivation Window

The Feature Derivation window represents the rolling window that is used to derive time series features and lags, relative to the Forecast Point. It is defined in terms of `feature_derivation_window_start` and `feature_derivation_window_end` which are integer values representing datetime offsets in terms of the `time_unit` (e.g. hours or days).

The Feature Derivation Window start and end must be less than or equal to zero, indicating they are positioned before the forecast point. Additionally, the window must be specified as an integer multiple of the `time_step` which defines the expected difference in time units between rows in the data.

The window is closed, meaning the edges are considered to be inside the window.

7.1.2 Forecast Window

The Forecast Window represents the rolling window of future values to predict, relative to the Forecast Point. It is defined in terms of the `forecast_window_start` and `forecast_window_end`, which are positive integer values indicating datetime offsets in terms of the `time_unit` (e.g. hours or days).

The Forecast Window start and end must be positive integers, indicating they are positioned after the forecast point. Additionally, the window must be specified as an integer multiple of the `time_step` which defines the expected difference in time units between rows in the data.

The window is closed, meaning the edges are considered to be inside the window.

7.1.3 Multiseries Projects

Certain time series problems represent multiple separate series of data, e.g. ÅIJI have five different stores that all have different customer bases. I want to predict how many units of a particular item will sell, and account for the different behavior of each storeÅI. When setting up the project, a column specifying series ids must be identified, so that each row from the same series has the same value in the multiseries id column.

Using a multiseries id column changes which partition columns are eligible for time series, as each series is required to be unique and regular, instead of the entire partition column being required to have those properties. In order to use a multiseries id column for partitioning, a detection job must first be run to analyze the relationship between the

partition and multiserries id columns. If needed, it will be automatically triggered by calling `datarobot.models.Feature.get_multiserries_properties()` on the desired partition column. The previously computed multiserries properties for a particular partition column can then be accessed via that method. The computation will also be automatically triggered when calling `datarobot.DatetimePartitioning.generate()` or `datarobot.models.Project.set_target()` with a multiserries id column specified.

Note that currently only one multiserries id column is supported, but all interfaces accept lists of id columns to ensure multiple id columns will be able to be supported in the future.

In order to create a multiserries project:

1. Set up a datetime partitioning specification with the desired partition column and multiserries id columns.
2. (Optionally) Use `datarobot.models.Feature.get_multiserries_properties()` to confirm the inferred time step and time unit of the partition column when used with the specified multiserries id column.
3. (Optionally) Specify the multiserries id column in order to preview the full datetime partitioning settings using `datarobot.DatetimePartitioning.generate()`.
4. Specify the multiserries id column when sending the target and partitioning settings via `datarobot.models.Project.set_target()`.

```
project = dr.Project.create('path/to/multiserries.csv', project_name='my multiserries_
↳project')
partitioning_spec = dr.DatetimePartitioningSpecification(
    'timestamp', use_time_series=True, multiserries_id_columns=['multiserries_id']
)

# manually confirm time step and time unit are as expected
datetime_feature = dr.Feature.get(project.id, 'timestamp')
multiserries_props = datetime_feature.get_multiserries_properties(['multiserries_id'])
print(multiserries_props)

# manually check out the partitioning settings like feature derivation window and_
↳backtests
# to make sure they make sense before moving on
full_part = dr.DatetimePartitioning.generate(project.id, partitioning_spec)
print(full_part.feature_derivation_window_start, full_part.feature_derivation_window_
↳end)
print(full_part.to_dataframe())

# finalize the project and start the autopilot
project.set_target('target', partitioning_method=partitioning_spec)
```

7.1.4 Feature Settings

`datarobot.FeatureSettings` constructor receives `feature_name` and settings. For now settings `known_in_advance` and `do_not_derive` are supported.

```
# I have 10 features, 8 of them are known in advance and two are not
# Also, I do not want to derive new features from previous_day_sales
not_known_in_advance_features = ['previous_day_sales', 'amount_in_stock']
do_not_derive_features = ['previous_day_sales']
feature_settings = [dr.FeatureSettings(feat_name, known_in_advance=False)
feature_settings += [dr.FeatureSettings(feat_name, do_not_derive=True) for feat_name_
↳in do_not_derive_features]
spec = dr.DatetimePartitioningSpecification(
    # ...
```

(continues on next page)

(continued from previous page)

```

default_to_known_in_advance=True,
feature_settings=feature_settings
)

```

7.2 Modeling Data and Time Series Features

In time series projects, a new set of modeling features is created after setting the partitioning options. If a featurelist is specified with the partitioning options, it will be used to select which features should be used to derived modeling features; if a featurelist is not specified, the default featurelist will be used.

These features are automatically derived from those in the project's dataset and are the features used for modeling - note that the Project methods `get_featurelists` and `get_modeling_featurelists` will return different data in time series projects. Modeling featurelists are the ones that can be used for modeling and will be accepted by the backend, while regular featurelists will continue to exist but cannot be used. Modeling features are only accessible once the target and partitioning options have been set. In projects that don't use time series modeling, once the target has been set, modeling and regular features and featurelists will behave the same.

7.3 Making Predictions

Prediction datasets are uploaded *as normal*. However, when uploading a prediction dataset, a new parameter `forecast_point` can be specified. The forecast point of a prediction dataset identifies the point in time relative which predictions should be generated, and if one is not specified when uploading a dataset, the server will choose the most recent possible forecast point. The forecast window specified when setting the partitioning options for the project determines how far into the future from the forecast point predictions should be calculated.

To simplify the predictions process, starting in version v2.20 a forecast point or prediction start and end dates can be specified when requesting predictions, instead of being specified at dataset upload. Upon uploading a dataset, DataRobot will calculate the range of dates available for use as a forecast point or for batch predictions. To that end, *Predictions* objects now also contain the following new fields:

- `forecast_point`: The default point relative to which predictions will be generated
- `predictions_start_date`: The start date for bulk historical predictions.
- `predictions_end_date`: The end date for bulk historical predictions.

When setting up a time series project, input features could be identified as known-in-advance features. These features are not used to generate lags, and are expected to be known for the rows in the forecast window at predict time (e.g. "how much money will have been spent on marketing", "is this a holiday").

Enough rows of historical data must be provided to cover the span of the effective Feature Derivation Window (which may be longer than the project's Feature Derivation Window depending on the differencing settings chosen). The effective Feature Derivation Window of any model can be checked via the `effective_feature_derivation_window_start` and `effective_feature_derivation_window_end` attributes of a *DatetimeModel*.

When uploading datasets to a time series project, the dataset might look something like the following, where "Time" is the datetime partition column, "Target" is the target column, and "Temp" is an input feature. If the dataset was uploaded with a forecast point of "2017-01-08" and the effective feature derivation window start and end for the model are -5 and -3 and the forecast window start and end were set to 1 and 3, then rows 1 through 3 are historical data, row 6 is the forecast point, and rows 7 through 9 are forecast rows that will have predictions when predictions are computed.


```

Row, Time, Target, Temp.
1, 2017-01-03, 16443, 72
2, 2017-01-04, 3013, 72
3, 2017-01-05, 1643, 68
4, 2017-01-06, ,
5, 2017-01-07, ,
6, 2017-01-08, ,
7, 2017-01-09, ,
8, 2017-01-10, ,
9, 2017-01-11, ,

```

On the other hand, if the project instead used `holiday` as an a priori input feature, the uploaded dataset might look like the following:

```

Row, Time, Target, Holiday
1, 2017-01-03, 16443, TRUE
2, 2017-01-04, 3013, FALSE
3, 2017-01-05, 1643, FALSE
4, 2017-01-06, , FALSE
5, 2017-01-07, , FALSE
6, 2017-01-08, , FALSE
7, 2017-01-09, , TRUE
8, 2017-01-10, , FALSE
9, 2017-01-11, , FALSE

```

7.4 Calendars

You can upload a *calendar file* containing a list of events relevant to your dataset. When provided, DataRobot automatically derives and creates time series features based on the calendar events (e.g., time until the next event, labeling the most recent event).

The calendar file:

- Should span the entire training data date range, as well as all future dates in which model will be forecasting.
- Must be in csv or xlsx format with a header row.
- Must have one date column which has values in the date-only format YYYY-MM-DD (i.e., no hour, month, or second).
- Can optionally include a second column that provides the event name or type.
- Can optionally include a series ID column which specifies which series an event is applicable to. This column name must match the name of the column set as the series ID.
 - Multiseries ID columns are used to add an ability to specify different sets of events for different series, e.g. holidays for different regions.
 - Values of the series ID may be absent for specific events. This means that the event is valid for all series in project dataset (e.g. New Year's Day is a holiday in all series in the example below).
 - If a multiseries ID column is not provided, all listed events will be applicable to all series in the project dataset.
- Cannot be updated in an active project. You must specify all future calendar events at project start. To update the calendar file, you will have to train a new project.

An example of a valid calendar file:

Date,	Name
2019-01-01,	New Year's Day
2019-02-14,	Valentine's Day
2019-04-01,	April Fools
2019-05-05,	Cinco de Mayo
2019-07-04,	July 4th

An example of a valid multiseries calendar file:

Date,	Name,	Country
2019-01-01,	New Year's Day,	
2019-05-27,	Memorial Day,	USA
2019-07-04,	July 4th,	USA
2019-11-28,	Thanksgiving,	USA
2019-02-04,	Constitution Day,	Mexico
2019-03-18,	Benito Juárez's birth,	Mexico
2019-12-25,	Christmas Day,	

Once created, a calendar can be used with a time series project by specifying the `calendar_id` field in the `datarobot.DatetimePartitioningSpecification` object for the project:

```
import datarobot as dr

# create the project
project = dr.Project.create('input_data.csv')
# create the calendar
calendar = dr.CalendarFile.create('calendar_file.csv')

# specify the calendar_id in the partitioning specification
datetime_spec = dr.DatetimePartitioningSpecification(
    use_time_series=True,
    datetime_partition_column='date'
    calendar_id=calendar.id
)

# start the project, specifying the partitioning method
project.set_target(
    target='project target',
    partitioning_method=datetime_spec
)
```

7.5 Prediction Intervals

For each model, prediction intervals estimate the range of values DataRobot expects actual values of the target to fall within. They are similar to a confidence interval of a prediction, but are based on the residual errors measured during the backtesting for the selected model.

Note that because calculation depends on the backtesting values, prediction intervals are not available for predictions on models that have not had all backtests completed. To that end, note that creating a prediction with prediction intervals through the API will automatically complete all backtests if they were not already completed. For start-end retrained models, the parent model will be used for backtesting. Additionally, prediction intervals are not available when the number of points per forecast distance is less than 10, due to insufficient data.

In a prediction request, users can specify a prediction intervals size, which specifies the desired probability of actual values falling within the interval range. Larger values are less precise, but more conservative. For example, speci-

fyng a size of 80 will result in a lower bound of 10% and an upper bound of 90%. More generally, for a specific `prediction_intervals_size`, the upper and lower bounds will be calculated as follows:

- `prediction_interval_upper_bound = 50% + (prediction_intervals_size / 2)`
- `prediction_interval_lower_bound = 50% - (prediction_intervals_size / 2)`

Prediction intervals can be calculated for a `DatetimeModel` using the `DatetimeModel.calculate_prediction_intervals` method. Users can also retrieve which intervals have already been calculated for the model using the `DatetimeModel.get_calculated_prediction_intervals` method.

To view prediction intervals data for a prediction, the prediction needs to have been created using the `DatetimeModel.request_predictions` method and specifying `include_prediction_intervals = True`. The size for the prediction interval can be specified with the `prediction_intervals_size` parameter for the same function, and will default to 80 if left unspecified. Specifying either of these fields will result in prediction interval bounds being included in the retrieved prediction data for that request (see the `Predictions` class for retrieval methods). Note that if the specified interval size has not already been calculated, this request will automatically calculate the specified size.

Prediction intervals are also supported for time series model deployments, and should be specified in deployment settings if desired. Use `Deployment.get_prediction_intervals_settings` to retrieve current prediction intervals settings for a deployment, and `Deployment.update_prediction_intervals_settings` to update prediction intervals settings for a deployment.

Prediction intervals are also supported for time series model export. See the optional `prediction_intervals_size` parameter in `Model.request_transferable_export` for usage.

Visual AI project support image data for modeling. The modeling must occur within a project that has one dataset used as the source to train the models.

8.1 Create a Visual AI Project

Setting up a Visual AI project requires you to create a dataset. The various ways to do this are covered in detail in DataRobot Platform Documentation, [Using Visual AI, Preparing Your Dataset](#).

For the examples given here the images are partitioned into named directories. The named directories serve as the class names that will be applied to images used in predictions. For example: to predict on images of food found at a baseball game, then some of the directory names might be hotdog, hamburger, and popcorn.

```
/home/user/data/imagedataset
  hamburger
    hamburger01.jpg
    hamburger02.jpg
  hotdog
    hotdog01.jpg
    hotdog02.jpg
```

You then compress the directory containing the named directories into a ZIP file, creating the dataset used for the project.

```
from datarobot.models import Project, Dataset
dataset = Dataset.create_from_file(file_path='/home/user/data/imagedataset.zip')
project = Project.create_from_dataset(dataset.id, project_name='My Image Project')
```

8.1.1 Target

Since this example uses named directories the target name must be `class`, which will contain the name of each directory in the ZIP file.

8.1.2 Other Parameters

Setting modeling parameters, such as partitioning method, queue mode, etc, functions in the same way as starting a non-image project.

8.2 Start Modeling

Once you have set modeling parameters, use the following code structure to specify parameters and start the modeling process.

```
from datarobot import AUTOPILOT_MODE
project.set_target(target='class', mode=AUTOPILOT_MODE.FULL_AUTO)
```

You can also pass optional parameters to `project.set_target` to change aspects of the modeling process. Some of those parameters include:

- `worker_count` – int, sets the number of workers used for modeling.
- `partitioning_method` – `PartitioningMethod` object.

For a full reference of available parameters, see [`Project.set_target`](#).

You can use the `mode` parameter to set the Autopilot mode. `AUTOPILOT_MODE.FULL_AUTO`, is the default, triggers modeling with no further actions necessary. Other accepted modes include `AUTOPILOT_MODE.MANUAL` for manual mode (choose your own models to run rather than running the full Autopilot) and `AUTOPILOT_MODE.QUICK` to run on a more limited set of models and get insights more quickly (â€œquick runâ€).

8.3 Interact with a Visual AI Project

The following code snippets may be used to access Visual AI images and insights.

8.3.1 List Sample Images

Sample images allow you to see a subset of images, chosen by DataRobot, in the dataset. The returned `SampleImage` objects have an associated `target_value` that will allow you to categorize the images (e.g. hamburger or hotdog). Until the project has reached specific stages of modeling the `target_value` will be `None`.

```
import io
import PIL.Image

from datarobot.models import Project
from datarobot.models.visualai import SampleImage

project_name = "My Image Project"
column_name = "image"
```

(continues on next page)

(continued from previous page)

```

project = Project.list(search_params={"project_name": project_name})[0]
for sample in SampleImage.list(project.id, column_name):
    # Display the image in the GUI
    bio = io.BytesIO(sample.image.image_bytes)
    img = PIL.Image.open(bio)
    img.show()

```

The results would be images such as:



8.3.2 List Duplicate Images

Duplicate images, images with different names but are determined by DataRobot to be the same, may exist in a dataset. If this happens, the code returns one of the images and the number of times it occurs in the dataset.

```

from datarobot.models import Project
from datarobot.models.visualai import DuplicateImage

project_name = "My Image Project"
column_name = "image"

project = Project.list(search_params={"project_name": project_name})[0]
for duplicate in DuplicateImage.list(project.id, column_name):
    # To show an image see the previous sample image example
    print(f"Image id = {duplicate.image.id} has {duplicate.count} duplicates")

```

8.3.3 Activation Maps

Activation maps are overlaid on the images to show which images areas the model is using when making predictions. Detailed explanations are available in DataRobot Platform Documentation, [Model insights](#).

Compute Activation Maps

You must compute activation maps before retrieving. The following snippet is an example of starting the computation. For each project and model, DataRobot returns a URL that can be used to determine when the computation is complete.

```
from datarobot.models import Project
from datarobot.models.visualai import ImageActivationMap

project_name = "My Image Project"
column_name = "image"

project = Project.list(search_params={"project_name": project_name})[0]
for model_id, feature_name in ImageActivationMap.models(project.id):
    if feature_name == column_name:
        ImageActivationMap.compute(project.id, model_id)
```

List Activation Maps

After activation maps are computed, you can download them from the DataRobot server. The following snippet is an example of how to get the activation maps for a project and model and print out the `ImageActivationMap` object. The activation map is a 2D matrix of values in the range `[0, 255]`.

```
from datarobot.models import Project
from datarobot.models.visualai import ImageActivationMap

project_name = "My Image Project"
column_name = "image"

project = Project.list(search_params={"project_name": project_name})[0]
for model_id, feature_name in ImageActivationMap.models(project.id):
    for amap in ImageActivationMap.list(project.id, model_id, column_name):
        print(amap)
```

When `ImageActivationMap.activation_values` are used to adjust the brightness of each region, the images would look similar to:





8.3.4 Image Embeddings

Image embeddings map individual images into a vector embedding space. An individual embedding may be used to perform linear computations on the images.

Detailed explanations are available in DataRobot Platform Documentation, [Model insights](#).

Compute Image Embeddings

You must compute image embeddings before retrieving. The following snippet is an example of how to start the computation. For each project and model, DataRobot returns a URL that can be used to determine when the computation is complete.

```
from datarobot.models import Project
from datarobot.models.visualai import ImageEmbedding

project_name = "My Image Project"
column_name = "image"

project = Project.list(search_params={"project_name": project_name})[0]
for model_id, feature_name in ImageEmbedding.models(project.id):
    url = ImageEmbedding.compute(project.id, model_id)
    print(url)
```

List Image Embeddings

After image embeddings are computed, you can download them from the DataRobot server. The following snippet is an example of how to get the embeddings for a project and model and print out the `ImageEmbedding` object.

```
from datarobot.models import Project
from datarobot.models.visualai import ImageEmbedding

project_name = "My Image Project"
column_name = "image"

project = Project.list(search_params={"project_name": project_name})[0]
for model_id, feature_name in ImageEmbedding.models(project.id):
```

(continues on next page)

(continued from previous page)

```
for embedding in ImageEmbedding.list(project.id, model_id, column_name):  
    print(embedding)
```

Unsupervised Projects (Anomaly Detection)

When the data is not labelled and the problem can be interpreted either as anomaly detection or time series anomaly detection, projects in unsupervised mode become useful.

9.1 Creating Unsupervised Projects

In order to create an unsupervised project set `unsupervised_mode` to `True` when setting the target.

```
>>> import datarobot as dr
>>> project = Project.create('dataset.csv', project_name='unsupervised')
>>> project.set_target(unsupervised_mode=True)
```

9.2 Creating Time Series Unsupervised Projects

To create a time series unsupervised project pass `unsupervised_mode=True` to datetime partitioning creation and to project aim. The forecast window will be automatically set to nowcasting, i.e. forecast distance zero (FW = 0, 0).

```
>>> import datarobot as dr
>>> project = Project.create('dataset.csv', project_name='unsupervised')
>>> spec = DatetimePartitioningSpecification('date',
...     use_time_series=True, unsupervised_mode=True,
...     feature_derivation_window_start=-4, feature_derivation_window_end=0)

# this step is optional - preview the default partitioning which will be applied
>>> partitioning_preview = DatetimePartitioning.generate(project.id, spec)
>>> full_spec = partitioning_preview.to_specification()
>>> project.set_target(unsupervised_mode=True, partitioning_method=full_spec)
```

9.3 Unsupervised Project Metrics

In unsupervised projects, metrics are not used for the model optimization. Instead, they are used for the purpose of model ranking. There are two available unsupervised metrics – Synthetic AUC and synthetic LogLoss – both of which are calculated on artificially-labelled validation samples.

9.4 Assessing Unsupervised Anomaly Detection Models on External Test Set

In unsupervised projects, if there is some labelled data, it may be used to assess anomaly detection models by checking computed classification metrics such as AUC and LogLoss, etc. and insights such as ROC and Lift. Such data is uploaded as a prediction dataset with a specified actual value column name, and, if it is a time series project, a prediction date range. The actual value column can contain only zeros and ones or True/False, and it should not have been seen during training time.

9.5 Requesting External Scores and Insights (Time Series)

There are two ways to specify an actual value column and compute scores and insights:

1. Upload a prediction dataset, specifying `predictions_start_date`, `predictions_end_date`, and `actual_value_column`, and request predictions on that dataset using a specific model.

```
>>> import datarobot as dr
# Upload dataset
>>> project = dr.Project(project_id)
>>> dataset = project.upload_dataset(
...     './data_to_predict.csv',
...     predictions_start_date=datetime(2000, 1, 1),
...     predictions_end_date=datetime(2015, 1, 1),
...     actual_value_column='actuals'
... )
# run prediction job which also will calculate requested scores and insights.
>>> predict_job = model.request_predictions(dataset.id)
# prediction output will have column with actuals
>>> result = pred_job.get_result_when_complete()
```

2. Upload a prediction dataset without specifying any options, and request predictions for specific model with `predictions_start_date`, `predictions_end_date`, and `actual_value_column` specified. Note, these settings cannot be changed for the dataset after making predictions.

```
>>> import datarobot as dr
# Upload dataset
>>> project = dr.Project(project_id)
>>> dataset = project.upload_dataset('./data_to_predict.csv')
# Check which columns are candidates for actual value columns
>>> dataset.detected_actual_value_columns
[{'missing_count': 25, 'name': 'label_column'}]

# run prediction job which also will calculate requested scores and insights.
>>> predict_job = model.request_predictions(
...     dataset.id,
...     predictions_start_date=datetime(2000, 1, 1),
```

(continues on next page)

(continued from previous page)

```
...     predictions_end_date=datetime(2015, 1, 1),
...     actual_value_column='label_column'
... )
>>> result = pred_job.get_result_when_complete()
```

9.6 Requesting External Scores and Insights for AutoML models

To compute scores and insights on an external dataset for unsupervised AutoML models (Non Time series)

Upload a prediction dataset that contains label column(s), request compute external test on one of `PredictionDataset.detected_actual_value_columns`

```
import datarobot as dr
# Upload dataset
project = dr.Project(project_id)
dataset = project.upload_dataset('./test_set.csv')
dataset.detected_actual_value_columns
>>> ['label_column_1', 'label_column_2']
# request external test to compute metric scores and insights on dataset
external_test_job = model.request_external_test(dataset.id, actual_value_column=
↳ 'label_column_1')
# once job is complete, scores and insights are ready for retrieving
external_test_job.wait_for_completion()
```

9.7 Retrieving External Scores and Insights

Upon completion of prediction, external scores and insights can be retrieved to assess model performance. For unsupervised projects Lift Chart and ROC Curve are computed. If the dataset is too small insights will not be computed. If the actual value column contained only one class, the ROC Curve will not be computed. Information about the dataset can be retrieved using `PredictionDataset.get`.

```
>>> import datarobot as dr
# Check which columns are candidates for actual value columns
>>> scores_list = ExternalScores.list(project_id)
>>> scores = ExternalScores.get(project_id, dataset_id=dataset_id, model_id=model_id)
>>> lift_list = ExternalLiftChart.list(project_id, model_id)
>>> roc = ExternalRocCurve.get(project_id, model, dataset_id)
# check dataset warnings, need to be called after predictions are computed.
>>> dataset = PredictionDataset.get(project_id, dataset_id)
>>> dataset.data_quality_warnings
{'single_class_actual_value_column': True,
'insufficient_rows_for_evaluating_models': False,
'has_kia_missing_values_in_forecast_window': False}
```


The set of computation paths that a dataset passes through before producing predictions from data is called a blueprint. A blueprint can be trained on a dataset to generate a model.

10.1 Quick Reference

The following code block summarizes the interactions available for blueprints.

```
# Get the set of blueprints recommended by datarobot
import datarobot as dr
my_projects = dr.Project.list()
project = my_projects[0]
menu = project.get_blueprints()

first_blueprint = menu[0]
project.train(first_blueprint)
```

10.2 List Blueprints

When a file is uploaded to a project and the target is set, DataRobot recommends a set of blueprints that are appropriate for the task at hand. You can use the `get_blueprints` method to get the list of blueprints recommended for a project:

```
project = dr.Project.get('5506fcd38bd88f5953219da0')
menu = project.get_blueprints()
blueprint = menu[0]
```

10.3 Get a blueprint

If you already have a `blueprint_id` from a model you can retrieve the blueprint directly.

```
project_id = '5506fcd38bd88f5953219da0'
project = dr.Project.get(project_id)
models = project.get_models()
model = models[0]
blueprint = Blueprint.get(project_id, model.blueprint_id)
```

10.4 Get a blueprint chart

For all blueprints - either from blueprint menu or already used in model - you can retrieve its chart. You can also get its representation in graphviz DOT format to render it into format you need.

```
project_id = '5506fcd38bd88f5953219da0'
blueprint_id = '4321fcd38bd88f595321554223'
bp_chart = BlueprintChart.get(project_id, blueprint_id)
print(bp_chart.to_graphviz())
```

10.5 Get a blueprint documentation

You can retrieve documentation on tasks used in blueprint. It will contain information about task, its parameters and (when available) links and references to additional sources. All documents are instances of `BlueprintTaskDocument` class.

```
project_id = '5506fcd38bd88f5953219da0'
blueprint_id = '4321fcd38bd88f595321554223'
bp = Blueprint.get(project_id, blueprint_id)
docs = bp.get_documents()
print(docs[0].task)
>>> Average Blend
print(docs[0].links[0]['url'])
>>> https://en.wikipedia.org/wiki/Ensemble_learning
```

10.6 Blueprint Attributes

The `Blueprint` class holds the data required to use the blueprint for modeling. This includes the `blueprint_id` and `project_id`. There are also two attributes that help distinguish blueprints: `model_type` and `processes`.

```
print(blueprint.id)
>>> u'8956elaecffa0fa6db2b84640fb3848'
print(blueprint.project_id)
>>> u5506fcd38bd88f5953219da0'
print(blueprint.model_type)
>>> Logistic Regression
print(blueprint.processes)
>>> [u'One-Hot Encoding',
     u'Missing Values Imputed',
```

(continues on next page)

(continued from previous page)

```
u'Standardize',  
u'Logistic Regression']
```

10.7 Create a Model from a Blueprint

You can use a blueprint instance to train a model. The default dataset for the project is used. Note that `Project.train` is used for non-datetime-partitioned projects. `Project.train_datetime` should be used for datetime partitioned projects.

```
model_job_id = project.train(blueprint)  
  
# For datetime partitioned projects  
model_job = project.train_datetime(blueprint.id)
```

Both `Project.train` and `Project.train_datetime` will put a new modeling job into the queue. However, note that `Project.train` returns the id of the created `ModelJob`, while `Project.train_datetime` returns the `ModelJob` object itself. You can pass a `ModelJob` id to `wait_for_async_model_creation` function, which polls the async model creation status and returns the newly created model when it's finished.

When a blueprint has been trained on a specific dataset at a specified sample size, the result is a model. Models can be inspected to analyze their accuracy.

11.1 Start Training a Model

To start training a model, use the `Project.train` method with a blueprint object:

```
import datarobot as dr
project = dr.Project.get('5506fcd38bd88f5953219da0')
blueprints = project.get_blueprints()
model_job_id = project.train(blueprints[0])
```

For a *Datetime Partitioned Project*, use `Project.train_datetime`:

```
import datarobot as dr
project = dr.Project.get('5506fcd38bd88f5953219da0')
blueprints = project.get_blueprints()
model_job_id = project.train_datetime(blueprints[0])
```

11.2 List Finished Models

You can use the `Project.get_models` method to return a list of the project models that have finished training:

```
import datarobot as dr
project = dr.Project.get('5506fcd38bd88f5953219da0')
models = project.get_models()
print(models[:5])
>>> [Model(Decision Tree Classifier (Gini)),
      Model(Auto-tuned K-Nearest Neighbors Classifier (Minkowski Distance)),
```

(continues on next page)

(continued from previous page)

```

        Model(Gradient Boosted Trees Classifier (R)),
        Model(Gradient Boosted Trees Classifier),
        Model(Logistic Regression)]
model = models[0]

project.id
>>> u'5506fcd38bd88f5953219da0'
model.id
>>> u'5506fcd98bd88f1641a720a3'

```

You can pass following parameters to change result:

- `search_params` – dict, used to filter returned projects. Currently you can query models by
 - `name`
 - `sample_pct`
 - `is_starred`
- `order_by` – str or list, if passed returned models are ordered by this attribute or attributes.
- `with_metric` – str, If not *None*, the returned models will only have scores for this metric. Otherwise all the metrics are returned.

List Models Example:

```

import datarobot as dr

dr.Project('5506fcd38bd88f5953219da0').get_models(order_by=['-created_time', 'sample_
↪pct', 'metric'])

# Getting models that contain "Ridge" in name
# and with sample_pct more than 64
dr.Project('5506fcd38bd88f5953219da0').get_models(
    search_params={
        'sample_pct__gt': 64,
        'name': "Ridge"
    })

# Getting models marked as starred
dr.Project('5506fcd38bd88f5953219da0').get_models(
    search_params={
        'is_starred': True
    })

```

11.3 Retrieve a Known Model

If you know the `model_id` and `project_id` values of a model, you can retrieve it directly:

```

import datarobot as dr
project_id = '5506fcd38bd88f5953219da0'
model_id = '5506fcd98bd88f1641a720a3'
model = dr.Model.get(project=project_id,
                      model_id=model_id)

```

You can also use an instance of `Project` as the parameter for `Model.get`

```
model = dr.Model.get(project=project,
                    model_id=model_id)
```

11.4 Train a Model on a Different Sample Size

One of the key insights into a model and the data behind it is how its performance varies with more training data. In Autopilot mode, DataRobot will run at several sample sizes by default, but you can also create a job that will run at a specific sample size. You can also specify featurelist that should be used for training of new model and scoring type. `Model.train` method of `Model` instance will put a new modeling job into the queue and return id of created `ModelJob`. You can pass `ModelJob` id to the `wait_for_async_model_creation` function, which polls async model creation status and returns newly created model when it's finished.

```
import datarobot as dr

model_job_id = model.train(sample_pct=33)

# Retrain a model on a custom featurelist using cross validation.
# Note that you can specify a custom value for `sample_pct`.
model_job_id = model.train(
    sample_pct=55,
    featurelist_id=custom_featurelist.id,
    scoring_type=dr.SCORING_TYPE.cross_validation,
)
```

11.5 Cross-Validating a Model

By default, models are trained using only the first validation partition. To start cross-validation, use the `Model.cross_validate` method:

```
import datarobot as dr

model_job_id = model.cross_validate()
```

For a *Datetime Partitioned Project*, backtesting is the only cross-validation method supported. To run backtesting for a datetime model, use the `DatetimeModel.score_backtests` method:

```
import datarobot as dr

# `model` here must be an instance of `dr.DatetimeModel`.
model_job_id = model.score_backtests()
```

11.6 Find the Features Used

Because each project can have many associated featurelists, it is important to know which features a model requires in order to run. This helps ensure that the necessary features are provided when generating predictions.

```
feature_names = model.get_features_used()
print(feature_names)
>>> ['MonthlyIncome',
```

(continues on next page)

(continued from previous page)

```
'VisitsLast8Weeks',  
'Age']
```

11.7 Feature Impact

Feature Impact measures how much worse a model's error score would be if DataRobot made predictions after randomly shuffling a particular column (a technique sometimes called *Permutation Importance*).

The following example code snippet shows how a featurelist with just the features with the highest feature impact could be created.

```
import datarobot as dr  
  
max_num_features = 10  
time_to_wait_for_impact = 4 * 60 # seconds  
  
feature_impacts = model.get_or_request_feature_impact(time_to_wait_for_impact)  
  
feature_impacts.sort(key=lambda x: x['impactNormalized'], reverse=True)  
final_names = [f['featureName'] for f in feature_impacts[:max_num_features]]  
  
project.create_featurelist('highest_impact', final_names)
```

11.8 Predict new data

After creating models you can use them to generate predictions on new data. See [PredictJob](#) for further information on how to request predictions from a model.

11.9 Model IDs Vs. Blueprint IDs

Each model has both an `model_id` and a `blueprint_id`. What is the difference between these two IDs?

A model is the result of training a blueprint on a dataset at a specified sample percentage. The `blueprint_id` is used to keep track of which blueprint was used to train the model, while the `model_id` is used to locate the trained model in the system.

11.10 Model parameters

Some models can have parameters that provide data needed to reproduce its predictions.

For additional usage information see DataRobot documentation, section [Coefficients](#) tab and pre-processing details.

```
import datarobot as dr  
  
model = dr.Model.get(project=project, model_id=model_id)  
mp = model.get_parameters()
```

(continues on next page)

(continued from previous page)

```
print(mp.derived_features)
>>> [{
    'coefficient': -0.015,
    'originalFeature': u'A1Cresult',
    'derivedFeature': u'A1Cresult->7',
    'type': u'CAT',
    'transformations': [{'name': u'One-hot', 'value': u">7'"}]}
    ]]
```

11.11 Create a Blender

You can blend multiple models; in many cases, the resulting blender model is more accurate than the parent models. To do so you need to select parent models and a blender method from `datarobot.enums.BLENDER_METHOD`. If this is a time series project, only methods in `datarobot.enums.TS_BLENDER_METHOD` are allowed.

Be aware that the tradeoff for better prediction accuracy is bigger resource consumption and slower predictions.

```
import datarobot as dr

pr = dr.Project.get(pid)
models = pr.get_models()
parent_models = [model.id for model in models[:2]]
pr.blend(parent_models, dr.enums.BLENDER_METHOD.AVERAGE)
```

11.12 Lift chart retrieval

You can use Model methods `get_lift_chart` and `get_all_lift_charts` to retrieve lift chart data. First will get it from specific source (validation data, cross validation or holdout, if holdout unlocked) and second will list all available data. Please refer to *Advanced model information* notebook for additional information about lift charts and how they can be visualised.

For multiclass models you can get list of per-class lift charts using Model method `get_multiclass_lift_chart`.

11.13 ROC curve retrieval

Same as with the lift chart you can use Model methods `get_roc_curve` and `get_all_roc_curves` to retrieve ROC curve data. Please refer to *Advanced model information* notebook for additional information about ROC curves and how they can be visualised. More information about working with ROC curves can be found in DataRobot web application documentation section “ROC Curve tab details”.

11.14 Residuals chart retrieval

Just as with the lift and ROC charts, you can use Model methods `get_residuals_chart` and `get_all_residuals_charts` to retrieve residuals chart data. The first will get it from a specific source (validation data, cross-validation data, or holdout, if unlocked). The second will retrieve all available data. Please refer to the *Advanced model information* notebook for more information about residuals charts and how they can be visualised.

11.15 Word Cloud

If your dataset contains text columns, DataRobot can create text processing models that will contain word cloud insight data. An example of such model is any *Auto-Tuned Word N-Gram Text Modeler* model. You can use `Model.get_word_cloud` method to retrieve those insights - it will provide up to 200 most important ngrams in the model and data about their influence. The *Advanced model information* notebook contains examples of how you can use that data and build a visualization in a way similar to how the DataRobot webapp does.

11.16 Scoring Code

Subset of models in DataRobot supports code generation. For each of those models you can download a JAR file with scoring code to make predictions locally using method `Model.download_scoring_code`. For details on how to do that see *Code Generation* section in DataRobot web application documentation. Optionally you can download source code in Java to see what calculations those models do internally.

Be aware that source code JAR isn't compiled so it cannot be used for making predictions.

11.17 Get a model blueprint chart

For all models you can retrieve its blueprint chart. You can also get its representation in graphviz DOT format to render it into format you need.

```
import datarobot as dr
project_id = '5506fcd38bd88f5953219da0'
model_id = '5506fcd98bd88f1641a720a3'
model = dr.Model.get(project=project_id,
                     model_id=model_id)
bp_chart = model.get_model_blueprint_chart()
print(bp_chart.to_graphviz())
```

11.18 Get a model missing values report

For the majority of models you can retrieve their missing values reports on training data per each numeric and categorical feature. Model needs to have at least one of the supported tasks in the blueprint in order to have a missing values report (blenders are not supported). Report is gathered for Numerical Imputation tasks and Categorical converters like Ordinal Encoding, One-Hot Encoding etc. Missing values report is available to users with access to full blueprint docs.

Report is collected for those features which are considered eligible by given blueprint task. For instance, categorical feature with a lot of unique values may not be considered as eligible in the One-Hot encoding task.

Please refer to *Missing report attributes description* for report interpretation.

```
import datarobot as dr
project_id = '5506fcd38bd88f5953219da0'
model_id = '5506fcd98bd88f1641a720a3'
model = dr.Model.get(project=project_id, model_id=model_id)
missing_reports_per_feature = model.get_missing_report_info()
for report_per_feature in missing_reports_per_feature:
    print(report_per_feature)
```


Consider following example. Given Decision Tree Classifier (Gini) blueprint chart representation:

```
print(blueprint_chart.to_graphviz())
>>> digraph "Blueprint Chart" {
    graph [rankdir=LR]
    0 [label="Data"]
    -2 [label="Numeric Variables"]
    2 [label="Missing Values Imputed"]
    3 [label="Decision Tree Classifier (Gini)"]
    4 [label="Prediction"]
    -1 [label="Categorical Variables"]
    1 [label="Ordinal encoding of categorical variables"]
    0 -> -2
    -2 -> 2
    2 -> 3
    3 -> 4
    0 -> -1
    -1 -> 1
    1 -> 3
}
```

and missing report:

```
print(report_per_feature1)
>>> {'feature': 'Veh Year',
    'type': 'Numeric',
    'missing_count': 150,
    'missing_percentage': 50.00,
    'tasks': [
        {'id': u'2',
         'name': u'Missing Values Imputed',
         'descriptions': [u'Imputed value: 2006']}
    ]
}

print(report_per_feature2)
>>> {'feature': 'Model',
    'type': 'Categorical',
    'missing_count': 100,
    'missing_percentage': 33.33,
    'tasks': [
        {'id': u'1',
         'name': u'Ordinal encoding of categorical variables',
         'descriptions': [u'Imputed value: -2']}
    ]
}
```

results can be interpreted in the following way:

Numeric feature "Veh Year" has 150 missing values and respectively 50% in training data. It was transformed by "Missing Values Imputed" task with imputed value 2006. Task has id 2, and its output goes into Decision Tree Classifier (Gini) - it can be inferred from the chart.

Categorical feature "Model" was transformed by "Ordinal encoding of categorical variables" task with imputed value -2.

11.19 Get a blueprint documentation

You can retrieve documentation on tasks used to build a model. It will contain information about task, its parameters and (when available) links and references to additional sources. All documents are instances of `BlueprintTaskDocument` class.

```
import datarobot as dr
project_id = '5506fcd38bd88f5953219da0'
model_id = '5506fcd98bd88f1641a720a3'
model = dr.Model.get(project=project_id,
                     model_id=model_id)
docs = model.get_model_blueprint_documents()
print(docs[0].task)
>>> Average Blend
print(docs[0].links[0]['url'])
>>> https://en.wikipedia.org/wiki/Ensemble_learning
```

11.20 Request training predictions

You can request a model's predictions for a particular subset of its training data. See `datarobot.models.Model.request_training_predictions()` reference for all the valid subsets.

See *training predictions reference* for more details.

```
import datarobot as dr
project_id = '5506fcd38bd88f5953219da0'
model_id = '5506fcd98bd88f1641a720a3'
model = dr.Model.get(project=project_id,
                     model_id=model_id)
training_predictions_job = model.request_training_predictions(dr.enums.DATA_SUBSET.
    ↳HOLDOUT)
training_predictions = training_predictions_job.get_result_when_complete()
for row in training_predictions.iterate_rows():
    print(row.row_id, row.prediction)
```

11.21 Advanced Tuning

You can perform advanced tuning on a model – generate a new model by taking an existing model and rerunning it with modified tuning parameters.

The `AdvancedTuningSession` class exists to track the creation of an Advanced Tuning model on the client. It enables browsing and setting advanced-tuning parameters one at a time, and using human-readable parameter names rather than requiring opaque parameter IDs in all cases. No information is sent to the server until the `run()` method is called on the `AdvancedTuningSession`.

See `datarobot.models.Model.get_advanced_tuning_parameters()` reference for a description of the types of parameters that can be passed in.

As of v2.17, all models other than blenders, open source, and user-created models support Advanced Tuning. The use of Advanced Tuning via API for non-Eureqa models is in beta, but is enabled by default for all users.

```

import datarobot as dr
project_id = '5506fcd38bd88f5953219da0'
model_id = '5506fcd98bd88f1641a720a3'
model = dr.Model.get(project=project_id,
                     model_id=model_id)
tune = model.start_advanced_tuning_session()

# Get available task names,
# and available parameter names for a task name that exists on this model
tune.get_task_names()
tune.get_parameter_names('Eureqa Generalized Additive Model Classifier (3000_
↳Generations)')

tune.set_parameter(
    task_name='Eureqa Generalized Additive Model Classifier (3000 Generations)',
    parameter_name='EUREQA_building_block__sine',
    value=1)

job = tune.run()

```

11.22 SHAP Impact

You can retrieve SHAP impact scores for features in a model. SHAP impact is computed by calculating the shap values on a sample of training data and then taking the mean absolute value for each column. The larger value of impact indicate more important feature.

See `datarobot.models.ShapImpact.create()` reference for a description of the types of parameters that can be passed in.

```

import datarobot as dr

project_id = '5ec3d6884cfad17cd8c0ed62'
model_id = '5ec3d6f44cfad17cd8c0ed78'
shap_impact_job = dr.ShapImpact.create(project_id=project_id, model_id=model_id)
shap_impact = shap_impact_job.get_result_when_complete()
print(shap_impact)
>>> [ShapImpact(count=36)]
print(shap_impact.shap_impacts[:1])
>>> [{'feature_name': 'number_inpatient', 'impact_normalized': 1.0, 'impact_
↳unnormalized': 0.07670175497683789}]

shap_impact = dr.ShapImpact.get(project_id=project_id, model_id=model_id)
print(shap_impact.shap_impacts[:1])
>>> [{'feature_name': 'number_inpatient', 'impact_normalized': 1.0, 'impact_
↳unnormalized': 0.07670175497683789}]

```

11.23 Number of Iterations Trained

Early-stopping models will train a subset of max estimators/iterations that are defined in advanced tuning. This method allows the user to retrieve the actual number of estimators that were trained by an early-stopping tree-based model (currently the only model type supported). The method returns the projectId, modelId, and a list of dictionaries containing the number of iterations trained for each model stage. In the case of single-stage models, this dictionary will contain only one entry.

```
import datarobot as dr
project_id = '5506fcd38bd88f5953219da0'
model_id = '5506fcd98bd88f1641a720a3'
model = dr.Model.get(project=project_id,
                     model_id=model_id)
num_iterations = model.get_num_iterations_trained()
print(num_iterations)
>>> {"projectId": "5506fcd38bd88f5953219da0", "modelId": "5506fcd98bd88f1641a720a3",
↪ "data" [{"stage": "FREQ", "numIterations": 250}, {"stage": "SEV", "numIterations": 50}
↪ ]}
```

The *Job* class is a generic representation of jobs running through a project's queue. Many tasks involved in modeling, such as creating a new model or computing feature impact for a model, will use a job to track the worker usage and progress of the associated task.

12.1 Checking the Contents of the Queue

To see what jobs running or waiting in the queue for a project, use the `Project.get_all_jobs` method.

```
from datarobot.enums import QUEUE_STATUS

jobs_list = project.get_all_jobs() # gives all jobs queued or inprogress
jobs_by_type = {}
for job in jobs_list:
    if job.job_type not in jobs_by_type:
        jobs_by_type[job.job_type] = [0, 0]
    if job.status == QUEUE_STATUS.QUEUE:
        jobs_by_type[job.job_type][0] += 1
    else:
        jobs_by_type[job.job_type][1] += 1
for type in jobs_by_type:
    (num_queued, num_inprogress) = jobs_by_type[type]
    print('{} jobs: {} queued, {} inprogress'.format(type, num_queued, num_
    ↪ inprogress))
```

12.2 Cancelling a Job

If a job is taking too long to run or no longer necessary, it can be cancelled easily from the `Job` object.

```
from datarobot.enums import QUEUE_STATUS

project.pause_autopilot()
bad_jobs = project.get_all_jobs(status=QUEUE_STATUS.QUEUE)
for job in bad_jobs:
    job.cancel()
project.unpause_autopilot()
```

12.3 Retrieving Results From a Job

Once you’ve found a particular job of interest, you can retrieve the results once it is complete. Note that the type of the returned object will vary depending on the `job_type`. All return types are documented in `Job.get_result`.

```
from datarobot.enums import JOB_TYPE

time_to_wait = 60 * 60 # how long to wait for the job to finish (in seconds) - i.e.
↳ an hour
assert my_job.job_type == JOB_TYPE.MODEL
my_model = my_job.get_result_when_complete(max_wait=time_to_wait)
```

Model creation is asynchronous process. This means that when explicitly invoking new model creation (with `project.train` or `model.train` for example) all you get is id of process, responsible for model creation. With this id you can get info about model that is being created or the model itself, when creation process is finished. For this you should use the `ModelJob` class.

13.1 Get an existing ModelJob

To retrieve existing `ModelJob` use `ModelJob.get` method. For this you need id of Project that is used for model creation and id of `ModelJob`. Having `ModelJob` might be useful if you want to know parameters of model creation, automatically chosen by API backend, before actual model was created.

If model is already created, `ModelJob.get` will raise `PendingJobFinished` exception

```
import time

import datarobot as dr

blueprint_id = '5506fcd38bd88f5953219da0'
model_job_id = project.train(blueprint_id)
model_job = dr.ModelJob.get(project_id=project.id,
                             model_job_id=model_job_id)

model_job.sample_pct
>>> 64.0

# wait for model to be created (in a very inefficient way)
time.sleep(10 * 60)
model_job = dr.ModelJob.get(project_id=project.id,
                             model_job_id=model_job_id)

>>> datarobot.errors.PendingJobFinished

# get the job attached to the model
```

(continues on next page)

(continued from previous page)

```
model_job.model
>>> Model('5d518cd3962d741512605e2b')
```

13.2 Get created model

After model is created, you can use `ModelJob.get_model` to get newly created model.

```
import datarobot as dr

model = dr.ModelJob.get_model(project_id=project.id,
                              model_job_id=model_job_id)
```

13.3 wait_for_async_model_creation function

If you just want to get the created model after getting the `ModelJob` id, you can use the [*wait_for_async_model_creation*](#) function. It will poll for the status of the model creation process until it's finished, and then will return the newly created model. Note the differences below between datetime partitioned projects and non-datetime-partitioned projects.

```
from datarobot.models.modeljob import wait_for_async_model_creation

# used during training based on blueprint
model_job_id = project.train(blueprint, sample_pct=33)
new_model = wait_for_async_model_creation(
    project_id=project.id,
    model_job_id=model_job_id,
)

# used during training based on existing model
model_job_id = existing_model.train(sample_pct=33)
new_model = wait_for_async_model_creation(
    project_id=existing_model.project_id,
    model_job_id=model_job_id,
)

# For datetime-partitioned projects, use project.train_datetime. Note that train_
↳ datetime returns a ModelJob instead
# of just an id.
model_job = project.train_datetime(blueprint)
new_model = wait_for_async_model_creation(
    project_id=project.id,
    model_job_id=model_job.id
)
```


Predictions generation is an asynchronous process. This means that when starting predictions with `Model.request_predictions` you will receive back a `PredictJob` for tracking the process responsible for fulfilling your request.

With this object you can get info about the predictions generation process before it has finished and be rerouted to the predictions themselves when the process is finished. For this you should use the *PredictJob* class.

14.1 Starting predictions generation

Before actually requesting predictions, you should upload the dataset you wish to predict via `Project.upload_dataset`. Previously uploaded datasets can be seen under `Project.get_datasets`. When uploading the dataset you can provide the path to a local file, a file object, raw file content, a `pandas.DataFrame` object, or the url to a publicly available dataset.

To start predicting on new data using a finished model use `Model.request_predictions`. It will create a new predictions generation process and return a `PredictJob` object tracking this process. With it, you can monitor an existing `PredictJob` and retrieve generated predictions when the corresponding `PredictJob` is finished.

```
import datarobot as dr

project_id = '5506fcd38bd88f5953219da0'
model_id = '5506fcd98bd88f1641a720a3'
project = dr.Project.get(project_id)
model = dr.Model.get(project=project_id,
                     model_id=model_id)

# Using path to local file to generate predictions
dataset_from_path = project.upload_dataset('./data_to_predict.csv')

# Using file object to generate predictions
with open('./data_to_predict.csv') as data_to_predict:
    dataset_from_file = project.upload_dataset(data_to_predict)
```

(continues on next page)

(continued from previous page)

```
predict_job_1 = model.request_predictions(dataset_from_path.id)
predict_job_2 = model.request_predictions(dataset_from_file.id)
```

14.2 Listing Predictions

You can use the `Predictions.list()` method to return a list of predictions generated on a project.

```
import datarobot as dr
predictions = dr.Predictions.list('58591727100d2b57196701b3')

print(predictions)
>>> [Predictions(prediction_id='5b6b163eca36c0108fc5d411',
                  project_id='5b61bd68ca36c04aed8aab7f',
                  model_id='5b61bd7aca36c05744846630',
                  dataset_id='5b6b1632ca36c03b5875e6a0'),
      Predictions(prediction_id='5b6b2315ca36c0108fc5d41b',
                  project_id='5b61bd68ca36c04aed8aab7f',
                  model_id='5b61bd7aca36c0574484662e',
                  dataset_id='5b6b1632ca36c03b5875e6a0'),
      Predictions(prediction_id='5b6b23b7ca36c0108fc5d422',
                  project_id='5b61bd68ca36c04aed8aab7f',
                  model_id='5b61bd7aca36c0574484662e',
                  dataset_id='5b6b1632ca36c03b5875e6a0')]
]
```

You can pass following parameters to filter the result:

- `model_id` – str, used to filter returned predictions by `model_id`.
- `dataset_id` – str, used to filter returned predictions by `dataset_id`.

14.3 Get an existing PredictJob

To retrieve an existing `PredictJob` use the `PredictJob.get` method. This will give you a `PredictJob` matching the latest status of the job if it has not completed.

If predictions have finished building, `PredictJob.get` will raise a `PendingJobFinished` exception.

```
import time

import datarobot as dr

predict_job = dr.PredictJob.get(project_id=project_id,
                                predict_job_id=predict_job_id)

predict_job.status
>>> 'queue'

# wait for generation of predictions (in a very inefficient way)
time.sleep(10 * 60)
predict_job = dr.PredictJob.get(project_id=project_id,
                                predict_job_id=predict_job_id)
>>> dr.errors.PendingJobFinished
```

(continues on next page)

(continued from previous page)

```
# now the predictions are finished
predictions = dr.PredictJob.get_predictions(project_id=project.id,
                                           predict_job_id=predict_job_id)
```

14.4 Get generated predictions

After predictions are generated, you can use `PredictJob.get_predictions` to get newly generated predictions.

If predictions have not yet been finished, it will raise a `JobNotFinished` exception.

```
import datarobot as dr

predictions = dr.PredictJob.get_predictions(project_id=project.id,
                                           predict_job_id=predict_job_id)
```

14.5 Wait for and Retrieve results

If you just want to get generated predictions from a `PredictJob`, you can use the `PredictJob.get_result_when_complete` function. It will poll the status of predictions generation process until it has finished, and then will return predictions.

```
dataset = project.get_datasets()[0]
predict_job = model.request_predictions(dataset.id)
predictions = predict_job.get_result_when_complete()
```

14.6 Get previously generated predictions

If you don't have a `Model.predict_job` on hand, there are two more ways to retrieve predictions from the Predictions interface:

1. Get all prediction rows as a `pandas.DataFrame` object:

```
import datarobot as dr

preds = dr.Predictions.get("5b61bd68ca36c04aed8aab7f", prediction_id=
    ↪ "5b6b163eca36c0108fc5d411")
df = preds.get_all_as_dataframe()
df_with_serializer = preds.get_all_as_dataframe(serializer='csv')
```

2. Download all prediction rows to a file as a CSV document:

```
import datarobot as dr

preds = dr.Predictions.get("5b61bd68ca36c04aed8aab7f", prediction_id=
    ↪ "5b6b163eca36c0108fc5d411")
preds.download_to_csv('predictions.csv')

preds.download_to_csv('predictions_with_serializer.csv', serializer='csv')
```

Prediction Explanations

To compute prediction explanations you need to have *feature impact* computed for a model, and *predictions* for an uploaded dataset computed with a selected model.

Computing prediction explanations is a resource-intensive task, but you can configure it with maximum explanations per row and prediction value thresholds to speed up the process.

15.1 Quick Reference

```
import datarobot as dr
# Get project
my_projects = dr.Project.list()
project = my_projects[0]
# Get model
models = project.get_models()
model = models[0]
# Compute feature impact
feature_impacts = model.get_or_request_feature_impact()
# Upload dataset
dataset = project.upload_dataset('./data_to_predict.csv')
# Compute predictions
predict_job = model.request_predictions(dataset.id)
predict_job.wait_for_completion()
# Initialize prediction explanations
pei_job = dr.PredictionExplanationsInitialization.create(project.id, model.id)
pei_job.wait_for_completion()
# Compute prediction explanations with default parameters
pe_job = dr.PredictionExplanations.create(project.id, model.id, dataset.id)
pe = pe_job.get_result_when_complete()
# Iterate through predictions with prediction explanations
for row in pe.get_rows():
    print(row.prediction)
    print(row.prediction_explanations)
```

(continues on next page)

(continued from previous page)

```
# download to a CSV file
pe.download_to_csv('prediction_explanations.csv')
```

15.2 List Prediction Explanations

You can use the `PredictionExplanations.list()` method to return a list of prediction explanations computed for a project's models:

```
import datarobot as dr
prediction_explanations = dr.PredictionExplanations.list('58591727100d2b57196701b3')
print(prediction_explanations)
>>> [PredictionExplanations(id=585967e7100d2b6afc93b13b,
                             project_id=58591727100d2b57196701b3,
                             model_id=585932c5100d2b7c298b8acf),
      PredictionExplanations(id=58596bc2100d2b639329eae4,
                             project_id=58591727100d2b57196701b3,
                             model_id=585932c5100d2b7c298b8acf),
      PredictionExplanations(id=58763db4100d2b66759cc187,
                             project_id=58591727100d2b57196701b3,
                             model_id=585932c5100d2b7c298b8acf),
      ...]
pe = prediction_explanations[0]

pe.project_id
>>> u'58591727100d2b57196701b3'
pe.model_id
>>> u'585932c5100d2b7c298b8acf'
```

You can pass following parameters to filter the result:

- `model_id` – str, used to filter returned prediction explanations by `model_id`.
- `limit` – int, limit for number of items returned, default: no limit.
- `offset` – int, number of items to skip, default: 0.

List Prediction Explanations Example:

```
project_id = '58591727100d2b57196701b3'
model_id = '585932c5100d2b7c298b8acf'
dr.PredictionExplanations.list(project_id, model_id=model_id, limit=20, offset=100)
```

15.3 Initialize Prediction Explanations

In order to compute prediction explanations you have to initialize it for a particular model.

```
dr.PredictionExplanationsInitialization.create(project_id, model_id)
```

15.4 Compute Prediction Explanations

If all prerequisites are in place, you can compute prediction explanations in the following way:

```
import datarobot as dr
project_id = '5506fcd38bd88f5953219da0'
model_id = '5506fcd98bd88f1641a720a3'
dataset_id = '5506fcd98bd88a8142b725c8'
pe_job = dr.PredictionExplanations.create(project_id, model_id, dataset_id,
                                          max_explanations=2, threshold_low=0.2, threshold_
                                          ↪high=0.8)
pe = pe_job.get_result_when_complete()
```

Where:

- `max_explanations` are the maximum number of prediction explanations to compute for each row.
- `threshold_low` and `threshold_high` are thresholds for the value of the prediction of the row. Prediction explanations will be computed for a row if the row's prediction value is higher than `threshold_high` or lower than `threshold_low`. If no thresholds are specified, prediction explanations will be computed for all rows.

15.5 Retrieving Prediction Explanations

You have three options for retrieving prediction explanations.

Note: `PredictionExplanations.get_all_as_dataframe()` and `PredictionExplanations.download_to_csv()` reformat prediction explanations to match the schema of CSV file downloaded from UI (RowId, Prediction, Explanation 1 Strength, Explanation 1 Feature, Explanation 1 Value, â€¦, Explanation N Strength, Explanation N Feature, Explanation N Value)

Get prediction explanations rows one by one as *PredictionExplanationsRow* objects:

```
import datarobot as dr
project_id = '5506fcd38bd88f5953219da0'
prediction_explanations_id = '5506fcd98bd88f1641a720a3'
pe = dr.PredictionExplanations.get(project_id, prediction_explanations_id)
for row in pe.get_rows():
    print(row.prediction_explanations)
```

Get all rows as `pandas.DataFrame`:

```
import datarobot as dr
project_id = '5506fcd38bd88f5953219da0'
prediction_explanations_id = '5506fcd98bd88f1641a720a3'
pe = dr.PredictionExplanations.get(project_id, prediction_explanations_id)
prediction_explanations_df = pe.get_all_as_dataframe()
```

Download all rows to a file as CSV document:

```
import datarobot as dr
project_id = '5506fcd38bd88f5953219da0'
prediction_explanations_id = '5506fcd98bd88f1641a720a3'
pe = dr.PredictionExplanations.get(project_id, prediction_explanations_id)
pe.download_to_csv('prediction_explanations.csv')
```

15.6 Adjusted Predictions In Prediction Explanations

In some projects such as insurance projects, the prediction adjusted by exposure is more useful compared with raw prediction. For example, the raw prediction (e.g. claim counts) is divided by exposure (e.g. time) in the project with exposure column. The adjusted prediction provides insights with regard to the predicted claim counts per unit of time. To include that information, set `exclude_adjusted_predictions` to `False` in correspondent method calls.

```
import datarobot as dr
project_id = '5506fcd38bd88f5953219da0'
prediction_explanations_id = '5506fcd98bd88f1641a720a3'
pe = dr.PredictionExplanations.get(project_id, prediction_explanations_id)
pe.download_to_csv('prediction_explanations.csv', exclude_adjusted_predictions=False)
prediction_explanations_df = pe.get_all_as_dataframe(exclude_adjusted_
↳ predictions=False)
```

15.7 Deprecated Reason Codes Interface

This feature was previously referred to using the *Reason Codes API*. This interface is now deprecated and should be replaced with the *Prediction Explanations* interface.

15.8 SHAP based prediction explanations

You can request SHAP based prediction explanations using previously uploaded scoring dataset for models that support SHAP. Unlike for XEMP prediction explanations you do not need to have *feature impact* computed for a model, and *predictions* for an uploaded dataset.

See `datarobot.models.ShapMatrix.create()` reference for a description of the types of parameters that can be passed in.

```
import datarobot as dr
project_id = '5ea6d3354cfad121cf33a5e1'
model_id = '5ea6d38b4cfad121cf33a60d'
project = dr.Project.get(project_id)
model = dr.Model.get(project=project_id, model_id=model_id)
# check if model supports SHAP
model_capabilities = model.get_supported_capabilities()
print(model_capabilities.get('supportsShap'))
>>> True
# upload dataset to generate prediction explanations
dataset_from_path = project.upload_dataset('./data_to_predict.csv')

shap_matrix_job = ShapMatrix.create(project_id=project_id, model_id=model_id, dataset_
↳ id=dataset_from_path.id)
shap_matrix_job
>>> Job(shapMatrix, status=inprogress)
# wait for job to finish
shap_matrix = shap_matrix_job.get_result_when_complete()
shap_matrix
>>> ShapMatrix(id='5ea84b624cfad1361c53f65d', project_id='5ea6d3354cfad121cf33a5e1',
↳ model_id='5ea6d38b4cfad121cf33a60d', dataset_id='5ea84b464cfad1361c53f655')

# retrieve SHAP matrix as pandas.DataFrame
```

(continues on next page)

(continued from previous page)

```
df = shap_matrix.get_as_dataframe()

# list as available SHAP matrices for a project
shap_matrices = dr.ShapMatrix.list(project_id)
shap_matrices
>>> [ShapMatrix(id='5ea84b624cfad1361c53f65d', project_id='5ea6d3354cfad121cf33a5e1',
↳ model_id='5ea6d38b4cfad121cf33a60d', dataset_id='5ea84b464cfad1361c53f655')]

shap_matrix = shap_matrices[0]
# retrieve SHAP matrix as pandas.DataFrame
df = shap_matrix.get_as_dataframe()
```


CHAPTER 16

Batch Predictions

The Batch Prediction API provides a way to score large datasets using flexible options for intake and output on the Prediction Servers you have already deployed.

The main features are:

- Flexible options for intake and output.
- Stream local files and start scoring while still uploading - while simultaneously downloading the results.
- Score large datasets from and to S3.
- Connect to your database using JDBC with bidirectional streaming of scoring data and results.
- Intake and output options can be mixed and doesn't need to match. So scoring from a JDBC source to an S3 target is also an option.
- Protection against overloading your prediction servers with the option to control the concurrency level for scoring.
- Prediction Explanations can be included (with option to add thresholds).
- Passthrough Columns are supported to correlate scored data with source data.
- Prediction Warnings can be included in the output.

To interact with Batch Predictions, you should use the *BatchPredictionJob* class.

16.1 Scoring local CSV files

We provide a small utility function for scoring from/to local CSV files: `BatchPredictionJob.score_to_file()`. The first parameter can be either:

- Path to a CSV dataset
- File-like object
- Pandas DataFrame

For larger datasets, you should avoid using a `DataFrame`, as that will load the entire dataset into memory. The other options don't.

```
import datarobot as dr

deployment_id = '5dc5b1015e6e762a6241f9aa'

dr.BatchPredictionJob.score_to_file(
    deployment_id,
    './data_to_predict.csv',
    './predicted.csv',
)
```

The input file will be streamed to our API and scoring will start immediately. As soon as results start coming in, we will initiate the download concurrently. The entire call will block until the file has been scored.

16.2 Scoring from and to S3

We provide a small utility function for scoring from/to CSV files hosted on S3: `BatchPredictionJob.score_s3()`. This requires that the intake and output buckets share the same credentials (see [Credentials](#)) or are public:

```
import datarobot as dr

deployment_id = '5dc5b1015e6e762a6241f9aa'

cred = dr.Credential.get('5a8ac9ab07a57a0001be501f')

dr.BatchPredictionJob.score_s3(
    deployment_id=deployment_id,
    's3://mybucket/data_to_predict.csv',
    's3://mybucket/predicted.csv',
    credential=cred,
)
```

Note: The S3 output functionality has a limit of 100 GB.

16.3 Wiring a Batch Prediction Job manually

If you can't use any of the utilities above, you are also free to configure your job manually. This requires configuring an intake and output option:

```
import datarobot as dr

deployment_id = '5dc5b1015e6e762a6241f9aa'

dr.BatchPredictionJob.score(
    deployment_id,
    intake_settings={
        'type': 's3',
        'url': 's3://public-bucket/data_to_predict.csv',
    },
)
```

(continues on next page)

(continued from previous page)

```

        'credential_id': '5a8ac9ab07a57a0001be501f',
    },
    output_settings={
        'type': 'localFile',
        'path': './predicted.csv',
    },
)

```

Credentials may be created with *Credentials API*.

16.3.1 Supported intake types

These are the supported intake types and descriptions of their configuration parameters:

Local file intake

This requires you to pass either a path to a CSV dataset, file-like object or a Pandas DataFrame as the `file` parameter:

```

intake_settings={
    'type': 'localFile',
    'file': './data_to_predict.csv',
}

```

S3 CSV intake

This requires you to pass an S3 URL to the CSV file your scoring in the `url` parameter:

```

intake_settings={
    'type': 's3',
    'url': 's3://public-bucket/data_to_predict.csv',
}

```

If the bucket is not publicly accessible, you can supply AWS credentials using the three parameters:

- `aws_access_key_id`
- `aws_secret_access_key`
- `aws_session_token`

And save it to the *Credential API*. Here is an example:

```

import datarobot as dr

# get to make sure it exists
cred = dr.Credential.get(credential_id)

intake_settings={
    'type': 's3',
    'url': 's3://private-bucket/data_to_predict.csv',
    'credential_id': cred.credential_id,
}

```

JDBC intake

This requires you to create a *DataStore* and *Credential* for your database:

```
# get to make sure it exists
data_store = dr.DataStore.get(datastore_id)
cred = dr.Credential.get(credential_id)

intake_settings = {
    'type': 'jdbc',
    'table': 'table_name',
    'schema': 'public', # optional, if supported by database
    'catalog': 'master', # optional, if supported by database
    'data_store_id': data_store.id,
    'credential_id': cred.credential_id,
}
```

AI Catalog intake

This requires you to create a *Dataset* and identify the *dataset_id* of that to use as input.

```
# get to make sure it exists
dataset = dr.Dataset.get(dataset_id)

intake_settings={
    'type': 'dataset',
    'dataset': dataset
}
```

Or, in case you want another *version_id* than the latest, supply your own.

```
# get to make sure it exists
dataset = dr.Dataset.get(dataset_id)

intake_settings={
    'type': 'dataset',
    'dataset': dataset,
    'dataset_version_id': 'another_version_id'
}
```

16.3.2 Supported output types

These are the supported output types and descriptions of their configuration parameters:

Local file output

For local file output you have two options. You can either pass a *path* parameter and have the client block and download the scored data concurrently. This is the fastest way to get predictions as it will upload, score and download concurrently:

```
output_settings={
    'type': 'localFile',
    'path': './predicted.csv',
}
```

Another option is to leave out the parameter and subsequently call `BatchPredictionJob.download()` at your own convenience. The `score()` call will then return as soon as the upload is complete.

If the job is not finished scoring, the call to `BatchPredictionJob.download()` will start streaming the data that has been scored so far and block until more data is available.

You can poll for job completion using `BatchPredictionJob.get_status()` or use `BatchPredictionJob.wait_for_completion()` to wait.

```
import datarobot as dr

deployment_id = '5dc5b1015e6e762a6241f9aa'

job = dr.BatchPredictionJob.score(
    deployment_id,
    intake_settings={
        'type': 'localFile',
        'file': './data_to_predict.csv',
    },
    output_settings={
        'type': 'localFile',
    },
)

job.wait_for_completion()

with open('./predicted.csv', 'wb') as f:
    job.download(f)
```

S3 CSV output

This requires you to pass an S3 URL to the CSV file where the scored data should be saved to in the `url` parameter:

```
output_settings={
    'type': 's3',
    'url': 's3://public-bucket/predicted.csv',
}
```

Most likely, the bucket is not publically accessible for writes, but you can supply AWS credentials using the three parameters:

- `aws_access_key_id`
- `aws_secret_access_key`
- `aws_session_token`

And save it to the *Credential API*. Here is an example:

```
# get to make sure it exists
cred = dr.Credential.get(credential_id)

output_settings={
    'type': 's3',
    'url': 's3://private-bucket/predicted.csv',
    'credential_id': cred.credential_id,
}
```

JDBC output

Same as for the input, this requires you to create a *DataStore* and *Credential* for your database, but for *output_settings* you also need to specify *statementType*, which should be one of `datarobot.enums.AVAILABLE_STATEMENT_TYPES`:

```
# get to make sure it exists
data_store = dr.DataStore.get(datastore_id)
cred = dr.Credential.get(credential_id)

output_settings = {
    'type': 'jdbc',
    'table': 'table_name',
    'schema': 'public', # optional, if supported by database
    'catalog': 'master', # optional, if supported by database
    'statementType': 'insert',
    'data_store_id': data_store.id,
    'credential_id': cred.credential_id,
}
```

16.4 Copying a previously submitted job

We provide a small utility function for submitting a job using parameters from a job previously submitted: `BatchPredictionJob.score_from_existing()`. The first parameter is the job id of another job.

```
import datarobot as dr

previously_submitted_job_id = '5dc5b1015e6e762a6241f9aa'

dr.BatchPredictionJob.score_to_file(
    previously_submitted_job_id,
)
```


CHAPTER 17

DataRobot Prime

DataRobot Prime allows the download of executable code approximating models. For more information about this feature, see the documentation within the DataRobot webapp. Contact your Account Executive or CFDS for information on enabling DataRobot Prime, if needed.

17.1 Approximate a Model

Given a `Model` you wish to approximate, `Model.request_approximation` will start a job creating several `Ruleset` objects approximating the parent model. Each of those rulesets will identify how many rules were used to approximate the model, as well as the validation score the approximation achieved.

```
rulesets_job = model.request_approximation()
rulesets = rulesets_job.get_result_when_complete()
for ruleset in rulesets:
    info = (ruleset.id, ruleset.rule_count, ruleset.score)
    print('id: {}, rule_count: {}, score: {}'.format(*info))
```

17.2 Prime Models vs. Models

Given a ruleset, you can create a model based on that ruleset. We consider such models to be Prime models. The `PrimeModel` class inherits from the `Model` class, so anything a `Model` can do, as `PrimeModel` can do as well.

The `PrimeModel` objects available within a `Project` can be listed by `project.get_prime_models`, or a particular one can be retrieve via `PrimeModel.get`. If a ruleset has not yet had a model built for it, `ruleset.request_model` can be used to start a job to make a `PrimeModel` using a particular ruleset.

```
rulesets = parent_model.get_rulesets()
selected_ruleset = sorted(rulesets, key=lambda x: x.score)[-1]
if selected_ruleset.model_id:
    prime_model = PrimeModel.get(selected_ruleset.project_id, selected_ruleset.model_
↪ id)
```

(continues on next page)

(continued from previous page)

```
else:
    prime_job = selected_ruleset.request_model()
    prime_model = prime_job.get_result_when_complete()
```

The `PrimeModel` class has two additional attributes and one additional method. The attributes are `ruleset`, which is the `Ruleset` used in the `PrimeModel`, and `parent_model_id` which is the id of the model it approximates.

Finally, the new method defined is `request_download_validation` which is used to prepare code download for the model and is discussed later on in this document.

17.3 Retrieving Code from a PrimeModel

Given a `PrimeModel`, you can download the code used to approximate the parent model, and view and execute it locally.

The first step is to validate the `PrimeModel`, which runs some basic validation of the generated code, as well as preparing it for download. We use the `PrimeFile` object to represent code that is ready to download. `PrimeFiles` can be prepared by the `request_download_validation` method on `PrimeModel` objects, and listed from a project with the `get_prime_files` method.

Once you have a `PrimeFile` you can check the `is_valid` attribute to verify the code passed basic validation, and then download it to a local file with `download`.

```
validation_job = prime_model.request_download_validation(enums.PRIME_LANGUAGE.PYTHON)
prime_file = validation_job.get_result_when_complete()
if not prime_file.is_valid:
    raise ValueError('File was not valid')
prime_file.download('/home/myuser/drCode/primeModelCode.py')
```

Rating Table

A rating table is an exportable csv representation of a Generalized Additive Model. They contain information about the features and coefficients used to make predictions. Users can influence predictions by downloading and editing values in a rating table, then reuploading the table and using it to create a new model.

See the page about interpreting Generalized Additive Models's output in the Datarobot user guide for more details on how to interpret and edit rating tables.

18.1 Download A Rating Table

You can retrieve a rating table from the list of rating tables in a project:

```
import datarobot as dr
project_id = '5506fcd38bd88f5953219da0'
project = dr.Project.get(project_id)
rating_tables = project.get_rating_tables()
rating_table = rating_tables[0]
```

Or you can retrieve a rating table from a specific model. The model must already exist:

```
import datarobot as dr
from datarobot.models import RatingTableModel, RatingTable
project_id = '5506fcd38bd88f5953219da0'
project = dr.Project.get(project_id)

# Get model from list of models with a rating table
rating_table_models = project.get_rating_table_models()
rating_table_model = rating_table_models[0]

# Or retrieve model by id. The model must have a rating table.
model_id = '5506fcd98bd88f1641a720a3'
rating_table_model = dr.RatingTableModel.get(project=project_id, model_id=model_id)
```

(continues on next page)

(continued from previous page)

```
# Then retrieve the rating table from the model
rating_table_id = rating_table_model.rating_table_id
rating_table = dr.RatingTable.get(projcet_id, rating_table_id)
```

Then you can download the contents of the rating table:

```
rating_table.download('./my_rating_table.csv')
```

18.2 Uploading A Rating Table

After you’ve retrieved the rating table CSV and made the necessary edits, you can re-upload the CSV so you can create a new model from it:

```
job = dr.RatingTable.create(project_id, model_id, './my_rating_table.csv')
new_rating_table = job.get_result_when_complete()
job = new_rating_table.create_model()
model = job.get_result_when_complete()
```

Training Predictions

The training predictions interface allows computing and retrieving out-of-sample predictions for a model using the original project dataset. The predictions can be computed for all the rows, or restricted to validation or holdout data. As the predictions generated will be out-of-sample, they can be expected to have different results than if the project dataset were reuploaded as a prediction dataset.

19.1 Quick reference

Training predictions generation is an asynchronous process. This means that when starting predictions with `datarobot.models.Model.request_training_predictions()` you will receive back a `datarobot.models.TrainingPredictionsJob` for tracking the process responsible for fulfilling your request. Actual predictions may be obtained with the help of a `datarobot.models.training_predictions.TrainingPredictions` object returned as the result of the training predictions job. There are three ways to retrieve them:

1. Iterate prediction rows one by one as named tuples:

```
import datarobot as dr

# Calculate new training predictions on all dataset
training_predictions_job = model.request_training_predictions(dr.enums.DATA_SUBSET.
    ↪ALL)
training_predictions = training_predictions_job.get_result_when_complete()

# Fetch rows from API and print them
for prediction in training_predictions.iterate_rows(batch_size=250):
    print(prediction.row_id, prediction.prediction)
```

2. Get all prediction rows as a `pandas.DataFrame` object:

```
import datarobot from dr

# Calculate new training predictions on holdout partition of dataset
```

(continues on next page)

(continued from previous page)

```
training_predictions_job = model.request_training_predictions(dr.enums.DATA_SUBSET.  
↪HOLDOUT)  
training_predictions = training_predictions_job.get_result_when_complete()  
  
# Fetch training predictions as data frame  
dataframe = training_predictions.get_all_as_dataframe()
```

3. Download all prediction rows to a file as a CSV document:

```
import datarobot from dr  
  
# Calculate new training predictions on all dataset  
training_predictions_job = model.request_training_predictions(dr.enums.DATA_SUBSET.  
↪ALL)  
training_predictions = training_predictions_job.get_result_when_complete()  
  
# Fetch training predictions and save them to file  
training_predictions.download_to_csv('my-training-predictions.csv')
```

Monotonic Constraints

Training with monotonic constraints allows users to force models to learn monotonic relationships with respect to some features and the target. This helps users create accurate models that comply with regulations (e.g. insurance, banking). Currently, only certain blueprints (e.g. xgboost) support this feature, and it is only supported for regression and binary classification projects. Typically working with monotonic constraints follows the following two workflows:

Workflow one - Running a project with default monotonic constraints

- set the target and specify default constraint lists for the project
- when running autopilot or manually training models without overriding constraint settings, all blueprints that support monotonic constraints will use the specified default constraint featurelists

Workflow two - Running a model with specific monotonic constraints

- create featurelists for monotonic constraints
- train a blueprint that supports monotonic constraints while specifying monotonic constraint featurelists
- the specified constraints will be used, regardless of the defaults on the blueprint

20.1 Creating featurelists

When specifying monotonic constraints, users must pass a reference to a featurelist containing only the features to be constrained, one for features that should monotonically increase with the target and another for those that should monotonically decrease with the target.

```
import datarobot as dr
project = dr.Project.get(project_id)
features_mono_up = ['feature_0', 'feature_1'] # features that have monotonically_
↪increasing relationship with target
features_mono_down = ['feature_2', 'feature_3'] # features that have monotonically_
↪decreasing relationship with target
flist_mono_up = project.create_featurelist(name='mono_up',
                                          features=features_mono_up)
```

(continues on next page)

(continued from previous page)

```
flist_mono_down = project.create_featurelist(name='mono_down',
                                             features=features_mono_down)
```

20.2 Specify default monotonic constraints for a project

When setting the target, the user can specify default monotonic constraints for the project, to ensure that autopilot models use the desired settings, and optionally to ensure that only blueprints supporting monotonic constraints appear in the project. Regardless of the defaults specified during target selection, the user can override them when manually training a particular model.

```
import datarobot as dr
from datarobot.enums import AUTOPILOT_MODE
advanced_options = dr.AdvancedOptions(
    monotonic_increasing_featurelist_id=flist_mono_up.id,
    monotonic_decreasing_featurelist_id=flist_mono_down.id,
    only_include_monotonic_blueprints=True)
project = dr.Project.get(project_id)
project.set_target(target='target', mode=AUTOPILOT_MODE.FULL_AUTO, advanced_
    ↪options=advanced_options)
```

20.3 Retrieve models and blueprints using monotonic constraints

When retrieving models, users can inspect to see which supports monotonic constraints, and which actually enforces them. Some models will not support monotonic constraints at all, and some may support constraints but not have any constrained features specified.

```
import datarobot as dr
project = dr.Project.get(project_id)
models = project.get_models()
# retrieve models that support monotonic constraints
models_support_mono = [model for model in models if model.supports_monotonic_
    ↪constraints]
# retrieve models that support and enforce monotonic constraints
models_enforce_mono = [model for model in models
    if (model.monotonic_increasing_featurelist_id or
        model.monotonic_decreasing_featurelist_id)]
```

When retrieving blueprints, users can check if they support monotonic constraints and see what default constraint lists are associated with them. The monotonic featurelist ids associated with a blueprint will be used everytime it is trained, unless the user specifically overrides them at model submission time.

```
import datarobot as dr
project = dr.Project.get(project_id)
blueprints = project.get_blueprints()
# retrieve blueprints that support monotonic constraints
blueprints_support_mono = [blueprint for blueprint in blueprints if blueprint.
    ↪supports_monotonic_constraints]
# retrieve blueprints that support and enforce monotonic constraints
blueprints_enforce_mono = [blueprint for blueprint in blueprints
    if (blueprint.monotonic_increasing_featurelist_id or
        blueprint.monotonic_decreasing_featurelist_id)]
```


20.4 Train a model with specific monotonic constraints

Even after specifying default settings for the project, users can override them to train a new model with different constraints, if desired.

```
import datarobot as dr
features_mono_up = ['feature_2', 'feature_3'] # features that have monotonically_
↪increasing relationship with target
features_mono_down = ['feature_0', 'feature_1'] # features that have monotonically_
↪decreasing relationship with target
project = dr.Project.get(project_id)
flist_mono_up = project.create_featurelist(name='mono_up',
                                          features=features_mono_up)
flist_mono_down = project.create_featurelist(name='mono_down',
                                             features=features_mono_down)
model_job_id = project.train(
    blueprint,
    sample_pct=55,
    featurelist_id=featurelist.id,
    monotonic_increasing_featurelist_id=flist_mono_up.id,
    monotonic_decreasing_featurelist_id=flist_mono_down.id
)
```

Database Connectivity

Databases are a widely used tool for carrying valuable business data. To enable integration with a variety of enterprise databases, DataRobot provides a “self-service” JDBC product for database connectivity setup. Once configured, you can read data from production databases for model building and predictions. This allows you to quickly train and retrain models on that data, and avoids the unnecessary step of exporting data from your enterprise database to a CSV for ingest to DataRobot. It allows access to more diverse data, which results in more accurate models.

The steps describing how to set up your database connections use the following terminology:

- **DataStore**: A configured connection to a database—it has a name, a specified driver, and a JDBC URL. You can register data stores with DataRobot for ease of re-use. A data store has one connector but can have many data sources.
- **DataSource**: A configured connection to the backing data store (the location of data within a given endpoint). A data source specifies, via SQL query or selected table and schema data, which data to extract from the data store to use for modeling or predictions. A data source has one data store and one connector but can have many datasets.
- **Driver**: The software that allows the DataRobot application to interact with a database; each data store is associated with one driver (created the admin). The driver configuration saves the storage location in DataRobot of the JAR file and any additional dependency files associated with the driver.
- **Dataset**: Data, a file or the content of a data source, at a particular point in time. A data source can produce multiple datasets; a dataset has exactly one data source.

The expected workflow when setting up projects or prediction datasets is:

1. The administrator sets up a `datarobot.DataDriver` for accessing a particular database. For any particular driver, this setup is done once for the entire system and then the resulting driver is used by all users.
2. Users create a `datarobot.DataStore` which represents an interface to a particular database, using that driver.
3. Users create a `datarobot.DataSource` representing a particular set of data to be extracted from the DataStore.
4. Users create projects and prediction datasets from a DataSource.

Besides the described workflow for creating projects and prediction datasets, users can manage their DataStores and DataSources and admins can manage Drivers by listing, retrieving, updating and deleting existing instances.

Cloud users: This feature is turned off by default. To enable the feature, contact your CFDS or DataRobot Support.

21.1 Creating Drivers

The admin should specify `class_name`, the name of the Java class in the Java archive which implements the `java.sql.Driver` interface; `canonical_name`, a user-friendly name for resulting driver to display in the API and the GUI; and `files`, a list of local files which contain the driver.

```
>>> import datarobot as dr
>>> driver = dr.DataDriver.create(
...     class_name='org.postgresql.Driver',
...     canonical_name='PostgreSQL',
...     files=['/tmp/postgresql-42.2.2.jar']
... )
>>> driver
DataDriver('PostgreSQL')
```

21.2 Creating DataStores

After the admin has created drivers, any user can use them for `DataStore` creation. A `DataStore` represents a JDBC database. When creating them, users should specify `type`, which currently must be `jdbc`; `canonical_name`, a user-friendly name to display in the API and GUI for the `DataStore`; `driver_id`, the id of the driver to use to connect to the database; and `jdbc_url`, the full URL specifying the database connection settings like database type, server address, port, and database name.

```
>>> import datarobot as dr
>>> data_store = dr.DataStore.create(
...     data_store_type='jdbc',
...     canonical_name='Demo DB',
...     driver_id='5a6af02eb15372000117c040',
...     jdbc_url='jdbc:postgresql://my.db.address.org:5432/perftest'
... )
>>> data_store
DataStore('Demo DB')
>>> data_store.test(username='username', password='password')
{'message': 'Connection successful'}
```

21.3 Creating DataSources

Once users have a `DataStore`, they can query datasets via the `DataSource` entity, which represents a query. When creating a `DataSource`, users first create a `datarobot.DataSourceParameters` object from a `DataStore`'s id and a query, and then create the `DataSource` with a `type`, currently always `jdbc`; a `canonical_name`, the user-friendly name to display in the API and GUI, and `params`, the `DataSourceParameters` object.

```
>>> import datarobot as dr
>>> params = dr.DataSourceParameters(
...     data_store_id='5a8ac90b07a57a0001be501e',
```

(continues on next page)

(continued from previous page)

```
...     query='SELECT * FROM airlines10mb WHERE "Year" >= 1995;'
... )
>>> data_source = dr.DataSource.create(
...     data_source_type='jdbc',
...     canonical_name='airlines stats after 1995',
...     params=params
... )
>>> data_source
DataSource('airlines stats after 1995')
```

21.4 Creating Projects

Given a DataSource, users can create new projects from it.

```
>>> import datarobot as dr
>>> project = dr.Project.create_from_data_source(
...     data_source_id='5ae6eee9962d740dd7b86886',
...     username='username',
...     password='password'
... )
```

21.5 Creating Predictions

Given a DataSource, new prediction datasets can be created for any project.

```
>>> import datarobot as dr
>>> project = dr.Project.get('5ae6f296962d740dd7b86887')
>>> prediction_dataset = project.upload_dataset_from_data_source(
...     data_source_id='5ae6eee9962d740dd7b86886',
...     username='username',
...     password='password'
... )
```

Model Recommendation

During the Autopilot modeling process, DataRobot will recommend up to three well-performing models.

Warning: Model recommendations are only generated when you run full Autopilot.

One of them (the most accurate individual, non-blender model) will be prepared for deployment. In the preparation process, DataRobot:

1. Calculates **feature impact** for the selected model and uses it to generate a **reduced feature list**.
2. Retrains the selected model on **the reduced feature list**. If the new model performs better than the original model, DataRobot uses the new model for the next stage. Otherwise, the original model is used.
3. Retrains the selected model on a **higher sample size**. If the new model performs better than the original model, DataRobot selects it as **Recommended for Deployment**. Otherwise, the original model is selected.

Note: The higher sample size DataRobot uses in Step 3 is either:

1. **Up to holdout** if the training sample size *does not* exceed the maximum Autopilot size threshold: sample size is the training set plus the validation set (for TVH) or 5-folds (for CV). In this case, DataRobot compares retrained and original models on the holdout score.
 2. **Up to validation** if the training sample size *does* exceed the maximum Autopilot size threshold: sample size is the training set (for TVH) or 4-folds (for CV). In this case, DataRobot compares retrained and original models on the validation score.
-

The three types of recommendations are the following:

- **Recommended for Deployment.** This is the most accurate individual, non-blender model on the Leaderboard. This model is ready for deployment.
- **Most Accurate.** Based on the validation or cross-validation results, this model is the most accurate model overall on the Leaderboard (in most cases, a blender).

- **Fast & Accurate.** This is the most accurate individual model on the Leaderboard that passes a set prediction speed guidelines. If no models meet the guideline, the badge is not applied.

22.1 Retrieve all recommendations

The following code will return all models recommended for the project.

```
import datarobot as dr

recommendations = dr.ModelRecommendation.get_all(project_id)
```

22.2 Retrieve a default recommendation

If you are unsure about the tradeoffs between the various types of recommendations, DataRobot can make this choice for you. The following route will return the Recommended for Deployment model to use for predictions for the project.

```
import datarobot as dr

recommendation = dr.ModelRecommendation.get(project_id)
```

22.3 Retrieve a specific recommendation

If you know which recommendation you want to use, you can select a specific recommendation using the following code.

```
import datarobot as dr

recommendation_type = dr.enums.RECOMMENDED_MODEL_TYPE.FAST_ACCURATE
recommendations = dr.ModelRecommendation.get(project_id, recommendation_type)
```

22.4 Get recommended model

You can use method `get_model()` of a recommendation object to retrieve a recommended model.

```
import datarobot as dr

recommendation = dr.ModelRecommendation.get(project_id)
recommended_model = recommendation.get_model()
```


Once you have created data stores or data sources, you may want to share them with collaborators. DataRobot provides an API for sharing the following entities:

- Data Sources and Data Stores (see *Database Connectivity* for more info on connecting to JDBC databases)
- Projects
- Calendar Files
- Model Deployments (Only in the REST API, not yet in this Python client)

23.1 Access Levels

Entities can be shared at varying access levels. For example, you can allow someone to create projects from a data source you have built without letting them delete it.

Each entity type uses slightly different permission names intended to convey more specifically what kind of actions are available, and these roles fall into three categories. These generic role names can be used in the sharing API for any entity.

For the complete set of actions granted by each role on a given entity, please see the user documentation in the web application.

- OWNER
 - used for all entities
 - allows any action including deletion
- READ_WRITE
 - known as as EDITOR on data sources and data stores
 - allows modifications to the state, e.g. renaming and creating data sources from a data store, but *not* deleting the entity
- READ_ONLY

- known as CONSUMER on data sources and data stores
- for data sources, enables creating projects and predictions; for data stores, allows viewing them only.

Finally, when a user's new role is specified as `None`, their access will be revoked.

In addition to the role, some entities (currently only data sources and data stores) allow separate control over whether a new user should be able to share that entity further. When granting access to a user, the `can_share` parameter determines whether that user can, in turn, share this entity with another user. When this parameter is specified as `false`, the user in question will have all the access to the entity granted by their role and be able to remove themselves if desired, but be unable to change the role of any other user.

23.2 Examples

Transfer access to the data source from `old_user@datarobot.com` to `new_user@datarobot.com`

```
import datarobot as dr

new_access = dr.SharingAccess(new_user@datarobot.com,
                              dr.enums.SHARING_ROLE.OWNER, can_share=True)
access_list = [dr.SharingAccess(old_user@datarobot.com, None), new_access]

dr.DataSource.get('my-data-source-id').share(access_list)
```

Checking access to a project

```
import datarobot as dr

project = dr.Project.create('mydata.csv', project_name='My Data')

access_list = project.get_access_list()

access_list[0].username
```

Transfer ownership of all projects owned by your account to `new_user@datarobot.com` without sending notifications.

```
import datarobot as dr

# Put path to YAML credentials below
dr.Client(config_path= '.yaml')

# Get all projects for your account and store the ids in a list
projects = dr.Project.list()

project_ids = [project.id for project in projects]

# List of emails to share with
share_targets = ['new_user@datarobot.com']

# Target role
target_role = dr.enums.SHARING_ROLE.OWNER

for pid in project_ids:

    project = dr.Project.get(project_id=pid)
```

(continues on next page)

(continued from previous page)

```
shares = []  
  
for user in share_targets:  
    shares.append(dr.SharingAccess(username=user, role=target_role))  
  
project.share(shares, send_notification=False)
```


Deployment is the central hub for users to deploy, manage and monitor their models.

24.1 Manage Deployments

The following commands can be used to manage deployments.

24.1.1 Create a Deployment

A new deployment can be created from:

- DataRobot model - use `create_from_learning_model()`
- Custom model version with dependency management - use `create_from_custom_model_version()`. Please refer to [Custom Model documentation](#) on how to create a custom model version
- Custom model image - use `create_from_custom_model_image()`. Please refer to [Custom Inference Image documentation](#) on how to create a custom model image

When creating a new deployment, a DataRobot `model_id`/`custom_model_image_id` and `label` must be provided. A `description` can be optionally provided to document the purpose of the deployment.

The default prediction server is used when making predictions against the deployment, and is a requirement for creating a deployment on DataRobot cloud. For on-prem installations, a user must not provide a default prediction server and a pre-configured prediction server will be used instead. Refer to [`datarobot.PredictionServer.list`](#) for more information on retrieving available prediction servers.

```
import datarobot as dr

project = dr.Project.get('5506fcd38bd88f5953219da0')
model = project.get_models()[0]
prediction_server = dr.PredictionServer.list()[0]
```

(continues on next page)

(continued from previous page)

```
deployment = dr.Deployment.create_from_learning_model(
    model.id, label='New Deployment', description='A new deployment',
    default_prediction_server_id=prediction_server.id)
deployment
>>> Deployment('New Deployment')
```

24.1.2 List Deployments

Use the following command to list deployments a user can view.

```
import datarobot as dr

deployments = dr.Deployment.list()
deployments
>>> [Deployment('New Deployment'), Deployment('Previous Deployment')]
```

Refer to *Deployment* for properties of the deployment object.

You can also filter the deployments that are returned by passing an instance of the *DeploymentListFilters* class to the *filters* keyword argument.

```
import datarobot as dr

filters = dr.models.deployment.DeploymentListFilters(
    role='OWNER',
    accuracy_health=dr.enums.DEPLOYMENT_ACCURACY_HEALTH_STATUS.FAILING
)
deployments = dr.Deployment.list(filters=filters)
deployments
>>> [Deployment('Deployment Owned by Me w/ Failing Accuracy 1'), Deployment(
    ↪ 'Deployment Owned by Me w/ Failing Accuracy 2')]
```

24.1.3 Retrieve a Deployment

It is possible to retrieve a single deployment with its identifier, rather than list all deployments.

```
import datarobot as dr

deployment = dr.Deployment.get(deployment_id='5c939e08962d741e34f609f0')
deployment.id
>>> '5c939e08962d741e34f609f0'
deployment.label
>>> 'New Deployment'
```

Refer to *Deployment* for properties of the deployment object.

24.1.4 Update a Deployment

Deployment's label and description can be updated.

```
import datarobot as dr

deployment = dr.Deployment.get(deployment_id='5c939e08962d741e34f609f0')
deployment.update(label='new label')
```

24.1.5 Delete a Deployment

To mark a deployment as deleted, use the following command.

```
import datarobot as dr

deployment = dr.Deployment.get(deployment_id='5c939e08962d741e34f609f0')
deployment.delete()
```

24.2 Model Replacement

The model of a deployment can be replaced effortlessly with zero interruption of predictions.

Model replacement is an asynchronous process, which means there are some preparatory works to complete before the process is fully finished. However, predictions made against this deployment will start using the new model as soon as you initiate the process. The `replace_model()` function won't return until this asynchronous process is fully finished.

Alongside the identifier of the new model, a `reason` is also required. The reason is stored in model history of the deployment for bookkeeping purpose. An enum `MODEL_REPLACEMENT_REASON` is provided for convenience, all possible values are documented below:

- `MODEL_REPLACEMENT_REASON.ACCURACY`
- `MODEL_REPLACEMENT_REASON.DATA_DRIFT`
- `MODEL_REPLACEMENT_REASON.ERRORS`
- `MODEL_REPLACEMENT_REASON.SCHEDULED_REFRESH`
- `MODEL_REPLACEMENT_REASON.SCORING_SPEED`
- `MODEL_REPLACEMENT_REASON.OTHER`

Here is an example of model replacement:

```
import datarobot as dr
from datarobot.enums import MODEL_REPLACEMENT_REASON

project = dr.Project.get('5cc899abc191a20104ff446a')
model = project.get_models()[0]

deployment = Deployment.get(deployment_id='5c939e08962d741e34f609f0')
deployment.model['id'], deployment.model['type']
>>> ('5c0a979859b00004ba52e431', 'Decision Tree Classifier (Gini)')

deployment.replace_model('5c0a969859b00004ba52e41b', MODEL_REPLACEMENT_REASON.
↳ ACCURACY)
deployment.model['id'], deployment.model['type']
>>> ('5c0a969859b00004ba52e41b', 'Support Vector Classifier (Linear Kernel)')
```

24.2.1 Validation

Before initiating the model replacement request, it is usually a good idea to use the `validate_replacement_model()` function to validate if the new model can be used as a replacement.

The `validate_replacement_model()` function returns the validation status, a message and a checks dictionary. If the status is `passing` or `warning`, use `replace_model()` to perform the replacement. If status is `failing`, refer to the `checks` dict for more details on why the new model cannot be used as a replacement.

```
import datarobot as dr

project = dr.Project.get('5cc899abc191a20104ff446a')
model = project.get_models()[0]
deployment = dr.Deployment.get(deployment_id='5c939e08962d741e34f609f0')
status, message, checks = deployment.validate_replacement_model(new_model_id=model.id)
status
>>> 'passing'

# `checks` can be inspected for detail, showing two examples here:
checks['target']
>>> {'status': 'passing', 'message': 'Target is compatible.'}
checks['permission']
>>> {'status': 'passing', 'message': 'User has permission to replace model.'}
```

24.3 Monitoring

Deployment monitoring can be categorized into several area of concerns:

- Service Stats & Service Stats Over Time
- Accuracy & Accuracy Over Time

With a `Deployment` object, get functions are provided to allow querying of the monitoring data. Alternatively, it is also possible to retrieve monitoring data directly using a deployment ID. For example:

```
from datarobot.models import Deployment, ServiceStats

deployment_id = '5c939e08962d741e34f609f0'

# call `get` functions on a `Deployment` object
deployment = Deployment.get(deployment_id)
service_stats = deployment.get_service_stats()

# directly fetch without a `Deployment` object
service_stats = ServiceStats.get(deployment_id)
```

When querying monitoring data, a start and end time can be optionally provided, will accept either a datetime object or a string. Note that only top of the hour datetimes are accepted, for example: 2019-08-01T00:00:00Z. By default, the end time of the query will be the next top of the hour, the start time will be 7 days before the end time.

In the over time variants, an optional `bucket_size` can be provided to specify the resolution of time buckets. For example, if start time is 2019-08-01T00:00:00Z, end time is 2019-08-02T00:00:00Z and `bucket_size` is T1H, then 24 time buckets will be generated, each providing data calculated over one hour. Use `construct_duration_string()` to help construct a bucket size string.

Note: The minimum bucket size is one hour.

24.3.1 Service Stats

Service stats are metrics tracking deployment utilization and how well deployments respond to prediction requests. Use `SERVICE_STAT_METRIC.ALL` to retrieve a list of supported metrics.

`ServiceStats` retrieves values for all service stats metrics; `ServiceStatsOverTime` can be used to fetch how one single metric changes over time.

```
from datetime import datetime
from datarobot.enums import SERVICE_STAT_METRIC
from datarobot.helpers.partitioning_methods import construct_duration_string
from datarobot.models import Deployment

deployment = Deployment.get(deployment_id='5c939e08962d741e34f609f0')
service_stats = deployment.get_service_stats(
    start_time=datetime(2019, 8, 1, hour=15),
    end_time=datetime(2019, 8, 8, hour=15)
)
service_stats[SERVICE_STAT_METRIC.TOTAL_PREDICTIONS]
>>> 12597

total_predictions = deployment.get_service_stats_over_time(
    start_time=datetime(2019, 8, 1, hour=15),
    end_time=datetime(2019, 8, 8, hour=15),
    bucket_size=construct_duration_string(days=1),
    metric=SERVICE_STAT_METRIC.TOTAL_PREDICTIONS
)
total_predictions.bucket_values
>>> OrderedDict([(datetime.datetime(2019, 8, 1, 15, 0, tzinfo=tzutc()), 1610),
                  (datetime.datetime(2019, 8, 2, 15, 0, tzinfo=tzutc()), 2249),
                  (datetime.datetime(2019, 8, 3, 15, 0, tzinfo=tzutc()), 254),
                  (datetime.datetime(2019, 8, 4, 15, 0, tzinfo=tzutc()), 943),
                  (datetime.datetime(2019, 8, 5, 15, 0, tzinfo=tzutc()), 1967),
                  (datetime.datetime(2019, 8, 6, 15, 0, tzinfo=tzutc()), 2810),
                  (datetime.datetime(2019, 8, 7, 15, 0, tzinfo=tzutc()), 2775)])
```

24.3.2 Data Drift

Data drift describe how much the distribution of target or a feature has changed comparing to the training data. Deployment's target drift and feature drift can be retrieved separately using `datarobot.models.TargetDrift` and `datarobot.models.FeatureDrift`. Use `DATA_DRIFT_METRIC.ALL` to retrieve a list of supported metrics.

```
from datetime import datetime
from datarobot.enums import DATA_DRIFT_METRIC
from datarobot.models import Deployment, FeatureDrift

deployment = Deployment.get(deployment_id='5c939e08962d741e34f609f0')
target_drift = deployment.get_target_drift(
    start_time=datetime(2019, 8, 1, hour=15),
    end_time=datetime(2019, 8, 8, hour=15)
```

(continues on next page)

(continued from previous page)

```

)
target_drift.drift_score
>>> 0.00408514

feature_drift_data = FeatureDrift.list(
    deployment_id='5c939e08962d741e34f609f0',
    start_time=datetime(2019, 8, 1, hour=15),
    end_time=datetime(2019, 8, 8, hour=15),
    metric=DATA_DRIFT_METRIC.HELLINGER
)
feature_drift = feature_drift_data[0]
feature_drift.name
>>> 'age'
feature_drift.drift_score
>>> 4.16981594

```

24.3.3 Accuracy

A collection of metrics are provided to measure the accuracy of a deployment's predictions. For deployments with classification model, use `ACCURACY_METRIC.ALL_CLASSIFICATION` for all supported metrics; in the case of deployment with regression model, use `ACCURACY_METRIC.ALL_REGRESSION` instead.

Similarly with Service Stats, [Accuracy](#) and [AccuracyOverTime](#) are provided to retrieve all default accuracy metrics and how one single metric change over time.

```

from datetime import datetime
from datarobot.enums import ACCURACY_METRIC
from datarobot.helpers.partitioning_methods import construct_duration_string
from datarobot.models import Deployment

deployment = Deployment.get(deployment_id='5c939e08962d741e34f609f0')
accuracy = deployment.get_accuracy(
    start_time=datetime(2019, 8, 1, hour=15),
    end_time=datetime(2019, 8, 1, 15, 0)
)
accuracy[ACCURACY_METRIC.RMSE]
>>> 943.225

rmse = deployment.get_accuracy_over_time(
    start_time=datetime(2019, 8, 1),
    end_time=datetime(2019, 8, 3),
    bucket_size=construct_duration_string(days=1),
    metric=ACCURACY_METRIC.RMSE
)
rmse.bucket_values
>>> OrderedDict([(datetime.datetime(2019, 8, 1, 15, 0, tzinfo=tzutc()), 1777.190657),
                  (datetime.datetime(2019, 8, 2, 15, 0, tzinfo=tzutc()), 1613.140772)])

```

It is also possible to retrieve how multiple metrics changes over the same period of time, enabling easier side by side comparison across different metrics.

```

from datarobot.enums import ACCURACY_METRIC
from datarobot.models import Deployment

```

(continues on next page)

(continued from previous page)

```
accuracy_over_time = AccuracyOverTime.get_as_dataframe(
    ram_app.id, [ACCURACY_METRIC.RMSE, ACCURACY_METRIC.GAMMA_DEVIANCE, ACCURACY_
    ↪METRIC.MAD])
```

24.4 Settings

24.4.1 Drift Tracking Settings

Drift tracking is used to help analyze and monitor the performance of a model after it is deployed. When the model of a deployment is replaced drift tracking status will not be altered.

Use `get_drift_tracking_settings()` to retrieve the current tracking status for target drift and feature drift.

```
import datarobot as dr

deployment = dr.Deployment.get(deployment_id='5c939e08962d741e34f609f0')
settings = deployment.get_drift_tracking_settings()
settings
>>> {'target_drift': {'enabled': True}, 'feature_drift': {'enabled': True}}
```

Use `update_drift_tracking_settings()` to update target drift and feature drift tracking status.

```
import datarobot as dr

deployment = dr.Deployment.get(deployment_id='5c939e08962d741e34f609f0')
deployment.update_drift_tracking_settings(target_drift_enabled=True, feature_drift_
    ↪enabled=True)
```

24.4.2 Association ID Settings

Association ID is used to identify predictions, so that when actuals are acquired, accuracy can be calculated.

Use `get_association_id_settings()` to retrieve current association ID settings.

```
import datarobot as dr

deployment = dr.Deployment.get(deployment_id='5c939e08962d741e34f609f0')
settings = deployment.get_association_id_settings()
settings
>>> {'column_names': ['application_id'], 'required_in_prediction_requests': True}
```

Use `update_association_id_settings()` to update association ID settings.

```
import datarobot as dr

deployment = dr.Deployment.get(deployment_id='5c939e08962d741e34f609f0')
deployment.update_association_id_settings(column_names=['application_id'], required_
    ↪in_prediction_requests=True)
```

24.4.3 Predictions Data Collection Settings

Predictions Data Collection configures whether prediction requests and results should be saved to Predictions Data Storage.

Use `get_predictions_data_collection_settings()` to retrieve current settings of predictions data collection.

```
import datarobot as dr

deployment = dr.Deployment.get(deployment_id='5c939e08962d741e34f609f0')
settings = deployment.get_predictions_data_collection_settings()
settings
>>> {'enabled': True}
```

Use `update_predictions_data_collection_settings()` to update predictions data collection settings.

```
import datarobot as dr

deployment = dr.Deployment.get(deployment_id='5c939e08962d741e34f609f0')
deployment.update_predictions_data_collection_settings(enabled=True)
```

24.4.4 Prediction Warning Settings

Prediction Warning is used to enable Humble AI for a deployment which determines if a model is misbehaving when a prediction goes outside of the calculated boundaries.

Use `get_prediction_warning_settings()` to retrieve the current prediction warning settings.

```
import datarobot as dr

deployment = dr.Deployment.get(deployment_id='5c939e08962d741e34f609f0')
settings = deployment.get_prediction_warning_settings()
settings
>>> {'enabled': True, 'custom_boundaries': {'upper': 1337, 'lower': 0}}
```

Use `update_prediction_warning_settings()` to update current prediction warning settings.

```
import datarobot as dr

# Set custom boundaries
deployment = dr.Deployment.get(deployment_id='5c939e08962d741e34f609f0')
deployment.update_prediction_warning_settings(
    prediction_warning_enabled=True,
    use_default_boundaries=False,
    lower_boundary=1337,
    upper_boundary=2000,
)

# Reset boundaries
deployment.update_prediction_warning_settings(
    prediction_warning_enabled=True,
    use_default_boundaries=True,
)
```

Custom models provide users the ability to run arbitrary modeling code in an environment defined by the user.

25.1 Manage Execution Environments

Execution Environment defines the runtime environment for custom models. Execution Environment Version is a revision of Execution Environment with an actual runtime definition. Please refer to DataRobot User Models (<https://github.com/datarobot/datarobot-user-models>) for sample environments.

25.1.1 Create Execution Environment

To create an Execution Environment run:

```
import datarobot as dr

execution_environment = dr.ExecutionEnvironment.create(
    name="Python3 PyTorch Environment",
    description="This environment contains Python3 pytorch library.",
)

execution_environment.id
>>> '5b6b2315ca36c0108fc5d41b'
```

There are 2 ways to create an Execution Environment Version: synchronous and asynchronous.

Synchronous way means that program execution will be blocked until an Execution Environment Version creation process is finished with either success or failure:

```
import datarobot as dr

# use execution_environment created earlier
```

(continues on next page)

(continued from previous page)

```

environment_version = dr.ExecutionEnvironmentVersion.create(
    execution_environment.id,
    docker_context_path="datarobot-user-models/public_dropin_environments/python3_
↳pytorch",
    max_wait=3600, # 1 hour timeout
)

environment_version.id
>>> '5eb538959bc057003b487b2d'
environment_version.build_status
>>> 'success'

```

Asynchronous way means that program execution will be not blocked, but an Execution Environment Version created will not be ready to be used for some time, until its creation process is finished. In such case, it will be required to manually call `refresh()` for the Execution Environment Version and check if its `build_status` is `success`. To create an Execution Environment Version without blocking a program, set `max_wait` to `None`:

```

import datarobot as dr

# use execution_environment created earlier

environment_version = dr.ExecutionEnvironmentVersion.create(
    execution_environment.id,
    docker_context_path="datarobot-user-models/public_dropin_environments/python3_
↳pytorch",
    max_wait=None, # set None to not block execution on this method
)

environment_version.id
>>> '5eb538959bc057003b487b2d'
environment_version.build_status
>>> 'processing'

# after some time
environment_version.refresh()
environment_version.build_status
>>> 'success'

```

25.1.2 List Execution Environments

Use the following command to list execution environments available to the user.

```

import datarobot as dr

execution_environments = dr.ExecutionEnvironment.list()
execution_environments
>>> [ExecutionEnvironment('[DataRobot] Python 3 PyTorch Drop-In'),
↳ExecutionEnvironment('[DataRobot] Java Drop-In')]

environment_versions = dr.ExecutionEnvironmentVersion.list(execution_environment.id)
environment_versions
>>> [ExecutionEnvironmentVersion('v1')]

```

Refer to `ExecutionEnvironment` for properties of the execution environment object and `ExecutionEnvironmentVersion` for properties of the execution environment object version.

You can also filter the execution environments that are returned by passing a string as *search_for* parameter - only the execution environments that contain the passed string in name or description will be returned.

```
import datarobot as dr

execution_environments = dr.ExecutionEnvironment.list(search_for='java')
execution_environments
>>> [ExecutionEnvironment('[DataRobot] Java Drop-In')]
```

Execution environment versions can be filtered by build status.

```
import datarobot as dr

environment_versions = dr.ExecutionEnvironmentVersion.list(
    execution_environment_id, dr.EXECUTION_ENVIRONMENT_VERSION_BUILD_STATUS.PROCESSING
)
environment_versions
>>> [ExecutionEnvironmentVersion('v1')]
```

25.1.3 Retrieve Execution Environment

To retrieve an execution environment and an execution environment version by identifier, rather than list all available ones, do the following:

```
import datarobot as dr

execution_environment = dr.ExecutionEnvironment.get(execution_environment_id=
↳ '5506fcd38bd88f5953219da0')
execution_environment
>>> ExecutionEnvironment('[DataRobot] Python 3 PyTorch Drop-In')

environment_version = dr.ExecutionEnvironmentVersion.get(
    execution_environment_id=execution_environment.id, version_id=
↳ '5eb538959bc057003b487b2d')
environment_version
>>> ExecutionEnvironmentVersion('v1')
```

25.1.4 Update Execution Environment

To update name and/or description of the execution environment run:

```
import datarobot as dr

execution_environment = dr.ExecutionEnvironment.get(execution_environment_id=
↳ '5506fcd38bd88f5953219da0')
execution_environment.update(name='new name', description='new description')
```

25.1.5 Delete Execution Environment

To delete the execution environment and execution environment version, use the following commands.

```
import datarobot as dr

execution_environment = dr.ExecutionEnvironment.get(execution_environment_id=
↳ '5506fcd38bd88f5953219da0')
execution_environment.delete()
```

25.1.6 Get Execution Environment build log

To get execution environment version build log run:

```
import datarobot as dr

environment_version = dr.ExecutionEnvironmentVersion.get(
    execution_environment_id='5506fcd38bd88f5953219da0', version_id=
↳ '5eb538959bc057003b487b2d')
log, error = environment_version.get_build_log()
```

25.2 Manage Custom Models

Custom Inference Model is user-defined modeling code that supports making predictions against it. Custom Inference Model supports regression and binary classification target types.

To upload actual modeling code Custom Model Version must be created for a custom model. Please see [Custom Model Version documentation](#).

25.2.1 Create Custom Inference Model

To create a regression Custom Inference Model run:

```
import datarobot as dr

custom_model = dr.CustomInferenceModel.create(
    name='Python 3 PyTorch Custom Model',
    target_type=dr.TARGET_TYPE.REGRESSION,
    target_name='MEDV',
    description='This is a Python3-based custom model. It has a simple PyTorch model_
↳ built on boston housing',
    language='python'
)

custom_model.id
>>> '5b6b2315ca36c0108fc5d41b'
```

When creating a binary classification Custom Inference Model, *positive_class_label* and *negative_class_label* must be set:

```
import datarobot as dr

custom_model = dr.CustomInferenceModel.create(
    name='Python 3 PyTorch Custom Model',
    target_type=dr.TARGET_TYPE.BINARY,
    target_name='readmitted',
```

(continues on next page)

(continued from previous page)

```

    positive_class_label='False',
    negative_class_label='True',
    description='This is a Python3-based custom model. It has a simple PyTorch model_
↳built on 10k_diabetes dataset',
    language='Python 3'
)

custom_model.id
>>> '5b6b2315ca36c0108fc5d41b'

```

25.2.2 List Custom Inference Models

Use the following command to list Custom Inference Models available to the user:

```

import datarobot as dr

dr.CustomInferenceModel.list()
>>> [CustomInferenceModel('my model 2'), CustomInferenceModel('my model 1')]

# use these parameters to filter results:
dr.CustomInferenceModel.list(
    is_deployed=True, # set to return only deployed models
    order_by='-updated', # set to define order of returned results
    search_for='model 1', # return only models containing 'model 1' in name or_
↳description
)
>>> CustomInferenceModel('my model 1')

```

Please refer to `list()` for detailed parameter description.

25.2.3 Retrieve Custom Inference Model

To retrieve a specific Custom Inference Model, run:

```

import datarobot as dr

dr.CustomInferenceModel.get('5ebe95044024035cc6a65602')
>>> CustomInferenceModel('my model 1')

```

25.2.4 Update Custom Model

To update Custom Inference Model properties execute the following:

```

import datarobot as dr

custom_model = dr.CustomInferenceModel.get('5ebe95044024035cc6a65602')

custom_model.update(
    name='new name',
    description='new description',
)

```

Please, refer to `update()` for the full list of properties that can be updated.

25.2.5 Download latest revision of Custom Inference Model

To download content of the latest Custom Model Version of *CustomInferenceModel* as a ZIP archive:

```
import datarobot as dr

path_to_download = '/home/user/Documents/myModel.zip'

custom_model = dr.CustomInferenceModel.get('5ebe96b84024035cc6a6560b')

custom_model.download_latest_version(path_to_download)
```

25.2.6 Assign training data to Custom Inference Model

To assign training data to Custom Inference Model, run:

```
import datarobot as dr

path_to_dataset = '/home/user/Documents/trainingDataset.csv'
dataset = dr.Dataset.create_from_file(file_path=path_to_dataset)

custom_model = dr.CustomInferenceModel.get('5ebe96b84024035cc6a6560b')

custom_model.assign_training_data(dataset.id)
```

To assign training data without blocking a program, set *max_wait* to *None*:

```
import datarobot as dr

path_to_dataset = '/home/user/Documents/trainingDataset.csv'
dataset = dr.Dataset.create_from_file(file_path=path_to_dataset)

custom_model = dr.CustomInferenceModel.get('5ebe96b84024035cc6a6560b')

custom_model.assign_training_data(
    dataset.id,
    max_wait=None
)

custom_model.training_data_assignment_in_progress
>>> True

# after some time
custom_model.refresh()
custom_model.training_data_assignment_in_progress
>>> False
```

Note: training data must be assigned to retrieve feature impact from Custom Inference Image. Please see to [Custom Inference Image documentation](#).

25.3 Manage Custom Model Versions

Modeling code for Custom Inference Models can be uploaded by creating a Custom Model Version. When creating a Custom Model Version, the version must be associated with a base execution environment. If the base environment

supports additional model dependencies (R or Python environments) and the Custom Model Version contains a valid `requirements.txt` file, the model version will run in an environment based on the base environment with the additional dependencies installed.

25.3.1 Create Custom Model Version

Upload actual custom model content by creating a clean Custom Model Version:

```
import os
import datarobot as dr

custom_model_folder = "datarobot-user-models/model_templates/python3_pytorch"

# add files from the folder to the custom model
model_version = dr.CustomModelVersion.create_clean(
    custom_model_id=custom_model.id,
    base_environment_id=execution_environment.id,
    folder_path=custom_model_folder,
)

custom_model.id
>>> '5b6b2315ca36c0108fc5d41b'

# or add a list of files to the custom model
model_version_2 = dr.CustomModelVersion.create_clean(
    custom_model_id=custom_model.id,
    files=[(os.path.join(custom_model_folder, 'custom.py'), 'custom.py')],
)
```

To create a new Custom Model Version from a previous one, with just some files added or removed, do the following:

```
import os
import datarobot as dr

custom_model_folder = "datarobot-user-models/model_templates/python3_pytorch"

file_to_delete = model_version_2.items[0].id

model_version_3 = dr.CustomModelVersion.create_from_previous(
    custom_model_id=custom_model.id,
    base_environment_id=execution_environment.id,
    files=[(os.path.join(custom_model_folder, 'custom.py'), 'custom.py')],
    files_to_delete=[file_to_delete],
)
```

Please refer to [CustomModelFileItem](#) for description of custom model file properties.

25.3.2 List Custom Model Versions

Use the following command to list Custom Model Versions available to the user:

```
import datarobot as dr

dr.CustomModelVersion.list(custom_model.id)

>>> [CustomModelVersion('v2.0'), CustomModelVersion('v1.0')]
```

25.3.3 Retrieve Custom Model Version

To retrieve a specific Custom Model Version, run:

```
import datarobot as dr

dr.CustomModelVersion.get(custom_model.id, custom_model_version_id=
↳ '5ebe96b84024035cc6a6560b')

>>> CustomModelVersion('v2.0')
```

25.3.4 Update Custom Model Version

To update Custom Model Version description execute the following:

```
import datarobot as dr

custom_model_version = dr.CustomModelVersion.get(
    custom_model.id,
    custom_model_version_id='5ebe96b84024035cc6a6560b',
)

custom_model_version.update(description='new description')

custom_model_version.description
>>> 'new description'
```

25.3.5 Download Custom Model Version

Download content of the Custom Model Version as a ZIP archive:

```
import datarobot as dr

path_to_download = '/home/user/Documents/myModel.zip'

custom_model_version = dr.CustomModelVersion.get(
    custom_model.id,
    custom_model_version_id='5ebe96b84024035cc6a6560b',
)

custom_model_version.download(path_to_download)
```

25.4 Preparing a Custom Model Version for Use

If your custom model version has dependencies, a dependency build must be completed before the model can be used. The dependency build installs your model's dependencies into the base environment associated with the model version.

25.4.1 Starting the Dependency Build

To start the Custom Model Version Dependency Build, run:

```
import datarobot as dr

build_info = dr.CustomModelVersionDependencyBuild.start_build(
    custom_model_id=custom_model.id,
    custom_model_version_id=model_version.id,
    max_wait=3600, # 1 hour timeout
)

build_info.build_status
>>> 'success'
```

To start Custom Model Version Dependency Build without blocking a program until the test finishes, set *max_wait* to *None*:

```
import datarobot as dr

build_info = dr.CustomModelVersionDependencyBuild.start_build(
    custom_model_id=custom_model.id,
    custom_model_version_id=model_version.id,
    max_wait=None,
)

build_info.build_status
>>> 'submitted'

# after some time
build_info.refresh()
build_info.build_status
>>> 'success'
```

In case the build fails, or you are just curious, do the following to retrieve the build log once complete:

```
print(build_info.get_log())
```

To cancel a Custom Model Version Dependency Build, simply run:

```
build_info.cancel()
```

25.5 Manage Custom Model Tests

A Custom Model Test represents testing performed on custom models.

25.5.1 Create Custom Model Test

To create Custom Model Test, run:

```
import datarobot as dr

path_to_dataset = '/home/user/Documents/testDataset.csv'
dataset = dr.Dataset.create_from_file(file_path=path_to_dataset)

custom_model_test = dr.CustomModelTest.create(
    custom_model_id=custom_model.id,
```

(continues on next page)

(continued from previous page)

```

        custom_model_version_id=model_version.id,
        dataset_id=dataset.id,
        max_wait=3600, # 1 hour timeout
    )

custom_model_test.overall_status
>>> 'succeeded'

```

To start Custom Model Test without blocking a program until the test finishes, set *max_wait* to *None*:

```

import datarobot as dr

path_to_dataset = '/home/user/Documents/testDataset.csv'
dataset = dr.Dataset.create_from_file(file_path=path_to_dataset)

custom_model_test = dr.CustomModelTest.create(
    custom_model_id=custom_model.id,
    custom_model_version_id=model_version.id,
    environment_id=execution_environment.id,
    environment_version_id=environment_version.id,
    dataset_id=dataset.id,
    max_wait=None,
)

custom_model_test.overall_status
>>> 'in_progress'

# after some time
custom_model_test.refresh()
custom_model_test.overall_status
>>> 'succeeded'

```

Running a Custom Model Test uses the Custom Model Version's base image with its dependencies installed as an execution environment. To start Custom Model Test using an execution environment *as-is*, without the model's dependencies installed, supply an environment ID and (optionally) and environment version ID:

```

import datarobot as dr

path_to_dataset = '/home/user/Documents/testDataset.csv'
dataset = dr.Dataset.create_from_file(file_path=path_to_dataset)

custom_model_test = dr.CustomModelTest.create(
    custom_model_id=custom_model.id,
    custom_model_version_id=model_version.id,
    environment_id=execution_environment.id,
    environment_version_id=environment_version.id,
    dataset_id=dataset.id,
    max_wait=3600, # 1 hour timeout
)

custom_model_test.overall_status
>>> 'succeeded'

```

In case a test fails, do the following to examine details of the failure:

```

for name, test in custom_model_test.detailed_status.items():

```

(continues on next page)

(continued from previous page)

```
print('Test: {}'.format(name))
print('Status: {}'.format(test['status']))
print('Message: {}'.format(test['message']))

print(custom_model_test.get_log())
```

To cancel a Custom Model Test, simply run:

```
custom_model_test.cancel()
```

25.5.2 List Custom Model Tests

Use the following command to list Custom Model Tests available to the user:

```
import datarobot as dr

dr.CustomModelTest.list(custom_model_id=custom_model.id)
>>> [CustomModelTest('5ec262604024031bed5aaa16')]
```

25.5.3 Retrieve Custom Model Test

To retrieve a specific Custom Model Test, run:

```
import datarobot as dr

dr.CustomModelTest.get(custom_model_test_id='5ec262604024031bed5aaa16')
>>> CustomModelTest('5ec262604024031bed5aaa16')
```

25.6 Manage Custom Inference Images

A Custom Inference Image pins a Custom Model, a Custom Model Version, an Execution Environment, and an Execution Environment version. The pinned image is used when deploying the custom model or when retrieving feature impact.

25.6.1 Create Custom Inference Image

To create a Custom Inference Image, run:

```
import datarobot as dr

custom_inference_image = dr.CustomInferenceImage.create(
    custom_model_id=custom_model.id,
    custom_model_version_id=model_version.id,
    environment_id=execution_environment.id,
    environment_version_id=environment_version.id,
)

custom_inference_image
>>> CustomInferenceImage('5ec26cfeb5ec7911cdae91b4')
```

25.6.2 List Custom Inference Images

Use the following command to list Custom Inference Images available to the user:

```
import datarobot as dr

dr.CustomInferenceImage.list()
>>> [CustomInferenceImage('5ec26cf53f750d11cdcec506')]

# use these parameters to filter results:
dr.CustomInferenceImage.list(
    # return only images with specified testing status
    testing_status='succeeded',
    # return only images with specified custom model id
    custom_model_id='5ec26cf25f2cc902bcceefd4',
    # return only images with specified custom model version id
    custom_model_version_id='5ec26cf53f750d11cdcec506',
    # return only images with specified execution environment id
    environment_id='5eb5299e4eda7b021026d696',
    # return only images with specified execution environment version id
    environment_version_id='5eb5299f9bc0570096487b14',
)
>>> [CustomInferenceImage('5ec26cf53f750d11cdcec506')]
```

Please refer to `list()` for detailed parameter description.

25.6.3 Retrieve Custom Inference Image

To retrieve a specific Custom Inference Image, run:

```
import datarobot as dr

dr.CustomInferenceImage.get('5ec26cf53f750d11cdcec506')
>>> CustomInferenceImage('5ec26cf53f750d11cdcec506')
```

25.6.4 Calculate Custom Inference Image feature impact

To trigger calculation of Custom Inference Image feature impact, training data must be assigned to a Custom Inference Model. Please refer to [Custom Inference Model documentation](#). If training data is assigned, run the following to trigger the calculation of the feature impact:

```
import datarobot as dr

image = dr.CustomInferenceImage.get('5ec26cf53f750d11cdcec506')

image.calculate_feature_impact()
```

To trigger calculating feature impact without blocking a program, set `max_wait` to `None`:

```
import datarobot as dr

image = dr.CustomInferenceImage.get('5ec26cf53f750d11cdcec506')

image.calculate_feature_impact(max_wait=None)
```


25.6.5 Retrieve Custom Inference Image feature impact

To retrieve Custom Inference Image feature impact, it must be calculated beforehand. Please refer to *Custom Inference Image feature impact documentation*. Run the following to get feature impact:

```
import datarobot as dr

image = dr.CustomInferenceImage.get('5ec26cf5ec7911cdae91b4')

image.get_feature_impact()
>>> [{'featureName': 'B', 'impactNormalized': 1.0, 'impactUnnormalized': 1.
↪1085356209402688, 'redundantWith': 'B'}...]
```

Compliance Documentation

Compliance Documentation allows users to automatically generate and download documentation to assist with deploying models in highly regulated industries. In most cases, Compliance Documentation is not available for Managed AI Cloud users. Interested users should contact their CFDS or DataRobot Support for additional information.

26.1 Generate and Download

Using the *ComplianceDocumentation* class, users can generate and download documentation as a DOCX.

```
import datarobot as dr
project = dr.Project.get('5c881d7b79bffe6efc2e16f8')
model = project.get_models()[0]

# Using the default template
doc = dr.ComplianceDocumentation(project.id, model.id)
# Start a job to generate documentation
job = doc.generate()
# Once the job is complete, download as a DOCX
job.wait_for_completion()
doc.download('/path/to/save')
```

If no *template_id* is specified, DataRobot will generate compliance documentation using a default template. To create a custom template, see below:

Compliance Documentation Template

Using the *ComplianceDocTemplate* class, users can define their own templates to make generated documents match their organization guidelines and requirements.

Templates are created from a list of *sections*, which are structured as follows:

- `contentId` : The identifier of the content in this section
- `sections` : A list of sub-section dicts nested under the parent section
- `title` : The title of the section
- `type` : The type of section - must be one of *datarobot*, *user*, or *table_of_contents*

Sections of type *user* are for custom content and include the ability to use two additional fields:

- `regularText` : regular text of the section, optionally separated by `n` to split paragraphs.
- `highlightedText` : highlighted text of the section, optionally separated by `n` to split paragraphs.

Within the above fields, users can embed DataRobot generated content using tags. Each tag looks like `{{ keyword }}` and on generation will be replaced with corresponding content. We also support parameterization for few of the tags that allow tweakable features found on the UI to be used in the templates. These can be used by placing a `|` after the keyword in the tag format `{{ keyword | parameter=value }}` Below you can find a table of currently supported tags:

Tag	Type	Parameters	Content Web Application UI Analog
{{ blueprint_diagram }}	Image		Graphical representation of the modeling pipeline. Leaderboard >> Model >> Describe >> Blueprint
{{ alternative_models }}	Table		Comparison of the model with alternatives Leaderboard built in the same project. Also known as challenger models.
{{ model_features }}	Table		Description of the model features Data >> Project Data and corresponding EDA statistics.
{{ missing_values }}	Table		Description of the missing values and their Leaderboard >> Model >> Describe >> Missing Values processing in the model.
{{ partitioning }}	Image		Graphical representation of the data partitioning. Data >> Show Advanced Options >> Partitioning (only available before project start)
{{ model_scores }}	Table		Metric scores of the model on different data sources Leaderboard >> Model
{{ lift_chart }}	Image	reverse: True, False (Default) source: validation, holdout, crossValidation bins: 10, 12, 15, 20, 30, 60	Lift Chart Leaderboard >> Model >> Evaluate
{{ feature_impact }}	Image		Feature Impact chart. Leaderboard >> Model >> Understand >> Feature Impact
{{ feature_impact_table }}	Table	sort_by: name	Table representation of Feature Impact data. Leaderboard >> Model >> Understand >> Feature Impact >> Export
{{ feature_effects }}	List of images	source: validation, holdout, crossValidation feature_names: feature1,feature2,feature3	Feature Effects charts for the top 3 features
{{ accuracy_over_time }}	Image		Accuracy over time chart. Leaderboard >> Model >> Evaluate >> Accuracy Over Time Available only for datetime partitioned projects.
{{ cv_scores }}	Table		Project metric scores for each fold. Currently unavailable in the UI Available only for projects with cross validation.

27.1 Creating a Custom Template

A common workflow includes retrieving the default template and using it as a base to extend and customize.

```
import datarobot as dr
default_template = dr.ComplianceDocTemplate.get_default()
# Download the template and edit sections on your local machine
default_template.sections_to_json_file('path/to/save')
# Create a new template from your local file
my_template = dr.ComplianceDocTemplate.create_from_json_file(name='my_template', path=
→ 'path/of/file')
```

Alternatively, custom templates can also be created from scratch.

```
sections = [{
    'title': 'Missing Values Report',
    'highlightedText': 'NOTICE',
    'regularText': 'This dataset had a lot of Missing Values. See the chart_
→ below: {{missing_values}}',
    'type': 'user'
},
{
    'title': 'Blueprints',
    'highlightedText': '',
    'regularText': '{{blueprint_diagram}} /n Blueprint for this model'
    'type': 'user'
}]
template = dr.ComplianceDocTemplate.create(name='Example', sections=sections)

# Specify the template_id to generate documentation using a custom template
doc = dr.ComplianceDocumentation(project.id, model.id, template.id)
job = doc.generate().wait_for_completion()
doc.download('/path/to/save')
```


Credentials for user with Database and Data Storage Connectivity can be stored by the system.

To interact with Credentials API, you should use the *Credential* class.

28.1 List credentials

In order to retrieve the list of all credentials accessible for current user you can use *Credential.list*.

```
import datarobot as dr

credentials = dr.Credential.list()
```

Each Credential object contains the *credential_id* string field which can be used e.g. in *Batch Predictions*.

28.2 Basic credentials

You can store generic user/password credentials:

```
>>> import datarobot as dr
>>> cred = dr.Credential.create_basic(
...     name='my_db_cred',
...     user='<user>',
...     password='<password>',
... )
>>> cred
Credential('5e429d6ecf8a5f36c5693e0f', 'my_db_cred', 'basic'),

# store cred.credential_id

>>> cred = dr.Credential.get(credential_id)
```

(continues on next page)

(continued from previous page)

```
>>> cred.credential_id
'5e429d6ecf8a5f36c5693e0f'
```

Stored credential can be used e.g. in *Batch Predictions for JDBC intake or output*.

28.3 S3 credentials

You can store AWS credentials using the three parameters:

- `aws_access_key_id`
- `aws_secret_access_key`
- `aws_session_token`

```
>>> import datarobot as dr
>>> cred = dr.Credential.create_s3(
...     name='my_s3_cred',
...     aws_access_key_id='<aws access key id>',
...     aws_secret_access_key='<aws secret access key>',
...     aws_session_token='<aws session token>',
... )
>>> cred
Credential('5e429d6ecf8a5f36c5693e03', 'my_s3_cred', 's3'),

# store cred.credential_id

>>> cred = dr.Credential.get(credential_id)
>>> cred.credential_id
'5e429d6ecf8a5f36c5693e03'
```

Stored credential can be used e.g. in *Batch Predictions for S3 intake or output*.

28.4 OAUTH credentials

You can store oauth credentials in the store:

```
>>> import datarobot as dr
>>> cred = dr.Credential.create_oauth(
...     name='my_oauth_cred',
...     token='<token>',
...     refresh_token='<refresh_token>',
... )
>>> cred
Credential('5e429d6ecf8a5f36c5693e0f', 'my_oauth_cred', 'oauth'),

# store cred.credential_id

>>> cred = dr.Credential.get(credential_id)
>>> cred.credential_id
'5e429d6ecf8a5f36c5693e0f'
```

External Testset

Testing with external datasets allows better evaluation model performance, you can compute metric scores and insights on external test dataset to ensure consistent performance prior to deployment.

Note: Not available for Time series models.

29.1 Requesting External Scores and Insights

To compute scores and insights on a dataset

Upload a prediction dataset that contains the target column `PredictionDataset.contains_target_values == True`. Dataset should be in the same structure as the original project.

```
import datarobot as dr
# Upload dataset
project = dr.Project(project_id)
dataset = project.upload_dataset('./test_set.csv')
dataset.contains_target_values
>>>True
# request external test to compute metric scores and insights on dataset
# select model using project.get_models()
external_test_job = model.request_external_test(dataset.id)
# once job is complete, scores and insights are ready for retrieving
external_test_job.wait_for_completion()
```

29.2 Retrieving External Metric Scores and Insights

After completion of external test job, metric scores and insights for external testsets will be ready.

Note: Please check `PredictionDataset.data_quality_warnings` for dataset warnings. Insights are not available if dataset is too small (less than 10 rows). ROC curve cannot be calculated if dataset has only one class in target column

29.3 Retrieving External Metric Scores

```
import datarobot as dr
# retrieving list of external metric scores on multiple datasets
metric_scores_list = dr.ExternalScores.list(project_id, model_id)
# retrieving external metric scores on one dataset
metric_scores = dr.ExternalScores.get(project_id, model_id, dataset_id)
```

29.4 Retrieving External Lift Chart

```
import datarobot as dr
# retrieving list of lift charts on multiple datasets
lift_list = dr.ExternalLiftChart.list(project_id, model_id)
# retrieving one lift chart for dataset
lift = dr.ExternalLiftChart.get(project_id, model_id, dataset_id)
```

29.5 Retrieving External Multiclass Lift Chart

Lift chart for Multiclass models only

```
import datarobot as dr
# retrieving list of lift charts on multiple datasets
lift_list = ExternalMulticlassLiftChart.list(project_id, model_id)
# retrieving one lift chart for dataset and a target class
lift = ExternalMulticlassLiftChart.get(project_id, model_id, dataset_id, target_class)
```

29.6 Retrieving External ROC Curve

Available for Binary classification models only

```
import datarobot as dr
# retrieving list of roc curves on multiple datasets
roc_list = ExternalRocCurve.list(project_id, model_id)
# retrieving one ROC curve for dataset
roc = ExternalRocCurve.get(project_id, model_id, dataset_id)
```

29.7 Retrieving Multiclass Confusion Matrix

Available for Multiclass classification models only

```
import datarobot as dr
# retrieving list of confusion charts on multiple datasets
confusion_list = ExternalConfusionChart.list(project_id, model_id)
# retrieving one confusion chart for dataset
confusion = ExternalConfusionChart.get(project_id, model_id, dataset_id)
```

29.8 Retrieving Residuals Chart

Available for Regression models only

```
import datarobot as dr
# retrieving list of residuals charts on multiple datasets
residuals_list = ExternalResidualsChart.list(project_id, model_id)
# retrieving one residuals chart for dataset
residuals = ExternalResidualsChart.get(project_id, model_id, dataset_id)
```


The Feature Discovery Project allows the user to generate features automatically from the secondary datasets which is connect to the Primary dataset(Training dataset). User can create such connection using Relationships Configuration.

30.1 Register Primary Dataset to start Project

To start the Feature Discovery Project you need to upload the primary (training) dataset *Projects*

```
import datarobot as dr
>>> primary_dataset = dr.Dataset.create_from_file(file_path='your-training_file.csv')
>>> project = dr.Project.create_from_dataset(primary_dataset.id, project_name=
↳ 'Lending Club')
```

Now, register all the secondary datasets which you want to connect with primary (training) dataset and among themselves.

30.2 Register Secondary Dataset(s) in AI Catalog

You can register the dataset using *Dataset.create_from_file* which can take either a path to a local file or any stream-able file object.

```
>>> profile_dataset = dr.Dataset.create_from_file(file_path='your_profile_file.csv')
>>> transaction_dataset = dr.Dataset.create_from_file(file_path='your_transaction_
↳ file.csv')
```

30.3 Create Relationships Configuration

Create the relationships configuration among the profile_dataset and transaction_dataset created above.

```

>>> profile_catalog_id = profile_dataset.id
>>> profile_catalog_version_id = profile_dataset.version_id

>>> transac_catalog_id = transaction_dataset.id
>>> transac_catalog_version_id = transaction_dataset.version_id

>>> dataset_definitions = [
    {
        'identifier': 'transaction',
        'catalogVersionId': transac_catalog_version_id,
        'catalogId': transac_catalog_id,
        'primaryTemporalKey': 'Date',
        'snapshotPolicy': 'latest',
    },
    {
        'identifier': 'profile',
        'catalogId': profile_catalog_id,
        'catalogVersionId': profile_catalog_version_id,
        'snapshotPolicy': 'latest',
    },
]

>>> relationships = [
    {
        'dataset2Identifier': 'profile',
        'dataset1Keys': ['CustomerID'],
        'dataset2Keys': ['CustomerID'],
        'featureDerivationWindowStart': -14,
        'featureDerivationWindowEnd': -1,
        'featureDerivationWindowTimeUnit': 'DAY',
        'predictionPointRounding': 1,
        'predictionPointRoundingTimeUnit': 'DAY',
    },
    {
        'dataset1Identifier': 'profile',
        'dataset2Identifier': 'transaction',
        'dataset1Keys': ['CustomerID'],
        'dataset2Keys': ['CustomerID'],
    },
]

# Create the relationships configuration to define connection between the datasets
>>> relationship_config = dr.RelationshipsConfiguration.create(dataset_
↳ definitions=dataset_definitions, relationships=relationships)

```

30.4 Create Feature Discovery Project

Once done with relationships configuration you can start the Feature Discovery project

```

# Set the date-time partition column which is date here
>>> partitioning_spec = dr.DatetimePartitioningSpecification('date')

# Set the target for the project and start Feature discovery
>>> project.set_target(target='BadLoan', relationships_configuration_id=relationship_
↳ config.id, mode='manual', partitioning_method=partitioning_spec)

```

(continues on next page)

(continued from previous page)

```
Project(train.csv)
```

30.4.1 Common Errors

Dataset registration Failed

```
datasetdr.Dataset.create_from_file(file_path='file.csv')
datarobot.errors.AsyncProcessUnsuccessfulError: The job did not complete successfully.
```

Solution

- Check the internet connectivity sometimes network flakiness cause upload error
- Is the dataset file too big then you might want to upload using URL rather than file

Creating relationships configuration throws some error

```
datarobot.errors.ClientError: 422 client error: {u'message': u'Invalid field data',
u'errors': {u'datasetDefinitions': {u'1': {u'identifier': u'value cannot contain_
↳characters: $ - " . { } / \\'},
u'0': {u'identifier': u'value cannot contain characters: $ - " . { } / \\'}}}}
```

Solution:

- Check the identifier name passed in datasets_definitions and relationships
- Pro tip: Dont use name of the dataset if you didnt specified the name of the dataset explicitly while registration

```
datarobot.errors.ClientError: 422 client error: {u'message': u'Invalid field data',
u'errors': {u'datasetDefinitions': {u'1': {u'primaryTemporalKey': u'date column_
↳doesn't exist'},
}}}
```

Solution:

- Check if the name of the column passed as primaryTemporalKey is correct, its case-sensitive

Relationships Configuration

A Relationships configuration specifies additional datasets to be included to a project and how these datasets are related to each other, and the primary dataset. When a relationships configuration is specified for a project, Feature Discovery will create features automatically from these datasets.

31.1 Create Relationships Configuration

You can create a relationships configuration from the uploaded catalog items. After uploading all the secondary datasets in the AI Catalog

- Create the datasets definition to define which datasets to be used as secondary datasets along with its details
- Create the relationships among the above datasets

```
import datarobot as dr
# Example of LendingClub project which has two datasets profile and transaction
>>> dataset_definitions = [
    {
        'identifier': 'transaction',
        'catalogVersionId': '5ec4aec268f0f30289a03901',
        'catalogId': '5ec4aec268f0f30289a03900',
        'primaryTemporalKey': 'Date',
        'snapshotPolicy': 'latest',
    },
    {
        'identifier': 'profile',
        'catalogId': '5ec4aec1f072bc028e3471ae',
        'catalogVersionId': '5ec4aec2f072bc028e3471b1',
        'snapshotPolicy': 'latest',
    },
]
>>> relationships = [
    {
        'dataset2Identifier': 'profile',
```

(continues on next page)

(continued from previous page)

```

        'dataset1Keys': ['CustomerID'],
        'dataset2Keys': ['CustomerID'],
        'featureDerivationWindowStart': -14,
        'featureDerivationWindowEnd': -1,
        'featureDerivationWindowTimeUnit': 'DAY',
        'predictionPointRounding': 1,
        'predictionPointRoundingTimeUnit': 'DAY',
    },
    {
        'dataset1Identifier': 'profile',
        'dataset2Identifier': 'transaction',
        'dataset1Keys': ['CustomerID'],
        'dataset2Keys': ['CustomerID'],
    },
]
>>> relationship_config = dr.RelationshipsConfiguration.create(dataset_
↳ definitions=dataset_definitions, relationships=relationships)

```

You can use the following commands to view the relationships configuration ID:

```

>>> relationship_config.id
u'5506fcd38bd88f5953219da0'

```

31.2 Retrieving Relationships Configuration

You can retrieve specific relationships configuration using the ID

```

>>> relationship_config_id = '5506fcd38bd88f5953219da0'
>>> relationship_config = dr.RelationshipsConfiguration(id=relationship_config_id).
↳ get()
>>> relationship_config.id == relationship_config_id
True
# Get all the datasets used in this relationships configuration
>> len(relationship_config.dataset_definitions) == 2
True
>> relationship_config.dataset_definitions[0]
{
    'feature_list_id': '5ec4af93603f596525d382d3',
    'snapshot_policy': 'latest',
    'catalog_id': '5ec4aec268f0f30289a03900',
    'catalog_version_id': '5ec4aec268f0f30289a03901',
    'primary_temporal_key': 'Date',
    'is_deleted': False,
    'identifier': 'transaction',
    'feature_lists':
        [
            {
                'name': 'Raw Features',
                'description': 'System created featurelist',
                'created_by': 'User1',
                'creation_date': datetime.datetime(2020, 5, 20, 4, 18, 27, 150000,
↳ tzinfo=tzutc()),
                'user_created': False,
                'dataset_id': '5ec4aec268f0f30289a03900',

```

(continues on next page)

(continued from previous page)

```

        'id': '5ec4af93603f596525d382d1',
        'features': [u'CustomerID', u'AccountID', u'Date', u'Amount', u
↪ 'Description']
    },
    {
        'name': 'universe',
        'description': 'System created featurelist',
        'created_by': 'User1',
        'creation_date': datetime.datetime(2020, 5, 20, 4, 18, 27, 172000,
↪ tzinfo=tzutc()),
        'user_created': False,
        'dataset_id': '5ec4aec268f0f30289a03900',
        'id': '5ec4af93603f596525d382d2',
        'features': [u'CustomerID', u'AccountID', u'Date', u'Amount', u
↪ 'Description']
    },
    {
        'features': [u'CustomerID', u'AccountID', u'Date', u'Amount', u
↪ 'Description'],
        'description': 'System created featurelist',
        'created_by': u'Garvit Bansal',
        'creation_date': datetime.datetime(2020, 5, 20, 4, 18, 27, 179000,
↪ tzinfo=tzutc()),
        'dataset_version_id': '5ec4aec268f0f30289a03901',
        'user_created': False,
        'dataset_id': '5ec4aec268f0f30289a03900',
        'id': u'5ec4af93603f596525d382d3',
        'name': 'Informative Features'
    }
]
}
# Get information regarding how the datasets are connected among themselves as well
↪ as primary dataset
>> relationship_config.relationships
[
    {
        'dataset2Identifier': 'profile',
        'dataset1Keys': ['CustomerID'],
        'dataset2Keys': ['CustomerID'],
        'featureDerivationWindowStart': -14,
        'featureDerivationWindowEnd': -1,
        'featureDerivationWindowTimeUnit': 'DAY',
        'predictionPointRounding': 1,
        'predictionPointRoundingTimeUnit': 'DAY',
    },
    {
        'dataset1Identifier': 'profile',
        'dataset2Identifier': 'transaction',
        'dataset1Keys': ['CustomerID'],
        'dataset2Keys': ['CustomerID'],
    },
]

```

31.3 Updating details of Relationships Configuration

You can update the details of the relationships configuration

```
>>> relationship_config_id = '5506fcd38bd88f5953219da0'
>>> relationship_config = dr.RelationshipsConfiguration(id=relationship_config_id)
# Remove the obsolete datasets definition and its relationships
>>> new_datasets_definiton =
[
    {
        'identifier': 'user',
        'catalogVersionId': '5c88a37770fc42a2fcc62759',
        'catalogId': '5c88a37770fc42a2fcc62759',
        'snapshotPolicy': 'latest',
    },
]

# Get information regarding how the datasets are connected among themselves as well_
↳as primary dataset
>>> new_relationships =
[
    {
        'dataset2Identifier': 'user',
        'dataset1Keys': ['user_id', 'dept_id'],
        'dataset2Keys': ['user_id', 'dept_id'],
    },
]
>>> new_config = relationship_config.replace(new_datasets_definiton, new_
↳relationships)
>>> new_config.id == relationship_config_id
True
>>> new_config.datasets_definition
[
    {
        'identifier': 'user',
        'catalogVersionId': '5c88a37770fc42a2fcc62759',
        'catalogId': '5c88a37770fc42a2fcc62759',
        'snapshotPolicy': 'latest',
    },
]
>>> new_config.relationships
[
    {
        'dataset2Identifier': 'user',
        'dataset1Keys': ['user_id', 'dept_id'],
        'dataset2Keys': ['user_id', 'dept_id'],
    },
]
```

31.4 Delete Relationships Configuration

You can delete the relationships configuration which is not used by any project

```
>>> relationship_config_id = '5506fcd38bd88f5953219da0'
>>> relationship_config = dr.RelationshipsConfiguration(id=relationship_config_id)
```

(continues on next page)

(continued from previous page)

```
>>> result = relationship_config.get()
>>> result.id == relationship_config_id
True
# Delete the relationships configuration
>>> relationship_config.delete()
>>> relationship_config.get()
ClientError: Relationships Configuration 5506fcd38bd88f5953219da0 not found
```


32.1 Advanced Options

```
class datarobot.helpers.AdvancedOptions (weights=None, response_cap=None,
blueprint_threshold=None, seed=None,
smart_downsampled=False, majority_downsampling_rate=None, offset=None, exposure=None, accuracy_optimized_mb=None, scale_out_modeling_mode=None, events_count=None,
monotonic_increasing_featurelist_id=None,
monotonic_decreasing_featurelist_id=None,
only_include_monotonic_blueprints=None,
allowed_pairwise_interaction_groups=None,
blend_best_models=None, scoring_code_only=None, prepare_model_for_deployment=None, consider_blenders_in_recommendation=None,
min_secondary_validation_model_count=None,
shap_only_mode=None)
```

Used when setting the target of a project to set advanced options of modeling process.

Parameters

- weights** [string, optional] The name of a column indicating the weight of each row
- response_cap** [float in [0.5, 1), optional] Quantile of the response distribution to use for response capping.
- blueprint_threshold** [int, optional] Number of hours models are permitted to run before being excluded from later autopilot stages Minimum 1
- seed** [int] a seed to use for randomization
- smart_downsampled** [bool] whether to use smart downsampling to throw away excess rows of the majority class. Only applicable to classification and zero-boosted regression projects.

majority_downsampling_rate [float] the percentage between 0 and 100 of the majority rows that should be kept. Specify only if using smart downsampling. May not cause the majority class to become smaller than the minority class.

offset [list of str, optional] (New in version v2.6) the list of the names of the columns containing the offset of each row

exposure [string, optional] (New in version v2.6) the name of a column containing the exposure of each row

accuracy_optimized_mb [bool, optional] (New in version v2.6) Include additional, longer-running models that will be run by the autopilot and available to run manually.

scaleout_modeling_mode [string, optional] (New in version v2.8) Specifies the behavior of Scaleout models for the project. This is one of `datarobot.enums.SCALEOUT_MODELING_MODE`. If `datarobot.enums.SCALEOUT_MODELING_MODE.DISABLED`, no models will run during autopilot or show in the list of available blueprints. Scaleout models must be disabled for some partitioning settings including projects using datetime partitioning or projects using offset or exposure columns. If `datarobot.enums.SCALEOUT_MODELING_MODE.REPOSITORY_ONLY`, scaleout models will be in the list of available blueprints but not run during autopilot. If `datarobot.enums.SCALEOUT_MODELING_MODE.AUTOPILOT`, scaleout models will run during autopilot and be in the list of available blueprints. Scaleout models are only supported in the Hadoop environment with the corresponding user permission set.

events_count [string, optional] (New in version v2.8) the name of a column specifying events count.

monotonic_increasing_featurelist_id [string, optional] (new in version 2.11) the id of the featurelist that defines the set of features with a monotonically increasing relationship to the target. If None, no such constraints are enforced. When specified, this will set a default for the project that can be overridden at model submission time if desired.

monotonic_decreasing_featurelist_id [string, optional] (new in version 2.11) the id of the featurelist that defines the set of features with a monotonically decreasing relationship to the target. If None, no such constraints are enforced. When specified, this will set a default for the project that can be overridden at model submission time if desired.

only_include_monotonic_blueprints [bool, optional] (new in version 2.11) when true, only blueprints that support enforcing monotonic constraints will be available in the project or selected for the autopilot.

allowed_pairwise_interaction_groups [list of tuple, optional] (New in version v2.19) For GAM models - specify groups of columns for which pairwise interactions will be allowed. E.g. if set to [(A, B, C), (C, D)] then GAM models will allow interactions between columns AxB, BxC, AxC, CxD. All others (AxD, BxD) will not be considered.

blend_best_models: bool, optional (New in version v2.19) blend best models during Autopilot run

scoring_code_only: bool, optional (New in version v2.19) Keep only models that can be converted to scorable java code during Autopilot run

shap_only_mode: bool, optional (New in version v2.21) Keep only models that support SHAP values during Autopilot run. Use SHAP-based insights wherever possible. Defaults to False.

prepare_model_for_deployment: bool, optional (New in version v2.19) Prepare model for deployment during Autopilot run. The preparation includes creating reduced feature list

models, retraining best model on higher sample size, computing insights and assigning **RECOMMENDED FOR DEPLOYMENT** label.

consider_blenders_in_recommendation: bool, optional (New in version 2.22.0) Include blenders when selecting a model to prepare for deployment in an Autopilot Run. Defaults to False.

min_secondary_validation_model_count: int, optional (New in version v2.19) Compute All backtest scores (datetime models) or cross validation scores for the specified number of highest ranking models on the Leaderboard, if over the Autopilot default.

Examples

```
import datarobot as dr
advanced_options = dr.AdvancedOptions(
    weights='weights_column',
    offset=['offset_column'],
    exposure='exposure_column',
    response_cap=0.7,
    blueprint_threshold=2,
    smart_downsampled=True, majority_downsampling_rate=75.0)
```

32.2 Batch Predictions

class `datarobot.models.BatchPredictionJob` (*data, completed_resource_url=None*)

A Batch Prediction Job is used to score large data sets on prediction servers using the Batch Prediction API.

Attributes

id [str] the id of the job

classmethod score (*deployment, intake_settings=None, output_settings=None, csv_settings=None, timeseries_settings=None, num_concurrent=None, passthrough_columns=None, passthrough_columns_set=None, max_explanations=None, threshold_high=None, threshold_low=None, prediction_warning_enabled=None, include_prediction_status=False, skip_drift_tracking=False, prediction_instance=None, abort_on_error=True, column_names_remapping=None, include_probabilities=True, include_probabilities_classes=None, download_timeout=120, download_read_timeout=660*)

Create new batch prediction job, upload the scoring dataset and return a batch prediction job.

The default intake and output options are both *localFile* which requires the caller to pass the *file* parameter and either download the results using the *download()* method afterwards or pass a path to a file where the scored data will be downloaded to afterwards.

Returns

BatchPredictionJob Instance of BatchPredictionJob

Attributes

deployment [Deployment or string ID] Deployment which will be used for scoring.

intake_settings [dict (optional)] A dict configuring how data is coming from. Supported options:

- **type** : string, either *localFile*, *s3*, *azure*, *gcp*, *dataset* or *jdbc*

To score from a local file, add the this parameter to the settings:

- `file` : file-like object, string path to file or a `pandas.DataFrame` of scoring data

To score from S3, add the next parameters to the settings:

- `url` : string, the URL to score (e.g.: `s3://bucket/key`)
- `credential_id` : string (optional)

To score from JDBC, add the next parameters to the settings:

- `data_store_id` : string, the ID of the external data store connected to the JDBC data source (see [Database Connectivity](#)).
- `query` : string (optional if `table`, `schema` and/or `catalog` is specified), a self-supplied SELECT statement of the data set you wish to predict.
- `table` : string (optional if `query` is specified), the name of specified database table.
- `schema` : string (optional if `query` is specified), the name of specified database schema.
- `catalog` : string (optional if `query` is specified), (new in v2.22) the name of specified database catalog.
- `fetch_size` : int (optional), Changing the `fetchSize` can be used to balance throughput and memory usage.
- `credential_id` : string (optional) the ID of the credentials holding information about a user with read-access to the JDBC data source (see [Credentials](#)).

output_settings [dict (optional)] A dict configuring how scored data is to be saved. Supported options:

- `type` : string, either `localFile`, `s3` or `jdbc`

To save scored data to a local file, add this parameters to the settings:

- `path` : string (optional), path to save the scored data as CSV. If a path is not specified, you must download the scored data yourself with `job.download()`. If a path is specified, the call will block until the job is done. if there are no other jobs currently processing for the targeted prediction instance, uploading, scoring, downloading will happen in parallel without waiting for a full job to complete. Otherwise, it will still block, but start downloading the scored data as soon as it starts generating data. This is the fastest method to get predictions.

To save scored data to S3, add the next parameters to the settings:

- `url` : string, the URL for storing the results (e.g.: `s3://bucket/key`)
- `credential_id` : string (optional)

To save scored data to JDBC, add the next parameters to the settings:

- `data_store_id` : string, the ID of the external data store connected to the JDBC data source (see [Database Connectivity](#)).
- `table` : string, the name of specified database table.
- `schema` : string (optional), the name of specified database schema.
- `catalog` : string (optional), (new in v2.22) the name of specified database catalog.
- `statement_type` : string, the type of insertion statement to create, one of `datarobot.enums.AVAILABLE_STATEMENT_TYPES`.

- *update_columns* : list(string) (optional), a list of strings containing those column names to be updated in case *statement_type* is set to a value related to update or upsert.
- *where_columns* : list(string) (optional), a list of strings containing those column names to be selected in case *statement_type* is set to a value related to insert or update.
- *credential_id* : string, the ID of the credentials holding information about a user with write-access to the JDBC data source (see [Credentials](#)).

csv_settings [dict (optional)] CSV intake and output settings. Supported options:

- *delimiter* : string (optional, default `,`), fields are delimited by this character. Use the string *tab* to denote TSV (TAB separated values). Must be either a one-character string or the string *tab*.
- *quotechar* : string (optional, default `"`), fields containing the delimiter must be quoted using this character.
- *encoding* : string (optional, default *utf-8*), encoding for the CSV files. For example (but not limited to): *shift_jis*, *latin_1* or *mskanji*.

timeseries_settings [dict (optional)] Configuration for time-series scoring. Supported options:

- *type* : string, must be *forecast* or *historical* (default if not passed is *forecast*). *forecast* mode makes predictions using *forecast_point* or rows in the dataset without target. *historical* enables bulk prediction mode which calculates predictions for all possible forecast points and forecast distances in the dataset within *predictions_start_date*/*predictions_end_date* range.
- *forecast_point* : datetime (optional), forecast point for the dataset, used for the forecast predictions, by default value will be inferred from the dataset. May be passed if `timeseries_settings.type=forecast`.
- *predictions_start_date* : datetime (optional), used for historical predictions in order to override date from which predictions should be calculated. By default value will be inferred automatically from the dataset. May be passed if `timeseries_settings.type=historical`.
- *predictions_end_date* : datetime (optional), used for historical predictions in order to override date from which predictions should be calculated. By default value will be inferred automatically from the dataset. May be passed if `timeseries_settings.type=historical`.
- *relax_known_in_advance_features_check* : bool, (default *False*). If True, missing values in the known in advance features are allowed in the forecast window at the prediction time. If omitted or False, missing values are not allowed.

num_concurrent [int (optional)] Number of concurrent chunks to score simultaneously. Defaults to the available number of cores of the deployment. Lower it to leave resources for real-time scoring.

passthrough_columns [list[string] (optional)] Keep these columns from the scoring dataset in the scored dataset. This is useful for correlating predictions with source data.

passthrough_columns_set [string (optional)] To pass through every column from the scoring dataset, set this to *all*. Takes precedence over *passthrough_columns* if set.

max_explanations [int (optional)] Compute prediction explanations for this amount of features.

threshold_high [float (optional)] Only compute prediction explanations for predictions above this threshold. Can be combined with *threshold_low*.

threshold_low [float (optional)] Only compute prediction explanations for predictions below this threshold. Can be combined with *threshold_high*.

prediction_warning_enabled [boolean (optional)] Add prediction warnings to the scored data. Currently only supported for regression models.

include_prediction_status [boolean (optional)] Include the prediction_status column in the output, defaults to *False*.

skip_drift_tracking [boolean (optional)] Skips drift tracking on any predictions made from this job. This is useful when running non-production workloads to not affect drift tracking and cause unnecessary alerts. Defaults to *False*.

prediction_instance [dict (optional)] Defaults to instance specified by deployment or system configuration. Supported options:

- *hostName* : string
- *sslEnabled* : boolean (optional, default *true*). Set to *false* to run prediction requests from the batch prediction job without SSL.
- *datarobotKey* : string (optional), if running a job against a prediction instance in the Managed AI Cloud, you must provide the organization level DataRobot-Key
- *apiKey* : string (optional), by default, prediction requests will use the API key of the user that created the job. This allows you to make requests on behalf of other users.

abort_on_error [boolean (optional)] Default behaviour is to abort the job if too many rows fail scoring. This will free up resources for other jobs that may score successfully. Set to *false* to unconditionally score every row no matter how many errors are encountered. Defaults to *True*.

column_names_remapping [dict (optional)] Mapping with column renaming for output table. Defaults to *{}*.

include_probabilities [boolean (optional)] Flag that enables returning of all probability columns. Defaults to *True*.

include_probabilities_classes [list (optional)] List the subset of classes if a user doesn't want all the classes. Defaults to *[]*.

download_timeout [int (optional)] New in version 2.22.

If using localFile output, wait this many seconds for the download to become available. See *download()*.

download_read_timeout [int (optional, default 660)] New in version 2.22.

If using localFile output, wait this many seconds for the server to respond between chunks.

classmethod score_to_file (*deployment, intake_path, output_path, **kwargs*)

Create new batch prediction job, upload the scoring dataset and download the scored CSV file concurrently.

Will block until the entire file is scored.

Refer to the *create* method for details on the other *kwargs* parameters.

Returns

BatchPredictionJob Instance of BatchPredictionJob

Attributes

deployment [Deployment or string ID] Deployment which will be used for scoring.

intake_path [file-like object/string path to file/pandas.DataFrame] Scoring data

output_path [str] Filename to save the result under

classmethod score_s3 (*deployment, source_url, destination_url, credential=None, **kwargs*)

Create new batch prediction job, with a scoring dataset from S3 and writing the result back to S3.

This returns immediately after the job has been created. You must poll for job completion using *get_status()* or *wait_for_completion()*.

Refer to the *create* method for details on the other *kwargs* parameters.

Returns

BatchPredictionJob Instance of BatchPredictionJob

Attributes

deployment [Deployment or string ID] Deployment which will be used for scoring.

source_url [string] The URL for the prediction dataset (e.g.: *s3://bucket/key*)

destination_url [string] The URL for the scored dataset (e.g.: *s3://bucket/key*)

credential [string or Credential (optional)] The AWS Credential object or credential id

classmethod score_azure (*deployment, source_url, destination_url, credential=None, **kwargs*)

Create new batch prediction job, with a scoring dataset from Azure blob storage and writing the result back to Azure blob storage.

This returns immediately after the job has been created. You must poll for job completion using *get_status()* or *wait_for_completion()*.

Refer to the *create* method for details on the other *kwargs* parameters.

Returns

BatchPredictionJob Instance of BatchPredictionJob

Attributes

deployment [Deployment or string ID] Deployment which will be used for scoring.

source_url [string] The URL for the prediction dataset (e.g.: *https://storage_account.blob.endpoint/container/blob_name*)

destination_url [string] The URL for the scored dataset (e.g.: *https://storage_account.blob.endpoint/container/blob_name*)

credential [string or Credential (optional)] The Azure Credential object or credential id

classmethod score_gcp (*deployment, source_url, destination_url, credential=None, **kwargs*)

Create new batch prediction job, with a scoring dataset from Google Cloud Storage and writing the result back to one.

This returns immediately after the job has been created. You must poll for job completion using *get_status()* or *wait_for_completion()*.

Refer to the *create* method for details on the other *kwargs* parameters.

Returns

BatchPredictionJob Instance of BatchPredictionJob

Attributes

deployment [Deployment or string ID] Deployment which will be used for scoring.

source_url [string] The URL for the prediction dataset (e.g.: *http(s)://storage.googleapis.com/[bucket]/[object]*)

destination_url [string] The URL for the scored dataset (e.g.: *http(s)://storage.googleapis.com/[bucket]/[object]*)

credential [string or Credential (optional)] The GCP Credential object or credential id

classmethod `score_from_existing` (*batch_prediction_job_id*)

Create a new batch prediction job based on the settings from a previously created one

Returns

BatchPredictionJob Instance of BatchPredictonJob

Attributes

batch_prediction_job_id: str ID of the previous batch prediction job

classmethod `get` (*batch_prediction_job_id*)

Get batch prediction job

Returns

BatchPredictionJob Instance of BatchPredictonJob

Attributes

batch_prediction_job_id: str ID of batch prediction job

download (*fileobj*, *timeout=120*, *read_timeout=660*)

Downloads the CSV result of a prediction job

Attributes

fileobj: file-like object Write CSV data to this file-like object

timeout [int (optional, default 120)] New in version 2.22.

Seconds to wait for the download to become available.

The download will not be available before the job has started processing. In case other jobs are occupying the queue, processing may not start immediately.

If the timeout is reached, the job will be aborted and *RuntimeError* is raised.

Set to -1 to wait infinitely.

read_timeout [int (optional, default 660)] New in version 2.22.

Seconds to wait for the server to respond between chunks.

delete ()

Cancel this job. If this job has not finished running, it will be removed and canceled.

get_status ()

Get status of batch prediction job

Returns

BatchPredictionJob status data Dict with job status

classmethod `list_by_status` (*statuses=None*)

Get jobs collection for specific set of statuses

Returns

BatchPredictionJob statuses List of job statuses dicts with specific statuses

Attributes

statuses List of statuses to filter jobs ([ABORTED|COMPLETED|*...*]) if statuses is not provided, returns all jobs for user

32.3 Blueprint

```
class datarobot.models.Blueprint (id=None, processes=None, model_type=None,  
                                project_id=None, blueprint_category=None, mono-  
                                tonic_increasing_featurelist_id=None, mono-  
                                tonic_decreasing_featurelist_id=None, sup-  
                                ports_monotonic_constraints=None, recom-  
                                mended_featurelist_id=None)
```

A Blueprint which can be used to fit models

Attributes

id [str] the id of the blueprint

processes [list of str] the processes used by the blueprint

model_type [str] the model produced by the blueprint

project_id [str] the project the blueprint belongs to

blueprint_category [str] (New in version v2.6) Describes the category of the blueprint and the kind of model it produces.

recommended_featurelist_id: str or null (New in v2.18) The ID of the feature list recommended for this blueprint. If this field is not present, then there is no recommended feature list.

```
classmethod get (project_id, blueprint_id)
```

Retrieve a blueprint.

Parameters

project_id [str] The project's id.

blueprint_id [str] Id of blueprint to retrieve.

Returns

blueprint [Blueprint] The queried blueprint.

```
get_chart ()
```

Retrieve a chart.

Returns

BlueprintChart The current blueprint chart.

```
get_documents ()
```

Get documentation for tasks used in the blueprint.

Returns

list of BlueprintTaskDocument All documents available for blueprint.

```
class datarobot.models.BlueprintTaskDocument (title=None, task=None, description=None,  
                                              parameters=None, links=None, refer-  
                                              ences=None)
```

Document describing a task from a blueprint.

Attributes

title [str] Title of document.

task [str] Name of the task described in document.

description [str] Task description.

parameters [list of dict(name, type, description)] Parameters that task can receive in human-readable format.

links [list of dict(name, url)] External links used in document

references [list of dict(name, url)] References used in document. When no link available url equals None.

```
class datarobot.models.BlueprintChart (nodes, edges)
```

A Blueprint chart that can be used to understand data flow in blueprint.

Attributes

nodes [list of dict (id, label)] Chart nodes, id unique in chart.

edges [list of tuple (id1, id2)] Directions of data flow between blueprint chart nodes.

```
classmethod get (project_id, blueprint_id)
```

Retrieve a blueprint chart.

Parameters

project_id [str] The project's id.

blueprint_id [str] Id of blueprint to retrieve chart.

Returns

BlueprintChart The queried blueprint chart.

```
to_graphviz ()
```

Get blueprint chart in graphviz DOT format.

Returns

unicode String representation of chart in graphviz DOT language.

```
class datarobot.models.ModelBlueprintChart (nodes, edges)
```

A Blueprint chart that can be used to understand data flow in model. Model blueprint chart represents reduced repository blueprint chart with only elements that used to build this particular model.

Attributes

nodes [list of dict (id, label)] Chart nodes, id unique in chart.

edges [list of tuple (id1, id2)] Directions of data flow between blueprint chart nodes.

```
classmethod get (project_id, model_id)
```

Retrieve a model blueprint chart.

Parameters

project_id [str] The project's id.

model_id [str] Id of model to retrieve model blueprint chart.

Returns

ModelBlueprintChart The queried model blueprint chart.

to_graphviz()

Get blueprint chart in graphviz DOT format.

Returns

unicode String representation of chart in graphviz DOT language.

32.4 Calendar File

```
class datarobot.CalendarFile(calendar_end_date=None, calendar_start_date=None, created=None, id=None, name=None, num_event_types=None, num_events=None, project_ids=None, role=None, multiseries_id_columns=None)
```

Represents the data for a calendar file.

For more information about calendar files, see the [calendar documentation](#).

Attributes

id [str] The id of the calendar file.

calendar_start_date [str] The earliest date in the calendar.

calendar_end_date [str] The last date in the calendar.

created [str] The date this calendar was created, i.e. uploaded to DR.

name [str] The name of the calendar.

num_event_types [int] The number of different event types.

num_events [int] The number of events this calendar has.

project_ids [list of strings] A list containing the projectIds of the projects using this calendar.

multiseries_id_columns: list of str or None A list of columns in calendar which uniquely identify events for different series. Currently, only one column is supported. If multiseries id columns are not provided, calendar is considered to be single series.

role [str] The access role the user has for this calendar.

```
classmethod create(file_path, calendar_name=None, multiseries_id_columns=None)
```

Creates a calendar using the given file. For information about calendar files, see the [calendar documentation](#)

The provided file must be a CSV in the format:

Date,	Event,	Series ID
<date>, <event_type>,<series id>		
<date>, <event_type>,<		

A header row is required, and the "Series ID" column is optional.

Once the CalendarFile has been created, pass its ID with the [DatetimePartitioningSpecification](#) when setting the target for a time series project in order to use it.

Parameters

file_path [string] A string representing a path to a local csv file.

calendar_name [string, optional] A name to assign to the calendar. Defaults to the name of the file if not provided.

multiseries_id_columns [list of str or None] a list of the names of multiseries id columns to define which series an event belongs to. Currently only one multiseries id column is supported.

Returns

calendar_file [CalendarFile] Instance with initialized data.

Raises

AsyncProcessUnsuccessfulError Raised if there was an error processing the provided calendar file.

Examples

```
# Creating a calendar with a specified name
cal = dr.CalendarFile.create('/home/calendars/somecalendar.csv',
                             calendar_name='Some Calendar Name')

cal.id
>>> 5c1d4904211c0a061bc93013
cal.name
>>> Some Calendar Name

# Creating a calendar without specifying a name
cal = dr.CalendarFile.create('/home/calendars/somecalendar.csv')
cal.id
>>> 5c1d4904211c0a061bc93012
cal.name
>>> somecalendar.csv

# Creating a calendar with multiseries id columns
cal = dr.CalendarFile.create('/home/calendars/somemultiseriescalendar.csv',
                             calendar_name='Some Multiseries Calendar Name',
                             multiseries_id_columns=['series_id'])

cal.id
>>> 5da9bb21962d746f97e4daee
cal.name
>>> Some Multiseries Calendar Name
cal.multiseries_id_columns
>>> ['series_id']
```

classmethod `get(calendar_id)`

Gets the details of a calendar, given the id.

Parameters

calendar_id [str] The identifier of the calendar.

Returns

calendar_file [CalendarFile] The requested calendar.

Raises

DataError Raised if the `calendar_id` is invalid, i.e. the specified `CalendarFile` does not exist.

Examples

```
cal = dr.CalendarFile.get(some_calendar_id)
cal.id
>>> some_calendar_id
```

classmethod list (*project_id=None, batch_size=None*)

Gets the details of all calendars this user has view access for.

Parameters

project_id [str, optional] If provided, will filter for calendars associated only with the specified project.

batch_size [int, optional] The number of calendars to retrieve in a single API call. If specified, the client may make multiple calls to retrieve the full list of calendars. If not specified, an appropriate default will be chosen by the server.

Returns

calendar_list [list of *CalendarFile*] A list of CalendarFile objects.

Examples

```
calendars = dr.CalendarFile.list()
len(calendars)
>>> 10
```

classmethod delete (*calendar_id*)

Deletes the calendar specified by calendar_id.

Parameters

calendar_id [str] The id of the calendar to delete. The requester must have OWNER access for this calendar.

Raises

ClientError Raised if an invalid calendar_id is provided.

Examples

```
# Deleting with a valid calendar_id
status_code = dr.CalendarFile.delete(some_calendar_id)
status_code
>>> 204
dr.CalendarFile.get(some_calendar_id)
>>> ClientError: Item not found
```

classmethod update_name (*calendar_id, new_calendar_name*)

Changes the name of the specified calendar to the specified name. The requester must have at least READ_WRITE permissions on the calendar.

Parameters

calendar_id [str] The id of the calendar to update.

new_calendar_name [str] The new name to set for the specified calendar.

Returns

status_code [int] 200 for success

Raises

ClientError Raised if an invalid calendar_id is provided.

Examples

```
response = dr.CalendarFile.update_name(some_calendar_id, some_new_name)
response
>>> 200
cal = dr.CalendarFile.get(some_calendar_id)
cal.name
>>> some_new_name
```

classmethod share (calendar_id, access_list)

Shares the calendar with the specified users, assigning the specified roles.

Parameters

calendar_id [str] The id of the calendar to update

access_list: A list of dr.SharingAccess objects. Specify *None* for the role to delete a user's access from the specified CalendarFile. For more information on specific access levels, see the [sharing](#) documentation.

Returns

status_code [int] 200 for success

Raises

ClientError Raised if unable to update permissions for a user.

AssertionError Raised if access_list is invalid.

Examples

```
# assuming some_user is a valid user, share this calendar with some_user
sharing_list = [dr.SharingAccess(some_user_username,
                                dr.enums.SHARING_ROLE.READ_WRITE)]
response = dr.CalendarFile.share(some_calendar_id, sharing_list)
response.status_code
>>> 200

# delete some_user from this calendar, assuming they have access of some kind_
↪ already
delete_sharing_list = [dr.SharingAccess(some_user_username,
                                         None)]
response = dr.CalendarFile.share(some_calendar_id, delete_sharing_list)
response.status_code
>>> 200

# Attempt to add an invalid user to a calendar
invalid_sharing_list = [dr.SharingAccess(invalid_username,
                                         dr.enums.SHARING_ROLE.READ_WRITE)]
```

(continues on next page)

(continued from previous page)

```
dr.CalendarFile.share(some_calendar_id, invalid_sharing_list)
>>> ClientError: Unable to update access for this calendar
```

classmethod `get_access_list` (*calendar_id*, *batch_size=None*)

Retrieve a list of users that have access to this calendar.

Parameters

calendar_id [str] The id of the calendar to retrieve the access list for.

batch_size [int, optional] The number of access records to retrieve in a single API call. If specified, the client may make multiple calls to retrieve the full list of calendars. If not specified, an appropriate default will be chosen by the server.

Returns

access_control_list [list of *SharingAccess*] A list of *SharingAccess* objects.

Raises

ClientError Raised if user does not have access to calendar or calendar does not exist.

32.5 Compliance Documentation Templates

```
class datarobot.models.compliance_doc_template.ComplianceDocTemplate (id, creator_id,
                                                                    creator_username,
                                                                    name,
                                                                    org_id=None,
                                                                    sections=None)
```

A *compliance documentation template*. Templates are used to customize contents of *ComplianceDocumentation*.

New in version v2.14.

Notes

Each section dictionary has the following schema:

- **title** : title of the section
- **type** : type of section. Must be one of "datarobot", "user" or "table_of_contents".

Each type of section has a different set of attributes described below.

Section of type "datarobot" represent a section owned by DataRobot. DataRobot sections have the following additional attributes:

- **content_id** : The identifier of the content in this section. You can get the default template with *get_default* for a complete list of possible DataRobot section content ids.
- **sections** : list of sub-section dicts nested under the parent section.

Section of type "user" represent a section with user-defined content. Those sections may contain text generated by user and have the following additional fields:

- **regularText** : regular text of the section, optionally separated by \n to split paragraphs.

- `highlightedText` : highlighted text of the section, optionally separated by `\n` to split paragraphs.
- `sections` : list of sub-section dicts nested under the parent section.

Section of type `"table_of_contents"` represent a table of contents and has no additional attributes.

Attributes

id [str] the id of the template

name [str] the name of the template.

creator_id [str] the id of the user who created the template

creator_username [str] username of the user who created the template

org_id [str] the id of the organization the template belongs to

sections [list of dicts] the sections of the template describing the structure of the document.
Section schema is described in Notes section above.

classmethod `get_default` (*template_type=None*)

Get a default DataRobot template. This template is used for generating compliance documentation when no template is specified.

Parameters

template_type [str or None] Type of the template. Currently supported values are `normal` and `time_series`

Returns

template [ComplianceDocTemplate] the default template object with `sections` attribute populated with default sections.

classmethod `create_from_json_file` (*name, path*)

Create a template with the specified name and sections in a JSON file.

This is useful when working with sections in a JSON file. Example:

```
default_template = ComplianceDocTemplate.get_default()
default_template.sections_to_json_file('path/to/example.json')
# ... edit example.json in your editor
my_template = ComplianceDocTemplate.create_from_json_file(
    name='my template',
    path='path/to/example.json'
)
```

Parameters

name [str] the name of the template. Must be unique for your user.

path [str] the path to find the JSON file at

Returns

template [ComplianceDocTemplate] the created template

classmethod `create` (*name, sections*)

Create a template with the specified name and sections.

Parameters

name [str] the name of the template. Must be unique for your user.

sections [list] list of section objects

Returns

template [ComplianceDocTemplate] the created template

classmethod get (*template_id*)

Retrieve a specific template.

Parameters

template_id [str] the id of the template to retrieve

Returns

template [ComplianceDocTemplate] the retrieved template

classmethod list (*name_part=None, limit=None, offset=None*)

Get a paginated list of compliance documentation template objects.

Parameters

name_part [str or None] Return only the templates with names matching specified string. The matching is case-insensitive.

limit [int] The number of records to return. The server will use a (possibly finite) default if not specified.

offset [int] The number of records to skip.

Returns

templates [list of ComplianceDocTemplate] the list of template objects

sections_to_json_file (*path, indent=2*)

Save sections of the template to a json file at the specified path

Parameters

path [str] the path to save the file to

indent [int] indentation to use in the json file.

update (*name=None, sections=None*)

Update the name or sections of an existing doc template.

Note that default or non-existent templates can not be updated.

Parameters

name [str, optional] the new name for the template

sections [list of dicts] list of sections

delete ()

Delete the compliance documentation template.

32.6 Compliance Documentation

class datarobot.models.compliance_documentation.**ComplianceDocumentation** (*project_id, model_id, template_id=None*)

A *compliance documentation* object.

New in version v2.14.

Examples

```
doc = ComplianceDocumentation('project-id', 'model-id')
job = doc.generate()
job.wait_for_completion()
doc.download('example.docx')
```

Attributes

project_id [str] the id of the project

model_id [str] the id of the model

template_id [str or None] optional id of the template for the generated doc. See documentation for [ComplianceDocTemplate](#) for more info.

generate()

Start a job generating model compliance documentation.

Returns

Job an instance of an async job

download(filepath)

Download the generated compliance documentation file and save it to the specified path. The generated file has a DOCX format.

Parameters

filepath [str] A file path, e.g. `~/path/to/save/compliance_documentation.docx`

32.7 Confusion Chart

class `datarobot.models.confusion_chart.ConfusionChart` (*source*, *data*,
source_model_id)
Confusion Chart data for model.

Notes

`ClassMetrics` is a dict containing the following:

- `class_name` (string) name of the class
- `actual_count` (int) number of times this class is seen in the validation data
- `predicted_count` (int) number of times this class has been predicted for the validation data
- `f1` (float) F1 score
- `recall` (float) recall score
- `precision` (float) precision score
- **`was_actual_percentages` (list of dict) one vs all actual percentages in format specified below.**
 - `other_class_name` (string) the name of the other class
 - `percentage` (float) the percentage of the times this class was predicted when it was actually class (from 0 to 1)

- **was_predicted_percentages** (list of dict) one vs all predicted percentages in format specified below.
 - `other_class_name` (string) the name of the other class
 - `percentage` (float) the percentage of the times this class was actual predicted (from 0 to 1)
- **confusion_matrix_one_vs_all** (list of list) 2d list representing 2x2 one vs all matrix.
 - This represents the True/False Negative/Positive rates as integer for each class. The data structure looks like:
 - `[[True Negative, False Positive], [False Negative, True Positive]]`

Attributes

- source** [str] Confusion Chart data source. Can be `validation`, `crossValidation` or `holdout`.
- raw_data** [dict] All of the raw data for the Confusion Chart
- confusion_matrix** [list of list] The NxN confusion matrix
- classes** [list] The names of each of the classes
- class_metrics** [list of dicts] List of dicts with schema described as `ClassMetrics` above.
- source_model_id** [str] ID of the model this Confusion chart represents; in some cases, insights from the parent of a frozen model may be used

32.8 Credentials

class `datarobot.models.Credential` (*credential_id=None, name=None, credential_type=None, creation_date=None, description=None*)

classmethod `list()`

Returns list of available credentials.

Returns

credentials [list of Credential instances] contains a list of available credentials.

Examples

```
>>> import datarobot as dr
>>> data_sources = dr.Credential.list()
>>> data_sources
[
  Credential('5e429d6ecf8a5f36c5693e03', 'my_s3_cred', 's3'),
  Credential('5e42cc4dcf8a5f3256865840', 'my_jdbc_cred', 'jdbc'),
]
```

classmethod `get(credential_id)`

Gets the Credential.

Parameters

credential_id [str] the identifier of the credential.

Returns

credential [Credential] the requested credential.

Examples

```
>>> import datarobot as dr
>>> cred = dr.Credential.get('5a8ac9ab07a57a0001be501f')
>>> cred
Credential('5e429d6ecf8a5f36c5693e03', 'my_s3_cred', 's3'),
```

delete()

Deletes the Credential the store.

Parameters

credential_id [str] the identifier of the credential.

Returns

credential [Credential] the requested credential.

Examples

```
>>> import datarobot as dr
>>> cred = dr.Credential.get('5a8ac9ab07a57a0001be501f')
>>> cred.delete()
```

classmethod create_basic (*name, user, password, description=None*)

Creates the credentials.

Parameters

name [str] the name to use for this set of credentials.

user [str] the username to store for this set of credentials.

password [str] the password to store for this set of credentials.

description [str, optional] the description to use for this set of credentials.

Returns

credential [Credential] the created credential.

Examples

```
>>> import datarobot as dr
>>> cred = dr.Credential.create_basic(
...     name='my_basic_cred',
...     user='username',
...     password='password',
... )
>>> cred
Credential('5e429d6ecf8a5f36c5693e03', 'my_basic_cred', 'basic'),
```

classmethod create_oauth (*name, token, refresh_token, description=None*)

Creates the OAUTH credentials.

Parameters

- name** [str] the name to use for this set of credentials.
- token: str** the OAUTH token
- refresh_token: str** The OAUTH token
- description** [str, optional] the description to use for this set of credentials.

Returns

- credential** [Credential] the created credential.

Examples

```
>>> import datarobot as dr
>>> cred = dr.Credential.create_oauth(
...     name='my_oauth_cred',
...     token='XXX',
...     refresh_token='YYY',
... )
>>> cred
Credential('5e429d6ecf8a5f36c5693e03', 'my_oauth_cred', 'oauth'),
```

classmethod create_s3 (name, aws_access_key_id=None, aws_secret_access_key=None, aws_session_token=None, description=None)
Creates the S3 credentials.

Parameters

- name** [str] the name to use for this set of credentials.
- aws_access_key_id** [str, optional] the AWS access key id.
- aws_secret_access_key** [str, optional] the AWS secret access key.
- aws_session_token** [str, optional] the AWS session token.
- description** [str, optional] the description to use for this set of credentials.

Returns

- credential** [Credential] the created credential.

Examples

```
>>> import datarobot as dr
>>> cred = dr.Credential.create_s3(
...     name='my_s3_cred',
...     aws_access_key_id='XXX',
...     aws_secret_access_key='YYY',
...     aws_session_token='ZZZ',
... )
>>> cred
Credential('5e429d6ecf8a5f36c5693e03', 'my_s3_cred', 's3'),
```

classmethod create_azure (name, azure_connection_string, description=None)
Creates the Azure storage credentials.

Parameters

name [str] the name to use for this set of credentials.

azure_connection_string [str] the Azure connection string.

description [str, optional] the description to use for this set of credentials.

Returns

credential [Credential] the created credential.

Examples

```
>>> import datarobot as dr
>>> cred = dr.Credential.create_azure(
...     name='my_azure_cred',
...     azure_connection_string='XXX',
... )
>>> cred
Credential('5e429d6ecf8a5f36c5693e03', 'my_azure_cred', 'azure'),
```

classmethod create_gcp (*name*, *gcp_key=None*, *description=None*)

Creates the GCP credentials.

Parameters

name [str] the name to use for this set of credentials.

gcp_key [str] the GCP key in json format.

description [str, optional] the description to use for this set of credentials.

Returns

credential [Credential] the created credential.

Examples

```
>>> import datarobot as dr
>>> cred = dr.Credential.create_gcp(
...     name='my_gcp_cred',
...     gcp_key='XXX',
... )
>>> cred
Credential('5e429d6ecf8a5f36c5693e03', 'my_gcp_cred', 'gcp'),
```

32.9 Custom Models

class `datarobot.models.custom_model_version.CustomModelFileItem` (*id*, *file_name*,
file_path,
file_source,
created_at=None)

A file item attached to a DataRobot custom model version.

New in version v2.21.

Attributes

id: str id of the file item

file_name: str name of the file item

file_path: str path of the file item

file_source: str source of the file item

created_at: str, optional ISO-8601 formatted timestamp of when the version was created

class `datarobot.CustomInferenceImage` (***kwargs*)

An image of a custom model.

New in version v2.21.

Attributes

id: str image id

custom_model: dict dict with 2 keys: *id* and *name*, where *id* is the ID of the custom model and *name* is the model name

custom_model_version: dict dict with 2 keys: *id* and *label*, where *id* is the ID of the custom model version and *label* is the version label

execution_environment: dict dict with 2 keys: *id* and *name*, where *id* is the ID of the execution environment and *name* is the environment name

execution_environment_version: dict dict with 2 keys: *id* and *label*, where *id* is the ID of the execution environment version and *label* is the version label

latest_test: dict, optional dict with 3 keys: *id*, *status* and *completedAt*, where *id* is the ID of the latest test, *status* is the testing status and *completedAt* is ISO-8601 formatted timestamp of when the testing was completed

classmethod `create` (*custom_model_id*, *custom_model_version_id*, *environment_id*, *environment_version_id=None*)

Create a custom model image.

New in version v2.21.

Parameters

custom_model_id: str the id of the custom model

custom_model_version_id: str the id of the custom model version

environment_id: str the id of the execution environment

environment_version_id: str, optional the id of the execution environment version

Returns

CustomInferenceImage created custom model image

Raises

datarobot.errors.ClientError if the server responded with 4xx status

datarobot.errors.ServerError if the server responded with 5xx status

classmethod `list` (*testing_status=None*, *custom_model_id=None*, *custom_model_version_id=None*, *environment_id=None*, *environment_version_id=None*)

List custom model images.

New in version v2.21.

Parameters

testing_status: str, optional the testing status to filter results by
custom_model_id: str, optional the id of the custom model
custom_model_version_id: str, optional the id of the custom model version
environment_id: str, optional the id of the execution environment
environment_version_id: str, optional the id of the execution environment version

Returns

List[CustomModelImage] a list of custom model images

Raises

datarobot.errors.ClientError if the server responded with 4xx status
datarobot.errors.ServerError if the server responded with 5xx status

classmethod **get** (*custom_model_image_id*)

Get custom model image by id.

New in version v2.21.

Parameters

custom_model_image_id: str the id of the custom model image

Returns

CustomInferenceImage retrieved custom model image

Raises

datarobot.errors.ClientError if the server responded with 4xx status.
datarobot.errors.ServerError if the server responded with 5xx status.

refresh ()

Update custom inference image with the latest data from server.

New in version v2.21.

Raises

datarobot.errors.ClientError if the server responded with 4xx status
datarobot.errors.ServerError if the server responded with 5xx status

get_feature_impact (*with_metadata=False*)

Get custom model feature impact.

New in version v2.21.

Parameters

with_metadata [bool] The flag indicating if the result should include the metadata as well.

Returns

feature_impacts [list of dict] The feature impact data. Each item is a dict with the keys `featureName`, `impactNormalized`, and `impactUnnormalized`, and `redundantWith`.

Raises

datarobot.errors.ClientError if the server responded with 4xx status.

datarobot.errors.ServerError if the server responded with 5xx status.

calculate_feature_impact (*max_wait=600*)

Calculate custom model feature impact.

New in version v2.22.

Parameters

max_wait: int, optional max time to wait for feature impact calculation. If set to None - method will return without waiting. Defaults to 10 min

Raises

datarobot.errors.ClientError if the server responded with 4xx status

datarobot.errors.ServerError if the server responded with 5xx status

class `datarobot.CustomInferenceModel` (**args, **kwargs*)

A custom inference model.

New in version v2.21.

Attributes

id: str id of the custom model

name: str name of the custom model

language: str programming language of the custom model. Can be `python`, `java`, or `other`

description: str description of the custom model

target_type: datarobot.TARGET_TYPE custom model target type. Can be `datarobot.TARGET_TYPE.BINARY` or `datarobot.TARGET_TYPE.REGRESSION`

latest_version: datarobot.CustomModelVersion or None latest version of the custom model if the model has a latest version

deployments_count: int number of a deployments of the custom models

target_name: str custom model target name

positive_class_label: str for binary classification projects, a label of a positive class

negative_class_label: str for binary classification projects, a label of a negative class

prediction_threshold: float for binary classification projects, a threshold used for predictions

training_data_assignment_in_progress: bool flag describing if training data assignment is in progress

training_dataset_id: str, optional id of a dataset assigned to the custom model

training_dataset_version_id: str, optional id of a dataset version assigned to the custom model

training_data_file_name: str, optional name of assigned training data file

training_data_partition_column: str, optional name of a partition column in a training dataset assigned to the custom model

created_by: str username of a user who user who created the custom model

updated_at: str ISO-8601 formatted timestamp of when the custom model was updated

created_at: str ISO-8601 formatted timestamp of when the custom model was created

classmethod `list` (*is_deployed=None, search_for=None, order_by=None*)

List custom inference models available to the user.

New in version v2.21.

Parameters

is_deployed: bool, optional flag for filtering custom inference models. If set to *True*, only deployed custom inference models are returned. If set to *False*, only not deployed custom inference models are returned

search_for: str, optional string for filtering custom inference models - only custom inference models that contain the string in name or description will be returned. If not specified, all custom models will be returned

order_by: str, optional property to sort custom inference models by. Supported properties are `created` and `updated`. Prefix the attribute name with a dash to sort in descending order, e.g. `order_by=-created`. By default, the `order_by` parameter is *None* which will result in custom models being returned in order of creation time descending

Returns

List[CustomInferenceModel] a list of custom inference models.

Raises

datarobot.errors.ClientError if the server responded with 4xx status

datarobot.errors.ServerError if the server responded with 5xx status

classmethod `get` (*custom_model_id*)

Get custom inference model by id.

New in version v2.21.

Parameters

custom_model_id: str id of the custom inference model

Returns

CustomInferenceModel retrieved custom inference model

Raises

datarobot.errors.ClientError if the server responded with 4xx status.

datarobot.errors.ServerError if the server responded with 5xx status.

download_latest_version (*file_path*)

Download the latest custom inference model version.

New in version v2.21.

Parameters

file_path: str path to create a file with custom model version content

Raises

datarobot.errors.ClientError if the server responded with 4xx status.

datarobot.errors.ServerError if the server responded with 5xx status.

classmethod create (*name, target_type, target_name, language=None, description=None, positive_class_label=None, negative_class_label=None, prediction_threshold=None*)

Create a custom inference model.

New in version v2.21.

Parameters

name: str name of the custom inference model

target_type: datarobot.TARGET_TYPE target type of the custom inference model. Can be *datarobot.TARGET_TYPE.BINARY* or *datarobot.TARGET_TYPE.REGRESSION*

language: str, optional programming language of the custom learning model

description: str, optional description of the custom learning model

positive_class_label: str, optional custom inference model positive class label

negative_class_label: str, optional custom inference model negative class label

prediction_threshold: float, optional custom inference model prediction threshold

Returns

CustomInferenceModel created a custom inference model

Raises

datarobot.errors.ClientError if the server responded with 4xx status

datarobot.errors.ServerError if the server responded with 5xx status

classmethod copy_custom_model (*custom_model_id*)

Create a custom inference model by copying existing one.

New in version v2.21.

Parameters

custom_model_id: str id of the custom inference model to copy

Returns

CustomInferenceModel created a custom inference model

Raises

datarobot.errors.ClientError if the server responded with 4xx status

datarobot.errors.ServerError if the server responded with 5xx status

update (*name=None, language=None, description=None, target_name=None, positive_class_label=None, negative_class_label=None, prediction_threshold=None*)

Update custom inference model properties.

New in version v2.21.

Parameters

name: str, optional new custom inference model name

language: str, optional new custom inference model programming language

description: str, optional new custom inference model description

target_name: str, optional new custom inference model target name

positive_class_label: str, optional new custom inference model positive class label

negative_class_label: str, optional new custom inference model negative class label
prediction_threshold: float, optional new custom inference model prediction threshold

Raises

datarobot.errors.ClientError if the server responded with 4xx status.
datarobot.errors.ServerError if the server responded with 5xx status.

refresh()

Update custom inference model with the latest data from server.

New in version v2.21.

Raises

datarobot.errors.ClientError if the server responded with 4xx status
datarobot.errors.ServerError if the server responded with 5xx status

delete()

Delete custom inference model.

New in version v2.21.

Raises

datarobot.errors.ClientError if the server responded with 4xx status
datarobot.errors.ServerError if the server responded with 5xx status

assign_training_data(dataset_id, partition_column=None, max_wait=600)

Assign training data to the custom inference model.

New in version v2.21.

Parameters

dataset_id: str the id of the training dataset to be assigned
partition_column: str, optional name of a partition column in the training dataset
max_wait: int, optional max time to wait for a training data assignment. If set to None - method will return without waiting. Defaults to 10 min

Raises

datarobot.errors.ClientError if the server responded with 4xx status
datarobot.errors.ServerError if the server responded with 5xx status

class datarobot.CustomModelTest(kwargs)**

An custom model test.

New in version v2.21.

Attributes

id: str test id
dataset_id: str id of a dataset used for testing
dataset_version_id: str id of a dataset version used for testing
custom_model_image_id: str id of a custom model image

image_type: str the type of the image, either CUSTOM_MODEL_IMAGE_TYPE.CUSTOM_MODEL_IMAGE if the testing attempt is using a CustomModelImage as its model or CUSTOM_MODEL_IMAGE_TYPE.CUSTOM_MODEL_VERSION if the testing attempt is using a CustomModelVersion with dependency management

overall_status: str a string representing testing status. Status can be - 'not_tested': the check not run - 'failed': the check failed - 'succeeded': the check succeeded - 'warning': the check resulted in a warning, or in non-critical failure - 'in_progress': the check is in progress

detailed_status: dict detailed testing status - maps the testing types to their status and message. The keys of the dict are one of 'errorCheck', 'nullValueImputation', 'longRunningService', 'sideEffects'. The values are dict with 'message' and 'status' keys.

created_by: str a user who created a test

completed_at: str, optional ISO-8601 formatted timestamp of when the test has completed

created_at: str, optional ISO-8601 formatted timestamp of when the version was created

classmethod create (*custom_model_id, custom_model_version_id, dataset_id, environment_id=None, environment_version_id=None, max_wait=600*)

Create and start a custom model test.

New in version v2.21.

Parameters

custom_model_id: str the id of the custom model

custom_model_version_id: str the id of the custom model version

dataset_id: str the id of the testing dataset

environment_id: str, optional the id of the execution environment. If specified, the environment will be used as is; if the custom model version has dependencies, they will not be installed at runtime.

environment_version_id: str, optional the id of the execution environment version

max_wait: int, optional max time to wait for a test completion. If set to None - method will return without waiting.

Returns

CustomModelTest created custom model test

Raises

datarobot.errors.ClientError if the server responded with 4xx status

datarobot.errors.ServerError if the server responded with 5xx status

classmethod list (*custom_model_id*)

List custom model tests.

New in version v2.21.

Parameters

custom_model_id: str the id of the custom model

Returns

List[CustomModelTest] a list of custom model tests

Raises

datarobot.errors.ClientError if the server responded with 4xx status

datarobot.errors.ServerError if the server responded with 5xx status

classmethod **get** (*custom_model_test_id*)

Get custom model test by id.

New in version v2.21.

Parameters

custom_model_test_id: str the id of the custom model test

Returns

CustomModelTest retrieved custom model test

Raises

datarobot.errors.ClientError if the server responded with 4xx status.

datarobot.errors.ServerError if the server responded with 5xx status.

get_log ()

Get log of a custom model test.

New in version v2.21.

Raises

datarobot.errors.ClientError if the server responded with 4xx status

datarobot.errors.ServerError if the server responded with 5xx status

get_log_tail ()

Get log tail of a custom model test.

New in version v2.21.

Raises

datarobot.errors.ClientError if the server responded with 4xx status

datarobot.errors.ServerError if the server responded with 5xx status

cancel ()

Cancel custom model test that is in progress.

New in version v2.21.

Raises

datarobot.errors.ClientError if the server responded with 4xx status

datarobot.errors.ServerError if the server responded with 5xx status

refresh ()

Update custom model test with the latest data from server.

New in version v2.21.

Raises

datarobot.errors.ClientError if the server responded with 4xx status

datarobot.errors.ServerError if the server responded with 5xx status

class `datarobot.CustomModelVersion` (**kwargs)

A version of a DataRobot custom model.

New in version v2.21.

Attributes

id: `str` id of the custom model version

custom_model_id: `str` id of the custom model

version_minor: `int` a minor version number of custom model version

version_major: `int` a major version number of custom model version

is_frozen: `bool` a flag if the custom model version is frozen

items: `List[CustomModelFileItem]` a list of file items attached to the custom model version

base_environment_id: `str` id of the environment to use with the model

label: `str`, **optional** short human readable string to label the version

description: `str`, **optional** custom model version description

created_at: `str`, **optional** ISO-8601 formatted timestamp of when the version was created

dependencies: `List[CustomDependency]` the parsed dependencies of the custom model version if the version has a valid requirements.txt file

classmethod `create_clean` (*custom_model_id*, *base_environment_id*, *is_major_update=True*, *folder_path=None*, *files=None*)

Create a custom model version without files from previous versions.

New in version v2.21.

Parameters

custom_model_id: `str` the id of the custom model

base_environment_id: `str` the id of the base environment to use with the custom model version

is_major_update: `bool` the flag defining if a custom model version will be a minor or a major version. Default to *True*

folder_path: `str`, **optional** the path to a folder containing files to be uploaded. Each file in the folder is uploaded under path relative to a folder path

files: `list`, **optional** the list of tuples, where values in each tuple are the local filesystem path and the path the file should be placed in the model. if list is of strings, then basenames will be used for tuples Example: [(*âĀĀI*/home/user/Documents/myModel/file1.txtâĀĀI, *âĀĀI*file1.txtâĀĀI), (*âĀĀI*/home/user/Documents/myModel/folder/file2.txtâĀĀI, *âĀĀI*folder/file2.txtâĀĀI)] or [*âĀĀI*/home/user/Documents/myModel/file1.txtâĀĀI, *âĀĀI*/home/user/Documents/myModel/folder/file2.txtâĀĀI]

Returns

CustomModelVersion created custom model version

Raises

datarobot.errors.ClientError if the server responded with 4xx status

datarobot.errors.ServerError if the server responded with 5xx status

```
classmethod create_from_previous (custom_model_id, base_environment_id,  
                                is_major_update=True, folder_path=None, files=None,  
                                files_to_delete=None)
```

Create a custom model version containing files from a previous version.

New in version v2.21.

Parameters

custom_model_id: str the id of the custom model

base_environment_id: str the id of the base environment to use with the custom model version

is_major_update: bool, optional the flag defining if a custom model version will be a minor or a major version. Default to *True*

folder_path: str, optional the path to a folder containing files to be uploaded. Each file in the folder is uploaded under path relative to a folder path

files: list, optional the list of tuples, where values in each tuple are the local filesystem path and the path the file should be placed in the model. if list is of strings, then basenames will be used for tuples Example: [(*âĀĬI/home/user/Documents/myModel/file1.txtâĀĬI*, *âĀĬIfile1.txtâĀĬI*), (*âĀĬI/home/user/Documents/myModel/folder/file2.txtâĀĬI*, *âĀĬIfolder/file2.txtâĀĬI*)] or [*âĀĬI/home/user/Documents/myModel/file1.txtâĀĬI*, *âĀĬI/home/user/Documents/myModel/folder/file2.txtâĀĬI*]

files_to_delete: list, optional the list of a file items ids to be deleted Example: [*âĀĬI5ea95f7a4024030aba48e4f9âĀĬI*, *âĀĬI5ea6b5da402403181895cc51âĀĬI*]

Returns

CustomModelVersion created custom model version

Raises

datarobot.errors.ClientError if the server responded with 4xx status

datarobot.errors.ServerError if the server responded with 5xx status

```
classmethod list (custom_model_id)
```

List custom model versions.

New in version v2.21.

Parameters

custom_model_id: str the id of the custom model

Returns

List[CustomModelVersion] a list of custom model versions

Raises

datarobot.errors.ClientError if the server responded with 4xx status

datarobot.errors.ServerError if the server responded with 5xx status

```
classmethod get (custom_model_id, custom_model_version_id)
```

Get custom model version by id.

New in version v2.21.

Parameters

custom_model_id: str the id of the custom model

custom_model_version_id: str the id of the custom model version to retrieve

Returns

CustomModelVersion retrieved custom model version

Raises

datarobot.errors.ClientError if the server responded with 4xx status.

datarobot.errors.ServerError if the server responded with 5xx status.

download (*file_path*)

Download custom model version.

New in version v2.21.

Parameters

file_path: str path to create a file with custom model version content

Raises

datarobot.errors.ClientError if the server responded with 4xx status.

datarobot.errors.ServerError if the server responded with 5xx status.

update (*description*)

Update custom model version properties.

New in version v2.21.

Parameters

description: str new custom model version description

Raises

datarobot.errors.ClientError if the server responded with 4xx status.

datarobot.errors.ServerError if the server responded with 5xx status.

refresh ()

Update custom model version with the latest data from server.

New in version v2.21.

Raises

datarobot.errors.ClientError if the server responded with 4xx status

datarobot.errors.ServerError if the server responded with 5xx status

class `datarobot.CustomModelVersionDependencyBuild` (***kwargs*)

Metadata about a DataRobot custom model version's dependency build

New in version v2.22.

Attributes

custom_model_id: str id of the custom model

custom_model_version_id: str id of the custom model version

build_status: str the status of the custom model version's dependency build

started_at: str ISO-8601 formatted timestamp of when the build was started

completed_at: str, optional ISO-8601 formatted timestamp of when the build has completed

classmethod `get_build_info (custom_model_id, custom_model_version_id)`

Retrieve information about a custom model version's dependency build

New in version v2.22.

Parameters

custom_model_id: str the id of the custom model

custom_model_version_id: str the id of the custom model version

Returns

CustomModelVersionDependencyBuild the dependency build information

classmethod `start_build (custom_model_id, custom_model_version_id, max_wait=600)`

Start the dependency build for a custom model version dependency build

New in version v2.22.

Parameters

custom_model_id: str the id of the custom model

custom_model_version_id: str the id of the custom model version

max_wait: int, optional max time to wait for a build completion. If set to None - method will return without waiting.

get_log ()

Get log of a custom model version dependency build.

New in version v2.22.

Raises

datarobot.errors.ClientError if the server responded with 4xx status

datarobot.errors.ServerError if the server responded with 5xx status

cancel ()

Cancel custom model version dependency build that is in progress.

New in version v2.22.

Raises

datarobot.errors.ClientError if the server responded with 4xx status

datarobot.errors.ServerError if the server responded with 5xx status

refresh ()

Update custom model version dependency build with the latest data from server.

New in version v2.22.

Raises

datarobot.errors.ClientError if the server responded with 4xx status

datarobot.errors.ServerError if the server responded with 5xx status

class `datarobot.ExecutionEnvironment (**kwargs)`

An execution environment entity.

New in version v2.21.

Attributes

id: **str** the id of the execution environment

name: **str** the name of the execution environment

description: **str, optional** the description of the execution environment

programming_language: **str, optional** the programming language of the execution environment. Can be `python`, `jr`, `java` or `other`

is_public: **bool, optional** public accessibility of environment, visible only for admin user

created_at: **str, optional** ISO-8601 formatted timestamp of when the execution environment version was created

latest_version: **ExecutionEnvironmentVersion, optional** the latest version of the execution environment

classmethod create (*name, description=None, programming_language=None*)
Create an execution environment.

New in version v2.21.

Parameters

name: **str** execution environment name

description: **str, optional** execution environment description

programming_language: **str, optional** programming language of the environment to be created. Can be `python`, `jr`, `java` or `other`. Default value - `other`

Returns

ExecutionEnvironment created execution environment

Raises

datarobot.errors.ClientError if the server responded with 4xx status

datarobot.errors.ServerError if the server responded with 5xx status

classmethod list (*search_for=None*)
List execution environments available to the user.

New in version v2.21.

Parameters

search_for: **str, optional** the string for filtering execution environment - only execution environments that contain the string in name or description will be returned.

Returns

List[ExecutionEnvironment] a list of execution environments.

Raises

datarobot.errors.ClientError if the server responded with 4xx status

datarobot.errors.ServerError if the server responded with 5xx status

classmethod get (*execution_environment_id*)
Get execution environment by its id.

New in version v2.21.

Parameters

execution_environment_id: str ID of the execution environment to retrieve

Returns

ExecutionEnvironment retrieved execution environment

Raises

datarobot.errors.ClientError if the server responded with 4xx status

datarobot.errors.ServerError if the server responded with 5xx status

delete()

Delete execution environment.

New in version v2.21.

Raises

datarobot.errors.ClientError if the server responded with 4xx status

datarobot.errors.ServerError if the server responded with 5xx status

update(name=None, description=None)

Update execution environment properties.

New in version v2.21.

Parameters

name: str, optional new execution environment name

description: str, optional new execution environment description

Raises

datarobot.errors.ClientError if the server responded with 4xx status

datarobot.errors.ServerError if the server responded with 5xx status

refresh()

Update execution environment with the latest data from server.

New in version v2.21.

Raises

datarobot.errors.ClientError if the server responded with 4xx status

datarobot.errors.ServerError if the server responded with 5xx status

class datarobot.ExecutionEnvironmentVersion(kwargs)**

A version of a DataRobot execution environment.

New in version v2.21.

Attributes

id: str the id of the execution environment version

environment_id: str the id of the execution environment the version belongs to

build_status: str the status of the execution environment version build

label: str, optional the label of the execution environment version

description: str, optional the description of the execution environment version

created_at: str, optional ISO-8601 formatted timestamp of when the execution environment version was created

classmethod create (*execution_environment_id*, *docker_context_path*, *label=None*, *description=None*, *max_wait=600*)

Create an execution environment version.

New in version v2.21.

Parameters

- execution_environment_id: str** the id of the execution environment
- docker_context_path: str** the path to a docker context archive or folder
- label: str, optional** short human readable string to label the version
- description: str, optional** execution environment version description
- max_wait: int, optional** max time to wait for a final build status (â€œsuccessâ€œ or â€œfailedâ€œ). If set to None - method will return without waiting.

Returns

ExecutionEnvironmentVersion created execution environment version

Raises

- datarobot.errors.AsyncTimeoutError** if version did not reach final state during timeout seconds
- datarobot.errors.ClientError** if the server responded with 4xx status
- datarobot.errors.ServerError** if the server responded with 5xx status

classmethod list (*execution_environment_id*, *build_status=None*)

List execution environment versions available to the user.

New in version v2.21.

Parameters

- execution_environment_id: str** the id of the execution environment
- build_status: str, optional** build status of the execution environment version to filter by. See `datarobot.enums.EXECUTION_ENVIRONMENT_VERSION_BUILD_STATUS` for valid options

Returns

List[ExecutionEnvironmentVersion] a list of execution environment versions.

Raises

- datarobot.errors.ClientError** if the server responded with 4xx status
- datarobot.errors.ServerError** if the server responded with 5xx status

classmethod get (*execution_environment_id*, *version_id*)

Get execution environment version by id.

New in version v2.21.

Parameters

- execution_environment_id: str** the id of the execution environment
- version_id: str** the id of the execution environment version to retrieve

Returns

ExecutionEnvironmentVersion retrieved execution environment version

Raises

datarobot.errors.ClientError if the server responded with 4xx status.

datarobot.errors.ServerError if the server responded with 5xx status.

download (*file_path*)

Download execution environment version.

New in version v2.21.

Parameters

file_path: str path to create a file with execution environment version content

Returns

ExecutionEnvironmentVersion retrieved execution environment version

Raises

datarobot.errors.ClientError if the server responded with 4xx status.

datarobot.errors.ServerError if the server responded with 5xx status.

get_build_log ()

Get execution environment version build log and error.

New in version v2.21.

Returns

Tuple[str, str] retrieved execution environment version build log and error. If there is no build error - None is returned.

Raises

datarobot.errors.ClientError if the server responded with 4xx status.

datarobot.errors.ServerError if the server responded with 5xx status.

refresh ()

Update execution environment version with the latest data from server.

New in version v2.21.

Raises

datarobot.errors.ClientError if the server responded with 4xx status

datarobot.errors.ServerError if the server responded with 5xx status

32.10 Database Connectivity

```
class datarobot.DataDriver (id=None, creator=None, base_names=None, class_name=None,  
                           canonical_name=None)
```

A data driver

Attributes

id [str] the id of the driver.

class_name [str] the Java class name for the driver.

canonical_name [str] the user-friendly name of the driver.

creator [str] the id of the user who created the driver.

base_names [list of str] a list of the file name(s) of the jar files.

classmethod list()

Returns list of available drivers.

Returns

drivers [list of DataDriver instances] contains a list of available drivers.

Examples

```
>>> import datarobot as dr
>>> drivers = dr.DataDriver.list()
>>> drivers
[DataDriver('mysql'), DataDriver('RedShift'), DataDriver('PostgreSQL')]
```

classmethod get(driver_id)

Gets the driver.

Parameters

driver_id [str] the identifier of the driver.

Returns

driver [DataDriver] the required driver.

Examples

```
>>> import datarobot as dr
>>> driver = dr.DataDriver.get('5ad08a1889453d0001ea7c5c')
>>> driver
DataDriver('PostgreSQL')
```

classmethod create(class_name, canonical_name, files)

Creates the driver. Only available to admin users.

Parameters

class_name [str] the Java class name for the driver.

canonical_name [str] the user-friendly name of the driver.

files [list of str] a list of the file paths on file system file_path(s) for the driver.

Returns

driver [DataDriver] the created driver.

Raises

ClientError raised if user is not granted for *Can manage JDBC database drivers* feature

Examples

```
>>> import datarobot as dr
>>> driver = dr.DataDriver.create(
...     class_name='org.postgresql.Driver',
...     canonical_name='PostgreSQL',
...     files=['/tmp/postgresql-42.2.2.jar']
... )
>>> driver
DataDriver('PostgreSQL')
```

update (*class_name=None, canonical_name=None*)

Updates the driver. Only available to admin users.

Parameters

class_name [str] the Java class name for the driver.

canonical_name [str] the user-friendly name of the driver.

Raises

ClientError raised if user is not granted for *Can manage JDBC database drivers* feature

Examples

```
>>> import datarobot as dr
>>> driver = dr.DataDriver.get('5ad08a1889453d0001ea7c5c')
>>> driver.canonical_name
'PostgreSQL'
>>> driver.update(canonical_name='postgres')
>>> driver.canonical_name
'postgres'
```

delete ()

Removes the driver. Only available to admin users.

Raises

ClientError raised if user is not granted for *Can manage JDBC database drivers* feature

class `datarobot.DataStore` (*data_store_id=None, data_store_type=None, canonical_name=None, creator=None, updated=None, params=None, role=None*)

A data store. Represents database

Attributes

id [str] the id of the data store.

data_store_type [str] the type of data store.

canonical_name [str] the user-friendly name of the data store.

creator [str] the id of the user who created the data store.

updated [datetime.datetime] the time of the last update

params [DataStoreParameters] a list specifying data store parameters.

classmethod `list` ()

Returns list of available data stores.

Returns

data_stores [list of DataStore instances] contains a list of available data stores.

Examples

```
>>> import datarobot as dr
>>> data_stores = dr.DataStore.list()
>>> data_stores
[DataStore('Demo'), DataStore('Airlines')]
```

classmethod `get(data_store_id)`

Gets the data store.

Parameters

data_store_id [str] the identifier of the data store.

Returns

data_store [DataStore] the required data store.

Examples

```
>>> import datarobot as dr
>>> data_store = dr.DataStore.get('5a8ac90b07a57a0001be501e')
>>> data_store
DataStore('Demo')
```

classmethod `create(data_store_type, canonical_name, driver_id, jdbc_url)`

Creates the data store.

Parameters

data_store_type [str] the type of data store.

canonical_name [str] the user-friendly name of the data store.

driver_id [str] the identifier of the DataDriver.

jdbc_url [str] the full JDBC url, for example `jdbc:postgresql://my.dbaddress.org:5432/my_db`.

Returns

data_store [DataStore] the created data store.

Examples

```
>>> import datarobot as dr
>>> data_store = dr.DataStore.create(
...     data_store_type='jdbc',
...     canonical_name='Demo DB',
...     driver_id='5a6af02eb15372000117c040',
...     jdbc_url='jdbc:postgresql://my.db.address.org:5432/perftest'
... )
>>> data_store
DataStore('Demo DB')
```

update (`canonical_name=None, driver_id=None, jdbc_url=None`)

Updates the data store.

Parameters

canonical_name [str] optional, the user-friendly name of the data store.

driver_id [str] optional, the identifier of the DataDriver.

jdbc_url [str] optional, the full JDBC url, for example
jdbc:postgresql://my.dbaddress.org:5432/my_db.

Examples

```
>>> import datarobot as dr
>>> data_store = dr.DataStore.get('5ad5d2afef5cd700014d3cae')
>>> data_store
DataStore('Demo DB')
>>> data_store.update(canonical_name='Demo DB updated')
>>> data_store
DataStore('Demo DB updated')
```

delete()

Removes the DataStore

test (username, password)

Tests database connection.

Parameters

username [str] the username for database authentication.

password [str] the password for database authentication. The password is encrypted at server side and never saved / stored

Returns

message [dict] message with status.

Examples

```
>>> import datarobot as dr
>>> data_store = dr.DataStore.get('5ad5d2afef5cd700014d3cae')
>>> data_store.test(username='db_username', password='db_password')
{'message': 'Connection successful'}
```

schemas (username, password)

Returns list of available schemas.

Parameters

username [str] the username for database authentication.

password [str] the password for database authentication. The password is encrypted at server side and never saved / stored

Returns

response [dict] dict with database name and list of str - available schemas

Examples

```
>>> import datarobot as dr
>>> data_store = dr.DataStore.get('5ad5d2afef5cd700014d3cae')
>>> data_store.schemas(username='db_username', password='db_password')
{'catalog': 'perfctest', 'schemas': ['demo', 'information_schema', 'public']}
```

tables (*username, password, schema=None*)

Returns list of available tables in schema.

Parameters

username [str] optional, the username for database authentication.

password [str] optional, the password for database authentication. The password is encrypted at server side and never saved / stored

schema [str] optional, the schema name.

Returns

response [dict] dict with catalog name and tables info

Examples

```
>>> import datarobot as dr
>>> data_store = dr.DataStore.get('5ad5d2afef5cd700014d3cae')
>>> data_store.tables(username='db_username', password='db_password', schema=
↪ 'demo')
{'tables': [{'type': 'TABLE', 'name': 'diagnosis', 'schema': 'demo'}, {'type':
↪ 'TABLE',
'name': 'kickcars', 'schema': 'demo'}, {'type': 'TABLE', 'name': 'patient',
'schema': 'demo'}, {'type': 'TABLE', 'name': 'transcript', 'schema': 'demo'}],
'catalog': 'perfctest'}
```

classmethod from_server_data (*data, keep_attrs=None*)

Instantiate an object of this class using the data directly from the server, meaning that the keys may have the wrong camel casing

Parameters

data [dict] The directly translated dict of JSON from the server. No casing fixes have taken place

keep_attrs [list] List of the dotted namespace notations for attributes to keep within the object structure even if their values are None

get_access_list ()

Retrieve what users have access to this data store

New in version v2.14.

Returns

list of *SharingAccess*

share (*access_list*)

Modify the ability of users to access this data store

New in version v2.14.

Parameters

access_list [list of *SharingAccess*] the modifications to make.

Raises

datarobot.ClientError : if you do not have permission to share this data store, if the user youâ€™re sharing with doesnâ€™t exist, if the same user appears multiple times in the `access_list`, or if these changes would leave the data store without an owner.

Examples

Transfer access to the data store from `old_user@datarobot.com` to `new_user@datarobot.com`

```
import datarobot as dr

new_access = dr.SharingAccess(new_user@datarobot.com,
                              dr.enums.SHARING_ROLE.OWNER, can_share=True)
access_list = [dr.SharingAccess(old_user@datarobot.com, None), new_access]

dr.DataStore.get('my-data-store-id').share(access_list)
```

```
class datarobot.DataSource(data_source_id=None, data_source_type=None, canonical_name=None, creator=None, updated=None, params=None, role=None)
```

A data source. Represents data request

Attributes

id [str] the id of the data source.

type [str] the type of data source.

canonical_name [str] the user-friendly name of the data source.

creator [str] the id of the user who created the data source.

updated [datetime.datetime] the time of the last update.

params [DataSourceParameters] a list specifying data source parameters.

classmethod list()

Returns list of available data sources.

Returns

data_sources [list of DataSource instances] contains a list of available data sources.

Examples

```
>>> import datarobot as dr
>>> data_sources = dr.DataSource.list()
>>> data_sources
[DataSource('Diagnostics'), DataSource('Airlines 100mb'), DataSource(
  ↪ 'Airlines 10mb')]
```

classmethod get(data_source_id)

Gets the data source.

Parameters

data_source_id [str] the identifier of the data source.

Returns

data_source [DataSource] the requested data source.

Examples

```
>>> import datarobot as dr
>>> data_source = dr.DataSource.get('5a8ac9ab07a57a0001be501f')
>>> data_source
DataSource('Diagnostics')
```

classmethod create (*data_source_type, canonical_name, params*)

Creates the data source.

Parameters

data_source_type [str] the type of data source.

canonical_name [str] the user-friendly name of the data source.

params [DataSourceParameters] a list specifying data source parameters.

Returns

data_source [DataSource] the created data source.

Examples

```
>>> import datarobot as dr
>>> params = dr.DataSourceParameters(
...     data_store_id='5a8ac90b07a57a0001be501e',
...     query='SELECT * FROM airlines10mb WHERE "Year" >= 1995;'
... )
>>> data_source = dr.DataSource.create(
...     data_source_type='jdbc',
...     canonical_name='airlines stats after 1995',
...     params=params
... )
>>> data_source
DataSource('airlines stats after 1995')
```

update (*canonical_name=None, params=None*)

Creates the data source.

Parameters

canonical_name [str] optional, the user-friendly name of the data source.

params [DataSourceParameters] optional, the identifier of the DataDriver.

Examples

```
>>> import datarobot as dr
>>> data_source = dr.DataSource.get('5ad840cc613b480001570953')
>>> data_source
DataSource('airlines stats after 1995')
>>> params = dr.DataSourceParameters(
...     query='SELECT * FROM airlines10mb WHERE "Year" >= 1990;'
... )
```

(continues on next page)

(continued from previous page)

```

... )
>>> data_source.update(
...     canonical_name='airlines stats after 1990',
...     params=params
... )
>>> data_source
DataSource('airlines stats after 1990')

```

delete()

Removes the DataSource

classmethod from_server_data (*data*, *keep_attrs=None*)

Instantiate an object of this class using the data directly from the server, meaning that the keys may have the wrong camel casing

Parameters

data [dict] The directly translated dict of JSON from the server. No casing fixes have taken place

keep_attrs [list] List of the dotted namespace notations for attributes to keep within the object structure even if their values are None

get_access_list()

Retrieve what users have access to this data source

New in version v2.14.

Returnslist of *SharingAccess***share** (*access_list*)

Modify the ability of users to access this data source

New in version v2.14.

Parameters

access_list [list of *SharingAccess*] the modifications to make.

Raises

datarobot.ClientError : if you do not have permission to share this data source, if the user youâ€™re sharing with doesnâ€™t exist, if the same user appears multiple times in the access_list, or if these changes would leave the data source without an owner

Examples

Transfer access to the data source from `old_user@datarobot.com` to `new_user@datarobot.com`

```

import datarobot as dr

new_access = dr.SharingAccess(new_user@datarobot.com,
                              dr.enums.SHARING_ROLE.OWNER, can_share=True)
access_list = [dr.SharingAccess(old_user@datarobot.com, None), new_access]

dr.DataSource.get('my-data-source-id').share(access_list)

```

```
create_dataset (username=None, password=None, do_snapshot=None, persist_data_after_ingestion=None, categories=None, credential_id=None, use_kerberos=None)
```

Create a *Dataset* from this data source.

New in version v2.22.

Parameters

username: **string, optional** The username for database authentication.

password: **string, optional** The password (in cleartext) for database authentication. The password will be encrypted on the server side in scope of HTTP request and never saved or stored.

do_snapshot: **bool, optional** If unset, uses the server default: True. If true, creates a snapshot dataset; if false, creates a remote dataset. Creating snapshots from non-file sources requires an additional permission, *Enable Create Snapshot Data Source*.

persist_data_after_ingestion: **bool, optional** If unset, uses the server default: True. If true, will enforce saving all data (for download and sampling) and will allow a user to view extended data profile (which includes data statistics like min/max/median/mean, histogram, etc.). If false, will not enforce saving data. The data schema (feature names and types) still will be available. Specifying this parameter to false and *doSnapshot* to true will result in an error.

categories: **list[string], optional** An array of strings describing the intended use of the dataset. The current supported options are "TRAINING" and "PREDICTION".

credential_id: **string, optional** The ID of the set of credentials to use instead of user and password. Note that with this change, username and password will become optional.

use_kerberos: **bool, optional** If unset, uses the server default: False. If true, use kerberos authentication for database authentication.

Returns

response: **Dataset** The Dataset created from the uploaded data

```
class datarobot.DataSourceParameters (data_store_id=None, table=None, schema=None, partition_column=None, query=None, fetch_size=None)
```

Data request configuration

Attributes

data_store_id [str] the id of the DataStore.

table [str] optional, the name of specified database table.

schema [str] optional, the name of the schema associated with the table.

partition_column [str] optional, the name of the partition column.

query [str] optional, the user specified SQL query.

fetch_size [int] optional, a user specified fetch size in the range [1, 20000]. By default a fetch-size will be assigned to balance throughput and memory usage

32.11 Datasets

```
class datarobot.Dataset (dataset_id, version_id, name, categories, created_at, created_by,  
                        is_data_engine_eligible, is_latest_version, is_snapshot, processing_state,  
                        data_persisted=None, size=None, row_count=None)
```

Represents a Dataset returned from the `api/v2/datasets/` endpoints.

Attributes

id: string The ID of this dataset

name: string The name of this dataset in the catalog

is_latest_version: bool Whether this dataset version is the latest version of this dataset

version_id: string The object ID of the catalog_version the dataset belongs to

categories: list(string) An array of strings describing the intended use of the dataset. The supported options are `TRAINING` and `PREDICTION`.

created_at: string The date when the dataset was created

created_by: string Username of the user who created the dataset

is_snapshot: bool Whether the dataset version is an immutable snapshot of data which has previously been retrieved and saved to Data_robot

data_persisted: bool, optional If true, user is allowed to view extended data profile (which includes data statistics like min/max/median/mean, histogram, etc.) and download data. If false, download is not allowed and only the data schema (feature names and types) will be available.

is_data_engine_eligible: bool Whether this dataset can be a data source of a data engine query.

processing_state: string Current ingestion process state of the dataset

row_count: int, optional The number of rows in the dataset.

size: int, optional The size of the dataset as a CSV in bytes.

```
classmethod create_from_file (file_path=None, filelike=None, categories=None,  
                             read_timeout=600, max_wait=600)
```

A blocking call that creates a new Dataset from a file. Returns when the dataset has been successfully uploaded and processed.

Warning: This function does not clean up its open files. If you pass a filelike, you are responsible for closing it. If you pass a file_path, this will create a file object from the file_path but will not close it.

Parameters

file_path: string, optional The path to the file. This will create a file object pointing to that file but will not close it.

filelike: file, optional An open and readable file object.

categories: list[string], optional An array of strings describing the intended use of the dataset. The current supported options are `TRAINING` and `PREDICTION`.

read_timeout: int, optional The maximum number of seconds to wait for the server to respond indicating that the initial upload is complete

max_wait [int, optional] Time in seconds after which project creation is considered unsuccessful

Returns

response: Dataset A fully armed and operational Dataset

classmethod create_from_in_memory_data (*data_frame=None, records=None, categories=None*)

A blocking call that creates a new Dataset from in-memory data. Returns when the dataset has been successfully uploaded and processed.

The data can be either a pandas DataFrame or a list of dictionaries with identical keys.

Parameters

data_frame: DataFrame, optional The data frame to upload

records: list[dict], optional A list of dictionaries with identical keys to upload

categories: list[string], optional An array of strings describing the intended use of the dataset. The current supported options are `â€œIJTRAININGâ€œ` and `â€œIJPREDICTIONâ€œ`.

Returns

response: Dataset The Dataset created from the uploaded data

classmethod create_from_url (*url, do_snapshot=None, persist_data_after_ingestion=None, categories=None*)

A blocking call that creates a new Dataset from data stored at a url. Returns when the dataset has been successfully uploaded and processed.

Parameters

url: string The URL to use as the source of data for the dataset being created.

do_snapshot: bool, optional If unset, uses the server default: True. If true, creates a snapshot dataset; if false, creates a remote dataset. Creating snapshots from non-file sources requires an additional permission, *Enable Create Snapshot Data Source*.

persist_data_after_ingestion: bool, optional If unset, uses the server default: True. If true, will enforce saving all data (for download and sampling) and will allow a user to view extended data profile (which includes data statistics like min/max/median/mean, histogram, etc.). If false, will not enforce saving data. The data schema (feature names and types) still will be available. Specifying this parameter to false and *doSnapshot* to true will result in an error.

categories: list[string], optional An array of strings describing the intended use of the dataset. The current supported options are `â€œIJTRAININGâ€œ` and `â€œIJPREDICTIONâ€œ`.

Returns

response: Dataset The Dataset created from the uploaded data

classmethod create_from_data_source (*data_source_id, username=None, password=None, do_snapshot=None, persist_data_after_ingestion=None, categories=None, credential_id=None, use_kerberos=None*)

A blocking call that creates a new Dataset from data stored at a DataSource. Returns when the dataset has been successfully uploaded and processed.

New in version v2.22.

Parameters

data_source_id: string The ID of the DataSource to use as the source of data.

username: string, optional The username for database authentication.

password: string, optional The password (in cleartext) for database authentication. The password will be encrypted on the server side in scope of HTTP request and never saved or stored.

do_snapshot: bool, optional If unset, uses the server default: True. If true, creates a snapshot dataset; if false, creates a remote dataset. Creating snapshots from non-file sources requires an additional permission, *Enable Create Snapshot Data Source*.

persist_data_after_ingestion: bool, optional If unset, uses the server default: True. If true, will enforce saving all data (for download and sampling) and will allow a user to view extended data profile (which includes data statistics like min/max/median/mean, histogram, etc.). If false, will not enforce saving data. The data schema (feature names and types) still will be available. Specifying this parameter to false and *doSnapshot* to true will result in an error.

categories: list[string], optional An array of strings describing the intended use of the dataset. The current supported options are `TRAINING` and `PREDICTION`.

credential_id: string, optional The ID of the set of credentials to use instead of user and password. Note that with this change, username and password will become optional.

use_kerberos: bool, optional If unset, uses the server default: False. If true, use kerberos authentication for database authentication.

Returns

response: Dataset The Dataset created from the uploaded data

classmethod get (*dataset_id*)

Get information about a dataset.

Parameters

dataset_id [string] the id of the dataset

Returns

dataset [Dataset] the queried dataset

classmethod delete (*dataset_id*)

Soft deletes a dataset. You cannot get it or list it or do actions with it, except for un-deleting it.

Parameters

dataset_id: string The id of the dataset to mark for deletion

Returns

None

classmethod un_delete (*dataset_id*)

Un-deletes a previously deleted dataset. If the dataset was not deleted, nothing happens.

Parameters

dataset_id: string The id of the dataset to un-delete

Returns

None

classmethod list (*category=None, filter_failed=None, order_by=None*)

List all datasets a user can view.

Parameters

category: string, optional Optional. If specified, only dataset versions that have the specified category will be included in the results. Categories identify the intended use of the dataset; supported categories are `TRAINING` and `PREDICTION`.

filter_failed: bool, optional If unset, uses the server default: `False`. Whether datasets that failed during import should be excluded from the results. If `True` invalid datasets will be excluded.

order_by: string, optional If unset, uses the server default: `created`. Sorting order which will be applied to catalog list, valid options are: `-created` – ascending order by creation datetime; `-created` – descending order by creation datetime.

Returns

list[Dataset] a list of datasets the user can view

classmethod iterate (*offset=None, limit=None, category=None, order_by=None, filter_failed=None*)

Get an iterator for the requested datasets a user can view. This lazily retrieves results. It does not get the next page from the server until the current page is exhausted.

Parameters

offset: int, optional If set, this many results will be skipped

limit: int, optional Specifies the size of each page retrieved from the server. If unset, uses the server default.

category: string, optional Optional. If specified, only dataset versions that have the specified category will be included in the results. Categories identify the intended use of the dataset; supported categories are `TRAINING` and `PREDICTION`.

filter_failed: bool, optional If unset, uses the server default: `False`. Whether datasets that failed during import should be excluded from the results. If `True` invalid datasets will be excluded.

order_by: string, optional If unset, uses the server default: `created`. Sorting order which will be applied to catalog list, valid options are: `-created` – ascending order by creation datetime; `-created` – descending order by creation datetime.

Yields

Dataset An iterator of the datasets the user can view

update()

Updates the Dataset attributes in place with the latest information from the server.

Returns

None

modify (*name=None, categories=None*)

Modifies the Dataset name and/or categories. Updates the object in place.

Parameters

name: string, optional The new name of the dataset

categories: list[string], optional A list of strings describing the intended use of the dataset. The supported options are `TRAINING` and `PREDICTION`. If any categories were previously specified for the dataset, they will be overwritten.

Returns

None

get_details()

Gets the details for this Dataset

Returns

DatasetDetails

get_all_features (*order_by=None*)

Get a list of all the features for this dataset.

Parameters

order_by: string, optional If unset, uses the server default: `name`. How the features should be ordered. Can be `name` or `featureType`.

Returns

list[DatasetFeature]

iterate_all_features (*offset=None, limit=None, order_by=None*)

Get an iterator for the requested features of a dataset. This lazily retrieves results. It does not get the next page from the server until the current page is exhausted.

Parameters

offset: int, optional If set, this many results will be skipped.

limit: int, optional Specifies the size of each page retrieved from the server. If unset, uses the server default.

order_by: string, optional If unset, uses the server default: `name`. How the features should be ordered. Can be `name` or `featureType`.

Yields

DatasetFeature

get_featurelists()

Get DatasetFeaturelists created on this Dataset

Returns

feature_lists: list[DatasetFeaturelist]

create_featurelist (*name, features*)

Create a new dataset featurelist

Parameters

name [str] the name of the modeling featurelist to create. Names must be unique within the dataset, or the server will return an error.

features [list of str] the names of the features to include in the dataset featurelist. Each feature must be a dataset feature.

Returns

featurelist [DatasetFeaturelist] the newly created featurelist

Examples

```
dataset = Dataset.get('1234deadbeeffeeddead4321')
dataset_features = dataset.get_all_features()
selected_features = [feat.name for feat in dataset_features][:5] # select_
↳ first five
new_flist = dataset.create_featurelist('Simple Features', selected_features)
```

get_file (*file_path=None, filelike=None*)

Retrieves all the originally uploaded data in CSV form. Writes it to either the file or a filelike object that can write bytes.

Only one of `file_path` or `filelike` can be provided and it must be provided as a keyword argument (i.e. `file_path=path-to-write-to`). If a file-like object is provided, the user is responsible for closing it when they are done.

The user must also have permission to download data.

Parameters

file_path: string, optional The destination to write the file to.

filelike: file, optional A file-like object to write to. The object must be able to write bytes. The user is responsible for closing the object

Returns

None

get_projects ()

Retrieves the Dataset's projects as `ProjectLocation` named tuples.

Returns

locations: list[ProjectLocation]

create_project (*project_name=None, user=None, password=None, credential_id=None, use_kerberos=None*)

Create a `datarobot.models.Project` from this dataset

Parameters

project_name: string, optional The name of the project to be created. If not specified, will be "Untitled Project" for database connections, otherwise the project name will be based on the file used.

user: string, optional The username for database authentication.

password: string, optional The password (in cleartext) for database authentication. The password will be encrypted on the server side in scope of HTTP request and never saved or stored

credential_id: string, optional The ID of the set of credentials to use instead of user and password.

use_kerberos: bool, optional Server default is False. If true, use kerberos authentication for database authentication.

Returns

Project

```
class datarobot.DatasetDetails(dataset_id, version_id, categories, created_by, created_at,  
data_source_type, error, is_latest_version,  
is_snapshot, is_data_engine_eligible, last_modification_date,  
last_modifier_full_name, name, uri, data_persisted=None,  
data_engine_query_id=None, data_source_id=None,  
description=None, eda1_modification_date=None,  
eda1_modifier_full_name=None, feature_count=None,  
feature_count_by_type=None, processing_state=None,  
row_count=None, size=None, tags=None)
```

Represents a detailed view of a Dataset. The `to_dataset` method creates a Dataset from this details view.

Attributes

dataset_id: string The ID of this dataset

name: string The name of this dataset in the catalog

is_latest_version: bool Whether this dataset version is the latest version of this dataset

version_id: string The object ID of the catalog_version the dataset belongs to

categories: list(string) An array of strings describing the intended use of the dataset. The supported options are "TRAINING" and "PREDICTION".

created_at: string The date when the dataset was created

created_by: string Username of the user who created the dataset

is_snapshot: bool Whether the dataset version is an immutable snapshot of data which has previously been retrieved and saved to Data_robot

data_persisted: bool, optional If true, user is allowed to view extended data profile (which includes data statistics like min/max/median/mean, histogram, etc.) and download data. If false, download is not allowed and only the data schema (feature names and types) will be available.

is_data_engine_eligible: bool Whether this dataset can be a data source of a data engine query.

processing_state: string Current ingestion process state of the dataset

row_count: int, optional The number of rows in the dataset.

size: int, optional The size of the dataset as a CSV in bytes.

data_engine_query_id: string, optional ID of the source data engine query

data_source_id: string, optional ID of the datasource used as the source of the dataset

data_source_type: string the type of the datasource that was used as the source of the dataset

description: string, optional the description of the dataset

eda1_modification_date: string, optional the ISO 8601 formatted date and time when the EDA1 for the dataset was updated

eda1_modifier_full_name: string, optional the user who was the last to update EDA1 for the dataset

error: string details of exception raised during ingestion process, if any

feature_count: int, optional total number of features in the dataset

feature_count_by_type: list[FeatureTypeCount] number of features in the dataset grouped by feature type

last_modification_date: **string** the ISO 8601 formatted date and time when the dataset was last modified

last_modifier_full_name: **string** full name of user who was the last to modify the dataset

tags: **list[string]** list of tags attached to the item

uri: **string** the uri to datasource like: - `file_name.csv` - `jdbc:DATA_SOURCE_GIVEN_NAME/SCHEMA.TABLE_NAME` - `jdbc:DATA_SOURCE_GIVEN_NAME/<query>` - for *query* based datasources - `https://s3.amazonaws.com/datarobot_test/kickcars-sample-200.csv` - etc.

classmethod **get** (*dataset_id*)

Get details for a Dataset from the server

Parameters

dataset_id: **str** The id for the Dataset from which to get details

Returns

DatasetDetails

to_dataset ()

Build a Dataset object from the information in this object

Returns

Dataset

32.12 Deployment

```
class datarobot.Deployment (id=None, label=None, description=None, default_prediction_server=None, model=None, capabilities=None, prediction_usage=None, permissions=None, service_health=None, model_health=None, accuracy_health=None)
```

A deployment created from a DataRobot model.

Attributes

id [str] the id of the deployment

label [str] the label of the deployment

description [str] the description of the deployment

default_prediction_server [dict] information on the default prediction server of the deployment

model [dict] information on the model of the deployment

capabilities [dict] information on the capabilities of the deployment

prediction_usage [dict] information on the prediction usage of the deployment

permissions [list] (New in version v2.18) user's permissions on the deployment

service_health [dict] information on the service health of the deployment

model_health [dict] information on the model health of the deployment

accuracy_health [dict] information on the accuracy health of the deployment

```
classmethod create_from_learning_model(model_id, label, description=None, default_prediction_server_id=None)
```

Create a deployment from a DataRobot model.

New in version v2.17.

Parameters

model_id [str] id of the DataRobot model to deploy

label [str] a human readable label of the deployment

description [str, optional] a human readable description of the deployment

default_prediction_server_id [str, optional] an identifier of a prediction server to be used as the default prediction server

Returns

deployment [Deployment] The created deployment

Examples

```
from datarobot import Project, Deployment
project = Project.get('5506fcd38bd88f5953219da0')
model = project.get_models()[0]
deployment = Deployment.create_from_learning_model(model.id, 'New Deployment')
deployment
>>> Deployment('New Deployment')
```

```
classmethod create_from_custom_model_image(custom_model_image_id, label, description=None, default_prediction_server_id=None, max_wait=600)
```

Create a deployment from a DataRobot custom model image.

Parameters

custom_model_image_id [str] id of the DataRobot custom model image to deploy

label [str] a human readable label of the deployment

description [str, optional] a human readable description of the deployment

default_prediction_server_id [str, optional] an identifier of a prediction server to be used as the default prediction server

max_wait [int, optional] seconds to wait for successful resolution of a deployment creation job. Deployment supports making predictions only after a deployment creating job has successfully finished

Returns

deployment [Deployment] The created deployment

```
classmethod create_from_custom_model_version(custom_model_version_id, label, description=None, default_prediction_server_id=None, max_wait=600)
```

Create a deployment from a DataRobot custom model image.

Parameters

custom_model_version_id [str] id of the DataRobot custom model version to deploy The version must have a `base_environment_id`.

label [str] a human readable label of the deployment

description [str, optional] a human readable description of the deployment

default_prediction_server_id [str, optional] an identifier of a prediction server to be used as the default prediction server

max_wait [int, optional] seconds to wait for successful resolution of a deployment creation job. Deployment supports making predictions only after a deployment creating job has successfully finished

Returns

deployment [Deployment] The created deployment

classmethod list (*order_by=None, search=None, filters=None*)

List all deployments a user can view.

New in version v2.17.

Parameters

order_by [str, optional] (New in version v2.18) the order to sort the deployment list by, defaults to *label*

Allowed attributes to sort by are:

- `label`
- `serviceHealth`
- `modelHealth`
- `accuracyHealth`
- `recentPredictions`
- `lastPredictionTimestamp`

If the sort attribute is preceded by a hyphen, deployments will be sorted in descending order, otherwise in ascending order.

For health related sorting, ascending means failing, warning, passing, unknown.

search [str, optional] (New in version v2.18) case insensitive search against deployment's label and description.

filters [datarobot.models.deployment.DeploymentListFilters, optional] (New in version v2.20) an object containing all filters that you'd like to apply to the resulting list of deployments. See [DeploymentListFilters](#) for details on usage.

Returns

deployments [list] a list of deployments the user can view

Examples

```
from datarobot import Deployment
deployments = Deployment.list()
deployments
>>> [Deployment('New Deployment'), Deployment('Previous Deployment')]
```

```
from datarobot import Deployment
from datarobot.enums import DEPLOYMENT_SERVICE_HEALTH
filters = DeploymentListFilters(
    role='OWNER',
    service_health=[DEPLOYMENT_SERVICE_HEALTH.FAILING]
)
filtered_deployments = Deployment.list(filters=filters)
filtered_deployments
>>> [Deployment('Deployment I Own w/ Failing Service Health')]
```

classmethod `get(deployment_id)`
Get information about a deployment.

New in version v2.17.

Parameters

deployment_id [str] the id of the deployment

Returns

deployment [Deployment] the queried deployment

Examples

```
from datarobot import Deployment
deployment = Deployment.get(deployment_id='5c939e08962d741e34f609f0')
deployment.id
>>> '5c939e08962d741e34f609f0'
deployment.label
>>> 'New Deployment'
```

update (*label=None, description=None*)
Update the label and description of this deployment.

New in version v2.19.

delete ()
Delete this deployment.

New in version v2.17.

replace_model (*new_model_id, reason, max_wait=600*)

Replace the model used in this deployment. To confirm model replacement eligibility, use `validate_replacement_model()` beforehand.

New in version v2.17.

Model replacement is an asynchronous process, which means some preparatory work may be performed after the initial request is completed. This function will not return until all preparatory work is fully finished.

Predictions made against this deployment will start using the new model as soon as the initial request is completed. There will be no interruption for predictions throughout the process.

Parameters

new_model_id [str] The id of the new model to use

reason [MODEL_REPLACEMENT_REASON] The reason for the model replacement. Must be one of `ACCURACY`, `DATA_DRIFT`, `ERRORS`, `SCHEDULED_REFRESH`, `SCORING_SPEED`, or `OTHER`. This value will be stored in the model history to keep track of why a model was replaced

max_wait [int, optional] (new in version 2.22) The maximum time to wait for model replacement job to complete before erroring

Examples

```
from datarobot import Deployment
deployment = Deployment.get(deployment_id='5c939e08962d741e34f609f0')
deployment.model['id'], deployment.model['type']
>>> ('5c0a979859b00004ba52e431', 'Decision Tree Classifier (Gini)')

deployment.replace_model('5c0a969859b00004ba52e41b', MODEL_REPLACEMENT_REASON.
↳ ACCURACY)
deployment.model['id'], deployment.model['type']
>>> ('5c0a969859b00004ba52e41b', 'Support Vector Classifier (Linear Kernel)')
```

validate_replacement_model (*new_model_id*)

Validate a model can be used as the replacement model of the deployment.

New in version v2.17.

Parameters

new_model_id [str] the id of the new model to validate

Returns

status [str] status of the validation, will be one of `passing`, `warning` or `failing`. If the status is passing or warning, use `replace_model()` to perform a model replacement. If the status is failing, refer to `checks` for more detail on why the new model cannot be used as a replacement.

message [str] message for the validation result

checks [dict] explain why the new model can or cannot replace the deployment's current model

get_features ()

Retrieve the list of features needed to make predictions on this deployment.

Returns

features: list a list of *feature* dict

Notes

Each *feature* dict contains the following structure:

- **name** : str, feature name
- **feature_type** : str, feature type
- **importance** : float, numeric measure of the relationship strength between the feature and target (independent of model or other features)
- **date_format** : str or None, the date format string for how this feature was interpreted, null if not a date feature, compatible with <https://docs.python.org/2/library/time.html#time.strptime>.

- `known_in_advance` : bool, whether the feature was selected as known in advance in a time series model, false for non-time series models.

Examples

```
from datarobot import Deployment
deployment = Deployment.get(deployment_id='5c939e08962d741e34f609f0')
features = deployment.get_features()
features[0]['feature_type']
>>>'Categorical'
features[0]['importance']
>>>0.133
```

submit_actuals (*data*, *batch_size=10000*)

Submit actuals for processing. The actuals submitted will be used to calculate accuracy metrics.

Parameters

data: list or `pandas.DataFrame`

batch_size: the max number of actuals in each request

If ‘data’ is a list, each item should be a dict-like object with the following keys and values; if ‘data’ is a `pandas.DataFrame`, it should contain the following columns:

- **association_id:** str, a unique identifier used with a prediction, max length 128 characters
- **actual_value:** str or int or float, the actual value of a prediction; should be numeric for deployments with regression models or string for deployments with classification model
- **was_acted_on:** bool, optional, indicates if the prediction was acted on in a way that could have affected the actual outcome
- **timestamp:** datetime or string in RFC3339 format, optional. If the datetime provided does not have a timezone, we assume it is UTC.

Raises

ValueError if input data is not a list of dict-like objects or a `pandas.DataFrame` if input data is empty

Examples

```
from datarobot import Deployment, AccuracyOverTime
deployment = Deployment.get(deployment_id='5c939e08962d741e34f609f0')
data = [{
    'association_id': '439917',
    'actual_value': 'True',
    'was_acted_on': True
}]
deployment.submit_actuals(data)
```

get_drift_tracking_settings ()

Retrieve drift tracking settings of this deployment.

New in version v2.17.

Returns

settings [dict] Drift tracking settings of the deployment containing two nested dicts with key `target_drift` and `feature_drift`, which are further described below.

Target drift setting contains:

enabled [bool] If target drift tracking is enabled for this deployment. To create or update existing `target_drift` settings, see `update_drift_tracking_settings()`

Feature drift setting contains:

enabled [bool] If feature drift tracking is enabled for this deployment. To create or update existing `feature_drift` settings, see `update_drift_tracking_settings()`

update_drift_tracking_settings (*target_drift_enabled=None, feature_drift_enabled=None, max_wait=600*)

Update drift tracking settings of this deployment.

New in version v2.17.

Updating drift tracking setting is an asynchronous process, which means some preparatory work may be performed after the initial request is completed. This function will not return until all preparatory work is fully finished.

Parameters

target_drift_enabled [bool, optional] if target drift tracking is to be turned on

feature_drift_enabled [bool, optional] if feature drift tracking is to be turned on

max_wait [int, optional] seconds to wait for successful resolution

get_association_id_settings ()

Retrieve association ID setting for this deployment.

New in version v2.19.

Returns

association_id_settings [dict in the following format:]

column_names [list[string], optional] name of the columns to be used as association ID,

required_in_prediction_requests [bool, optional] whether the association ID column is required in prediction requests

update_association_id_settings (*column_names=None, required_in_prediction_requests=None, max_wait=600*) *re-*

Update association ID setting for this deployment.

New in version v2.19.

Parameters

column_names [list[string], optional] name of the columns to be used as association ID, currently only support a list of one string

required_in_prediction_requests [bool, optional] whether the association ID column is required in prediction requests

max_wait [int, optional] seconds to wait for successful resolution

get_predictions_data_collection_settings()

Retrieve predictions data collection settings of this deployment.

New in version v2.21.

Returns

predictions_data_collection_settings [dict in the following format:]

enabled [bool] If predictions data collection is enabled for this deployment. To update existing `predictions_data_collection` settings, see [`update_predictions_data_collection_settings\(\)`](#)

update_predictions_data_collection_settings(enabled, max_wait=600)

Update predictions data collection settings of this deployment.

New in version v2.21.

Updating predictions data collection setting is an asynchronous process, which means some preparatory work may be performed after the initial request is completed. This function will not return until all preparatory work is fully finished.

Parameters

enabled: bool if predictions data collection is to be turned on

max_wait [int, optional] seconds to wait for successful resolution

get_prediction_warning_settings()

Retrieve prediction warning settings of this deployment.

New in version v2.19.

Returns

settings [dict in the following format:]

enabled [bool] If target prediction_warning is enabled for this deployment. To create or update existing `prediction_warning` settings, see [`update_prediction_warning_settings\(\)`](#)

custom_boundaries [dict or None]

If None default boundaries for a model are used. Otherwise has following keys:

upper [float] All predictions greater than provided value are considered anomalous

lower [float] All predictions less than provided value are considered anomalous

update_prediction_warning_settings(prediction_warning_enabled, use_default_boundaries=None, lower_boundary=None, upper_boundary=None, max_wait=600)

Update prediction warning settings of this deployment.

New in version v2.19.

Parameters

prediction_warning_enabled [bool] If prediction warnings should be turned on.

use_default_boundaries [bool, optional] If default boundaries of the model should be used for the deployment.

upper_boundary [float, optional] All predictions greater than provided value will be considered anomalous

lower_boundary [float, optional] All predictions less than provided value will be considered anomalous

max_wait [int, optional] seconds to wait for successful resolution

get_prediction_intervals_settings ()

Retrieve prediction intervals settings for this deployment.

New in version v2.19.

Returns

dict in the following format:

enabled [bool] Whether prediction intervals are enabled for this deployment

percentiles [list[int]] List of enabled prediction intervals sizes for this deployment. Currently we only support one percentile at a time.

Notes

Note that prediction intervals are only supported for time series deployments.

update_prediction_intervals_settings (*percentiles*, *enabled=True*, *max_wait=600*)

Update prediction intervals settings for this deployment.

New in version v2.19.

Parameters

percentiles [list[int]] The prediction intervals percentiles to enable for this deployment. Currently we only support setting one percentile at a time.

enabled [bool, optional (defaults to True)] Whether to enable showing prediction intervals in the results of predictions requested using this deployment.

max_wait [int, optional] seconds to wait for successful resolution

Raises

AssertionError If *percentiles* is in an invalid format

AsyncFailureError If any of the responses from the server are unexpected

AsyncProcessUnsuccessfulError If the prediction intervals calculation job has failed or has been cancelled.

AsyncTimeoutError If the prediction intervals calculation job did not resolve in time

Notes

Updating prediction intervals settings is an asynchronous process, which means some preparatory work may be performed before the settings request is completed. This function will not return until all work is fully finished.

Note that prediction intervals are only supported for time series deployments.

get_service_stats (*model_id=None*, *start_time=None*, *end_time=None*, *execution_time_quantile=None*, *response_time_quantile=None*, *slow_requests_threshold=None*)

Retrieve value of service stat metrics over a certain time period.

New in version v2.18.

Parameters

model_id [str, optional] the id of the model

start_time [datetime, optional] start of the time period

end_time [datetime, optional] end of the time period

execution_time_quantile [float, optional] quantile for *executionTime*, defaults to 0.5

response_time_quantile [float, optional] quantile for *responseTime*, defaults to 0.5

slow_requests_threshold [float, optional] threshold for *slowRequests*, defaults to 1000

Returns

service_stats [ServiceStats] the queried service stats metrics information

get_service_stats_over_time (*metric=None, model_id=None, start_time=None, end_time=None, bucket_size=None, quantile=None, threshold=None*)

Retrieve information about how a service stat metric changes over a certain time period.

New in version v2.18.

Parameters

metric [SERVICE_STAT_METRIC, optional] the service stat metric to retrieve

model_id [str, optional] the id of the model

start_time [datetime, optional] start of the time period

end_time [datetime, optional] end of the time period

bucket_size [str, optional] time duration of a bucket, in ISO 8601 time duration format

quantile [float, optional] quantile for *executionTime* or *responseTime*, ignored when querying other metrics

threshold [int, optional] threshold for *slowQueries*, ignored when querying other metrics

Returns

service_stats_over_time [ServiceStatsOverTime] the queried service stats metric over time information

get_target_drift (*model_id=None, start_time=None, end_time=None, metric=None*)

Retrieve target drift information over a certain time period.

New in version v2.21.

Parameters

model_id [str] the id of the model

start_time [datetime] start of the time period

end_time [datetime] end of the time period

metric [str] (New in version v2.22) metric used to calculate the drift score

Returns

target_drift [TargetDrift] the queried target drift information

get_feature_drift (*model_id=None, start_time=None, end_time=None, metric=None*)

Retrieve drift information for deployment's features over a certain time period.

New in version v2.21.

Parameters

model_id [str] the id of the model

start_time [datetime] start of the time period

end_time [datetime] end of the time period

metric [str] (New in version v2.22) metric used to calculate the drift score

Returns

feature_drift_data [[FeatureDrift]] the queried feature drift information

get_accuracy (*model_id=None, start_time=None, end_time=None, start=None, end=None*)

Retrieve values of accuracy metrics over a certain time period.

New in version v2.18.

Parameters

model_id [str] the id of the model

start_time [datetime] start of the time period

end_time [datetime] end of the time period

Returns

accuracy [Accuracy] the queried accuracy metrics information

get_accuracy_over_time (*metric=None, model_id=None, start_time=None, end_time=None, bucket_size=None*)

Retrieve information about how an accuracy metric changes over a certain time period.

New in version v2.18.

Parameters

metric [ACCURACY_METRIC] the accuracy metric to retrieve

model_id [str] the id of the model

start_time [datetime] start of the time period

end_time [datetime] end of the time period

bucket_size [str] time duration of a bucket, in ISO 8601 time duration format

Returns

accuracy_over_time [AccuracyOverTime] the queried accuracy metric over time information

```
class datarobot.models.deployment.DeploymentListFilters (role=None, service_health=None, model_health=None, accuracy_health=None, execution_environment_type=None, importance=None)
```

Construct a set of filters to pass to `Deployment.list()`

New in version v2.20.

Parameters

role [str] A user role. If specified, then take those deployments that the user can view, then filter them down to those that the user has the specified role for, and return only them. Allowed options are OWNER and USER.

service_health [list of str] A list of service health status values. If specified, then only deployments whose service health status is one of these will be returned. See `datarobot.enums.DEPLOYMENT_SERVICE_HEALTH_STATUS` for allowed values. Supports comma-separated lists.

model_health [list of str] A list of model health status values. If specified, then only deployments whose model health status is one of these will be returned. See `datarobot.enums.DEPLOYMENT_MODEL_HEALTH_STATUS` for allowed values. Supports comma-separated lists.

accuracy_health [list of str] A list of accuracy health status values. If specified, then only deployments whose accuracy health status is one of these will be returned. See `datarobot.enums.DEPLOYMENT_ACCURACY_HEALTH_STATUS` for allowed values. Supports comma-separated lists.

execution_environment_type [list of str] A list of strings representing the type of the deployments' execution environment. If provided, then only return those deployments whose execution environment type is one of those provided. See `datarobot.enums.DEPLOYMENT_EXECUTION_ENVIRONMENT_TYPE` for allowed values. Supports comma-separated lists.

importance [list of str] A list of strings representing the deployments' importance. If provided, then only return those deployments whose importance is one of those provided. See `datarobot.enums.DEPLOYMENT_IMPORTANCE` for allowed values. Supports comma-separated lists. Note that Approval Workflows must be enabled for your account to use this filter, otherwise the API will return a 403.

Examples

Multiple filters can be combined in interesting ways to return very specific subsets of deployments.

Performing AND logic

Providing multiple different parameters will result in AND logic between them. For example, the following will return all deployments that I own whose service health status is failing.

```
from datarobot import Deployment
from datarobot.models.deployment import DeploymentListFilters
from datarobot.enums import DEPLOYMENT_SERVICE_HEALTH
filters = DeploymentListFilters(
    role='OWNER',
    service_health=[DEPLOYMENT_SERVICE_HEALTH.FAILING]
)
deployments = Deployment.list(filters=filters)
```

Performing OR logic

Some filters support comma-separated lists (and will say so if they do). Providing a comma-separated list of values to a single filter performs OR logic between those values. For example, the following will return all deployments whose service health is either warning OR failing.

```

from datarobot import Deployment
from datarobot.models.deployment import DeploymentListFilters
from datarobot.enums import DEPLOYMENT_SERVICE_HEALTH
filters = DeploymentListFilters(
    service_health=[
        DEPLOYMENT_SERVICE_HEALTH.WARNING,
        DEPLOYMENT_SERVICE_HEALTH.FAILING,
    ]
)
deployments = Deployment.list(filters=filters)

```

Performing OR logic across different filter types is not supported.

Note: In all cases, you may only retrieve deployments for which you have at least the USER role for. Deployments for which you are a CONSUMER of will not be returned, regardless of the filters applied.

class `datarobot.models.ServiceStats` (*period=None, metrics=None, model_id=None*)
 Deployment service stats information.

Attributes

model_id [str] the model used to retrieve service stats metrics
period [dict] the time period used to retrieve service stats metrics
metrics [dict] the service stats metrics

classmethod `get` (*deployment_id, model_id=None, start_time=None, end_time=None, execution_time_quantile=None, response_time_quantile=None, slow_requests_threshold=None*)

Retrieve value of service stat metrics over a certain time period.

New in version v2.18.

Parameters

deployment_id [str] the id of the deployment
model_id [str, optional] the id of the model
start_time [datetime, optional] start of the time period
end_time [datetime, optional] end of the time period
execution_time_quantile [float, optional] quantile for *executionTime*, defaults to 0.5
response_time_quantile [float, optional] quantile for *responseTime*, defaults to 0.5
slow_requests_threshold [float, optional] threshold for *slowRequests*, defaults to 1000

Returns

service_stats [ServiceStats] the queried service stats metrics

class `datarobot.models.ServiceStatsOverTime` (*buckets=None, summary=None, metric=None, model_id=None*)

Deployment service stats over time information.

Attributes

model_id [str] the model used to retrieve accuracy metric
metric [str] the service stat metric being retrieved

buckets [dict] how the service stat metric changes over time

summary [dict] summary for the service stat metric

classmethod **get** (*deployment_id*, *metric=None*, *model_id=None*, *start_time=None*,
end_time=None, *bucket_size=None*, *quantile=None*, *threshold=None*)

Retrieve information about how a service stat metric changes over a certain time period.

New in version v2.18.

Parameters

deployment_id [str] the id of the deployment

metric [SERVICE_STAT_METRIC, optional] the service stat metric to retrieve

model_id [str, optional] the id of the model

start_time [datetime, optional] start of the time period

end_time [datetime, optional] end of the time period

bucket_size [str, optional] time duration of a bucket, in ISO 8601 time duration format

quantile [float, optional] quantile for “executionTime” or “responseTime”, ignored when querying other metrics

threshold [int, optional] threshold for “slowQueries”, ignored when querying other metrics

Returns

service_stats_over_time [ServiceStatsOverTime] the queried service stat over time information

bucket_values

The metric value for all time buckets, keyed by start time of the bucket.

Returns

bucket_values: OrderedDict

class `datarobot.models.TargetDrift` (*period=None*, *metric=None*, *model_id=None*, *target_name=None*, *drift_score=None*, *sample_size=None*,
baseline_sample_size=None)

Deployment target drift information.

Attributes

model_id [str] the model used to retrieve target drift metric

period [dict] the time period used to retrieve target drift metric

metric [str] the data drift metric

target_name [str] name of the target

drift_score [float] target drift score

sample_size [int] count of data points for comparison

baseline_sample_size [int] count of data points for baseline

classmethod **get** (*deployment_id*, *model_id=None*, *start_time=None*, *end_time=None*, *metric=None*)

Retrieve target drift information over a certain time period.

New in version v2.21.

Parameters

deployment_id [str] the id of the deployment

model_id [str] the id of the model

start_time [datetime] start of the time period

end_time [datetime] end of the time period

metric [str] (New in version v2.22) metric used to calculate the drift score

Returns

target_drift [TargetDrift] the queried target drift information

Examples

```
from datarobot import Deployment, TargetDrift
deployment = Deployment.get(deployment_id='5c939e08962d741e34f609f0')
target_drift = TargetDrift.get(deployment.id)
target_drift.period['end']
>>>'2019-08-01 00:00:00+00:00'
target_drift.drift_score
>>>0.03423
accuracy.target_name
>>>'readmitted'
```

```
class datarobot.models.FeatureDrift (period=None, metric=None, model_id=None,
                                     name=None, drift_score=None, feature_impact=None,
                                     sample_size=None, baseline_sample_size=None)
```

Deployment feature drift information.

Attributes

model_id [str] the model used to retrieve feature drift metric

period [dict] the time period used to retrieve feature drift metric

metric [str] the data drift metric

name [str] name of the feature

drift_score [float] feature drift score

sample_size [int] count of data points for comparison

baseline_sample_size [int] count of data points for baseline

```
classmethod list (deployment_id, model_id=None, start_time=None, end_time=None, metric=None)
```

Retrieve drift information for deployment's features over a certain time period.

New in version v2.21.

Parameters

deployment_id [str] the id of the deployment

model_id [str] the id of the model

start_time [datetime] start of the time period

end_time [datetime] end of the time period

metric [str] (New in version v2.22) metric used to calculate the drift score

Returns

feature_drift_data [[FeatureDrift]] the queried feature drift information

Examples

```
from datarobot import Deployment, TargetDrift
deployment = Deployment.get(deployment_id='5c939e08962d741e34f609f0')
feature_drift = FeatureDrift.list(deployment.id)[0]
feature_drift.period
>>>'2019-08-01 00:00:00+00:00'
feature_drift.drift_score
>>>0.252
feature_drift.name
>>>'age'
```

class `datarobot.models.Accuracy` (*period=None, metrics=None, model_id=None*)
Deployment accuracy information.

Attributes

model_id [str] the model used to retrieve accuracy metrics

period [dict] the time period used to retrieve accuracy metrics

metrics [dict] the accuracy metrics

classmethod `get` (*deployment_id, model_id=None, start_time=None, end_time=None*)
Retrieve values of accuracy metrics over a certain time period.

New in version v2.18.

Parameters

deployment_id [str] the id of the deployment

model_id [str] the id of the model

start_time [datetime] start of the time period

end_time [datetime] end of the time period

Returns

accuracy [Accuracy] the queried accuracy metrics information

Examples

```
from datarobot import Deployment, Accuracy
deployment = Deployment.get(deployment_id='5c939e08962d741e34f609f0')
accuracy = Accuracy.get(deployment.id)
accuracy.period['end']
>>>'2019-08-01 00:00:00+00:00'
accuracy.metric['LogLoss']['value']
>>>0.7533
accuracy.metric_values['LogLoss']
>>>0.7533
```

metric_values

The value for all metrics, keyed by metric name.

Returns

metric_values: `OrderedDict`

metric_baselines

The baseline value for all metrics, keyed by metric name.

Returns

metric_baselines: `OrderedDict`

percent_changes

The percent change of value over baseline for all metrics, keyed by metric name.

Returns

percent_changes: `OrderedDict`

class `datarobot.models.AccuracyOverTime` (*buckets=None, summary=None, baseline=None, metric=None, model_id=None*)

Deployment accuracy over time information.

Attributes

model_id [str] the model used to retrieve accuracy metric

metric [str] the accuracy metric being retrieved

buckets [dict] how the accuracy metric changes over time

summary [dict] summary for the accuracy metric

baseline [dict] baseline for the accuracy metric

classmethod `get` (*deployment_id, metric=None, model_id=None, start_time=None, end_time=None, bucket_size=None*)

Retrieve information about how an accuracy metric changes over a certain time period.

New in version v2.18.

Parameters

deployment_id [str] the id of the deployment

metric [ACCURACY_METRIC] the accuracy metric to retrieve

model_id [str] the id of the model

start_time [datetime] start of the time period

end_time [datetime] end of the time period

bucket_size [str] time duration of a bucket, in ISO 8601 time duration format

Returns

accuracy_over_time [AccuracyOverTime] the queried accuracy metric over time information

Examples

```
from datarobot import Deployment, AccuracyOverTime
from datarobot.enums import ACCURACY_METRICS
deployment = Deployment.get(deployment_id='5c939e08962d741e34f609f0')
accuracy_over_time = AccuracyOverTime.get(deployment.id, metric=ACCURACY_
↳METRIC.LOGLOSS)
accuracy_over_time.metric
>>>'LogLoss'
accuracy_over_time.metric_values
>>>{datetime.datetime(2019, 8, 1): 0.73, datetime.datetime(2019, 8, 2): 0.55}
```

classmethod `get_as_dataframe` (*deployment_id*, *metrics*, *model_id=None*, *start_time=None*,
end_time=None, *bucket_size=None*)

Retrieve information about how a list of accuracy metrics change over a certain time period as pandas DataFrame.

In the returned DataFrame, the columns corresponds to the metrics being retrieved; the rows are labeled with the start time of each bucket.

Parameters

deployment_id [str] the id of the deployment

metrics [[`ACCURACY_METRIC`]] the accuracy metrics to retrieve

model_id [str] the id of the model

start_time [datetime] start of the time period

end_time [datetime] end of the time period

bucket_size [str] time duration of a bucket, in ISO 8601 time duration format

Returns

accuracy_over_time: `pd.DataFrame`

bucket_values

The metric value for all time buckets, keyed by start time of the bucket.

Returns

bucket_values: `OrderedDict`

bucket_sample_sizes

The sample size for all time buckets, keyed by start time of the bucket.

Returns

bucket_sample_sizes: `OrderedDict`

32.13 External Scores and Insights

class `datarobot.ExternalScores` (*project_id*, *scores*, *model_id=None*, *dataset_id=None*, *ac-*
tual_value_column=None)

Metric scores on prediction dataset with target or actual value column in unsupervised case. Contains project metrics for supervised and special classification metrics set for unsupervised projects.

New in version v2.21.

Examples

List all scores for a dataset

```
import datarobot as dr
scores = dr.Scores.list(project_id, dataset_id=dataset_id)
```

Attributes

project_id: str id of the project the model belongs to

model_id: str id of the model

dataset_id: str id of the prediction dataset with target or actual value column for unsupervised case

actual_value_column: str, optional For unsupervised projects only. Actual value column which was used to calculate the classification metrics and insights on the prediction dataset.

scores: list of dicts in a form of {*label*: *metric_name*, *value*: *score*}
Scores on the dataset.

classmethod create (*project_id*, *model_id*, *dataset_id*, *actual_value_column=None*)

Compute an external dataset insights for the specified model.

Parameters

project_id [str] id of the project the model belongs to

model_id [str] id of the model for which insights is requested

dataset_id [str] id of the dataset for which insights is requested

actual_value_column [str, optional] actual values column label, for unsupervised projects only

Returns

job [Job] an instance of created async job

classmethod list (*project_id*, *model_id=None*, *dataset_id=None*, *offset=0*, *limit=100*)

Fetch external scores list for the project and optionally for model and dataset.

Parameters

project_id: str id of the project

model_id: str, optional if specified, only scores for this model will be retrieved

dataset_id: str, optional if specified, only scores for this dataset will be retrieved

offset: int, optional this many results will be skipped, default: 0

limit: int, optional at most this many results are returned, default: 100, max 1000. To return all results, specify 0

Returns

A list of [External Scores](#) objects

classmethod get (*project_id*, *model_id*, *dataset_id*)

Retrieve external scores for the project, model and dataset.

Parameters

project_id: str id of the project

model_id: str if specified, only scores for this model will be retrieved

dataset_id: str if specified, only scores for this dataset will be retrieved

Returns

External Scores object

class datarobot.**ExternalLiftChart** (*dataset_id, bins*)

Lift chart for the model and prediction dataset with target or actual value column in unsupervised case.

New in version v2.21.

LiftChartBin is a dict containing the following:

- **actual** (float) Sum of actual target values in bin
- **predicted** (float) Sum of predicted target values in bin
- **bin_weight** (float) The weight of the bin. For weighted projects, it is the sum of the weights of the rows in the bin. For unweighted projects, it is the number of rows in the bin.

Attributes

dataset_id: str id of the prediction dataset with target or actual value column for unsupervised case

bins: list of dict List of dicts with schema described as LiftChartBin above.

classmethod **list** (*project_id, model_id, dataset_id=None, offset=0, limit=100*)

Retrieve list of the lift charts for the model.

Parameters

project_id: str id of the project

model_id: str if specified, only lift chart for this model will be retrieved

dataset_id: str, optional if specified, only lift chart for this dataset will be retrieved

offset: int, optional this many results will be skipped, default: 0

limit: int, optional at most this many results are returned, default: 100, max 1000. To return all results, specify 0

Returns

A list of *ExternalLiftChart* objects

classmethod **get** (*project_id, model_id, dataset_id*)

Retrieve lift chart for the model and prediction dataset.

Parameters

project_id: str project id

model_id: str model id

dataset_id: str prediction dataset id with target or actual value column for unsupervised case

Returns

ExternalLiftChart object

class `datarobot.ExternalRocCurve` (*dataset_id, roc_points, negative_class_predictions, positive_class_predictions*)
 ROC curve data for the model and prediction dataset with target or actual value column in unsupervised case.
 New in version v2.21.

Attributes

dataset_id: str id of the prediction dataset with target or actual value column for unsupervised case

roc_points: list of dict List of precalculated metrics associated with thresholds for ROC curve.

negative_class_predictions: list of float List of predictions from example for negative class

positive_class_predictions: list of float List of predictions from example for positive class

classmethod list (*project_id, model_id, dataset_id=None, offset=0, limit=100*)
 Retrieve list of the roc curves for the model.

Parameters

project_id: str id of the project

model_id: str if specified, only lift chart for this model will be retrieved

dataset_id: str, optional if specified, only lift chart for this dataset will be retrieved

offset: int, optional this many results will be skipped, default: 0

limit: int, optional at most this many results are returned, default: 100, max 1000. To return all results, specify 0

Returns

A list of [`ExternalRocCurve`](#) objects

classmethod get (*project_id, model_id, dataset_id*)
 Retrieve ROC curve chart for the model and prediction dataset.

Parameters

project_id: str project id

model_id: str model id

dataset_id: str prediction dataset id with target or actual value column for unsupervised case

Returns

[`ExternalRocCurve`](#) object

32.14 Feature

class `datarobot.models.Feature` (*id, project_id=None, name=None, feature_type=None, importance=None, low_information=None, unique_count=None, na_count=None, date_format=None, min=None, max=None, mean=None, median=None, std_dev=None, time_series_eligible=None, time_series_eligibility_reason=None, time_step=None, time_unit=None, target_leakage=None, key_summary=None*)

A feature from a project's dataset

These are features either included in the originally uploaded dataset or added to it via feature transformations. In time series projects, these will be distinct from the *ModelingFeatures* created during partitioning; otherwise, they will correspond to the same features. For more information about input and modeling features, see the *time series documentation*.

The `min`, `max`, `mean`, `median`, and `std_dev` attributes provide information about the distribution of the feature in the EDA sample data. For non-numeric features or features created prior to these summary statistics becoming available, they will be `None`. For features where the summary statistics are available, they will be in a format compatible with the data type, i.e. date type features will have their summary statistics expressed as ISO-8601 formatted date strings.

Attributes

id [int] the id for the feature - note that *name* is used to reference the feature instead of *id*

project_id [str] the id of the project the feature belongs to

name [str] the name of the feature

feature_type [str] the type of the feature, e.g. `âCategoricalâ`, `âTextâ`

importance [float or None] numeric measure of the strength of relationship between the feature and target (independent of any model or other features); may be `None` for non-modeling features such as partition columns

low_information [bool] whether a feature is considered too uninformative for modeling (e.g. because it has too few values)

unique_count [int] number of unique values

na_count [int or None] number of missing values

date_format [str or None] For Date features, the date format string for how this feature was interpreted, compatible with <https://docs.python.org/2/library/time.html#time.strptime> . For other feature types, `None`.

min [str, int, float, or None] The minimum value of the source data in the EDA sample

max [str, int, float, or None] The maximum value of the source data in the EDA sample

mean [str, int, or, float] The arithmetic mean of the source data in the EDA sample

median [str, int, float, or None] The median of the source data in the EDA sample

std_dev [str, int, float, or None] The standard deviation of the source data in the EDA sample

time_series_eligible [bool] Whether this feature can be used as the datetime partition column in a time series project.

time_series_eligibility_reason [str] Why the feature is ineligible for the datetime partition column in a time series project, or `âsuitableâ` when it is eligible.

time_step [int or None] For time series eligible features, a positive integer determining the interval at which windows can be specified. If used as the datetime partition column on a time series project, the feature derivation and forecast windows must start and end at an integer multiple of this value. `None` for features that are not time series eligible.

time_unit [str or None] For time series eligible features, the time unit covered by a single time step, e.g. `âHOURâ`, or `None` for features that are not time series eligible.

target_leakage [str] Whether a feature is considered to have target leakage or not. A value of `âSKIPPED_DETECTIONâ` indicates that target leakage detection was not run on the feature. `âFALSEâ` indicates no leakage, `âMODERATEâ` indicates a moderate risk of target leakage, and `âHIGH_RISKâ` indicates a high risk of target leakage

key_summary: list of dict Statistics for top 50 keys (truncated to 103 characters) of Summarized Categorical column example:

```
{'key': 'DataRobot', 'summary': {'min': 0, 'max': 29815.0, 'stdDev': 6498.029, 'mean': 1490.75, 'median': 0.0, 'pctRows': 5.0}}
```

where,

key: string or None name of the key

summary: dict statistics of the key

max: maximum value of the key. min: minimum value of the key. mean: mean value of the key. median: median value of the key. stdDev: standard deviation of the key. pctRows: percentage occurrence of key in the EDA sample of the feature.

classmethod get (*project_id, feature_name*)

Retrieve a single feature

Parameters

project_id [str] The ID of the project the feature is associated with.

feature_name [str] The name of the feature to retrieve

Returns

feature [Feature] The queried instance

get_multiseries_properties (*multiseries_id_columns, max_wait=600*)

Retrieve time series properties for a potential multiseries datetime partition column

Multiseries time series projects use multiseries id columns to model multiple distinct series within a single project. This function returns the time series properties (time step and time unit) of this column if it were used as a datetime partition column with the specified multiseries id columns, running multiseries detection automatically if it had not previously been successfully ran.

Parameters

multiseries_id_columns [list of str] the name(s) of the multiseries id columns to use with this datetime partition column. Currently only one multiseries id column is supported.

max_wait [int, optional] if a multiseries detection task is run, the maximum amount of time to wait for it to complete before giving up

Returns

properties [dict] A dict with three keys:

- **time_series_eligible** : bool, whether the column can be used as a partition column
- **time_unit** : str or null, the inferred time unit if used as a partition column
- **time_step** : int or null, the inferred time step if used as a partition column

get_cross_series_properties (*datetime_partition_column, cross_series_group_by_columns, max_wait=600*)

Retrieve cross-series properties for multiseries ID column.

This function returns the cross-series properties (eligibility as group-by column) of this column if it were used with specified datetime partition column and with current multiseries id column, running cross-series group-by validation automatically if it had not previously been successfully ran.

Parameters

datetime_partition_column [datetime partition column]

cross_series_group_by_columns [list of str] the name(s) of the columns to use with this multiserries ID column. Currently only one cross-series group-by column is supported.

max_wait [int, optional] if a multiserries detection task is run, the maximum amount of time to wait for it to complete before giving up

Returns

properties [dict] A dict with three keys:

- **name** : str, column name
- **eligibility** : str, reason for column eligibility
- **isEligible** : bool, is column eligible as cross-series group-by

```
class datarobot.models.ModelingFeature (project_id=None, name=None, feature_type=None,
                                         importance=None, low_information=None,
                                         unique_count=None, na_count=None,
                                         date_format=None, min=None, max=None,
                                         mean=None, median=None, std_dev=None,
                                         parent_feature_names=None, key_summary=None)
```

A feature used for modeling

In time series projects, a new set of modeling features is created after setting the partitioning options. These features are automatically derived from those in the project's dataset and are the features used for modeling. Modeling features are only accessible once the target and partitioning options have been set. In projects that don't use time series modeling, once the target has been set, ModelingFeatures and Features will behave the same.

For more information about input and modeling features, see the [time series documentation](#).

As with the [Feature](#) object, the *min*, *max*, *mean*, *median*, and *std_dev* attributes provide information about the distribution of the feature in the EDA sample data. For non-numeric features, they will be None. For features where the summary statistics are available, they will be in a format compatible with the data type, i.e. date type features will have their summary statistics expressed as ISO-8601 formatted date strings.

Attributes

project_id [str] the id of the project the feature belongs to

name [str] the name of the feature

feature_type [str] the type of the feature, e.g. 'Categorical', 'Text'

importance [float or None] numeric measure of the strength of relationship between the feature and target (independent of any model or other features); may be None for non-modeling features such as partition columns

low_information [bool] whether a feature is considered too uninformative for modeling (e.g. because it has too few values)

unique_count [int] number of unique values

na_count [int or None] number of missing values

date_format [str or None] For Date features, the date format string for how this feature was interpreted, compatible with <https://docs.python.org/2/library/time.html#time.strptime> . For other feature types, None.

min [str, int, float, or None] The minimum value of the source data in the EDA sample

max [str, int, float, or None] The maximum value of the source data in the EDA sample

mean [str, int, or, float] The arithmetic mean of the source data in the EDA sample

median [str, int, float, or None] The median of the source data in the EDA sample

std_dev [str, int, float, or None] The standard deviation of the source data in the EDA sample

parent_feature_names [list of str] A list of the names of input features used to derive this modeling feature. In cases where the input features and modeling features are the same, this will simply contain the feature's name. Note that if a derived feature was used to create this modeling feature, the values here will not necessarily correspond to the features that must be supplied at prediction time.

key_summary: list of dict Statistics for top 50 keys (truncated to 103 characters) of Summarized Categorical column example:

```
{'key': 'DataRobot', 'summary': {'min': 0, 'max': 29815.0, 'stdDev': 6498.029, 'mean': 1490.75, 'median': 0.0, 'pctRows': 5.0}}
```

where,

key: string or None name of the key

summary: dict statistics of the key

max: maximum value of the key. min: minimum value of the key. mean: mean value of the key. median: median value of the key. stdDev: standard deviation of the key. pctRows: percentage occurrence of key in the EDA sample of the feature.

classmethod get (*project_id*, *feature_name*)

Retrieve a single modeling feature

Parameters

project_id [str] The ID of the project the feature is associated with.

feature_name [str] The name of the feature to retrieve

Returns

feature [ModelingFeature] The requested feature

```
class datarobot.models.DatasetFeature(id_, dataset_id=None, dataset_version_id=None,
                                     name=None, feature_type=None,
                                     low_information=None, unique_count=None,
                                     na_count=None, date_format=None, min_=None,
                                     max_=None, mean=None, median=None,
                                     std_dev=None, time_series_eligible=None,
                                     time_series_eligibility_reason=None,
                                     time_step=None, time_unit=None, target_leakage=None, target_leakage_reason=None)
```

A feature from a project's dataset

These are features either included in the originally uploaded dataset or added to it via feature transformations.

The `min`, `max`, `mean`, `median`, and `std_dev` attributes provide information about the distribution of the feature in the EDA sample data. For non-numeric features or features created prior to these summary statistics becoming available, they will be `None`. For features where the summary statistics are available, they will be in a format compatible with the data type, i.e. date type features will have their summary statistics expressed as ISO-8601 formatted date strings.

Attributes

id [int] the id for the feature - note that *name* is used to reference the feature instead of *id*

dataset_id [str] the id of the dataset the feature belongs to

dataset_version_id [str] the id of the dataset version the feature belongs to

name [str] the name of the feature

feature_type [str, optional] the type of the feature, e.g. `âCategoricalâ`, `âTextâ`

low_information [bool, optional] whether a feature is considered too uninformative for modeling (e.g. because it has too few values)

unique_count [int, optional] number of unique values

na_count [int, optional] number of missing values

date_format [str, optional] For Date features, the date format string for how this feature was interpreted, compatible with <https://docs.python.org/2/library/time.html#time.strptime> . For other feature types, None.

min [str, int, float, optional] The minimum value of the source data in the EDA sample

max [str, int, float, optional] The maximum value of the source data in the EDA sample

mean [str, int, float, optional] The arithmetic mean of the source data in the EDA sample

median [str, int, float, optional] The median of the source data in the EDA sample

std_dev [str, int, float, optional] The standard deviation of the source data in the EDA sample

time_series_eligible [bool, optional] Whether this feature can be used as the datetime partition column in a time series project.

time_series_eligibility_reason [str, optional] Why the feature is ineligible for the datetime partition column in a time series project, or `âsuitableâ` when it is eligible.

time_step [int, optional] For time series eligible features, a positive integer determining the interval at which windows can be specified. If used as the datetime partition column on a time series project, the feature derivation and forecast windows must start and end at an integer multiple of this value. None for features that are not time series eligible.

time_unit [str, optional] For time series eligible features, the time unit covered by a single time step, e.g. `âHOURâ`, or None for features that are not time series eligible.

target_leakage [str, optional] Whether a feature is considered to have target leakage or not. A value of `âSKIPPED_DETECTIONâ` indicates that target leakage detection was not run on the feature. `âFALSEâ` indicates no leakage, `âMODERATEâ` indicates a moderate risk of target leakage, and `âHIGH_RISKâ` indicates a high risk of target leakage

target_leakage_reason: string, optional The descriptive text explaining the reason for target leakage, if any.

get_histogram (*bin_limit=None*)

Retrieve a feature histogram

Parameters

bin_limit [int or None] Desired max number of histogram bins. If omitted, by default endpoint will use 60.

Returns

featureHistogram [DatasetFeatureHistogram] The requested histogram with desired number or bins

class `datarobot.models.DatasetFeatureHistogram` (*plot*)

classmethod `get(dataset_id, feature_name, bin_limit=None, key_name=None)`

Retrieve a single feature histogram

Parameters

dataset_id [str] The ID of the Dataset the feature is associated with.

feature_name [str] The name of the feature to retrieve

bin_limit [int or None] Desired max number of histogram bins. If omitted, by default the endpoint will use 60.

key_name: string or None (Only required for summarized categorical feature) Name of the top 50 keys for which plot to be retrieved

Returns

featureHistogram [FeatureHistogram] The queried instance with *plot* attribute in it.

class `datarobot.models.FeatureHistogram(plot)`

classmethod `get(project_id, feature_name, bin_limit=None, key_name=None)`

Retrieve a single feature histogram

Parameters

project_id [str] The ID of the project the feature is associated with.

feature_name [str] The name of the feature to retrieve

bin_limit [int or None] Desired max number of histogram bins. If omitted, by default endpoint will use 60.

key_name: string or None (Only required for summarized categorical feature) Name of the top 50 keys for which plot to be retrieved

Returns

featureHistogram [FeatureHistogram] The queried instance with *plot* attribute in it.

class `datarobot.models.InteractionFeature(rows, source_columns, bars, bubbles)`

Interaction feature data

New in version v2.21.

Attributes

rows: int Total number of rows

source_columns: list(str) names of two categorical features which were combined into this one

bars: list(dict) dictionaries representing frequencies of each independent value from the source columns

bubbles: list(dict) dictionaries representing frequencies of each combined value in the interaction feature.

classmethod `get(project_id, feature_name)`

Retrieve a single Interaction feature

Parameters

project_id [str] The id of the project the feature belongs to

feature_name [str] The name of the Interaction feature to retrieve

Returns

feature [InteractionFeature] The queried instance

32.15 Feature Engineering

```
class datarobot.models.FeatureEngineeringGraph (id=None, name=None, de-  
scription=None, created=None,  
last_modified=None, cre-  
ator_full_name=None, mod-  
ifier_full_name=None,  
creator_user_id=None,  
last_modified_user_id=None,  
number_of_projects=None,  
linkage_keys=None, ta-  
ble_definitions=None, relation-  
ships=None, time_unit=None, fea-  
ture_derivation_window_start=None,  
feature_derivation_window_end=None,  
is_draft=True)
```

A Feature Engineering Graph for the Project. A Feature Engineering Graph is graph which allow to specify relationships between two or more tables so it can automatically generate features from that

Attributes

id [str] the id of the created feature engineering graph

name: **str** name of the feature engineering graph

description: **str** description of the feature engineering graph

created: **datetime.datetime** creation date of the feature engineering graph

creator_user_id: **str** id of the user who created the feature engineering graph

creator_full_name: **str** full name of the user who created the feature engineering graph

last_modified: **datetime.datetime** last modification date of the feature engineering graph

last_modified_user_id: **str** id of the user who last modified the feature engineering graph

modifier_full_name: **str** full name of the user who last modified the feature engineering graph

number_of_projects: **int** number of projects that are used in the feature engineering graph

linkage_keys: **list** or **str** a list of strings specifying the name of the columns that link the feature engineering graph with the primary table.

table_definitions: **list** each element is a table_definition for a table.

relationships: **list** each element is a relationship between two tables

time_unit: **str**, or **None** time unit of the feature derivation window. Supported values are MIL-
LISECOND, SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER, YEAR. If
present, the feature engineering Graph will perform time-aware joins.

feature_derivation_window_start: **int**, or **None** how many time_units of each table's pri-
mary temporal key into the past relative to the datetimePartitionColumn the feature deriva-
tion window should begin. Will be a negative integer, If present, the feature engineering
Graph will perform time-aware joins.

feature_derivation_window_end: int, or None how many timeUnits of each table's record primary temporal key into the past relative to the datetimePartitionColumn the feature derivation window should end. Will be a non-positive integer, if present. If present, the feature engineering Graph will perform time-aware joins.

is_draft: bool (default=True) a draft (is_draft=True) feature engineering graph can be updated, while a non-draft(is_draft=False) feature engineering graph is immutable

The 'table_definitions' structure is

identifier: str alias of the table (used directly as part of the generated feature names)

catalog_id: str, or None identifier of the catalog item

catalog_version_id: str identifier of the catalog item version

feature_list_id: str, or None identifier of the feature list. This decides which columns in the table are used for feature generation

primary_temporal_key: str, or None name of the column indicating time of record creation

snapshot_policy: str policy to use when creating a project or making predictions. Must be one of the following values: 'specified': Use specific snapshot specified by catalogVersionId 'latest': Use latest snapshot from the same catalog item 'dynamic': Get data from the source (only applicable for JDBC datasets)

feature_lists: list list of feature list info

data_source: dict data source info if the table is from data source

is_deleted: bool or None whether the table is deleted or not

The 'relationship' structure is

table1_identifier: str or None identifier of the first table in this relationship. This is specified in the identifier field of table_definition structure. If None, then the relationship is with the primary dataset.

table2_identifier: str identifier of the second table in this relationship. This is specified in the identifier field of table_definition schema.

table1_keys: list of str (max length: 10 min length: 1) column(s) from the first table which are used to join to the second table

table2_keys: list of str (max length: 10 min length: 1) column(s) from the second table that are used to join to the first table

The 'feature list info' structure is

id [str] the id of the featurelist

name [str] the name of the featurelist

features [list of str] the names of all the Features in the featurelist

dataset_id [str] the project the featurelist belongs to

creation_date [datetime.datetime] when the featurelist was created

user_created [bool] whether the featurelist was created by a user or by DataRobot automation

created_by: str the name of user who created it

description [str] the description of the featurelist. Can be updated by the user and may be supplied by default for DataRobot-created featurelists.

dataset_id: str dataset which is associated with the feature list

dataset_version_id: **str** or **None** version of the dataset which is associated with feature list.
Only relevant for Informative features

The ‘data source info’ structured is

data_store_id: **str** the id of the data store.

data_store_name [str] the user-friendly name of the data store.

url [str] the url used to connect to the data store.

dbtable [str] the name of table from the data store.

schema: **str** schema definition of the table from the data store

```
classmethod create (name, description, table_definitions, relationships,
                    time_unit=None, feature_derivation_window_start=None, fea-
                    ture_derivation_window_end=None, is_draft=True)
Create a feature engineering graph.
```

Parameters

name [str] the name of the feature engineering graph

description [str] the description of the feature engineering graph

table_definitions: **list of dict** each element is a TableDefinition for a table. The *TableDefinition* schema is

identifier: **str** alias of the table (used directly as part of the generated feature names)

catalog_id: **str, or None** identifier of the catalog item

catalog_version_id: **str** identifier of the catalog item version

feature_list_id: **str, or None** identifier of the feature list. This decides which columns in the table are used for feature generation

primary_temporal_key: **str, or None** name of the column indicating time of record creation

snapshot_policy: **str** policy to use when creating a project or making predictions. Must be one of the following values: ‘specified’: Use specific snapshot specified by catalogVersionId ‘latest’: Use latest snapshot from the same catalog item ‘dynamic’: Get data from the source (only applicable for JDBC datasets)

relationships: **list of dict** each element is a Relationship between two tables The *Relationship* schema is

table1_identifier: **str or None** identifier of the first table in this relationship. This is specified in the identifier field of table_definition structure. If None, then the relationship is with the primary dataset.

table2_identifier: **str** identifier of the second table in this relationship. This is specified in the identifier field of table_definition schema.

table1_keys: **list of str (max length: 10 min length: 1)** column(s) from the first table which are used to join to the second table

table2_keys: **list of str (max length: 10 min length: 1)** column(s) from the second table that are used to join to the first table

time_unit: **str, or None** time unit of the feature derivation window. Supported values are MILLISECOND, SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER, YEAR. If present, the feature engineering Graph will perform time-aware joins.

feature_derivation_window_start: int, or None how many time_units of each table's primary temporal key into the past relative to the datetimePartitionColumn the feature derivation window should begin. Will be a negative integer, If present, the feature engineering Graph will perform time-aware joins.

feature_derivation_window_end: int, or None how many timeUnits of each table's record primary temporal key into the past relative to the datetimePartitionColumn the feature derivation window should end. Will be a non-positive integer, if present. If present, the feature engineering Graph will perform time-aware joins.

is_draft: bool (default=True) a draft (is_draft=True) feature engineering graph can be updated, while a non-draft(is_draft=False) feature engineering graph is immutable

Returns

feature_engineering_graphs: FeatureEngineeringGraph the created feature engineering graph

replace (*id*, *name*, *description*, *table_definitions*, *relationships*, *time_unit=None*, *feature_derivation_window_start=None*, *feature_derivation_window_end=None*, *is_draft=True*)

Replace a feature engineering graph.

Parameters

id [str] the id of the created feature engineering graph

name [str] the name of the feature engineering graph

description [str] the description of the feature engineering graph

items: list of dict each element is a TableDefinition for a table. The *TableDefinition* schema is

identifier: str alias of the table (used directly as part of the generated feature names)

catalog_id: str, or None identifier of the catalog item

catalog_version_id: str identifier of the catalog item version

feature_list_id: str, or None identifier of the feature list. This decides which columns in the table are used for feature generation

primary_temporal_key: str, or None name of the column indicating time of record creation

snapshot_policy: str policy to use when creating a project or making predictions. Must be one of the following values: "specified": Use specific snapshot specified by catalogVersionId "latest": Use latest snapshot from the same catalog item "dynamic": Get data from the source (only applicable for JDBC datasets)

relationships: list of dict each element is a Relationship between two tables The *Relationship* schema is

table1_identifier: str or None identifier of the first table in this relationship. This is specified in the identifier field of table_definition structure. If None, then the relationship is with the primary dataset.

table2_identifier: str identifier of the second table in this relationship. This is specified in the identifier field of table_definition schema.

table1_keys: list of str (max length: 10 min length: 1) column(s) from the first table which are used to join to the second table

table2_keys: list of str (max length: 10 min length: 1) column(s) from the second table that are used to join to the first table

time_unit: str, or None time unit of the feature derivation window. Supported values are `MILLISECOND`, `SECOND`, `MINUTE`, `HOURL`, `DAY`, `WEEK`, `MONTH`, `QUARTER`, `YEAR`. If present, the feature engineering Graph will perform time-aware joins.

feature_derivation_window_start: int, or None how many time_units of each table's primary temporal key into the past relative to the `datetimePartitionColumn` the feature derivation window should begin. Will be a negative integer, If present, the feature engineering Graph will perform time-aware joins.

feature_derivation_window_end: int, or None how many timeUnits of each table's record primary temporal key into the past relative to the `datetimePartitionColumn` the feature derivation window should end. Will be a non-positive integer, if present. If present, the feature engineering Graph will perform time-aware joins.

is_draft: bool (default=True) a draft (is_draft=True) feature engineering graph can be updated, while a non-draft(is_draft=False) feature engineering graph is immutable

Returns

feature_engineering_graphs: `FeatureEngineeringGraph` the updated feature engineering graph

update (*name*, *description*)

Update the Feature engineering graph name and description.

Parameters

name [str] the name of the feature engineering graph

description [str] the description of the feature engineering graph

classmethod get (*feature_engineering_graph_id*)

Retrieve a single feature engineering graph

Parameters

feature_engineering_graph_id [str] The ID of the feature engineering graph to retrieve.

Returns

feature_engineering_graph [`FeatureEngineeringGraph`] The requested feature engineering graph

classmethod list (*project_id=None*, *secondary_dataset_id=None*, *include_drafts=None*)

Returns list of feature engineering graphs.

Parameters

project_id: str, optional The Id of project to filter the feature engineering graph list for returning only those feature engineering Graphs which are related to this project If not specified, it will return all the feature engineering graphs irrespective of the project

secondary_dataset_id: str, optional ID of the dataset to filter feature engineering graphs which use the dataset as the secondary dataset If not specified, return all the feature engineering graphs without filtering on secondary dataset id.

include_drafts: bool (default=False) include draft feature engineering graphs If *True*, return all the draft (mutable) as well as non-draft (immutable) feature engineering graphs

Returns

feature_engineering_graphs [list of FeatureEngineeringGraph instances] a list of available feature engineering graphs.

delete()

Delete the Feature Engineering Graph

share (*access_list*)

Modify the ability of users to access this feature engineering graph

Parameters

access_list [list of *SharingAccess*] the modifications to make.

Raises

datarobot.ClientError : if you do not have permission to share this feature engineering graph or if the user you’re sharing with doesn’t exist

get_access_list()

Retrieve what users have access to this feature engineering graph

Returns

list of *SharingAccess*

32.16 Feature List

class `datarobot.DatasetFeaturelist` (*id=None, name=None, features=None, dataset_id=None, dataset_version_id=None, creation_date=None, created_by=None, user_created=None, description=None*)

A set of features attached to a dataset in the AI Catalog

Attributes

id [str] the id of the dataset featurelist

dataset_id [str] the id of the dataset the featurelist belongs to

dataset_version_id: str, optional the version id of the dataset this featurelist belongs to

name [str] the name of the dataset featurelist

features [list of str] a list of the names of features included in this dataset featurelist

creation_date [datetime.datetime] when the featurelist was created

created_by [str] the user name of the user who created this featurelist

user_created [bool] whether the featurelist was created by a user or by DataRobot automation

description [basestring, optional] the description of the featurelist. Only present on DataRobot-created featurelists.

classmethod `get` (*dataset_id, featurelist_id*)

Retrieve a dataset featurelist

Parameters

dataset_id [str] the id of the dataset the featurelist belongs to

featurelist_id [str] the id of the dataset featurelist to retrieve

Returns

featurelist [DatasetFeatureList] the specified featurelist

delete()

Delete a dataset featurelist

Featurelists configured into the dataset as a default featurelist cannot be deleted.

update (*name=None*)

Update the name of an existing featurelist

Note that only user-created featurelists can be renamed, and that names must not conflict with names used by other featurelists.

Parameters

name [str, optional] the new name for the featurelist

```
class datarobot.models.Featurelist (id=None, name=None, features=None,  
                                     project_id=None, created=None, is_user_created=None,  
                                     num_models=None, description=None)
```

A set of features used in modeling

Attributes

id [str] the id of the featurelist

name [str] the name of the featurelist

features [list of str] the names of all the Features in the featurelist

project_id [str] the project the featurelist belongs to

created [datetime.datetime] (New in version v2.13) when the featurelist was created

is_user_created [bool] (New in version v2.13) whether the featurelist was created by a user or by DataRobot automation

num_models [int] (New in version v2.13) the number of models currently using this featurelist. A model is considered to use a featurelist if it is used to train the model or as a monotonic constraint featurelist, or if the model is a blender with at least one component model using the featurelist.

description [basestring] (New in version v2.13) the description of the featurelist. Can be updated by the user and may be supplied by default for DataRobot-created featurelists.

classmethod get (*project_id, featurelist_id*)

Retrieve a known feature list

Parameters

project_id [str] The id of the project the featurelist is associated with

featurelist_id [str] The ID of the featurelist to retrieve

Returns

featurelist [Featurelist] The queried instance

delete (*dry_run=False, delete_dependencies=False*)

Delete a featurelist, and any models and jobs using it

All models using a featurelist, whether as the training featurelist or as a monotonic constraint featurelist, will also be deleted when the deletion is executed and any queued or running jobs using it will be cancelled. Similarly, predictions made on these models will also be deleted. All the entities that are to be deleted with a featurelist are described as “dependencies” of it. To preview the results of deleting a featurelist, call delete with *dry_run=True*

When deleting a featurelist with dependencies, users must specify `delete_dependencies=True` to confirm they want to delete the featurelist and all its dependencies. Without that option, only featurelists with no dependencies may be successfully deleted and others will error.

Featurelists configured into the project as a default featurelist or as a default monotonic constraint featurelist cannot be deleted.

Featurelists used in a model deployment cannot be deleted until the model deployment is deleted.

Parameters

dry_run [bool, optional] specify True to preview the result of deleting the featurelist, instead of actually deleting it.

delete_dependencies [bool, optional] specify True to successfully delete featurelists with dependencies; if left False by default, featurelists without dependencies can be successfully deleted and those with dependencies will error upon attempting to delete them.

Returns

result [dict]

A dictionary describing the result of deleting the featurelist, with the following keys

- **dry_run** : bool, whether the deletion was a dry run or an actual deletion
- **can_delete** : bool, whether the featurelist can actually be deleted
- **deletion_blocked_reason** : str, why the featurelist can't be deleted (if it can't)
- **num_affected_models** : int, the number of models using this featurelist
- **num_affected_jobs** : int, the number of jobs using this featurelist

update (*name=None, description=None*)

Update the name or description of an existing featurelist

Note that only user-created featurelists can be renamed, and that names must not conflict with names used by other featurelists.

Parameters

name [str, optional] the new name for the featurelist

description [str, optional] the new description for the featurelist

```
class datarobot.models.ModelingFeaturelist (id=None, name=None, features=None,  
                                           project_id=None, created=None,  
                                           is_user_created=None, num_models=None,  
                                           description=None)
```

A set of features that can be used to build a model

In time series projects, a new set of modeling features is created after setting the partitioning options. These features are automatically derived from those in the project's dataset and are the features used for modeling. Modeling features are only accessible once the target and partitioning options have been set. In projects that don't use time series modeling, once the target has been set, ModelingFeaturelists and Featurelists will behave the same.

For more information about input and modeling features, see the [time series documentation](#).

Attributes

id [str] the id of the modeling featurelist

project_id [str] the id of the project the modeling featurelist belongs to

name [str] the name of the modeling featurelist

features [list of str] a list of the names of features included in this modeling featurelist

created [datetime.datetime] (New in version v2.13) when the featurelist was created

is_user_created [bool] (New in version v2.13) whether the featurelist was created by a user or by DataRobot automation

num_models [int] (New in version v2.13) the number of models currently using this featurelist. A model is considered to use a featurelist if it is used to train the model or as a monotonic constraint featurelist, or if the model is a blender with at least one component model using the featurelist.

description [basestring] (New in version v2.13) the description of the featurelist. Can be updated by the user and may be supplied by default for DataRobot-created featurelists.

classmethod get (*project_id, featurelist_id*)

Retrieve a modeling featurelist

Modeling featurelists can only be retrieved once the target and partitioning options have been set.

Parameters

project_id [str] the id of the project the modeling featurelist belongs to

featurelist_id [str] the id of the modeling featurelist to retrieve

Returns

featurelist [ModelingFeaturelist] the specified featurelist

delete (*dry_run=False, delete_dependencies=False*)

Delete a featurelist, and any models and jobs using it

All models using a featurelist, whether as the training featurelist or as a monotonic constraint featurelist, will also be deleted when the deletion is executed and any queued or running jobs using it will be cancelled. Similarly, predictions made on these models will also be deleted. All the entities that are to be deleted with a featurelist are described as “dependencies” of it. To preview the results of deleting a featurelist, call delete with *dry_run=True*

When deleting a featurelist with dependencies, users must specify *delete_dependencies=True* to confirm they want to delete the featurelist and all its dependencies. Without that option, only featurelists with no dependencies may be successfully deleted and others will error.

Featurelists configured into the project as a default featurelist or as a default monotonic constraint featurelist cannot be deleted.

Featurelists used in a model deployment cannot be deleted until the model deployment is deleted.

Parameters

dry_run [bool, optional] specify True to preview the result of deleting the featurelist, instead of actually deleting it.

delete_dependencies [bool, optional] specify True to successfully delete featurelists with dependencies; if left False by default, featurelists without dependencies can be successfully deleted and those with dependencies will error upon attempting to delete them.

Returns

result [dict]

A dictionary describing the result of deleting the featurelist, with the following keys

- `dry_run` : bool, whether the deletion was a dry run or an actual deletion
- `can_delete` : bool, whether the featurelist can actually be deleted
- `deletion_blocked_reason` : str, why the featurelist can't be deleted (if it can't)
- `num_affected_models` : int, the number of models using this featurelist
- `num_affected_jobs` : int, the number of jobs using this featurelist

update (*name=None, description=None*)

Update the name or description of an existing featurelist

Note that only user-created featurelists can be renamed, and that names must not conflict with names used by other featurelists.

Parameters

- name** [str, optional] the new name for the featurelist
- description** [str, optional] the new description for the featurelist

32.17 Job

class `datarobot.models.Job` (*data, completed_resource_url=None*)

Tracks asynchronous work being done within a project

Attributes

- id** [int] the id of the job
- project_id** [str] the id of the project the job belongs to
- status** [str] the status of the job - will be one of `datarobot.enums.QUEUE_STATUS`
- job_type** [str] what kind of work the job is doing - will be one of `datarobot.enums.JOB_TYPE`
- is_blocked** [bool] if true, the job is blocked (cannot be executed) until its dependencies are resolved

classmethod `get` (*project_id, job_id*)

Fetches one job.

Parameters

- project_id** [str] The identifier of the project in which the job resides
- job_id** [str] The job id

Returns

- job** [Job] The job

Raises

- AsyncFailureError** Querying this resource gave a status code other than 200 or 303

cancel ()

Cancel this job. If this job has not finished running, it will be removed and canceled.

get_result (*params=None*)

Parameters

- params** [dict or None] Query parameters to be added to request to get results.

For `featureEffects` and `featureFit`, `source` param is required to define source, otherwise the default is ‘training’

Returns

result [object]

Return type depends on the job type:

- for model jobs, a `Model` is returned
- for predict jobs, a `pandas.DataFrame` (with predictions) is returned
- for `featureImpact` jobs, a list of dicts by default (see `with_metadata` parameter of the `FeatureImpactJob` class and its `get()` method).
- for `primeRulesets` jobs, a list of `Rulesets`
- for `primeModel` jobs, a `PrimeModel`
- for `primeDownloadValidation` jobs, a `PrimeFile`
- for `reasonCodesInitialization` jobs, a `ReasonCodesInitialization`
- for `reasonCodes` jobs, a `ReasonCodes`
- for `predictionExplanationInitialization` jobs, a `PredictionExplanationsInitialization`
- for `predictionExplanations` jobs, a `PredictionExplanations`
- for `featureEffects`, a `FeatureEffects`
- for `featureFit`, a `FeatureFit`

Raises

JobNotFinished If the job is not finished, the result is not available.

AsyncProcessUnsuccessfulError If the job errored or was aborted

get_result_when_complete (*max_wait=600, params=None*)

Parameters

max_wait [int, optional] How long to wait for the job to finish.

params [dict, optional] Query parameters to be added to request.

Returns

result: object Return type is the same as would be returned by *Job.get_result*.

Raises

AsyncTimeoutError If the job does not finish in time

AsyncProcessUnsuccessfulError If the job errored or was aborted

refresh()

Update this object with the latest job data from the server.

wait_for_completion (*max_wait=600*)

Waits for job to complete.

Parameters

max_wait [int, optional] How long to wait for the job to finish.

```
class datarobot.models.TrainingPredictionsJob(data, model_id, data_subset, **kwargs)
```

```
classmethod get (project_id, job_id, model_id=None, data_subset=None)
```

Fetches one training predictions job.

The resulting *TrainingPredictions* object will be annotated with *model_id* and *data_subset*.

Parameters

project_id [str] The identifier of the project in which the job resides

job_id [str] The job id

model_id [str] The identifier of the model used for computing training predictions

data_subset [dr.enums.DATA_SUBSET, optional] Data subset used for computing training predictions

Returns

job [TrainingPredictionsJob] The job

```
refresh ()
```

Update this object with the latest job data from the server.

```
cancel ()
```

Cancel this job. If this job has not finished running, it will be removed and canceled.

```
get_result (params=None)
```

Parameters

params [dict or None] Query parameters to be added to request to get results.

For featureEffects and featureFit, source param is required to define source, otherwise the default is ‘training’

Returns

result [object]

Return type depends on the job type:

- for model jobs, a Model is returned
- for predict jobs, a pandas.DataFrame (with predictions) is returned
- for featureImpact jobs, a list of dicts by default (see *with_metadata* parameter of the *FeatureImpactJob* class and its *get()* method).
- for primeRulesets jobs, a list of Rulesets
- for primeModel jobs, a PrimeModel
- for primeDownloadValidation jobs, a PrimeFile
- for reasonCodesInitialization jobs, a ReasonCodesInitialization
- for reasonCodes jobs, a ReasonCodes
- for predictionExplanationInitialization jobs, a PredictionExplanationsInitialization
- for predictionExplanations jobs, a PredictionExplanations
- for featureEffects, a FeatureEffects
- for featureFit, a FeatureFit

Raises

JobNotFinished If the job is not finished, the result is not available.

AsyncProcessUnsuccessfulError If the job errored or was aborted

get_result_when_complete (*max_wait=600, params=None*)

Parameters

max_wait [int, optional] How long to wait for the job to finish.

params [dict, optional] Query parameters to be added to request.

Returns

result: object Return type is the same as would be returned by *Job.get_result*.

Raises

AsyncTimeoutError If the job does not finish in time

AsyncProcessUnsuccessfulError If the job errored or was aborted

wait_for_completion (*max_wait=600*)

Waits for job to complete.

Parameters

max_wait [int, optional] How long to wait for the job to finish.

class datarobot.models.**ShapMatrixJob** (*data, model_id, dataset_id, **kwargs*)

classmethod **get** (*project_id, job_id, model_id=None, dataset_id=None*)

Fetches one SHAP matrix job.

Parameters

project_id [str] The identifier of the project in which the job resides

job_id [str] The job identifier

model_id [str] The identifier of the model used for computing prediction explanations

dataset_id [str] The identifier of the dataset against which prediction explanations should be computed

Returns

job [ShapMatrixJob] The job

Raises

AsyncFailureError Querying this resource gave a status code other than 200 or 303

refresh ()

Update this object with the latest job data from the server.

cancel ()

Cancel this job. If this job has not finished running, it will be removed and canceled.

get_result (*params=None*)

Parameters

params [dict or None] Query parameters to be added to request to get results.

For featureEffects and featureFit, source param is required to define source,

otherwise the default is ‘training’

Returns

result [object]

Return type depends on the job type:

- for model jobs, a Model is returned
- for predict jobs, a pandas.DataFrame (with predictions) is returned
- for featureImpact jobs, a list of dicts by default (see `with_metadata` parameter of the `FeatureImpactJob` class and its `get()` method).
- for primeRulesets jobs, a list of Rulesets
- for primeModel jobs, a PrimeModel
- for primeDownloadValidation jobs, a PrimeFile
- for reasonCodesInitialization jobs, a ReasonCodesInitialization
- for reasonCodes jobs, a ReasonCodes
- for predictionExplanationInitialization jobs, a PredictionExplanationsInitialization
- for predictionExplanations jobs, a PredictionExplanations
- for featureEffects, a FeatureEffects
- for featureFit, a FeatureFit

Raises

JobNotFinished If the job is not finished, the result is not available.

AsyncProcessUnsuccessfulError If the job errored or was aborted

get_result_when_complete (*max_wait=600, params=None*)

Parameters

max_wait [int, optional] How long to wait for the job to finish.

params [dict, optional] Query parameters to be added to request.

Returns

result: object Return type is the same as would be returned by `Job.get_result`.

Raises

AsyncTimeoutError If the job does not finish in time

AsyncProcessUnsuccessfulError If the job errored or was aborted

wait_for_completion (*max_wait=600*)

Waits for job to complete.

Parameters

max_wait [int, optional] How long to wait for the job to finish.

class `datarobot.models.FeatureImpactJob` (*data,* *completed_resource_url=None,*
with_metadata=False)

Custom Feature Impact job to handle different return value structures.

The original implementation had just the the data and the new one also includes some metadata.

In general, we aim to keep the number of Job classes low by just utilizing the *job_type* attribute to control any specific formatting; however in this case when we needed to support a new representation with the *_same_job_type*, customizing the behavior of *_make_result_from_location* allowed us to achieve our ends without complicating the *_make_result_from_json* method.

classmethod `get` (*project_id*, *job_id*, *with_metadata=False*)

Fetches one job.

Parameters

project_id [str] The identifier of the project in which the job resides

job_id [str] The job id

with_metadata [bool] To make this job return the metadata (i.e. the full object of the completed resource) set the *with_metadata* flag to True.

Returns

job [Job] The job

Raises

AsyncFailureError Querying this resource gave a status code other than 200 or 303

cancel ()

Cancel this job. If this job has not finished running, it will be removed and canceled.

get_result (*params=None*)

Parameters

params [dict or None] Query parameters to be added to request to get results.

For featureEffects and featureFit, source param is required to define source, otherwise the default is ‘training’

Returns

result [object]

Return type depends on the job type:

- for model jobs, a Model is returned
- for predict jobs, a pandas.DataFrame (with predictions) is returned
- for featureImpact jobs, a list of dicts by default (see *with_metadata* parameter of the *FeatureImpactJob* class and its *get ()* method).
- for primeRulesets jobs, a list of Rulesets
- for primeModel jobs, a PrimeModel
- for primeDownloadValidation jobs, a PrimeFile
- for reasonCodesInitialization jobs, a ReasonCodesInitialization
- for reasonCodes jobs, a ReasonCodes
- for predictionExplanationInitialization jobs, a PredictionExplanationsInitialization
- for predictionExplanations jobs, a PredictionExplanations
- for featureEffects, a FeatureEffects
- for featureFit, a FeatureFit

Raises

JobNotFinished If the job is not finished, the result is not available.

AsyncProcessUnsuccessfulError If the job errored or was aborted

get_result_when_complete (*max_wait=600, params=None*)

Parameters

max_wait [int, optional] How long to wait for the job to finish.

params [dict, optional] Query parameters to be added to request.

Returns

result: object Return type is the same as would be returned by *Job.get_result*.

Raises

AsyncTimeoutError If the job does not finish in time

AsyncProcessUnsuccessfulError If the job errored or was aborted

refresh ()

Update this object with the latest job data from the server.

wait_for_completion (*max_wait=600*)

Waits for job to complete.

Parameters

max_wait [int, optional] How long to wait for the job to finish.

32.18 Lift Chart

class datarobot.models.lift_chart.**LiftChart** (*source, bins, source_model_id, target_class*)
Lift chart data for model.

Notes

LiftChartBin is a dict containing the following:

- **actual** (float) Sum of actual target values in bin
- **predicted** (float) Sum of predicted target values in bin
- **bin_weight** (float) The weight of the bin. For weighted projects, it is the sum of the weights of the rows in the bin. For unweighted projects, it is the number of rows in the bin.

Attributes

source [str] Lift chart data source. Can be `validation`, `crossValidation` or `holdout`.

bins [list of dict] List of dicts with schema described as `LiftChartBin` above.

source_model_id [str] ID of the model this lift chart represents; in some cases, insights from the parent of a frozen model may be used

target_class [str, optional] For multiclass lift - target class for this lift chart data.

32.19 Missing Values Report

class datarobot.models.missing_report.**MissingValuesReport** (*missing_values_report*)
Missing values report for model, contains list of reports per feature sorted by missing count in descending order.

Notes

Report per feature contains:

- `feature` : feature name.
- `type` : feature type – ‘‘Numeric’’ or ‘‘Categorical’’.
- `missing_count` : missing values count in training data.
- `missing_percentage` : missing values percentage in training data.
- `tasks` : list of information per each task, which was applied to feature.

task information contains:

- `id` : a number of task in the blueprint diagram.
- `name` : task name.
- `descriptions` : human readable aggregated information about how the task handles missing values. The following descriptions may be present: what value is imputed for missing values, whether the feature being missing is treated as a feature by the task, whether missing values are treated as infrequent values, whether infrequent values are treated as missing values, and whether missing values are ignored.

classmethod `get` (*project_id*, *model_id*)
Retrieve a missing report.

Parameters

project_id [str] The project’s id.

model_id [str] The model’s id.

Returns

MissingValuesReport The queried missing report.

32.20 Models

32.20.1 Model

```
class datarobot.models.Model (id=None, processes=None, featurelist_name=None, featurelist_id=None, project_id=None, sample_pct=None, training_row_count=None, training_duration=None, training_start_date=None, training_end_date=None, model_type=None, model_category=None, is_frozen=None, blueprint_id=None, metrics=None, project=None, data=None, monotonic_increasing_featurelist_id=None, monotonic_decreasing_featurelist_id=None, supports_monotonic_constraints=None, is_starred=None, prediction_threshold=None, prediction_threshold_read_only=None, model_number=None, parent_model_id=None, use_project_settings=None)
```

A model trained on a project's dataset capable of making predictions

All durations are specified with a duration string such as those returned by the [partitioning_methods.construct_duration_string](#) helper method. Please see [datetime partitioned project documentation](#) for more information on duration strings.

Attributes

id [str] the id of the model

project_id [str] the id of the project the model belongs to

processes [list of str] the processes used by the model

featurelist_name [str] the name of the featurelist used by the model

featurelist_id [str] the id of the featurelist used by the model

sample_pct [float or None] the percentage of the project dataset used in training the model. If the project uses datetime partitioning, the sample_pct will be None. See *training_row_count*, *training_duration*, and *training_start_date* and *training_end_date* instead.

training_row_count [int or None] the number of rows of the project dataset used in training the model. In a datetime partitioned project, if specified, defines the number of rows used to train the model and evaluate backtest scores; if unspecified, either *training_duration* or *training_start_date* and *training_end_date* was used to determine that instead.

training_duration [str or None] only present for models in datetime partitioned projects. If specified, a duration string specifying the duration spanned by the data used to train the model and evaluate backtest scores.

training_start_date [datetime or None] only present for frozen models in datetime partitioned projects. If specified, the start date of the data used to train the model.

training_end_date [datetime or None] only present for frozen models in datetime partitioned projects. If specified, the end date of the data used to train the model.

model_type [str] what model this is, e.g. "Nystroem Kernel SVM Regressor"

model_category [str] what kind of model this is - "prime" for DataRobot Prime models, "blend" for blender models, and "model" for other models

is_frozen [bool] whether this model is a frozen model

blueprint_id [str] the id of the blueprint used in this model

metrics [dict] a mapping from each metric to the model's scores for that metric

monotonic_increasing_featurelist_id [str] optional, the id of the featurelist that defines the set of features with a monotonically increasing relationship to the target. If None, no such constraints are enforced.

monotonic_decreasing_featurelist_id [str] optional, the id of the featurelist that defines the set of features with a monotonically decreasing relationship to the target. If None, no such constraints are enforced.

supports_monotonic_constraints [bool] optional, whether this model supports enforcing monotonic constraints

is_starred [bool] whether this model marked as starred

prediction_threshold [float] for binary classification projects, the threshold used for predictions

prediction_threshold_read_only [bool] indicated whether modification of the prediction threshold is forbidden. Threshold modification is forbidden once a model has had a deployment created or predictions made via the dedicated prediction API.

model_number [integer] model number assigned to a model

parent_model_id [str or None] (New in version v2.20) the id of the model that tuning parameters are derived from

use_project_settings [bool or None] (New in version v2.20) Only present for models in datetime-partitioned projects. If `True`, indicates that the custom backtest partitioning settings specified by the user were used to train the model and evaluate backtest scores.

classmethod **get** (*project, model_id*)

Retrieve a specific model.

Parameters

project [str] The project's id.

model_id [str] The `model_id` of the leaderboard item to retrieve.

Returns

model [Model] The queried instance.

Raises

ValueError passed `project` parameter value is of not supported type

classmethod **fetch_resource_data** (*url, join_endpoint=True*)

(Deprecated.) Used to acquire model data directly from its url.

Consider using `get` instead, as this is a convenience function used for development of datarobot

Parameters

url [str] The resource we are acquiring

join_endpoint [boolean, optional] Whether the client's endpoint should be joined to the URL before sending the request. Location headers are returned as absolute locations, so `will_not_need` the endpoint

Returns

model_data [dict] The queried model's data

get_features_used()

Query the server to determine which features were used.

Note that the data returned by this method is possibly different than the names of the features in the featurelist used by this model. This method will return the raw features that must be supplied in order for predictions to be generated on a new set of data. The featurelist, in contrast, would also include the names of derived features.

Returns

features [list of str] The names of the features used in the model.

get_supported_capabilities()

Retrieves a summary of the capabilities supported by a model.

New in version v2.14.

Returns

supportsBlending: bool whether the model supports blending

supportsMonotonicConstraints: bool whether the model supports monotonic constraints

hasWordCloud: bool whether the model has word cloud data available

eligibleForPrime: bool whether the model is eligible for Prime

hasParameters: bool whether the model has parameters that can be retrieved

supportsCodeGeneration: bool (New in version v2.18) whether the model supports code generation

supportsShap: bool

(New in version v2.18) True if the model supports Shapley package. i.e. Shapley based feature Importance

supportsEarlyStopping: bool (New in version v2.22) True if this is an early stopping tree-based model and number of trained iterations can be retrieved.

get_num_iterations_trained()

Retrieves the number of estimators trained by early-stopping tree-based models

– versionadded:: v2.22

Returns

projectId: str id of project containing the model

modelId: str id of the model

data: array list of *numEstimatorsItem* objects, one for each modeling stage.

numEstimatorsItem will be of the form:

stage: str indicates the modeling stage (for multi-stage models); None of single-stage models

numIterations: int the number of estimators or iterations trained by the model

delete()

Delete a model from the project's leaderboard.

get_leaderboard_ui_permalink()**Returns**

url [str] Permanent static hyperlink to this model at leaderboard.

open_model_browser()

Opens model at project leaderboard in web browser.

Note: If text-mode browsers are used, the calling process will block until the user exits the browser.

train (*sample_pct=None*, *featurelist_id=None*, *scoring_type=None*, *training_row_count=None*, *monotonic_increasing_featurelist_id=<object object>*, *monotonic_decreasing_featurelist_id=<object object>*)

Train the blueprint used in model on a particular featurelist or amount of data.

This method creates a new training job for worker and appends it to the end of the queue for this project. After the job has finished you can get the newly trained model by retrieving it from the project leaderboard, or by retrieving the result of the job.

Either *sample_pct* or *training_row_count* can be used to specify the amount of data to use, but not both. If neither are specified, a default of the maximum amount of data that can safely be used to train any blueprint without going into the validation data will be selected.

In smart-sampled projects, *sample_pct* and *training_row_count* are assumed to be in terms of rows of the minority class.

Note: For datetime partitioned projects, see *train_datetime* instead.

Parameters

sample_pct [float, optional] The amount of data to use for training, as a percentage of the project dataset from 0 to 100.

featurelist_id [str, optional] The identifier of the featurelist to use. If not defined, the featurelist of this model is used.

scoring_type [str, optional] Either `SCORING_TYPE.validation` or `SCORING_TYPE.cross_validation`. `SCORING_TYPE.validation` is available for every partitioning type, and indicates that the default model validation should be used for the project. If the project uses a form of cross-validation partitioning, `SCORING_TYPE.cross_validation` can also be used to indicate that all of the available training/validation combinations should be used to evaluate the model.

training_row_count [int, optional] The number of rows to use to train the requested model.

monotonic_increasing_featurelist_id [str] (new in version 2.11) optional, the id of the featurelist that defines the set of features with a monotonically increasing relationship to the target. Passing `None` disables increasing monotonicity constraint. Default (`dr.enums.MONOTONICITY_FEATURELIST_DEFAULT`) is the one specified by the blueprint.

monotonic_decreasing_featurelist_id [str] (new in version 2.11) optional, the id of the featurelist that defines the set of features with a monotonically decreasing relationship to the target. Passing `None` disables decreasing monotonicity constraint. Default (`dr.enums.MONOTONICITY_FEATURELIST_DEFAULT`) is the one specified by the blueprint.

Returns

model_job_id [str] id of created job, can be used as parameter to `ModelJob.get` method or `wait_for_async_model_creation` function

Examples

```
project = Project.get('p-id')
model = Model.get('p-id', 'l-id')
model_job_id = model.train(training_row_count=project.max_train_rows)
```

train_datetime (*featurelist_id=None, training_row_count=None, training_duration=None, time_window_sample_pct=None, monotonic_increasing_featurelist_id=<object object>, monotonic_decreasing_featurelist_id=<object object>, use_project_settings=False*)

Train this model on a different featurelist or amount of data

Requires that this model is part of a datetime partitioned project; otherwise, an error will occur.

All durations should be specified with a duration string such as those returned by the [partitioning_methods.construct_duration_string](#) helper method. Please see [datetime partitioned project documentation](#) for more information on duration strings.

Parameters

featurelist_id [str, optional] the featurelist to use to train the model. If not specified, the featurelist of this model is used.

training_row_count [int, optional] the number of rows of data that should be used to train the model. If specified, neither `training_duration` nor `use_project_settings` may be specified.

training_duration [str, optional] a duration string specifying what time range the data used to train the model should span. If specified, neither `training_row_count` nor `use_project_settings` may be specified.

use_project_settings [bool, optional] (New in version v2.20) defaults to `False`. If `True`, indicates that the custom backtest partitioning settings specified by the user will be used to train the model and evaluate backtest scores. If specified, neither `training_row_count` nor `training_duration` may be specified.

time_window_sample_pct [int, optional] may only be specified when the requested model is a time window (e.g. duration or start and end dates). An integer between 1 and 99 indicating the percentage to sample by within the window. The points kept are determined by a random uniform sample. If specified, `training_duration` must be specified otherwise, the number of rows used to train the model and evaluate backtest scores and an error will occur.

monotonic_increasing_featurelist_id [str, optional] (New in version v2.18) optional, the id of the featurelist that defines the set of features with a monotonically increasing relationship to the target. Passing `None` disables increasing monotonicity constraint. Default (`dr.enums.MONOTONICITY_FEATURELIST_DEFAULT`) is the one specified by the blueprint.

monotonic_decreasing_featurelist_id [str, optional] (New in version v2.18) optional, the id of the featurelist that defines the set of features with a monotonically decreasing relationship to the target. Passing `None` disables decreasing monotonicity constraint. Default (`dr.enums.MONOTONICITY_FEATURELIST_DEFAULT`) is the one specified by the blueprint.

Returns

job [ModelJob] the created job to build the model

retrain (*sample_pct=None, featurelist_id=None, training_row_count=None*)
Submit a job to the queue to train a blender model.

Parameters

sample_pct: **str, optional** The sample size in percents (1 to 100) to use in training. If this parameter is used then `training_row_count` should not be given.

featurelist_id [str, optional] The featurelist id

training_row_count [str, optional] The number of rows to train the model. If this parameter is used then `sample_pct` should not be given.

Returns

job [ModelJob] The created job that is retraining the model

request_predictions (*dataset_id*, *include_prediction_intervals=None*, *prediction_intervals_size=None*, *forecast_point=None*, *predictions_start_date=None*, *predictions_end_date=None*, *actual_value_column=None*, *explanation_algorithm=None*, *max_explanations=None*)

Request predictions against a previously uploaded dataset

Parameters

dataset_id [string] The dataset to make predictions against (as uploaded from `Project.upload_dataset`)

include_prediction_intervals [bool, optional] (New in v2.16) For *time series* projects only. Specifies whether prediction intervals should be calculated for this request. Defaults to True if `prediction_intervals_size` is specified, otherwise defaults to False.

prediction_intervals_size [int, optional] (New in v2.16) For *time series* projects only. Represents the percentile to use for the size of the prediction intervals. Defaults to 80 if `include_prediction_intervals` is True. Prediction intervals size must be between 1 and 100 (inclusive).

forecast_point [datetime.datetime or None, optional] (New in version v2.20) For time series projects only. This is the default point relative to which predictions will be generated, based on the forecast window of the project. See the time series *[prediction documentation](#)* for more information.

predictions_start_date [datetime.datetime or None, optional] (New in version v2.20) For time series projects only. The start date for bulk predictions. Note that this parameter is for generating historical predictions using the training data. This parameter should be provided in conjunction with `predictions_end_date`. Can't be provided with the `forecast_point` parameter.

predictions_end_date [datetime.datetime or None, optional] (New in version v2.20) For time series projects only. The end date for bulk predictions, exclusive. Note that this parameter is for generating historical predictions using the training data. This parameter should be provided in conjunction with `predictions_start_date`. Can't be provided with the `forecast_point` parameter.

actual_value_column [string, optional] (New in version v2.21) For time series unsupervised projects only. Actual value column can be used to calculate the classification metrics and insights on the prediction dataset. Can't be provided with the `forecast_point` parameter.

explanation_algorithm: (New in version v2.21) optional; If set to `'shap'`, the response will include prediction explanations based on the SHAP explainer (SHapley Additive exPlanations). Defaults to null (no prediction explanations).

max_explanations: (New in version v2.21) optional; specifies the maximum number of explanation values that should be returned for each row, ordered by absolute value, greatest to least. If null, no limit. In the case of `shap`: if the number of features is greater than the limit, the sum of remaining values will also be returned as `shapRemainingTotal`. Defaults to null. Cannot be set if `explanation_algorithm` is omitted.

Returns

job [PredictJob] The job computing the predictions

get_feature_impact (*with_metadata=False*)

Retrieve the computed Feature Impact results, a measure of the relevance of each feature in the model.

Feature Impact is computed for each column by creating new data with that column randomly permuted (but the others left unchanged), and seeing how the error metric score for the predictions is affected. The `impactUnnormalized` is how much worse the error metric score is when making predictions on this modified data. The `impactNormalized` is normalized so that the largest value is 1. In both cases, larger values indicate more important features.

If a feature is a redundant feature, i.e. once other features are considered it doesn't contribute much in addition, the `redundantWith` value is the name of feature that has the highest correlation with this feature. Note that redundancy detection is only available for jobs run after the addition of this feature. When retrieving data that predates this functionality, a `NoRedundancyImpactAvailable` warning will be used.

Elsewhere this technique is sometimes called `Permutation Importance`.

Requires that Feature Impact has already been computed with `request_feature_impact`.

Parameters

with_metadata [bool] The flag indicating if the result should include the metadata as well.

Returns

list or dict The feature impact data response depends on the `with_metadata` parameter. The response is either a dict with metadata and a list with actual data or just a list with that data.

Each List item is a dict with the keys `featureName`, `impactNormalized`, and `impactUnnormalized`, `redundantWith` and `count`.

For dict response available keys are:

- **featureImpacts** - Feature Impact data as a dictionary. Each item is a dict with keys: `featureName`, `impactNormalized`, and `impactUnnormalized`, and `redundantWith`.
- **shapBased** - A boolean that indicates whether Feature Impact was calculated using Shapley values.
- **ranRedundancyDetection** - A boolean that indicates whether redundant feature identification was run while calculating this Feature Impact.
- **rowCount** - An integer or None that indicates the number of rows that was used to calculate Feature Impact. For the Feature Impact calculated with the default logic, without specifying the `rowCount`, we return None here.
- `count` - An integer with the number of features under the `featureImpacts`.

Raises

ClientError (404) If the feature impacts have not been computed.

get_multiclass_feature_impact()

For multiclass it's possible to calculate feature impact separately for each target class. The method for calculation is exactly the same, calculated in one-vs-all style for each target class.

Requires that Feature Impact has already been computed with [request_feature_impact](#).

Returns

feature_impacts [list of dict] The feature impact data. Each item is a dict with the keys `featureImpacts` (list), `class` (str). Each item in `featureImpacts` is a dict with the keys `featureName`, `impactNormalized`, and `impactUnnormalized`, and `redundantWith`.

Raises

ClientError (404) If the multiclass feature impacts have not been computed.

request_feature_impact (*row_count=None, with_metadata=False*)

Request feature impacts to be computed for the model.

See [get_feature_impact](#) for more information on the result of the job.

Parameters

row_count [int] The sample size (specified in rows) to use for Feature Impact computation. This is not supported for unsupervised, multi-class (that has a separate method) and time series projects.

Returns

job [Job] A Job representing the feature impact computation. To get the completed feature impact data, use `job.get_result` or `job.get_result_when_complete`.

Raises

JobAlreadyRequested (422) If the feature impacts have already been requested.

request_external_test (*dataset_id, actual_value_column=None*)

Request external test to compute scores and insights on an external test dataset

Parameters

dataset_id [string] The dataset to make predictions against (as uploaded from `Project.upload_dataset`)

actual_value_column [string, optional] (New in version v2.21) For time series unsupervised projects only. Actual value column can be used to calculate the classification metrics and insights on the prediction dataset. Can't be provided with the `forecast_point` parameter.

Returns

•

job [Job] a Job representing external dataset insights computation

get_or_request_feature_impact (*max_wait=600, **kwargs*)

Retrieve feature impact for the model, requesting a job if it hasn't been run previously

Parameters

max_wait [int, optional] The maximum time to wait for a requested feature impact job to complete before erroring

****kwargs** Arbitrary keyword arguments passed to [request_feature_impact](#).

Returns

feature_impacts [list or dict] The feature impact data. See [get_feature_impact](#) for the exact schema.

get_feature_effect_metadata()

Retrieve Feature Effect metadata. Response contains status and available model sources.

- Feature Fit of *training* is always available (except for the old project which supports only Feature Fit for *validation*).
- When a model is trained into *validation* or *holdout* without stacked prediction (e.g. no out-of-sample prediction in *validation* or *holdout*), Feature Effect is not available for *validation* or *holdout*.
- Feature Effect for *holdout* is not available when there is no holdout configured for the project.

source is expected parameter to retrieve Feature Effect. One of provided sources shall be used.

Returns

feature_effect_metadata: FeatureEffectMetadata

get_feature_fit_metadata()

Retrieve Feature Fit metadata. Response contains status and available model sources.

- Feature Fit of *training* is always available (except for the old project which supports only Feature Fit for *validation*).
- When a model is trained into *validation* or *holdout* without stacked prediction (e.g. no out-of-sample prediction in *validation* or *holdout*), Feature Fit is not available for *validation* or *holdout*.
- Feature Fit for *holdout* is not available when there is no holdout configured for the project.

source is expected parameter to retrieve Feature Fit. One of provided sources shall be used.

Returns

feature_effect_metadata: FeatureFitMetadata

request_feature_effect(row_count=None)

Request feature effects to be computed for the model.

See [get_feature_effect](#) for more information on the result of the job.

Parameters

row_count [int] (New in version v2.21) The sample size to use for Feature Impact computation. Minimum is 10 rows. Maximum is 100000 rows or the training sample size of the model, whichever is less.

Returns

job [Job] A Job representing the feature effect computation. To get the completed feature effect data, use *job.get_result* or *job.get_result_when_complete*.

Raises

JobAlreadyRequested (422) If the feature effect have already been requested.

get_feature_effect (*source*)

Retrieve Feature Effects for the model.

Feature Effects provides partial dependence and predicted vs actual values for top-500 features ordered by feature impact score.

The partial dependence shows marginal effect of a feature on the target variable after accounting for the average effects of all other predictive features. It indicates how, holding all other variables except the feature of interest as they were, the value of this feature affects your prediction.

Requires that Feature Effects has already been computed with [request_feature_effect](#).

See [get_feature_effect_metadata](#) for retrieving information the available sources.

Parameters

source [string] The source Feature Effects are retrieved for.

Returns

feature_effects [FeatureEffects] The feature effects data.

Raises

ClientError (404) If the feature effects have not been computed or source is not valid value.

get_or_request_feature_effect (*source*, *max_wait=600*, *row_count=None*)

Retrieve feature effect for the model, requesting a job if it hasn't been run previously

See [get_feature_effect_metadata](#) for retrieving information of source.

Parameters

max_wait [int, optional] The maximum time to wait for a requested feature effect job to complete before erroring

row_count [int, optional] (New in version v2.21) The sample size to use for Feature Impact computation. Minimum is 10 rows. Maximum is 100000 rows or the training sample size of the model, whichever is less.

source [string] The source Feature Effects are retrieved for.

Returns

feature_effects [FeatureEffects] The feature effects data.

request_feature_fit ()

Request feature fit to be computed for the model.

See [get_feature_effect](#) for more information on the result of the job.

Returns

job [Job] A Job representing the feature fit computation. To get the completed feature fit data, use [job.get_result](#) or [job.get_result_when_complete](#).

Raises

JobAlreadyRequested (422) If the feature effect have already been requested.

get_feature_fit (*source*)

Retrieve Feature Fit for the model.

Feature Fit provides partial dependence and predicted vs actual values for top-500 features ordered by feature importance score.

The partial dependence shows marginal effect of a feature on the target variable after accounting for the average effects of all other predictive features. It indicates how, holding all other variables except the feature of interest as they were, the value of this feature affects your prediction.

Requires that Feature Fit has already been computed with `request_feature_effect`.

See `get_feature_fit_metadata` for retrieving information the available sources.

Parameters

source [string] The source Feature Fit are retrieved for. One value of [FeatureFitMetadata.sources].

Returns

feature_fit [FeatureFit] The feature fit data.

Raises

ClientError (404) If the feature fit have not been computed or source is not valid value.

get_or_request_feature_fit (*source*, *max_wait*=600)

Retrieve feature fit for the model, requesting a job if it hasn't been run previously

See `get_feature_fit_metadata` for retrieving information of source.

Parameters

max_wait [int, optional] The maximum time to wait for a requested feature fit job to complete before erroring

source [string] The source Feature Fit are retrieved for. One value of [FeatureFitMetadata.sources].

Returns

feature_effects [FeatureFit] The feature fit data.

get_prime_eligibility ()

Check if this model can be approximated with DataRobot Prime

Returns

prime_eligibility [dict] a dict indicating whether a model can be approximated with DataRobot Prime (key *can_make_prime*) and why it may be ineligible (key *message*)

request_approximation ()

Request an approximation of this model using DataRobot Prime

This will create several rulesets that could be used to approximate this model. After comparing their scores and rule counts, the code used in the approximation can be downloaded and run locally.

Returns

job [Job] the job generating the rulesets

get_rulesets ()

List the rulesets approximating this model generated by DataRobot Prime

If this model hasn't been approximated yet, will return an empty list. Note that these are rulesets approximating this model, not rulesets used to construct this model.

Returns

rulesets [list of Ruleset]

download_export (*filepath*)

Download an exportable model file for use in an on-premise DataRobot standalone prediction environment.

This function can only be used if model export is enabled, and will only be useful if you have an on-premise environment in which to import it.

Parameters

filepath [str] The path at which to save the exported model file.

request_transferable_export (*prediction_intervals_size=None*)

Request generation of an exportable model file for use in an on-premise DataRobot standalone prediction environment.

This function can only be used if model export is enabled, and will only be useful if you have an on-premise environment in which to import it.

This function does not download the exported file. Use `download_export` for that.

Parameters

prediction_intervals_size [int, optional] (New in v2.19) For *time series* projects only. Represents the percentile to use for the size of the prediction intervals. Prediction intervals size must be between 1 and 100 (inclusive).

Examples

```
model = datarobot.Model.get('p-id', 'l-id')
job = model.request_transferable_export()
job.wait_for_completion()
model.download_export('my_exported_model.drmodel')

# Client must be configured to use standalone prediction server for import:
datarobot.Client(token='my-token-at-standalone-server',
                 endpoint='standalone-server-url/api/v2')

imported_model = datarobot.ImportedModel.create('my_exported_model.drmodel')
```

request_frozen_model (*sample_pct=None, training_row_count=None*)

Train a new frozen model with parameters from this model

Note: This method only works if project the model belongs to is *not* datetime partitioned. If it is, use `request_frozen_datetime_model` instead.

Frozen models use the same tuning parameters as their parent model instead of independently optimizing them to allow efficiently retraining models on larger amounts of the training data.

Parameters

sample_pct [float] optional, the percentage of the dataset to use with the model. If not provided, will use the value from this model.

training_row_count [int] (New in version v2.9) optional, the integer number of rows of the dataset to use with the model. Only one of *sample_pct* and *training_row_count* should be specified.

Returns

model_job [ModelJob] the modeling job training a frozen model

request_frozen_datetime_model (*training_row_count=None, training_duration=None, training_start_date=None, training_end_date=None, time_window_sample_pct=None*)

Train a new frozen model with parameters from this model

Requires that this model belongs to a datetime partitioned project. If it does not, an error will occur when submitting the job.

Frozen models use the same tuning parameters as their parent model instead of independently optimizing them to allow efficiently retraining models on larger amounts of the training data.

In addition of `training_row_count` and `training_duration`, frozen datetime models may be trained on an exact date range. Only one of `training_row_count`, `training_duration`, or `training_start_date` and `training_end_date` should be specified.

Models specified using `training_start_date` and `training_end_date` are the only ones that can be trained into the holdout data (once the holdout is unlocked).

All durations should be specified with a duration string such as those returned by the [partitioning_methods.construct_duration_string](#) helper method. Please see [datetime partitioned project documentation](#) for more information on duration strings.

Parameters

training_row_count [int, optional] the number of rows of data that should be used to train the model. If specified, `training_duration` may not be specified.

training_duration [str, optional] a duration string specifying what time range the data used to train the model should span. If specified, `training_row_count` may not be specified.

training_start_date [datetime.datetime, optional] the start date of the data to train to model on. Only rows occurring at or after this datetime will be used. If `training_start_date` is specified, `training_end_date` must also be specified.

training_end_date [datetime.datetime, optional] the end date of the data to train the model on. Only rows occurring strictly before this datetime will be used. If `training_end_date` is specified, `training_start_date` must also be specified.

time_window_sample_pct [int, optional] may only be specified when the requested model is a time window (e.g. duration or start and end dates). An integer between 1 and 99 indicating the percentage to sample by within the window. The points kept are determined by a random uniform sample. If specified, `training_duration` must be specified otherwise, the number of rows used to train the model and evaluate backtest scores and an error will occur.

Returns

model_job [ModelJob] the modeling job training a frozen model

get_parameters ()

Retrieve model parameters.

Returns

ModelParameters Model parameters for this model.

get_lift_chart (*source, fallback_to_parent_insights=False*)

Retrieve model lift chart for the specified source.

Parameters

source [str] Lift chart data source. Check `datarobot.enums.CHART_DATA_SOURCE` for possible values.

fallback_to_parent_insights [bool] (New in version v2.14) Optional, if True, this will return lift chart data for this model's parent if the lift chart is not available for this model and the model has a defined parent model. If omitted or False, or there is no parent model, will not attempt to return insight data from this model's parent.

Returns

LiftChart Model lift chart data

Raises

ClientError If the insight is not available for this model

get_multiclass_lift_chart (*source, fallback_to_parent_insights=False*)

Retrieve model lift chart for the specified source.

Parameters

source [str] Lift chart data source. Check `datarobot.enums.CHART_DATA_SOURCE` for possible values.

fallback_to_parent_insights [bool] Optional, if True, this will return lift chart data for this model's parent if the lift chart is not available for this model and the model has a defined parent model. If omitted or False, or there is no parent model, will not attempt to return insight data from this model's parent.

Returns

list of LiftChart Model lift chart data for each saved target class

Raises

ClientError If the insight is not available for this model

get_all_lift_charts (*fallback_to_parent_insights=False*)

Retrieve a list of all lift charts available for the model.

Parameters

fallback_to_parent_insights [bool] (New in version v2.14) Optional, if True, this will return lift chart data for this model's parent for any source that is not available for this model and if this model has a defined parent model. If omitted or False, or this model has no parent, this will not attempt to retrieve any data from this model's parent.

Returns

list of LiftChart Data for all available model lift charts.

get_residuals_chart (*source, fallback_to_parent_insights=False*)

Retrieve model residuals chart for the specified source.

Parameters

source [str] Residuals chart data source. Check `datarobot.enums.CHART_DATA_SOURCE` for possible values.

fallback_to_parent_insights [bool] Optional, if True, this will return residuals chart data for this model's parent if the residuals chart is not available for this model and the model has a defined parent model. If omitted or False, or there is no parent model, will not attempt to return residuals data from this model's parent.

Returns

ResidualsChart Model residuals chart data

Raises

ClientError If the insight is not available for this model

get_all_residuals_charts (*fallback_to_parent_insights=False*)
Retrieve a list of all lift charts available for the model.

Parameters

fallback_to_parent_insights [bool] Optional, if True, this will return residuals chart data for this model's parent for any source that is not available for this model and if this model has a defined parent model. If omitted or False, or this model has no parent, this will not attempt to retrieve any data from this model's parent.

Returns

list of ResidualsChart Data for all available model residuals charts.

get_pareto_front ()
Retrieve the Pareto Front for a Eureqa model.
This method is only supported for Eureqa models.

Returns

ParetoFront Model ParetoFront data

get_confusion_chart (*source, fallback_to_parent_insights=False*)
Retrieve model's confusion chart for the specified source.

Parameters

source [str] Confusion chart source. Check datarobot.enums.CHART_DATA_SOURCE for possible values.

fallback_to_parent_insights [bool] (New in version v2.14) Optional, if True, this will return confusion chart data for this model's parent if the confusion chart is not available for this model and the defined parent model. If omitted or False, or there is no parent model, will not attempt to return insight data from this model's parent.

Returns

ConfusionChart Model ConfusionChart data

Raises

ClientError If the insight is not available for this model

get_all_confusion_charts (*fallback_to_parent_insights=False*)
Retrieve a list of all confusion charts available for the model.

Parameters

fallback_to_parent_insights [bool] (New in version v2.14) Optional, if True, this will return confusion chart data for this model's parent for any source that is not available for this model and if this has a defined parent model. If omitted or False, or this model has no parent, this will not attempt to retrieve any data from this model's parent.

Returns

list of ConfusionChart Data for all available confusion charts for model.

get_roc_curve (*source, fallback_to_parent_insights=False*)
Retrieve model ROC curve for the specified source.

Parameters

source [str] ROC curve data source. Check `datarobot.enums.CHART_DATA_SOURCE` for possible values.

fallback_to_parent_insights [bool] (New in version v2.14) Optional, if True, this will return ROC curve data for this model's parent if the ROC curve is not available for this model and the model has a defined parent model. If omitted or False, or there is no parent model, will not attempt to return data from this model's parent.

Returns

RocCurve Model ROC curve data

Raises

ClientError If the insight is not available for this model

get_all_roc_curves (*fallback_to_parent_insights=False*)

Retrieve a list of all ROC curves available for the model.

Parameters

fallback_to_parent_insights [bool] (New in version v2.14) Optional, if True, this will return ROC curve data for this model's parent for any source that is not available for this model and if this model has a defined parent model. If omitted or False, or this model has no parent, this will not attempt to retrieve any data from this model's parent.

Returns

list of RocCurve Data for all available model ROC curves.

get_word_cloud (*exclude_stop_words=False*)

Retrieve a word cloud data for the model.

Parameters

exclude_stop_words [bool, optional] Set to True if you want stopwords filtered out of response.

Returns

WordCloud Word cloud data for the model.

download_scoring_code (*file_name, source_code=False*)

Download scoring code JAR.

Parameters

file_name [str] File path where scoring code will be saved.

source_code [bool, optional] Set to True to download source code archive. It will not be executable.

get_model_blueprint_documents ()

Get documentation for tasks used in this model.

Returns

list of BlueprintTaskDocument All documents available for the model.

get_model_blueprint_chart ()

Retrieve a model blueprint chart that can be used to understand data flow in blueprint.

Returns

ModelBlueprintChart The queried model blueprint chart.

get_missing_report_info()

Retrieve a model missing data report on training data that can be used to understand missing values treatment in a model. Report consists of missing values reports for features which took part in modelling and are numeric or categorical.

Returns

An iterable of MissingReportPerFeature The queried model missing report, sorted by missing count (DESCENDING order).

get_frozen_child_models()

Retrieves the ids for all the models that are frozen from this model

Returns

A list of Models

request_training_predictions (*data_subset*, *explanation_algorithm=None*, *max_explanations=None*)

Start a job to build training predictions

Parameters

data_subset [str] data set definition to build predictions on. Choices are:

- *dr.enums.DATA_SUBSET.ALL* or string *all* for all data available. Not valid for models in datetime partitioned projects
- *dr.enums.DATA_SUBSET.VALIDATION_AND_HOLDOUT* or string *validationAndHoldout* for all data except training set. Not valid for models in datetime partitioned projects
- *dr.enums.DATA_SUBSET.HOLDOUT* or string *holdout* for holdout data set only
- *dr.enums.DATA_SUBSET.ALL_BACKTESTS* or string *allBacktests* for downloading the predictions for all backtest validation folds. Requires the model to have successfully scored all backtests. Datetime partitioned projects only.

explanation_algorithm [dr.enums.EXPLANATIONS_ALGORITHM] (New in v2.21) Optional. If set to *dr.enums.EXPLANATIONS_ALGORITHM.SHAP*, the response will include prediction explanations based on the SHAP explainer (SHapley Additive exPlanations). Defaults to *None* (no prediction explanations).

max_explanations [int] (New in v2.21) Optional. Specifies the maximum number of explanation values that should be returned for each row, ordered by absolute value, greatest to least. In the case of *dr.enums.EXPLANATIONS_ALGORITHM.SHAP*: If not set, explanations are returned for all features. If the number of features is greater than the *max_explanations*, the sum of remaining values will also be returned as *shap_remaining_total*. Max 100. Defaults to null for datasets narrower than 100 columns, defaults to 100 for datasets wider than 100 columns. Is ignored if *explanation_algorithm* is not set.

Returns

Job an instance of created async job

cross_validate()

Run Cross Validation on this model.

Note: To perform Cross Validation on a new model with new parameters, use `train` instead.

Returns

ModelJob The created job to build the model

get_cross_validation_scores (*partition=None, metric=None*)

Returns a dictionary keyed by metric showing cross validation scores per partition.

Cross Validation should already have been performed using `cross_validate` or `train`.

Note: Models that computed cross validation before this feature was added will need to be deleted and retrained before this method can be used.

Parameters

partition [float] optional, the id of the partition (1,2,3,0,4,0,etc) to filter results by can be a whole number positive integer or float value.

metric: unicode optional name of the metric to filter to resulting cross validation scores by

Returns

cross_validation_scores: dict A dictionary keyed by metric showing cross validation scores per partition.

advanced_tune (*params, description=None*)

Generate a new model with the specified advanced-tuning parameters

As of v2.17, all models other than blenders, open source, prime, scaleout, baseline and user-created support Advanced Tuning.

Parameters

params [dict] Mapping of parameter ID to parameter value. The list of valid parameter IDs for a model can be found by calling `get_advanced_tuning_parameters()`. This endpoint does not need to include values for all parameters. If a parameter is omitted, its *current_value* will be used.

description [unicode] Human-readable string describing the newly advanced-tuned model

Returns

ModelJob The created job to build the model

get_advanced_tuning_parameters ()

Get the advanced-tuning parameters available for this model.

As of v2.17, all models other than blenders, open source, prime, scaleout, baseline and user-created support Advanced Tuning.

Returns

dict A dictionary describing the advanced-tuning parameters for the current model. There are two top-level keys, *tuningDescription* and *tuningParameters*.

tuningDescription an optional value. If not *None*, then it indicates the user-specified description of this set of tuning parameter.

tuningParameters is a list of a dicts, each has the following keys

- **parameterName** : (**unicode**) name of the parameter (unique per task, see below)
- **parameterId** : (**unicode**) opaque ID string uniquely identifying parameter
- **defaultValue** : (*) default value of the parameter for the blueprint

- `currentValue` : (*) value of the parameter that was used for this model
- `taskName` : (**unicode**) name of the task that this parameter belongs to
- `constraints`: (**dict**) see the notes below

Notes

The type of `defaultValue` and `currentValue` is defined by the `constraints` structure. It will be a string or numeric Python type.

`constraints` is a dict with *at least one*, possibly more, of the following keys. The presence of a key indicates that the parameter may take on the specified type. (If a key is absent, this means that the parameter may not take on the specified type.) If a key on `constraints` is present, its value will be a dict containing all of the fields described below for that key.

```
"constraints": {
  "select": {
    "values": [<list (basestring or number) : possible values>]
  },
  "ascii": {},
  "unicode": {},
  "int": {
    "min": <int : minimum valid value>,
    "max": <int : maximum valid value>,
    "supports_grid_search": <bool : True if Grid Search may be
                             requested for this param>
  },
  "float": {
    "min": <float : minimum valid value>,
    "max": <float : maximum valid value>,
    "supports_grid_search": <bool : True if Grid Search may be
                             requested for this param>
  },
  "intList": {
    "length": {
      "min_length": <int : minimum valid length>,
      "max_length": <int : maximum valid length>
    },
    "min_val": <int : minimum valid value>,
    "max_val": <int : maximum valid value>
    "supports_grid_search": <bool : True if Grid Search may be
                             requested for this param>
  },
  "floatList": {
    "min_length": <int : minimum valid length>,
    "max_length": <int : maximum valid length>
    "min_val": <float : minimum valid value>,
    "max_val": <float : maximum valid value>
    "supports_grid_search": <bool : True if Grid Search may be
                             requested for this param>
  }
}
```

The keys have meaning as follows:

- `select`: Rather than specifying a specific data type, if present, it indicates that the parameter is permitted to take on any of the specified values. Listed values may be of any string or real (non-complex) numeric type.

- *ascii*: The parameter may be a *unicode* object that encodes simple ASCII characters. (A-Z, a-z, 0-9, whitespace, and certain common symbols.) In addition to listed constraints, ASCII keys currently may not contain either newlines or semicolons.
- *unicode*: The parameter may be any Python *unicode* object.
- *int*: The value may be an object of type *int* within the specified range (inclusive). Please note that the value will be passed around using the JSON format, and some JSON parsers have undefined behavior with integers outside of the range $[-(2^{53})+1, (2^{53})-1]$.
- *float*: The value may be an object of type *float* within the specified range (inclusive).
- *intList*, *floatList*: The value may be a list of *int* or *float* objects, respectively, following constraints as specified respectively by the *int* and *float* types (above).

Many parameters only specify one key under *constraints*. If a parameter specifies multiple keys, the parameter may take on any value permitted by any key.

start_advanced_tuning_session()

Start an Advanced Tuning session. Returns an object that helps set up arguments for an Advanced Tuning model execution.

As of v2.17, all models other than blenders, open source, prime, scaleout, baseline and user-created support Advanced Tuning.

Returns

AdvancedTuningSession Session for setting up and running Advanced Tuning on a model

star_model()

Mark the model as starred

Model stars propagate to the web application and the API, and can be used to filter when listing models.

unstar_model()

Unmark the model as starred

Model stars propagate to the web application and the API, and can be used to filter when listing models.

set_prediction_threshold(threshold)

Set a custom prediction threshold for the model

May not be used once `prediction_threshold_read_only` is `True` for this model.

Parameters

threshold [float] only used for binary classification projects. The threshold to when deciding between the positive and negative classes when making predictions. Should be between 0.0 and 1.0 (inclusive).

32.20.2 PrimeModel

```
class datarobot.models.PrimeModel (id=None, processes=None, featurelist_name=None, featurelist_id=None, project_id=None, sample_pct=None, training_row_count=None, training_duration=None, training_start_date=None, training_end_date=None, model_type=None, model_category=None, is_frozen=None, blueprint_id=None, metrics=None, parent_model_id=None, ruleset_id=None, rule_count=None, score=None, monotonic_increasing_featurelist_id=None, monotonic_decreasing_featurelist_id=None, supports_monotonic_constraints=None, is_starred=None, prediction_threshold=None, prediction_threshold_read_only=None, model_number=None)
```

A DataRobot Prime model approximating a parent model with downloadable code

All durations are specified with a duration string such as those returned by the [partitioning_methods.construct_duration_string](#) helper method. Please see [datetime partitioned project documentation](#) for more information on duration strings.

Attributes

- id** [str] the id of the model
- project_id** [str] the id of the project the model belongs to
- processes** [list of str] the processes used by the model
- featurelist_name** [str] the name of the featurelist used by the model
- featurelist_id** [str] the id of the featurelist used by the model
- sample_pct** [float] the percentage of the project dataset used in training the model
- training_row_count** [int or None] the number of rows of the project dataset used in training the model. In a datetime partitioned project, if specified, defines the number of rows used to train the model and evaluate backtest scores; if unspecified, either *training_duration* or *training_start_date* and *training_end_date* was used to determine that instead.
- training_duration** [str or None] only present for models in datetime partitioned projects. If specified, a duration string specifying the duration spanned by the data used to train the model and evaluate backtest scores.
- training_start_date** [datetime or None] only present for frozen models in datetime partitioned projects. If specified, the start date of the data used to train the model.
- training_end_date** [datetime or None] only present for frozen models in datetime partitioned projects. If specified, the end date of the data used to train the model.
- model_type** [str] what model this is, e.g. "DataRobot Prime"
- model_category** [str] what kind of model this is - always "prime" for DataRobot Prime models
- is_frozen** [bool] whether this model is a frozen model
- blueprint_id** [str] the id of the blueprint used in this model
- metrics** [dict] a mapping from each metric to the model's scores for that metric
- ruleset** [Ruleset] the ruleset used in the Prime model
- parent_model_id** [str] the id of the model that this Prime model approximates

monotonic_increasing_featurelist_id [str] optional, the id of the featurelist that defines the set of features with a monotonically increasing relationship to the target. If None, no such constraints are enforced.

monotonic_decreasing_featurelist_id [str] optional, the id of the featurelist that defines the set of features with a monotonically decreasing relationship to the target. If None, no such constraints are enforced.

supports_monotonic_constraints [bool] optional, whether this model supports enforcing monotonic constraints

is_starred [bool] whether this model is marked as starred

prediction_threshold [float] for binary classification projects, the threshold used for predictions

prediction_threshold_read_only [bool] indicated whether modification of the prediction threshold is forbidden. Threshold modification is forbidden once a model has had a deployment created or predictions made via the dedicated prediction API.

classmethod **get** (*project_id*, *model_id*)

Retrieve a specific prime model.

Parameters

project_id [str] The id of the project the prime model belongs to

model_id [str] The `model_id` of the prime model to retrieve.

Returns

model [PrimeModel] The queried instance.

request_download_validation (*language*)

Prep and validate the downloadable code for the ruleset associated with this model

Parameters

language [str] the language the code should be downloaded in - see `datarobot.enums.PRIME_LANGUAGE` for available languages

Returns

job [Job] A job tracking the code preparation and validation

advanced_tune (*params*, *description=None*)

Generate a new model with the specified advanced-tuning parameters

As of v2.17, all models other than blenders, open source, prime, scaleout, baseline and user-created support Advanced Tuning.

Parameters

params [dict] Mapping of parameter ID to parameter value. The list of valid parameter IDs for a model can be found by calling `get_advanced_tuning_parameters()`. This endpoint does not need to include values for all parameters. If a parameter is omitted, its `current_value` will be used.

description [unicode] Human-readable string describing the newly advanced-tuned model

Returns

ModelJob The created job to build the model

cross_validate()

Run Cross Validation on this model.

Note: To perform Cross Validation on a new model with new parameters, use `train` instead.

Returns

ModelJob The created job to build the model

delete()

Delete a model from the project's leaderboard.

download_export(filepath)

Download an exportable model file for use in an on-premise DataRobot standalone prediction environment.

This function can only be used if model export is enabled, and will only be useful if you have an on-premise environment in which to import it.

Parameters

filepath [str] The path at which to save the exported model file.

download_scoring_code(file_name, source_code=False)

Download scoring code JAR.

Parameters

file_name [str] File path where scoring code will be saved.

source_code [bool, optional] Set to True to download source code archive. It will not be executable.

classmethod fetch_resource_data(url, join_endpoint=True)

(Deprecated.) Used to acquire model data directly from its url.

Consider using `get` instead, as this is a convenience function used for development of datarobot

Parameters

url [str] The resource we are acquiring

join_endpoint [boolean, optional] Whether the client's endpoint should be joined to the URL before sending the request. Location headers are returned as absolute locations, so `will_not_need` the endpoint

Returns

model_data [dict] The queried model's data

get_advanced_tuning_parameters()

Get the advanced-tuning parameters available for this model.

As of v2.17, all models other than blenders, open source, prime, scaleout, baseline and user-created support Advanced Tuning.

Returns

dict A dictionary describing the advanced-tuning parameters for the current model. There are two top-level keys, *tuningDescription* and *tuningParameters*.

tuningDescription an optional value. If not *None*, then it indicates the user-specified description of this set of tuning parameter.

tuningParameters is a list of a dicts, each has the following keys

- *parameterName* : (**unicode**) name of the parameter (unique per task, see below)
- *parameterId* : (**unicode**) opaque ID string uniquely identifying parameter
- *defaultValue* : (*) default value of the parameter for the blueprint
- *currentValue* : (*) value of the parameter that was used for this model
- *taskName* : (**unicode**) name of the task that this parameter belongs to
- *constraints* : (**dict**) see the notes below

Notes

The type of *defaultValue* and *currentValue* is defined by the *constraints* structure. It will be a string or numeric Python type.

constraints is a dict with *at least one*, possibly more, of the following keys. The presence of a key indicates that the parameter may take on the specified type. (If a key is absent, this means that the parameter may not take on the specified type.) If a key on *constraints* is present, its value will be a dict containing all of the fields described below for that key.

```
"constraints": {
  "select": {
    "values": [<list(basestring or number) : possible values>]
  },
  "ascii": {},
  "unicode": {},
  "int": {
    "min": <int : minimum valid value>,
    "max": <int : maximum valid value>,
    "supports_grid_search": <bool : True if Grid Search may be
                           requested for this param>
  },
  "float": {
    "min": <float : minimum valid value>,
    "max": <float : maximum valid value>,
    "supports_grid_search": <bool : True if Grid Search may be
                           requested for this param>
  },
  "intList": {
    "length": {
      "min_length": <int : minimum valid length>,
      "max_length": <int : maximum valid length>
    },
    "min_val": <int : minimum valid value>,
    "max_val": <int : maximum valid value>
    "supports_grid_search": <bool : True if Grid Search may be
                           requested for this param>
  },
  "floatList": {
    "min_length": <int : minimum valid length>,
    "max_length": <int : maximum valid length>
    "min_val": <float : minimum valid value>,
    "max_val": <float : maximum valid value>
    "supports_grid_search": <bool : True if Grid Search may be
                           requested for this param>
  }
}
```

(continues on next page)

(continued from previous page)

```
}
}
```

The keys have meaning as follows:

- *select*: Rather than specifying a specific data type, if present, it indicates that the parameter is permitted to take on any of the specified values. Listed values may be of any string or real (non-complex) numeric type.
- *ascii*: The parameter may be a *unicode* object that encodes simple ASCII characters. (A-Z, a-z, 0-9, whitespace, and certain common symbols.) In addition to listed constraints, ASCII keys currently may not contain either newlines or semicolons.
- *unicode*: The parameter may be any Python *unicode* object.
- *int*: The value may be an object of type *int* within the specified range (inclusive). Please note that the value will be passed around using the JSON format, and some JSON parsers have undefined behavior with integers outside of the range $[-(2^{53})+1, (2^{53})-1]$.
- *float*: The value may be an object of type *float* within the specified range (inclusive).
- *intList*, *floatList*: The value may be a list of *int* or *float* objects, respectively, following constraints as specified respectively by the *int* and *float* types (above).

Many parameters only specify one key under *constraints*. If a parameter specifies multiple keys, the parameter may take on any value permitted by any key.

get_all_confusion_charts (*fallback_to_parent_insights=False*)

Retrieve a list of all confusion charts available for the model.

Parameters

fallback_to_parent_insights [bool] (New in version v2.14) Optional, if True, this will return confusion chart data for this model's parent for any source that is not available for this model and if this has a defined parent model. If omitted or False, or this model has no parent, this will not attempt to retrieve any data from this model's parent.

Returns

list of ConfusionChart Data for all available confusion charts for model.

get_all_lift_charts (*fallback_to_parent_insights=False*)

Retrieve a list of all lift charts available for the model.

Parameters

fallback_to_parent_insights [bool] (New in version v2.14) Optional, if True, this will return lift chart data for this model's parent for any source that is not available for this model and if this model has a defined parent model. If omitted or False, or this model has no parent, this will not attempt to retrieve any data from this model's parent.

Returns

list of LiftChart Data for all available model lift charts.

get_all_residuals_charts (*fallback_to_parent_insights=False*)

Retrieve a list of all lift charts available for the model.

Parameters

fallback_to_parent_insights [bool] Optional, if True, this will return residuals chart data for this model's parent for any source that is not available for this model and if this

model has a defined parent model. If omitted or False, or this model has no parent, this will not attempt to retrieve any data from this model's parent.

Returns

list of ResidualsChart Data for all available model residuals charts.

get_all_roc_curves (*fallback_to_parent_insights=False*)

Retrieve a list of all ROC curves available for the model.

Parameters

fallback_to_parent_insights [bool] (New in version v2.14) Optional, if True, this will return ROC curve data for this model's parent for any source that is not available for this model and if this model has a defined parent model. If omitted or False, or this model has no parent, this will not attempt to retrieve any data from this model's parent.

Returns

list of RocCurve Data for all available model ROC curves.

get_confusion_chart (*source, fallback_to_parent_insights=False*)

Retrieve model's confusion chart for the specified source.

Parameters

source [str] Confusion chart source. Check `datarobot.enums.CHART_DATA_SOURCE` for possible values.

fallback_to_parent_insights [bool] (New in version v2.14) Optional, if True, this will return confusion chart data for this model's parent if the confusion chart is not available for this model and the defined parent model. If omitted or False, or there is no parent model, will not attempt to return insight data from this model's parent.

Returns

ConfusionChart Model ConfusionChart data

Raises

ClientError If the insight is not available for this model

get_cross_validation_scores (*partition=None, metric=None*)

Returns a dictionary keyed by metric showing cross validation scores per partition.

Cross Validation should already have been performed using `cross_validate` or `train`.

Note: Models that computed cross validation before this feature was added will need to be deleted and retrained before this method can be used.

Parameters

partition [float] optional, the id of the partition (1,2,3.0,4.0,etc) to filter results by can be a whole number positive integer or float value.

metric: unicode optional name of the metric to filter to resulting cross validation scores by

Returns

cross_validation_scores: dict A dictionary keyed by metric showing cross validation scores per partition.

get_feature_effect (*source*)

Retrieve Feature Effects for the model.

Feature Effects provides partial dependence and predicted vs actual values for top-500 features ordered by feature impact score.

The partial dependence shows marginal effect of a feature on the target variable after accounting for the average effects of all other predictive features. It indicates how, holding all other variables except the feature of interest as they were, the value of this feature affects your prediction.

Requires that Feature Effects has already been computed with [request_feature_effect](#).

See [get_feature_effect_metadata](#) for retrieving information the available sources.

Parameters

source [string] The source Feature Effects are retrieved for.

Returns

feature_effects [FeatureEffects] The feature effects data.

Raises

ClientError (404) If the feature effects have not been computed or source is not valid value.

get_feature_effect_metadata ()

Retrieve Feature Effect metadata. Response contains status and available model sources.

- Feature Fit of *training* is always available (except for the old project which supports only Feature Fit for *validation*).
- When a model is trained into *validation* or *holdout* without stacked prediction (e.g. no out-of-sample prediction in *validation* or *holdout*), Feature Effect is not available for *validation* or *holdout*.
- Feature Effect for *holdout* is not available when there is no holdout configured for the project.

source is expected parameter to retrieve Feature Effect. One of provided sources shall be used.

Returns

feature_effect_metadata: FeatureEffectMetadata

get_feature_fit (*source*)

Retrieve Feature Fit for the model.

Feature Fit provides partial dependence and predicted vs actual values for top-500 features ordered by feature importance score.

The partial dependence shows marginal effect of a feature on the target variable after accounting for the average effects of all other predictive features. It indicates how, holding all other variables except the feature of interest as they were, the value of this feature affects your prediction.

Requires that Feature Fit has already been computed with [request_feature_effect](#).

See [get_feature_fit_metadata](#) for retrieving information the available sources.

Parameters

source [string] The source Feature Fit are retrieved for. One value of [FeatureFitMetadata.sources].

Returns

feature_fit [FeatureFit] The feature fit data.

Raises

ClientError (404) If the feature fit have not been computed or source is not valid value.

get_feature_fit_metadata()

Retrieve Feature Fit metadata. Response contains status and available model sources.

- Feature Fit of *training* is always available (except for the old project which supports only Feature Fit for *validation*).
- When a model is trained into *validation* or *holdout* without stacked prediction (e.g. no out-of-sample prediction in *validation* or *holdout*), Feature Fit is not available for *validation* or *holdout*.
- Feature Fit for *holdout* is not available when there is no holdout configured for the project.

source is expected parameter to retrieve Feature Fit. One of provided sources shall be used.

Returns

feature_effect_metadata: FeatureFitMetadata

get_feature_impact (*with_metadata=False*)

Retrieve the computed Feature Impact results, a measure of the relevance of each feature in the model.

Feature Impact is computed for each column by creating new data with that column randomly permuted (but the others left unchanged), and seeing how the error metric score for the predictions is affected. The *impactUnnormalized* is how much worse the error metric score is when making predictions on this modified data. The *impactNormalized* is normalized so that the largest value is 1. In both cases, larger values indicate more important features.

If a feature is a redundant feature, i.e. once other features are considered it doesn't contribute much in addition, the *redundantWith* value is the name of feature that has the highest correlation with this feature. Note that redundancy detection is only available for jobs run after the addition of this feature. When retrieving data that predates this functionality, a *NoRedundancyImpactAvailable* warning will be used.

Elsewhere this technique is sometimes called *Permutation Importance*.

Requires that Feature Impact has already been computed with *request_feature_impact*.

Parameters

with_metadata [bool] The flag indicating if the result should include the metadata as well.

Returns

list or dict The feature impact data response depends on the *with_metadata* parameter. The response is either a dict with metadata and a list with actual data or just a list with that data.

Each List item is a dict with the keys *featureName*, *impactNormalized*, and *impactUnnormalized*, *redundantWith* and *count*.

For dict response available keys are:

- **featureImpacts - Feature Impact data as a dictionary. Each item is a dict with keys:** *featureName*, *impactNormalized*, and *impactUnnormalized*, and *redundantWith*.

- **shapBased** - A boolean that indicates whether Feature Impact was calculated using Shapley values.
- **ranRedundancyDetection** - A boolean that indicates whether redundant feature identification was run while calculating this Feature Impact.
- **rowCount** - An integer or None that indicates the number of rows that was used to calculate Feature Impact. For the Feature Impact calculated with the default logic, without specifying the rowCount, we return None here.
- **count** - An integer with the number of features under the `featureImpacts`.

Raises

ClientError (404) If the feature impacts have not been computed.

get_features_used()

Query the server to determine which features were used.

Note that the data returned by this method is possibly different than the names of the features in the featurelist used by this model. This method will return the raw features that must be supplied in order for predictions to be generated on a new set of data. The featurelist, in contrast, would also include the names of derived features.

Returns

features [list of str] The names of the features used in the model.

get_frozen_child_models()

Retrieves the ids for all the models that are frozen from this model

Returns

A list of Models

get_leaderboard_ui_permalink()**Returns**

url [str] Permanent static hyperlink to this model at leaderboard.

get_lift_chart(source, fallback_to_parent_insights=False)

Retrieve model lift chart for the specified source.

Parameters

source [str] Lift chart data source. Check `datarobot.enums.CHART_DATA_SOURCE` for possible values.

fallback_to_parent_insights [bool] (New in version v2.14) Optional, if True, this will return lift chart data for this model's parent if the lift chart is not available for this model and the model has a defined parent model. If omitted or False, or there is no parent model, will not attempt to return insight data from this model's parent.

Returns

LiftChart Model lift chart data

Raises

ClientError If the insight is not available for this model

get_missing_report_info()

Retrieve a model missing data report on training data that can be used to understand missing values treatment in a model. Report consists of missing values reports for features which took part in modelling and are numeric or categorical.

Returns

An iterable of MissingReportPerFeature The queried model missing report, sorted by missing count (DESCENDING order).

get_model_blueprint_chart()

Retrieve a model blueprint chart that can be used to understand data flow in blueprint.

Returns

ModelBlueprintChart The queried model blueprint chart.

get_model_blueprint_documents()

Get documentation for tasks used in this model.

Returns

list of BlueprintTaskDocument All documents available for the model.

get_multiclass_feature_impact()

For multiclass it's possible to calculate feature impact separately for each target class. The method for calculation is exactly the same, calculated in one-vs-all style for each target class.

Requires that Feature Impact has already been computed with `request_feature_impact`.

Returns

feature_impacts [list of dict] The feature impact data. Each item is a dict with the keys `featureImpacts` (list), `class` (str). Each item in `featureImpacts` is a dict with the keys `featureName`, `impactNormalized`, and `impactUnnormalized`, and `redundantWith`.

Raises

ClientError (404) If the multiclass feature impacts have not been computed.

get_multiclass_lift_chart(source, fallback_to_parent_insights=False)

Retrieve model lift chart for the specified source.

Parameters

source [str] Lift chart data source. Check `datarobot.enums.CHART_DATA_SOURCE` for possible values.

fallback_to_parent_insights [bool] Optional, if True, this will return lift chart data for this model's parent if the lift chart is not available for this model and the model has a defined parent model. If omitted or False, or there is no parent model, will not attempt to return insight data from this model's parent.

Returns

list of LiftChart Model lift chart data for each saved target class

Raises

ClientError If the insight is not available for this model

get_num_iterations_trained()

Retrieves the number of estimators trained by early-stopping tree-based models

– versionadded:: v2.22

Returns

projectId: str id of project containing the model

modelId: str id of the model

data: array list of *numEstimatorsItem* objects, one for each modeling stage.

numEstimatorsItem will be of the form:

stage: str indicates the modeling stage (for multi-stage models); None of single-stage models

numIterations: int the number of estimators or iterations trained by the model

get_or_request_feature_effect (*source*, *max_wait=600*, *row_count=None*)

Retrieve feature effect for the model, requesting a job if it hasn't been run previously

See [get_feature_effect_metadata](#) for retrieving information of source.

Parameters

max_wait [int, optional] The maximum time to wait for a requested feature effect job to complete before erroring

row_count [int, optional] (New in version v2.21) The sample size to use for Feature Impact computation. Minimum is 10 rows. Maximum is 100000 rows or the training sample size of the model, whichever is less.

source [string] The source Feature Effects are retrieved for.

Returns

feature_effects [FeatureEffects] The feature effects data.

get_or_request_feature_fit (*source*, *max_wait=600*)

Retrieve feature fit for the model, requesting a job if it hasn't been run previously

See [get_feature_fit_metadata](#) for retrieving information of source.

Parameters

max_wait [int, optional] The maximum time to wait for a requested feature fit job to complete before erroring

source [string] The source Feature Fit are retrieved for. One value of [FeatureFitMetadata.sources].

Returns

feature_effects [FeatureFit] The feature fit data.

get_or_request_feature_impact (*max_wait=600*, ***kwargs*)

Retrieve feature impact for the model, requesting a job if it hasn't been run previously

Parameters

max_wait [int, optional] The maximum time to wait for a requested feature impact job to complete before erroring

****kwargs** Arbitrary keyword arguments passed to [request_feature_impact](#).

Returns

feature_impacts [list or dict] The feature impact data. See [get_feature_impact](#) for the exact schema.

get_parameters ()

Retrieve model parameters.

Returns

ModelParameters Model parameters for this model.

get_pareto_front()

Retrieve the Pareto Front for a Eureqa model.

This method is only supported for Eureqa models.

Returns

ParetoFront Model ParetoFront data

get_prime_eligibility()

Check if this model can be approximated with DataRobot Prime

Returns

prime_eligibility [dict] a dict indicating whether a model can be approximated with DataRobot Prime (key *can_make_prime*) and why it may be ineligible (key *message*)

get_residuals_chart(source, fallback_to_parent_insights=False)

Retrieve model residuals chart for the specified source.

Parameters

source [str] Residuals chart data source. Check `datarobot.enums.CHART_DATA_SOURCE` for possible values.

fallback_to_parent_insights [bool] Optional, if True, this will return residuals chart data for this model's parent if the residuals chart is not available for this model and the model has a defined parent model. If omitted or False, or there is no parent model, will not attempt to return residuals data from this model's parent.

Returns

ResidualsChart Model residuals chart data

Raises

ClientError If the insight is not available for this model

get_roc_curve(source, fallback_to_parent_insights=False)

Retrieve model ROC curve for the specified source.

Parameters

source [str] ROC curve data source. Check `datarobot.enums.CHART_DATA_SOURCE` for possible values.

fallback_to_parent_insights [bool] (New in version v2.14) Optional, if True, this will return ROC curve data for this model's parent if the ROC curve is not available for this model and the model has a defined parent model. If omitted or False, or there is no parent model, will not attempt to return data from this model's parent.

Returns

RocCurve Model ROC curve data

Raises

ClientError If the insight is not available for this model

get_rulesets()

List the rulesets approximating this model generated by DataRobot Prime

If this model hasn't been approximated yet, will return an empty list. Note that these are rulesets approximating this model, not rulesets used to construct this model.

Returns

rulesets [list of Ruleset]

get_supported_capabilities ()

Retrieves a summary of the capabilities supported by a model.

New in version v2.14.

Returns

supportsBlending: bool whether the model supports blending

supportsMonotonicConstraints: bool whether the model supports monotonic constraints

hasWordCloud: bool whether the model has word cloud data available

eligibleForPrime: bool whether the model is eligible for Prime

hasParameters: bool whether the model has parameters that can be retrieved

supportsCodeGeneration: bool (New in version v2.18) whether the model supports code generation

supportsShap: bool

(New in version v2.18) True if the model supports Shapley package. i.e. Shapley based feature Importance

supportsEarlyStopping: bool (New in version v2.22) True if this is an early stopping tree-based model and number of trained iterations can be retrieved.

get_word_cloud (*exclude_stop_words=False*)

Retrieve a word cloud data for the model.

Parameters

exclude_stop_words [bool, optional] Set to True if you want stopwords filtered out of response.

Returns

WordCloud Word cloud data for the model.

open_model_browser ()

Opens model at project leaderboard in web browser.

Note: If text-mode browsers are used, the calling process will block until the user exits the browser.

request_external_test (*dataset_id, actual_value_column=None*)

Request external test to compute scores and insights on an external test dataset

Parameters

dataset_id [string] The dataset to make predictions against (as uploaded from Project.upload_dataset)

actual_value_column [string, optional] (New in version v2.21) For time series unsupervised projects only. Actual value column can be used to calculate the classification metrics and insights on the prediction dataset. Can't be provided with the `forecast_point` parameter.

Returns

Job

job [Job] a Job representing external dataset insights computation

request_feature_effect (*row_count=None*)

Request feature effects to be computed for the model.

See [*get_feature_effect*](#) for more information on the result of the job.

Parameters

row_count [int] (New in version v2.21) The sample size to use for Feature Impact computation. Minimum is 10 rows. Maximum is 100000 rows or the training sample size of the model, whichever is less.

Returns

job [Job] A Job representing the feature effect computation. To get the completed feature effect data, use *job.get_result* or *job.get_result_when_complete*.

Raises

JobAlreadyRequested (422) If the feature effect have already been requested.

request_feature_fit ()

Request feature fit to be computed for the model.

See [*get_feature_effect*](#) for more information on the result of the job.

Returns

job [Job] A Job representing the feature fit computation. To get the completed feature fit data, use *job.get_result* or *job.get_result_when_complete*.

Raises

JobAlreadyRequested (422) If the feature effect have already been requested.

request_feature_impact (*row_count=None, with_metadata=False*)

Request feature impacts to be computed for the model.

See [*get_feature_impact*](#) for more information on the result of the job.

Parameters

row_count [int] The sample size (specified in rows) to use for Feature Impact computation. This is not supported for unsupervised, multi-class (that has a separate method) and time series projects.

Returns

job [Job] A Job representing the feature impact computation. To get the completed feature impact data, use *job.get_result* or *job.get_result_when_complete*.

Raises

JobAlreadyRequested (422) If the feature impacts have already been requested.

request_predictions (*dataset_id,* *include_prediction_intervals=None,* *prediction_intervals_size=None,* *forecast_point=None,* *predictions_start_date=None,* *predictions_end_date=None,* *actual_value_column=None,* *explanation_algorithm=None,* *max_explanations=None*)

Request predictions against a previously uploaded dataset

Parameters

dataset_id [string] The dataset to make predictions against (as uploaded from *Project.upload_dataset*)

include_prediction_intervals [bool, optional] (New in v2.16) For *time series* projects only. Specifies whether prediction intervals should be calculated for this request. Defaults to True if *prediction_intervals_size* is specified, otherwise defaults to False.

prediction_intervals_size [int, optional] (New in v2.16) For *time series* projects only. Represents the percentile to use for the size of the prediction intervals. Defaults to 80 if *include_prediction_intervals* is True. Prediction intervals size must be between 1 and 100 (inclusive).

forecast_point [datetime.datetime or None, optional] (New in version v2.20) For time series projects only. This is the default point relative to which predictions will be generated, based on the forecast window of the project. See the time series *prediction documentation* for more information.

predictions_start_date [datetime.datetime or None, optional] (New in version v2.20) For time series projects only. The start date for bulk predictions. Note that this parameter is for generating historical predictions using the training data. This parameter should be provided in conjunction with *predictions_end_date*. Can't be provided with the *forecast_point* parameter.

predictions_end_date [datetime.datetime or None, optional] (New in version v2.20) For time series projects only. The end date for bulk predictions, exclusive. Note that this parameter is for generating historical predictions using the training data. This parameter should be provided in conjunction with *predictions_start_date*. Can't be provided with the *forecast_point* parameter.

actual_value_column [string, optional] (New in version v2.21) For time series unsupervised projects only. Actual value column can be used to calculate the classification metrics and insights on the prediction dataset. Can't be provided with the *forecast_point* parameter.

explanation_algorithm: (New in version v2.21) optional; If set to `shap`, the response will include prediction explanations based on the SHAP explainer (SHapley Additive exPlanations). Defaults to null (no prediction explanations).

max_explanations: (New in version v2.21) optional; specifies the maximum number of explanation values that should be returned for each row, ordered by absolute value, greatest to least. If null, no limit. In the case of `shap`: if the number of features is greater than the limit, the sum of remaining values will also be returned as *shapRemainingTotal*. Defaults to null. Cannot be set if *explanation_algorithm* is omitted.

Returns

job [PredictJob] The job computing the predictions

request_training_predictions (*data_subset*, *explanation_algorithm=None*,
max_explanations=None)

Start a job to build training predictions

Parameters

data_subset [str] data set definition to build predictions on. Choices are:

- *dr.enums.DATA_SUBSET.ALL* or string *all* for all data available. Not valid for models in datetime partitioned projects
- *dr.enums.DATA_SUBSET.VALIDATION_AND_HOLDOUT* or string *validationAndHoldout* for all data except training set. Not valid for models in datetime partitioned projects
- *dr.enums.DATA_SUBSET.HOLDOUT* or string *holdout* for holdout data set only

- *dr.enums.DATA_SUBSET_ALL_BACKTESTS* or string *allBacktests* for downloading the predictions for all backtest validation folds. Requires the model to have successfully scored all backtests. Datetime partitioned projects only.

explanation_algorithm [*dr.enums.EXPLANATIONS_ALGORITHM*] (New in v2.21) Optional. If set to *dr.enums.EXPLANATIONS_ALGORITHM.SHAP*, the response will include prediction explanations based on the SHAP explainer (SHapley Additive exPlanations). Defaults to *None* (no prediction explanations).

max_explanations [int] (New in v2.21) Optional. Specifies the maximum number of explanation values that should be returned for each row, ordered by absolute value, greatest to least. In the case of *dr.enums.EXPLANATIONS_ALGORITHM.SHAP*: If not set, explanations are returned for all features. If the number of features is greater than the *max_explanations*, the sum of remaining values will also be returned as *shap_remaining_total*. Max 100. Defaults to null for datasets narrower than 100 columns, defaults to 100 for datasets wider than 100 columns. Is ignored if *explanation_algorithm* is not set.

Returns

Job an instance of created async job

request_transferable_export (*prediction_intervals_size=None*)

Request generation of an exportable model file for use in an on-premise DataRobot standalone prediction environment.

This function can only be used if model export is enabled, and will only be useful if you have an on-premise environment in which to import it.

This function does not download the exported file. Use *download_export* for that.

Parameters

prediction_intervals_size [int, optional] (New in v2.19) For *time series* projects only. Represents the percentile to use for the size of the prediction intervals. Prediction intervals size must be between 1 and 100 (inclusive).

Examples

```
model = datarobot.Model.get('p-id', 'l-id')
job = model.request_transferable_export()
job.wait_for_completion()
model.download_export('my_exported_model.drmodel')

# Client must be configured to use standalone prediction server for import:
datarobot.Client(token='my-token-at-standalone-server',
                 endpoint='standalone-server-url/api/v2')

imported_model = datarobot.ImportedModel.create('my_exported_model.drmodel')
```

retrain (*sample_pct=None, featurelist_id=None, training_row_count=None*)

Submit a job to the queue to train a blender model.

Parameters

sample_pct: str, optional The sample size in percents (1 to 100) to use in training. If this parameter is used then *training_row_count* should not be given.

featurelist_id [str, optional] The featurelist id

training_row_count [str, optional] The number of rows to train the model. If this parameter is used then `sample_pct` should not be given.

Returns

job [ModelJob] The created job that is retraining the model

set_prediction_threshold (*threshold*)

Set a custom prediction threshold for the model

May not be used once `prediction_threshold_read_only` is True for this model.

Parameters

threshold [float] only used for binary classification projects. The threshold to when deciding between the positive and negative classes when making predictions. Should be between 0.0 and 1.0 (inclusive).

star_model ()

Mark the model as starred

Model stars propagate to the web application and the API, and can be used to filter when listing models.

start_advanced_tuning_session ()

Start an Advanced Tuning session. Returns an object that helps set up arguments for an Advanced Tuning model execution.

As of v2.17, all models other than blenders, open source, prime, scaleout, baseline and user-created support Advanced Tuning.

Returns

AdvancedTuningSession Session for setting up and running Advanced Tuning on a model

unstar_model ()

Unmark the model as starred

Model stars propagate to the web application and the API, and can be used to filter when listing models.

32.20.3 BlenderModel

```
class datarobot.models.BlenderModel (id=None, processes=None, featurelist_name=None, featurelist_id=None, project_id=None, sample_pct=None, training_row_count=None, training_duration=None, training_start_date=None, training_end_date=None, model_type=None, model_category=None, is_frozen=None, blueprint_id=None, metrics=None, model_ids=None, blender_method=None, monotonic_increasing_featurelist_id=None, monotonic_decreasing_featurelist_id=None, supports_monotonic_constraints=None, is_starred=None, prediction_threshold=None, prediction_threshold_read_only=None, model_number=None, parent_model_id=None)
```

Blender model that combines prediction results from other models.

All durations are specified with a duration string such as those returned by the `partitioning_methods.construct_duration_string` helper method. Please see [datetime partitioned project documentation](#) for more information on duration strings.

Attributes

id [str] the id of the model

project_id [str] the id of the project the model belongs to

processes [list of str] the processes used by the model

featurelist_name [str] the name of the featurelist used by the model

featurelist_id [str] the id of the featurelist used by the model

sample_pct [float] the percentage of the project dataset used in training the model

training_row_count [int or None] the number of rows of the project dataset used in training the model. In a datetime partitioned project, if specified, defines the number of rows used to train the model and evaluate backtest scores; if unspecified, either *training_duration* or *training_start_date* and *training_end_date* was used to determine that instead.

training_duration [str or None] only present for models in datetime partitioned projects. If specified, a duration string specifying the duration spanned by the data used to train the model and evaluate backtest scores.

training_start_date [datetime or None] only present for frozen models in datetime partitioned projects. If specified, the start date of the data used to train the model.

training_end_date [datetime or None] only present for frozen models in datetime partitioned projects. If specified, the end date of the data used to train the model.

model_type [str] what model this is, e.g. `“DataRobot Prime”`

model_category [str] what kind of model this is - always `“prime”` for DataRobot Prime models

is_frozen [bool] whether this model is a frozen model

blueprint_id [str] the id of the blueprint used in this model

metrics [dict] a mapping from each metric to the model’s scores for that metric

model_ids [list of str] List of model ids used in blender

blender_method [str] Method used to blend results from underlying models

monotonic_increasing_featurelist_id [str] optional, the id of the featurelist that defines the set of features with a monotonically increasing relationship to the target. If None, no such constraints are enforced.

monotonic_decreasing_featurelist_id [str] optional, the id of the featurelist that defines the set of features with a monotonically decreasing relationship to the target. If None, no such constraints are enforced.

supports_monotonic_constraints [bool] optional, whether this model supports enforcing monotonic constraints

is_starred [bool] whether this model marked as starred

prediction_threshold [float] for binary classification projects, the threshold used for predictions

prediction_threshold_read_only [bool] indicated whether modification of the prediction threshold is forbidden. Threshold modification is forbidden once a model has had a deployment created or predictions made via the dedicated prediction API.

model_number [integer] model number assigned to a model

parent_model_id [str or None] (New in version v2.20) the id of the model that tuning parameters are derived from

classmethod `get(project_id, model_id)`

Retrieve a specific blender.

Parameters

project_id [str] The project's id.

model_id [str] The `model_id` of the leaderboard item to retrieve.

Returns

model [BlenderModel] The queried instance.

advanced_tune (*params*, *description=None*)

Generate a new model with the specified advanced-tuning parameters

As of v2.17, all models other than blenders, open source, prime, scaleout, baseline and user-created support Advanced Tuning.

Parameters

params [dict] Mapping of parameter ID to parameter value. The list of valid parameter IDs for a model can be found by calling `get_advanced_tuning_parameters()`. This endpoint does not need to include values for all parameters. If a parameter is omitted, its *current_value* will be used.

description [unicode] Human-readable string describing the newly advanced-tuned model

Returns

ModelJob The created job to build the model

cross_validate ()

Run Cross Validation on this model.

Note: To perform Cross Validation on a new model with new parameters, use `train` instead.

Returns

ModelJob The created job to build the model

delete ()

Delete a model from the project's leaderboard.

download_export (*filepath*)

Download an exportable model file for use in an on-premise DataRobot standalone prediction environment.

This function can only be used if model export is enabled, and will only be useful if you have an on-premise environment in which to import it.

Parameters

filepath [str] The path at which to save the exported model file.

download_scoring_code (*file_name*, *source_code=False*)

Download scoring code JAR.

Parameters

file_name [str] File path where scoring code will be saved.

source_code [bool, optional] Set to True to download source code archive. It will not be executable.

classmethod `fetch_resource_data(url, join_endpoint=True)`

(Deprecated.) Used to acquire model data directly from its url.

Consider using `get` instead, as this is a convenience function used for development of datarobot

Parameters

url [str] The resource we are acquiring

join_endpoint [boolean, optional] Whether the client's endpoint should be joined to the URL before sending the request. Location headers are returned as absolute locations, so `will_not_need` the endpoint

Returns

model_data [dict] The queried model's data

get_advanced_tuning_parameters()

Get the advanced-tuning parameters available for this model.

As of v2.17, all models other than blenders, open source, prime, scaleout, baseline and user-created support Advanced Tuning.

Returns

dict A dictionary describing the advanced-tuning parameters for the current model. There are two top-level keys, *tuningDescription* and *tuningParameters*.

tuningDescription an optional value. If not *None*, then it indicates the user-specified description of this set of tuning parameter.

tuningParameters is a list of a dicts, each has the following keys

- **parameterName** : (**unicode**) name of the parameter (unique per task, see below)
- **parameterId** : (**unicode**) opaque ID string uniquely identifying parameter
- **defaultValue** : (*) default value of the parameter for the blueprint
- **currentValue** : (*) value of the parameter that was used for this model
- **taskName** : (**unicode**) name of the task that this parameter belongs to
- **constraints** : (**dict**) see the notes below

Notes

The type of *defaultValue* and *currentValue* is defined by the *constraints* structure. It will be a string or numeric Python type.

constraints is a dict with *at least one*, possibly more, of the following keys. The presence of a key indicates that the parameter may take on the specified type. (If a key is absent, this means that the parameter may not take on the specified type.) If a key on *constraints* is present, its value will be a dict containing all of the fields described below for that key.

```
"constraints": {
  "select": {
    "values": [<list (basestring or number) : possible values>]
  },
  "ascii": {},
  "unicode": {},
  "int": {
    "min": <int : minimum valid value>,

```

(continues on next page)

(continued from previous page)

```

    "max": <int : maximum valid value>,
    "supports_grid_search": <bool : True if Grid Search may be
                           requested for this param>
  },
  "float": {
    "min": <float : minimum valid value>,
    "max": <float : maximum valid value>,
    "supports_grid_search": <bool : True if Grid Search may be
                           requested for this param>
  },
  "intList": {
    "length": {
      "min_length": <int : minimum valid length>,
      "max_length": <int : maximum valid length>
    },
    "min_val": <int : minimum valid value>,
    "max_val": <int : maximum valid value>
    "supports_grid_search": <bool : True if Grid Search may be
                           requested for this param>
  },
  "floatList": {
    "min_length": <int : minimum valid length>,
    "max_length": <int : maximum valid length>
    "min_val": <float : minimum valid value>,
    "max_val": <float : maximum valid value>
    "supports_grid_search": <bool : True if Grid Search may be
                           requested for this param>
  }
}

```

The keys have meaning as follows:

- *select*: Rather than specifying a specific data type, if present, it indicates that the parameter is permitted to take on any of the specified values. Listed values may be of any string or real (non-complex) numeric type.
- *ascii*: The parameter may be a *unicode* object that encodes simple ASCII characters. (A-Z, a-z, 0-9, whitespace, and certain common symbols.) In addition to listed constraints, ASCII keys currently may not contain either newlines or semicolons.
- *unicode*: The parameter may be any Python *unicode* object.
- *int*: The value may be an object of type *int* within the specified range (inclusive). Please note that the value will be passed around using the JSON format, and some JSON parsers have undefined behavior with integers outside of the range $[-(2^{53})+1, (2^{53})-1]$.
- *float*: The value may be an object of type *float* within the specified range (inclusive).
- *intList*, *floatList*: The value may be a list of *int* or *float* objects, respectively, following constraints as specified respectively by the *int* and *float* types (above).

Many parameters only specify one key under *constraints*. If a parameter specifies multiple keys, the parameter may take on any value permitted by any key.

get_all_confusion_charts (*fallback_to_parent_insights=False*)

Retrieve a list of all confusion charts available for the model.

Parameters

fallback_to_parent_insights [bool] (New in version v2.14) Optional, if True, this will return confusion chart data for this model's parent for any source that is not available for

this model and if this has a defined parent model. If omitted or False, or this model has no parent, this will not attempt to retrieve any data from this model's parent.

Returns

list of ConfusionChart Data for all available confusion charts for model.

get_all_lift_charts (*fallback_to_parent_insights=False*)

Retrieve a list of all lift charts available for the model.

Parameters

fallback_to_parent_insights [bool] (New in version v2.14) Optional, if True, this will return lift chart data for this model's parent for any source that is not available for this model and if this model has a defined parent model. If omitted or False, or this model has no parent, this will not attempt to retrieve any data from this model's parent.

Returns

list of LiftChart Data for all available model lift charts.

get_all_residuals_charts (*fallback_to_parent_insights=False*)

Retrieve a list of all lift charts available for the model.

Parameters

fallback_to_parent_insights [bool] Optional, if True, this will return residuals chart data for this model's parent for any source that is not available for this model and if this model has a defined parent model. If omitted or False, or this model has no parent, this will not attempt to retrieve any data from this model's parent.

Returns

list of ResidualsChart Data for all available model residuals charts.

get_all_roc_curves (*fallback_to_parent_insights=False*)

Retrieve a list of all ROC curves available for the model.

Parameters

fallback_to_parent_insights [bool] (New in version v2.14) Optional, if True, this will return ROC curve data for this model's parent for any source that is not available for this model and if this model has a defined parent model. If omitted or False, or this model has no parent, this will not attempt to retrieve any data from this model's parent.

Returns

list of RocCurve Data for all available model ROC curves.

get_confusion_chart (*source, fallback_to_parent_insights=False*)

Retrieve model's confusion chart for the specified source.

Parameters

source [str] Confusion chart source. Check `datarobot.enums.CHART_DATA_SOURCE` for possible values.

fallback_to_parent_insights [bool] (New in version v2.14) Optional, if True, this will return confusion chart data for this model's parent if the confusion chart is not available for this model and the defined parent model. If omitted or False, or there is no parent model, will not attempt to return insight data from this model's parent.

Returns

ConfusionChart Model ConfusionChart data

Raises

ClientError If the insight is not available for this model

get_cross_validation_scores (*partition=None, metric=None*)

Returns a dictionary keyed by metric showing cross validation scores per partition.

Cross Validation should already have been performed using `cross_validate` or `train`.

Note: Models that computed cross validation before this feature was added will need to be deleted and retrained before this method can be used.

Parameters

partition [float] optional, the id of the partition (1,2,3,0,4,0,etcâ€¦) to filter results by can be a whole number positive integer or float value.

metric: unicode optional name of the metric to filter to resulting cross validation scores by

Returns

cross_validation_scores: dict A dictionary keyed by metric showing cross validation scores per partition.

get_feature_effect (*source*)

Retrieve Feature Effects for the model.

Feature Effects provides partial dependence and predicted vs actual values for top-500 features ordered by feature impact score.

The partial dependence shows marginal effect of a feature on the target variable after accounting for the average effects of all other predictive features. It indicates how, holding all other variables except the feature of interest as they were, the value of this feature affects your prediction.

Requires that Feature Effects has already been computed with `request_feature_effect`.

See `get_feature_effect_metadata` for retrieving information the available sources.

Parameters

source [string] The source Feature Effects are retrieved for.

Returns

feature_effects [FeatureEffects] The feature effects data.

Raises

ClientError (404) If the feature effects have not been computed or source is not valid value.

get_feature_effect_metadata ()

Retrieve Feature Effect metadata. Response contains status and available model sources.

- Feature Fit of *training* is always available (except for the old project which supports only Feature Fit for *validation*).
- When a model is trained into *validation* or *holdout* without stacked prediction (e.g. no out-of-sample prediction in *validation* or *holdout*), Feature Effect is not available for *validation* or *holdout*.
- Feature Effect for *holdout* is not available when there is no holdout configured for the project.

source is expected parameter to retrieve Feature Effect. One of provided sources shall be used.

Returns

feature_effect_metadata: FeatureEffectMetadata

get_feature_fit (*source*)

Retrieve Feature Fit for the model.

Feature Fit provides partial dependence and predicted vs actual values for top-500 features ordered by feature importance score.

The partial dependence shows marginal effect of a feature on the target variable after accounting for the average effects of all other predictive features. It indicates how, holding all other variables except the feature of interest as they were, the value of this feature affects your prediction.

Requires that Feature Fit has already been computed with [request_feature_effect](#).

See [get_feature_fit_metadata](#) for retrieving information the available sources.

Parameters

source [string] The source Feature Fit are retrieved for. One value of [FeatureFitMetadata.sources].

Returns

feature_fit [FeatureFit] The feature fit data.

Raises

ClientError (404) If the feature fit have not been computed or source is not valid value.

get_feature_fit_metadata ()

Retrieve Feature Fit metadata. Response contains status and available model sources.

- Feature Fit of *training* is always available (except for the old project which supports only Feature Fit for *validation*).
- When a model is trained into *validation* or *holdout* without stacked prediction (e.g. no out-of-sample prediction in *validation* or *holdout*), Feature Fit is not available for *validation* or *holdout*.
- Feature Fit for *holdout* is not available when there is no holdout configured for the project.

source is expected parameter to retrieve Feature Fit. One of provided sources shall be used.

Returns

feature_effect_metadata: FeatureFitMetadata

get_feature_impact (*with_metadata=False*)

Retrieve the computed Feature Impact results, a measure of the relevance of each feature in the model.

Feature Impact is computed for each column by creating new data with that column randomly permuted (but the others left unchanged), and seeing how the error metric score for the predictions is affected. The “*impactUnnormalized*” is how much worse the error metric score is when making predictions on this modified data. The “*impactNormalized*” is normalized so that the largest value is 1. In both cases, larger values indicate more important features.

If a feature is a redundant feature, i.e. once other features are considered it doesn’t contribute much in addition, the “*redundantWith*” value is the name of feature that has the highest correlation with this feature. Note that redundancy detection is only available for jobs run after the addition of this feature. When retrieving data that predates this functionality, a *NoRedundancyImpactAvailable* warning will be used.

Elsewhere this technique is sometimes called *Permutation Importance*.

Requires that Feature Impact has already been computed with `request_feature_impact`.

Parameters

with_metadata [bool] The flag indicating if the result should include the metadata as well.

Returns

list or dict The feature impact data response depends on the `with_metadata` parameter. The response is either a dict with metadata and a list with actual data or just a list with that data.

Each List item is a dict with the keys `featureName`, `impactNormalized`, and `impactUnnormalized`, `redundantWith` and `count`.

For dict response available keys are:

- **featureImpacts** - Feature Impact data as a dictionary. Each item is a dict with keys: `featureName`, `impactNormalized`, and `impactUnnormalized`, and `redundantWith`.
- **shapBased** - A boolean that indicates whether Feature Impact was calculated using Shapley values.
- **ranRedundancyDetection** - A boolean that indicates whether redundant feature identification was run while calculating this Feature Impact.
- **rowCount** - An integer or None that indicates the number of rows that was used to calculate Feature Impact. For the Feature Impact calculated with the default logic, without specifying the `rowCount`, we return None here.
- **count** - An integer with the number of features under the `featureImpacts`.

Raises

ClientError (404) If the feature impacts have not been computed.

`get_features_used()`

Query the server to determine which features were used.

Note that the data returned by this method is possibly different than the names of the features in the `featurelist` used by this model. This method will return the raw features that must be supplied in order for predictions to be generated on a new set of data. The `featurelist`, in contrast, would also include the names of derived features.

Returns

features [list of str] The names of the features used in the model.

`get_frozen_child_models()`

Retrieves the ids for all the models that are frozen from this model

Returns

A list of Models

`get_leaderboard_ui_permalink()`

Returns

url [str] Permanent static hyperlink to this model at leaderboard.

`get_lift_chart(source, fallback_to_parent_insights=False)`

Retrieve model lift chart for the specified source.

Parameters

source [str] Lift chart data source. Check `datarobot.enums.CHART_DATA_SOURCE` for possible values.

fallback_to_parent_insights [bool] (New in version v2.14) Optional, if True, this will return lift chart data for this model's parent if the lift chart is not available for this model and the model has a defined parent model. If omitted or False, or there is no parent model, will not attempt to return insight data from this model's parent.

Returns

LiftChart Model lift chart data

Raises

ClientError If the insight is not available for this model

get_missing_report_info()

Retrieve a model missing data report on training data that can be used to understand missing values treatment in a model. Report consists of missing values reports for features which took part in modelling and are numeric or categorical.

Returns

An iterable of MissingReportPerFeature The queried model missing report, sorted by missing count (DESCENDING order).

get_model_blueprint_chart()

Retrieve a model blueprint chart that can be used to understand data flow in blueprint.

Returns

ModelBlueprintChart The queried model blueprint chart.

get_model_blueprint_documents()

Get documentation for tasks used in this model.

Returns

list of BlueprintTaskDocument All documents available for the model.

get_multiclass_feature_impact()

For multiclass it's possible to calculate feature impact separately for each target class. The method for calculation is exactly the same, calculated in one-vs-all style for each target class.

Requires that Feature Impact has already been computed with `request_feature_impact`.

Returns

feature_impacts [list of dict] The feature impact data. Each item is a dict with the keys `featureImpacts` (list), `class` (str). Each item in `featureImpacts` is a dict with the keys `featureName`, `impactNormalized`, and `impactUnnormalized`, and `redundantWith`.

Raises

ClientError (404) If the multiclass feature impacts have not been computed.

get_multiclass_lift_chart(source, fallback_to_parent_insights=False)

Retrieve model lift chart for the specified source.

Parameters

source [str] Lift chart data source. Check `datarobot.enums.CHART_DATA_SOURCE` for possible values.

fallback_to_parent_insights [bool] Optional, if True, this will return lift chart data for this model's parent if the lift chart is not available for this model and the model has a defined parent model. If omitted or False, or there is no parent model, will not attempt to return insight data from this model's parent.

Returns

list of LiftChart Model lift chart data for each saved target class

Raises

ClientError If the insight is not available for this model

`get_num_estimators_trained()`

Retrieves the number of estimators trained by early-stopping tree-based models

– versionadded:: v2.22

Returns

projectId: str id of project containing the model

modelId: str id of the model

data: array list of *numEstimatorsItem* objects, one for each modeling stage.

numEstimatorsItem will be of the form:

stage: str indicates the modeling stage (for multi-stage models); None of single-stage models

numIterations: int the number of estimators or iterations trained by the model

`get_or_request_feature_effect(source, max_wait=600, row_count=None)`

Retrieve feature effect for the model, requesting a job if it hasn't been run previously

See [get_feature_effect_metadata](#) for retrieving information of source.

Parameters

max_wait [int, optional] The maximum time to wait for a requested feature effect job to complete before erroring

row_count [int, optional] (New in version v2.21) The sample size to use for Feature Impact computation. Minimum is 10 rows. Maximum is 100000 rows or the training sample size of the model, whichever is less.

source [string] The source Feature Effects are retrieved for.

Returns

feature_effects [FeatureEffects] The feature effects data.

`get_or_request_feature_fit(source, max_wait=600)`

Retrieve feature fit for the model, requesting a job if it hasn't been run previously

See [get_feature_fit_metadata](#) for retrieving information of source.

Parameters

max_wait [int, optional] The maximum time to wait for a requested feature fit job to complete before erroring

source [string] The source Feature Fit are retrieved for. One value of [FeatureFitMetadata.sources].

Returns

feature_effects [FeatureFit] The feature fit data.

get_or_request_feature_impact (*max_wait=600, **kwargs*)

Retrieve feature impact for the model, requesting a job if it hasn't been run previously

Parameters

max_wait [int, optional] The maximum time to wait for a requested feature impact job to complete before erroring

****kwargs** Arbitrary keyword arguments passed to [request_feature_impact](#).

Returns

feature_impacts [list or dict] The feature impact data. See [get_feature_impact](#) for the exact schema.

get_parameters ()

Retrieve model parameters.

Returns

ModelParameters Model parameters for this model.

get_pareto_front ()

Retrieve the Pareto Front for a Eureqa model.

This method is only supported for Eureqa models.

Returns

ParetoFront Model ParetoFront data

get_prime_eligibility ()

Check if this model can be approximated with DataRobot Prime

Returns

prime_eligibility [dict] a dict indicating whether a model can be approximated with DataRobot Prime (key *can_make_prime*) and why it may be ineligible (key *message*)

get_residuals_chart (*source, fallback_to_parent_insights=False*)

Retrieve model residuals chart for the specified source.

Parameters

source [str] Residuals chart data source. Check `datarobot.enums.CHART_DATA_SOURCE` for possible values.

fallback_to_parent_insights [bool] Optional, if True, this will return residuals chart data for this model's parent if the residuals chart is not available for this model and the model has a defined parent model. If omitted or False, or there is no parent model, will not attempt to return residuals data from this model's parent.

Returns

ResidualsChart Model residuals chart data

Raises

ClientError If the insight is not available for this model

get_roc_curve (*source, fallback_to_parent_insights=False*)

Retrieve model ROC curve for the specified source.

Parameters

source [str] ROC curve data source. Check `datarobot.enums.CHART_DATA_SOURCE` for possible values.

fallback_to_parent_insights [bool] (New in version v2.14) Optional, if True, this will return ROC curve data for this model's parent if the ROC curve is not available for this model and the model has a defined parent model. If omitted or False, or there is no parent model, will not attempt to return data from this model's parent.

Returns

RocCurve Model ROC curve data

Raises

ClientError If the insight is not available for this model

`get_rulesets()`

List the rulesets approximating this model generated by DataRobot Prime

If this model hasn't been approximated yet, will return an empty list. Note that these are rulesets approximating this model, not rulesets used to construct this model.

Returns

rulesets [list of Ruleset]

`get_supported_capabilities()`

Retrieves a summary of the capabilities supported by a model.

New in version v2.14.

Returns

supportsBlending: bool whether the model supports blending

supportsMonotonicConstraints: bool whether the model supports monotonic constraints

hasWordCloud: bool whether the model has word cloud data available

eligibleForPrime: bool whether the model is eligible for Prime

hasParameters: bool whether the model has parameters that can be retrieved

supportsCodeGeneration: bool (New in version v2.18) whether the model supports code generation

supportsShap: bool

(New in version v2.18) True if the model supports Shapley package. i.e. Shapley based feature Importance

supportsEarlyStopping: bool (New in version v2.22) True if this is an early stopping tree-based model and number of trained iterations can be retrieved.

`get_word_cloud(exclude_stop_words=False)`

Retrieve a word cloud data for the model.

Parameters

exclude_stop_words [bool, optional] Set to True if you want stopwords filtered out of response.

Returns

WordCloud Word cloud data for the model.

open_model_browser()

Opens model at project leaderboard in web browser.

Note: If text-mode browsers are used, the calling process will block until the user exits the browser.

request_approximation()

Request an approximation of this model using DataRobot Prime

This will create several rulesets that could be used to approximate this model. After comparing their scores and rule counts, the code used in the approximation can be downloaded and run locally.

Returns

job [Job] the job generating the rulesets

request_external_test (*dataset_id*, *actual_value_column=None*)

Request external test to compute scores and insights on an external test dataset

Parameters

dataset_id [string] The dataset to make predictions against (as uploaded from `Project.upload_dataset`)

actual_value_column [string, optional] (New in version v2.21) For time series unsupervised projects only. Actual value column can be used to calculate the classification metrics and insights on the prediction dataset. Can't be provided with the `forecast_point` parameter.

Returns

Job

job [Job] a Job representing external dataset insights computation

request_feature_effect (*row_count=None*)

Request feature effects to be computed for the model.

See [get_feature_effect](#) for more information on the result of the job.

Parameters

row_count [int] (New in version v2.21) The sample size to use for Feature Impact computation. Minimum is 10 rows. Maximum is 100000 rows or the training sample size of the model, whichever is less.

Returns

job [Job] A Job representing the feature effect computation. To get the completed feature effect data, use `job.get_result` or `job.get_result_when_complete`.

Raises

JobAlreadyRequested (422) If the feature effect have already been requested.

request_feature_fit()

Request feature fit to be computed for the model.

See [get_feature_effect](#) for more information on the result of the job.

Returns

job [Job] A Job representing the feature fit computation. To get the completed feature fit data, use `job.get_result` or `job.get_result_when_complete`.

Raises

JobAlreadyRequested (422) If the feature effect have already been requested.

request_feature_impact (*row_count=None, with_metadata=False*)

Request feature impacts to be computed for the model.

See [get_feature_impact](#) for more information on the result of the job.

Parameters

row_count [int] The sample size (specified in rows) to use for Feature Impact computation. This is not supported for unsupervised, multi-class (that has a separate method) and time series projects.

Returns

job [Job] A Job representing the feature impact computation. To get the completed feature impact data, use *job.get_result* or *job.get_result_when_complete*.

Raises

JobAlreadyRequested (422) If the feature impacts have already been requested.

request_frozen_datetime_model (*training_row_count=None, training_duration=None, training_start_date=None, training_end_date=None, time_window_sample_pct=None*)

Train a new frozen model with parameters from this model

Requires that this model belongs to a datetime partitioned project. If it does not, an error will occur when submitting the job.

Frozen models use the same tuning parameters as their parent model instead of independently optimizing them to allow efficiently retraining models on larger amounts of the training data.

In addition of *training_row_count* and *training_duration*, frozen datetime models may be trained on an exact date range. Only one of *training_row_count*, *training_duration*, or *training_start_date* and *training_end_date* should be specified.

Models specified using *training_start_date* and *training_end_date* are the only ones that can be trained into the holdout data (once the holdout is unlocked).

All durations should be specified with a duration string such as those returned by the [partitioning_methods.construct_duration_string](#) helper method. Please see [datetime partitioned project documentation](#) for more information on duration strings.

Parameters

training_row_count [int, optional] the number of rows of data that should be used to train the model. If specified, *training_duration* may not be specified.

training_duration [str, optional] a duration string specifying what time range the data used to train the model should span. If specified, *training_row_count* may not be specified.

training_start_date [datetime.datetime, optional] the start date of the data to train to model on. Only rows occurring at or after this datetime will be used. If *training_start_date* is specified, *training_end_date* must also be specified.

training_end_date [datetime.datetime, optional] the end date of the data to train the model on. Only rows occurring strictly before this datetime will be used. If *training_end_date* is specified, *training_start_date* must also be specified.

time_window_sample_pct [int, optional] may only be specified when the requested model is a time window (e.g. duration or start and end dates). An integer between 1 and 99 indicating the percentage to sample by within the window. The points kept are determined by a random uniform sample. If specified, *training_duration* must be specified otherwise, the number of rows used to train the model and evaluate backtest scores and an error will occur.

Returns

model_job [ModelJob] the modeling job training a frozen model

request_frozen_model (*sample_pct=None, training_row_count=None*)

Train a new frozen model with parameters from this model

Note: This method only works if project the model belongs to is *not* datetime partitioned. If it is, use `request_frozen_datetime_model` instead.

Frozen models use the same tuning parameters as their parent model instead of independently optimizing them to allow efficiently retraining models on larger amounts of the training data.

Parameters

sample_pct [float] optional, the percentage of the dataset to use with the model. If not provided, will use the value from this model.

training_row_count [int] (New in version v2.9) optional, the integer number of rows of the dataset to use with the model. Only one of *sample_pct* and *training_row_count* should be specified.

Returns

model_job [ModelJob] the modeling job training a frozen model

request_predictions (*dataset_id,* *include_prediction_intervals=None,* *prediction_intervals_size=None,* *forecast_point=None,* *predictions_start_date=None,* *predictions_end_date=None,* *actual_value_column=None,* *explanation_algorithm=None,* *max_explanations=None*)

Request predictions against a previously uploaded dataset

Parameters

dataset_id [string] The dataset to make predictions against (as uploaded from `Project.upload_dataset`)

include_prediction_intervals [bool, optional] (New in v2.16) For *time series* projects only. Specifies whether prediction intervals should be calculated for this request. Defaults to True if *prediction_intervals_size* is specified, otherwise defaults to False.

prediction_intervals_size [int, optional] (New in v2.16) For *time series* projects only. Represents the percentile to use for the size of the prediction intervals. Defaults to 80 if *include_prediction_intervals* is True. Prediction intervals size must be between 1 and 100 (inclusive).

forecast_point [datetime.datetime or None, optional] (New in version v2.20) For time series projects only. This is the default point relative to which predictions will be generated, based on the forecast window of the project. See the time series [prediction documentation](#) for more information.

predictions_start_date [datetime.datetime or None, optional] (New in version v2.20) For time series projects only. The start date for bulk predictions. Note that this parameter is for generating historical predictions using the training data. This parameter should be provided in conjunction with *predictions_end_date*. Can't be provided with the *forecast_point* parameter.

predictions_end_date [datetime.datetime or None, optional] (New in version v2.20) For time series projects only. The end date for bulk predictions, exclusive. Note that this parameter is for generating historical predictions using the training data. This parameter

should be provided in conjunction with `predictions_start_date`. Can't be provided with the `forecast_point` parameter.

actual_value_column [string, optional] (New in version v2.21) For time series unsupervised projects only. Actual value column can be used to calculate the classification metrics and insights on the prediction dataset. Can't be provided with the `forecast_point` parameter.

explanation_algorithm: (New in version v2.21) optional; If set to `shap`, the response will include prediction explanations based on the SHAP explainer (SHapley Additive exPlanations). Defaults to null (no prediction explanations).

max_explanations: (New in version v2.21) optional; specifies the maximum number of explanation values that should be returned for each row, ordered by absolute value, greatest to least. If null, no limit. In the case of `shap`: if the number of features is greater than the limit, the sum of remaining values will also be returned as `shapRemainingTotal`. Defaults to null. Cannot be set if `explanation_algorithm` is omitted.

Returns

job [PredictJob] The job computing the predictions

request_training_predictions (*data_subset*, *explanation_algorithm=None*,
max_explanations=None)

Start a job to build training predictions

Parameters

data_subset [str] data set definition to build predictions on. Choices are:

- *dr.enums.DATA_SUBSET.ALL* or string *all* for all data available. Not valid for models in datetime partitioned projects
- *dr.enums.DATA_SUBSET.VALIDATION_AND_HOLDOUT* or string *validationAndHoldout* for all data except training set. Not valid for models in datetime partitioned projects
- *dr.enums.DATA_SUBSET.HOLDOUT* or string *holdout* for holdout data set only
- *dr.enums.DATA_SUBSET.ALL_BACKTESTS* or string *allBacktests* for downloading the predictions for all backtest validation folds. Requires the model to have successfully scored all backtests. Datetime partitioned projects only.

explanation_algorithm [dr.enums.EXPLANATIONS_ALGORITHM] (New in v2.21) Optional. If set to *dr.enums.EXPLANATIONS_ALGORITHM.SHAP*, the response will include prediction explanations based on the SHAP explainer (SHapley Additive exPlanations). Defaults to *None* (no prediction explanations).

max_explanations [int] (New in v2.21) Optional. Specifies the maximum number of explanation values that should be returned for each row, ordered by absolute value, greatest to least. In the case of *dr.enums.EXPLANATIONS_ALGORITHM.SHAP*: If not set, explanations are returned for all features. If the number of features is greater than the `max_explanations`, the sum of remaining values will also be returned as `shap_remaining_total`. Max 100. Defaults to null for datasets narrower than 100 columns, defaults to 100 for datasets wider than 100 columns. Is ignored if `explanation_algorithm` is not set.

Returns

Job an instance of created async job

request_transferable_export (*prediction_intervals_size=None*)

Request generation of an exportable model file for use in an on-premise DataRobot standalone prediction environment.

This function can only be used if model export is enabled, and will only be useful if you have an on-premise environment in which to import it.

This function does not download the exported file. Use `download_export` for that.

Parameters

prediction_intervals_size [int, optional] (New in v2.19) For *time series* projects only. Represents the percentile to use for the size of the prediction intervals. Prediction intervals size must be between 1 and 100 (inclusive).

Examples

```
model = datarobot.Model.get('p-id', 'l-id')
job = model.request_transferable_export()
job.wait_for_completion()
model.download_export('my_exported_model.drmodel')

# Client must be configured to use standalone prediction server for import:
datarobot.Client(token='my-token-at-standalone-server',
                  endpoint='standalone-server-url/api/v2')

imported_model = datarobot.ImportedModel.create('my_exported_model.drmodel')
```

retrain (*sample_pct=None, featurelist_id=None, training_row_count=None*)

Submit a job to the queue to train a blender model.

Parameters

sample_pct: str, optional The sample size in percents (1 to 100) to use in training. If this parameter is used then `training_row_count` should not be given.

featurelist_id [str, optional] The featurelist id

training_row_count [str, optional] The number of rows to train the model. If this parameter is used then `sample_pct` should not be given.

Returns

job [ModelJob] The created job that is retraining the model

set_prediction_threshold (*threshold*)

Set a custom prediction threshold for the model

May not be used once `prediction_threshold_read_only` is True for this model.

Parameters

threshold [float] only used for binary classification projects. The threshold to when deciding between the positive and negative classes when making predictions. Should be between 0.0 and 1.0 (inclusive).

star_model ()

Mark the model as starred

Model stars propagate to the web application and the API, and can be used to filter when listing models.

start_advanced_tuning_session ()

Start an Advanced Tuning session. Returns an object that helps set up arguments for an Advanced Tuning model execution.

As of v2.17, all models other than blenders, open source, prime, scaleout, baseline and user-created support Advanced Tuning.

Returns

AdvancedTuningSession Session for setting up and running Advanced Tuning on a model

train (*sample_pct=None*, *featurelist_id=None*, *scoring_type=None*, *training_row_count=None*, *monotonic_increasing_featurelist_id=<object object>*, *monotonic_decreasing_featurelist_id=<object object>*)

Train the blueprint used in model on a particular featurelist or amount of data.

This method creates a new training job for worker and appends it to the end of the queue for this project. After the job has finished you can get the newly trained model by retrieving it from the project leaderboard, or by retrieving the result of the job.

Either *sample_pct* or *training_row_count* can be used to specify the amount of data to use, but not both. If neither are specified, a default of the maximum amount of data that can safely be used to train any blueprint without going into the validation data will be selected.

In smart-sampled projects, *sample_pct* and *training_row_count* are assumed to be in terms of rows of the minority class.

Note: For datetime partitioned projects, see *train_datetime* instead.

Parameters

sample_pct [float, optional] The amount of data to use for training, as a percentage of the project dataset from 0 to 100.

featurelist_id [str, optional] The identifier of the featurelist to use. If not defined, the featurelist of this model is used.

scoring_type [str, optional] Either `SCORING_TYPE.validation` or `SCORING_TYPE.cross_validation`. `SCORING_TYPE.validation` is available for every partitioning type, and indicates that the default model validation should be used for the project. If the project uses a form of cross-validation partitioning, `SCORING_TYPE.cross_validation` can also be used to indicate that all of the available training/validation combinations should be used to evaluate the model.

training_row_count [int, optional] The number of rows to use to train the requested model.

monotonic_increasing_featurelist_id [str] (new in version 2.11) optional, the id of the featurelist that defines the set of features with a monotonically increasing relationship to the target. Passing `None` disables increasing monotonicity constraint. Default (`dr.enums.MONOTONICITY_FEATURELIST_DEFAULT`) is the one specified by the blueprint.

monotonic_decreasing_featurelist_id [str] (new in version 2.11) optional, the id of the featurelist that defines the set of features with a monotonically decreasing relationship to the target. Passing `None` disables decreasing monotonicity constraint. Default (`dr.enums.MONOTONICITY_FEATURELIST_DEFAULT`) is the one specified by the blueprint.

Returns

model_job_id [str] id of created job, can be used as parameter to `ModelJob.get` method or `wait_for_async_model_creation` function

Examples

```
project = Project.get('p-id')
model = Model.get('p-id', 'l-id')
model_job_id = model.train(training_row_count=project.max_train_rows)
```

train_datetime (*featurelist_id=None, training_row_count=None, training_duration=None, time_window_sample_pct=None, monotonic_increasing_featurelist_id=<object object>, monotonic_decreasing_featurelist_id=<object object>, use_project_settings=False*)

Train this model on a different featurelist or amount of data

Requires that this model is part of a datetime partitioned project; otherwise, an error will occur.

All durations should be specified with a duration string such as those returned by the [partitioning_methods.construct_duration_string](#) helper method. Please see [datetime partitioned project documentation](#) for more information on duration strings.

Parameters

featurelist_id [str, optional] the featurelist to use to train the model. If not specified, the featurelist of this model is used.

training_row_count [int, optional] the number of rows of data that should be used to train the model. If specified, neither `training_duration` nor `use_project_settings` may be specified.

training_duration [str, optional] a duration string specifying what time range the data used to train the model should span. If specified, neither `training_row_count` nor `use_project_settings` may be specified.

use_project_settings [bool, optional] (New in version v2.20) defaults to `False`. If `True`, indicates that the custom backtest partitioning settings specified by the user will be used to train the model and evaluate backtest scores. If specified, neither `training_row_count` nor `training_duration` may be specified.

time_window_sample_pct [int, optional] may only be specified when the requested model is a time window (e.g. duration or start and end dates). An integer between 1 and 99 indicating the percentage to sample by within the window. The points kept are determined by a random uniform sample. If specified, `training_duration` must be specified otherwise, the number of rows used to train the model and evaluate backtest scores and an error will occur.

monotonic_increasing_featurelist_id [str, optional] (New in version v2.18) optional, the id of the featurelist that defines the set of features with a monotonically increasing relationship to the target. Passing `None` disables increasing monotonicity constraint. Default (`dr.enums.MONOTONICITY_FEATURELIST_DEFAULT`) is the one specified by the blueprint.

monotonic_decreasing_featurelist_id [str, optional] (New in version v2.18) optional, the id of the featurelist that defines the set of features with a monotonically decreasing relationship to the target. Passing `None` disables decreasing monotonicity constraint. Default (`dr.enums.MONOTONICITY_FEATURELIST_DEFAULT`) is the one specified by the blueprint.

Returns

job [ModelJob] the created job to build the model

unstar_model()

Unmark the model as starred

Model stars propagate to the web application and the API, and can be used to filter when listing models.

32.20.4 DatetimeModel

```
class datarobot.models.DatetimeModel (id=None, processes=None, featurelist_name=None,
                                       featurelist_id=None, project_id=None, sam-
                                       ple_pct=None, training_row_count=None, train-
                                       ing_duration=None, training_start_date=None, train-
                                       ing_end_date=None, time_window_sample_pct=None,
                                       model_type=None, model_category=None,
                                       is_frozen=None, blueprint_id=None, met-
                                       rics=None, training_info=None, hold-
                                       out_score=None, holdout_status=None,
                                       data_selection_method=None, backtests=None,
                                       monotonic_increasing_featurelist_id=None,
                                       monotonic_decreasing_featurelist_id=None,
                                       supports_monotonic_constraints=None,
                                       is_starred=None, prediction_threshold=None,
                                       prediction_threshold_read_only=None, effec-
                                       tive_feature_derivation_window_start=None, ef-
                                       fective_feature_derivation_window_end=None,
                                       forecast_window_start=None, fore-
                                       cast_window_end=None, windows_basis_unit=None,
                                       model_number=None, parent_model_id=None,
                                       use_project_settings=None)
```

A model from a datetime partitioned project

All durations are specified with a duration string such as those returned by the [partitioning_methods.construct_duration_string](#) helper method. Please see [datetime partitioned project documentation](#) for more information on duration strings.

Note that only one of *training_row_count*, *training_duration*, and *training_start_date* and *training_end_date* will be specified, depending on the *data_selection_method* of the model. Whichever method was selected determines the amount of data used to train on when making predictions and scoring the backtests and the holdout.

Attributes

- id** [str] the id of the model
- project_id** [str] the id of the project the model belongs to
- processes** [list of str] the processes used by the model
- featurelist_name** [str] the name of the featurelist used by the model
- featurelist_id** [str] the id of the featurelist used by the model
- sample_pct** [float] the percentage of the project dataset used in training the model
- training_row_count** [int or None] If specified, an int specifying the number of rows used to train the model and evaluate backtest scores.
- training_duration** [str or None] If specified, a duration string specifying the duration spanned by the data used to train the model and evaluate backtest scores.
- training_start_date** [datetime or None] only present for frozen models in datetime partitioned projects. If specified, the start date of the data used to train the model.

training_end_date [datetime or None] only present for frozen models in datetime partitioned projects. If specified, the end date of the data used to train the model.

time_window_sample_pct [int or None] An integer between 1 and 99 indicating the percentage of sampling within the training window. The points kept are determined by a random uniform sample. If not specified, no sampling was done.

model_type [str] what model this is, e.g. `Nystroem Kernel SVM Regressor`

model_category [str] what kind of model this is - `prime` for DataRobot Prime models, `blend` for blender models, and `model` for other models

is_frozen [bool] whether this model is a frozen model

blueprint_id [str] the id of the blueprint used in this model

metrics [dict] a mapping from each metric to the model's scores for that metric. The keys in metrics are the different metrics used to evaluate the model, and the values are the results. The dictionaries inside of metrics will be as described here: `validation`, the score for a single backtest; `crossValidation`, always None; `backtesting`, the average score for all backtests if all are available and computed, or None otherwise; `backtestingScores`, a list of scores for all backtests where the score is None if that backtest does not have a score available; and `holdout`, the score for the holdout or None if the holdout is locked or the score is unavailable.

backtests [list of dict] describes what data was used to fit each backtest, the score for the project metric, and why the backtest score is unavailable if it is not provided.

data_selection_method [str] which of `training_row_count`, `training_duration`, or `training_start_data` and `training_end_date` were used to determine the data used to fit the model. One of `rowCount`, `duration`, or `selectedDateRange`.

training_info [dict] describes which data was used to train on when scoring the holdout and making predictions. `training_info` will have the following keys: `holdout_training_start_date`, `holdout_training_duration`, `holdout_training_row_count`, `holdout_training_end_date`, `prediction_training_start_date`, `prediction_training_duration`, `prediction_training_row_count`, `prediction_training_end_date`. Start and end dates will be datetimes, durations will be duration strings, and rows will be integers.

holdout_score [float or None] the score against the holdout, if available and the holdout is unlocked, according to the project metric.

holdout_status [string or None] the status of the holdout score, e.g. `IJCOMPLETED`, `IJHOLDOUT_BOUNDARIES_EXCEEDED`. Unavailable if the holdout fold was disabled in the partitioning configuration.

monotonic_increasing_featurelist_id [str] optional, the id of the featurelist that defines the set of features with a monotonically increasing relationship to the target. If None, no such constraints are enforced.

monotonic_decreasing_featurelist_id [str] optional, the id of the featurelist that defines the set of features with a monotonically decreasing relationship to the target. If None, no such constraints are enforced.

supports_monotonic_constraints [bool] optional, whether this model supports enforcing monotonic constraints

is_starred [bool] whether this model marked as starred

prediction_threshold [float] for binary classification projects, the threshold used for predictions

prediction_threshold_read_only [bool] indicated whether modification of the prediction threshold is forbidden. Threshold modification is forbidden once a model has had a deployment created or predictions made via the dedicated prediction API.

effective_feature_derivation_window_start [int or None] (New in v2.16) For *time series* projects only. How many units of the `windows_basis_unit` into the past relative to the forecast point the user needs to provide history for at prediction time. This can differ from the `feature_derivation_window_start` set on the project due to the differencing method and period selected, or if the model is a time series native model such as ARIMA. Will be a negative integer in time series projects and *None* otherwise.

effective_feature_derivation_window_end [int or None] (New in v2.16) For *time series* projects only. How many units of the `windows_basis_unit` into the past relative to the forecast point the feature derivation window should end. Will be a non-positive integer in time series projects and *None* otherwise.

forecast_window_start [int or None] (New in v2.16) For *time series* projects only. How many units of the `windows_basis_unit` into the future relative to the forecast point the forecast window should start. Note that this field will be the same as what is shown in the project settings. Will be a non-negative integer in time series projects and *None* otherwise.

forecast_window_end [int or None] (New in v2.16) For *time series* projects only. How many units of the `windows_basis_unit` into the future relative to the forecast point the forecast window should end. Note that this field will be the same as what is shown in the project settings. Will be a non-negative integer in time series projects and *None* otherwise.

windows_basis_unit [str or None] (New in v2.16) For *time series* projects only. Indicates which unit is the basis for the feature derivation window and the forecast window. Note that this field will be the same as what is shown in the project settings. In time series projects, will be either the detected time unit or `â€œROWâ€œ`, and *None* otherwise.

model_number [integer] model number assigned to a model

parent_model_id [str or None] (New in version v2.20) the id of the model that tuning parameters are derived from

use_project_settings [bool or None] (New in version v2.20) If `True`, indicates that the custom backtest partitioning settings specified by the user were used to train the model and evaluate backtest scores.

classmethod `get(project, model_id)`

Retrieve a specific datetime model

If the project does not use datetime partitioning, a `ClientError` will occur.

Parameters

project [str] the id of the project the model belongs to

model_id [str] the id of the model to retrieve

Returns

model [DatetimeModel] the model

score_backtests ()

Compute the scores for all available backtests

Some backtests may be unavailable if the model is trained into their validation data.

Returns

job [Job] a job tracking the backtest computation. When it is complete, all available backtests will have scores computed.

cross_validate ()

Inherited from Model - DatetimeModels cannot request Cross Validation,

Use `score_backtests` instead.

get_cross_validation_scores (*partition=None, metric=None*)

Inherited from Model - DatetimeModels cannot request Cross Validation scores,

Use `backtests` instead.

request_training_predictions (*data_subset*)

Start a job to build training predictions

Parameters

data_subset [str] data set definition to build predictions on. Choices are:

- *dr.enums.DATA_SUBSET.HOLDOUT* for holdout data set only
- *dr.enums.DATA_SUBSET.ALL_BACKTESTS* for downloading the predictions for all backtest validation folds. Requires the model to have successfully scored all backtests.

Returns

Job

an instance of created async job

get_series_accuracy_as_dataframe (*offset=0, limit=100, metric=None, multiseries_value=None, order_by=None, reverse=False*)

Retrieve the Series Accuracy for the specified model as a pandas.DataFrame.

Parameters

offset [int, optional] The number of results to skip. Defaults to 0 if not specified.

limit [int, optional] The maximum number of results to return. Defaults to 100 if not specified.

metric [str, optional] The name of the metric to retrieve scores for. If omitted, the default project metric will be used.

multiseries_value [str, optional] If specified, only the series containing the given value in one of the series ID columns will be returned.

order_by [str, optional] Used for sorting the series. Attribute must be one of `datarobot.enums.SERIES_ACCURACY_ORDER_BY`.

reverse [bool, optional] Used for sorting the series. If `True`, will sort the series in descending order by the attribute specified by `order_by`.

Returns

data A pandas.DataFrame with the Series Accuracy for the specified model.

download_series_accuracy_as_csv (*filename, encoding='utf-8', offset=0, limit=100, metric=None, multiseries_value=None, order_by=None, reverse=False*)

Save the Series Accuracy for the specified model into a csv file.

Parameters

filename [str or file object] The path or file object to save the data to.

encoding [str, optional] A string representing the encoding to use in the output csv file. Defaults to `utf-8`.

offset [int, optional] The number of results to skip. Defaults to 0 if not specified.

limit [int, optional] The maximum number of results to return. Defaults to 100 if not specified.

metric [str, optional] The name of the metric to retrieve scores for. If omitted, the default project metric will be used.

multiseries_value [str, optional] If specified, only the series containing the given value in one of the series ID columns will be returned.

order_by [str, optional] Used for sorting the series. Attribute must be one of `datarobot.enums.SERIES_ACCURACY_ORDER_BY`.

reverse [bool, optional] Used for sorting the series. If `True`, will sort the series in descending order by the attribute specified by `order_by`.

compute_series_accuracy()

Compute the Series Accuracy for this model

Returns

Job an instance of the created async job

retrain(*time_window_sample_pct=None, featurelist_id=None, training_row_count=None, training_duration=None, training_start_date=None, training_end_date=None*)

Submit a job to the queue to train a blender model.

All durations should be specified with a duration string such as those returned by the [partitioning_methods.construct_duration_string](#) helper method. Please see [datetime partitioned project documentation](#) for more information on duration strings.

Parameters

featurelist_id [str, optional] The featurelist id

training_row_count [str, optional] The number of rows to train the model. If this parameter is used then `sample_pct` should not be given.

time_window_sample_pct [int, optional] An int between 1 and 99 indicating the percentage of sampling within the time window. The points kept are determined by a random uniform sample. If specified, `training_row_count` must not be specified and `training_duration` or `training_start_date` and `training_end_date` must be specified.

training_duration [str, optional] A duration string representing the training duration for the submitted model. If specified then `training_row_count` must not be specified.

training_start_date [str, optional] A datetime string representing the start date of the data to use for training this model. If specified, `training_end_date` must also be specified. The value must be before the `training_end_date` value.

training_end_date [str, optional] A datetime string representing the end date of the data to use for training this model. If specified, `training_start_date` must also be specified. The value must be after the `training_start_date` value.

Returns

job [ModelJob] The created job that is retraining the model

get_feature_effect_metadata()

Retrieve Feature Effect metadata for each backtest. Response contains status and available sources for each backtest of the model.

- Each backtest is available for *training* and *validation*
- If holdout is configured for the project it has *holdout* as *backtestIndex*. It has *training* and *holdout* sources available.

Start/stop models contain a single response item with *startstop* value for *backtestIndex*.

- Feature Effect of *training* is always available (except for the old project which supports only Feature Effect for *validation*).
- When a model is trained into *validation* or *holdout* without stacked prediction (e.g. no out-of-sample prediction in *validation* or *holdout*), Feature Effect is not available for *validation* or *holdout*.
- Feature Effect for *holdout* is not available when there is no holdout configured for the project.

source is expected parameter to retrieve Feature Effect. One of provided sources shall be used.

backtestIndex is expected parameter to submit compute request and retrieve Feature Effect. One of provided backtest indexes shall be used.

Returns

feature_effect_metadata: FeatureEffectMetadataDatetime

get_feature_fit_metadata()

Retrieve Feature Fit metadata for each backtest. Response contains status and available sources for each backtest of the model.

- Each backtest is available for *training* and *validation*
- If holdout is configured for the project it has *holdout* as *backtestIndex*. It has *training* and *holdout* sources available.

Start/stop models contain a single response item with *startstop* value for *backtestIndex*.

- Feature Fit of *training* is always available (except for the old project which supports only Feature Effect for *validation*).
- When a model is trained into *validation* or *holdout* without stacked prediction (e.g. no out-of-sample prediction in *validation* or *holdout*), Feature Fit is not available for *validation* or *holdout*.
- Feature Fit for *holdout* is not available when there is no holdout configured for the project.

source is expected parameter to retrieve Feature Fit. One of provided sources shall be used.

backtestIndex is expected parameter to submit compute request and retrieve Feature Fit. One of provided backtest indexes shall be used.

Returns

feature_effect_metadata: FeatureFitMetadataDatetime

request_feature_effect(backtest_index)

Request feature effects to be computed for the model.

See [get_feature_effect](#) for more information on the result of the job.

See [get_feature_effect_metadata](#) for retrieving information of backtest_index.

Parameters

backtest_index: string, FeatureEffectMetadataDatetime.backtest_index. The backtest index to retrieve Feature Effects for.

Returns

job [Job] A Job representing the feature effect computation. To get the completed feature effect data, use `job.get_result` or `job.get_result_when_complete`.

Raises

JobAlreadyRequested (422) If the feature effect have already been requested.

get_feature_effect (*source, backtest_index*)

Retrieve Feature Effects for the model.

Feature Effects provides partial dependence and predicted vs actual values for top-500 features ordered by feature impact score.

The partial dependence shows marginal effect of a feature on the target variable after accounting for the average effects of all other predictive features. It indicates how, holding all other variables except the feature of interest as they were, the value of this feature affects your prediction.

Requires that Feature Effects has already been computed with `request_feature_effect`.

See `get_feature_effect_metadata` for retrieving information of source, backtest_index.

Parameters

source: string The source Feature Effects are retrieved for. One value of [FeatureEffect-MetadataDatetime.sources]. To retrieve the available sources for feature effect.

backtest_index: string, FeatureEffectMetadataDatetime.backtest_index. The backtest index to retrieve Feature Effects for.

Returns

feature_effects: FeatureEffects The feature effects data.

Raises

ClientError (404) If the feature effects have not been computed or source is not valid value.

get_or_request_feature_effect (*source, backtest_index, max_wait=600*)

Retrieve feature effect for the model, requesting a job if it hasn't been run previously

See `get_feature_effect_metadata` for retrieving information of source, backtest_index.

Parameters

max_wait [int, optional] The maximum time to wait for a requested feature effect job to complete before erroring

source [string] The source Feature Effects are retrieved for. One value of [FeatureEffect-MetadataDatetime.sources]. To retrieve the available sources for feature effect.

backtest_index: string, FeatureEffectMetadataDatetime.backtest_index. The backtest index to retrieve Feature Effects for.

Returns

feature_effects [FeatureEffects] The feature effects data.

request_feature_fit (*backtest_index*)

Request feature fit to be computed for the model.

See `get_feature_fit` for more information on the result of the job.

See `get_feature_fit_metadata` for retrieving information of backtest_index.

Parameters

backtest_index: `string`, `FeatureFitMetadataDatetime.backtest_index`. The backtest index to retrieve Feature Fit for.

Returns

job [Job] A Job representing the feature fit computation. To get the completed feature fit data, use `job.get_result` or `job.get_result_when_complete`.

Raises

JobAlreadyRequested (422) If the feature fit have already been requested.

get_feature_fit (*source*, *backtest_index*)

Retrieve Feature Fit for the model.

Feature Fit provides partial dependence and predicted vs actual values for top-500 features ordered by feature impact score.

The partial dependence shows marginal effect of a feature on the target variable after accounting for the average effects of all other predictive features. It indicates how, holding all other variables except the feature of interest as they were, the value of this feature affects your prediction.

Requires that Feature Fit has already been computed with `request_feature_fit`.

See `get_feature_fit_metadata` for retrieving information of source, backtest_index.

Parameters

source: `string` The source Feature Fit are retrieved for. One value of [`FeatureFitMetadataDatetime.sources`]. To retrieve the available sources for feature fit.

backtest_index: `string`, `FeatureFitMetadataDatetime.backtest_index`. The backtest index to retrieve Feature Fit for.

Returns

feature_fit: `FeatureFit` The feature fit data.

Raises

ClientError (404) If the feature fit have not been computed or source is not valid value.

get_or_request_feature_fit (*source*, *backtest_index*, *max_wait=600*)

Retrieve feature fit for the model, requesting a job if it hasn't been run previously

See `get_feature_fit_metadata` for retrieving information of source, backtest_index.

Parameters

max_wait [int, optional] The maximum time to wait for a requested feature fit job to complete before erroring

source [string] The source Feature Fit are retrieved for. One value of [`FeatureFitMetadataDatetime.sources`]. To retrieve the available sources for feature effect.

backtest_index: `string`, `FeatureFitMetadataDatetime.backtest_index`. The backtest index to retrieve Feature Fit for.

Returns

feature_fit [`FeatureFit`] The feature fit data.

calculate_prediction_intervals (*prediction_intervals_size*)

Calculate prediction intervals for this `DatetimeModel` for the specified size.

New in version v2.19.

Parameters

prediction_intervals_size [int] The prediction intervals size to calculate for this model. See the [prediction intervals](#) documentation for more information.

Returns

job [Job] a [Job](#) tracking the prediction intervals computation

get_calculated_prediction_intervals (*offset=None, limit=None*)

Retrieve a list of already-calculated prediction intervals for this model

New in version v2.19.

Parameters

offset [int, optional] If provided, this many results will be skipped

limit [int, optional] If provided, at most this many results will be returned. If not provided, will return at most 100 results.

Returns

list[int] A descending-ordered list of already-calculated prediction interval sizes

advanced_tune (*params, description=None*)

Generate a new model with the specified advanced-tuning parameters

As of v2.17, all models other than blenders, open source, prime, scaleout, baseline and user-created support Advanced Tuning.

Parameters

params [dict] Mapping of parameter ID to parameter value. The list of valid parameter IDs for a model can be found by calling [get_advanced_tuning_parameters\(\)](#). This endpoint does not need to include values for all parameters. If a parameter is omitted, its *current_value* will be used.

description [unicode] Human-readable string describing the newly advanced-tuned model

Returns

ModelJob The created job to build the model

delete ()

Delete a model from the project's leaderboard.

download_export (*filepath*)

Download an exportable model file for use in an on-premise DataRobot standalone prediction environment.

This function can only be used if model export is enabled, and will only be useful if you have an on-premise environment in which to import it.

Parameters

filepath [str] The path at which to save the exported model file.

download_scoring_code (*file_name, source_code=False*)

Download scoring code JAR.

Parameters

file_name [str] File path where scoring code will be saved.

source_code [bool, optional] Set to True to download source code archive. It will not be executable.

classmethod `fetch_resource_data(url, join_endpoint=True)`

(Deprecated.) Used to acquire model data directly from its url.

Consider using `get` instead, as this is a convenience function used for development of datarobot

Parameters

url [str] The resource we are acquiring

join_endpoint [boolean, optional] Whether the client's endpoint should be joined to the URL before sending the request. Location headers are returned as absolute locations, so `will_not_need` the endpoint

Returns

model_data [dict] The queried model's data

get_advanced_tuning_parameters()

Get the advanced-tuning parameters available for this model.

As of v2.17, all models other than blenders, open source, prime, scaleout, baseline and user-created support Advanced Tuning.

Returns

dict A dictionary describing the advanced-tuning parameters for the current model. There are two top-level keys, *tuningDescription* and *tuningParameters*.

tuningDescription an optional value. If not *None*, then it indicates the user-specified description of this set of tuning parameter.

tuningParameters is a list of a dicts, each has the following keys

- **parameterName** : (**unicode**) name of the parameter (unique per task, see below)
- **parameterId** : (**unicode**) opaque ID string uniquely identifying parameter
- **defaultValue** : (*) default value of the parameter for the blueprint
- **currentValue** : (*) value of the parameter that was used for this model
- **taskName** : (**unicode**) name of the task that this parameter belongs to
- **constraints** : (**dict**) see the notes below

Notes

The type of *defaultValue* and *currentValue* is defined by the *constraints* structure. It will be a string or numeric Python type.

constraints is a dict with *at least one*, possibly more, of the following keys. The presence of a key indicates that the parameter may take on the specified type. (If a key is absent, this means that the parameter may not take on the specified type.) If a key on *constraints* is present, its value will be a dict containing all of the fields described below for that key.

```
"constraints": {
  "select": {
    "values": [<list (basestring or number) : possible values>]
  },
  "ascii": {},
  "unicode": {},
  "int": {
    "min": <int : minimum valid value>,

```

(continues on next page)

(continued from previous page)

```

        "max": <int : maximum valid value>,
        "supports_grid_search": <bool : True if Grid Search may be
                                requested for this param>
    },
    "float": {
        "min": <float : minimum valid value>,
        "max": <float : maximum valid value>,
        "supports_grid_search": <bool : True if Grid Search may be
                                requested for this param>
    },
    "intList": {
        "length": {
            "min_length": <int : minimum valid length>,
            "max_length": <int : maximum valid length>
            "min_val": <int : minimum valid value>,
            "max_val": <int : maximum valid value>
            "supports_grid_search": <bool : True if Grid Search may be
                                    requested for this param>
        },
        "floatList": {
            "min_length": <int : minimum valid length>,
            "max_length": <int : maximum valid length>
            "min_val": <float : minimum valid value>,
            "max_val": <float : maximum valid value>
            "supports_grid_search": <bool : True if Grid Search may be
                                    requested for this param>
        }
    }
}

```

The keys have meaning as follows:

- *select*: Rather than specifying a specific data type, if present, it indicates that the parameter is permitted to take on any of the specified values. Listed values may be of any string or real (non-complex) numeric type.
- *ascii*: The parameter may be a *unicode* object that encodes simple ASCII characters. (A-Z, a-z, 0-9, whitespace, and certain common symbols.) In addition to listed constraints, ASCII keys currently may not contain either newlines or semicolons.
- *unicode*: The parameter may be any Python *unicode* object.
- *int*: The value may be an object of type *int* within the specified range (inclusive). Please note that the value will be passed around using the JSON format, and some JSON parsers have undefined behavior with integers outside of the range $[-(2^{53})+1, (2^{53})-1]$.
- *float*: The value may be an object of type *float* within the specified range (inclusive).
- *intList*, *floatList*: The value may be a list of *int* or *float* objects, respectively, following constraints as specified respectively by the *int* and *float* types (above).

Many parameters only specify one key under *constraints*. If a parameter specifies multiple keys, the parameter may take on any value permitted by any key.

get_all_confusion_charts (*fallback_to_parent_insights=False*)

Retrieve a list of all confusion charts available for the model.

Parameters

fallback_to_parent_insights [bool] (New in version v2.14) Optional, if True, this will return confusion chart data for this model's parent for any source that is not available for

this model and if this has a defined parent model. If omitted or False, or this model has no parent, this will not attempt to retrieve any data from this model's parent.

Returns

list of ConfusionChart Data for all available confusion charts for model.

get_all_lift_charts (*fallback_to_parent_insights=False*)

Retrieve a list of all lift charts available for the model.

Parameters

fallback_to_parent_insights [bool] (New in version v2.14) Optional, if True, this will return lift chart data for this model's parent for any source that is not available for this model and if this model has a defined parent model. If omitted or False, or this model has no parent, this will not attempt to retrieve any data from this model's parent.

Returns

list of LiftChart Data for all available model lift charts.

get_all_residuals_charts (*fallback_to_parent_insights=False*)

Retrieve a list of all lift charts available for the model.

Parameters

fallback_to_parent_insights [bool] Optional, if True, this will return residuals chart data for this model's parent for any source that is not available for this model and if this model has a defined parent model. If omitted or False, or this model has no parent, this will not attempt to retrieve any data from this model's parent.

Returns

list of ResidualsChart Data for all available model residuals charts.

get_all_roc_curves (*fallback_to_parent_insights=False*)

Retrieve a list of all ROC curves available for the model.

Parameters

fallback_to_parent_insights [bool] (New in version v2.14) Optional, if True, this will return ROC curve data for this model's parent for any source that is not available for this model and if this model has a defined parent model. If omitted or False, or this model has no parent, this will not attempt to retrieve any data from this model's parent.

Returns

list of RocCurve Data for all available model ROC curves.

get_confusion_chart (*source, fallback_to_parent_insights=False*)

Retrieve model's confusion chart for the specified source.

Parameters

source [str] Confusion chart source. Check `datarobot.enums.CHART_DATA_SOURCE` for possible values.

fallback_to_parent_insights [bool] (New in version v2.14) Optional, if True, this will return confusion chart data for this model's parent if the confusion chart is not available for this model and the defined parent model. If omitted or False, or there is no parent model, will not attempt to return insight data from this model's parent.

Returns

ConfusionChart Model ConfusionChart data

Raises

ClientError If the insight is not available for this model

get_feature_impact (*with_metadata=False*)

Retrieve the computed Feature Impact results, a measure of the relevance of each feature in the model.

Feature Impact is computed for each column by creating new data with that column randomly permuted (but the others left unchanged), and seeing how the error metric score for the predictions is affected. The `impactUnnormalized` is how much worse the error metric score is when making predictions on this modified data. The `impactNormalized` is normalized so that the largest value is 1. In both cases, larger values indicate more important features.

If a feature is a redundant feature, i.e. once other features are considered it doesn't contribute much in addition, the `redundantWith` value is the name of feature that has the highest correlation with this feature. Note that redundancy detection is only available for jobs run after the addition of this feature. When retrieving data that predates this functionality, a `NoRedundancyImpactAvailable` warning will be used.

Elsewhere this technique is sometimes called `Permutation Importance`.

Requires that Feature Impact has already been computed with `request_feature_impact`.

Parameters

with_metadata [bool] The flag indicating if the result should include the metadata as well.

Returns

list or dict The feature impact data response depends on the `with_metadata` parameter. The response is either a dict with metadata and a list with actual data or just a list with that data.

Each List item is a dict with the keys `featureName`, `impactNormalized`, and `impactUnnormalized`, `redundantWith` and `count`.

For dict response available keys are:

- **featureImpacts** - Feature Impact data as a dictionary. Each item is a dict with keys: `featureName`, `impactNormalized`, and `impactUnnormalized`, and `redundantWith`.
- **shapBased** - A boolean that indicates whether Feature Impact was calculated using Shapley values.
- **ranRedundancyDetection** - A boolean that indicates whether redundant feature identification was run while calculating this Feature Impact.
- **rowCount** - An integer or None that indicates the number of rows that was used to calculate Feature Impact. For the Feature Impact calculated with the default logic, without specifying the `rowCount`, we return None here.
- **count** - An integer with the number of features under the `featureImpacts`.

Raises

ClientError (404) If the feature impacts have not been computed.

get_features_used ()

Query the server to determine which features were used.

Note that the data returned by this method is possibly different than the names of the features in the `featurelist` used by this model. This method will return the raw features that must be supplied in order for

predictions to be generated on a new set of data. The featurelist, in contrast, would also include the names of derived features.

Returns

features [list of str] The names of the features used in the model.

get_frozen_child_models()

Retrieves the ids for all the models that are frozen from this model

Returns

A list of Models

get_leaderboard_ui_permalink()

Returns

url [str] Permanent static hyperlink to this model at leaderboard.

get_lift_chart(source, fallback_to_parent_insights=False)

Retrieve model lift chart for the specified source.

Parameters

source [str] Lift chart data source. Check datarobot.enums.CHART_DATA_SOURCE for possible values.

fallback_to_parent_insights [bool] (New in version v2.14) Optional, if True, this will return lift chart data for this model's parent if the lift chart is not available for this model and the model has a defined parent model. If omitted or False, or there is no parent model, will not attempt to return insight data from this model's parent.

Returns

LiftChart Model lift chart data

Raises

ClientError If the insight is not available for this model

get_missing_report_info()

Retrieve a model missing data report on training data that can be used to understand missing values treatment in a model. Report consists of missing values reports for features which took part in modelling and are numeric or categorical.

Returns

An iterable of MissingReportPerFeature The queried model missing report, sorted by missing count (DESCENDING order).

get_model_blueprint_chart()

Retrieve a model blueprint chart that can be used to understand data flow in blueprint.

Returns

ModelBlueprintChart The queried model blueprint chart.

get_model_blueprint_documents()

Get documentation for tasks used in this model.

Returns

list of BlueprintTaskDocument All documents available for the model.

get_multiclass_feature_impact()

For multiclass it's possible to calculate feature impact separately for each target class. The method for calculation is exactly the same, calculated in one-vs-all style for each target class.

Requires that Feature Impact has already been computed with `request_feature_impact`.

Returns

feature_impacts [list of dict] The feature impact data. Each item is a dict with the keys `featureImpacts` (list), `class` (str). Each item in `featureImpacts` is a dict with the keys `featureName`, `impactNormalized`, and `impactUnnormalized`, and `redundantWith`.

Raises

ClientError (404) If the multiclass feature impacts have not been computed.

get_multiclass_lift_chart(source, fallback_to_parent_insights=False)

Retrieve model lift chart for the specified source.

Parameters

source [str] Lift chart data source. Check `datarobot.enums.CHART_DATA_SOURCE` for possible values.

fallback_to_parent_insights [bool] Optional, if True, this will return lift chart data for this model's parent if the lift chart is not available for this model and the model has a defined parent model. If omitted or False, or there is no parent model, will not attempt to return insight data from this model's parent.

Returns

list of LiftChart Model lift chart data for each saved target class

Raises

ClientError If the insight is not available for this model

get_num_estimators_trained()

Retrieves the number of estimators trained by early-stopping tree-based models

– versionadded:: v2.22

Returns

projectId: str id of project containing the model

modelId: str id of the model

data: array list of `numEstimatorsItem` objects, one for each modeling stage.

numEstimatorsItem will be of the form:

stage: str indicates the modeling stage (for multi-stage models); None of single-stage models

numIterations: int the number of estimators or iterations trained by the model

get_or_request_feature_impact(max_wait=600, **kwargs)

Retrieve feature impact for the model, requesting a job if it hasn't been run previously

Parameters

max_wait [int, optional] The maximum time to wait for a requested feature impact job to complete before erroring

****kwargs** Arbitrary keyword arguments passed to `request_feature_impact`.

Returns

feature_impacts [list or dict] The feature impact data. See [get_feature_impact](#) for the exact schema.

get_parameters ()

Retrieve model parameters.

Returns

ModelParameters Model parameters for this model.

get_pareto_front ()

Retrieve the Pareto Front for a Eureqa model.

This method is only supported for Eureqa models.

Returns

ParetoFront Model ParetoFront data

get_prime_eligibility ()

Check if this model can be approximated with DataRobot Prime

Returns

prime_eligibility [dict] a dict indicating whether a model can be approximated with DataRobot Prime (key *can_make_prime*) and why it may be ineligible (key *message*)

get_residuals_chart (*source, fallback_to_parent_insights=False*)

Retrieve model residuals chart for the specified source.

Parameters

source [str] Residuals chart data source. Check `datarobot.enums.CHART_DATA_SOURCE` for possible values.

fallback_to_parent_insights [bool] Optional, if True, this will return residuals chart data for this model's parent if the residuals chart is not available for this model and the model has a defined parent model. If omitted or False, or there is no parent model, will not attempt to return residuals data from this model's parent.

Returns

ResidualsChart Model residuals chart data

Raises

ClientError If the insight is not available for this model

get_roc_curve (*source, fallback_to_parent_insights=False*)

Retrieve model ROC curve for the specified source.

Parameters

source [str] ROC curve data source. Check `datarobot.enums.CHART_DATA_SOURCE` for possible values.

fallback_to_parent_insights [bool] (New in version v2.14) Optional, if True, this will return ROC curve data for this model's parent if the ROC curve is not available for this model and the model has a defined parent model. If omitted or False, or there is no parent model, will not attempt to return data from this model's parent.

Returns

RocCurve Model ROC curve data

Raises

ClientError If the insight is not available for this model

get_rulesets()

List the rulesets approximating this model generated by DataRobot Prime

If this model hasn't been approximated yet, will return an empty list. Note that these are rulesets approximating this model, not rulesets used to construct this model.

Returns

rulesets [list of Ruleset]

get_supported_capabilities()

Retrieves a summary of the capabilities supported by a model.

New in version v2.14.

Returns

supportsBlending: bool whether the model supports blending

supportsMonotonicConstraints: bool whether the model supports monotonic constraints

hasWordCloud: bool whether the model has word cloud data available

eligibleForPrime: bool whether the model is eligible for Prime

hasParameters: bool whether the model has parameters that can be retrieved

supportsCodeGeneration: bool (New in version v2.18) whether the model supports code generation

supportsShap: bool

(New in version v2.18) True if the model supports Shapley package. i.e. Shapley based feature Importance

supportsEarlyStopping: bool (New in version v2.22) True if this is an early stopping tree-based model and number of trained iterations can be retrieved.

get_word_cloud(exclude_stop_words=False)

Retrieve a word cloud data for the model.

Parameters

exclude_stop_words [bool, optional] Set to True if you want stopwords filtered out of response.

Returns

WordCloud Word cloud data for the model.

open_model_browser()

Opens model at project leaderboard in web browser.

Note: If text-mode browsers are used, the calling process will block until the user exits the browser.

request_approximation()

Request an approximation of this model using DataRobot Prime

This will create several rulesets that could be used to approximate this model. After comparing their scores and rule counts, the code used in the approximation can be downloaded and run locally.

Returns

job [Job] the job generating the rulesets

request_external_test (*dataset_id*, *actual_value_column=None*)

Request external test to compute scores and insights on an external test dataset

Parameters

dataset_id [string] The dataset to make predictions against (as uploaded from `Project.upload_dataset`)

actual_value_column [string, optional] (New in version v2.21) For time series unsupervised projects only. Actual value column can be used to calculate the classification metrics and insights on the prediction dataset. Can't be provided with the `forecast_point` parameter.

Returns

âĀĤâĤ

job [Job] a Job representing external dataset insights computation

request_feature_impact (*row_count=None*, *with_metadata=False*)

Request feature impacts to be computed for the model.

See [get_feature_impact](#) for more information on the result of the job.

Parameters

row_count [int] The sample size (specified in rows) to use for Feature Impact computation. This is not supported for unsupervised, multi-class (that has a separate method) and time series projects.

Returns

job [Job] A Job representing the feature impact computation. To get the completed feature impact data, use `job.get_result` or `job.get_result_when_complete`.

Raises

JobAlreadyRequested (422) If the feature impacts have already been requested.

request_frozen_datetime_model (*training_row_count=None*, *training_duration=None*,
training_start_date=None, *training_end_date=None*,
time_window_sample_pct=None)

Train a new frozen model with parameters from this model

Requires that this model belongs to a datetime partitioned project. If it does not, an error will occur when submitting the job.

Frozen models use the same tuning parameters as their parent model instead of independently optimizing them to allow efficiently retraining models on larger amounts of the training data.

In addition of `training_row_count` and `training_duration`, frozen datetime models may be trained on an exact date range. Only one of `training_row_count`, `training_duration`, or `training_start_date` and `training_end_date` should be specified.

Models specified using `training_start_date` and `training_end_date` are the only ones that can be trained into the holdout data (once the holdout is unlocked).

All durations should be specified with a duration string such as those returned by the [partitioning_methods.construct_duration_string](#) helper method. Please see [datetime partitioned project documentation](#) for more information on duration strings.

Parameters

training_row_count [int, optional] the number of rows of data that should be used to train the model. If specified, `training_duration` may not be specified.

training_duration [str, optional] a duration string specifying what time range the data used to train the model should span. If specified, `training_row_count` may not be specified.

training_start_date [datetime.datetime, optional] the start date of the data to train to model on. Only rows occurring at or after this datetime will be used. If `training_start_date` is specified, `training_end_date` must also be specified.

training_end_date [datetime.datetime, optional] the end date of the data to train the model on. Only rows occurring strictly before this datetime will be used. If `training_end_date` is specified, `training_start_date` must also be specified.

time_window_sample_pct [int, optional] may only be specified when the requested model is a time window (e.g. duration or start and end dates). An integer between 1 and 99 indicating the percentage to sample by within the window. The points kept are determined by a random uniform sample. If specified, `training_duration` must be specified otherwise, the number of rows used to train the model and evaluate backtest scores and an error will occur.

Returns

model_job [ModelJob] the modeling job training a frozen model

request_predictions (*dataset_id*, *include_prediction_intervals=None*, *prediction_intervals_size=None*, *forecast_point=None*, *predictions_start_date=None*, *predictions_end_date=None*, *actual_value_column=None*, *explanation_algorithm=None*, *max_explanations=None*)

Request predictions against a previously uploaded dataset

Parameters

dataset_id [string] The dataset to make predictions against (as uploaded from `Project.upload_dataset`)

include_prediction_intervals [bool, optional] (New in v2.16) For *time series* projects only. Specifies whether prediction intervals should be calculated for this request. Defaults to True if *prediction_intervals_size* is specified, otherwise defaults to False.

prediction_intervals_size [int, optional] (New in v2.16) For *time series* projects only. Represents the percentile to use for the size of the prediction intervals. Defaults to 80 if *include_prediction_intervals* is True. Prediction intervals size must be between 1 and 100 (inclusive).

forecast_point [datetime.datetime or None, optional] (New in version v2.20) For time series projects only. This is the default point relative to which predictions will be generated, based on the forecast window of the project. See the time series *prediction documentation* for more information.

predictions_start_date [datetime.datetime or None, optional] (New in version v2.20) For time series projects only. The start date for bulk predictions. Note that this parameter is for generating historical predictions using the training data. This parameter should be provided in conjunction with `predictions_end_date`. Can't be provided with the `forecast_point` parameter.

predictions_end_date [datetime.datetime or None, optional] (New in version v2.20) For time series projects only. The end date for bulk predictions, exclusive. Note that this parameter is for generating historical predictions using the training data. This parameter should be provided in conjunction with `predictions_start_date`. Can't be provided with the `forecast_point` parameter.

actual_value_column [string, optional] (New in version v2.21) For time series unsupervised projects only. Actual value column can be used to calculate the classification metrics and insights on the prediction dataset. Can't be provided with the `forecast_point` parameter.

explanation_algorithm: (New in version v2.21) optional; If set to `'shap'`, the response will include prediction explanations based on the SHAP explainer (SHapley Additive exPlanations). Defaults to null (no prediction explanations).

max_explanations: (New in version v2.21) optional; specifies the maximum number of explanation values that should be returned for each row, ordered by absolute value, greatest to least. If null, no limit. In the case of `'shap'`: if the number of features is greater than the limit, the sum of remaining values will also be returned as `shapRemainingTotal`. Defaults to null. Cannot be set if `explanation_algorithm` is omitted.

Returns

job [PredictJob] The job computing the predictions

request_transferable_export (*prediction_intervals_size=None*)

Request generation of an exportable model file for use in an on-premise DataRobot standalone prediction environment.

This function can only be used if model export is enabled, and will only be useful if you have an on-premise environment in which to import it.

This function does not download the exported file. Use `download_export` for that.

Parameters

prediction_intervals_size [int, optional] (New in v2.19) For *time series* projects only. Represents the percentile to use for the size of the prediction intervals. Prediction intervals size must be between 1 and 100 (inclusive).

Examples

```
model = datarobot.Model.get('p-id', 'l-id')
job = model.request_transferable_export()
job.wait_for_completion()
model.download_export('my_exported_model.drmodel')

# Client must be configured to use standalone prediction server for import:
datarobot.Client(token='my-token-at-standalone-server',
                  endpoint='standalone-server-url/api/v2')

imported_model = datarobot.ImportedModel.create('my_exported_model.drmodel')
```

set_prediction_threshold (*threshold*)

Set a custom prediction threshold for the model

May not be used once `prediction_threshold_read_only` is True for this model.

Parameters

threshold [float] only used for binary classification projects. The threshold to when deciding between the positive and negative classes when making predictions. Should be between 0.0 and 1.0 (inclusive).

star_model ()

Mark the model as starred

Model stars propagate to the web application and the API, and can be used to filter when listing models.

start_advanced_tuning_session()

Start an Advanced Tuning session. Returns an object that helps set up arguments for an Advanced Tuning model execution.

As of v2.17, all models other than blenders, open source, prime, scaleout, baseline and user-created support Advanced Tuning.

Returns

AdvancedTuningSession Session for setting up and running Advanced Tuning on a model

train_datetime (*featurelist_id=None*, *training_row_count=None*, *training_duration=None*, *time_window_sample_pct=None*, *monotonic_increasing_featurelist_id=<object object>*, *monotonic_decreasing_featurelist_id=<object object>*, *use_project_settings=False*)

Train this model on a different featurelist or amount of data

Requires that this model is part of a datetime partitioned project; otherwise, an error will occur.

All durations should be specified with a duration string such as those returned by the [partitioning_methods.construct_duration_string](#) helper method. Please see [datetime partitioned project documentation](#) for more information on duration strings.

Parameters

featurelist_id [str, optional] the featurelist to use to train the model. If not specified, the featurelist of this model is used.

training_row_count [int, optional] the number of rows of data that should be used to train the model. If specified, neither `training_duration` nor `use_project_settings` may be specified.

training_duration [str, optional] a duration string specifying what time range the data used to train the model should span. If specified, neither `training_row_count` nor `use_project_settings` may be specified.

use_project_settings [bool, optional] (New in version v2.20) defaults to `False`. If `True`, indicates that the custom backtest partitioning settings specified by the user will be used to train the model and evaluate backtest scores. If specified, neither `training_row_count` nor `training_duration` may be specified.

time_window_sample_pct [int, optional] may only be specified when the requested model is a time window (e.g. duration or start and end dates). An integer between 1 and 99 indicating the percentage to sample by within the window. The points kept are determined by a random uniform sample. If specified, `training_duration` must be specified otherwise, the number of rows used to train the model and evaluate backtest scores and an error will occur.

monotonic_increasing_featurelist_id [str, optional] (New in version v2.18) optional, the id of the featurelist that defines the set of features with a monotonically increasing relationship to the target. Passing `None` disables increasing monotonicity constraint. Default (`dr.enums.MONOTONICITY_FEATURELIST_DEFAULT`) is the one specified by the blueprint.

monotonic_decreasing_featurelist_id [str, optional] (New in version v2.18) optional, the id of the featurelist that defines the set of features with a monotonically decreasing relationship to the target. Passing `None` disables decreasing monotonicity constraint. Default (`dr.enums.MONOTONICITY_FEATURELIST_DEFAULT`) is the one specified by the blueprint.

Returns

job [ModelJob] the created job to build the model

unstar_model()

Unmark the model as starred

Model stars propagate to the web application and the API, and can be used to filter when listing models.

32.20.5 Frozen Model

```
class datarobot.models.FrozenModel (id=None, processes=None, featurelist_name=None, featurelist_id=None, project_id=None, sample_pct=None, training_row_count=None, training_duration=None, training_start_date=None, training_end_date=None, model_type=None, model_category=None, is_frozen=None, blueprint_id=None, metrics=None, parent_model_id=None, monotonic_increasing_featurelist_id=None, monotonic_decreasing_featurelist_id=None, supports_monotonic_constraints=None, is_starred=None, prediction_threshold=None, prediction_threshold_read_only=None, model_number=None)
```

A model tuned with parameters which are derived from another model

All durations are specified with a duration string such as those returned by the [partitioning_methods.construct_duration_string](#) helper method. Please see [datetime partitioned project documentation](#) for more information on duration strings.

Attributes

id [str] the id of the model

project_id [str] the id of the project the model belongs to

processes [list of str] the processes used by the model

featurelist_name [str] the name of the featurelist used by the model

featurelist_id [str] the id of the featurelist used by the model

sample_pct [float] the percentage of the project dataset used in training the model

training_row_count [int or None] the number of rows of the project dataset used in training the model. In a datetime partitioned project, if specified, defines the number of rows used to train the model and evaluate backtest scores; if unspecified, either *training_duration* or *training_start_date* and *training_end_date* was used to determine that instead.

training_duration [str or None] only present for models in datetime partitioned projects. If specified, a duration string specifying the duration spanned by the data used to train the model and evaluate backtest scores.

training_start_date [datetime or None] only present for frozen models in datetime partitioned projects. If specified, the start date of the data used to train the model.

training_end_date [datetime or None] only present for frozen models in datetime partitioned projects. If specified, the end date of the data used to train the model.

model_type [str] what model this is, e.g. "Nystroem Kernel SVM Regressor"

model_category [str] what kind of model this is - "prime" for DataRobot Prime models, "blend" for blender models, and "model" for other models

is_frozen [bool] whether this model is a frozen model

parent_model_id [str] the id of the model that tuning parameters are derived from

blueprint_id [str] the id of the blueprint used in this model

metrics [dict] a mapping from each metric to the model's scores for that metric

monotonic_increasing_featurelist_id [str] optional, the id of the featurelist that defines the set of features with a monotonically increasing relationship to the target. If None, no such constraints are enforced.

monotonic_decreasing_featurelist_id [str] optional, the id of the featurelist that defines the set of features with a monotonically decreasing relationship to the target. If None, no such constraints are enforced.

supports_monotonic_constraints [bool] optional, whether this model supports enforcing monotonic constraints

is_starred [bool] whether this model marked as starred

prediction_threshold [float] for binary classification projects, the threshold used for predictions

prediction_threshold_read_only [bool] indicated whether modification of the prediction threshold is forbidden. Threshold modification is forbidden once a model has had a deployment created or predictions made via the dedicated prediction API.

model_number [integer] model number assigned to a model

classmethod **get** (*project_id*, *model_id*)

Retrieve a specific frozen model.

Parameters

project_id [str] The project's id.

model_id [str] The `model_id` of the leaderboard item to retrieve.

Returns

model [FrozenModel] The queried instance.

32.20.6 Imported Model

Note: Imported Models are used in Stand Alone Scoring Engines. If you are not an administrator of such an engine, they are not relevant to you.

```
class datarobot.models.ImportedModel (id, imported_at=None, model_id=None, target=None,
                                     featurelist_name=None, dataset_name=None,
                                     model_name=None, project_id=None, version=None, note=None, origin_url=None, imported_by_username=None, project_name=None,
                                     created_by_username=None, created_by_id=None,
                                     imported_by_id=None, display_name=None)
```

Represents an imported model available for making predictions. These are only relevant for administrators of on-premise Stand Alone Scoring Engines.

ImportedModels are trained in one DataRobot application, exported as a `.drmodel` file, and then imported for use in a Stand Alone Scoring Engine.

Attributes

id [str] id of the import

model_name [str] model type describing the model generated by DataRobot

display_name [str] manually specified human-readable name of the imported model

note [str] manually added note about this imported model

imported_at [datetime] the time the model was imported

imported_by_username [str] username of the user who imported the model

imported_by_id [str] id of the user who imported the model

origin_url [str] URL of the application the model was exported from

model_id [str] original id of the model prior to export

featurelist_name [str] name of the featurelist used to train the model

project_id [str] id of the project the model belonged to prior to export

project_name [str] name of the project the model belonged to prior to export

target [str] the target of the project the model belonged to prior to export

version [float] project version of the project the model belonged to

dataset_name [str] filename of the dataset used to create the project the model belonged to

created_by_username [str] username of the user who created the model prior to export

created_by_id [str] id of the user who created the model prior to export

classmethod create (*path, max_wait=600*)

Import a previously exported model for predictions.

Parameters

path [str] The path to the exported model file

max_wait [int, optional] Time in seconds after which model import is considered unsuccessful

classmethod get (*import_id*)

Retrieve imported model info

Parameters

import_id [str] The ID of the imported model.

Returns

imported_model [ImportedModel] The ImportedModel instance

classmethod list (*limit=None, offset=None*)

List the imported models.

Parameters

limit [int] The number of records to return. The server will use a (possibly finite) default if not specified.

offset [int] The number of records to skip.

Returns

imported_models [list[ImportedModel]]

update (*display_name=None, note=None*)

Update the display name or note for an imported model. The ImportedModel object is updated in place.

Parameters

display_name [str] The new display name.

note [str] The new note.

delete ()

Delete this imported model.

32.20.7 RatingTableModel

```
class datarobot.models.RatingTableModel (id=None, processes=None, featurelist_name=None, featurelist_id=None, project_id=None, sample_pct=None, training_row_count=None, training_duration=None, training_start_date=None, training_end_date=None, model_type=None, model_category=None, is_frozen=None, blueprint_id=None, metrics=None, rating_table_id=None, monotonic_increasing_featurelist_id=None, monotonic_decreasing_featurelist_id=None, supports_monotonic_constraints=None, is_starred=None, prediction_threshold=None, prediction_threshold_read_only=None, model_number=None)
```

A model that has a rating table.

All durations are specified with a duration string such as those returned by the `partitioning_methods.construct_duration_string` helper method. Please see [datetime partitioned project documentation](#) for more information on duration strings.

Attributes

id [str] the id of the model

project_id [str] the id of the project the model belongs to

processes [list of str] the processes used by the model

featurelist_name [str] the name of the featurelist used by the model

featurelist_id [str] the id of the featurelist used by the model

sample_pct [float or None] the percentage of the project dataset used in training the model. If the project uses datetime partitioning, the sample_pct will be None. See *training_row_count*, *training_duration*, and *training_start_date* and *training_end_date* instead.

training_row_count [int or None] the number of rows of the project dataset used in training the model. In a datetime partitioned project, if specified, defines the number of rows used to train the model and evaluate backtest scores; if unspecified, either *training_duration* or *training_start_date* and *training_end_date* was used to determine that instead.

training_duration [str or None] only present for models in datetime partitioned projects. If specified, a duration string specifying the duration spanned by the data used to train the model and evaluate backtest scores.

training_start_date [datetime or None] only present for frozen models in datetime partitioned projects. If specified, the start date of the data used to train the model.

training_end_date [datetime or None] only present for frozen models in datetime partitioned projects. If specified, the end date of the data used to train the model.

model_type [str] what model this is, e.g. "Nystroem Kernel SVM Regressor"

model_category [str] what kind of model this is - "prime" for DataRobot Prime models, "blender" for blender models, and "model" for other models

is_frozen [bool] whether this model is a frozen model

blueprint_id [str] the id of the blueprint used in this model

metrics [dict] a mapping from each metric to the model's scores for that metric

rating_table_id [str] the id of the rating table that belongs to this model

monotonic_increasing_featurelist_id [str] optional, the id of the featurelist that defines the set of features with a monotonically increasing relationship to the target. If None, no such constraints are enforced.

monotonic_decreasing_featurelist_id [str] optional, the id of the featurelist that defines the set of features with a monotonically decreasing relationship to the target. If None, no such constraints are enforced.

supports_monotonic_constraints [bool] optional, whether this model supports enforcing monotonic constraints

is_starred [bool] whether this model marked as starred

prediction_threshold [float] for binary classification projects, the threshold used for predictions

prediction_threshold_read_only [bool] indicated whether modification of the prediction threshold is forbidden. Threshold modification is forbidden once a model has had a deployment created or predictions made via the dedicated prediction API.

model_number [integer] model number assigned to a model

classmethod **get** (*project_id*, *model_id*)

Retrieve a specific rating table model

If the project does not have a rating table, a ClientError will occur.

Parameters

project_id [str] the id of the project the model belongs to

model_id [str] the id of the model to retrieve

Returns

model [RatingTableModel] the model

classmethod **create_from_rating_table** (*project_id*, *rating_table_id*)

Creates a new model from a validated rating table record. The RatingTable must not be associated with an existing model.

Parameters

project_id [str] the id of the project the rating table belongs to

rating_table_id [str] the id of the rating table to create this model from

Returns

job: Job an instance of created async job

Raises

ClientError (422) Raised if creating model from a RatingTable that failed validation

JobAlreadyRequested Raised if creating model from a RatingTable that is already associated with a RatingTableModel

advanced_tune (*params, description=None*)

Generate a new model with the specified advanced-tuning parameters

As of v2.17, all models other than blenders, open source, prime, scaleout, baseline and user-created support Advanced Tuning.

Parameters

params [dict] Mapping of parameter ID to parameter value. The list of valid parameter IDs for a model can be found by calling *get_advanced_tuning_parameters()*. This endpoint does not need to include values for all parameters. If a parameter is omitted, its *current_value* will be used.

description [unicode] Human-readable string describing the newly advanced-tuned model

Returns

ModelJob The created job to build the model

cross_validate ()

Run Cross Validation on this model.

Note: To perform Cross Validation on a new model with new parameters, use `train` instead.

Returns

ModelJob The created job to build the model

delete ()

Delete a model from the project's leaderboard.

download_export (*filepath*)

Download an exportable model file for use in an on-premise DataRobot standalone prediction environment.

This function can only be used if model export is enabled, and will only be useful if you have an on-premise environment in which to import it.

Parameters

filepath [str] The path at which to save the exported model file.

download_scoring_code (*file_name, source_code=False*)

Download scoring code JAR.

Parameters

file_name [str] File path where scoring code will be saved.

source_code [bool, optional] Set to True to download source code archive. It will not be executable.

classmethod fetch_resource_data (*url, join_endpoint=True*)

(Deprecated.) Used to acquire model data directly from its url.

Consider using *get* instead, as this is a convenience function used for development of datarobot

Parameters

url [str] The resource we are acquiring

join_endpoint [boolean, optional] Whether the client's endpoint should be joined to the URL before sending the request. Location headers are returned as absolute locations, so `will_not_need` the endpoint

Returns

model_data [dict] The queried model's data

`get_advanced_tuning_parameters()`

Get the advanced-tuning parameters available for this model.

As of v2.17, all models other than blenders, open source, prime, scaleout, baseline and user-created support Advanced Tuning.

Returns

dict A dictionary describing the advanced-tuning parameters for the current model. There are two top-level keys, *tuningDescription* and *tuningParameters*.

tuningDescription an optional value. If not *None*, then it indicates the user-specified description of this set of tuning parameter.

tuningParameters is a list of a dicts, each has the following keys

- **parameterName** : (**unicode**) name of the parameter (unique per task, see below)
- **parameterId** : (**unicode**) opaque ID string uniquely identifying parameter
- **defaultValue** : (*) default value of the parameter for the blueprint
- **currentValue** : (*) value of the parameter that was used for this model
- **taskName** : (**unicode**) name of the task that this parameter belongs to
- **constraints**: (**dict**) see the notes below

Notes

The type of *defaultValue* and *currentValue* is defined by the *constraints* structure. It will be a string or numeric Python type.

constraints is a dict with *at least one*, possibly more, of the following keys. The presence of a key indicates that the parameter may take on the specified type. (If a key is absent, this means that the parameter may not take on the specified type.) If a key on *constraints* is present, its value will be a dict containing all of the fields described below for that key.

```
"constraints": {
  "select": {
    "values": [<list (basestring or number) : possible values>]
  },
  "ascii": {},
  "unicode": {},
  "int": {
    "min": <int : minimum valid value>,
    "max": <int : maximum valid value>,
    "supports_grid_search": <bool : True if Grid Search may be
```

(continues on next page)

(continued from previous page)

```

                                requested for this param>
    },
    "float": {
        "min": <float : minimum valid value>,
        "max": <float : maximum valid value>,
        "supports_grid_search": <bool : True if Grid Search may be
                                requested for this param>
    },
    "intList": {
        "length": {
            "min_length": <int : minimum valid length>,
            "max_length": <int : maximum valid length>
            "min_val": <int : minimum valid value>,
            "max_val": <int : maximum valid value>
            "supports_grid_search": <bool : True if Grid Search may be
                                requested for this param>
        },
        "floatList": {
            "min_length": <int : minimum valid length>,
            "max_length": <int : maximum valid length>
            "min_val": <float : minimum valid value>,
            "max_val": <float : maximum valid value>
            "supports_grid_search": <bool : True if Grid Search may be
                                requested for this param>
        }
    }
}

```

The keys have meaning as follows:

- *select*: Rather than specifying a specific data type, if present, it indicates that the parameter is permitted to take on any of the specified values. Listed values may be of any string or real (non-complex) numeric type.
- *ascii*: The parameter may be a *unicode* object that encodes simple ASCII characters. (A-Z, a-z, 0-9, whitespace, and certain common symbols.) In addition to listed constraints, ASCII keys currently may not contain either newlines or semicolons.
- *unicode*: The parameter may be any Python *unicode* object.
- *int*: The value may be an object of type *int* within the specified range (inclusive). Please note that the value will be passed around using the JSON format, and some JSON parsers have undefined behavior with integers outside of the range $[-(2^{53})+1, (2^{53})-1]$.
- *float*: The value may be an object of type *float* within the specified range (inclusive).
- *intList*, *floatList*: The value may be a list of *int* or *float* objects, respectively, following constraints as specified respectively by the *int* and *float* types (above).

Many parameters only specify one key under *constraints*. If a parameter specifies multiple keys, the parameter may take on any value permitted by any key.

get_all_confusion_charts (*fallback_to_parent_insights=False*)

Retrieve a list of all confusion charts available for the model.

Parameters

fallback_to_parent_insights [bool] (New in version v2.14) Optional, if True, this will return confusion chart data for this model's parent for any source that is not available for this model and if this has a defined parent model. If omitted or False, or this model has no parent, this will not attempt to retrieve any data from this model's parent.

Returns

list of ConfusionChart Data for all available confusion charts for model.

get_all_lift_charts (*fallback_to_parent_insights=False*)

Retrieve a list of all lift charts available for the model.

Parameters

fallback_to_parent_insights [bool] (New in version v2.14) Optional, if True, this will return lift chart data for this model's parent for any source that is not available for this model and if this model has a defined parent model. If omitted or False, or this model has no parent, this will not attempt to retrieve any data from this model's parent.

Returns

list of LiftChart Data for all available model lift charts.

get_all_residuals_charts (*fallback_to_parent_insights=False*)

Retrieve a list of all lift charts available for the model.

Parameters

fallback_to_parent_insights [bool] Optional, if True, this will return residuals chart data for this model's parent for any source that is not available for this model and if this model has a defined parent model. If omitted or False, or this model has no parent, this will not attempt to retrieve any data from this model's parent.

Returns

list of ResidualsChart Data for all available model residuals charts.

get_all_roc_curves (*fallback_to_parent_insights=False*)

Retrieve a list of all ROC curves available for the model.

Parameters

fallback_to_parent_insights [bool] (New in version v2.14) Optional, if True, this will return ROC curve data for this model's parent for any source that is not available for this model and if this model has a defined parent model. If omitted or False, or this model has no parent, this will not attempt to retrieve any data from this model's parent.

Returns

list of RocCurve Data for all available model ROC curves.

get_confusion_chart (*source, fallback_to_parent_insights=False*)

Retrieve model's confusion chart for the specified source.

Parameters

source [str] Confusion chart source. Check datarobot.enums.CHART_DATA_SOURCE for possible values.

fallback_to_parent_insights [bool] (New in version v2.14) Optional, if True, this will return confusion chart data for this model's parent if the confusion chart is not available for this model and the defined parent model. If omitted or False, or there is no parent model, will not attempt to return insight data from this model's parent.

Returns

ConfusionChart Model ConfusionChart data

Raises

ClientError If the insight is not available for this model

get_cross_validation_scores (*partition=None, metric=None*)

Returns a dictionary keyed by metric showing cross validation scores per partition.

Cross Validation should already have been performed using `cross_validate` or `train`.

Note: Models that computed cross validation before this feature was added will need to be deleted and retrained before this method can be used.

Parameters

partition [float] optional, the id of the partition (1,2,3,0,4,0,etcâ€¦) to filter results by can be a whole number positive integer or float value.

metric: unicode optional name of the metric to filter to resulting cross validation scores by

Returns

cross_validation_scores: dict A dictionary keyed by metric showing cross validation scores per partition.

get_feature_effect (*source*)

Retrieve Feature Effects for the model.

Feature Effects provides partial dependence and predicted vs actual values for top-500 features ordered by feature impact score.

The partial dependence shows marginal effect of a feature on the target variable after accounting for the average effects of all other predictive features. It indicates how, holding all other variables except the feature of interest as they were, the value of this feature affects your prediction.

Requires that Feature Effects has already been computed with `request_feature_effect`.

See `get_feature_effect_metadata` for retrieving information the available sources.

Parameters

source [string] The source Feature Effects are retrieved for.

Returns

feature_effects [FeatureEffects] The feature effects data.

Raises

ClientError (404) If the feature effects have not been computed or source is not valid value.

get_feature_effect_metadata ()

Retrieve Feature Effect metadata. Response contains status and available model sources.

- Feature Fit of *training* is always available (except for the old project which supports only Feature Fit for *validation*).
- When a model is trained into *validation* or *holdout* without stacked prediction (e.g. no out-of-sample prediction in *validation* or *holdout*), Feature Effect is not available for *validation* or *holdout*.
- Feature Effect for *holdout* is not available when there is no holdout configured for the project.

source is expected parameter to retrieve Feature Effect. One of provided sources shall be used.

Returns

feature_effect_metadata: FeatureEffectMetadata**get_feature_fit** (*source*)

Retrieve Feature Fit for the model.

Feature Fit provides partial dependence and predicted vs actual values for top-500 features ordered by feature importance score.

The partial dependence shows marginal effect of a feature on the target variable after accounting for the average effects of all other predictive features. It indicates how, holding all other variables except the feature of interest as they were, the value of this feature affects your prediction.

Requires that Feature Fit has already been computed with [request_feature_effect](#).

See [get_feature_fit_metadata](#) for retrieving information the available sources.

Parameters

source [string] The source Feature Fit are retrieved for. One value of [FeatureFitMetadata.sources].

Returns

feature_fit [FeatureFit] The feature fit data.

Raises

ClientError (404) If the feature fit have not been computed or source is not valid value.

get_feature_fit_metadata ()

Retrieve Feature Fit metadata. Response contains status and available model sources.

- Feature Fit of *training* is always available (except for the old project which supports only Feature Fit for *validation*).
- When a model is trained into *validation* or *holdout* without stacked prediction (e.g. no out-of-sample prediction in *validation* or *holdout*), Feature Fit is not available for *validation* or *holdout*.
- Feature Fit for *holdout* is not available when there is no holdout configured for the project.

source is expected parameter to retrieve Feature Fit. One of provided sources shall be used.

Returns**feature_effect_metadata: FeatureFitMetadata****get_feature_impact** (*with_metadata=False*)

Retrieve the computed Feature Impact results, a measure of the relevance of each feature in the model.

Feature Impact is computed for each column by creating new data with that column randomly permuted (but the others left unchanged), and seeing how the error metric score for the predictions is affected. The `impactUnnormalized` is how much worse the error metric score is when making predictions on this modified data. The `impactNormalized` is normalized so that the largest value is 1. In both cases, larger values indicate more important features.

If a feature is a redundant feature, i.e. once other features are considered it doesn't contribute much in addition, the `redundantWith` value is the name of feature that has the highest correlation with this feature. Note that redundancy detection is only available for jobs run after the addition of this feature. When retrieving data that predates this functionality, a `NoRedundancyImpactAvailable` warning will be used.

Elsewhere this technique is sometimes called `Permutation Importance`.

Requires that Feature Impact has already been computed with `request_feature_impact`.

Parameters

with_metadata [bool] The flag indicating if the result should include the metadata as well.

Returns

list or dict The feature impact data response depends on the `with_metadata` parameter. The response is either a dict with metadata and a list with actual data or just a list with that data.

Each List item is a dict with the keys `featureName`, `impactNormalized`, and `impactUnnormalized`, `redundantWith` and `count`.

For dict response available keys are:

- **featureImpacts** - Feature Impact data as a dictionary. Each item is a dict with keys: `featureName`, `impactNormalized`, and `impactUnnormalized`, and `redundantWith`.
- **shapBased** - A boolean that indicates whether Feature Impact was calculated using Shapley values.
- **ranRedundancyDetection** - A boolean that indicates whether redundant feature identification was run while calculating this Feature Impact.
- **rowCount** - An integer or None that indicates the number of rows that was used to calculate Feature Impact. For the Feature Impact calculated with the default logic, without specifying the `rowCount`, we return None here.
- `count` - An integer with the number of features under the `featureImpacts`.

Raises

ClientError (404) If the feature impacts have not been computed.

`get_features_used()`

Query the server to determine which features were used.

Note that the data returned by this method is possibly different than the names of the features in the `featurelist` used by this model. This method will return the raw features that must be supplied in order for predictions to be generated on a new set of data. The `featurelist`, in contrast, would also include the names of derived features.

Returns

features [list of str] The names of the features used in the model.

`get_frozen_child_models()`

Retrieves the ids for all the models that are frozen from this model

Returns

A list of Models

`get_leaderboard_ui_permalink()`

Returns

url [str] Permanent static hyperlink to this model at leaderboard.

`get_lift_chart(source, fallback_to_parent_insights=False)`

Retrieve model lift chart for the specified source.

Parameters

source [str] Lift chart data source. Check `datarobot.enums.CHART_DATA_SOURCE` for possible values.

fallback_to_parent_insights [bool] (New in version v2.14) Optional, if True, this will return lift chart data for this model's parent if the lift chart is not available for this model and the model has a defined parent model. If omitted or False, or there is no parent model, will not attempt to return insight data from this model's parent.

Returns

LiftChart Model lift chart data

Raises

ClientError If the insight is not available for this model

`get_missing_report_info()`

Retrieve a model missing data report on training data that can be used to understand missing values treatment in a model. Report consists of missing values reports for features which took part in modelling and are numeric or categorical.

Returns

An iterable of MissingReportPerFeature The queried model missing report, sorted by missing count (DESCENDING order).

`get_model_blueprint_chart()`

Retrieve a model blueprint chart that can be used to understand data flow in blueprint.

Returns

ModelBlueprintChart The queried model blueprint chart.

`get_model_blueprint_documents()`

Get documentation for tasks used in this model.

Returns

list of BlueprintTaskDocument All documents available for the model.

`get_multiclass_feature_impact()`

For multiclass it's possible to calculate feature impact separately for each target class. The method for calculation is exactly the same, calculated in one-vs-all style for each target class.

Requires that Feature Impact has already been computed with `request_feature_impact`.

Returns

feature_impacts [list of dict] The feature impact data. Each item is a dict with the keys `featureImpacts` (list), `class` (str). Each item in `featureImpacts` is a dict with the keys `featureName`, `impactNormalized`, and `impactUnnormalized`, and `redundantWith`.

Raises

ClientError (404) If the multiclass feature impacts have not been computed.

`get_multiclass_lift_chart(source, fallback_to_parent_insights=False)`

Retrieve model lift chart for the specified source.

Parameters

source [str] Lift chart data source. Check `datarobot.enums.CHART_DATA_SOURCE` for possible values.

fallback_to_parent_insights [bool] Optional, if True, this will return lift chart data for this model's parent if the lift chart is not available for this model and the model has a defined parent model. If omitted or False, or there is no parent model, will not attempt to return insight data from this model's parent.

Returns

list of LiftChart Model lift chart data for each saved target class

Raises

ClientError If the insight is not available for this model

`get_num_estimators_trained()`

Retrieves the number of estimators trained by early-stopping tree-based models

– versionadded:: v2.22

Returns

projectId: str id of project containing the model

modelId: str id of the model

data: array list of *numEstimatorsItem* objects, one for each modeling stage.

numEstimatorsItem will be of the form:

stage: str indicates the modeling stage (for multi-stage models); None of single-stage models

numIterations: int the number of estimators or iterations trained by the model

`get_or_request_feature_effect(source, max_wait=600, row_count=None)`

Retrieve feature effect for the model, requesting a job if it hasn't been run previously

See [get_feature_effect_metadata](#) for retrieving information of source.

Parameters

max_wait [int, optional] The maximum time to wait for a requested feature effect job to complete before erroring

row_count [int, optional] (New in version v2.21) The sample size to use for Feature Impact computation. Minimum is 10 rows. Maximum is 100000 rows or the training sample size of the model, whichever is less.

source [string] The source Feature Effects are retrieved for.

Returns

feature_effects [FeatureEffects] The feature effects data.

`get_or_request_feature_fit(source, max_wait=600)`

Retrieve feature fit for the model, requesting a job if it hasn't been run previously

See [get_feature_fit_metadata](#) for retrieving information of source.

Parameters

max_wait [int, optional] The maximum time to wait for a requested feature fit job to complete before erroring

source [string] The source Feature Fit are retrieved for. One value of [FeatureFitMetadata.sources].

Returns

feature_effects [FeatureFit] The feature fit data.

get_or_request_feature_impact (*max_wait=600, **kwargs*)

Retrieve feature impact for the model, requesting a job if it hasn't been run previously

Parameters

max_wait [int, optional] The maximum time to wait for a requested feature impact job to complete before erroring

****kwargs** Arbitrary keyword arguments passed to [request_feature_impact](#).

Returns

feature_impacts [list or dict] The feature impact data. See [get_feature_impact](#) for the exact schema.

get_parameters ()

Retrieve model parameters.

Returns

ModelParameters Model parameters for this model.

get_pareto_front ()

Retrieve the Pareto Front for a Eureqa model.

This method is only supported for Eureqa models.

Returns

ParetoFront Model ParetoFront data

get_prime_eligibility ()

Check if this model can be approximated with DataRobot Prime

Returns

prime_eligibility [dict] a dict indicating whether a model can be approximated with DataRobot Prime (key *can_make_prime*) and why it may be ineligible (key *message*)

get_residuals_chart (*source, fallback_to_parent_insights=False*)

Retrieve model residuals chart for the specified source.

Parameters

source [str] Residuals chart data source. Check `datarobot.enums.CHART_DATA_SOURCE` for possible values.

fallback_to_parent_insights [bool] Optional, if True, this will return residuals chart data for this model's parent if the residuals chart is not available for this model and the model has a defined parent model. If omitted or False, or there is no parent model, will not attempt to return residuals data from this model's parent.

Returns

ResidualsChart Model residuals chart data

Raises

ClientError If the insight is not available for this model

get_roc_curve (*source, fallback_to_parent_insights=False*)

Retrieve model ROC curve for the specified source.

Parameters

source [str] ROC curve data source. Check `datarobot.enums.CHART_DATA_SOURCE` for possible values.

fallback_to_parent_insights [bool] (New in version v2.14) Optional, if True, this will return ROC curve data for this model's parent if the ROC curve is not available for this model and the model has a defined parent model. If omitted or False, or there is no parent model, will not attempt to return data from this model's parent.

Returns

RocCurve Model ROC curve data

Raises

ClientError If the insight is not available for this model

`get_rulesets()`

List the rulesets approximating this model generated by DataRobot Prime

If this model hasn't been approximated yet, will return an empty list. Note that these are rulesets approximating this model, not rulesets used to construct this model.

Returns

rulesets [list of Ruleset]

`get_supported_capabilities()`

Retrieves a summary of the capabilities supported by a model.

New in version v2.14.

Returns

supportsBlending: bool whether the model supports blending

supportsMonotonicConstraints: bool whether the model supports monotonic constraints

hasWordCloud: bool whether the model has word cloud data available

eligibleForPrime: bool whether the model is eligible for Prime

hasParameters: bool whether the model has parameters that can be retrieved

supportsCodeGeneration: bool (New in version v2.18) whether the model supports code generation

supportsShap: bool

(New in version v2.18) True if the model supports Shapley package. i.e. Shapley based feature Importance

supportsEarlyStopping: bool (New in version v2.22) True if this is an early stopping tree-based model and number of trained iterations can be retrieved.

`get_word_cloud(exclude_stop_words=False)`

Retrieve a word cloud data for the model.

Parameters

exclude_stop_words [bool, optional] Set to True if you want stopwords filtered out of response.

Returns

WordCloud Word cloud data for the model.

open_model_browser()

Opens model at project leaderboard in web browser.

Note: If text-mode browsers are used, the calling process will block until the user exits the browser.

request_approximation()

Request an approximation of this model using DataRobot Prime

This will create several rulesets that could be used to approximate this model. After comparing their scores and rule counts, the code used in the approximation can be downloaded and run locally.

Returns

job [Job] the job generating the rulesets

request_external_test (*dataset_id*, *actual_value_column=None*)

Request external test to compute scores and insights on an external test dataset

Parameters

dataset_id [string] The dataset to make predictions against (as uploaded from `Project.upload_dataset`)

actual_value_column [string, optional] (New in version v2.21) For time series unsupervised projects only. Actual value column can be used to calculate the classification metrics and insights on the prediction dataset. Can't be provided with the `forecast_point` parameter.

Returns

Job

job [Job] a Job representing external dataset insights computation

request_feature_effect (*row_count=None*)

Request feature effects to be computed for the model.

See [get_feature_effect](#) for more information on the result of the job.

Parameters

row_count [int] (New in version v2.21) The sample size to use for Feature Impact computation. Minimum is 10 rows. Maximum is 100000 rows or the training sample size of the model, whichever is less.

Returns

job [Job] A Job representing the feature effect computation. To get the completed feature effect data, use `job.get_result` or `job.get_result_when_complete`.

Raises

JobAlreadyRequested (422) If the feature effect have already been requested.

request_feature_fit()

Request feature fit to be computed for the model.

See [get_feature_effect](#) for more information on the result of the job.

Returns

job [Job] A Job representing the feature fit computation. To get the completed feature fit data, use `job.get_result` or `job.get_result_when_complete`.

Raises

JobAlreadyRequested (422) If the feature effect have already been requested.

request_feature_impact (*row_count=None, with_metadata=False*)

Request feature impacts to be computed for the model.

See [get_feature_impact](#) for more information on the result of the job.

Parameters

row_count [int] The sample size (specified in rows) to use for Feature Impact computation. This is not supported for unsupervised, multi-class (that has a separate method) and time series projects.

Returns

job [Job] A Job representing the feature impact computation. To get the completed feature impact data, use *job.get_result* or *job.get_result_when_complete*.

Raises

JobAlreadyRequested (422) If the feature impacts have already been requested.

request_frozen_datetime_model (*training_row_count=None, training_duration=None, training_start_date=None, training_end_date=None, time_window_sample_pct=None*)

Train a new frozen model with parameters from this model

Requires that this model belongs to a datetime partitioned project. If it does not, an error will occur when submitting the job.

Frozen models use the same tuning parameters as their parent model instead of independently optimizing them to allow efficiently retraining models on larger amounts of the training data.

In addition of *training_row_count* and *training_duration*, frozen datetime models may be trained on an exact date range. Only one of *training_row_count*, *training_duration*, or *training_start_date* and *training_end_date* should be specified.

Models specified using *training_start_date* and *training_end_date* are the only ones that can be trained into the holdout data (once the holdout is unlocked).

All durations should be specified with a duration string such as those returned by the [partitioning_methods.construct_duration_string](#) helper method. Please see [datetime partitioned project documentation](#) for more information on duration strings.

Parameters

training_row_count [int, optional] the number of rows of data that should be used to train the model. If specified, *training_duration* may not be specified.

training_duration [str, optional] a duration string specifying what time range the data used to train the model should span. If specified, *training_row_count* may not be specified.

training_start_date [datetime.datetime, optional] the start date of the data to train to model on. Only rows occurring at or after this datetime will be used. If *training_start_date* is specified, *training_end_date* must also be specified.

training_end_date [datetime.datetime, optional] the end date of the data to train the model on. Only rows occurring strictly before this datetime will be used. If *training_end_date* is specified, *training_start_date* must also be specified.

time_window_sample_pct [int, optional] may only be specified when the requested model is a time window (e.g. duration or start and end dates). An integer between 1 and 99 indicating the percentage to sample by within the window. The points kept are determined by a random uniform sample. If specified, *training_duration* must be specified otherwise, the number of rows used to train the model and evaluate backtest scores and an error will occur.

Returns

model_job [ModelJob] the modeling job training a frozen model

request_frozen_model (*sample_pct=None, training_row_count=None*)

Train a new frozen model with parameters from this model

Note: This method only works if project the model belongs to is *not* datetime partitioned. If it is, use `request_frozen_datetime_model` instead.

Frozen models use the same tuning parameters as their parent model instead of independently optimizing them to allow efficiently retraining models on larger amounts of the training data.

Parameters

sample_pct [float] optional, the percentage of the dataset to use with the model. If not provided, will use the value from this model.

training_row_count [int] (New in version v2.9) optional, the integer number of rows of the dataset to use with the model. Only one of *sample_pct* and *training_row_count* should be specified.

Returns

model_job [ModelJob] the modeling job training a frozen model

request_predictions (*dataset_id,* *include_prediction_intervals=None,* *prediction_intervals_size=None,* *forecast_point=None,* *predictions_start_date=None,* *predictions_end_date=None,* *actual_value_column=None,* *explanation_algorithm=None,* *max_explanations=None*)

Request predictions against a previously uploaded dataset

Parameters

dataset_id [string] The dataset to make predictions against (as uploaded from `Project.upload_dataset`)

include_prediction_intervals [bool, optional] (New in v2.16) For *time series* projects only. Specifies whether prediction intervals should be calculated for this request. Defaults to True if *prediction_intervals_size* is specified, otherwise defaults to False.

prediction_intervals_size [int, optional] (New in v2.16) For *time series* projects only. Represents the percentile to use for the size of the prediction intervals. Defaults to 80 if *include_prediction_intervals* is True. Prediction intervals size must be between 1 and 100 (inclusive).

forecast_point [datetime.datetime or None, optional] (New in version v2.20) For time series projects only. This is the default point relative to which predictions will be generated, based on the forecast window of the project. See the time series [prediction documentation](#) for more information.

predictions_start_date [datetime.datetime or None, optional] (New in version v2.20) For time series projects only. The start date for bulk predictions. Note that this parameter is for generating historical predictions using the training data. This parameter should be provided in conjunction with *predictions_end_date*. Can't be provided with the *forecast_point* parameter.

predictions_end_date [datetime.datetime or None, optional] (New in version v2.20) For time series projects only. The end date for bulk predictions, exclusive. Note that this parameter is for generating historical predictions using the training data. This parameter

should be provided in conjunction with `predictions_start_date`. Can't be provided with the `forecast_point` parameter.

actual_value_column [string, optional] (New in version v2.21) For time series unsupervised projects only. Actual value column can be used to calculate the classification metrics and insights on the prediction dataset. Can't be provided with the `forecast_point` parameter.

explanation_algorithm: (New in version v2.21) optional; If set to `shap`, the response will include prediction explanations based on the SHAP explainer (SHapley Additive exPlanations). Defaults to null (no prediction explanations).

max_explanations: (New in version v2.21) optional; specifies the maximum number of explanation values that should be returned for each row, ordered by absolute value, greatest to least. If null, no limit. In the case of `shap`: if the number of features is greater than the limit, the sum of remaining values will also be returned as `shapRemainingTotal`. Defaults to null. Cannot be set if `explanation_algorithm` is omitted.

Returns

job [PredictJob] The job computing the predictions

request_training_predictions (*data_subset*, *explanation_algorithm=None*,
max_explanations=None)

Start a job to build training predictions

Parameters

data_subset [str] data set definition to build predictions on. Choices are:

- *dr.enums.DATA_SUBSET.ALL* or string *all* for all data available. Not valid for models in datetime partitioned projects
- *dr.enums.DATA_SUBSET.VALIDATION_AND_HOLDOUT* or string *validationAndHoldout* for all data except training set. Not valid for models in datetime partitioned projects
- *dr.enums.DATA_SUBSET.HOLDOUT* or string *holdout* for holdout data set only
- *dr.enums.DATA_SUBSET.ALL_BACKTESTS* or string *allBacktests* for downloading the predictions for all backtest validation folds. Requires the model to have successfully scored all backtests. Datetime partitioned projects only.

explanation_algorithm [dr.enums.EXPLANATIONS_ALGORITHM] (New in v2.21) Optional. If set to *dr.enums.EXPLANATIONS_ALGORITHM.SHAP*, the response will include prediction explanations based on the SHAP explainer (SHapley Additive exPlanations). Defaults to *None* (no prediction explanations).

max_explanations [int] (New in v2.21) Optional. Specifies the maximum number of explanation values that should be returned for each row, ordered by absolute value, greatest to least. In the case of *dr.enums.EXPLANATIONS_ALGORITHM.SHAP*: If not set, explanations are returned for all features. If the number of features is greater than the `max_explanations`, the sum of remaining values will also be returned as `shap_remaining_total`. Max 100. Defaults to null for datasets narrower than 100 columns, defaults to 100 for datasets wider than 100 columns. Is ignored if `explanation_algorithm` is not set.

Returns

Job an instance of created async job

request_transferable_export (*prediction_intervals_size=None*)

Request generation of an exportable model file for use in an on-premise DataRobot standalone prediction environment.

This function can only be used if model export is enabled, and will only be useful if you have an on-premise environment in which to import it.

This function does not download the exported file. Use `download_export` for that.

Parameters

prediction_intervals_size [int, optional] (New in v2.19) For *time series* projects only. Represents the percentile to use for the size of the prediction intervals. Prediction intervals size must be between 1 and 100 (inclusive).

Examples

```
model = datarobot.Model.get('p-id', 'l-id')
job = model.request_transferable_export()
job.wait_for_completion()
model.download_export('my_exported_model.drmodel')

# Client must be configured to use standalone prediction server for import:
datarobot.Client(token='my-token-at-standalone-server',
                  endpoint='standalone-server-url/api/v2')

imported_model = datarobot.ImportedModel.create('my_exported_model.drmodel')
```

retrain (*sample_pct=None, featurelist_id=None, training_row_count=None*)

Submit a job to the queue to train a blender model.

Parameters

sample_pct: str, optional The sample size in percents (1 to 100) to use in training. If this parameter is used then `training_row_count` should not be given.

featurelist_id [str, optional] The featurelist id

training_row_count [str, optional] The number of rows to train the model. If this parameter is used then `sample_pct` should not be given.

Returns

job [ModelJob] The created job that is retraining the model

set_prediction_threshold (*threshold*)

Set a custom prediction threshold for the model

May not be used once `prediction_threshold_read_only` is True for this model.

Parameters

threshold [float] only used for binary classification projects. The threshold to when deciding between the positive and negative classes when making predictions. Should be between 0.0 and 1.0 (inclusive).

star_model ()

Mark the model as starred

Model stars propagate to the web application and the API, and can be used to filter when listing models.

start_advanced_tuning_session ()

Start an Advanced Tuning session. Returns an object that helps set up arguments for an Advanced Tuning model execution.

As of v2.17, all models other than blenders, open source, prime, scaleout, baseline and user-created support Advanced Tuning.

Returns

AdvancedTuningSession Session for setting up and running Advanced Tuning on a model

train (*sample_pct=None*, *featurelist_id=None*, *scoring_type=None*, *training_row_count=None*, *monotonic_increasing_featurelist_id=<object object>*, *monotonic_decreasing_featurelist_id=<object object>*)

Train the blueprint used in model on a particular featurelist or amount of data.

This method creates a new training job for worker and appends it to the end of the queue for this project. After the job has finished you can get the newly trained model by retrieving it from the project leaderboard, or by retrieving the result of the job.

Either *sample_pct* or *training_row_count* can be used to specify the amount of data to use, but not both. If neither are specified, a default of the maximum amount of data that can safely be used to train any blueprint without going into the validation data will be selected.

In smart-sampled projects, *sample_pct* and *training_row_count* are assumed to be in terms of rows of the minority class.

Note: For datetime partitioned projects, see *train_datetime* instead.

Parameters

sample_pct [float, optional] The amount of data to use for training, as a percentage of the project dataset from 0 to 100.

featurelist_id [str, optional] The identifier of the featurelist to use. If not defined, the featurelist of this model is used.

scoring_type [str, optional] Either `SCORING_TYPE.validation` or `SCORING_TYPE.cross_validation`. `SCORING_TYPE.validation` is available for every partitioning type, and indicates that the default model validation should be used for the project. If the project uses a form of cross-validation partitioning, `SCORING_TYPE.cross_validation` can also be used to indicate that all of the available training/validation combinations should be used to evaluate the model.

training_row_count [int, optional] The number of rows to use to train the requested model.

monotonic_increasing_featurelist_id [str] (new in version 2.11) optional, the id of the featurelist that defines the set of features with a monotonically increasing relationship to the target. Passing `None` disables increasing monotonicity constraint. Default (`dr.enums.MONOTONICITY_FEATURELIST_DEFAULT`) is the one specified by the blueprint.

monotonic_decreasing_featurelist_id [str] (new in version 2.11) optional, the id of the featurelist that defines the set of features with a monotonically decreasing relationship to the target. Passing `None` disables decreasing monotonicity constraint. Default (`dr.enums.MONOTONICITY_FEATURELIST_DEFAULT`) is the one specified by the blueprint.

Returns

model_job_id [str] id of created job, can be used as parameter to `ModelJob.get` method or `wait_for_async_model_creation` function

Examples

```
project = Project.get('p-id')
model = Model.get('p-id', 'l-id')
model_job_id = model.train(training_row_count=project.max_train_rows)
```

train_datetime (*featurelist_id=None, training_row_count=None, training_duration=None, time_window_sample_pct=None, monotonic_increasing_featurelist_id=<object object>, monotonic_decreasing_featurelist_id=<object object>, use_project_settings=False*)

Train this model on a different featurelist or amount of data

Requires that this model is part of a datetime partitioned project; otherwise, an error will occur.

All durations should be specified with a duration string such as those returned by the [partitioning_methods.construct_duration_string](#) helper method. Please see [datetime partitioned project documentation](#) for more information on duration strings.

Parameters

featurelist_id [str, optional] the featurelist to use to train the model. If not specified, the featurelist of this model is used.

training_row_count [int, optional] the number of rows of data that should be used to train the model. If specified, neither `training_duration` nor `use_project_settings` may be specified.

training_duration [str, optional] a duration string specifying what time range the data used to train the model should span. If specified, neither `training_row_count` nor `use_project_settings` may be specified.

use_project_settings [bool, optional] (New in version v2.20) defaults to `False`. If `True`, indicates that the custom backtest partitioning settings specified by the user will be used to train the model and evaluate backtest scores. If specified, neither `training_row_count` nor `training_duration` may be specified.

time_window_sample_pct [int, optional] may only be specified when the requested model is a time window (e.g. duration or start and end dates). An integer between 1 and 99 indicating the percentage to sample by within the window. The points kept are determined by a random uniform sample. If specified, `training_duration` must be specified otherwise, the number of rows used to train the model and evaluate backtest scores and an error will occur.

monotonic_increasing_featurelist_id [str, optional] (New in version v2.18) optional, the id of the featurelist that defines the set of features with a monotonically increasing relationship to the target. Passing `None` disables increasing monotonicity constraint. Default (`dr.enums.MONOTONICITY_FEATURELIST_DEFAULT`) is the one specified by the blueprint.

monotonic_decreasing_featurelist_id [str, optional] (New in version v2.18) optional, the id of the featurelist that defines the set of features with a monotonically decreasing relationship to the target. Passing `None` disables decreasing monotonicity constraint. Default (`dr.enums.MONOTONICITY_FEATURELIST_DEFAULT`) is the one specified by the blueprint.

Returns

job [ModelJob] the created job to build the model

unstar_model()

Unmark the model as starred

Model stars propagate to the web application and the API, and can be used to filter when listing models.

32.20.8 Advanced Tuning

class `datarobot.models.advanced_tuning.AdvancedTuningSession` (*model*)

A session enabling users to configure and run advanced tuning for a model.

Every model contains a set of one or more tasks. Every task contains a set of zero or more parameters. This class allows tuning the values of each parameter on each task of a model, before running that model.

This session is client-side only and is not persistent. Only the final model, constructed when *run* is called, is persisted on the DataRobot server.

Attributes

description [basestring] Description for the new advance-tuned model. Defaults to the same description as the base model.

get_task_names ()

Get the list of task names that are available for this model

Returns

list(basestring) List of task names

get_parameter_names (*task_name*)

Get the list of parameter names available for a specific task

Returns

list(basestring) List of parameter names

set_parameter (*value*, *task_name=None*, *parameter_name=None*, *parameter_id=None*)

Set the value of a parameter to be used

The caller must supply enough of the optional arguments to this function to uniquely identify the parameter that is being set. For example, a less-common parameter name such as `building_block_complementary_error_function` might only be used once (if at all) by a single task in a model. In which case it may be sufficient to simply specify `parameter_name`. But a more-common name such as `random_seed` might be used by several of the model's tasks, and it may be necessary to also specify `task_name` to clarify which task's random seed is to be set. This function only affects client-side state. It will not check that the new parameter value(s) are valid.

Parameters

task_name [basestring] Name of the task whose parameter needs to be set

parameter_name [basestring] Name of the parameter to set

parameter_id [basestring] ID of the parameter to set

value [int, float, list, or basestring] New value for the parameter, with legal values determined by the parameter being set

Raises

NoParametersFoundException if no matching parameters are found.

NonUniqueParametersException if multiple parameters matched the specified filtering criteria

get_parameters ()

Returns the set of parameters available to this model

The returned parameters have one additional key, `value`, reflecting any new values that have been set in this `AdvancedTuningSession`. When the session is run, `value` will be used, or if it is unset, `current_value`.

Returns

parameters [dict] `Parameters` dictionary, same as specified on `Model.get_advanced_tuning_params`.

An additional field is added per parameter to the `tuningParameters` list in the dictionary:

value [int, float, list, or basestring] The current value of the parameter. *None* if none has been specified.

run()

Submit this model for Advanced Tuning.

Returns

datarobot.models.modeljob.ModelJob The created job to build the model

32.21 ModelJob

`datarobot.models.modeljob.wait_for_async_model_creation(project_id, model_job_id, max_wait=600)`

Given a Project id and ModelJob id poll for status of process responsible for model creation until model is created.

Parameters

project_id [str] The identifier of the project

model_job_id [str] The identifier of the ModelJob

max_wait [int, optional] Time in seconds after which model creation is considered unsuccessful

Returns

model [Model] Newly created model

Raises

AsyncModelCreationError Raised if status of fetched ModelJob object is error

AsyncTimeoutError Model wasn't created in time, specified by `max_wait` parameter

class `datarobot.models.ModelJob(data, completed_resource_url=None)`

Tracks asynchronous work being done within a project

Attributes

id [int] the id of the job

project_id [str] the id of the project the job belongs to

status [str] the status of the job - will be one of `datarobot.enums.QUEUE_STATUS`

job_type [str] what kind of work the job is doing - will be `model` for modeling jobs

is_blocked [bool] if true, the job is blocked (cannot be executed) until its dependencies are resolved

sample_pct [float] the percentage of the project's dataset used in this modeling job

model_type [str] the model this job builds (e.g. "Nystroem Kernel SVM Regressor")

processes [list of str] the processes used by the model

featurelist_id [str] the id of the featurelist used in this modeling job

blueprint [Blueprint] the blueprint used in this modeling job

classmethod from_job (*job*)

Transforms a generic Job into a ModelJob

Parameters

job: Job A generic job representing a ModelJob

Returns

model_job: ModelJob A fully populated ModelJob with all the details of the job

Raises

ValueError: If the generic Job was not a model job, e.g. `job_type != JOB_TYPE.MODEL`

classmethod get (*project_id, model_job_id*)

Fetches one ModelJob. If the job finished, raises PendingJobFinished exception.

Parameters

project_id [str] The identifier of the project the model belongs to

model_job_id [str] The identifier of the model_job

Returns

model_job [ModelJob] The pending ModelJob

Raises

PendingJobFinished If the job being queried already finished, and the server is re-routing to the finished model.

AsyncFailureError Querying this resource gave a status code other than 200 or 303

classmethod get_model (*project_id, model_job_id*)

Fetches a finished model from the job used to create it.

Parameters

project_id [str] The identifier of the project the model belongs to

model_job_id [str] The identifier of the model_job

Returns

model [Model] The finished model

Raises

JobNotFinished If the job has not finished yet

AsyncFailureError Querying the model_job in question gave a status code other than 200 or 303

cancel ()

Cancel this job. If this job has not finished running, it will be removed and canceled.

get_result (*params=None*)

Parameters

params [dict or None] Query parameters to be added to request to get results.

For featureEffects and featureFit, source param is required to define source, otherwise the default is ‘training’

Returns

result [object]

Return type depends on the job type:

- for model jobs, a Model is returned
- for predict jobs, a pandas.DataFrame (with predictions) is returned
- for featureImpact jobs, a list of dicts by default (see `with_metadata` parameter of the `FeatureImpactJob` class and its `get()` method).
- for primeRulesets jobs, a list of Rulesets
- for primeModel jobs, a PrimeModel
- for primeDownloadValidation jobs, a PrimeFile
- for reasonCodesInitialization jobs, a ReasonCodesInitialization
- for reasonCodes jobs, a ReasonCodes
- for predictionExplanationInitialization jobs, a PredictionExplanationsInitialization
- for predictionExplanations jobs, a PredictionExplanations
- for featureEffects, a FeatureEffects
- for featureFit, a FeatureFit

Raises

JobNotFinished If the job is not finished, the result is not available.

AsyncProcessUnsuccessfulError If the job errored or was aborted

get_result_when_complete (*max_wait=600, params=None*)

Parameters

max_wait [int, optional] How long to wait for the job to finish.

params [dict, optional] Query parameters to be added to request.

Returns

result: object Return type is the same as would be returned by *Job.get_result*.

Raises

AsyncTimeoutError If the job does not finish in time

AsyncProcessUnsuccessfulError If the job errored or was aborted

refresh()

Update this object with the latest job data from the server.

wait_for_completion (*max_wait=600*)

Waits for job to complete.

Parameters

max_wait [int, optional] How long to wait for the job to finish.

32.22 Pareto Front

class datarobot.models.pareto_front.**ParetoFront** (*project_id, error_metric, hyperparameters, target_type, solutions*)

Pareto front data for a Eureqa model.

The pareto front reflects the tradeoffs between error and complexity for particular model. The solutions reflect possible Eureqa models that are different levels of complexity. By default, only one solution will have a corresponding model, but models can be created for each solution.

Attributes

project_id [str] the ID of the project the model belongs to

error_metric [str] Eureqa error-metric identifier used to compute error metrics for this search. Note that Eureqa error metrics do NOT correspond 1:1 with DataRobot error metrics – the available metrics are not the same, and are computed from a subset of the training data rather than from the validation data.

hyperparameters [dict] Hyperparameters used by this run of the Eureqa blueprint

target_type [str] Indicating what kind of modeling is being done in this project, either “Regression”, “Binary” (Binary classification), or “Multiclass” (Multiclass classification).

solutions [list(Solution)] Solutions that Eureqa has found to model this data. Some solutions will have greater accuracy. Others will have slightly less accuracy but will use simpler expressions.

class datarobot.models.pareto_front.**Solution** (*eureqa_solution_id, complexity, error, expression, expression_annotated, best_model, project_id*)

Eureqa Solution.

A solution represents a possible Eureqa model; however not all solutions have models associated with them. It must have a model created before it can be used to make predictions, etc.

Attributes

eureqa_solution_id: str ID of this Solution

complexity: int Complexity score for this solution. Complexity score is a function of the mathematical operators used in the current solution. The Complexity calculation can be tuned via model hyperparameters.

error: float Error for the current solution, as computed by Eureqa using the “error_metric” error metric.

expression: str Eureqa model equation string.

expression_annotated: str Eureqa model equation string with variable names tagged for easy identification.

best_model: bool True, if the model is determined to be the best

create_model()

Add this solution to the leaderboard, if it is not already present.

32.23 Partitioning

class `datarobot.RandomCV` (*holdout_pct*, *reps*, *seed=0*)

A partition in which observations are randomly assigned to cross-validation groups and the holdout set.

Parameters

holdout_pct [int] the desired percentage of dataset to assign to holdout set

reps [int] number of cross validation folds to use

seed [int] a seed to use for randomization

class `datarobot.StratifiedCV` (*holdout_pct*, *reps*, *seed=0*)

A partition in which observations are randomly assigned to cross-validation groups and the holdout set, preserving in each group the same ratio of positive to negative cases as in the original data.

Parameters

holdout_pct [int] the desired percentage of dataset to assign to holdout set

reps [int] number of cross validation folds to use

seed [int] a seed to use for randomization

class `datarobot.GroupCV` (*holdout_pct*, *reps*, *partition_key_cols*, *seed=0*)

A partition in which one column is specified, and rows sharing a common value for that column are guaranteed to stay together in the partitioning into cross-validation groups and the holdout set.

Parameters

holdout_pct [int] the desired percentage of dataset to assign to holdout set

reps [int] number of cross validation folds to use

partition_key_cols [list] a list containing a single string, where the string is the name of the column whose values should remain together in partitioning

seed [int] a seed to use for randomization

class `datarobot.UserCV` (*user_partition_col*, *cv_holdout_level*, *seed=0*)

A partition where the cross-validation folds and the holdout set are specified by the user.

Parameters

user_partition_col [string] the name of the column containing the partition assignments

cv_holdout_level the value of the partition column indicating a row is part of the holdout set

seed [int] a seed to use for randomization

class `datarobot.RandomTVH` (*holdout_pct*, *validation_pct*, *seed=0*)

Specifies a partitioning method in which rows are randomly assigned to training, validation, and holdout.

Parameters

holdout_pct [int] the desired percentage of dataset to assign to holdout set

validation_pct [int] the desired percentage of dataset to assign to validation set

seed [int] a seed to use for randomization

class `datarobot.UserTVH` (*user_partition_col*, *training_level*, *validation_level*, *holdout_level*, *seed=0*)

Specifies a partitioning method in which rows are assigned by the user to training, validation, and holdout sets.

Parameters

user_partition_col [string] the name of the column containing the partition assignments

training_level the value of the partition column indicating a row is part of the training set

validation_level the value of the partition column indicating a row is part of the validation set

holdout_level the value of the partition column indicating a row is part of the holdout set (use None if you want no holdout set)

seed [int] a seed to use for randomization

class `datarobot.StratifiedTVH` (*holdout_pct, validation_pct, seed=0*)

A partition in which observations are randomly assigned to train, validation, and holdout sets, preserving in each group the same ratio of positive to negative cases as in the original data.

Parameters

holdout_pct [int] the desired percentage of dataset to assign to holdout set

validation_pct [int] the desired percentage of dataset to assign to validation set

seed [int] a seed to use for randomization

class `datarobot.GroupTVH` (*holdout_pct, validation_pct, partition_key_cols, seed=0*)

A partition in which one column is specified, and rows sharing a common value for that column are guaranteed to stay together in the partitioning into the training, validation, and holdout sets.

Parameters

holdout_pct [int] the desired percentage of dataset to assign to holdout set

validation_pct [int] the desired percentage of dataset to assign to validation set

partition_key_cols [list] a list containing a single string, where the string is the name of the column whose values should remain together in partitioning

seed [int] a seed to use for randomization

```
class datarobot.DatetimePartitioningSpecification (datetime_partition_column, autopilot_data_selection_method=None, validation_duration=None, holdout_start_date=None, holdout_duration=None, disable_holdout=None, gap_duration=None, number_of_backtests=None, backtests=None, use_time_series=False, default_to_known_in_advance=False, default_to_do_not_derive=False, feature_derivation_window_start=None, feature_derivation_window_end=None, feature_settings=None, forecast_window_start=None, forecast_window_end=None, windows_basis_unit=None, treat_as_exponential=None, differencing_method=None, periodicities=None, multi_series_id_columns=None, use_cross_series_features=None, aggregation_type=None, cross_series_group_by_columns=None, calendar_id=None, holdout_end_date=None, unsupervised_mode=False, model_splits=None)
```

Uniquely defines a DatetimePartitioning for some project

Includes only the attributes of DatetimePartitioning that are directly controllable by users, not those determined by the DataRobot application based on the project dataset and the user-controlled settings.

This is the specification that should be passed to `Project.set_target` via the `partitioning_method` parameter. To see the full partitioning based on the project dataset, use `DatetimePartitioning.generate`.

All durations should be specified with a duration string such as those returned by the `partitioning_methods.construct_duration_string` helper method. Please see [datetime partitioned project documentation](#) for more information on duration strings.

Note that either (holdout_start_date, holdout_duration) or (holdout_start_date, holdout_end_date) can be used to specify holdout partitioning settings.

Attributes

datetime_partition_column [str] the name of the column whose values as dates are used to assign a row to a particular partition

autopilot_data_selection_method [str] one of `datarobot.enums.DATETIME_AUTOPILOT_DATA_SELECTION_METHOD`. Whether models created by the autopilot should use `rowCount` or `duration` as their `data_selection_method`.

validation_duration [str or None] the default validation_duration for the backtests

holdout_start_date [datetime.datetime or None] The start date of holdout scoring data. If `holdout_start_date` is specified, either `holdout_duration` or `holdout_end_date` must also be specified. If `disable_holdout` is set to `True`, `holdout_start_date`, `holdout_duration`, and `holdout_end_date` may not be specified.

holdout_duration [str or None] The duration of the holdout scoring data. If `holdout_duration` is specified, `holdout_start_date` must also be specified. If `disable_holdout` is set to `True`, `holdout_duration`, `holdout_start_date`, and `holdout_end_date` may not be specified.

holdout_end_date [datetime.datetime or None] The end date of holdout scoring data. If `holdout_end_date` is specified, `holdout_start_date` must also be specified. If `disable_holdout` is set to `True`, `holdout_end_date`, `holdout_start_date`, and `holdout_duration` may not be specified.

disable_holdout [bool or None] (New in version v2.8) Whether to suppress allocating a holdout fold. If set to `True`, `holdout_start_date`, `holdout_duration`, and `holdout_end_date` may not be specified.

gap_duration [str or None] The duration of the gap between training and holdout scoring data

number_of_backtests [int or None] the number of backtests to use

backtests [list of *BacktestSpecification*] the exact specification of backtests to use. The indexes of the specified backtests should range from 0 to `number_of_backtests - 1`. If any backtest is left unspecified, a default configuration will be chosen.

use_time_series [bool] (New in version v2.8) Whether to create a time series project (if `True`) or an OTV project which uses datetime partitioning (if `False`). The default behaviour is to create an OTV project.

default_to_known_in_advance [bool] (New in version v2.11) Optional, default `False`. Used for time series projects only. Sets whether all features default to being treated as known in advance. Known in advance features are expected to be known for dates in the future when making predictions, e.g., “Is this a holiday?”. Individual features can be set to a value different than the default using the `feature_settings` parameter.

default_to_do_not_derive [bool] (New in v2.17) Optional, default `False`. Used for time series projects only. Sets whether all features default to being treated as do-not-derive features, excluding them from feature derivation. Individual features can be set to a value different than the default by using the `feature_settings` parameter.

feature_derivation_window_start [int or None] (New in version v2.8) Only used for time series projects. Offset into the past to define how far back relative to the forecast point the feature derivation window should start. Expressed in terms of the `windows_basis_unit` and should be negative or zero.

feature_derivation_window_end [int or None] (New in version v2.8) Only used for time series projects. Offset into the past to define how far back relative to the forecast point the feature derivation window should end. Expressed in terms of the `windows_basis_unit` and should be a positive value.

feature_settings [list of *FeatureSettings*] (New in version v2.9) Optional, a list specifying per feature settings, can be left unspecified.

forecast_window_start [int or None] (New in version v2.8) Only used for time series projects. Offset into the future to define how far forward relative to the forecast point the forecast window should start. Expressed in terms of the `windows_basis_unit`.

forecast_window_end [int or None] (New in version v2.8) Only used for time series projects. Offset into the future to define how far forward relative to the forecast point the forecast window should end. Expressed in terms of the `windows_basis_unit`.

windows_basis_unit [string, optional] (New in version v2.14) Only used for time series projects. Indicates which unit is a basis for feature derivation window and forecast window. Valid options are detected time unit (one of the `datarobot.enums.TIME_UNITS`) or `â€œIROWâ€œ`. If omitted, the default value is the detected time unit.

treat_as_exponential [string, optional] (New in version v2.9) defaults to `â€œIautoâ€œ`. Used to specify whether to treat data as exponential trend and apply transformations like log-transform. Use values from the `datarobot.enums.TREAT_AS_EXPONENTIAL` enum.

differencing_method [string, optional] (New in version v2.9) defaults to `â€œIautoâ€œ`. Used to specify which differencing method to apply of case if data is stationary. Use values from `datarobot.enums.DIFFERENCING_METHOD` enum.

periodicities [list of Periodicity, optional] (New in version v2.9) a list of `datarobot.Periodicity`. Periodicities units should be `â€œIROWâ€œ`, if the `windows_basis_unit` is `â€œIROWâ€œ`.

multiseries_id_columns [list of str or null] (New in version v2.11) a list of the names of multiseries id columns to define series within the training data. Currently only one multiseries id column is supported.

use_cross_series_features [bool] (New in version v2.14) Whether to use cross series features.

aggregation_type [str, optional] (New in version v2.14) The aggregation type to apply when creating cross series features. Optional, must be one of `â€œItotalâ€œ` or `â€œIaverageâ€œ`.

cross_series_group_by_columns [list of str, optional] (New in version v2.15) List of columns (currently of length 1). Optional setting that indicates how to further split series into related groups. For example, if every series is sales of an individual product, the series group-by could be the product category with values like `â€œImenâ€œs clothingâ€œ`, `â€œIsports equipmentâ€œ`, etc.. Can only be used in a multiseries project with `use_cross_series_features` set to `True`.

calendar_id [str, optional] (New in version v2.15) The id of the `CalendarFile` to use with this project.

unsupervised_mode: bool, optional (New in version v2.20) defaults to `False`, indicates whether partitioning should be constructed for the unsupervised project.

model_splits: int, optional (New in version v2.21) Sets the cap on the number of jobs per model used when building models to control number of jobs in the queue. Higher number of model splits will allow for less downsampling leading to the use of more post-processed data.

collect_payload()

Set up the dict that should be sent to the server when setting the target Returns `â€œIâ€œ partitioning_spec : dict`

prep_payload(project_id, max_wait=600)

Run any necessary validation and prep of the payload, including async operations

Mainly used for the datetime partitioning spec but implemented in general for consistency


```
class datarobot.BacktestSpecification (index, gap_duration=None, validation_start_date=None, validation_duration=None, validation_end_date=None, primary_training_start_date=None, primary_training_end_date=None)
```

Uniquely defines a Backtest used in a DatetimePartitioning

Includes only the attributes of a backtest directly controllable by users. The other attributes are assigned by the DataRobot application based on the project dataset and the user-controlled settings.

There are two ways to specify an individual backtest:

Option 1: Use index, gap_duration, validation_start_date, and validation_duration. All durations should be specified with a duration string such as those returned by the `partitioning_methods.construct_duration_string` helper method.

```
import datarobot as dr

partitioning_spec = dr.DatetimePartitioningSpecification(
    backtests=[
        # modify the first backtest using option 1
        dr.BacktestSpecification(
            index=0,
            gap_duration=dr.partitioning_methods.construct_duration_string(),
            validation_start_date=datetime(year=2010, month=1, day=1),
            validation_duration=dr.partitioning_methods.construct_duration_
↪string(years=1),
        )
    ],
    # other partitioning settings...
)
```

Option 2 (New in version v2.20): Use index, primary_training_start_date, primary_training_end_date, validation_start_date, and validation_end_date. In this case, note that setting primary_training_end_date and validation_start_date to the same timestamp will result with no gap being created.

```
import datarobot as dr

partitioning_spec = dr.DatetimePartitioningSpecification(
    backtests=[
        # modify the first backtest using option 2
        dr.BacktestSpecification(
            index=0,
            primary_training_start_date=datetime(year=2005, month=1, day=1),
            primary_training_end_date=datetime(year=2010, month=1, day=1),
            validation_start_date=datetime(year=2010, month=1, day=1),
            validation_end_date=datetime(year=2011, month=1, day=1),
        )
    ],
    # other partitioning settings...
)
```

All durations should be specified with a duration string such as those returned by the `partitioning_methods.construct_duration_string` helper method. Please see [datetime partitioned project documentation](#) for more information on duration strings.

Attributes

index [int] the index of the backtest to update

gap_duration [str] a duration string specifying the desired duration of the gap between training and validation scoring data for the backtest

validation_start_date [datetime.datetime] the desired start date of the validation scoring data for this backtest

validation_duration [str] a duration string specifying the desired duration of the validation scoring data for this backtest

validation_end_date [datetime.datetime] the desired end date of the validation scoring data for this backtest

primary_training_start_date [datetime.datetime] the desired start date of the training partition for this backtest

primary_training_end_date [datetime.datetime] the desired end date of the training partition for this backtest

class `datarobot.FeatureSettings` (*feature_name*, *known_in_advance=None*,
do_not_derive=None)

Per feature settings

Attributes

feature_name [string] name of the feature

known_in_advance [bool] (New in version v2.11) Optional, for time series projects only. Sets whether the feature is known in advance, i.e., values for future dates are known at prediction time. If not specified, the feature uses the value from the *default_to_known_in_advance* flag.

do_not_derive [bool] (New in v2.17) Optional, for time series projects only. Sets whether the feature is excluded from feature derivation. If not specified, the feature uses the value from the *default_to_do_not_derive* flag.

class `datarobot.Periodicity` (*time_steps*, *time_unit*)

Periodicity configuration

Parameters

time_steps [int] Time step value

time_unit [string] Time step unit, valid options are values from *datarobot.enums.TIME_UNITS*

Examples

```
from datarobot as dr
periodicities = [
    dr.Periodicity(time_steps=10, time_unit=dr.enums.TIME_UNITS.HOUR),
    dr.Periodicity(time_steps=600, time_unit=dr.enums.TIME_UNITS.MINUTE)]
spec = dr.DatetimePartitioningSpecification(
    # ...
    periodicities=periodicities
)
```

```

class datarobot.DatetimePartitioning (project_id=None,  datetime_partition_column=None,
                                     date_format=None,  autopilot_data_selection_method=None,
                                     validation_duration=None,  available_training_start_date=None,
                                     available_training_duration=None,  available_training_row_count=None,
                                     available_training_end_date=None,  primary_training_start_date=None,
                                     primary_training_duration=None,  primary_training_row_count=None,
                                     primary_training_end_date=None, gap_start_date=None,
                                     gap_duration=None,  gap_row_count=None,
                                     gap_end_date=None,  holdout_start_date=None,
                                     holdout_duration=None,  holdout_row_count=None,
                                     holdout_end_date=None,  number_of_backtests=None,
                                     backtests=None,  total_row_count=None,
                                     use_time_series=False,
                                     default_to_known_in_advance=False,
                                     default_to_do_not_derive=False,  feature_derivation_window_start=None,
                                     feature_derivation_window_end=None,  feature_settings=None,
                                     forecast_window_start=None,
                                     forecast_window_end=None,  windows_basis_unit=None,
                                     treat_as_exponential=None,
                                     differencing_method=None,  periodicities=None,
                                     multiseries_id_columns=None,
                                     number_of_known_in_advance_features=0,
                                     number_of_do_not_derive_features=0,
                                     use_cross_series_features=None,
                                     aggregation_type=None,
                                     cross_series_group_by_columns=None,  calendar_id=None,
                                     calendar_name=None,
                                     model_splits=None)

```

Full partitioning of a project for datetime partitioning.

To instantiate, use `DatetimePartitioning.get(project_id)`.

Includes both the attributes specified by the user, as well as those determined by the DataRobot application based on the project dataset. In order to use a partitioning to set the target, call `to_specification` and pass the resulting `DatetimePartitioningSpecification` to `Project.set_target` via the `partitioning_method` parameter.

The available training data corresponds to all the data available for training, while the primary training data corresponds to the data that can be used to train while ensuring that all backtests are available. If a model is trained with more data than is available in the primary training data, then all backtests may not have scores available.

All durations are specified with a duration string such as those returned by the `partitioning_methods.construct_duration_string` helper method. Please see [datetime partitioned project documentation](#) for more information on duration strings.

Attributes

project_id [str] the id of the project this partitioning applies to

datetime_partition_column [str] the name of the column whose values as dates are used to assign a row to a particular partition

date_format [str] the format (e.g. `âĀIJ%Y-%m-%d %H:%M:%SâĀĪ`) by which the partition column was interpreted (compatible with `strftime`)

autopilot_data_selection_method [str] one of `datarobot.enums.DATETIME_AUTOPILOT_DATA_SELECTION_METHOD`. Whether models created by the autopilot use `âĀĪrowCountâĀĪ` or `âĀĪdurationâĀĪ` as their `data_selection_method`.

validation_duration [str or None] the validation duration specified when initializing the partitioning - not directly significant if the backtests have been modified, but used as the default `validation_duration` for the backtests. Can be absent if this is a time series project with an irregular primary date/time feature.

available_training_start_date [datetime.datetime] The start date of the available training data for scoring the holdout

available_training_duration [str] The duration of the available training data for scoring the holdout

available_training_row_count [int or None] The number of rows in the available training data for scoring the holdout. Only available when retrieving the partitioning after setting the target.

available_training_end_date [datetime.datetime] The end date of the available training data for scoring the holdout

primary_training_start_date [datetime.datetime or None] The start date of primary training data for scoring the holdout. Unavailable when the holdout fold is disabled.

primary_training_duration [str] The duration of the primary training data for scoring the holdout

primary_training_row_count [int or None] The number of rows in the primary training data for scoring the holdout. Only available when retrieving the partitioning after setting the target.

primary_training_end_date [datetime.datetime or None] The end date of the primary training data for scoring the holdout. Unavailable when the holdout fold is disabled.

gap_start_date [datetime.datetime or None] The start date of the gap between training and holdout scoring data. Unavailable when the holdout fold is disabled.

gap_duration [str] The duration of the gap between training and holdout scoring data

gap_row_count [int or None] The number of rows in the gap between training and holdout scoring data. Only available when retrieving the partitioning after setting the target.

gap_end_date [datetime.datetime or None] The end date of the gap between training and holdout scoring data. Unavailable when the holdout fold is disabled.

holdout_start_date [datetime.datetime or None] The start date of holdout scoring data. Unavailable when the holdout fold is disabled.

holdout_duration [str] The duration of the holdout scoring data

holdout_row_count [int or None] The number of rows in the holdout scoring data. Only available when retrieving the partitioning after setting the target.

holdout_end_date [datetime.datetime or None] The end date of the holdout scoring data. Unavailable when the holdout fold is disabled.

number_of_backtests [int] the number of backtests used.

backtests [list of `Backtest`] the configured backtests.

- total_row_count** [int] the number of rows in the project dataset. Only available when retrieving the partitioning after setting the target.
- use_time_series** [bool] (New in version v2.8) Whether to create a time series project (if `True`) or an OTV project which uses datetime partitioning (if `False`). The default behaviour is to create an OTV project.
- default_to_known_in_advance** [bool] (New in version v2.11) Optional, default `False`. Used for time series projects only. Sets whether all features default to being treated as known in advance. Known in advance features are expected to be known for dates in the future when making predictions, e.g., “Is this a holiday?”. Individual features can be set to a value different from the default using the `feature_settings` parameter.
- default_to_do_not_derive** [bool] (New in v2.17) Optional, default `False`. Used for time series projects only. Sets whether all features default to being treated as do-not-derive features, excluding them from feature derivation. Individual features can be set to a value different from the default by using the `feature_settings` parameter.
- feature_derivation_window_start** [int or None] (New in version v2.8) Only used for time series projects. Offset into the past to define how far back relative to the forecast point the feature derivation window should start. Expressed in terms of the `windows_basis_unit`.
- feature_derivation_window_end** [int or None] (New in version v2.8) Only used for time series projects. Offset into the past to define how far back relative to the forecast point the feature derivation window should end. Expressed in terms of the `windows_basis_unit`.
- feature_settings** [list of `FeatureSettings`] (New in version v2.9) Optional, a list specifying per feature settings, can be left unspecified.
- forecast_window_start** [int or None] (New in version v2.8) Only used for time series projects. Offset into the future to define how far forward relative to the forecast point the forecast window should start. Expressed in terms of the `windows_basis_unit`.
- forecast_window_end** [int or None] (New in version v2.8) Only used for time series projects. Offset into the future to define how far forward relative to the forecast point the forecast window should end. Expressed in terms of the `windows_basis_unit`.
- windows_basis_unit** [string, optional] (New in version v2.14) Only used for time series projects. Indicates which unit is a basis for feature derivation window and forecast window. Valid options are detected time unit (one of the `datarobot.enums.TIME_UNITS`) or “ROW”. If omitted, the default value is detected time unit.
- treat_as_exponential** [string, optional] (New in version v2.9) defaults to “auto”. Used to specify whether to treat data as exponential trend and apply transformations like log-transform. Use values from the `datarobot.enums.TREAT_AS_EXPONENTIAL` enum.
- differencing_method** [string, optional] (New in version v2.9) defaults to “auto”. Used to specify which differencing method to apply of case if data is stationary. Use values from the `datarobot.enums.DIFFERENCING_METHOD` enum.
- periodicities** [list of `Periodicity`, optional] (New in version v2.9) a list of `datarobot.Periodicity`. Periodicities units should be “ROW”, if the `windows_basis_unit` is “ROW”.
- multiseries_id_columns** [list of str or null] (New in version v2.11) a list of the names of multiseries id columns to define series within the training data. Currently only one multiseries id column is supported.
- number_of_known_in_advance_features** [int] (New in version v2.14) Number of features that are marked as known in advance.

number_of_do_not_derive_features [int] (New in v2.17) Number of features that are excluded from derivation.

use_cross_series_features [bool] (New in version v2.14) Whether to use cross series features.

aggregation_type [str, optional] (New in version v2.14) The aggregation type to apply when creating cross series features. Optional, must be one of `total` or `average`.

cross_series_group_by_columns [list of str, optional] (New in version v2.15) List of columns (currently of length 1). Optional setting that indicates how to further split series into related groups. For example, if every series is sales of an individual product, the series group-by could be the product category with values like `men's clothing`, `sports equipment`, etc.. Can only be used in a multiseries project with `use_cross_series_features` set to `True`.

calendar_id [str, optional] (New in version v2.15) Only available for time series projects. The id of the `CalendarFile` to use with this project.

calendar_name [str, optional] (New in version v2.17) Only available for time series projects. The name of the `CalendarFile` used with this project.

model_splits: int, optional (New in version v2.21) Sets the cap on the number of jobs per model used when building models to control number of jobs in the queue. Higher number of model splits will allow for less downsampling leading to the use of more post-processed data.

classmethod generate (*project_id, spec, max_wait=600*)

Preview the full partitioning determined by a `DatetimePartitioningSpecification`

Based on the project dataset and the partitioning specification, inspect the full partitioning that would be used if the same specification were passed into `Project.set_target`.

Parameters

project_id [str] the id of the project

spec [`DatetimePartitioningSpec`] the desired partitioning

max_wait [int, optional] For some settings (e.g. generating a partitioning preview for a multiseries project for the first time), an asynchronous task must be run to analyze the dataset. `max_wait` governs the maximum time (in seconds) to wait before giving up. In all non-multiseries projects, this is unused.

Returns

DatetimePartitioning : the full generated partitioning

classmethod get (*project_id*)

Retrieve the `DatetimePartitioning` from a project

Only available if the project has already set the target as a datetime project.

Parameters

project_id [str] the id of the project to retrieve partitioning for

Returns

DatetimePartitioning [the full partitioning for the project]

classmethod generate_optimized (*project_id, spec, target, max_wait=600*)

Preview the full partitioning determined by a `DatetimePartitioningSpecification`

Based on the project dataset and the partitioning specification, inspect the full partitioning that would be used if the same specification were passed into `Project.set_target`.

Parameters

- project_id** [str] the id of the project
- spec** [DatetimePartitioningSpec] the desired partitioning
- target** [str] the name of the target column. For unsupervised projects target may be None.
- max_wait** [int, optional] Governs the maximum time (in seconds) to wait before giving up.

Returns

DatetimePartitioning : the full generated partitioning

classmethod get_optimized (*project_id, datetime_partitioning_id*)

Retrieve an Optimized DatetimePartitioning from a project for the specified datetime_partitioning_id. A datetime_partitioning_id is created by using the *generate_optimized* function.

Parameters

- project_id** [str] the id of the project to retrieve partitioning for
- datetime_partitioning_id** [ObjectId] the ObjectId associated with the project to retrieve from mongo

Returns

DatetimePartitioning [the full partitioning for the project]

classmethod feature_log_list (*project_id, offset=None, limit=None*)

Retrieve the feature derivation log content and log length for a time series project.

The Time Series Feature Log provides details about the feature generation process for a time series project. It includes information about which features are generated and their priority, as well as the detected properties of the time series data such as whether the series is stationary, and periodicities detected.

This route is only supported for time series projects that have finished partitioning.

The feature derivation log will include information about:

- Detected stationarity of the series:
e.g. "Series detected as non-stationary"
- Detected presence of multiplicative trend in the series:
e.g. "Multiplicative trend detected"
- Detected presence of multiplicative trend in the series:
e.g. "Detected periodicities: 7 day"
- Maximum number of feature to be generated:
e.g. "Maximum number of feature to be generated is 1440"
- Window sizes used in rolling statistics / lag extractors
e.g. "The window sizes chosen to be: 2 months
(because the time step is 1 month and Feature Derivation Window is 2 months)"
- Features that are specified as known-in-advance
e.g. "Variables treated as apriori: holiday"
- Details about why certain variables are transformed in the input data
e.g. "Generating variable 'log' from 'Iy' because multiplicative trend is detected"
- Details about features generated as timeseries features, and their priority

e.g. `Generating feature date (actual) from date (priority: 1)`

Parameters

project_id [str] project id to retrieve a feature derivation log for.

offset [int] optional, defaults is 0, this many results will be skipped.

limit [int] optional, defaults to 100, at most this many results are returned. To specify

no limit, use 0. The default may change without notice.

classmethod feature_log_retrieve (*project_id*)

Retrieve the feature derivation log content and log length for a time series project.

The Time Series Feature Log provides details about the feature generation process for a time series project. It includes information about which features are generated and their priority, as well as the detected properties of the time series data such as whether the series is stationary, and periodicities detected.

This route is only supported for time series projects that have finished partitioning.

The feature derivation log will include information about:

- Detected stationarity of the series:
e.g. `Series detected as non-stationary`
- Detected presence of multiplicative trend in the series:
e.g. `Multiplicative trend detected`
- Detected presence of multiplicative trend in the series:
e.g. `Detected periodicities: 7 day`
- Maximum number of feature to be generated:
e.g. `Maximum number of feature to be generated is 1440`
- Window sizes used in rolling statistics / lag extractors
e.g. `The window sizes chosen to be: 2 months`
(because the time step is 1 month and Feature Derivation Window is 2 months)
- Features that are specified as known-in-advance
e.g. `Variables treated as apriori: holiday`
- Details about why certain variables are transformed in the input data
e.g. `Generating variable y (log) from y` because multiplicative trend is detected
- Details about features generated as timeseries features, and their priority
e.g. `Generating feature date (actual) from date (priority: 1)`

Parameters

project_id [str] project id to retrieve a feature derivation log for.

to_specification (*use_holdout_start_end_format=False, use_backtest_start_end_format=False*)

Render the DatetimePartitioning as a *DatetimePartitioningSpecification*

The resulting specification can be used when setting the target, and contains only the attributes directly controllable by users.

Parameters

use_holdout_start_end_format [bool, optional] Defaults to False. If True, will use `holdout_end_date` when configuring the holdout partition. If False, will use `holdout_duration` instead.

use_backtest_start_end_format [bool, optional] Defaults to False. If False, will use a duration-based approach for specifying backtests (`gap_duration`, `validation_start_date`, and `validation_duration`). If True, will use a start/end date approach for specifying backtests (`primary_training_start_date`, `primary_training_end_date`, `validation_start_date`, and `validation_end_date`).

Returns

DatetimePartitioningSpecification the specification for this partitioning

to_dataframe()

Render the partitioning settings as a dataframe for convenience of display

Excludes `project_id`, `datetime_partition_column`, `date_format`, `autopilot_data_selection_method`, `validation_duration`, and `number_of_backtests`, as well as the row count information, if present.

Also excludes the time series specific parameters for `use_time_series`, `default_to_known_in_advance`, `default_to_do_not_derive`, and defining the feature derivation and forecast windows.

```
class datarobot.helpers.partitioning_methods.Backtest (index=None,          avail-
                                                    able_training_start_date=None,
                                                    avail-
                                                    able_training_duration=None,
                                                    avail-
                                                    able_training_row_count=None,
                                                    avail-
                                                    able_training_end_date=None,
                                                    pri-
                                                    mary_training_start_date=None,
                                                    pri-
                                                    mary_training_duration=None,
                                                    pri-
                                                    mary_training_row_count=None,
                                                    pri-
                                                    mary_training_end_date=None,
                                                    gap_start_date=None,
                                                    gap_duration=None,
                                                    gap_row_count=None,
                                                    gap_end_date=None,      val-
                                                    idation_start_date=None,
                                                    validation_duration=None,
                                                    validation_row_count=None,
                                                    validation_end_date=None,
                                                    total_row_count=None)
```

A backtest used to evaluate models trained in a datetime partitioned project

When setting up a datetime partitioning project, backtests are specified by a [*BacktestSpecification*](#).

The available training data corresponds to all the data available for training, while the primary training data corresponds to the data that can be used to train while ensuring that all backtests are available. If a model is trained with more data than is available in the primary training data, then all backtests may not have scores available.

All durations are specified with a duration string such as those returned by the [*partitioning_methods*](#).

`construct_duration_string` helper method. Please see *datetime partitioned project documentation* for more information on duration strings.

Attributes

index [int] the index of the backtest

available_training_start_date [datetime.datetime] the start date of the available training data for this backtest

available_training_duration [str] the duration of available training data for this backtest

available_training_row_count [int or None] the number of rows of available training data for this backtest. Only available when retrieving from a project where the target is set.

available_training_end_date [datetime.datetime] the end date of the available training data for this backtest

primary_training_start_date [datetime.datetime] the start date of the primary training data for this backtest

primary_training_duration [str] the duration of the primary training data for this backtest

primary_training_row_count [int or None] the number of rows of primary training data for this backtest. Only available when retrieving from a project where the target is set.

primary_training_end_date [datetime.datetime] the end date of the primary training data for this backtest

gap_start_date [datetime.datetime] the start date of the gap between training and validation scoring data for this backtest

gap_duration [str] the duration of the gap between training and validation scoring data for this backtest

gap_row_count [int or None] the number of rows in the gap between training and validation scoring data for this backtest. Only available when retrieving from a project where the target is set.

gap_end_date [datetime.datetime] the end date of the gap between training and validation scoring data for this backtest

validation_start_date [datetime.datetime] the start date of the validation scoring data for this backtest

validation_duration [str] the duration of the validation scoring data for this backtest

validation_row_count [int or None] the number of rows of validation scoring data for this backtest. Only available when retrieving from a project where the target is set.

validation_end_date [datetime.datetime] the end date of the validation scoring data for this backtest

total_row_count [int or None] the number of rows in this backtest. Only available when retrieving from a project where the target is set.

to_specification (*use_start_end_format=False*)

Render this backtest as a *BacktestSpecification*.

The resulting specification includes only the attributes users can directly control, not those indirectly determined by the project dataset.

Parameters

use_start_end_format [bool] Default False. If False, will use a duration-based approach for specifying backtests (`gap_duration`, `validation_start_date`, and `validation_duration`). If True, will use a start/end date approach for specifying backtests (`primary_training_start_date`, `primary_training_end_date`, `validation_start_date`, `validation_end_date`).

Returns

BacktestSpecification the specification for this backtest

to_dataframe()

Render this backtest as a dataframe for convenience of display

Returns

backtest_partitioning [pandas.DataFrame] the backtest attributes, formatted into a dataframe

`datarobot.helpers.partitioning_methods.construct_duration_string` (*years=0, months=0, days=0, hours=0, minutes=0, seconds=0*)

Construct a valid string representing a duration in accordance with ISO8601

A duration of six months, 3 days, and 12 hours could be represented as P6M3DT12H.

Parameters

years [int] the number of years in the duration

months [int] the number of months in the duration

days [int] the number of days in the duration

hours [int] the number of hours in the duration

minutes [int] the number of minutes in the duration

seconds [int] the number of seconds in the duration

Returns

duration_string: str The duration string, specified compatibly with ISO8601

32.24 PayoffMatrix

class `datarobot.models.PayoffMatrix` (*project_id, id, name=None, true_positive_value=None, true_negative_value=None, false_positive_value=None, false_negative_value=None*)

Represents a Payoff Matrix, a costs/benefit scenario used for creating a profit curve.

Examples

```
import datarobot as dr

# create a payoff matrix
```

(continues on next page)

(continued from previous page)

```

payoff_matrix = dr.PayoffMatrix.create(project_id, name, true_positive_value=100,
                                       true_negative_value=10, false_positive_value=0, false_negative_
                                       ↪value=-10)

# list available payoff matrices
payoff_matrices = dr.PayoffMatrix.list(project_id)
payoff_matrix = payoff_matrices[0]

```

Attributes**project_id** [str] id of the project with which the payoff matrix is associated.**id** [str] id of the payoff matrix.**name** [str] User-supplied label for the payoff matrix.**true_positive_value** [float] Cost or benefit of a true positive classification**true_negative_value: float** Cost or benefit of a true negative classification**false_positive_value: float** Cost or benefit of a false positive classification**false_negative_value: float** Cost or benefit of a false negative classification

classmethod create (*project_id*, *name*, *true_positive_value=1*, *true_negative_value=1*,
false_positive_value=-1, *false_negative_value=-1*)

Create a payoff matrix associated with a specific project.

Parameters**project_id** [str] id of the project with which the payoff matrix will be associated**Returns****payoff_matrix** [*PayoffMatrix*] The newly created payoff matrix

classmethod list (*project_id*)

Fetch all the payoff matrices for a project.

Parameters**project_id** [str] id of the project**Returns**

•••••

List of PayoffMatrix A list of *PayoffMatrix* objects**Raises**

•••••

datarobot.errors.ClientError if the server responded with 4xx status**datarobot.errors.ServerError** if the server responded with 5xx status

classmethod get (*project_id*, *id*)

Retrieve a specified payoff matrix.

Parameters**project_id** [str] id of the project the model belongs to**id** [str] id of the payoff matrix

Returns

PayoffMatrix object representing specified
payoff matrix

Raises

datarobot.errors.ClientError if the server responded with 4xx status

datarobot.errors.ServerError if the server responded with 5xx status

classmethod update (*project_id*, *id*, *name*, *true_positive_value*, *true_negative_value*,
false_positive_value, *false_negative_value*)

Update (replace) a payoff matrix. Note that all data fields are required.

Parameters

project_id [str] id of the project to which the payoff matrix belongs

id [str] id of the payoff matrix

name [str] User-supplied label for the payoff matrix

true_positive_value [float] True positive payoff value to use for the profit curve

true_negative_value [float] True negative payoff value to use for the profit curve

false_positive_value [float] False positive payoff value to use for the profit curve

false_negative_value [float] False negative payoff value to use for the profit curve

Returns

payoff_matrix PayoffMatrix with updated values

Raises

datarobot.errors.ClientError if the server responded with 4xx status

datarobot.errors.ServerError if the server responded with 5xx status

classmethod delete (*project_id*, *id*)

Delete a specified payoff matrix.

Parameters

project_id [str] id of the project the model belongs to

id [str] id of the payoff matrix

Returns

response [requests.Response] Empty response (204)

Raises

datarobot.errors.ClientError if the server responded with 4xx status

datarobot.errors.ServerError if the server responded with 5xx status

classmethod from_server_data (*data*, *keep_attrs=None*)

Instantiate an object of this class using the data directly from the server, meaning that the keys may have the wrong camel casing

Parameters

data [dict] The directly translated dict of JSON from the server. No casing fixes have taken place

keep_attrs [list] List of the dotted namespace notations for attributes to keep within the object structure even if their values are None

32.25 PredictJob

`datarobot.models.predict_job.wait_for_async_predictions` (*project_id*, *predict_job_id*,
max_wait=600)

Given a Project id and PredictJob id poll for status of process responsible for predictions generation until it's finished

Parameters

project_id [str] The identifier of the project

predict_job_id [str] The identifier of the PredictJob

max_wait [int, optional] Time in seconds after which predictions creation is considered unsuccessful

Returns

predictions [pandas.DataFrame] Generated predictions.

Raises

AsyncPredictionsGenerationError Raised if status of fetched PredictJob object is `error`

AsyncTimeoutError Predictions weren't generated in time, specified by `max_wait` parameter

class `datarobot.models.PredictJob` (*data*, *completed_resource_url=None*)

Tracks asynchronous work being done within a project

Attributes

id [int] the id of the job

project_id [str] the id of the project the job belongs to

status [str] the status of the job - will be one of `datarobot.enums.QUEUE_STATUS`

job_type [str] what kind of work the job is doing - will be `'predict'` for predict jobs

is_blocked [bool] if true, the job is blocked (cannot be executed) until its dependencies are resolved

message [str] a message about the state of the job, typically explaining why an error occurred

classmethod `from_job` (*job*)

Transforms a generic Job into a PredictJob

Parameters

job: Job A generic job representing a PredictJob

Returns

predict_job: PredictJob A fully populated PredictJob with all the details of the job

Raises

ValueError: If the generic Job was not a predict job, e.g. `job_type != JOB_TYPE.PREDICT`

classmethod `create` (*model*, *sourcedata*)

Note: Deprecated in v2.3 in favor of `Project.upload_dataset` and `Model.request_predictions`. That workflow allows you to reuse the same dataset for predictions from multiple models within one project.

Starts predictions generation for provided data using previously created model.

Parameters

model [Model] Model to use for predictions generation

sourcedata [str, file or pandas.DataFrame] Data to be used for predictions. If this parameter is a str, it can be either a path to a local file or raw file content. If using a file on disk, the filename must consist of ASCII characters only. The file must be a CSV, and cannot be compressed

Returns

predict_job_id [str] id of created job, can be used as parameter to `PredictJob.get` or `PredictJob.get_predictions` methods or `wait_for_async_predictions` function

Raises

InputNotUnderstoodError If the parameter for `sourcedata` didn't resolve into known data types

Examples

```
model = Model.get('p-id', 'l-id')
predict_job = PredictJob.create(model, './data_to_predict.csv')
```

classmethod `get` (*project_id*, *predict_job_id*)

Fetches one `PredictJob`. If the job finished, raises `PendingJobFinished` exception.

Parameters

project_id [str] The identifier of the project the model on which prediction was started belongs to

predict_job_id [str] The identifier of the `predict_job`

Returns

predict_job [`PredictJob`] The pending `PredictJob`

Raises

PendingJobFinished If the job being queried already finished, and the server is re-routing to the finished predictions.

AsyncFailureError Querying this resource gave a status code other than 200 or 303

classmethod `get_predictions` (*project_id*, *predict_job_id*, *class_prefix*='class_')

Fetches finished predictions from the job used to generate them.

Note: The prediction API for classifications now returns an additional `prediction_values` dictionary that is converted into a series of `class_prefixed` columns in the final dataframe. For example, `<label> = 1.0` is

converted to `class_1.0`. If you are on an older version of the client (prior to v2.8), you must update to v2.8 to correctly pivot this data.

Parameters

project_id [str] The identifier of the project to which belongs the model used for predictions generation

predict_job_id [str] The identifier of the predict_job

class_prefix [str] The prefix to append to labels in the final dataframe (e.g., apple -> class_apple)

Returns

predictions [pandas.DataFrame] Generated predictions

Raises

JobNotFinished If the job has not finished yet

AsyncFailureError Querying the predict_job in question gave a status code other than 200 or 303

cancel()

Cancel this job. If this job has not finished running, it will be removed and canceled.

get_result (*params=None*)

Parameters

params [dict or None] Query parameters to be added to request to get results.

For featureEffects and featureFit, source param is required to define source, otherwise the default is ‘training’

Returns

result [object]

Return type depends on the job type:

- for model jobs, a Model is returned
- for predict jobs, a pandas.DataFrame (with predictions) is returned
- for featureImpact jobs, a list of dicts by default (see `with_metadata` parameter of the `FeatureImpactJob` class and its `get()` method).
- for primeRulesets jobs, a list of Rulesets
- for primeModel jobs, a PrimeModel
- for primeDownloadValidation jobs, a PrimeFile
- for reasonCodesInitialization jobs, a ReasonCodesInitialization
- for reasonCodes jobs, a ReasonCodes
- for predictionExplanationInitialization jobs, a PredictionExplanationsInitialization
- for predictionExplanations jobs, a PredictionExplanations
- for featureEffects, a FeatureEffects
- for featureFit, a FeatureFit

Raises

JobNotFinished If the job is not finished, the result is not available.

AsyncProcessUnsuccessfulError If the job errored or was aborted

get_result_when_complete (*max_wait=600, params=None*)

Parameters

max_wait [int, optional] How long to wait for the job to finish.

params [dict, optional] Query parameters to be added to request.

Returns

result: object Return type is the same as would be returned by *Job.get_result*.

Raises

AsyncTimeoutError If the job does not finish in time

AsyncProcessUnsuccessfulError If the job errored or was aborted

refresh ()

Update this object with the latest job data from the server.

wait_for_completion (*max_wait=600*)

Waits for job to complete.

Parameters

max_wait [int, optional] How long to wait for the job to finish.

32.26 Prediction Dataset

```
class datarobot.models.PredictionDataset (project_id, id, name, created, num_rows,
                                         num_columns, forecast_point=None,
                                         predictions_start_date=None, predictions_end_date=None,
                                         relax_known_in_advance_features_check=None,
                                         data_quality_warnings=None, forecast_point_range=None,
                                         data_start_date=None, data_end_date=None, max_forecast_date=None,
                                         actual_value_column=None, detected_actual_value_columns=None,
                                         contains_target_values=None)
```

A dataset uploaded to make predictions

Typically created via *project.upload_dataset*

Attributes

id [str] the id of the dataset

project_id [str] the id of the project the dataset belongs to

created [str] the time the dataset was created

name [str] the name of the dataset

num_rows [int] the number of rows in the dataset

num_columns [int] the number of columns in the dataset

forecast_point [datetime.datetime or None] For time series projects only. This is the default point relative to which predictions will be generated, based on the forecast window of the project. See the time series [predictions documentation](#) for more information.

predictions_start_date [datetime.datetime or None, optional] For time series projects only. The start date for bulk predictions. Note that this parameter is for generating historical predictions using the training data. This parameter should be provided in conjunction with `predictions_end_date`. Can't be provided with the `forecast_point` parameter.

predictions_end_date [datetime.datetime or None, optional] For time series projects only. The end date for bulk predictions, exclusive. Note that this parameter is for generating historical predictions using the training data. This parameter should be provided in conjunction with `predictions_start_date`. Can't be provided with the `forecast_point` parameter.

relax_known_in_advance_features_check [bool, optional] (New in version v2.15) For time series projects only. If True, missing values in the known in advance features are allowed in the forecast window at the prediction time. If omitted or False, missing values are not allowed.

data_quality_warnings [dict, optional] (New in version v2.15) A dictionary that contains available warnings about potential problems in this prediction dataset. Available warnings include:

has_kia_missing_values_in_forecast_window [bool] Applicable for time series projects. If True, known in advance features have missing values in forecast window which may decrease prediction accuracy.

insufficient_rows_for_evaluating_models [bool] Applicable for datasets which are used as external test sets. If True, there is not enough rows in dataset to calculate insights.

single_class_actual_value_column [bool] Applicable for datasets which are used as external test sets. If True, actual value column has only one class and such insights as ROC curve can not be calculated. Only applies for binary classification projects or unsupervised projects.

forecast_point_range [list[datetime.datetime] or None, optional] (New in version v2.20) For time series projects only. Specifies the range of dates available for use as a forecast point.

data_start_date [datetime.datetime or None, optional] (New in version v2.20) For time series projects only. The minimum primary date of this prediction dataset.

data_end_date [datetime.datetime or None, optional] (New in version v2.20) For time series projects only. The maximum primary date of this prediction dataset.

max_forecast_date [datetime.datetime or None, optional] (New in version v2.20) For time series projects only. The maximum forecast date of this prediction dataset.

actual_value_column [string, optional] (New in version v2.21) Optional, only available for unsupervised projects, in case dataset was uploaded with actual value column specified. Name of the column which will be used to calculate the classification metrics and insights.

detected_actual_value_columns [list of dict, optional] (New in version v2.21) For unsupervised projects only, list of detected actual value columns information containing missing count and name for each column.

contains_target_values [bool, optional] (New in version v2.21) Only for supervised projects. If True, dataset contains target values and can be used to calculate the classification metrics and insights.

classmethod `get` (*project_id*, *dataset_id*)

Retrieve information about a dataset uploaded for predictions

Parameters

project_id: the id of the project to query

dataset_id: the id of the dataset to retrieve

Returns

dataset: PredictionDataset A dataset uploaded to make predictions

delete ()

Delete a dataset uploaded for predictions

Will also delete predictions made using this dataset and cancel any predict jobs using this dataset.

32.27 Prediction Explanations

class `datarobot.PredictionExplanationsInitialization` (*project_id*, *model_id*, *prediction_explanations_sample=None*)

Represents a prediction explanations initialization of a model.

Attributes

project_id [str] id of the project the model belongs to

model_id [str] id of the model the prediction explanations initialization is for

prediction_explanations_sample [list of dict] a small sample of prediction explanations that could be generated for the model

classmethod `get` (*project_id*, *model_id*)

Retrieve the prediction explanations initialization for a model.

Prediction explanations initializations are a prerequisite for computing prediction explanations, and include a sample what the computed prediction explanations for a prediction dataset would look like.

Parameters

project_id [str] id of the project the model belongs to

model_id [str] id of the model the prediction explanations initialization is for

Returns

prediction_explanations_initialization [PredictionExplanationsInitialization] The queried instance.

Raises

ClientError (404) If the project or model does not exist or the initialization has not been computed.

classmethod `create` (*project_id*, *model_id*)

Create a prediction explanations initialization for the specified model.

Parameters

project_id [str] id of the project the model belongs to

model_id [str] id of the model for which initialization is requested

Returns

job [Job] an instance of created async job

delete()

Delete this prediction explanations initialization.

```
class datarobot.PredictionExplanations(id, project_id, model_id, dataset_id,  
                                         max_explanations, num_columns, finish_time,  
                                         prediction_explanations_location, thresh-  
                                         old_low=None, threshold_high=None)
```

Represents prediction explanations metadata and provides access to computation results.

Examples

```
prediction_explanations = dr.PredictionExplanations.get(project_id, explanations_  
↪id)  
for row in prediction_explanations.get_rows():  
    print(row) # row is an instance of PredictionExplanationsRow
```

Attributes

id [str] id of the record and prediction explanations computation result

project_id [str] id of the project the model belongs to

model_id [str] id of the model the prediction explanations are for

dataset_id [str] id of the prediction dataset prediction explanations were computed for

max_explanations [int] maximum number of prediction explanations to supply per row of the dataset

threshold_low [float] the lower threshold, below which a prediction must score in order for prediction explanations to be computed for a row in the dataset

threshold_high [float] the high threshold, above which a prediction must score in order for prediction explanations to be computed for a row in the dataset

num_columns [int] the number of columns prediction explanations were computed for

finish_time [float] timestamp referencing when computation for these prediction explanations finished

prediction_explanations_location [str] where to retrieve the prediction explanations

classmethod get (*project_id*, *prediction_explanations_id*)

Retrieve a specific prediction explanations.

Parameters

project_id [str] id of the project the explanations belong to

prediction_explanations_id [str] id of the prediction explanations

Returns

prediction_explanations [PredictionExplanations] The queried instance.

classmethod create (*project_id*, *model_id*, *dataset_id*, *max_explanations=None*, *thresh-*
old_low=None, threshold_high=None)

Create prediction explanations for the specified dataset.

In order to create PredictionExplanations for a particular model and dataset, you must first:

- Compute feature impact for the model via `datarobot.Model.get_feature_impact()`
- Compute a `PredictionExplanationsInitialization` for the model via `datarobot.PredictionExplanationsInitialization.create(project_id, model_id)`
- Compute predictions for the model and dataset via `datarobot.Model.request_predictions(dataset_id)`

`threshold_high` and `threshold_low` are optional filters applied to speed up computation. When at least one is specified, only the selected outlier rows will have prediction explanations computed. Rows are considered to be outliers if their predicted value (in case of regression projects) or probability of being the positive class (in case of classification projects) is less than `threshold_low` or greater than `threshold_high`. If neither is specified, prediction explanations will be computed for all rows.

Parameters

project_id [str] id of the project the model belongs to

model_id [str] id of the model for which prediction explanations are requested

dataset_id [str] id of the prediction dataset for which prediction explanations are requested

threshold_low [float, optional] the lower threshold, below which a prediction must score in order for prediction explanations to be computed for a row in the dataset. If neither `threshold_high` nor `threshold_low` is specified, prediction explanations will be computed for all rows.

threshold_high [float, optional] the high threshold, above which a prediction must score in order for prediction explanations to be computed. If neither `threshold_high` nor `threshold_low` is specified, prediction explanations will be computed for all rows.

max_explanations [int, optional] the maximum number of prediction explanations to supply per row of the dataset, default: 3.

Returns

job: **Job** an instance of created async job

classmethod list (*project_id, model_id=None, limit=None, offset=None*)

List of prediction explanations for a specified project.

Parameters

project_id [str] id of the project to list prediction explanations for

model_id [str, optional] if specified, only prediction explanations computed for this model will be returned

limit [int or None] at most this many results are returned, default: no limit

offset [int or None] this many results will be skipped, default: 0

Returns

prediction_explanations [list[PredictionExplanations]]

get_rows (*batch_size=None, exclude_adjusted_predictions=True*)

Retrieve prediction explanations rows.

Parameters

batch_size [int or None, optional] maximum number of prediction explanations rows to retrieve per request

exclude_adjusted_predictions [bool] Optional, defaults to True. Set to False to include adjusted predictions, which will differ from the predictions on some projects, e.g. those with an exposure column specified.

Yields

prediction_explanations_row [PredictionExplanationsRow] Represents prediction explanations computed for a prediction row.

get_all_as_dataframe (*exclude_adjusted_predictions=True*)

Retrieve all prediction explanations rows and return them as a pandas.DataFrame.

Returned dataframe has the following structure:

- **row_id** : row id from prediction dataset
- **prediction** : the output of the model for this row
- **adjusted_prediction** : adjusted prediction values (only appears for projects that utilize prediction adjustments, e.g. projects with an exposure column)
- **class_0_label** : a class level from the target (only appears for classification projects)
- **class_0_probability** : the probability that the target is this class (only appears for classification projects)
- **class_1_label** : a class level from the target (only appears for classification projects)
- **class_1_probability** : the probability that the target is this class (only appears for classification projects)
- **explanation_0_feature** : the name of the feature contributing to the prediction for this explanation
- **explanation_0_feature_value** : the value the feature took on
- **explanation_0_label** : the output being driven by this explanation. For regression projects, this is the name of the target feature. For classification projects, this is the class label whose probability increasing would correspond to a positive strength.
- **explanation_0_qualitative_strength** : a human-readable description of how strongly the feature affected the prediction (e.g. `â€”+++â€”`, `â€”-â€”`, `â€”+â€”`) for this explanation
- **explanation_0_strength** : the amount this featureâ€™s value affected the prediction
- **â€”**
- **explanation_N_feature** : the name of the feature contributing to the prediction for this explanation
- **explanation_N_feature_value** : the value the feature took on
- **explanation_N_label** : the output being driven by this explanation. For regression projects, this is the name of the target feature. For classification projects, this is the class label whose probability increasing would correspond to a positive strength.
- **explanation_N_qualitative_strength** : a human-readable description of how strongly the feature affected the prediction (e.g. `â€”+++â€”`, `â€”-â€”`, `â€”+â€”`) for this explanation
- **explanation_N_strength** : the amount this featureâ€™s value affected the prediction

For classification projects, the server does not guarantee any ordering on the prediction values, however within this function we sort the values so that *class_X* corresponds to the same class from row to row.

Parameters

exclude_adjusted_predictions [bool] Optional, defaults to True. Set this to False to include adjusted prediction values in the returned dataframe.

Returns**dataframe:** `pandas.DataFrame`**download_to_csv** (*filename*, *encoding*='utf-8', *exclude_adjusted_predictions*=True)

Save prediction explanations rows into CSV file.

Parameters**filename** [str or file object] path or file object to save prediction explanations rows**encoding** [string, optional] A string representing the encoding to use in the output file, defaults to 'utf-8'**exclude_adjusted_predictions** [bool] Optional, defaults to True. Set to False to include adjusted predictions, which will differ from the predictions on some projects, e.g. those with an exposure column specified.**get_prediction_explanations_page** (*limit*=None, *offset*=None, *exclude_adjusted_predictions*=True)

Get prediction explanations.

If you don't want use a generator interface, you can access paginated prediction explanations directly.

Parameters**limit** [int or None] the number of records to return, the server will use a (possibly finite) default if not specified**offset** [int or None] the number of records to skip, default 0**exclude_adjusted_predictions** [bool] Optional, defaults to True. Set to False to include adjusted predictions, which will differ from the predictions on some projects, e.g. those with an exposure column specified.**Returns****prediction_explanations** [PredictionExplanationsPage]**delete** ()

Delete these prediction explanations.

```
class datarobot.models.prediction_explanations.PredictionExplanationsRow (row_id,
                                                                           pre-
                                                                           dic-
                                                                           tion,
                                                                           pre-
                                                                           dic-
                                                                           tion_values,
                                                                           pre-
                                                                           dic-
                                                                           tion_explanations=None,
                                                                           ad-
                                                                           justed_prediction=None,
                                                                           ad-
                                                                           justed_prediction_values=
```

Represents prediction explanations computed for a prediction row.

Notes

PredictionValue contains:

- **label** : describes what this model output corresponds to. For regression projects, it is the name of the target feature. For classification projects, it is a level from the target feature.
- **value** : the output of the prediction. For regression projects, it is the predicted value of the target. For classification projects, it is the predicted probability the row belongs to the class identified by the label.

`PredictionExplanation` contains:

- **label** : described what output was driven by this explanation. For regression projects, it is the name of the target feature. For classification projects, it is the class whose probability increasing would correspond to a positive strength of this prediction explanation.
- **feature** : the name of the feature contributing to the prediction
- **feature_value** : the value the feature took on for this row
- **strength** : the amount this feature's value affected the prediction
- **qualitative_strength** : a human-readable description of how strongly the feature affected the prediction (e.g. "+++", "−", "+")

Attributes

row_id [int] which row this `PredictionExplanationsRow` describes

prediction [float] the output of the model for this row

adjusted_prediction [float or None] adjusted prediction value for projects that provide this information, None otherwise

prediction_values [list] an array of dictionaries with a schema described as `PredictionValue`

adjusted_prediction_values [list] same as `prediction_values` but for adjusted predictions

prediction_explanations [list] an array of dictionaries with a schema described as `PredictionExplanation`

```
class datarobot.models.prediction_explanations.PredictionExplanationsPage(id,
                                                                              count=None,
                                                                              pre-
                                                                              vi-
                                                                              ous=None,
                                                                              next=None,
                                                                              data=None,
                                                                              pre-
                                                                              dic-
                                                                              tion_explanations_recon-
                                                                              ad-
                                                                              just-
                                                                              ment_method=None)
```

Represents a batch of prediction explanations received by one request.

Attributes

id [str] id of the prediction explanations computation result

data [list[dict]] list of raw prediction explanations; each row corresponds to a row of the prediction dataset

count [int] total number of rows computed

previous_page [str] where to retrieve previous page of prediction explanations, None if current page is the first

next_page [str] where to retrieve next page of prediction explanations, None if current page is the last

prediction_explanations_record_location [str] where to retrieve the prediction explanations metadata

adjustment_method [str] Adjustment method that was applied to predictions, or `None` if no adjustments were done.

classmethod **get** (*project_id*, *prediction_explanations_id*, *limit=None*, *offset=0*, *exclude_adjusted_predictions=True*)

Retrieve prediction explanations.

Parameters

project_id [str] id of the project the model belongs to

prediction_explanations_id [str] id of the prediction explanations

limit [int or None] the number of records to return; the server will use a (possibly finite) default if not specified

offset [int or None] the number of records to skip, default 0

exclude_adjusted_predictions [bool] Optional, defaults to True. Set to False to include adjusted predictions, which will differ from the predictions on some projects, e.g. those with an exposure column specified.

Returns

prediction_explanations [PredictionExplanationsPage] The queried instance.

class `datarobot.models.ShapMatrix` (*project_id*, *id*, *model_id=None*, *dataset_id=None*)

Represents SHAP based prediction explanations and provides access to score values.

Examples

```
import datarobot as dr

# request SHAP matrix calculation
shap_matrix_job = dr.ShapMatrix.create(project_id, model_id, dataset_id)
shap_matrix = shap_matrix_job.get_result_when_complete()

# list available SHAP matrices
shap_matrices = dr.ShapMatrix.list(project_id)
shap_matrix = shap_matrices[0]

# get SHAP matrix as dataframe
shap_matrix_values = shap_matrix.get_as_dataframe()
```

Attributes

project_id [str] id of the project the model belongs to

shap_matrix_id [str] id of the generated SHAP matrix

model_id [str] id of the model used to

dataset_id [str] id of the prediction dataset SHAP values were computed for

classmethod `create` (*project_id*, *model_id*, *dataset_id*)

Calculate SHAP based prediction explanations against previously uploaded dataset.

Parameters

project_id [str] id of the project the model belongs to

model_id [str] id of the model for which prediction explanations are requested

dataset_id [str] id of the prediction dataset for which prediction explanations are requested
(as uploaded from `Project.upload_dataset`)

Returns

job [ShapMatrixJob] The job computing the SHAP based prediction explanations

Raises

ClientError If the server responded with 4xx status. Possible reasons are project, model or dataset don't exist, user is not allowed or model doesn't support SHAP based prediction explanations

ServerError If the server responded with 5xx status

classmethod `list` (*project_id*)

Fetch all the computed SHAP prediction explanations for a project.

Parameters

project_id [str] id of the project

Returns

List of ShapMatrix A list of *ShapMatrix* objects

Raises

datarobot.errors.ClientError if the server responded with 4xx status

datarobot.errors.ServerError if the server responded with 5xx status

classmethod `get` (*project_id*, *id*)

Retrieve the specific SHAP matrix.

Parameters

project_id [str] id of the project the model belongs to

id [str] id of the SHAP matrix

Returns

ShapMatrix object representing specified record

`get_as_dataframe()`

Retrieve SHAP matrix values as dataframe.

Returns

dataframe [pandas.DataFrame] A dataframe with SHAP scores

Raises

datarobot.dse.errors.ClientError if the server responded with 4xx status.

datarobot.dse.errors.ServerError if the server responded with 5xx status.

32.28 Predictions

```
class datarobot.models.Predictions(project_id, prediction_id, model_id=None,
                                   dataset_id=None, includes_prediction_intervals=None,
                                   prediction_intervals_size=None, forecast_point=None,
                                   predictions_start_date=None, predictions_end_date=None,
                                   actual_value_column=None, explanation_algorithm=None,
                                   max_explanations=None, shap_warnings=None)
```

Represents predictions metadata and provides access to prediction results.

Examples

List all predictions for a project

```
import datarobot as dr

# Fetch all predictions for a project
all_predictions = dr.Predictions.list(project_id)

# Inspect all calculated predictions
for predictions in all_predictions:
    print(predictions) # repr includes project_id, model_id, and dataset_id
```

Retrieve predictions by id

```
import datarobot as dr

# Getting predictions by id
predictions = dr.Predictions.get(project_id, prediction_id)

# Dump actual predictions
df = predictions.get_all_as_dataframe()
print(df)
```

Attributes

project_id [str] id of the project the model belongs to

model_id [str] id of the model

prediction_id [str] id of generated predictions

includes_prediction_intervals [bool, optional] (New in v2.16) For *time series* projects only. Indicates if prediction intervals will be part of the response. Defaults to False.

prediction_intervals_size [int, optional] (New in v2.16) For *time series* projects only. Indicates the percentile used for prediction intervals calculation. Will be present only if *includes_prediction_intervals* is True.

forecast_point [datetime.datetime, optional] (New in v2.20) For *time series* projects only. This is the default point relative to which predictions will be generated, based on the forecast window of the project. See the time series *prediction documentation* for more information.

predictions_start_date [datetime.datetime or None, optional] (New in v2.20) For *time series* projects only. The start date for bulk predictions. Note that this parameter is for

generating historical predictions using the training data. This parameter should be provided in conjunction with `predictions_end_date`. Canâ€™t be provided with the `forecast_point` parameter.

predictions_end_date [datetime.datetime or None, optional] (New in v2.20) For *time series* projects only. The end date for bulk predictions, exclusive. Note that this parameter is for generating historical predictions using the training data. This parameter should be provided in conjunction with `predictions_start_date`. Canâ€™t be provided with the `forecast_point` parameter.

actual_value_column [string, optional] (New in version v2.21) For *time series* unsupervised projects only. Actual value column which was used to calculate the classification metrics and insights on the prediction dataset. Canâ€™t be provided with the `forecast_point` parameter.

explanation_algorithm [datarobot.enums.EXPLANATIONS_ALGORITHM, optional] (New in version v2.21) If set to `shap`, the response will include prediction explanations based on the SHAP explainer (SHapley Additive exPlanations). Defaults to null (no prediction explanations).

max_explanations [int, optional] (New in version v2.21) The maximum number of explanation values that should be returned for each row, ordered by absolute value, greatest to least. If null, no limit. In the case of `shap`: if the number of features is greater than the limit, the sum of remaining values will also be returned as *shapRemainingTotal*. Defaults to null. Cannot be set if *explanation_algorithm* is omitted.

shap_warnings [dict, optional] (New in version v2.21) Will be present if *explanation_algorithm* was set to `datarobot.enums.EXPLANATIONS_ALGORITHM.SHAP` and there were additivity failures during SHAP values calculation.

classmethod list (*project_id*, *model_id=None*, *dataset_id=None*)

Fetch all the computed predictions metadata for a project.

Parameters

project_id [str] id of the project

model_id [str, optional] if specified, only predictions metadata for this model will be retrieved

dataset_id [str, optional] if specified, only predictions metadata for this dataset will be retrieved

Returns

A list of *Predictions* objects

classmethod get (*project_id*, *prediction_id*)

Retrieve the specific predictions metadata

Parameters

project_id [str] id of the project the model belongs to

prediction_id [str] id of the prediction set

Returns

Predictions object representing specified

predictions

get_all_as_dataframe (*class_prefix='class_', serializer='json'*)
 Retrieve all prediction rows and return them as a pandas.DataFrame.

Parameters

class_prefix [str, optional] The prefix to append to labels in the final dataframe. Default is `class_` (e.g., apple -> class_apple)

serializer [str, optional] Serializer to use for the download. Options: `json` (default) or `csv`.

Returns

dataframe: pandas.DataFrame

Raises

datarobot.dse.errors.ClientError if the server responded with 4xx status.

datarobot.dse.errors.ServerError if the server responded with 5xx status.

download_to_csv (*filename, encoding='utf-8', serializer='json'*)
 Save prediction rows into CSV file.

Parameters

filename [str or file object] path or file object to save prediction rows

encoding [string, optional] A string representing the encoding to use in the output file, defaults to `utf-8`

serializer [str, optional] Serializer to use for the download. Options: `json` (default) or `csv`.

32.29 PredictionServer

class `datarobot.PredictionServer` (*id=None, url=None, datarobot_key=None*)
 A prediction server can be used to make predictions

Attributes

id [str] the id of the prediction server

url [str] the url of the prediction server

datarobot_key [str] the *datarobot-key* header used in requests to this prediction server

classmethod `list()`

Returns a list of prediction servers a user can use to make predictions.

New in version v2.17.

Returns

prediction_servers [list of PredictionServer instances] Contains a list of prediction servers that can be used to make predictions.

Examples

```
prediction_servers = PredictionServer.list()
prediction_servers
>>> [PredictionServer('https://example.com')]
```

32.30 Ruleset

```
class datarobot.models.Ruleset (project_id=None, parent_model_id=None, model_id=None,  
                                ruleset_id=None, rule_count=None, score=None)
```

Represents an approximation of a model with DataRobot Prime

Attributes

id [str] the id of the ruleset

rule_count [int] the number of rules used to approximate the model

score [float] the validation score of the approximation

project_id [str] the project the approximation belongs to

parent_model_id [str] the model being approximated

model_id [str or None] the model using this ruleset (if it exists). Will be None if no such model has been trained.

```
request_model ()
```

Request training for a model using this ruleset

Training a model using a ruleset is a necessary prerequisite for being able to download the code for a ruleset.

Returns

job: Job the job fitting the new Prime model

32.31 PrimeFile

```
class datarobot.models.PrimeFile (id=None, project_id=None, parent_model_id=None,  
                                model_id=None, ruleset_id=None, language=None,  
                                is_valid=None)
```

Represents an executable file available for download of the code for a DataRobot Prime model

Attributes

id [str] the id of the PrimeFile

project_id [str] the id of the project this PrimeFile belongs to

parent_model_id [str] the model being approximated by this PrimeFile

model_id [str] the prime model this file represents

ruleset_id [int] the ruleset being used in this PrimeFile

language [str] the language of the code in this file - see enums.LANGUAGE for possibilities

is_valid [bool] whether the code passed basic validation

```
download (filepath)
```

Download the code and save it to a file

Parameters

filepath: string the location to save the file to

32.32 Project

```
class datarobot.models.Project (id=None, project_name=None, mode=None, target=None,
                                target_type=None, holdout_unlocked=None, metric=None,
                                stage=None, partition=None, positive_class=None, cre-
                                ated=None, advanced_options=None, recommender=None,
                                max_train_pct=None, max_train_rows=None, scale-
                                out_max_train_pct=None, scaleout_max_train_rows=None,
                                file_name=None, feature_engineering_graphs=None, cre-
                                dentials=None, feature_engineering_prediction_point=None,
                                unsupervised_mode=None, use_feature_discovery=None, rela-
                                tionships_configuration_id=None, project_description=None)
```

A project built from a particular training dataset

Attributes

id [str] the id of the project

project_name [str] the name of the project

project_description [str] an optional description for the project

mode [int] the autopilot mode currently selected for the project - 0 for full autopilot, 1 for semi-automatic, and 2 for manual

target [str] the name of the selected target features

target_type [str] Indicating what kind of modeling is being done in this project Options are: "Regression", "Binary" (Binary classification), "Multiclass" (Multiclass classification)

holdout_unlocked [bool] whether the holdout has been unlocked

metric [str] the selected project metric (e.g. *LogLoss*)

stage [str] the stage the project has reached - one of `datarobot.enums.PROJECT_STAGE`

partition [dict] information about the selected partitioning options

positive_class [str] for binary classification projects, the selected positive class; otherwise, None

created [datetime] the time the project was created

advanced_options [dict] information on the advanced options that were selected for the project settings, e.g. a weights column or a cap of the runtime of models that can advance autopilot stages

recommender [dict] information on the recommender settings of the project (i.e. whether it is a recommender project, or the id columns)

max_train_pct [float] the maximum percentage of the project dataset that can be used without going into the validation data or being too large to submit any blueprint for training

max_train_rows [int] the maximum number of rows that can be trained on without going into the validation data or being too large to submit any blueprint for training

scaleout_max_train_pct [float] the maximum percentage of the project dataset that can be used to successfully train a scaleout model without going into the validation data. May exceed *max_train_pct*, in which case only scaleout models can be trained up to this point.

scaleout_max_train_rows [int] the maximum number of rows that can be used to successfully train a scaleout model without going into the validation data. May exceed *max_train_rows*, in which case only scaleout models can be trained up to this point.

file_name [str] the name of the file uploaded for the project dataset

feature_engineering_graphs: list, optional information about feature engineering graph such as id of the graph and linkage_keys used to connect relationships in the graph.

credentials [list, optional] a list of credentials for the feature engineering graphs.

feature_engineering_prediction_point [str, optional] additional aim parameter

unsupervised_mode [bool, optional] (New in version v2.20) defaults to False, indicates whether this is an unsupervised project.

relationships_configuration_id [str, optional] (New in version v2.21) id of the relationships configuration to use

classmethod get (*project_id*)

Gets information about a project.

Parameters

project_id [str] The identifier of the project you want to load.

Returns

project [Project] The queried project

Examples

```
import datarobot as dr
p = dr.Project.get(project_id='54e639a18bd88f08078ca831')
p.id
>>>'54e639a18bd88f08078ca831'
p.project_name
>>>'Some project name'
```

classmethod create (*sourcedata*, *project_name*='Untitled Project', *max_wait*=600, *read_timeout*=600, *dataset_filename*=None)

Creates a project with provided data.

Project creation is asynchronous process, which means that after initial request we will keep polling status of async process that is responsible for project creation until it's finished. For SDK users this only means that this method might raise exceptions related to it's async nature.

Parameters

sourcedata [basestring, file, pathlib.Path or pandas.DataFrame] Dataset to use for the project. If string can be either a path to a local file, url to publicly available file or raw file content. If using a file, the filename must consist of ASCII characters only.

project_name [str, unicode, optional] The name to assign to the empty project.

max_wait [int, optional] Time in seconds after which project creation is considered unsuccessful

read_timeout: int The maximum number of seconds to wait for the server to respond indicating that the initial upload is complete

dataset_filename [string or None, optional] (New in version v2.14) File name to use for dataset. Ignored for url and file path sources.

Returns

project [Project] Instance with initialized data.

Raises

InputNotUnderstoodError Raised if *sourcedata* isn't one of supported types.

AsyncFailureError Polling for status of async process resulted in response with unsupported status code. Beginning in version 2.1, this will be `ProjectAsyncFailureError`, a subclass of `AsyncFailureError`

AsyncProcessUnsuccessfulError Raised if project creation was unsuccessful

AsyncTimeoutError Raised if project creation took more time, than specified by `max_wait` parameter

Examples

```
p = Project.create('/home/datasets/somedataset.csv',
                  project_name="New API project")
p.id
>>> '5921731dkqshda8yd28h'
p.project_name
>>> 'New API project'
```

classmethod encrypted_string (*plaintext*)

Sends a string to DataRobot to be encrypted

This is used for passwords that DataRobot uses to access external data sources

Parameters

plaintext [str] The string to encrypt

Returns

ciphertext [str] The encrypted string

classmethod create_from_hdfs (*url*, *port=None*, *project_name=None*, *max_wait=600*)

Create a project from a datasource on a WebHDFS server.

Parameters

url [str] The location of the WebHDFS file, both server and full path. Per the DataRobot specification, must begin with *hdfs://*, e.g. *hdfs:///tmp/10kDiabetes.csv*

port [int, optional] The port to use. If not specified, will default to the server default (50070)

project_name [str, optional] A name to give to the project

max_wait [int] The maximum number of seconds to wait before giving up.

Returns

Project

Examples

```
p = Project.create_from_hdfs('hdfs:///tmp/somedataset.csv',
                             project_name="New API project")

p.id
>>> '5921731dkqshda8yd28h'
p.project_name
>>> 'New API project'
```

classmethod `create_from_data_source` (*data_source_id*, *username*, *password*,
project_name=None, *max_wait=600*)

Create a project from a data source. Either `data_source` or `data_source_id` should be specified.

Parameters

data_source_id [str] the identifier of the data source.

username [str] the username for database authentication.

password [str] the password for database authentication. The password is encrypted at server side and never saved / stored.

project_name [str, optional] optional, a name to give to the project.

max_wait [int] optional, the maximum number of seconds to wait before giving up.

Returns

Project

classmethod `create_from_dataset` (*dataset_id*, *dataset_version_id=None*,
project_name=None, *user=None*, *password=None*,
credential_id=None, *use_kerberos=None*)

Create a Project from a `datarobot.Dataset`

Parameters

dataset_id: string The ID of the dataset entry to user for the project's Dataset

dataset_version_id: string, optional The ID of the dataset version to use for the project dataset. If not specified - uses latest version associated with `dataset_id`

project_name: string, optional The name of the project to be created. If not specified, will be "Untitled Project" for database connections, otherwise the project name will be based on the file used.

user: string, optional The username for database authentication.

password: string, optional The password (in cleartext) for database authentication. The password will be encrypted on the server side in scope of HTTP request and never saved or stored

credential_id: string, optional The ID of the set of credentials to use instead of user and password.

use_kerberos: bool, optional Server default is False. If true, use kerberos authentication for database authentication.

Returns

Project

classmethod `from_async` (*async_location*, *max_wait=600*)

Given a temporary async status location poll for no more than `max_wait` seconds until the async process (project creation or setting the target, for example) finishes successfully, then return the ready project

Parameters

async_location [str] The URL for the temporary async status resource. This is returned as a header in the response to a request that initiates an async process

max_wait [int] The maximum number of seconds to wait before giving up.

Returns

project [Project] The project, now ready

Raises

ProjectAsyncFailureError If the server returned an unexpected response while polling for the asynchronous operation to resolve

AsyncProcessUnsuccessfulError If the final result of the asynchronous operation was a failure

AsyncTimeoutError If the asynchronous operation did not resolve within the time specified

```
classmethod start (sourcedata,          target=None,          project_name='Untitled Project',
                    worker_count=None,      metric=None,      autopilot_on=True,
                    blueprint_threshold=None, response_cap=None, partitioning_method=None,
                    positive_class=None,    target_type=None, unsupervised_mode=False,
                    blend_best_models=None, prepare_model_for_deployment=None, consider_blenders_in_recommendation=None,
                                         scoring_code_only=None,
                    min_secondary_validation_model_count=None, shap_only_mode=None)
```

Chain together project creation, file upload, and target selection.

Note: While this function provides a simple means to get started, it does not expose all possible parameters. For advanced usage, using `create` and `set_target` directly is recommended.

Parameters

sourcedata [str or pandas.DataFrame] The path to the file to upload. Can be either a path to a local file or a publicly accessible URL (starting with `http://`, `https://`, `file://`, or `s3://`). If the source is a DataFrame, it will be serialized to a temporary buffer. If using a file, the filename must consist of ASCII characters only.

target [str, optional] The name of the target column in the uploaded file. Should not be provided if `unsupervised_mode` is `True`.

project_name [str] The project name.

Returns

project [Project] The newly created and initialized project.

Other Parameters

worker_count [int, optional] The number of workers that you want to allocate to this project.

metric [str, optional] The name of metric to use.

autopilot_on [boolean, default `True`] Whether or not to begin modeling automatically.

blueprint_threshold [int, optional] Number of hours the model is permitted to run. Minimum 1

response_cap [float, optional] Quantile of the response distribution to use for response capping. Must be in range 0.5 .. 1.0

partitioning_method [PartitioningMethod object, optional] It should be one of PartitioningMethod object.

positive_class [str, float, or int; optional] Specifies a level of the target column that should be treated as the positive class for binary classification. May only be specified for binary classification targets.

target_type [str, optional] Override the automatically selected target_type. An example usage would be setting the target_type="Mutliclass" when you want to preform a multiclass classification task on a numeric column that has a low cardinality. You can use TARGET_TYPE enum.

unsupervised_mode [boolean, default False] Specifies whether to create an unsupervised project.

blend_best_models: bool, optional blend best models during Autopilot run

scoring_code_only: bool, optional Keep only models that can be converted to scorable java code during Autopilot run.

shap_only_mode: bool, optional Keep only models that support SHAP values during Autopilot run. Use SHAP-based insights wherever possible. Defaults to False.

prepare_model_for_deployment: bool, optional Prepare model for deployment during Autopilot run. The preparation includes creating reduced feature list models, retraining best model on higher sample size, computing insights and assigning "RECOMMENDED FOR DEPLOYMENT" label.

consider_blenders_in_recommendation: bool, optional Include blenders when selecting a model to prepare for deployment in an Autopilot Run. Defaults to False.

min_secondary_validation_model_count: int, optional Compute "backtest" scores (datetime models) or cross validation scores for the specified number of highest ranking models on the Leaderboard, if over the Autopilot default.

Raises

AsyncFailureError Polling for status of async process resulted in response with unsupported status code

AsyncProcessUnsuccessfulError Raised if project creation or target setting was unsuccessful

AsyncTimeoutError Raised if project creation or target setting timed out

Examples

```
Project.start("./tests/fixtures/file.csv",
              "a_target",
              project_name="test_name",
              worker_count=4,
              metric="a_metric")
```

This is an example of using a URL to specify the datasource:

```
Project.start("https://example.com/data/file.csv",
              "a_target",
              project_name="test_name",
              worker_count=4,
              metric="a_metric")
```

classmethod list (*search_params=None*)

Returns the projects associated with this account.

Parameters

search_params [dict, optional.] If not *None*, the returned projects are filtered by lookup. Currently you can query projects by:

- `project_name`

Returns

projects [list of Project instances] Contains a list of projects associated with this user account.

Raises

TypeError Raised if `search_params` parameter is provided, but is not of supported type.

Examples

List all projects .. code-block:: python

```
p_list = Project.list() p_list >>> [Project(âProject Oneâ), Project(âTwoâ)]
```

Search for projects by name .. code-block:: python

```
Project.list(search_params={âproject_nameâ: âredâ}) >>>
[Project(âPredtimeâ), Project(âFred Projectâ)]
```

refresh()

Fetches the latest state of the project, and updates this object with that information. This is an inplace update, not a new object.

Returns

self [Project] the now-updated project

delete()

Removes this project from your account.

set_target (*target=None, mode='quick', metric=None, quickrun=None, worker_count=None, positive_class=None, partitioning_method=None, featurelist_id=None, advanced_options=None, max_wait=600, target_type=None, feature_engineering_graphs=None, credentials=None, feature_engineering_prediction_point=None, unsupervised_mode=False, relationships_configuration_id=None*)

Set target variable of an existing project and begin the autopilot process (unless manual mode is specified).

Target setting is asynchronous process, which means that after initial request we will keep polling status of async process that is responsible for target setting until itâs finished. For SDK users this only means that this method might raise exceptions related to itâs async nature.

When execution returns to the caller, the autopilot process will already have commenced (again, unless manual mode is specified).

Parameters

target [str, optional] The name of the target column in the uploaded file. Should not be provided if `unsupervised_mode` is `True`.

mode [str, optional] You can use `AUTOPILOT_MODE` enum to choose between

- `AUTOPILOT_MODE.FULL_AUTO`

- `AUTOPILOT_MODE.MANUAL`
- `AUTOPILOT_MODE.QUICK`

If unspecified, `QUICK` is used. If the `MANUAL` value is used, the model creation process will need to be started by executing the `start_autopilot` function with the desired featurelist. It will start immediately otherwise.

metric [str, optional] Name of the metric to use for evaluating models. You can query the metrics available for the target by way of `Project.get_metrics`. If none is specified, then the default recommended by DataRobot is used.

quickrun [bool, optional] Deprecated - pass `AUTOPILOT_MODE.QUICK` as mode instead. Sets whether project should be run in `quick run` mode. This setting causes DataRobot to recommend a more limited set of models in order to get a base set of models and insights more quickly.

worker_count [int, optional] The number of concurrent workers to request for this project. If *None*, then the default is used. (New in version v2.14) Setting this to -1 will request the maximum number available to your account.

partitioning_method [PartitioningMethod object, optional] It should be one of PartitioningMethod object.

positive_class [str, float, or int; optional] Specifies a level of the target column that should be treated as the positive class for binary classification. May only be specified for binary classification targets.

featurelist_id [str, optional] Specifies which feature list to use.

advanced_options [AdvancedOptions, optional] Used to set advanced options of project creation.

max_wait [int, optional] Time in seconds after which target setting is considered unsuccessful.

target_type [str, optional] Override the automatically selected target_type. An example usage would be setting the `target_type='Multiclass'` when you want to preform a multiclass classification task on a numeric column that has a low cardinality. You can use `TARGET_TYPE` enum.

feature_engineering_graphs: list, optional information about feature engineering graph such as id of the graph and linkage_keys used to connect relationships in the graph.

credentials: list, optional, a list of credentials for the feature engineering graphs.

feature_engineering_prediction_point [str, optional] additional aim parameter.

unsupervised_mode [boolean, default `False`] (New in version v2.20) Specifies whether to create an unsupervised project. If `True`, `target` may not be provided.

relationships_configuration_id [str, optional] (New in version v2.21) id of the relationships configuration to use

Returns

project [Project] The instance with updated attributes.

Raises

AsyncFailureError Polling for status of async process resulted in response with unsupported status code

AsyncProcessUnsuccessfulError Raised if target setting was unsuccessful

AsyncTimeoutError Raised if target setting took more time, than specified by `max_wait` parameter

TypeError Raised if `advanced_options`, `partitioning_method` or `target_type` is provided, but is not of supported type

See also:

`datarobot.models.Project.start` combines project creation, file upload, and target selection. Provides fewer options, but is useful for getting started quickly.

get_models (*order_by=None, search_params=None, with_metric=None*)

List all completed, successful models in the leaderboard for the given project.

Parameters

order_by [str or list of strings, optional] If not *None*, the returned models are ordered by this attribute. If *None*, the default return is the order of default project metric.

Allowed attributes to sort by are:

- `metric`
- `sample_pct`

If the sort attribute is preceded by a hyphen, models will be sorted in descending order, otherwise in ascending order.

Multiple sort attributes can be included as a comma-delimited string or in a list e.g. `order_by='sample_pct,-metric'` or `order_by=[sample_pct, -metric]`

Using *metric* to sort by will result in models being sorted according to their validation score by how well they did according to the project metric.

search_params [dict, optional.] If not *None*, the returned models are filtered by lookup. Currently you can query models by:

- `name`
- `sample_pct`
- `is_starred`

with_metric [str, optional.] If not *None*, the returned models will only have scores for this metric. Otherwise all the metrics are returned.

Returns

models [a list of Model instances.] All of the models that have been trained in this project.

Raises

TypeError Raised if `order_by` or `search_params` parameter is provided, but is not of supported type.

Examples

```
Project.get('pid').get_models(order_by=['-sample_pct',
                                         'metric'])

# Getting models that contain "Ridge" in name
# and with sample_pct more than 64
```

(continues on next page)

(continued from previous page)

```

Project.get('pid').get_models(
    search_params={
        'sample_pct__gt': 64,
        'name': "Ridge"
    })

# Filtering models based on 'starred' flag:
Project.get('pid').get_models(search_params={'is_starred': True})

```

get_datetime_models()

List all models in the project as DatetimeModels

Requires the project to be datetime partitioned. If it is not, a ClientError will occur.

Returns

models [list of DatetimeModel] the datetime models

get_prime_models()

List all DataRobot Prime models for the project Prime models were created to approximate a parent model, and have downloadable code.

Returns

models [list of PrimeModel]

get_prime_files(parent_model_id=None, model_id=None)

List all downloadable code files from DataRobot Prime for the project

Parameters

parent_model_id [str, optional] Filter for only those prime files approximating this parent model

model_id [str, optional] Filter for only those prime files with code for this prime model

Returns

files: list of PrimeFile

get_datasets()

List all the datasets that have been uploaded for predictions

Returns

datasets [list of PredictionDataset instances]

upload_dataset(sourcedata, max_wait=600, read_timeout=600, forecast_point=None, predictions_start_date=None, predictions_end_date=None, dataset_filename=None, relax_known_in_advance_features_check=None, credentials=None, actual_value_column=None)

Upload a new dataset to make predictions against

Parameters

sourcedata [str, file or pandas.DataFrame] Data to be used for predictions. If string, can be either a path to a local file, a publicly accessible URL (starting with `http://`, `https://`, `file://`, or `s3://`), or raw file content. If using a file on disk, the filename must consist of ASCII characters only.

max_wait [int, optional] The maximum number of seconds to wait for the uploaded dataset to be processed before raising an error.

read_timeout [int, optional] The maximum number of seconds to wait for the server to respond indicating that the initial upload is complete

forecast_point [datetime.datetime or None, optional] (New in version v2.8) May only be specified for time series projects, otherwise the upload will be rejected. The time in the dataset relative to which predictions should be generated in a time series project. See the [Time Series documentation](#) for more information. If not provided, will default to using the latest forecast point in the dataset.

predictions_start_date [datetime.datetime or None, optional] (New in version v2.11) May only be specified for time series projects. The start date for bulk predictions. Note that this parameter is for generating historical predictions using the training data. This parameter should be provided in conjunction with `predictions_end_date`. Cannot be provided with the `forecast_point` parameter.

predictions_end_date [datetime.datetime or None, optional] (New in version v2.11) May only be specified for time series projects. The end date for bulk predictions, exclusive. Note that this parameter is for generating historical predictions using the training data. This parameter should be provided in conjunction with `predictions_start_date`. Cannot be provided with the `forecast_point` parameter.

actual_value_column [string, optional] (New in version v2.21) Actual value column name, valid for the prediction files if the project is unsupervised and the dataset is considered as bulk predictions dataset. Cannot be provided with the `forecast_point` parameter.

dataset_filename [string or None, optional] (New in version v2.14) File name to use for the dataset. Ignored for url and file path sources.

relax_known_in_advance_features_check [bool, optional] (New in version v2.15) For time series projects only. If True, missing values in the known in advance features are allowed in the forecast window at the prediction time. If omitted or False, missing values are not allowed.

credentials: list, optional, a list of credentials for the feature engineering graphs used in Feature discovery project

Returns

âĀĤâĤâĤ-

dataset [PredictionDataset] The newly uploaded dataset.

Raises

InputNotUnderstoodError Raised if `sourcedata` isnâĀĤt one of supported types.

AsyncFailureError Raised if polling for the status of an async process resulted in a response with an unsupported status code.

AsyncProcessUnsuccessfulError Raised if project creation was unsuccessful (i.e. the server reported an error in uploading the dataset).

AsyncTimeoutError Raised if processing the uploaded dataset took more time than specified by the `max_wait` parameter.

ValueError Raised if `forecast_point` or `predictions_start_date` and `predictions_end_date` are provided, but are not of the supported type.

upload_dataset_from_data_source (`data_source_id`, `username`, `password`, `max_wait=600`, `forecast_point=None`, `relax_known_in_advance_features_check=None`, `credentials=None`, `predictions_start_date=None`, `predictions_end_date=None`, `actual_value_column=None`)

Upload a new dataset from a data source to make predictions against

Parameters

data_source_id [str] The identifier of the data source.

username [str] The username for database authentication.

password [str] The password for database authentication. The password is encrypted at server side and never saved / stored.

max_wait [int, optional] Optional, the maximum number of seconds to wait before giving up.

forecast_point [datetime.datetime or None, optional] (New in version v2.8) For time series projects only. This is the default point relative to which predictions will be generated, based on the forecast window of the project. See the time series [prediction documentation](#) for more information.

relax_known_in_advance_features_check [bool, optional] (New in version v2.15) For time series projects only. If True, missing values in the known in advance features are allowed in the forecast window at the prediction time. If omitted or False, missing values are not allowed.

credentials: list, optional, a list of credentials for the feature engineering graphs used in Feature discovery project

predictions_start_date [datetime.datetime or None, optional] (New in version v2.20) For time series projects only. The start date for bulk predictions. Note that this parameter is for generating historical predictions using the training data. This parameter should be provided in conjunction with `predictions_end_date`. Can't be provided with the `forecast_point` parameter.

predictions_end_date [datetime.datetime or None, optional] (New in version v2.20) For time series projects only. The end date for bulk predictions, exclusive. Note that this parameter is for generating historical predictions using the training data. This parameter should be provided in conjunction with `predictions_start_date`. Can't be provided with the `forecast_point` parameter.

actual_value_column [string, optional] (New in version v2.21) Actual value column name, valid for the prediction files if the project is unsupervised and the dataset is considered as bulk predictions dataset. Cannot be provided with the `forecast_point` parameter.

Returns

dataset [PredictionDataset] the newly uploaded dataset

get_blueprints()

List all blueprints recommended for a project.

Returns

menu [list of Blueprint instances] All the blueprints recommended by DataRobot for a project

get_features()

List all features for this project

Returns

list of Feature all features for this project

get_modeling_features (*batch_size=None*)

List all modeling features for this project

Only available once the target and partitioning settings have been set. For more information on the distinction between input and modeling features, see the *time series documentation<input_vs_modeling>*.

Parameters

batch_size [int, optional] The number of features to retrieve in a single API call. If specified, the client may make multiple calls to retrieve the full list of features. If not specified, an appropriate default will be chosen by the server.

Returns

list of ModelingFeature All modeling features in this project

get_featurelists()

List all featurelists created for this project

Returns

list of Featurelist all featurelists created for this project

get_associations (*assoc_type, metric, featurelist_id=None*)

Get the association statistics and metadata for a project's informative features

New in version v2.17.

Parameters

assoc_type [string or None] the type of association, must be either "association" or "correlation"

metric [string or None] the specified association metric, belongs under either association or correlation umbrella

featurelist_id [string or None] the desired featurelist for which to get association statistics (New in version v2.19)

Returns

association_data [dict] pairwise metric strength data, clustering data, and ordering data for Feature Association Matrix visualization

get_association_featurelists()

List featurelists and get feature association status for each

New in version v2.19.

Returns

feature_lists [dict] dict with "featurelists" as key, with list of featurelists as values

get_association_matrix_details (*feature1, feature2*)

Get a sample of the actual values used to measure the association between a pair of features

New in version v2.17.

Parameters

feature1 [str] Feature name for the first feature of interest

feature2 [str] Feature name for the second feature of interest

Returns

dict This data has 3 keys: features, values, and types

values [list] a list of triplet lists e.g. {`values`: [[460.0, 428.5, 0.001], [1679.3, 259.0, 0.001], ...]} The first entry of each list is a value of feature1, the second entry of each list is a value of feature2, and the third is the relative frequency of the pair of datapoints in the sample.

features [list of the passed features, [feature1, feature2]]

types [list of the passed features' types inferred by DataRobot, e.g. [`float`, `float`, `float`]]

get_modeling_featurelists (*batch_size=None*)

List all modeling featurelists created for this project

Modeling featurelists can only be created after the target and partitioning options have been set for a project. In time series projects, these are the featurelists that can be used for modeling; in other projects, they behave the same as regular featurelists.

See the [time series documentation](#) for more information.

Parameters

batch_size [int, optional] The number of featurelists to retrieve in a single API call. If specified, the client may make multiple calls to retrieve the full list of features. If not specified, an appropriate default will be chosen by the server.

Returns

list of ModelingFeaturelist all modeling featurelists in this project

create_type_transform_feature (*name, parent_name, variable_type, replacement=None, date_extraction=None, max_wait=600*)

Create a new feature by transforming the type of an existing feature in the project

Note that only the following transformations are supported:

1. Text to categorical or numeric
2. Categorical to text or numeric
3. Numeric to categorical
4. Date to categorical or numeric

Note: Special considerations when casting numeric to categorical

There are two parameters which can be used for `variableType` to convert numeric data to categorical levels. These differ in the assumptions they make about the input data, and are very important when considering the data that will be used to make predictions. The assumptions that each makes are:

- `categorical` : The data in the column is all integral, and there are no missing values. If either of these conditions do not hold in the training set, the transformation will be rejected. During predictions, if any of the values in the parent column are missing, the predictions will error. Note that `CATEGORICAL` is deprecated in v2.21.
- `categoricalInt` : **New in v2.6** All of the data in the column should be considered categorical in its string form when cast to an int by truncation. For example the value 3 will be cast as the string 3 and the value 3.14 will also be cast as the string 3. Further, the value -3.6 will become the string -3. Missing values will still be recognized as missing.

For convenience these are represented in the enum `VARIABLE_TYPE_TRANSFORM` with the names `CATEGORICAL` and `CATEGORICAL_INT`.

Parameters

- name** [str] The name to give to the new feature
- parent_name** [str] The name of the feature to transform
- variable_type** [str] The type the new column should have. See the values within `datarobot.enums.VARIABLE_TYPE_TRANSFORM`. Note that `CATEGORICAL` is deprecated in v2.21.
- replacement** [str or float, optional] The value that missing or unconverable data should have
- date_extraction** [str, optional] Must be specified when `parent_name` is a date column (and left `None` otherwise). Specifies which value from a date should be extracted. See the list of values in `datarobot.enums.DATE_EXTRACTION`
- max_wait** [int, optional] The maximum amount of time to wait for DataRobot to finish processing the new column. This process can take more time with more data to process. If this operation times out, an `AsyncTimeoutError` will occur. DataRobot continues the processing and the new column may successfully be constructed.

Returns

Feature The data of the new Feature

Raises

- AsyncFailureError** If any of the responses from the server are unexpected
- AsyncProcessUnsuccessfulError** If the job being waited for has failed or has been cancelled
- AsyncTimeoutError** If the resource did not resolve in time

create_featurelist (*name, features*)

Creates a new featurelist

Parameters

- name** [str] The name to give to this new featurelist. Names must be unique, so an error will be returned from the server if this name has already been used in this project.
- features** [list of str] The names of the features. Each feature must exist in the project already.

Returns

Featurelist newly created featurelist

Raises

- DuplicateFeaturesError** Raised if *features* variable contains duplicate features

Examples

```
project = Project.get('5223deadbeefdeadbeef0101')
flists = project.get_featurelists()

# Create a new featurelist using a subset of features from an
# existing featurelist
flist = flists[0]
features = flist.features[::2] # Half of the features
```

(continues on next page)

(continued from previous page)

```
new_flist = project.create_featurelist(name='Feature Subset',
                                       features=features)
```

create_modeling_featurelist (*name, features*)

Create a new modeling featurelist

Modeling featurelists can only be created after the target and partitioning options have been set for a project. In time series projects, these are the featurelists that can be used for modeling; in other projects, they behave the same as regular featurelists.

See the [time series documentation](#) for more information.

Parameters

name [str] the name of the modeling featurelist to create. Names must be unique within the project, or the server will return an error.

features [list of str] the names of the features to include in the modeling featurelist. Each feature must be a modeling feature.

Returns

featurelist [ModelingFeaturelist] the newly created featurelist

Examples

```
project = Project.get('1234deadbeeffeeddead4321')
modeling_features = project.get_modeling_features()
selected_features = [feat.name for feat in modeling_features][:5] # select_
↳first five
new_flist = project.create_modeling_featurelist('Model This', selected_
↳features)
```

get_metrics (*feature_name*)

Get the metrics recommended for modeling on the given feature.

Parameters

feature_name [str] The name of the feature to query regarding which metrics are recommended for modeling.

Returns

feature_name: str The name of the feature that was looked up

available_metrics: list of str An array of strings representing the appropriate metrics. If the feature cannot be selected as the target, then this array will be empty.

metric_details: list of dict The list of *metricDetails* objects

metric_name: str Name of the metric

supports_timeseries: boolean This metric is valid for timeseries

supports_multiclass: boolean This metric is valid for mutliclass classifciaton

supports_binary: boolean This metric is valid for binary classifciaton

supports_regression: boolean This metric is valid for regression

ascending: boolean Should the metric be sorted in ascending order

get_status()

Query the server for project status.

Returns

status [dict] Contains:

- **autopilot_done**: a boolean.
- **stage**: a short string indicating which stage the project is in.
- **stage_description**: a description of what stage means.

Examples

```
{ "autopilot_done": False,
  "stage": "modeling",
  "stage_description": "Ready for modeling" }
```

pause_autopilot()

Pause autopilot, which stops processing the next jobs in the queue.

Returns

paused [boolean] Whether the command was acknowledged

unpause_autopilot()

Unpause autopilot, which restarts processing the next jobs in the queue.

Returns

unpaused [boolean] Whether the command was acknowledged.

start_autopilot (*featurelist_id*, *mode='auto'*, *blend_best_models=True*, *scoring_code_only=False*, *prepare_model_for_deployment=True*)

Starts autopilot on provided featurelist with the specified Autopilot settings, halting the current autopilot run.

Only one autopilot can be running at the time. That's why any ongoing autopilot on a different featurelist will be halted - modeling jobs in queue would not be affected but new jobs would not be added to queue by the halted autopilot.

Parameters

featurelist_id [str] Identifier of featurelist that should be used for autopilot

mode [str, optional] The Autopilot mode to run. You can use `AUTOPILOT_MODE` enum to choose between

- `AUTOPILOT_MODE.FULL_AUTO`

If unspecified, `FULL_AUTO` is used.

blend_best_models [bool, optional] Blend best models during Autopilot run. This option is not supported in SHAP-only mode.

scoring_code_only [bool, optional] Keep only models that can be converted to scorable java code during Autopilot run.

prepare_model_for_deployment [bool, optional] Prepare model for deployment during Autopilot run. The preparation includes creating reduced feature list models, retraining best model on higher sample size, computing insights and assigning `RECOMMENDED FOR DEPLOYMENT` label.

Raises

AppPlatformError Raised project's target was not selected or the settings for Autopilot are invalid for the project project.

train (*trainable*, *sample_pct=None*, *featurelist_id=None*, *source_project_id=None*, *scoring_type=None*, *training_row_count=None*, *monotonic_increasing_featurelist_id=<object object>*, *monotonic_decreasing_featurelist_id=<object object>*)
Submit a job to the queue to train a model.

Either *sample_pct* or *training_row_count* can be used to specify the amount of data to use, but not both. If neither are specified, a default of the maximum amount of data that can safely be used to train any blueprint without going into the validation data will be selected.

In smart-sampled projects, *sample_pct* and *training_row_count* are assumed to be in terms of rows of the minority class.

Note: If the project uses datetime partitioning, use *Project.train_datetime* instead.

Parameters

trainable [str or Blueprint] For str, this is assumed to be a blueprint_id. If no *source_project_id* is provided, the *project_id* will be assumed to be the project that this instance represents.

Otherwise, for a Blueprint, it contains the blueprint_id and source_project_id that we want to use. *featurelist_id* will assume the default for this project if not provided, and *sample_pct* will default to using the maximum training value allowed for this project's partition setup. *source_project_id* will be ignored if a Blueprint instance is used for this parameter

sample_pct [float, optional] The amount of data to use for training, as a percentage of the project dataset from 0 to 100.

featurelist_id [str, optional] The identifier of the featurelist to use. If not defined, the default for this project is used.

source_project_id [str, optional] Which project created this blueprint_id. If None, it defaults to looking in this project. Note that you must have read permissions in this project.

scoring_type [str, optional] Either `SCORING_TYPE.validation` or `SCORING_TYPE.cross_validation`. `SCORING_TYPE.validation` is available for every partitioning type, and indicates that the default model validation should be used for the project. If the project uses a form of cross-validation partitioning, `SCORING_TYPE.cross_validation` can also be used to indicate that all of the available training/validation combinations should be used to evaluate the model.

training_row_count [int, optional] The number of rows to use to train the requested model.

monotonic_increasing_featurelist_id [str, optional] (new in version 2.11) the id of the featurelist that defines the set of features with a monotonically increasing relationship to the target. Passing None disables increasing monotonicity constraint. Default (`dr.enums.MONOTONICITY_FEATURELIST_DEFAULT`) is the one specified by the blueprint.

monotonic_decreasing_featurelist_id [str, optional] (new in version 2.11) the id of the featurelist that defines the set of features with a monotonically decreasing relationship to the target. Passing None disables decreasing monotonicity constraint. Default (`dr.enums.MONOTONICITY_FEATURELIST_DEFAULT`) is the one specified by the blueprint.

Returns

model_job_id [str] id of created job, can be used as parameter to `ModelJob.get` method or `wait_for_async_model_creation` function

Examples

Use a Blueprint instance:

```
blueprint = project.get_blueprints()[0]
model_job_id = project.train(blueprint, training_row_count=project.max_train_
↪rows)
```

Use a `blueprint_id`, which is a string. In the first case, it is assumed that the blueprint was created by this project. If you are using a blueprint used by another project, you will need to pass the id of that other project as well.

```
blueprint_id = 'e1c7fc29ba2e612a72272324b8a842af'
project.train(blueprint, training_row_count=project.max_train_rows)

another_project.train(blueprint, source_project_id=project.id)
```

You can also easily use this interface to train a new model using the data from an existing model:

```
model = project.get_models()[0]
model_job_id = project.train(model.blueprint.id,
                             sample_pct=100)
```

train_datetime (*blueprint_id*, *featurelist_id=None*, *training_row_count=None*,
training_duration=None, *source_project_id=None*, *mono-*
tonic_increasing_featurelist_id=<object object>, *mono-*
tonic_decreasing_featurelist_id=<object object>, *use_project_settings=False*)

Create a new model in a datetime partitioned project

If the project is not datetime partitioned, an error will occur.

All durations should be specified with a duration string such as those returned by the [partitioning_methods.construct_duration_string](#) helper method. Please see [datetime partitioned project documentation](#) for more information on duration strings.

Parameters

blueprint_id [str] the blueprint to use to train the model

featurelist_id [str, optional] the featurelist to use to train the model. If not specified, the project default will be used.

training_row_count [int, optional] the number of rows of data that should be used to train the model. If specified, neither `training_duration` nor `use_project_settings` may be specified.

training_duration [str, optional] a duration string specifying what time range the data used to train the model should span. If specified, neither `training_row_count` nor `use_project_settings` may be specified.

use_project_settings [bool, optional] (New in version v2.20) defaults to `False`. If `True`, indicates that the custom backtest partitioning settings specified by the user will be used to train the model and evaluate backtest scores. If specified, neither `training_row_count` nor `training_duration` may be specified.

source_project_id [str, optional] the id of the project this blueprint comes from, if not this project. If left unspecified, the blueprint must belong to this project.

monotonic_increasing_featurelist_id [str, optional] (New in version v2.18) optional, the id of the featurelist that defines the set of features with a monotonically increasing relationship to the target. Passing `None` disables increasing monotonicity constraint. Default (`dr.enums.MONOTONICITY_FEATURELIST_DEFAULT`) is the one specified by the blueprint.

monotonic_decreasing_featurelist_id [str, optional] (New in version v2.18) optional, the id of the featurelist that defines the set of features with a monotonically decreasing relationship to the target. Passing `None` disables decreasing monotonicity constraint. Default (`dr.enums.MONOTONICITY_FEATURELIST_DEFAULT`) is the one specified by the blueprint.

Returns

job [ModelJob] the created job to build the model

blend (*model_ids*, *blender_method*)

Submit a job for creating blender model. Upon success, the new job will be added to the end of the queue.

Parameters

model_ids [list of str] List of model ids that will be used to create blender. These models should have completed validation stage without errors, and can't be blenders, DataRobot Prime or scaleout models.

blender_method [str] Chosen blend method, one from `datarobot.enums.BLENDER_METHOD`. If this is a time series project, only methods in `datarobot.enums.TS_BLENDER_METHOD` are allowed.

Returns

model_job [ModelJob] New `ModelJob` instance for the blender creation job in queue.

See also:

[`datarobot.models.Project.check_blendable`](#) to confirm if models can be blended

check_blendable (*model_ids*, *blender_method*)

Check if the specified models can be successfully blended

Parameters

model_ids [list of str] List of model ids that will be used to create blender. These models should have completed validation stage without errors, and can't be blenders, DataRobot Prime or scaleout models.

blender_method [str] Chosen blend method, one from `datarobot.enums.BLENDER_METHOD`. If this is a time series project, only methods in `datarobot.enums.TS_BLENDER_METHOD` are allowed.

Returns

[*EligibilityResult*](#)

get_all_jobs (*status=None*)

Get a list of jobs

This will give Jobs representing any type of job, including modeling or predict jobs.

Parameters

status [QUEUE_STATUS enum, optional] If called with QUEUE_STATUS.INPROGRESS, will return the jobs that are currently running.

If called with QUEUE_STATUS.QUEUE, will return the jobs that are waiting to be run.

If called with QUEUE_STATUS.ERROR, will return the jobs that have errored.

If no value is provided, will return all jobs currently running or waiting to be run.

Returns

jobs [list] Each is an instance of Job

get_blenders ()

Get a list of blender models.

Returns

list of BlenderModel list of all blender models in project.

get_frozen_models ()

Get a list of frozen models

Returns

list of FrozenModel list of all frozen models in project.

get_model_jobs (*status=None*)

Get a list of modeling jobs

Parameters

status [QUEUE_STATUS enum, optional] If called with QUEUE_STATUS.INPROGRESS, will return the modeling jobs that are currently running.

If called with QUEUE_STATUS.QUEUE, will return the modeling jobs that are waiting to be run.

If called with QUEUE_STATUS.ERROR, will return the modeling jobs that have errored.

If no value is provided, will return all modeling jobs currently running or waiting to be run.

Returns

jobs [list] Each is an instance of ModelJob

get_predict_jobs (*status=None*)

Get a list of prediction jobs

Parameters

status [QUEUE_STATUS enum, optional] If called with QUEUE_STATUS.INPROGRESS, will return the prediction jobs that are currently running.

If called with QUEUE_STATUS.QUEUE, will return the prediction jobs that are waiting to be run.

If called with QUEUE_STATUS.ERROR, will return the prediction jobs that have errored.

If called without a status, will return all prediction jobs currently running or waiting to be run.

Returns

jobs [list] Each is an instance of PredictJob

wait_for_autopilot (*check_interval=20.0, timeout=86400, verbosity=1*)

Blocks until autopilot is finished. This will raise an exception if the autopilot mode is changed from AUTOPILOT_MODE.FULL_AUTO.

It makes API calls to sync the project state with the server and to look at which jobs are enqueued.

Parameters

check_interval [float or int] The maximum time (in seconds) to wait between checks for whether autopilot is finished

timeout [float or int or None] After this long (in seconds), we give up. If None, never timeout.

verbosity: This should be VERBOSITY_LEVEL.SILENT or VERBOSITY_LEVEL.VERBOSE. For VERBOSITY_LEVEL.SILENT, nothing will be displayed about progress. For VERBOSITY_LEVEL.VERBOSE, the number of jobs in progress or queued is shown. Note that new jobs are added to the queue along the way.

Raises

AsyncTimeoutError If autopilot does not finished in the amount of time specified

RuntimeError If a condition is detected that indicates that autopilot will not complete on its own

rename (*project_name*)

Update the name of the project.

Parameters

project_name [str] The new name

set_project_description (*project_description*)

Set or Update the project description.

Parameters

project_description [str] The new description for this project.

unlock_holdout ()

Unlock the holdout for this project.

This will cause subsequent queries of the models of this project to contain the metric values for the holdout set, if it exists.

Take care, as this cannot be undone. Remember that best practice is to select a model before analyzing the model performance on the holdout set

set_worker_count (*worker_count*)

Sets the number of workers allocated to this project.

Note that this value is limited to the number allowed by your account. Lowering the number will not stop currently running jobs, but will cause the queue to wait for the appropriate number of jobs to finish before attempting to run more jobs.

Parameters

worker_count [int] The number of concurrent workers to request from the pool of workers. (New in version v2.14) Setting this to -1 will update the number of workers to the maximum available to your account.

get_leaderboard_ui_permalink ()

Returns

url [str] Permanent static hyperlink to a project leaderboard.

open_leaderboard_browser ()

Opens project leaderboard in web browser.

Note: If text-mode browsers are used, the calling process will block until the user exits the browser.

get_rating_table_models ()

Get a list of models with a rating table

Returns

list of RatingTableModel list of all models with a rating table in project.

get_rating_tables ()

Get a list of rating tables

Returns

list of RatingTable list of rating tables in project.

get_access_list ()

Retrieve users who have access to this project and their access levels

New in version v2.15.

Returns

list of *SharingAccess*

share (*access_list*, *send_notification=None*, *include_feature_discovery_entities=None*)

Modify the ability of users to access this project

New in version v2.15.

Parameters

access_list [list of *SharingAccess*] the modifications to make.

send_notification [boolean, default `True`] (New in version v2.21) optional, whether or not an email notification should be sent, default to `True`

include_feature_discovery_entities [boolean, default `False`] (New in version v2.21) optional (default: `False`), whether or not to share all the related entities (feature engineering graphs and datasets) for a project with Feature Discovery enabled

Raises

datarobot.ClientError : if you do not have permission to share this project, if the user you're sharing with doesn't exist, if the same user appears multiple times in the `access_list`, or if these changes would leave the project without an owner

Examples

Transfer access to the project from `old_user@datarobot.com` to `new_user@datarobot.com`

```
import datarobot as dr

new_access = dr.SharingAccess(new_user@datarobot.com,
                              dr.enums.SHARING_ROLE.OWNER, can_share=True)
access_list = [dr.SharingAccess(old_user@datarobot.com, None), new_access]
```

(continues on next page)

(continued from previous page)

```
dr.Project.get('my-project-id').share(access_list)
```

batch_features_type_transform(*parent_names*, *variable_type*, *prefix=None*, *suffix=None*, *max_wait=600*)

Create new features by transforming the type of existing ones.

New in version v2.17.

Note: The following transformations are only supported in batch mode:

1. Text to categorical or numeric
2. Categorical to text or numeric
3. Numeric to categorical

See [here](#) for special considerations when casting numeric to categorical. Date to categorical or numeric transformations are not currently supported for batch mode but can be performed individually using [create_type_transform_feature](#). Note that CATEGORICAL is deprecated in v2.21.

Parameters

parent_names [list] The list of variable names to be transformed.

variable_type [str] The type new columns should have. Can be one of 'CATEGORICAL', 'CATEGORICAL_INT', 'NUMERIC', and 'TEXT' - supported values can be found in `datarobot.enums.VARIABLE_TYPE_TRANSFORM`.

prefix [str, optional]

Note: Either `prefix`, `suffix`, or both must be provided.

The string that will preface all feature names. At least one of `prefix` and `suffix` must be specified.

suffix [str, optional]

Note: Either `prefix`, `suffix`, or both must be provided.

The string that will be appended at the end to all feature names. At least one of `prefix` and `suffix` must be specified.

max_wait [int, optional] The maximum amount of time to wait for DataRobot to finish processing the new column. This process can take more time with more data to process. If this operation times out, an `AsyncTimeoutError` will occur. DataRobot continues the processing and the new column may successfully be constructed.

Returns

list of Features all features for this project after transformation.

Raises

TypeError: If *parent_names* is not a list.

ValueError If value of `variable_type` is not from `datarobot.enums.VARIABLE_TYPE_TRANSFORM`.

AsyncFailureError If any of the responses from the server are unexpected.

AsyncProcessUnsuccessfulError If the job being waited for has failed or has been cancelled.

AsyncTimeoutError If the resource did not resolve in time.

clone_project (*new_project_name=None, max_wait=600*)

Create a fresh (post-EDA1) copy of this project that is ready for setting targets and modeling options.

Parameters

new_project_name [str, optional] The desired name of the new project. If omitted, the API will default to “Copy of <original project>”

max_wait [int, optional] Time in seconds after which project creation is considered unsuccessful

create_interaction_feature (*name, features, separator, max_wait=600*)

Create a new interaction feature by combining two categorical ones.

New in version v2.21.

Parameters

name [str] The name of final Interaction Feature

features [list(str)] List of two categorical feature names

separator [str] The character used to join the two data values, one of these ‘+ - / | & . _ , ‘

max_wait [int, optional] Time in seconds after which project creation is considered unsuccessful.

Returns

interactionFeature: datarobot.models.InteractionFeature The data of the new Interaction feature

Raises

ClientError If requested Interaction feature can not be created. Possible reasons for example are:

- one of *features* either does not exist or is of unsupported type
- feature with requested *name* already exists
- invalid separator character submitted.

AsyncFailureError If any of the responses from the server are unexpected

AsyncProcessUnsuccessfulError If the job being waited for has failed or has been cancelled

AsyncTimeoutError If the resource did not resolve in time

get_relationships_configuration ()

Get the relationships configuration for a given project

New in version v2.21.

Returns

relationships_configuration: RelationshipsConfiguration relationships configuration applied to project

class datarobot.helpers.eligibility_result.**EligibilityResult** (*supported, reason="", context=""*)

Represents whether a particular operation is supported

For instance, a function to check whether a set of models can be blended can return an EligibilityResult specifying whether or not blending is supported and why it may not be supported.

Attributes

supported [bool] whether the operation this result represents is supported

reason [str] why the operation is or is not supported

context [str] what operation isn't supported

32.33 VisualAI

class datarobot.models.visualai.**Image** (***kwargs*)

An image stored in a project's dataset.

Attributes

id: str Image ID for this image.

image_type: str Image media type. Accessing this may require a server request and an associated delay in returning.

image_bytes: [octet] Raw octets of this image. Accessing this may require a server request and an associated delay in returning.

height: int Height of the image in pixels (72 pixels per inch).

width: int Width of the image in pixels (72 pixels per inch).

classmethod **get** (*project_id, image_id*)

Get a single image object from project.

Parameters

project_id: str Project that contains the images.

image_id: str ID of image to load from the project.

class datarobot.models.visualai.**SampleImage** (***kwargs*)

A sample image in a project's dataset.

If `Project.stage` is `datarobot.enums.PROJECT_STAGE.EDA2` then the `target_*` attributes of this class will have values, otherwise the values will all be `None`.

Attributes

image: Image Image object.

target_value: str Value associated with the `feature_name`.

classmethod **list** (*project_id, feature_name, target_value=None, offset=None, limit=None*)

Get sample images from a project.

Parameters

project_id: str Project that contains the images.

feature_name: str Name of feature column that contains images.

target_value: str Target value to filter images.

offset: int Number of images to be skipped.

limit: int Number of images to be returned.

class `datarobot.models.visualai.DuplicateImage` (***kwargs*)
An image that was duplicated in the project dataset.

Attributes

image: Image Image object.

count: int Number of times the image was duplicated.

classmethod `list` (*project_id, feature_name, offset=None, limit=None*)
Get all duplicate images in a project.

Parameters

project_id: str Project that contains the images.

feature_name: str Name of feature column that contains images.

offset: int Number of images to be skipped.

limit: int Number of images to be returned.

class `datarobot.models.visualai.ImageEmbedding` (***kwargs*)
Vector representation of an image in an embedding space.

A vector in an embedding space will allow linear computations to be carried out between images: for example computing the Euclidean distance of the images.

Attributes

image: Image Image object used to create this map.

feature_name: str Name of the feature column this embedding is associated with.

position_x: int X coordinate of the image in the embedding space.

position_y: int Y coordinate of the image in the embedding space.

actual_target_value: object Actual target value of the dataset row.

classmethod `compute` (*project_id, model_id*)
Start creation of image embeddings for the model.

Parameters

project_id: str Project to start creation in.

model_id: str Project's model to start creation in.

Returns

str URL to check for image embeddings progress.

Raises

datarobot.errors.ClientError Server rejected creation due to client error. Most likely cause is bad `project_id` or `model_id`.

classmethod `models` (*project_id*)
List the models in a project.

Parameters

project_id: str Project that contains the models.

Returns

list(tuple(model_id, feature_name)) List of model and feature name pairs.

classmethod list (*project_id, model_id, feature_name*)

Return a list of ImageEmbedding objects.

Parameters

project_id: str Project that contains the images.

model_id: str Model that contains the images.

feature_name: str Name of feature column that contains images.

class datarobot.models.visualai.**ImageActivationMap** (***kwargs*)

Mark areas of image with weight of impact on training.

This is a technique to display how various areas of the region were used in training, and their effect on predictions. Larger values in `activation_values` indicates a larger impact.

Attributes

image: Image Image object used to create this map.

overlay_image: Image Image object composited with activation heat map.

feature_name: str Name of the feature column that contains the value this map is based on.

height: int Height of the original image in pixels.

width: int Width of the original image in pixels.

actual_target_value: object Actual target value of the dataset row.

predicted_target_value: object Predicted target value of the dataset row that contains this image.

activation_values: [[int]] A row-column matrix that contains the activation strengths for image regions. Values are integers in the range [0, 255].

classmethod compute (*project_id, model_id*)

Start creation of a activation map in the given model.

Parameters

project_id: str Project to start creation in.

model_id: str Project's model to start creation in.

Returns

str URL to check for image embeddings progress.

Raises

datarobot.errors.ClientError Server rejected creation due to client error. Most likely cause is bad `project_id` or `model_id`.

classmethod models (*project_id*)

List the models in a project.

Parameters

project_id: str Project that contains the models.

Returns

list(tuple(model_id, feature_name)) List of model and feature name pairs.

classmethod list (*project_id, model_id, feature_name, offset=None, limit=None*)

Return a list of ImageActivationMap objects.

Parameters

project_id: str Project that contains the images.

model_id: str Model that contains the images.

feature_name: str Name of feature column that contains images.

offset: int Number of images to be skipped.

limit: int Number of images to be returned.

32.34 Feature Association

class datarobot.models.feature_association.**FeatureAssociation** (*metric=None, assoc_type=None, featurelistId=None*)

Feature association statistics for a project.

Attributes

type [str] Either 'association' or 'correlation' the class of the pairwise stats

metric [str] the metric of either class of pairwise stats 'spearman', 'pearson', etc for correlation, 'mutualInfo', 'cramersV' for association

32.35 Feature Association Matrix Details

class datarobot.models.feature_association.**FeatureAssociationMatrixDetails** (*feature1=None, feature2=None*)

Plotting details for a pair of passed features present in the feature association matrix

Attributes

feature1 [str] Feature name for the first feature of interest

feature2 [str] Feature name for the second feature of interest

32.36 Feature Association Featurelists

class datarobot.models.feature_association.**FeatureAssociationFeaturelists**

Get project featurelists and see if they have association statistics

32.37 Rating Table

class datarobot.models.**RatingTable** (*id, rating_table_name, original_filename, project_id, parent_model_id, model_id=None, model_job_id=None, validation_job_id=None, validation_error=None*)

Interface to modify and download rating tables.

Attributes

id [str] The id of the rating table.

project_id [str] The id of the project this rating table belongs to.

rating_table_name [str] The name of the rating table.

original_filename [str] The name of the file used to create the rating table.

parent_model_id [str] The model id of the model the rating table was validated against.

model_id [str] The model id of the model that was created from the rating table. Can be None if a model has not been created from the rating table.

model_job_id [str] The id of the job to create a model from this rating table. Can be None if a model has not been created from the rating table.

validation_job_id [str] The id of the created job to validate the rating table. Can be None if the rating table has not been validated.

validation_error [str] Contains a description of any errors caused during validation.

classmethod **get** (*project_id, rating_table_id*)

Retrieve a single rating table

Parameters

project_id [str] The ID of the project the rating table is associated with.

rating_table_id [str] The ID of the rating table

Returns

rating_table [RatingTable] The queried instance

classmethod **create** (*project_id, parent_model_id, filename, rating_table_name='Uploaded Rating Table'*)

Uploads and validates a new rating table CSV

Parameters

project_id [str] id of the project the rating table belongs to

parent_model_id [str] id of the model for which this rating table should be validated against

filename [str] The path of the CSV file containing the modified rating table.

rating_table_name [str, optional] A human friendly name for the new rating table. The string may be truncated and a suffix may be added to maintain unique names of all rating tables.

Returns

job: Job an instance of created async job

Raises

InputNotUnderstoodError Raised if *filename* isn't one of supported types.

ClientError (400) Raised if *parent_model_id* is invalid.

download (*filepath*)

Download a csv file containing the contents of this rating table

Parameters

filepath [str] The path at which to save the rating table file.

rename (*rating_table_name*)

Renames a rating table to a different name.

Parameters

rating_table_name [str] The new name to rename the rating table to.

create_model ()

Creates a new model from this rating table record. This rating table must not already be associated with a model and must be valid.

Returns

job: Job an instance of created async job

Raises

ClientError (422) Raised if creating model from a RatingTable that failed validation

JobAlreadyRequested Raised if creating model from a RatingTable that is already associated with a RatingTableModel

32.38 Reason Codes (Deprecated)

This interface is considered deprecated. Please use *PredictionExplanations* instead.

class datarobot.**ReasonCodesInitialization** (*project_id*, *model_id*, *reason_codes_sample=None*)

Represents a reason codes initialization of a model.

Attributes

project_id [str] id of the project the model belongs to

model_id [str] id of the model reason codes initialization is for

reason_codes_sample [list of dict] a small sample of reason codes that could be generated for the model

classmethod **get** (*project_id*, *model_id*)

Retrieve the reason codes initialization for a model.

Reason codes initializations are a prerequisite for computing reason codes, and include a sample what the computed reason codes for a prediction dataset would look like.

Parameters

project_id [str] id of the project the model belongs to

model_id [str] id of the model reason codes initialization is for

Returns

reason_codes_initialization [ReasonCodesInitialization] The queried instance.

Raises

ClientError (404) If the project or model does not exist or the initialization has not been computed.

classmethod create (*project_id, model_id*)

Create a reason codes initialization for the specified model.

Parameters

project_id [str] id of the project the model belongs to

model_id [str] id of the model for which initialization is requested

Returns

job [Job] an instance of created async job

delete ()

Delete this reason codes initialization.

class datarobot.**ReasonCodes** (*id, project_id, model_id, dataset_id, max_codes, num_columns, finish_time, reason_codes_location, threshold_low=None, threshold_high=None*)

Represents reason codes metadata and provides access to computation results.

Examples

```
reason_codes = dr.ReasonCodes.get(project_id, reason_codes_id)
for row in reason_codes.get_rows():
    print(row)  # row is an instance of ReasonCodesRow
```

Attributes

id [str] id of the record and reason codes computation result

project_id [str] id of the project the model belongs to

model_id [str] id of the model reason codes initialization is for

dataset_id [str] id of the prediction dataset reason codes were computed for

max_codes [int] maximum number of reason codes to supply per row of the dataset

threshold_low [float] the lower threshold, below which a prediction must score in order for reason codes to be computed for a row in the dataset

threshold_high [float] the high threshold, above which a prediction must score in order for reason codes to be computed for a row in the dataset

num_columns [int] the number of columns reason codes were computed for

finish_time [float] timestamp referencing when computation for these reason codes finished

reason_codes_location [str] where to retrieve the reason codes

classmethod get (*project_id, reason_codes_id*)

Retrieve a specific reason codes.

Parameters

project_id [str] id of the project the model belongs to

reason_codes_id [str] id of the reason codes

Returns

reason_codes [ReasonCodes] The queried instance.

classmethod create (*project_id, model_id, dataset_id, max_codes=None, threshold_low=None, threshold_high=None*)

Create a reason codes for the specified dataset.

In order to create ReasonCodesPage for a particular model and dataset, you must first:

- Compute feature impact for the model via `datarobot.Model.get_feature_impact()`
- Compute a ReasonCodesInitialization for the model via `datarobot.ReasonCodesInitialization.create(project_id, model_id)`
- Compute predictions for the model and dataset via `datarobot.Model.request_predictions(dataset_id)`

`threshold_high` and `threshold_low` are optional filters applied to speed up computation. When at least one is specified, only the selected outlier rows will have reason codes computed. Rows are considered to be outliers if their predicted value (in case of regression projects) or probability of being the positive class (in case of classification projects) is less than `threshold_low` or greater than `thresholdHigh`. If neither is specified, reason codes will be computed for all rows.

Parameters

project_id [str] id of the project the model belongs to

model_id [str] id of the model for which reason codes are requested

dataset_id [str] id of the prediction dataset for which reason codes are requested

threshold_low [float, optional] the lower threshold, below which a prediction must score in order for reason codes to be computed for a row in the dataset. If neither `threshold_high` nor `threshold_low` is specified, reason codes will be computed for all rows.

threshold_high [float, optional] the high threshold, above which a prediction must score in order for reason codes to be computed. If neither `threshold_high` nor `threshold_low` is specified, reason codes will be computed for all rows.

max_codes [int, optional] the maximum number of reason codes to supply per row of the dataset, default: 3.

Returns

job: Job an instance of created async job

classmethod list (*project_id, model_id=None, limit=None, offset=None*)

List of reason codes for a specified project.

Parameters

project_id [str] id of the project to list reason codes for

model_id [str, optional] if specified, only reason codes computed for this model will be returned

limit [int or None] at most this many results are returned, default: no limit

offset [int or None] this many results will be skipped, default: 0

Returns

reason_codes [list[ReasonCodes]]

get_rows (*batch_size=None, exclude_adjusted_predictions=True*)

Retrieve reason codes rows.

Parameters

batch_size [int] maximum number of reason codes rows to retrieve per request

exclude_adjusted_predictions [bool] Optional, defaults to True. Set to False to include adjusted predictions, which will differ from the predictions on some projects, e.g. those with an exposure column specified.

Yields

reason_codes_row [ReasonCodesRow] Represents reason codes computed for a prediction row.

get_all_as_dataframe (*exclude_adjusted_predictions=True*)

Retrieve all reason codes rows and return them as a pandas.DataFrame.

Returned dataframe has the following structure:

- **row_id** : row id from prediction dataset
- **prediction** : the output of the model for this row
- **adjusted_prediction** : adjusted prediction values (only appears for projects that utilize prediction adjustments, e.g. projects with an exposure column)
- **class_0_label** : a class level from the target (only appears for classification projects)
- **class_0_probability** : the probability that the target is this class (only appears for classification projects)
- **class_1_label** : a class level from the target (only appears for classification projects)
- **class_1_probability** : the probability that the target is this class (only appears for classification projects)
- **reason_0_feature** : the name of the feature contributing to the prediction for this reason
- **reason_0_feature_value** : the value the feature took on
- **reason_0_label** : the output being driven by this reason. For regression projects, this is the name of the target feature. For classification projects, this is the class label whose probability increasing would correspond to a positive strength.
- **reason_0_qualitative_strength** : a human-readable description of how strongly the feature affected the prediction (e.g. `â+++â`, `âââ`, `âââ`) for this reason
- **reason_0_strength** : the amount this featureâs value affected the prediction
- `â`
- **reason_N_feature** : the name of the feature contributing to the prediction for this reason
- **reason_N_feature_value** : the value the feature took on
- **reason_N_label** : the output being driven by this reason. For regression projects, this is the name of the target feature. For classification projects, this is the class label whose probability increasing would correspond to a positive strength.
- **reason_N_qualitative_strength** : a human-readable description of how strongly the feature affected the prediction (e.g. `â+++â`, `âââ`, `âââ`) for this reason
- **reason_N_strength** : the amount this featureâs value affected the prediction

Parameters

exclude_adjusted_predictions [bool] Optional, defaults to True. Set this to False to include adjusted prediction values in the returned dataframe.

Returns

dataframe: `pandas.DataFrame`

download_to_csv (*filename, encoding='utf-8', exclude_adjusted_predictions=True*)

Save reason codes rows into CSV file.

Parameters

filename [str or file object] path or file object to save reason codes rows

encoding [string, optional] A string representing the encoding to use in the output file, defaults to `'utf-8'`

exclude_adjusted_predictions [bool] Optional, defaults to True. Set to False to include adjusted predictions, which will differ from the predictions on some projects, e.g. those with an exposure column specified.

get_reason_codes_page (*limit=None, offset=None, exclude_adjusted_predictions=True*)

Get reason codes.

If you don't want use a generator interface, you can access paginated reason codes directly.

Parameters

limit [int or None] the number of records to return, the server will use a (possibly finite) default if not specified

offset [int or None] the number of records to skip, default 0

exclude_adjusted_predictions [bool] Optional, defaults to True. Set to False to include adjusted predictions, which will differ from the predictions on some projects, e.g. those with an exposure column specified.

Returns

reason_codes [ReasonCodesPage]

delete ()

Delete this reason codes.

```
class datarobot.models.reason_codes.ReasonCodesRow (row_id, prediction, prediction_values, reason_codes=None, adjusted_prediction=None, adjusted_prediction_values=None)
```

Represents reason codes computed for a prediction row.

Notes

PredictionValue contains:

- **label** : describes what this model output corresponds to. For regression projects, it is the name of the target feature. For classification projects, it is a level from the target feature.
- **value** : the output of the prediction. For regression projects, it is the predicted value of the target. For classification projects, it is the predicted probability the row belongs to the class identified by the label.

ReasonCode contains:

- **label** : described what output was driven by this reason code. For regression projects, it is the name of the target feature. For classification projects, it is the class whose probability increasing would correspond to a positive strength of this reason code.
- **feature** : the name of the feature contributing to the prediction
- **feature_value** : the value the feature took on for this row
- **strength** : the amount this feature's value affected the prediction
- **qualitative_strength** : a human-readable description of how strongly the feature affected the prediction (e.g. "+++", "++", "+", "0", "-", "--", "---")

Attributes

row_id [int] which row this ReasonCodeRow describes

prediction [float] the output of the model for this row

adjusted_prediction [float or None] adjusted prediction value for projects that provide this information, None otherwise

prediction_values [list] an array of dictionaries with a schema described as PredictionValue

adjusted_prediction_values [list] same as prediction_values but for adjusted predictions

reason_codes [list] an array of dictionaries with a schema described as ReasonCode

```
class datarobot.models.reason_codes.ReasonCodesPage(id, count=None, previous=None,  
                                                    next=None, data=None, rea-  
                                                    son_codes_record_location=None,  
                                                    adjustment_method=None)
```

Represents batch of reason codes received by one request.

Attributes

id [str] id of the reason codes computation result

data [list[dict]] list of raw reason codes, each row corresponds to a row of the prediction dataset

count [int] total number of rows computed

previous_page [str] where to retrieve previous page of reason codes, None if current page is the first

next_page [str] where to retrieve next page of reason codes, None if current page is the last

reason_codes_record_location [str] where to retrieve the reason codes metadata

adjustment_method [str] Adjustment method that was applied to predictions, or "N/A" if no adjustments were done.

```
classmethod get(project_id, reason_codes_id, limit=None, offset=0, ex-  
                clude_adjusted_predictions=True)
```

Retrieve reason codes.

Parameters

project_id [str] id of the project the model belongs to

reason_codes_id [str] id of the reason codes

limit [int or None] the number of records to return, the server will use a (possibly finite) default if not specified

offset [int or None] the number of records to skip, default 0

exclude_adjusted_predictions [bool] Optional, defaults to True. Set to False to include adjusted predictions, which will differ from the predictions on some projects, e.g. those with an exposure column specified.

Returns

reason_codes [ReasonCodesPage] The queried instance.

32.39 Recommended Models

class `datarobot.models.ModelRecommendation` (*project_id*, *model_id*, *recommendation_type*)

A collection of information about a recommended model for a project.

Attributes

project_id [str] the id of the project the model belongs to

model_id [str] the id of the recommended model

recommendation_type [str] the type of model recommendation

classmethod `get` (*project_id*, *recommendation_type=None*)

Retrieves the default or specified by *recommendation_type* recommendation.

Parameters

project_id [str] The project's id.

recommendation_type [enums.RECOMMENDED_MODEL_TYPE] The type of recommendation to get. If None, returns the default recommendation.

Returns

recommended_model [ModelRecommendation]

classmethod `get_all` (*project_id*)

Retrieves all of the current recommended models for the project.

Parameters

project_id [str] The project's id.

Returns

recommended_models [list of ModelRecommendation]

classmethod `get_recommendation` (*recommended_models*, *recommendation_type*)

Returns the model in the given list with the requested type.

Parameters

recommended_models [list of ModelRecommendation]

recommendation_type [enums.RECOMMENDED_MODEL_TYPE] the type of model to extract from the *recommended_models* list

Returns

recommended_model [ModelRecommendation or None if no model with the requested type exists]

get_model ()

Returns the Model associated with this ModelRecommendation.

Returns

recommended_model [Model]

32.40 ROC Curve

class datarobot.models.roc_curve.**RocCurve** (*source, roc_points, negative_class_predictions, positive_class_predictions, source_model_id*)

ROC curve data for model.

Attributes

source [str] ROC curve data source. Can be `validation`, `crossValidation` or `holdout`.

roc_points [list of dict] List of precalculated metrics associated with thresholds for ROC curve.

negative_class_predictions [list of float] List of predictions from example for negative class

positive_class_predictions [list of float] List of predictions from example for positive class

source_model_id [str] ID of the model this ROC curve represents; in some cases, insights from the parent of a frozen model may be used

32.41 SharingAccess

class datarobot.**SharingAccess** (*username, role, can_share=None, user_id=None*)

Represents metadata about whom a entity (e.g. a data store) has been shared with

New in version v2.14.

Currently *DataStores*, *DataSources*, *Projects* (new in version v2.15) and *CalendarFiles* (new in version 2.15) can be shared.

This class can represent either access that has already been granted, or be used to grant access to additional users.

Attributes

username [str] a particular user

role [str or None] if a string, represents a particular level of access and should be one of `datarobot.enums.SHARING_ROLE`. For more information on the specific access levels, see the *sharing* documentation. If None, can be passed to a *share* function to revoke access for a specific user.

can_share [bool or None] if a bool, indicates whether this user is permitted to further share. When False, the user has access to the entity, but can only revoke their own access but not modify any user's access role. When True, the user can share with any other user at a access role up to their own. May be None if the SharingAccess was not retrieved from the DataRobot server but intended to be passed into a *share* function; this will be equivalent to passing True.

user_id [str] the id of the user

32.42 Training Predictions

class datarobot.models.training_predictions.**TrainingPredictionsIterator** (*client*,
path,
limit=None)

Lazily fetches training predictions from DataRobot API in chunks of specified size and then iterates rows from responses as named tuples. Each row represents a training prediction computed for a dataset's row. Each named tuple has the following structure:

Notes

Each `PredictionValue` dict contains these keys:

label describes what this model output corresponds to. For regression projects, it is the name of the target feature. For classification and multiclass projects, it is a label from the target feature.

value the output of the prediction. For regression projects, it is the predicted value of the target. For classification and multiclass projects, it is the predicted probability that the row belongs to the class identified by the label.

Each `PredictionExplanations` dictionary contains these keys:

label [string] describes what output was driven by this prediction explanation. For regression projects, it is the name of the target feature. For classification projects, it is the class whose probability increasing would correspond to a positive strength of this prediction explanation.

feature [string] the name of the feature contributing to the prediction

feature_value [object] the value the feature took on for this row. The type corresponds to the feature (boolean, integer, number, string)

strength [float] algorithm-specific explanation value attributed to feature in this row

`ShapMetadata` dictionary contains these keys:

shap_remaining_total [float] The total of SHAP values for features beyond the `max_explanations`. This can be identically 0 in all rows, if `max_explanations` is greater than the number of features and thus all features are returned.

shap_base_value [float] the model's average prediction over the training data. SHAP values are deviations from the base value.

warnings [dict or None] SHAP values calculation warnings (e.g. additivity check failures in XGBoost models). Schema described as `ShapWarnings`.

`ShapWarnings` dictionary contains these keys:

mismatch_row_count [int] the count of rows for which additivity check failed

max_normalized_mismatch [float] the maximal relative normalized mismatch value

Examples

```
import datarobot as dr

# Fetch existing training predictions by their id
training_predictions = dr.TrainingPredictions.get(project_id, prediction_id)
```

(continues on next page)

(continued from previous page)

```
# Iterate over predictions
for row in training_predictions.iterate_rows():
    print(row.row_id, row.prediction)
```

Attributes

row_id [int] id of the record in original dataset for which training prediction is calculated

partition_id [str or float] id of the data partition that the row belongs to

prediction [float] the model's prediction for this data row

prediction_values [list of dictionaries] an array of dictionaries with a schema described as `PredictionValue`

timestamp [str or None] (New in version v2.11) an ISO string representing the time of the prediction in time series project; may be None for non-time series projects

forecast_point [str or None] (New in version v2.11) an ISO string representing the point in time used as a basis to generate the predictions in time series project; may be None for non-time series projects

forecast_distance [str or None] (New in version v2.11) how many time steps are between the forecast point and the timestamp in time series project; None for non-time series projects

series_id [str or None] (New in version v2.11) the id of the series in a multiseries project; may be NaN for single series projects; None for non-time series projects

prediction_explanations [list of dict or None] (New in version v2.21) The prediction explanations for each feature. The total elements in the array are bounded by `max_explanations` and feature count. Only present if prediction explanations were requested. Schema described as `PredictionExplanations`.

shap_metadata [dict or None] (New in version v2.21) The additional information necessary to understand SHAP based prediction explanations. Only present if `explanation_algorithm` equals `datarobot.enums.EXPLANATIONS_ALGORITHM.SHAP` was added in compute request. Schema described as `ShapMetadata`.

```
class datarobot.models.training_predictions.TrainingPredictions(project_id,
                                                                prediction_id,
                                                                model_id=None,
                                                                data_subset=None,
                                                                explanation_algorithm=None,
                                                                max_explanations=None,
                                                                shap_warnings=None)
```

Represents training predictions metadata and provides access to prediction results.

Notes

Each element in `shap_warnings` has the following schema:

partition_name [str] the partition used for the prediction record.

value [object] the warnings related to this partition.

The objects in `value` are:

mismatch_row_count [int] the count of rows for which additivity check failed.

max_normalized_mismatch [float] the maximal relative normalized mismatch value.

Examples

Compute training predictions for a model on the whole dataset

```
import datarobot as dr

# Request calculation of training predictions
training_predictions_job = model.request_training_predictions(dr.enums.DATA_
↳SUBSET.ALL)
training_predictions = training_predictions_job.get_result_when_complete()
print('Training predictions {} are ready'.format(training_predictions.prediction_
↳id))

# Iterate over actual predictions
for row in training_predictions.iterate_rows():
    print(row.row_id, row.partition_id, row.prediction)
```

List all training predictions for a project

```
import datarobot as dr

# Fetch all training predictions for a project
all_training_predictions = dr.TrainingPredictions.list(project_id)

# Inspect all calculated training predictions
for training_predictions in all_training_predictions:
    print(
        'Prediction {} is made for data subset "{}"'.format(
            training_predictions.prediction_id,
            training_predictions.data_subset,
        )
    )
```

Retrieve training predictions by id

```
import datarobot as dr

# Getting training predictions by id
training_predictions = dr.TrainingPredictions.get(project_id, prediction_id)

# Iterate over actual predictions
for row in training_predictions.iterate_rows():
    print(row.row_id, row.partition_id, row.prediction)
```

Attributes

project_id [str] id of the project the model belongs to

model_id [str] id of the model

prediction_id [str] id of generated predictions

data_subset [datarobot.enums.DATA_SUBSET] data set definition used to build predictions.
Choices are:

- **datarobot.enums.DATA_SUBSET.ALL** for all data available. Not valid for models in datetime partitioned projects.

- `datarobot.enums.DATA_SUBSET.VALIDATION_AND_HOLDOUT` for all data except training set. Not valid for models in datetime partitioned projects.
- `datarobot.enums.DATA_SUBSET.HOLDOUT` for holdout data set only.
- `datarobot.enums.DATA_SUBSET.ALL_BACKTESTS` for downloading the predictions for all backtest validation folds. Requires the model to have successfully scored all backtests. Datetime partitioned projects only.

explanation_algorithm [`datarobot.enums.EXPLANATIONS_ALGORITHM`] (New in version v2.21) Optional. If set to `shap`, the response will include prediction explanations based on the SHAP explainer (SHapley Additive exPlanations). Defaults to `null` (no prediction explanations).

max_explanations [`int`] (New in version v2.21) The number of top contributors that are included in prediction explanations. Max 100. Defaults to `null` for datasets narrower than 100 columns, defaults to 100 for datasets wider than 100 columns.

shap_warnings [`list`] (New in version v2.21) Will be present if `explanation_algorithm` was set to `datarobot.enums.EXPLANATIONS_ALGORITHM.SHAP` and there were additivity failures during SHAP values calculation.

classmethod `list` (*project_id*)

Fetch all the computed training predictions for a project.

Parameters

project_id [`str`] id of the project

Returns

A list of *TrainingPredictions* objects

classmethod `get` (*project_id*, *prediction_id*)

Retrieve training predictions on a specified data set.

Parameters

project_id [`str`] id of the project the model belongs to

prediction_id [`str`] id of the prediction set

Returns

TrainingPredictions object which is ready to operate with specified predictions

iterate_rows (*batch_size=None*)

Retrieve training prediction rows as an iterator.

Parameters

batch_size [`int`, optional] maximum number of training prediction rows to fetch per request

Returns

iterator [*TrainingPredictionsIterator*] an iterator which yields named tuples representing training prediction rows

get_all_as_dataframe (*class_prefix='class_', serializer='json'*)

Retrieve all training prediction rows and return them as a `pandas.DataFrame`.

Returned dataframe has the following structure:

- `row_id` : row id from the original dataset
- `prediction` : the model's prediction for this row

- `class_<label>` : the probability that the target is this class (only appears for classification and multiclass projects)
- `timestamp` : the time of the prediction (only appears for out of time validation or time series projects)
- `forecast_point` : the point in time used as a basis to generate the predictions (only appears for time series projects)
- `forecast_distance` : how many time steps are between `timestamp` and `forecast_point` (only appears for time series projects)
- `series_id` : the id of the series in a multiseries project or `None` for a single series project (only appears for time series projects)

Parameters

class_prefix [str, optional] The prefix to append to labels in the final dataframe. Default is `class_` (e.g., `apple` -> `class_apple`)

serializer [str, optional] Serializer to use for the download. Options: `json` (default) or `csv`.

Returns

dataframe: `pandas.DataFrame`

download_to_csv (*filename*, *encoding='utf-8'*, *serializer='json'*)

Save training prediction rows into CSV file.

Parameters

filename [str or file object] path or file object to save training prediction rows

encoding [string, optional] A string representing the encoding to use in the output file, defaults to `'utf-8'`

serializer [str, optional] Serializer to use for the download. Options: `json` (default) or `csv`.

32.43 Word Cloud

class `datarobot.models.word_cloud.WordCloud` (*ngrams*)

Word cloud data for the model.

Notes

`WordCloudNgram` is a dict containing the following:

- `ngram` (str) Word or ngram value.
- `coefficient` (float) Value from `[-1.0, 1.0]` range, describes effect of this ngram on the target. Large negative value means strong effect toward negative class in classification and smaller target value in regression models. Large positive - toward positive class and bigger value respectively.
- `count` (int) Number of rows in the training sample where this ngram appears.
- `frequency` (float) Value from `(0.0, 1.0]` range, relative frequency of given ngram to most frequent ngram.
- `is_stopword` (bool) True for ngrams that DataRobot evaluates as stopwords.

- `class` (str or None) For classification - values of the target class for corresponding word or ngram. For regression - None.

Attributes

ngrams [list of dicts] List of dicts with schema described as `WordCloudNgram` above.

most_frequent (*top_n=5*)

Return most frequent ngrams in the word cloud.

Parameters

top_n [int] Number of ngrams to return

Returns

list of dict Up to `top_n` top most frequent ngrams in the word cloud. If `top_n` bigger then total number of ngrams in word cloud - return all sorted by frequency in descending order.

most_important (*top_n=5*)

Return most important ngrams in the word cloud.

Parameters

top_n [int] Number of ngrams to return

Returns

list of dict Up to `top_n` top most important ngrams in the word cloud. If `top_n` bigger then total number of ngrams in word cloud - return all sorted by absolute coefficient value in descending order.

ngrams_per_class ()

Split ngrams per target class values. Useful for multiclass models.

Returns

dict Dictionary in the format of (class label) -> (list of ngrams for that class)

32.44 Feature Discovery / Safer

class `datarobot.models.SecondaryDatasetConfigurations` (*id=None, project_id=None, config=None*)

Create secondary dataset configurations for a given project

New in version v2.20.

Attributes

id [str] id of this secondary dataset configuration

project_id [str] id of the associated project.

config: list of DatasetConfiguration list of secondary dataset configurations

classmethod create (*project_id, dataset_configurations*)

create secondary dataset configurations

New in version v2.20.

Parameters

project_id [str] id of the associated project.

dataset_configurations: list of **DatasetConfiguration** list of dataset configurations

Returns

an instance of **SecondaryDatasetConfigurations**

Raises

ClientError raised if incorrect configuration parameters are provided

class `datarobot.models.RelationshipsConfiguration` (*id*, *dataset_definitions=None*, *relationships=None*)

A Relationships configuration specifies a set of secondary datasets as well as the relationships among them. It is used to configure Feature Discovery for a project to generate features automatically from these datasets.

Attributes

id [str] the id of the created relationships configuration

dataset_definitions: list each element is a dataset_definitions for a dataset.

relationships: list each element is a relationship between two datasets

The ‘dataset_definitions’ structure is

identifier: str alias of the dataset (used directly as part of the generated feature names)

catalog_id: str, or None identifier of the catalog item

catalog_version_id: str identifier of the catalog item version

primary_temporal_key: str, or None name of the column indicating time of record creation

feature_list_id: str, or None identifier of the feature list. This decides which columns in the dataset are used for feature generation

snapshot_policy: str policy to use when creating a project or making predictions. Must be one of the following values: ‘specified’: Use specific snapshot specified by catalogVersionId ‘latest’: Use latest snapshot from the same catalog item ‘dynamic’: Get data from the source (only applicable for JDBC datasets)

feature_lists: list list of feature list info

data_source: dict data source info if the dataset is from data source

is_deleted: bool or None whether the dataset is deleted or not

The ‘data source info’ structured is

data_store_id: str the id of the data store.

data_store_name [str] the user-friendly name of the data store.

url [str] the url used to connect to the data store.

dbtable [str] the name of table from the data store.

schema: str schema definition of the table from the data store

The ‘feature list info’ structure is

id [str] the id of the featurelist

name [str] the name of the featurelist

features [list of str] the names of all the Features in the featurelist

dataset_id [str] the project the featurelist belongs to

creation_date [datetime.datetime] when the featurelist was created

user_created [bool] whether the featurelist was created by a user or by DataRobot automation

created_by: **str** the name of user who created it

description [str] the description of the featurelist. Can be updated by the user and may be supplied by default for DataRobot-created featurelists.

dataset_id: **str** dataset which is associated with the feature list

dataset_version_id: **str or None** version of the dataset which is associated with feature list. Only relevant for Informative features

The ‘relationships’ schema is

dataset1_identifier: **str or None** identifier of the first dataset in this relationship. This is specified in the identifier field of dataset_definition structure. If None, then the relationship is with the primary dataset.

dataset2_identifier: **str** identifier of the second dataset in this relationship. This is specified in the identifier field of dataset_definition schema.

dataset1_keys: **list of str (max length: 10 min length: 1)** column(s) from the first dataset which are used to join to the second dataset

dataset2_keys: **list of str (max length: 10 min length: 1)** column(s) from the second dataset that are used to join to the first dataset

time_unit: **str, or None** time unit of the feature derivation window. Supported values are MILLISECOND, SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER, YEAR. If present, the feature engineering Graph will perform time-aware joins.

feature_derivation_window_start: **int, or None** how many time_units of each dataset’s primary temporal key into the past relative to the datetimePartitionColumn the feature derivation window should begin. Will be a negative integer, If present, the feature engineering Graph will perform time-aware joins.

feature_derivation_window_end: **int, or None** how many timeUnits of each dataset’s record primary temporal key into the past relative to the datetimePartitionColumn the feature derivation window should end. Will be a non-positive integer, if present. If present, the feature engineering Graph will perform time-aware joins.

feature_derivation_window_time_unit: **int or None** time unit of the feature derivation window. Supported values are MILLISECOND, SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER, YEAR If present, time-aware joins will be used. Only applicable when dataset1Identifier is not provided.

prediction_point_rounding: **int, or None** closest value of prediction_point_rounding_time_unit to round the prediction point into the past when applying the feature derivation window. Will be a positive integer, if present. Only applicable when dataset1_identifier is not provided.

prediction_point_rounding_time_unit: **str, or None** time unit of the prediction point rounding. Supported values are MILLISECOND, SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER, YEAR Only applicable when dataset1_identifier is not provided.

classmethod create (*dataset_definitions, relationships*)

Create a Relationships Configuration

Parameters

dataset_definitions: **list of dict** each element is a DatasetDefinition . The *DatasetDefinition* schema is

identifier: **str** alias of the table (used directly as part of the generated feature names)

catalog_id: **str, or None** identifier of the catalog item

catalog_version_id: **str** identifier of the catalog item version

feature_list_id: **str, or None** identifier of the feature list. This decides which columns in the table are used for feature generation

snapshot_policy: **str** policy to use when creating a project or making predictions. Must be one of the following values: `specified`: Use specific snapshot specified by `catalogVersionId` `latest`: Use latest snapshot from the same catalog item `dynamic`: Get data from the source (only applicable for JDBC datasets)

relationships: **list of dict** each element is a Relationship between two datasets The *Relationship* schema is

dataset1_identifier: **str or None** identifier of the first dataset in this relationship. This is specified in the `identifier` field of `dataset_definition` structure. If `None`, then the relationship is with the primary dataset.

dataset2_identifier: **str** identifier of the second dataset in this relationship. This is specified in the `identifier` field of `dataset_definition` schema.

dataset1_keys: **list of str (max length: 10 min length: 1)** column(s) from the first dataset which are used to join to the second dataset

dataset2_keys: **list of str (max length: 10 min length: 1)** column(s) from the second dataset that are used to join to the first dataset

time_unit: **str, or None** time unit of the feature derivation window. Supported values are `MILLISECOND`, `SECOND`, `MINUTE`, `HOURL`, `DAY`, `WEEK`, `MONTH`, `QUARTER`, `YEAR`. If present, the feature engineering Graph will perform time-aware joins.

feature_derivation_window_start: **int, or None** how many `time_units` of each dataset's primary temporal key into the past relative to the `datetimePartitionColumn` the feature derivation window should begin. Will be a negative integer, If present, the feature engineering Graph will perform time-aware joins.

feature_derivation_window_end: **int, or None** how many `timeUnits` of each dataset's record primary temporal key into the past relative to the `datetimePartitionColumn` the feature derivation window should end. Will be a non-positive integer, if present. If present, the feature engineering Graph will perform time-aware joins.

feature_derivation_window_time_unit: **int or None** time unit of the feature derivation window. Supported values are `MILLISECOND`, `SECOND`, `MINUTE`, `HOURL`, `DAY`, `WEEK`, `MONTH`, `QUARTER`, `YEAR` If present, time-aware joins will be used. Only applicable when `dataset1Identifier` is not provided.

prediction_point_rounding: **int, or None** closest value of `prediction_point_rounding_time_unit` to round the prediction point into the past when applying the feature derivation window. Will be a positive integer, if present. Only applicable when `dataset1_identifier` is not provided.

prediction_point_rounding_time_unit: **str, or None** time unit of the prediction point rounding. Supported values are `MILLISECOND`, `SECOND`, `MINUTE`, `HOURL`, `DAY`, `WEEK`, `MONTH`, `QUARTER`, `YEAR` Only applicable when `dataset1_identifier` is not provided.

Returns

relationships_configuration: RelationshipsConfiguration the created relationships configuration

get()

Retrieve the Relationships configuration for a given id

Returns

relationships_configuration: RelationshipsConfiguration The requested relationships configuration

Raises

ClientError Raised if an invalid relationships config id is provided.

Examples

```
relationships_config = dr.RelationshipsConfiguration(valid_config_id)
result = relationships_config.get()
>>> result.id
'5c88a37770fc42a2fcc62759'
```

replace(dataset_definitions, relationships)

Update the Relationships Configuration which is not used in the feature discovery Project

Parameters

dataset_definitions: list of dict each element is a DatasetDefinition . The *DatasetDefinition* schema is

identifier: str alias of the table (used directly as part of the generated feature names)

catalog_id: str, or None identifier of the catalog item

catalog_version_id: str identifier of the catalog item version

feature_list_id: str, or None identifier of the feature list. This decides which columns in the table are used for feature generation

snapshot_policy: str policy to use when creating a project or making predictions. Must be one of the following values: *specified*: Use specific snapshot specified by catalogVersionId *latest*: Use latest snapshot from the same catalog item *dynamic*: Get data from the source (only applicable for JDBC datasets)

relationships: list of dict each element is a Relationship between two datasets The *Relationship* schema is

dataset1_identifier: str or None identifier of the first dataset in this relationship. This is specified in the identifier field of dataset_definition structure. If None, then the relationship is with the primary dataset.

dataset2_identifier: str identifier of the second dataset in this relationship. This is specified in the identifier field of dataset_definition schema.

dataset1_keys: list of str (max length: 10 min length: 1) column(s) from the first dataset which are used to join to the second dataset

dataset2_keys: list of str (max length: 10 min length: 1) column(s) from the second dataset that are used to join to the first dataset

time_unit: str, or None time unit of the feature derivation window. Supported values are `MILLISECOND`, `SECOND`, `MINUTE`, `HOURL`, `DAY`, `WEEK`, `MONTH`, `QUARTER`, `YEAR`. If present, the feature engineering Graph will perform time-aware joins.

feature_derivation_window_start: int, or None how many time_units of each dataset's primary temporal key into the past relative to the `datetimePartitionColumn` the feature derivation window should begin. Will be a negative integer, If present, the feature engineering Graph will perform time-aware joins.

feature_derivation_window_end: int, or None how many timeUnits of each dataset's record primary temporal key into the past relative to the `datetimePartitionColumn` the feature derivation window should end. Will be a non-positive integer, if present. If present, the feature engineering Graph will perform time-aware joins.

feature_derivation_window_time_unit: int or None time unit of the feature derivation window. Supported values are `MILLISECOND`, `SECOND`, `MINUTE`, `HOURL`, `DAY`, `WEEK`, `MONTH`, `QUARTER`, `YEAR` If present, time-aware joins will be used. Only applicable when `dataset1Identifier` is not provided.

prediction_point_rounding: int, or None closest value of `prediction_point_rounding_time_unit` to round the prediction point into the past when applying the feature derivation window. Will be a positive integer, if present. Only applicable when `dataset1_identifier` is not provided.

prediction_point_rounding_time_unit: str, or None time unit of the prediction point rounding. Supported values are `MILLISECOND`, `SECOND`, `MINUTE`, `HOURL`, `DAY`, `WEEK`, `MONTH`, `QUARTER`, `YEAR` Only applicable when `dataset1_identifier` is not provided.

Returns

relationships_configuration: RelationshipsConfiguration the updated relationships configuration

`delete()`

Delete the Relationships configuration

Raises

ClientError Raised if an invalid relationships config id is provided.

Examples

```
# Deleting with a valid id
relationships_config = dr.RelationshipsConfiguration(valid_config_id)
status_code = relationships_config.delete()
status_code
>>> 204
relationships_config.get()
>>> ClientError: Relationships Configuration not found
```

32.45 SHAP

class `datarobot.models.ShapImpact` (*count, shap_impacts*)

Represents SHAP impact score for a feature in a model.

New in version v2.21.

Notes

SHAP impact score for a feature has the following structure:

- `feature_name`: (str) the feature name in dataset
- `impact_normalized`: (float) normalized impact score value (largest value is 1)
- `impact_unnormalized`: (float) raw impact score value

Attributes

count [int] the number of SHAP Impact object returned

shap_impacts [list] a list which contains SHAP impact scores for top 1000 features used by a model

classmethod **create** (*project_id, model_id*)

Create SHAP impact for the specified model.

Parameters

project_id [str] id of the project the model belongs to

model_id [str] id of the model to calculate shap impact for

Returns

job [Job] an instance of created async job

classmethod **get** (*project_id, model_id*)

Retrieve SHAP impact scores for features in a model.

Parameters

project_id [str] id of the project the model belongs to

model_id [str] id of the model the SHAP impact is for

Returns

shap_impact [ShapImpact] The queried instance.

Raises

ClientError (404) If the project or model does not exist or the SHAP impact has not been computed.

CHAPTER 33

Examples

The examples are only available in the html build of the documentation

34.1 2.22.2

34.1.1 Bugfixes

fixed several instances of client code failing with a newer server version

34.2 2.22.1

34.2.1 New Features

- Batch Prediction jobs now support *dataset* as intake settings for *BatchPredictionJob.score*.
- Create a Dataset from DataSource:
 - *Dataset.create_from_data_source*
 - *DataSource.create_dataset*
- Added support for Custom Model Dependency Management. Please see *custom model documentation*. New features added:
 - Added new argument *base_environment_id* to methods *CustomModelVersion.create_clean* and *CustomModelVersion.create_from_previous*
 - New fields *base_environment_id* and *dependencies* to class *datarobot.CustomModelVersion*
 - New class *datarobot.CustomModelVersionDependencyBuild* to prepare custom model versions with dependencies.
 - Made argument *environment_id* of *CustomModelTest.create* optional to enable using custom model versions with dependencies

- New field `image_type` added to class `datarobot.CustomModelTest`
- `Deployment.create_from_custom_model_version` can be used to create a deployment from a custom model version.
- Added new parameters for starting and re-running Autopilot with customizable settings within `Project.start_autopilot`.
- Added a new method to trigger Feature Impact calculation for a Custom Inference Image: `CustomInferenceImage.calculate_feature_impact`
- Added new method to retrieve number of iterations trained for early stopping models. Currently supports only tree-based models. `Model.get_num_iterations_trained`.

34.2.2 Enhancements

- A description can now be added or updated for a project. `Project.set_project_description`.
- Added new parameters `read_timeout` and `max_wait` to method `Dataset.create_from_file`. Values larger than the default can be specified for both to avoid timeouts when uploading large files.
- Added new parameter `metric` to `datarobot.models.TargetDrift`, `datarobot.models.FeatureDrift`, `Deployment.get_target_drift` and `Deployment.get_feature_drift`.
- Added new parameter `timeout` to `BatchPredictionJob.download` to indicate how many seconds to wait for the download to start (in case the job doesn't start processing immediately). Set to `-1` to disable. This parameter can also be sent as `download_timeout` to `BatchPredictionJob.score` and `BatchPredictionJob.score`. If the timeout occurs, the pending job will be aborted.
- Added new parameter `read_timeout` to `BatchPredictionJob.download` to indicate how many seconds to wait between each downloaded chunk. This parameter can also be sent as `download_read_timeout` to `BatchPredictionJob.score` and `BatchPredictionJob.score`.
- Added parameter `catalog` to `BatchPredictionJob` to both intake and output adapters for type `jdbc`.
- Consider blenders in recommendation can now be specified in `AdvancedOptions`. Blenders will be included when autopilot chooses a model to prepare and recommend for deployment.
- Added optional parameter `max_wait` to `Deployment.replace_model` to indicate the maximum time to wait for model replacement job to complete before erroring.

34.2.3 Bugfixes

- Handle null values in `predictionExplanationMetadata["shapRemainingTotal"]` while converting a predictions response to a data frame.
- Handle null values in `customModel["latestVersion"]`
- Removed an extra column `status` from `BatchPredictionJob` as it caused issues with never version of Trafaret validation.
- Make `predicted_vs_actual` optional in Feature Effects data because a feature may have insufficient qualified samples.
- Make `jdbc_url` optional in Data Store data because some data stores will not have it.
- The method `Project.get_datetime_models` now correctly returns all `DatetimeModel` objects for the project, instead of just the first 100.
- Fixed a documentation error related to `snake_case` vs `camelCase` in the JDBC settings payload.

- Make trafaret validator for datasets use a syntax that works properly with a wider range of trafaret versions.
- Handle extra keys in CustomModelTests and CustomModelVersions
- ImageEmbedding and ImageActivationMap now supports regression projects.

34.2.4 API Changes

- The default value for the mode param in `Project.set_target` has been changed from `AUTOPILOT_MODE.FULL_AUTO` to `AUTOPILOT_MODE.QUICK`

34.2.5 Deprecation Summary

34.2.6 Configuration Changes

34.2.7 Documentation Changes

- Added links to classes with duration parameters such as `validation_duration` and `holdout_duration` to provide duration string examples to users.
- The *models documentation* has been revised to include section on how to train a new model and how to run cross-validation or backtesting for a model.

34.3 2.21.0

34.3.1 New Features

- Added new arguments `explanation_algorithm` and `max_explanations` to method `Model.request_training_predictions`. New fields `explanation_algorithm`, `max_explanations` and `shap_warnings` have been added to class `TrainingPredictions`. New fields `prediction_explanations` and `shap_metadata` have been added to class `TrainingPredictionsIterator` that is returned by method `TrainingPredictions.iterate_rows`.
- Added new arguments `explanation_algorithm` and `max_explanations` to method `Model.request_predictions`. New fields `explanation_algorithm`, `max_explanations` and `shap_warnings` have been added to class `Predictions`. Method `Predictions.get_all_as_dataframe` has new argument `serializer` that specifies the retrieval and results validation method (`json` or `csv`) for the predictions.
- Added possibility to compute `ShapImpact.create` and request `ShapImpact.get` SHAP impact scores for features in a model.
- Added support for accessing Visual AI images and insights. See the DataRobot Python Package documentation, Visual AI Projects, section for details.
- User can specify custom row count when requesting Feature Effects. Extended methods are `Model.request_feature_effect` and `Model.get_or_request_feature_effect`.
- Users can request SHAP based predictions explanations for a models that support SHAP scores using `ShapMatrix.create`.
- Added two new methods to `Dataset` to lazily retrieve paginated responses.
 - `Dataset.iterate` returns an iterator of the datasets that a user can view.

- `Dataset.iterate_all_features` returns an iterator of the features of a dataset.
- It's possible to create an Interaction feature by combining two categorical features together using `Project.create_interaction_feature`. Operation result represented by `models.InteractionFeature`. Specific information about an interaction feature may be retrieved by its name using `models.InteractionFeature.get`
- Added the `DatasetFeaturelist` class to support featurelists on datasets in the AI Catalog. `DatasetFeaturelists` can be updated or deleted. Two new methods were also added to `Dataset` to interact with `DatasetFeaturelists`. These are `Dataset.get_featurelists` and `Dataset.create_featurelist` which list existing featurelists and create new featurelists on a dataset, respectively.
- Added `model_splits` to `DatetimePartitioningSpecification` and to `DatetimePartitioning`. This will allow users to control the jobs per model used when building models. A higher number of `model_splits` will result in less downsampling, allowing the use of more post-processed data.
- Added support for *unsupervised projects*.
- Added support for external test set. Please see *testset documentation*
- A new workflow is available for assessing models on external test sets in time series unsupervised projects. More information can be found in the *documentation*.
 - `Project.upload_dataset` and `Model.request_predictions` now accept `actual_value_column` - name of the actual value column, can be passed only with date range.
 - `PredictionDataset` objects now contain the following new fields:
 - * `actual_value_column`: Actual value column which was selected for this dataset.
 - * `detected_actual_value_column`: A list of detected actual value column info.
 - New warning is added to `data_quality_warnings` of `datarobot.models.PredictionDataset`: `single_class_actual_value_column`.
 - Scores and insights on external test sets can be retrieved using `ExternalScores`, `ExternalLiftChart`, `ExternalRocCurve`.
- Users can create payoff matrices for generating profit curves for binary classification projects using `PayoffMatrix.create`.
- Deployment Improvements:
 - `datarobot.models.TargetDrift` can be used to retrieve target drift information.
 - `datarobot.models.FeatureDrift` can be used to retrieve feature drift information.
 - `Deployment.submit_actuals` will submit actuals in batches if the total number of actuals exceeds the limit of one single request.
 - `Deployment.create_from_custom_model_image` can be used to create a deployment from a custom model image.
 - Deployments now support predictions data collection that enables prediction requests and results to be saved in Predictions Data Storage. See `Deployment.get_predictions_data_collection_settings` and `Deployment.update_predictions_data_collection_settings` for usage.
- New arguments `send_notification` and `include_feature_discovery_entities` are added to `Project.share`.

- Now it is possible to specify the number of training rows to use in feature impact computation on supported project types (that is everything except unsupervised, multi-class, time-series). This does not affect SHAP based feature impact. Extended methods:

- `Model.request_feature_impact`
- `Model.get_or_request_feature_impact`

- A new class `FeatureImpactJob` is added to retrieve Feature Impact records with metadata. The regular `Job` still works as before.

- Added support for custom models. Please see [custom model documentation](#). Classes added:

- `datarobot.ExecutionEnvironment` and `datarobot.ExecutionEnvironmentVersion` to create and manage custom model executions environments
- `datarobot.CustomInferenceModel` and `datarobot.CustomModelVersion` to create and manage custom inference models
- `datarobot.CustomModelTest` to perform testing of custom models

- Batch Prediction jobs now support forecast and historical Time Series predictions using the new argument `timeseries_settings` for `BatchPredictionJob.score`.

- Batch Prediction jobs now support scoring to Azure and Google Cloud Storage with methods `BatchPredictionJob.score_azure` and `BatchPredictionJob.score_gcp`.

- **Now it's possible to create Relationships Configurations to introduce secondary datasets to projects. A configuration s**

- `RelationshipsConfiguration.create` creates a new relationships configuration between datasets
- `RelationshipsConfiguration.retrieve` retrieve the requested relationships configuration
- `RelationshipsConfiguration.replace` replace the relationships configuration details with new one
- `RelationshipsConfiguration.delete` delete the relationships configuration

34.3.2 Enhancements

- Made creating projects from a dataset easier through the new `Dataset.create_project`.
- These methods now provide additional metadata fields in Feature Impact results if called with `with_metadata=True`. Fields added: `rowCount`, `shapBased`, `ranRedundancyDetection`, `count`.

- `Model.get_feature_impact`
- `Model.request_feature_impact`
- `Model.get_or_request_feature_impact`

- Secondary dataset configuration retrieve and deletion is easier now though new `SecondaryDatasetConfigurations.delete` soft deletes a Secondary dataset configuration. `SecondaryDatasetConfigurations.get` retrieve a Secondary dataset configuration.

- Retrieve relationships configuration which is applied on the given feature discovery project using `Project.get_relationships_configuration`.

34.3.3 Bugfixes

- An issue with input validation of the Batch Prediction module
- `parent_model_id` was not visible for all frozen models
- Batch Prediction jobs that used other output types than *local_file* failed when using `.wait_for_completion()`
- A race condition in the Batch Prediction file scoring logic

34.3.4 API Changes

- Three new fields were added to the *Dataset* object. This reflects the updated fields in the public API routes at *api/v2/datasets/*. The added fields are:
 - `processing_state`: Current ingestion process state of the dataset
 - `row_count`: The number of rows in the dataset.
 - `size`: The size of the dataset as a CSV in bytes.

34.3.5 Deprecation Summary

- `datarobot.enums.VARIABLE_TYPE_TRANSFORM.CATEGORICAL` for is deprecated for the following and will be removed:
 - meth:*Project.batch_features_type_transform*
 - meth:*Project.create_type_transform_feature*

34.4 2.20.0

34.4.1 New Features

- There is a new *Dataset* object that implements some of the public API routes at *api/v2/datasets/*. This also adds two new feature classes and a details class.
 - *DatasetFeature*
 - *DatasetFeatureHistogram*
 - *DatasetDetails*

Functionality:

- Create a Dataset by uploading from a file, URL or in-memory datasource.
 - * *Dataset.create_from_file*
 - * *Dataset.create_from_in_memory_data*
 - * *Dataset.create_from_url*
- Get Datasets or elements of Dataset with:
 - * *Dataset.list* lists available Datasets
 - * *Dataset.get* gets a specified Dataset
 - * *Dataset.update* updates the Dataset with the latest server information.

- * `Dataset.get_details` gets the `DatasetDetails` of the `Dataset`.
- * `Dataset.get_all_features` gets a list of the `Dataset`'s Features.
- * `Dataset.get_file` downloads the `Dataset` as a csv file.
- * `Dataset.get_projects` gets a list of `Projects` that use the `Dataset`.
- Modify, delete or un-delete a `Dataset`:
 - * `Dataset.modify` Changes the name and categories of the `Dataset`
 - * `Dataset.delete` soft deletes a `Dataset`.
 - * `Dataset.un_delete` un-deletes the `Dataset`. You cannot retrieve the IDs of deleted `Datasets`, so if you want to un-delete a `Dataset`, you need to store its ID before deletion.
- You can also create a `Project` using a `Dataset` with:
 - * `Project.create_from_dataset`
- Now it's possible to connect two or more datasets by specifying the relationships between them using Feature Engineering Graph so that DataRobot can automatically generate features based on connection between datasets. The `FeatureEngineeringGraph` class can now create, update, retrieve, list, delete feature engineering graphs call to methods
 - `FeatureEngineeringGraph.create` creates a new feature engineering graph
 - `FeatureEngineeringGraph.update` updates the name and description of the feature engineering graph
 - `FeatureEngineeringGraph.replace` replace the content of the feature engineering graph
 - `FeatureEngineeringGraph.delete` delete the feature engineering graph
 - `FeatureEngineeringGraph.retrieve` retrieve the feature engineering graph
 - `FeatureEngineeringGraph.list` list all the feature engineering graphs
- It's possible to share the feature engineering graph with others and list all the users who have access to a given feature engineering graph.
 - `FeatureEngineeringGraph.share` allows to share the given feature engineering graph with other users
 - `FeatureEngineeringGraph.get_access_list` list all the users who have access to the given feature engineering graph
- It is possible to create an alternative configuration for the secondary dataset which can be used during the prediction
 - `SecondaryDatasetConfigurations.create` allow to create secondary dataset configuration
- You can now filter the deployments returned by the `Deployment.list` command. You can do this by passing an instance of the `DeploymentListFilters` class to the `filters` keyword argument. The currently supported filters are:
 - `role`
 - `service_health`
 - `model_health`
 - `accuracy_health`
 - `execution_environment_type`
 - `materiality`

- A new workflow is available for making predictions in time series projects. To that end, `PredictionDataset` objects now contain the following new fields:
 - `forecast_point_range`: The start and end date of the range of dates available for use as the forecast point, detected based on the uploaded prediction dataset
 - `data_start_date`: A datestring representing the minimum primary date of the prediction dataset
 - `data_end_date`: A datestring representing the maximum primary date of the prediction dataset
 - `max_forecast_date`: A datestring representing the maximum forecast date of this prediction dataset
- Additionally, users no longer need to specify a `forecast_point` or `predictions_start_date` and `predictions_end_date` when uploading datasets for predictions in time series projects. More information can be found in the [time series predictions](#) documentation.
- Per-class lift chart data is now available for multiclass models using `Model.get_multiclass_lift_chart`.
- Unsupervised projects can now be created using the `Project.start` and `Project.set_target` methods by providing `unsupervised_mode=True`, provided that the user has access to unsupervised machine learning functionality. Contact support for more information.
- A new boolean attribute `unsupervised_mode` was added to `datarobot.DatetimePartitioningSpecification`. When it is set to `True`, datetime partitioning for unsupervised time series projects will be constructed for nowcasting: `forecast_window_start=forecast_window_end=0`.
- Users can now configure the start and end of the training partition as well as the end of the validation partition for backtests in a datetime-partitioned project. More information and example usage can be found in the [backtesting documentation](#).

34.4.2 Enhancements

- Updated the user agent header to show which python version.
- `Model.get_frozen_child_models` can be used to retrieve models that are frozen from a given model
- Added `datarobot.enums.TS_BLENDER_METHOD` to make it clearer which blender methods are allowed for use in time series projects.

34.4.3 Bugfixes

- An issue where uploaded CSVs would lose quotes during serialization causing issues when columns containing line terminators were loaded in a dataframe, has been fixed
- `Project.get_association_featurelists` is now using the correct endpoint name, but the old one will continue to work
- Python API `PredictionServer` supports now on-premise format of API response.

34.4.4 API Changes

34.4.5 Deprecation Summary

34.4.6 Configuration Changes

34.4.7 Documentation Changes

34.5 2.19.0

34.5.1 New Features

- Projects can be cloned using `Project.clone_project`
- Calendars used in time series projects now support having series-specific events, for instance if a holiday only affects some stores. This can be controlled by using new argument of the `CalendarFile.create` method. If multiseries id columns are not provided, calendar is considered to be single series and all events are applied to all series.
- We have expanded prediction intervals availability to the following use-cases:
 - Time series model deployments now support prediction intervals. See `Deployment.get_prediction_intervals_settings` and `Deployment.update_prediction_intervals_settings` for usage.
 - Prediction intervals are now supported for model exports for time series. To that end, a new optional parameter `prediction_intervals_size` has been added to `Model.request_transferable_export`.

More details on prediction intervals can be found in the [prediction intervals documentation](#).

- Allowed pairwise interaction groups can now be specified in `AdvancedOptions`. They will be used in GAM models during training.
- New deployments features:
 - Update the label and description of a deployment using `Deployment.update`.
 - `Association ID setting` can be retrieved and updated.
 - Regression deployments now support `prediction warnings`.
- For multiclass models now it's possible to get feature impact for each individual target class using `Model.get_multiclass_feature_impact`
- Added support for new `Batch Prediction API`.
- It is now possible to create and retrieve basic, oauth and s3 credentials with `Credential`.
- It's now possible to get feature association statuses for featurelists using `Project.get_association_featurelists`
- You can also pass a specific featurelist_id into `Project.get_associations`

34.5.2 Enhancements

- Added documentation to `Project.get_metrics` to detail the new ascending field that indicates how a metric should be sorted.

- Retraining of a model is processed asynchronously and returns a `ModelJob` immediately.
- Blender models can be retrained on a different set of data or a different feature list.
- Word cloud ngrams now has `variable` field representing the source of the ngram.
- Method `WordCloud.ngrams_per_class` can be used to split ngrams for better usability in multiclass projects.
- Method `Project.set_target` support new optional parameters `featureEngineeringGraphs` and `credentials`.
- Method `Project.upload_dataset` and `Project.upload_dataset_from_data_source` support new optional parameter `credentials`.
- Series accuracy retrieval methods (`DatetimeModel.get_series_accuracy_as_dataframe` and `DatetimeModel.download_series_accuracy_as_csv`) for multiseries time series projects now support additional parameters for specifying what data to retrieve, including:
 - `metric`: Which metric to retrieve scores for
 - `multiseries_value`: Only returns series with a matching multiseries ID
 - `order_by`: An attribute by which to sort the results

34.5.3 Bugfixes

- An issue when using `Feature.get` and `ModelingFeature.get` to retrieve summarized categorical feature has been fixed.

34.5.4 API Changes

- The `datarobot` package is now no longer a `namespace package`.
- `datarobot.enums.BLENDER_METHOD.FORECAST_DISTANCE` is removed (deprecated in 2.18.0).

34.5.5 Documentation Changes

- Updated *Residuals charts* documentation to reflect that the data rows include row numbers from the source dataset for projects created in DataRobot 5.3 and newer.

34.6 2.18.0

34.6.1 New Features

- *Residuals charts* can now be retrieved for non-time-aware regression models.
- *Deployment monitoring* can now be used to retrieve service stats, service health, accuracy info, permissions, and feature lists for deployments.
- *Time series* projects now support the Average by Forecast Distance blender, configured with more than one Forecast Distance. The blender blends the selected models, selecting the best three models based on the backtesting score for each Forecast Distance and averaging their predictions. The new blender method `FORECAST_DISTANCE_AVG` has been added to `datarobot.enums.BLENDER_METHOD`.

- `Deployment.submit_actuals` can now be used to submit data about actual results from a deployed model, which can be used to calculate accuracy metrics.

34.6.2 Enhancements

- Monotonic constraints are now supported for OTV projects. To that end, the parameters `monotonic_increasing_featurelist_id` and `monotonic_decreasing_featurelist_id` can be specified in calls to `Model.train_datetime` or `Project.train_datetime`.
- When *retrieving information about features*, information about summarized categorical variables is now available in a new `keySummary`.
- For *Word Clouds* in multiclass projects, values of the target class for corresponding word or ngram can now be passed using the new `class` parameter.
- Listing deployments using `Deployment.list` now support sorting and searching the results using the new `order_by` and `search` parameters.
- You can now get the model associated with a model job by getting the `model` variable on the *model job object*.
- The *Blueprint* class can now retrieve the `recommended_featurelist_id`, which indicates which feature list is recommended for this blueprint. If the field is not present, then there is no recommended feature list for this blueprint.
- The *Model* class now can be used to retrieve the `model_number`.
- The method `Model.get_supported_capabilities` now has an extra field `supportsCodeGeneration` to explain whether the model supports code generation.
- Calls to `Project.start` and `Project.upload_dataset` now support uploading data via S3 URI and *pathlib.Path* objects.
- Errors upon connecting to DataRobot are now clearer when an incorrect API Token is used.
- The *datarobot* package is now a *namespace package*.

34.6.3 Deprecation Summary

- `datarobot.enums.BLENDER_METHOD.FORECAST_DISTANCE` is deprecated and will be removed in 2.19. Use `FORECAST_DISTANCE_ENET` instead.

34.6.4 Documentation Changes

- Various typo and wording issues have been addressed.
- A new notebook showing regression-specific features is now been added to the *examples*.
- Documentation for *Access lists* has been added.

34.7 2.17.0

34.7.1 New Features

- *Deployments* can now be managed via the API by using the new *Deployment* class.

- Users can now list available prediction servers using `PredictionServer.list`.
- When *specifying datetime partitioning* settings, *time series* projects can now mark individual features as excluded from feature derivation using the `FeatureSettings.do_not_derive` attribute. Any features not specified will be assigned according to the `DatetimePartitioningSpecification.default_to_do_not_derive` value.
- Users can now submit multiple feature type transformations in a single batch request using `Project.batch_features_type_transform`.
- *Advanced Tuning* for non-Eureqa models (beta feature) is now enabled by default for all users. As of v2.17, all models are now supported other than blenders, open source, prime, scaleout, baseline and user-created.
- Information on feature clustering and the association strength between pairs of numeric or categorical features is now available. `Project.get_associations` can be used to retrieve pairwise feature association statistics and `Project.get_association_matrix_details` can be used to get a sample of the actual values used to measure association strength.

34.7.2 Enhancements

- `number_of_do_not_derive_features` has been added to the `datarobot.DatetimePartitioning` class to specify the number of features that are marked as excluded from derivation.
- Users with PyYAML \geq 5.1 will no longer receive a warning when using the `datarobot` package
- It is now possible to use files with unicode names for creating projects and prediction jobs.
- Users can now embed DataRobot-generated content in a `ComplianceDocTemplate` using keyword tags. [See here](#) for more details.
- The field `calendar_name` has been added to `datarobot.DatetimePartitioning` to display the name of the calendar used for a project.
- *Prediction intervals* are now supported for start-end retrained models in a time series project.
- Previously, all backtests had to be run before *prediction intervals* for a time series project could be requested with predictions. Now, backtests will be computed automatically if needed when prediction intervals are requested.

34.7.3 Bugfixes

- An issue affecting time series project creation for irregularly spaced dates has been fixed.
- `ComplianceDocTemplate` now supports empty text blocks in user sections.
- An issue when using `Predictions.get` to retrieve predictions metadata has been fixed.

34.7.4 Documentation Changes

- An overview on working with `ComplianceDocumentation` and `ComplianceDocTemplate` has been created. [See here](#) for more details.

34.8 2.16.0

34.8.1 New Features

- Three new methods for Series Accuracy have been added to the `DatetimeModel` class.

- Start a request to calculate Series Accuracy with `DatetimeModel.compute_series_accuracy`
- Once computed, Series Accuracy can be retrieved as a pandas.DataFrame using `DatetimeModel.get_series_accuracy_as_dataframe`
- Or saved as a CSV using `DatetimeModel.download_series_accuracy_as_csv`
- Users can now access *prediction intervals* data for each prediction with a `DatetimeModel`. For each model, prediction intervals estimate the range of values DataRobot expects actual values of the target to fall within. They are similar to a confidence interval of a prediction, but are based on the residual errors measured during the backtesting for the selected model.

34.8.2 Enhancements

- Information on the effective feature derivation window is now available for *time series projects* to specify the full span of historical data required at prediction time. It may be longer than the feature derivation window of the project depending on the differencing settings used.

Additionally, more of the project partitioning settings are also available on the `DatetimeModel` class. The new attributes are:

- `effective_feature_derivation_window_start`
- `effective_feature_derivation_window_end`
- `forecast_window_start`
- `forecast_window_end`
- `windows_basis_unit`
- Prediction metadata is now included in the return of `Predictions.get`

34.8.3 Documentation Changes

- Various typo and wording issues have been addressed.
- The example data that was meant to accompany the Time Series examples has been added to the zip file of the download in the *examples*.

34.9 2.15.1

34.9.1 Enhancements

- `CalendarFile.get_access_list` has been added to the `CalendarFile` class to return a list of users with access to a calendar file.
- A `role` attribute has been added to the `CalendarFile` class to indicate the access level a current user has to a calendar file. For more information on the specific access levels, see the *sharing* documentation.

34.9.2 Bugfixes

- Previously, attempting to retrieve the `calendar_id` of a project without a set target would result in an error. This has been fixed to return `None` instead.

34.10 2.15.0

34.10.1 New Features

- Previously available for only Eureqa models, Advanced Tuning methods and objects, including `Model.start_advanced_tuning_session`, `Model.get_advanced_tuning_parameters`, `Model.advanced_tune`, and `AdvancedTuningSession`, now support all models other than blender, open source, and user-created models. Use of Advanced Tuning via API for non-Eureqa models is in beta and not available by default, but can be enabled.
- Calendar Files for time series projects can now be created and managed through the `CalendarFile` class.

34.10.2 Enhancements

- The `dataframe` returned from `datarobot.PredictionExplanations.get_all_as_dataframe()` will now have each class label `class_X` be the same from row to row.
- The client is now more robust to networking issues by default. It will retry on more errors and respects `Retry-After` headers in HTTP 413, 429, and 503 responses.
- Added Forecast Distance blender for Time-Series projects configured with more than one Forecast Distance. It blends the selected models creating separate linear models for each Forecast Distance.
- `Project` can now be *shared* with other users.
- `Project.upload_dataset` and `Project.upload_dataset_from_data_source` will return a `PredictionDataset` with `data_quality_warnings` if potential problems exist around the uploaded dataset.
- `relax_known_in_advance_features_check` has been added to `Project.upload_dataset` and `Project.upload_dataset_from_data_source` to allow missing values from the known in advance features in the forecast window at prediction time.
- `cross_series_group_by_columns` has been added to `datarobot.DatetimePartitioning` to allow users the ability to indicate how to further split series into related groups.
- Information retrieval for `ROC Curve` has been extended to include `fraction_predicted_as_positive`, `fraction_predicted_as_negative`, `lift_positive` and `lift_negative`

34.10.3 Bugfixes

- Fixes an issue where the client would not be usable if it could not be sure it was compatible with the configured server

34.10.4 API Changes

- Methods for creating `datarobot.models.Project`: `create_from_mysql`, `create_from_oracle`, and `create_from_postgresql`, deprecated in 2.11, have now been removed. Use `datarobot.models.Project.create_from_data_source()` instead.
- `datarobot.FeatureSettings` attribute `apriori`, deprecated in 2.11, has been removed. Use `datarobot.FeatureSettings.known_in_advance` instead.

- `datarobot.DatetimePartitioning` attribute `default_to_a_priori`, deprecated in 2.11, has been removed. Use `datarobot.DatetimePartitioning.known_in_advance` instead.
- `datarobot.DatetimePartitioningSpecification` attribute `default_to_a_priori`, deprecated in 2.11, has been removed. Use `datarobot.DatetimePartitioningSpecification.known_in_advance` instead.

34.10.5 Deprecation Summary

34.10.6 Configuration Changes

- Now requires dependency on package `requests` to be at least version 2.21.
- Now requires dependency on package `urllib3` to be at least version 1.24.

34.10.7 Documentation Changes

- Advanced model insights notebook extended to contain information on visualisation of cumulative gains and lift charts.

34.11 2.14.2

34.11.1 Bugfixes

- Fixed an issue where searches of the HTML documentation would sometimes hang indefinitely

34.11.2 Documentation Changes

- Python3 is now the primary interpreter used to build the docs (this does not affect the ability to use the package with Python2)

34.12 2.14.1

34.12.1 Documentation Changes

- Documentation for the Model Deployment interface has been removed after the corresponding interface was removed in 2.13.0.

34.13 2.14.0

34.13.1 New Features

- The new method `Model.get_supported_capabilities` retrieves a summary of the capabilities supported by a particular model, such as whether it is eligible for Prime and whether it has word cloud data available.
- New class for working with model compliance documentation feature of DataRobot: `ComplianceDocumentation`

- New class for working with compliance documentation templates: *ComplianceDocTemplate*
- New class *FeatureHistogram* has been added to retrieve feature histograms for a requested maximum bin count
- Time series projects now support binary classification targets.
- Cross series features can now be created within time series multiserries projects using the `use_cross_series_features` and `aggregation_type` attributes of the *datarobot.DatetimePartitioningSpecification*. See the *Time Series* documentation for more info.

34.13.2 Enhancements

- Client instantiation now checks the endpoint configuration and provides more informative error messages. It also automatically corrects HTTP to HTTPS if the server responds with a redirect to HTTPS.
- *Project.upload_dataset* and *Project.create* now accept an optional parameter of `dataset_filename` to specify a file name for the dataset. This is ignored for url and file path sources.
- New optional parameter `fallback_to_parent_insights` has been added to *Model.get_lift_chart*, *Model.get_all_lift_charts*, *Model.get_confusion_chart*, *Model.get_all_confusion_charts*, *Model.get_roc_curve*, and *Model.get_all_roc_curves*. When *True*, a frozen model with missing insights will attempt to retrieve the missing insight data from its parent model.
- New `number_of_known_in_advance_features` attribute has been added to the *datarobot.DatetimePartitioning* class. The attribute specifies number of features that are marked as known in advance.
- *Project.set_worker_count* can now update the worker count on a project to the maximum number available to the user.
- *Recommended Models API* can now be used to retrieve model recommendations for datetime partitioned projects
- Timeseries projects can now accept feature derivation and forecast windows intervals in terms of number of the rows rather than a fixed time unit. *DatetimePartitioningSpecification* and *Project.set_target* support new optional parameter `windowsBasisUnit`, either `ROW` or detected time unit.
- Timeseries projects can now accept feature derivation intervals, forecast windows, forecast points and prediction start/end dates in milliseconds.
- *DataSources* and *DataStores* can now be *shared* with other users.
- Training predictions for datetime partitioned projects now support the new data subset *dr.enums.DATA_SUBSET_ALL_BACKTESTS* for requesting the predictions for all backtest validation folds.

34.13.3 API Changes

- The model recommendation type `Recommended` (deprecated in version 2.13.0) has been removed.

34.13.4 Documentation Changes

- **Example notebooks have been updated:**
 - Notebooks now work in Python 2 and Python 3
 - A notebook illustrating time series capability has been added
 - The financial data example has been replaced with an updated introductory example.

- To supplement the embedded Python notebooks in both the PDF and HTML docs bundles, the notebook files and supporting data can now be downloaded from the HTML docs bundle.
- Fixed a minor typo in the code sample for `get_or_request_feature_impact`

34.14 2.13.0

34.14.1 New Features

- The new method `Model.get_or_request_feature_impact` functionality will attempt to request feature impact and return the newly created feature impact object or the existing object so two calls are no longer required.
- New methods and objects, including `Model.start_advanced_tuning_session`, `Model.get_advanced_tuning_parameters`, `Model.advanced_tune`, and `AdvancedTuningSession`, were added to support the setting of Advanced Tuning parameters. This is currently supported for Eureka models only.
- New `is_starred` attribute has been added to the `Model` class. The attribute specifies whether a model has been marked as starred by user or not.
- Model can be marked as starred or being unstarred with `Model.star_model` and `Model.unstar_model`.
- When listing models with `Project.get_models`, the model list can now be filtered by the `is_starred` value.
- A custom prediction threshold may now be configured for each model via `Model.set_prediction_threshold`. When making predictions in binary classification projects, this value will be used when deciding between the positive and negative classes.
- `Project.check_blendable` can be used to confirm if a particular group of models are eligible for blending as some are not, e.g. scaleout models and datetime models with different training lengths.
- Individual cross validation scores can be retrieved for new models using `Model.get_cross_validation_scores`.

34.14.2 Enhancements

- Python 3.7 is now supported.
- Feature impact now returns not only the impact score for the features but also whether they were detected to be redundant with other high-impact features.
- A new `is_blocked` attribute has been added to the `Job` class, specifying whether a job is blocked from execution because one or more dependencies are not yet met.
- The `Featurelist` object now has new attributes reporting its creation time, whether it was created by a user or by DataRobot, and the number of models using the featurelist, as well as a new description field.
- Featurelists can now be renamed and have their descriptions updated with `Featurelist.update` and `ModelingFeaturelist.update`.
- Featurelists can now be deleted with `Featurelist.delete` and `ModelingFeaturelist.delete`.
- `ModelRecommendation.get` now accepts an optional parameter of type `datarobot.enums.RECOMMENDED_MODEL_TYPE` which can be used to get a specific kind of recommendation.
- Previously computed predictions can now be listed and retrieved with the `Predictions` class, without requiring a reference to the original `PredictJob`.

34.14.3 Bugfixes

- The Model Deployment interface which was previously visible in the client has been removed to allow the interface to mature, although the raw API is available as a `beta` API without full backwards compatibility support.

34.14.4 API Changes

- Added support for retrieving the Pareto Front of a Eureka model. See [ParetoFront](#).
- A new recommendation type `Recommended for Deployment` has been added to [ModelRecommendation](#) which is now returns as the default recommended model when available. See [Model Recommendation](#).

34.14.5 Deprecation Summary

- The feature previously referred to as `Reason Codes` has been renamed to `Prediction Explanation`, to provide increased clarity and accessibility. The old [ReasonCodes](#) interface has been deprecated and replaced with [PredictionExplanations](#).
- The recommendation type `Recommended` is deprecated and will no longer be returned in v2.14 of the API.

34.14.6 Documentation Changes

- Added a new documentation section [Model Recommendation](#).
- Time series projects support multiserries as well as single series data. They are now documented in the [Time Series Projects](#) documentation.

34.15 2.12.0

34.15.1 New Features

- Some models now have Missing Value reports allowing users with access to uncensored blueprints to retrieve a detailed breakdown of how numeric imputation and categorical converter tasks handled missing values. See the [documentation](#) for more information on the report.

34.16 2.11.0

34.16.1 New Features

- The new `ModelRecommendation` class can be used to retrieve the recommended models for a project.
- A new helper method `cross_validate` was added to class `Model`. This method can be used to request `Model's` Cross Validation score.

- Training a model with monotonic constraints is now supported. Training with monotonic constraints allows users to force models to learn monotonic relationships with respect to some features and the target. This helps users create accurate models that comply with regulations (e.g. insurance, banking). Currently, only certain blueprints (e.g. xgboost) support this feature, and it is only supported for regression and binary classification projects.
- DataRobot now supports [Database Connectivity](#), allowing databases to be used as the source of data for projects and prediction datasets. The feature works on top of the JDBC standard, so a variety of databases conforming to that standard are available; a list of databases with tested support for DataRobot is available in the user guide in the web application. See [Database Connectivity](#) for details.
- Added a new feature to retrieve feature logs for time series projects. Check `datarobot.DatetimePartitioning.feature_log_list()` and `datarobot.DatetimePartitioning.feature_log_retrieve()` for details.

34.16.2 API Changes

- New attributes supporting monotonic constraints have been added to the [AdvancedOptions](#), [Project](#), [Model](#), and [Blueprint](#) classes. See [monotonic constraints](#) for more information on how to configure monotonic constraints.
- New parameters `predictions_start_date` and `predictions_end_date` added to `Project.upload_dataset` to support bulk predictions upload for time series projects.

34.16.3 Deprecation Summary

- Methods for creating `datarobot.models.Project`: `create_from_mysql`, `create_from_oracle`, and `create_from_postgresql`, have been deprecated and will be removed in 2.14. Use `datarobot.models.Project.create_from_data_source()` instead.
- `datarobot.FeatureSettings` attribute `apriori`, has been deprecated and will be removed in 2.14. Use `datarobot.FeatureSettings.known_in_advance` instead.
- `datarobot.DatetimePartitioning` attribute `default_to_a_priori`, has been deprecated and will be removed in 2.14. `datarobot.DatetimePartitioning.known_in_advance` instead.
- `datarobot.DatetimePartitioningSpecification` attribute `default_to_a_priori`, has been deprecated and will be removed in 2.14. Use `datarobot.DatetimePartitioningSpecification.known_in_advance` instead.

34.16.4 Configuration Changes

- Retry settings compatible with those offered by urllib3's [Retry](#) interface can now be configured. By default, we will now retry connection errors that prevented requests from arriving at the server.

34.16.5 Documentation Changes

- [Advanced Model Insights](#) example has been updated to properly handle bin weights when rebinning.

34.17 2.9.0

34.17.1 New Features

- New `ModelDeployment` class can be used to track status and health of models deployed for predictions.

34.17.2 Enhancements

- DataRobot API now supports creating 3 new blender types - Random Forest, TensorFlow, LightGBM.
- Multiclass projects now support blenders creation for 3 new blender types as well as Average and ENET blenders.
- Models can be trained by requesting a particular row count using the new `training_row_count` argument with `Project.train`, `Model.train` and `Model.request_frozen_model` in non-datetime partitioned projects, as an alternative to the previous option of specifying a desired percentage of the project dataset. Specifying model size by row count is recommended when the float precision of `sample_pct` could be problematic, e.g. when training on a small percentage of the dataset or when training up to partition boundaries.
- New attributes `max_train_rows`, `scaleout_max_train_pct`, and `scaleout_max_train_rows` have been added to `Project`. `max_train_rows` specified the equivalent value to the existing `max_train_pct` as a row count. The scaleout fields can be used to see how far scaleout models can be trained on projects, which for projects taking advantage of scalable ingest may exceed the limits on the data available to non-scaleout blueprints.
- Individual features can now be marked as a priori or not a priori using the new `feature_settings` attribute when setting the target or specifying datetime partitioning settings on time series projects. Any features not specified in the `feature_settings` parameter will be assigned according to the `default_to_a_priori` value.
- Three new options have been made available in the `datarobot.DatetimePartitioningSpecification` class to fine-tune how time-series projects derive modeling features. `treat_as_exponential` can control whether data is analyzed as an exponential trend and transformations like log-transform are applied. `differencing_method` can control which differencing method to use for stationary data. `periodicities` can be used to specify periodicities occurring within the data. All are optional and defaults will be chosen automatically if they are unspecified.

34.17.3 API Changes

- Now `training_row_count` is available on non-datetime models as well as `rowCount` based datetime models. It reports the number of rows used to train the model (equivalent to `sample_pct`).
- Features retrieved from `Feature.get` now include `target_leakage`.

34.18 2.8.1

34.18.1 Bugfixes

- The documented default `connect_timeout` will now be correctly set for all configuration mechanisms, so that requests that fail to reach the DataRobot server in a reasonable amount of time will now error instead of hanging indefinitely. If you observe that you have started seeing `ConnectTimeout` errors, please configure your `connect_timeout` to a larger value.

- Version of `trafaret` library this package depends on is now pinned to `trafaret>=0.7, <1.1` since versions outside that range are known to be incompatible.

34.19 2.8.0

34.19.1 New Features

- The DataRobot API supports the creation, training, and predicting of multiclass classification projects. DataRobot, by default, handles a dataset with a numeric target column as regression. If your data has a numeric cardinality of fewer than 11 classes, you can override this behavior to instead create a multiclass classification project from the data. To do so, use the `set_target` function, setting `target_type="Multiclass"`. If DataRobot recognizes your data as categorical, and it has fewer than 11 classes, using multiclass will create a project that classifies which label the data belongs to.
- The DataRobot API now includes Rating Tables. A rating table is an exportable csv representation of a model. Users can influence predictions by modifying them and creating a new model with the modified table. See the [documentation](#) for more information on how to use rating tables.
- `scaleout_modeling_mode` has been added to the `AdvancedOptions` class used when setting a project target. It can be used to control whether scaleout models appear in the autopilot and/or available blueprints. Scaleout models are only supported in the Hadoop environment with the corresponding user permission set.
- A new premium add-on product, Time Series, is now available. New projects can be created as time series projects which automatically derive features from past data and forecast the future. See the [time series documentation](#) for more information.
- The `Feature` object now returns the EDA summary statistics (i.e., mean, median, min, max, and standard deviation) for features where this is available (e.g., numeric, date, time, currency, and length features). These summary statistics will be formatted in the same format as the data it summarizes.
- The DataRobot API now supports Training Predictions workflow. Training predictions are made by a model for a subset of data from original dataset. User can start a job which will make those predictions and retrieve them. See the [documentation](#) for more information on how to use training predictions.
- DataRobot now supports retrieving a [model blueprint chart](#) and a [model blueprint docs](#).
- With the introduction of Multiclass Classification projects, DataRobot needed a better way to explain the performance of a multiclass model so we created a new Confusion Chart. The API now supports retrieving and interacting with confusion charts.

34.19.2 Enhancements

- `DatetimePartitioningSpecification` now includes the optional `disable_holdout` flag that can be used to disable the holdout fold when creating a project with datetime partitioning.
- When retrieving reason codes on a project using an exposure column, predictions that are adjusted for exposure can be retrieved.
- File URIs can now be used as sourcedata when creating a project or uploading a prediction dataset. The file URI must refer to an allowed location on the server, which is configured as described in the user guide documentation.
- The advanced options available when setting the target have been extended to include the new parameter `events_count` as a part of the `AdvancedOptions` object to allow specifying the events count column. See the user guide documentation in the webapp for more information on events count.
- `PredictJob.get_predictions` now returns predicted probability for each class in the dataframe.

- `PredictJob.get_predictions` now accepts `prefix` parameter to prefix the classes name returned in the predictions dataframe.

34.19.3 API Changes

- Add `target_type` parameter to `set_target()` and `start()`, used to override the project default.

34.20 2.7.2

34.20.1 Documentation Changes

- Updated link to the publicly hosted documentation.

34.21 2.7.1

34.21.1 Documentation Changes

- Online documentation hosting has migrated from PythonHosted to Read The Docs. Minor code changes have been made to support this.

34.22 2.7.0

34.22.1 New Features

- Lift chart data for models can be retrieved using the `Model.get_lift_chart` and `Model.get_all_lift_charts` methods.
- ROC curve data for models in classification projects can be retrieved using the `Model.get_roc_curve` and `Model.get_all_roc_curves` methods.
- Semi-automatic autopilot mode is removed.
- Word cloud data for text processing models can be retrieved using `Model.get_word_cloud` method.
- Scoring code JAR file can be downloaded for models supporting code generation.

34.22.2 Enhancements

- A `__repr__` method has been added to the `PredictionDataset` class to improve readability when using the client interactively.
- `Model.get_parameters` now includes an additional key in the derived features it includes, showing the coefficients for individual stages of multistage models (e.g. Frequency-Severity models).
- When training a `DatetimeModel` on a window of data, a `time_window_sample_pct` can be specified to take a uniform random sample of the training data instead of using all data within the window.
- Installing of DataRobot package now has an “Extra Requirements” section that will install all of the dependencies needed to run the example notebooks.

34.22.3 Documentation Changes

- A new example notebook describing how to visualize some of the newly available model insights including lift charts, ROC curves, and word clouds has been added to the examples section.
- A new section for *Common Issues* has been added to *Getting Started* to help debug issues related to client installation and usage.

34.23 2.6.1

34.23.1 Bugfixes

- Fixed a bug with *Model.get_parameters* raising an exception on some valid parameter values.

34.23.2 Documentation Changes

- Fixed sorting order in Feature Impact example code snippet.

34.24 2.6.0

34.24.1 New Features

- A new partitioning method (datetime partitioning) has been added. The recommended workflow is to preview the partitioning by creating a *DatetimePartitioningSpecification* and passing it into *DatetimePartitioning.generate*, inspect the results and adjust as needed for the specific project dataset by adjusting the *DatetimePartitioningSpecification* and re-generating, and then set the target by passing the final *DatetimePartitioningSpecification* object to the partitioning_method parameter of *Project.set_target*.
- When interacting with datetime partitioned projects, *DatetimeModel* can be used to access more information specific to models in datetime partitioned projects. See [the documentation](#) for more information on differences in the modeling workflow for datetime partitioned projects.
- The advanced options available when setting the target have been extended to include the new parameters `offset` and `exposure` (part of the *AdvancedOptions* object) to allow specifying offset and exposure columns to apply to predictions generated by models within the project. See the user guide documentation in the webapp for more information on offset and exposure columns.
- Blueprints can now be retrieved directly by project_id and blueprint_id via *Blueprint.get*.
- Blueprint charts can now be retrieved directly by project_id and blueprint_id via *BlueprintChart.get*. If you already have an instance of *Blueprint* you can retrieve its chart using *Blueprint.get_chart*.
- Model parameters can now be retrieved using *ModelParameters.get*. If you already have an instance of *Model* you can retrieve its parameters using *Model.get_parameters*.
- Blueprint documentation can now be retrieved using *Blueprint.get_documents*. It will contain information about the task, its parameters and (when available) links and references to additional sources.
- The DataRobot API now includes Reason Codes. You can now compute reason codes for prediction datasets. You are able to specify thresholds on which rows to compute reason codes for to speed up computation by skipping rows based on the predictions they generate. See the reason codes [documentation](#) for more information.

34.24.2 Enhancements

- A new parameter has been added to the *AdvancedOptions* used with *Project.set_target*. By specifying *accuracyOptimizedMb=True* when creating *AdvancedOptions*, longer-running models that may have a high accuracy will be included in the autopilot and made available to run manually.
- A new option for *Project.create_type_transform_feature* has been added which explicitly truncates data when casting numerical data as categorical data.
- Added 2 new blenders for projects that use MAD or Weighted MAD as a metric. The MAE blender uses BFGS optimization to find linear weights for the blender that minimize mean absolute error (compared to the GLM blender, which finds linear weights that minimize RMSE), and the MAEL1 blender uses BFGS optimization to find linear weights that minimize MAE + a L1 penalty on the coefficients (compared to the ENET blender, which minimizes RMSE + a combination of the L1 and L2 penalty on the coefficients).

34.24.3 Bugfixes

- Fixed a bug (affecting Python 2 only) with printing any model (including frozen and prime models) whose *model_type* is not ascii.
- *FrozenModels* were unable to correctly use methods inherited from *Model*. This has been fixed.
- When calling *get_result* for a *Job*, *ModelJob*, or *PredictJob* that has errored, *AsyncProcessUnsuccessfulError* will now be raised instead of *JobNotFinished*, consistently with the behaviour of *get_result_when_complete*.

34.24.4 Deprecation Summary

- Support for the experimental Recommender Problems projects has been removed. Any code relying on *RecommenderSettings* or the *recommender_settings* argument of *Project.set_target* and *Project.start* will error.
- *Project.update*, deprecated in v2.2.32, has been removed in favor of specific updates: *rename*, *unlock_holdout*, *set_worker_count*.

34.24.5 Documentation Changes

- The link to Configuration from the Quickstart page has been fixed.

34.25 2.5.1

34.25.1 Bugfixes

- Fixed a bug (affecting Python 2 only) with printing blueprints whose names are not ascii.
- Fixed an issue where the weights column (for weighted projects) did not appear in the *advanced_options* of a *Project*.

34.26 2.5.0

34.26.1 New Features

- Methods to work with blender models have been added. Use *Project.blend* method to create new blenders, *Project.get_blenders* to get the list of existing blenders and *BlenderModel.get* to retrieve a model with blender-specific information.
- Projects created via the API can now use smart downsampling when setting the target by passing *smart_downsampled* and *majority_downsampling_rate* into the *AdvancedOptions* object used with *Project.set_target*. The smart sampling options used with an existing project will be available as part of *Project.advanced_options*.
- Support for frozen models, which use tuning parameters from a parent model for more efficient training, has been added. Use *Model.request_frozen_model* to create a new frozen model, *Project.get_frozen_models* to get the list of existing frozen models and *FrozenModel.get* to retrieve a particular frozen model.

34.26.2 Enhancements

- The inferred date format (e.g. `âĀĬJ%Y-%m-%d %H:%M:%SâĀĬI`) is now included in the Feature object. For non-date features, it will be None.
- When specifying the API endpoint in the configuration, the client will now behave correctly for endpoints with and without trailing slashes.

34.27 2.4.0

34.27.1 New Features

- The premium add-on product *DataRobot Prime* has been added. You can now approximate a model on the leaderboard and download executable code for it. See documentation for further details, or talk to your account representative if the feature is not available on your account.
- (Only relevant for on-premise users with a Standalone Scoring cluster.) Methods (*request_transferable_export* and *download_export*) have been added to the *Model* class for exporting models (which will only work if model export is turned on). There is a new class *ImportedModel* for managing imported models on a Standalone Scoring cluster.
- It is now possible to create projects from a WebHDFS, PostgreSQL, Oracle or MySQL data source. For more information see the documentation for the relevant *Project* classmethods: *create_from_hdfs*, *create_from_postgresql*, *create_from_oracle* and *create_from_mysql*.
- *Job.wait_for_completion*, which waits for a job to complete without returning anything, has been added.

34.27.2 Enhancements

- The client will now check the API version offered by the server specified in configuration, and give a warning if the client version is newer than the server version. The DataRobot server is always backwards compatible with old clients, but new clients may have functionality that is not implemented on older server versions. This issue mainly affects users with on-premise deployments of DataRobot.

34.27.3 Bugfixes

- Fixed an issue where *Model.request_predictions* might raise an error when predictions finished very quickly instead of returning the job.

34.27.4 API Changes

- To set the target with quickrun autopilot, call *Project.set_target* with *mode=AUTOPILOT_MODE.QUICK* instead of specifying *quickrun=True*.

34.27.5 Deprecation Summary

- Semi-automatic mode for autopilot has been deprecated and will be removed in 3.0. Use manual or fully automatic instead.
- Use of the *quickrun* argument in *Project.set_target* has been deprecated and will be removed in 3.0. Use *mode=AUTOPILOT_MODE.QUICK* instead.

34.27.6 Configuration Changes

- It is now possible to control the SSL certificate verification by setting the parameter *ssl_verify* in the config file.

34.27.7 Documentation Changes

- The “Modeling Airline Delay” example notebook has been updated to work with the new 2.3 enhancements.
- Documentation for the generic *Job* class has been added.
- Class attributes are now documented in the *API Reference* section of the documentation.
- The changelog now appears in the documentation.
- There is a new section dedicated to configuration, which lists all of the configuration options and their meanings.

34.28 2.3.0

34.28.1 New Features

- The DataRobot API now includes Feature Impact, an approach to measuring the relevance of each feature that can be applied to any model. The *Model* class now includes methods *request_feature_impact* (which creates and returns a feature impact job) and *get_feature_impact* (which can retrieve completed feature impact results).
- A new improved workflow for predictions now supports first uploading a dataset via *Project.upload_dataset*, then requesting predictions via *Model.request_predictions*. This allows us to better support predictions on larger datasets and non-ascii files.
- Datasets previously uploaded for predictions (represented by the *PredictionDataset* class) can be listed from *Project.get_datasets* and retrieve and deleted via *PredictionDataset.get* and *PredictionDataset.delete*.
- You can now create a new feature by re-interpreting the type of an existing feature in a project by using the *Project.create_type_transform_feature* method.

- The *Job* class now includes a *get* method for retrieving a job and a *cancel* method for canceling a job.
- All of the jobs classes (*Job*, *ModelJob*, *PredictJob*) now include the following new methods: *refresh* (for refreshing the data in the job object), *get_result* (for getting the completed resource resulting from the job), and *get_result_when_complete* (which waits until the job is complete and returns the results, or times out).
- A new method *Project.refresh* can be used to update *Project* objects with the latest state from the server.
- A new function *datarobot.async.wait_for_async_resolution* can be used to poll for the resolution of any generic asynchronous operation on the server.

34.28.2 Enhancements

- The *JOB_TYPE* enum now includes *FEATURE_IMPACT*.
- The *QUEUE_STATUS* enum now includes *ABORTED* and *COMPLETED*.
- The *Project.create* method now has a *read_timeout* parameter which can be used to keep open the connection to DataRobot while an uploaded file is being processed. For very large files this time can be substantial. Appropriately raising this value can help avoid timeouts when uploading large files.
- The method *Project.wait_for_autopilot* has been enhanced to error if the project enters a state where autopilot may not finish. This avoids a situation that existed previously where users could wait indefinitely on their project that was not going to finish. However, users are still responsible to make sure a project has more than zero workers, and that the queue is not paused.
- *Feature.get* now supports retrieving features by feature name. (For backwards compatibility, feature IDs are still supported until 3.0.)
- File paths that have unicode directory names can now be used for creating projects and PredictJobs. The filename itself must still be ascii, but containing directory names can have other encodings.
- Now raises more specific *JobAlreadyRequested* exception when we refuse a model fitting request as a duplicate. Users can explicitly catch this exception if they want it to be ignored.
- A *file_name* attribute has been added to the *Project* class, identifying the file name associated with the original project dataset. Note that if the project was created from a data frame, the file name may not be helpful.
- The connect timeout for establishing a connection to the server can now be set directly. This can be done in the yaml configuration of the client, or directly in the code. The default timeout has been lowered from 60 seconds to 6 seconds, which will make detecting a bad connection happen much quicker.

34.28.3 Bugfixes

- Fixed a bug (affecting Python 2 only) with printing features and featurelists whose names are not ascii.

34.28.4 API Changes

- Job class hierarchy is rearranged to better express the relationship between these objects. See documentation for *datarobot.models.job* for details.
- *Featurelist* objects now have a *project_id* attribute to indicate which project they belong to. Directly accessing the *project* attribute of a *Featurelist* object is now deprecated
- Support INI-style configuration, which was deprecated in v2.1, has been removed. yaml is the only supported configuration format.

- The method *Project.get_jobs* method, which was deprecated in v2.1, has been removed. Users should use the *Project.get_model_jobs* method instead to get the list of model jobs.

34.28.5 Deprecation Summary

- *PredictJob.create* has been deprecated in favor of the alternate workflow using *Model.request_predictions*.
- *Feature.converter* (used internally for object construction) has been made private.
- **Model.fetch_resource_data has been deprecated and will be removed in 3.0. To fetch a model from its ID, use Model.get.**
- The ability to use *Feature.get* with feature IDs (rather than names) is deprecated and will be removed in 3.0.
- Instantiating a *Project*, *Model*, *Blueprint*, *Featurelist*, or *Feature* instance from a *dict* of data is now deprecated. Please use the *from_data* classmethod of these classes instead. Additionally, instantiating a *Model* from a tuple or by using the keyword argument *data* is also deprecated.
- Use of the attribute *Featurelist.project* is now deprecated. You can use the *project_id* attribute of a *Featurelist* to instantiate a *Project* instance using *Project.get*.
- Use of the attributes *Model.project*, *Model.blueprint*, and *Model.featurelist* are all deprecated now to avoid use of partially instantiated objects. Please use the ids of these objects instead.
- Using a *Project* instance as an argument in *Featurelist.get* is now deprecated. Please use a *project_id* instead. Similarly, using a *Project* instance in *Model.get* is also deprecated, and a *project_id* should be used in its place.

34.28.6 Configuration Changes

- Previously it was possible (though unintended) that the client configuration could be mixed through environment variables, configuration files, and arguments to *datarobot.Client*. This logic is now simpler - please see the *Getting Started* section of the documentation for more information.

34.29 2.2.33

34.29.1 Bugfixes

- Fixed a bug with non-ascii project names using the package with Python 2.
- Fixed an error that occurred when printing projects that had been constructed from an ID only or printing printing models that had been constructed from a tuple (which impacted printing *PredictJobs*).
- Fixed a bug with project creation from non-ascii file names. Project creation from non-ascii file names is not supported, so this now raises a more informative exception. The project name is no longer used as the file name in cases where we do not have a file name, which prevents non-ascii project names from causing problems in those circumstances.
- Fixed a bug (affecting Python 2 only) with printing projects, features, and featurelists whose names are not ascii.

34.30 2.2.32

34.30.1 New Features

- `Project.get_features` and `Feature.get` methods have been added for feature retrieval.
- A generic `Job` entity has been added for use in retrieving the entire queue at once. Calling `Project.get_all_jobs` will retrieve all (appropriately filtered) jobs from the queue. Those can be cancelled directly as generic jobs, or transformed into instances of the specific job class using `ModelJob.from_job` and `PredictJob.from_job`, which allow all functionality previously available via the `ModelJob` and `PredictJob` interfaces.
- `Model.train` now supports `featurelist_id` and `scoring_type` parameters, similar to `Project.train`.

34.30.2 Enhancements

- Deprecation warning filters have been updated. By default, a filter will be added ensuring that usage of deprecated features will display a warning once per new usage location. In order to hide deprecation warnings, a filter like `warnings.filterwarnings('ignore', category=DataRobotDeprecationWarning)` can be added to a script so no such warnings are shown. Watching for deprecation warnings to avoid reliance on deprecated features is recommended.
- If your client is misconfigured and does not specify an endpoint, the cloud production server is no longer used as the default as in many cases this is not the correct default.
- This changelog is now included in the distributable of the client.

34.30.3 Bugfixes

- Fixed an issue where updating the global client would not affect existing objects with cached clients. Now the global client is used for every API call.
- An issue where mistyping a filepath for use in a file upload has been resolved. Now an error will be raised if it looks like the raw string content for modeling or predictions is just one single line.

34.30.4 API Changes

- Use of username and password to authenticate is no longer supported - use an API token instead.
- Usage of `start_time` and `finish_time` parameters in `Project.get_models` is not supported both in filtering and ordering of models
- Default value of `sample_pct` parameter of `Model.train` method is now `None` instead of `100`. If the default value is used, models will be trained with all of the available *training* data based on project configuration, rather than with entire dataset including holdout for the previous default value of `100`.
- `order_by` parameter of `Project.list` which was deprecated in v2.0 has been removed.
- `recommendation_settings` parameter of `Project.start` which was deprecated in v0.2 has been removed.
- `Project.status` method which was deprecated in v0.2 has been removed.
- `Project.wait_for_aim_stage` method which was deprecated in v0.2 has been removed.

- `Delay`, `ConstantDelay`, `NoDelay`, `ExponentialBackoffDelay`, `RetryManager` classes from `retry` module which were deprecated in v2.1 were removed.
- Package renamed to `datarobot`.

34.30.5 Deprecation Summary

- `Project.update` deprecated in favor of specific updates: `rename`, `unlock_holdout`, `set_worker_count`.

34.30.6 Documentation Changes

- A new use case involving financial data has been added to the `examples` directory.
- Added documentation for the partition methods.

34.31 2.1.31

34.31.1 Bugfixes

- In Python 2, using a unicode token to instantiate the client will now work correctly.

34.32 2.1.30

34.32.1 Bugfixes

- The minimum required version of `trafaret` has been upgraded to 0.7.1 to get around an incompatibility between it and `setuptools`.

34.33 2.1.29

34.33.1 Enhancements

- Minimal used version of `requests_toolbelt` package changed from 0.4 to 0.6

34.34 2.1.28

34.34.1 New Features

- Default to reading YAML config file from `~/.config/datarobot/drconfig.yaml`
- Allow `config_path` argument to client
- `wait_for_autopilot` method added to `Project`. This method can be used to block execution until autopilot has finished running on the project.
- Support for specifying which featurelist to use with initial autopilot in `Project.set_target`

- `Project.get_predict_jobs` method has been added, which looks up all prediction jobs for a project
- `Project.start_autopilot` method has been added, which starts autopilot on specified featurelist
- The schema for `PredictJob` in DataRobot API v2.1 now includes a message. This attribute has been added to the `PredictJob` class.
- `PredictJob.cancel` now exists to cancel prediction jobs, mirroring `ModelJob.cancel`
- `Project.from_async` is a new classmethod that can be used to wait for an async resolution in project creation. Most users will not need to know about it as it is used behind the scenes in `Project.create` and `Project.set_target`, but power users who may run into periodic connection errors will be able to catch the new `ProjectAsyncFailureError` and decide if they would like to resume waiting for async process to resolve

34.34.2 Enhancements

- `AUTOPILOT_MODE` enum now uses string names for autopilot modes instead of numbers

34.34.3 Deprecation Summary

- `ConstantDelay`, `NoDelay`, `ExponentialBackoffDelay`, and `RetryManager` utils are now deprecated
- INI-style config files are now deprecated (in favor of YAML config files)
- Several functions in the `utils` submodule are now deprecated (they are being moved elsewhere and are not considered part of the public interface)
- `Project.get_jobs` has been renamed `Project.get_model_jobs` for clarity and deprecated
- Support for the experimental date partitioning has been removed in DataRobot API, so it is being removed from the client immediately.

34.34.4 API Changes

- In several places where `AppPlatformError` was being raised, now `TypeError`, `ValueError` or `InputNotUnderstoodError` are now used. With this change, one can now safely assume that when catching an `AppPlatformError` it is because of an unexpected response from the server.
- `AppPlatformError` has gained a two new attributes, `status_code` which is the HTTP status code of the unexpected response from the server, and `error_code` which is a DataRobot-defined error code. `error_code` is not used by any routes in DataRobot API 2.1, but will be in the future. In cases where it is not provided, the instance of `AppPlatformError` will have the attribute `error_code` set to `None`.
- Two new subclasses of `AppPlatformError` have been introduced, `ClientError` (for 400-level response status codes) and `ServerError` (for 500-level response status codes). These will make it easier to build automated tooling that can recover from periodic connection issues while polling.
- If a `ClientError` or `ServerError` occurs during a call to `Project.from_async`, then a `ProjectAsyncFailureError` (a subclass of `AsyncFailureError`) will be raised. That exception will have the `status_code` of the unexpected response from the server, and the location that was being polled to wait for the asynchronous process to resolve.

34.35 2.0.27

34.35.1 New Features

- `PredictJob` class was added to work with prediction jobs
- `wait_for_async_predictions` function added to *`predict_job`* module

34.35.2 Deprecation Summary

- The *`order_by`* parameter of the `Project.list` is now deprecated.

34.36 0.2.26

34.36.1 Enhancements

- `Project.set_target` will re-fetch the project data after it succeeds, keeping the client side in sync with the state of the project on the server
- `Project.create_featurelist` now throws `DuplicateFeaturesError` exception if passed list of features contains duplicates
- `Project.get_models` now supports *`snake_case`* arguments to its *`order_by`* keyword

34.36.2 Deprecation Summary

- `Project.wait_for_aim_stage` is now deprecated, as the REST Async flow is a more reliable method of determining that project creation has completed successfully
- `Project.status` is deprecated in favor of `Project.get_status`
- *`recommendation_settings`* parameter of `Project.start` is deprecated in favor of *`recommender_settings`*

34.36.3 Bugfixes

- `Project.wait_for_aim_stage` changed to support Python 3
- Fixed incorrect value of `SCORING_TYPE.cross_validation`
- Models returned by `Project.get_models` will now be correctly ordered when the *`order_by`* keyword is used

34.37 0.2.25

- Pinned versions of required libraries

34.38 0.2.24

Official release of v0.2

34.39 0.1.24

- Updated documentation
- Renamed parameter *name* of *Project.create* and *Project.start* to *project_name*
- Removed *Model.predict* method
- *wait_for_async_model_creation* function added to *modeljob* module
- *wait_for_async_status_service* of *Project* class renamed to *_wait_for_async_status_service*
- Can now use *auth_token* in config file to configure SDK

34.40 0.1.23

- Fixes a method that pointed to a removed route

34.41 0.1.22

- Added *featurelist_id* attribute to *ModelJob* class

34.42 0.1.21

- Removes *model* attribute from *ModelJob* class

34.43 0.1.20

- Project creation raises *AsyncProjectCreationError* if it was unsuccessful
- Removed *Model.list_prime_rulesets* and *Model.get_prime_ruleset* methods
- Removed *Model.predict_batch* method
- Removed *Project.create_prime_model* method
- Removed *PrimeRuleSet* model
- Adds backwards compatibility bridge for *ModelJob* async
- Adds *ModelJob.get* and *ModelJob.get_model*

34.44 0.1.19

- Minor bugfixes in *wait_for_async_status_service*

34.45 0.1.18

- Removes *submit_model* from Project until serverside implementation is improved
- Switches training URLs for new resource-based route at `/projects/<project_id>/models/`
- Job renamed to ModelJob, and using modelJobs route
- Fixes an inconsistency in argument order for *train* methods

34.46 0.1.17

- *wait_for_async_status_service* timeout increased from 60s to 600s

34.47 0.1.16

- *Project.create* will now handle both async/sync project creation

34.48 0.1.15

- All routes pluralized to sync with changes in API
- *Project.get_jobs* will request all jobs when no param specified
- dataframes from *predict* method will have pythonic names
- *Project.get_status* created, *Project.status* now deprecated
- *Project.unlock_holdout* created.
- Added *quickrun* parameter to *Project.set_target*
- Added *modelCategory* to Model schema
- Add *permalinks* featrue to Project and Model objects.
- *Project.create_prime_model* created

34.49 0.1.14

- *Project.set_worker_count* fix for compatibility with API change in project update.

34.50 0.1.13

- Add positive class to *set_target*.
- **Change attributes names of *Project*, *Model*, *Job* and *Blueprint***
 - *features* in *Model*, *Job* and *Blueprint* are now *processes*
 - *dataset_id* and *dataset_name* migrated to *featurelist_id* and *featurelist_name*.

– *samplepct* -> *sample_pct*

- *Model* has now *blueprint*, *project*, and *featurlist* attributes.
- Minor bugfixes.

34.51 0.1.12

- Minor fixes regarding rename *Job* attributes. *features* attributes now named *processes*, *samplepct* now is *sample_pct*.

34.52 0.1.11

(May 27, 2015)

- Minor fixes regarding migrating API from under_score names to camelCase.

34.53 0.1.10

(May 20, 2015)

- Remove *Project.upload_file*, *Project.upload_file_from_url* and *Project.attach_file* methods. Moved all logic that uploading file to *Project.create* method.

34.54 0.1.9

(May 15, 2015)

- Fix uploading file causing a lot of memory usage. Minor bugfixes.

CHAPTER 35

Indices and tables

- `genindex`
- `modindex`
- `search`

d

`datarobot.models.training_predictions,`
[419](#)

A

Accuracy (class in *datarobot.models*), 212
 AccuracyOverTime (class in *datarobot.models*), 213
 advanced_tune() (*datarobot.models.BlenderModel* method), 279
 advanced_tune() (*datarobot.models.DatetimeModel* method), 305
 advanced_tune() (*datarobot.models.Model* method), 258
 advanced_tune() (*datarobot.models.PrimeModel* method), 262
 advanced_tune() (*datarobot.models.RatingTableModel* method), 323
 AdvancedOptions (class in *datarobot.helpers*), 143
 AdvancedTuningSession (class in *datarobot.models.advanced_tuning*), 341
 assign_training_data() (*datarobot.CustomInferenceModel* method), 170

B

Backtest (class in *datarobot.helpers.partitioning_methods*), 359
 BacktestSpecification (class in *datarobot*), 350
 batch_features_type_transform() (*datarobot.models.Project* method), 404
 BatchPredictionJob (class in *datarobot.models*), 145
 blend() (*datarobot.models.Project* method), 400
 BlenderModel (class in *datarobot.models*), 277
 Blueprint (class in *datarobot.models*), 151
 BlueprintChart (class in *datarobot.models*), 152
 BlueprintTaskDocument (class in *datarobot.models*), 151
 bucket_sample_sizes (*datarobot.models.AccuracyOverTime* attribute), 214
 bucket_values (*datarobot.models.AccuracyOverTime* attribute), 214

bucket_values (*datarobot.models.ServiceStatsOverTime* attribute), 210

C

calculate_feature_impact() (*datarobot.CustomInferenceImage* method), 167
 calculate_prediction_intervals() (*datarobot.models.DatetimeModel* method), 304
 CalendarFile (class in *datarobot*), 153
 cancel() (*datarobot.CustomModelTest* method), 172
 cancel() (*datarobot.CustomModelVersionDependencyBuild* method), 176
 cancel() (*datarobot.models.FeatureImpactJob* method), 238
 cancel() (*datarobot.models.Job* method), 233
 cancel() (*datarobot.models.ModelJob* method), 343
 cancel() (*datarobot.models.PredictJob* method), 366
 cancel() (*datarobot.models.ShapMatrixJob* method), 236
 cancel() (*datarobot.models.TrainingPredictionsJob* method), 235
 check_blendable() (*datarobot.models.Project* method), 400
 clone_project() (*datarobot.models.Project* method), 405
 collect_payload() (*datarobot.DatetimePartitioningSpecification* method), 350
 ComplianceDocTemplate (class in *datarobot.models.compliance_doc_template*), 157
 ComplianceDocumentation (class in *datarobot.models.compliance_documentation*), 159
 compute() (*datarobot.models.visualai.ImageActivationMap* class method), 408
 compute() (*datarobot.models.visualai.ImageEmbedding* class method), 407

`compute_series_accuracy()`
 (`datarobot.models.DatetimeModel` *method*), 301
`ConfusionChart` (*class* *in* `datarobot.models.confusion_chart`), 160
`construct_duration_string()` (*in module* `datarobot.helpers.partitioning_methods`), 361
`copy_custom_model()`
 (`datarobot.CustomInferenceModel` *class method*), 169
`create()` (`datarobot.CalendarFile` *class method*), 153
`create()` (`datarobot.CustomInferenceImage` *class method*), 165
`create()` (`datarobot.CustomInferenceModel` *class method*), 168
`create()` (`datarobot.CustomModelTest` *class method*), 171
`create()` (`datarobot.DataDriver` *class method*), 181
`create()` (`datarobot.DataSource` *class method*), 187
`create()` (`datarobot.DataStore` *class method*), 183
`create()` (`datarobot.ExecutionEnvironment` *class method*), 177
`create()` (`datarobot.ExecutionEnvironmentVersion` *class method*), 178
`create()` (`datarobot.ExternalScores` *class method*), 215
`create()` (`datarobot.models.compliance_doc_template.ComplianceDocTemplate` *class method*), 158
`create()` (`datarobot.models.FeatureEngineeringGraph` *class method*), 226
`create()` (`datarobot.models.ImportedModel` *class method*), 320
`create()` (`datarobot.models.PayoffMatrix` *class method*), 362
`create()` (`datarobot.models.PredictJob` *class method*), 364
`create()` (`datarobot.models.Project` *class method*), 382
`create()` (`datarobot.models.RatingTable` *class method*), 410
`create()` (`datarobot.models.RelationshipsConfiguration` *class method*), 426
`create()` (`datarobot.models.SecondaryDatasetConfigurations` *class method*), 424
`create()` (`datarobot.models.ShapImpact` *class method*), 430
`create()` (`datarobot.models.ShapMatrix` *class method*), 376
`create()` (`datarobot.PredictionExplanations` *class method*), 370
`create()` (`datarobot.PredictionExplanationsInitialization` *class method*), 369
`create()` (`datarobot.ReasonCodes` *class method*), 413
`create()` (`datarobot.ReasonCodesInitialization` *class method*), 412
`create_azure()` (`datarobot.models.Credential` *class method*), 163
`create_basic()` (`datarobot.models.Credential` *class method*), 162
`create_clean()` (`datarobot.CustomModelVersion` *class method*), 173
`create_dataset()` (`datarobot.DataSource` *method*), 188
`create_featurelist()` (`datarobot.Dataset` *method*), 194
`create_featurelist()` (`datarobot.models.Project` *method*), 395
`create_from_custom_model_image()`
 (`datarobot.Deployment` *class method*), 198
`create_from_custom_model_version()`
 (`datarobot.Deployment` *class method*), 198
`create_from_data_source()` (`datarobot.Dataset` *class method*), 191
`create_from_data_source()`
 (`datarobot.models.Project` *class method*), 384
`create_from_dataset()`
 (`datarobot.models.Project` *class method*), 384
`create_from_file()` (`datarobot.Dataset` *class method*), 191
`create_from_hdfs()` (`datarobot.models.Project` *class method*), 383
`create_from_in_memory_data()`
 (`datarobot.Dataset` *class method*), 191
`create_from_json_file()`
 (`datarobot.models.compliance_doc_template.ComplianceDocTemplate` *class method*), 158
`create_from_learning_model()`
 (`datarobot.Deployment` *class method*), 197
`create_from_previous()`
 (`datarobot.CustomModelVersion` *class method*), 173
`create_from_rating_table()`
 (`datarobot.models.RatingTableModel` *class method*), 322
`create_from_url()` (`datarobot.Dataset` *class method*), 191
`create_gcp()` (`datarobot.models.Credential` *class method*), 164
`create_interaction_feature()`
 (`datarobot.models.Project` *method*), 405
`create_model()` (`datarobot.models.pareto_front.Solution` *method*), 345
`create_model()` (`datarobot.models.RatingTable` *method*), 411
`create_modeling_featurelist()`
 (`datarobot.models.Project` *method*), 396

[create_oauth\(\)](#) (*datarobot.models.Credential class method*), 162
[create_project\(\)](#) (*datarobot.Dataset method*), 195
[create_s3\(\)](#) (*datarobot.models.Credential class method*), 163
[create_type_transform_feature\(\)](#) (*datarobot.models.Project method*), 394
[Credential](#) (*class in datarobot.models*), 161
[cross_validate\(\)](#) (*datarobot.models.BlenderModel method*), 279
[cross_validate\(\)](#) (*datarobot.models.DatetimeModel method*), 300
[cross_validate\(\)](#) (*datarobot.models.Model method*), 257
[cross_validate\(\)](#) (*datarobot.models.PrimeModel method*), 262
[cross_validate\(\)](#) (*datarobot.models.RatingTableModel method*), 323
[CustomInferenceImage](#) (*class in datarobot*), 165
[CustomInferenceModel](#) (*class in datarobot*), 167
[CustomModelFileItem](#) (*class in datarobot.models.custom_model_version*), 164
[CustomModelTest](#) (*class in datarobot*), 170
[CustomModelVersion](#) (*class in datarobot*), 172
[CustomModelVersionDependencyBuild](#) (*class in datarobot*), 175

D

[DataDriver](#) (*class in datarobot*), 180
[datarobot.models.training_predictions](#) (*module*), 419
[Dataset](#) (*class in datarobot*), 190
[DatasetDetails](#) (*class in datarobot*), 195
[DatasetFeature](#) (*class in datarobot.models*), 221
[DatasetFeatureHistogram](#) (*class in datarobot.models*), 222
[DatasetFeaturelist](#) (*class in datarobot*), 229
[DataSource](#) (*class in datarobot*), 186
[DataSourceParameters](#) (*class in datarobot*), 189
[DataStore](#) (*class in datarobot*), 182
[DatetimeModel](#) (*class in datarobot.models*), 297
[DatetimePartitioning](#) (*class in datarobot*), 352
[DatetimePartitioningSpecification](#) (*class in datarobot*), 347
[delete\(\)](#) (*datarobot.CalendarFile class method*), 155
[delete\(\)](#) (*datarobot.CustomInferenceModel method*), 170
[delete\(\)](#) (*datarobot.DataDriver method*), 182
[delete\(\)](#) (*datarobot.Dataset class method*), 192
[delete\(\)](#) (*datarobot.DatasetFeaturelist method*), 229
[delete\(\)](#) (*datarobot.DataSource method*), 188
[delete\(\)](#) (*datarobot.DataStore method*), 184
[delete\(\)](#) (*datarobot.Deployment method*), 200
[delete\(\)](#) (*datarobot.ExecutionEnvironment method*), 178
[delete\(\)](#) (*datarobot.models.BatchPredictionJob method*), 150
[delete\(\)](#) (*datarobot.models.BlenderModel method*), 279
[delete\(\)](#) (*datarobot.models.compliance_doc_template.ComplianceDocTemplate method*), 159
[delete\(\)](#) (*datarobot.models.Credential method*), 162
[delete\(\)](#) (*datarobot.models.DatetimeModel method*), 305
[delete\(\)](#) (*datarobot.models.FeatureEngineeringGraph method*), 229
[delete\(\)](#) (*datarobot.models.Featurelist method*), 230
[delete\(\)](#) (*datarobot.models.ImportedModel method*), 321
[delete\(\)](#) (*datarobot.models.Model method*), 243
[delete\(\)](#) (*datarobot.models.ModelingFeaturelist method*), 232
[delete\(\)](#) (*datarobot.models.PayoffMatrix class method*), 363
[delete\(\)](#) (*datarobot.models.PredictionDataset method*), 369
[delete\(\)](#) (*datarobot.models.PrimeModel method*), 263
[delete\(\)](#) (*datarobot.models.Project method*), 387
[delete\(\)](#) (*datarobot.models.RatingTableModel method*), 323
[delete\(\)](#) (*datarobot.models.RelationshipsConfiguration method*), 429
[delete\(\)](#) (*datarobot.PredictionExplanations method*), 373
[delete\(\)](#) (*datarobot.PredictionExplanationsInitialization method*), 370
[delete\(\)](#) (*datarobot.ReasonCodes method*), 415
[delete\(\)](#) (*datarobot.ReasonCodesInitialization method*), 412
[Deployment](#) (*class in datarobot*), 197
[DeploymentListFilters](#) (*class in datarobot.models.deployment*), 207
[download\(\)](#) (*datarobot.CustomModelVersion method*), 175
[download\(\)](#) (*datarobot.ExecutionEnvironmentVersion method*), 180
[download\(\)](#) (*datarobot.models.BatchPredictionJob method*), 150
[download\(\)](#) (*datarobot.models.compliance_documentation.ComplianceDocTemplate method*), 160
[download\(\)](#) (*datarobot.models.PrimeFile method*), 380
[download\(\)](#) (*datarobot.models.RatingTable method*), 411
[download_export\(\)](#) (*datarobot.models.BlenderModel method*),

- 279
- `download_export()`
(*datarobot.models.DatetimeModel* method), 305
- `download_export()` (*datarobot.models.Model* method), 251
- `download_export()` (*datarobot.models.PrimeModel* method), 263
- `download_export()`
(*datarobot.models.RatingTableModel* method), 323
- `download_latest_version()`
(*datarobot.CustomInferenceModel* method), 168
- `download_scoring_code()`
(*datarobot.models.BlenderModel* method), 279
- `download_scoring_code()`
(*datarobot.models.DatetimeModel* method), 305
- `download_scoring_code()`
(*datarobot.models.Model* method), 256
- `download_scoring_code()`
(*datarobot.models.PrimeModel* method), 263
- `download_scoring_code()`
(*datarobot.models.RatingTableModel* method), 323
- `download_series_accuracy_as_csv()`
(*datarobot.models.DatetimeModel* method), 300
- `download_to_csv()` (*datarobot.models.Predictions* method), 379
- `download_to_csv()`
(*datarobot.models.training_predictions.TrainingPredictions* method), 423
- `download_to_csv()`
(*datarobot.PredictionExplanations* method), 373
- `download_to_csv()` (*datarobot.ReasonCodes* method), 415
- `DuplicateImage` (class in *datarobot.models.visualai*), 407
- ## E
- `EligibilityResult` (class in *datarobot.helpers.eligibility_result*), 406
- `encrypted_string()` (*datarobot.models.Project* class method), 383
- `ExecutionEnvironment` (class in *datarobot*), 176
- `ExecutionEnvironmentVersion` (class in *datarobot*), 178
- `ExternalLiftChart` (class in *datarobot*), 216
- `ExternalRocCurve` (class in *datarobot*), 216
- `ExternalScores` (class in *datarobot*), 214
- ## F
- `Feature` (class in *datarobot.models*), 217
- `feature_log_list()`
(*datarobot.DatetimePartitioning* class method), 357
- `feature_log_retrieve()`
(*datarobot.DatetimePartitioning* class method), 358
- `FeatureAssociation` (class in *datarobot.models.feature_association*), 409
- `FeatureAssociationFeaturelists` (class in *datarobot.models.feature_association*), 409
- `FeatureAssociationMatrixDetails` (class in *datarobot.models.feature_association*), 409
- `FeatureDrift` (class in *datarobot.models*), 211
- `FeatureEngineeringGraph` (class in *datarobot.models*), 224
- `FeatureHistogram` (class in *datarobot.models*), 223
- `FeatureImpactJob` (class in *datarobot.models*), 237
- `Featurelist` (class in *datarobot.models*), 230
- `FeatureSettings` (class in *datarobot*), 352
- `fetch_resource_data()`
(*datarobot.models.BlenderModel* class method), 279
- `fetch_resource_data()`
(*datarobot.models.DatetimeModel* class method), 305
- `fetch_resource_data()` (*datarobot.models.Model* class method), 242
- `fetch_resource_data()`
(*datarobot.models.PrimeModel* class method), 263
- `fetch_resource_data()`
(*datarobot.models.RatingTableModel* class method), 323
- `from_async()` (*datarobot.models.Project* class method), 384
- `from_job()` (*datarobot.models.ModelJob* class method), 343
- `from_job()` (*datarobot.models.PredictJob* class method), 364
- `from_server_data()` (*datarobot.DataSource* class method), 188
- `from_server_data()` (*datarobot.DataStore* class method), 185
- `from_server_data()`
(*datarobot.models.PayoffMatrix* class method), 363
- `FrozenModel` (class in *datarobot.models*), 318
- ## G
- `generate()` (*datarobot.DatetimePartitioning* class

- [method\), 356](#)
- [generate\(\) \(datarobot.models.compliance_documentation.ComplianceDocumentation class method\), 160](#)
- [generate_optimized\(\) \(datarobot.DatetimePartitioning class method\), 356](#)
- [get\(\) \(datarobot.CalendarFile class method\), 154](#)
- [get\(\) \(datarobot.CustomInferenceImage class method\), 166](#)
- [get\(\) \(datarobot.CustomInferenceModel class method\), 168](#)
- [get\(\) \(datarobot.CustomModelTest class method\), 172](#)
- [get\(\) \(datarobot.CustomModelVersion class method\), 174](#)
- [get\(\) \(datarobot.DataDriver class method\), 181](#)
- [get\(\) \(datarobot.Dataset class method\), 192](#)
- [get\(\) \(datarobot.DatasetDetails class method\), 197](#)
- [get\(\) \(datarobot.DatasetFeaturelist class method\), 229](#)
- [get\(\) \(datarobot.DataSource class method\), 186](#)
- [get\(\) \(datarobot.DataStore class method\), 183](#)
- [get\(\) \(datarobot.DatetimePartitioning class method\), 356](#)
- [get\(\) \(datarobot.Deployment class method\), 200](#)
- [get\(\) \(datarobot.ExecutionEnvironment class method\), 177](#)
- [get\(\) \(datarobot.ExecutionEnvironmentVersion class method\), 179](#)
- [get\(\) \(datarobot.ExternalLiftChart class method\), 216](#)
- [get\(\) \(datarobot.ExternalRocCurve class method\), 217](#)
- [get\(\) \(datarobot.ExternalScores class method\), 215](#)
- [get\(\) \(datarobot.models.Accuracy class method\), 212](#)
- [get\(\) \(datarobot.models.AccuracyOverTime class method\), 213](#)
- [get\(\) \(datarobot.models.BatchPredictionJob class method\), 150](#)
- [get\(\) \(datarobot.models.BlenderModel class method\), 278](#)
- [get\(\) \(datarobot.models.Blueprint class method\), 151](#)
- [get\(\) \(datarobot.models.BlueprintChart class method\), 152](#)
- [get\(\) \(datarobot.models.compliance_doc_template.ComplianceDocTemplate class method\), 159](#)
- [get\(\) \(datarobot.models.Credential class method\), 161](#)
- [get\(\) \(datarobot.models.DatasetFeatureHistogram class method\), 222](#)
- [get\(\) \(datarobot.models.DatetimeModel class method\), 299](#)
- [get\(\) \(datarobot.models.Feature class method\), 219](#)
- [get\(\) \(datarobot.models.FeatureEngineeringGraph class method\), 228](#)
- [get\(\) \(datarobot.models.FeatureHistogram class method\), 223](#)
- [get\(\) \(datarobot.models.FeatureImpactJob class method\), 238](#)
- [get\(\) \(datarobot.models.Featurelist class method\), 230](#)
- [get\(\) \(datarobot.models.Featurelist class method\), 230](#)
- [get\(\) \(datarobot.models.FrozenModel class method\), 319](#)
- [get\(\) \(datarobot.models.ImportedModel class method\), 320](#)
- [get\(\) \(datarobot.models.InteractionFeature class method\), 223](#)
- [get\(\) \(datarobot.models.Job class method\), 233](#)
- [get\(\) \(datarobot.models.missing_report.MissingValuesReport class method\), 240](#)
- [get\(\) \(datarobot.models.Model class method\), 242](#)
- [get\(\) \(datarobot.models.ModelBlueprintChart class method\), 152](#)
- [get\(\) \(datarobot.models.ModelingFeature class method\), 221](#)
- [get\(\) \(datarobot.models.ModelingFeaturelist class method\), 232](#)
- [get\(\) \(datarobot.models.ModelJob class method\), 343](#)
- [get\(\) \(datarobot.models.ModelRecommendation class method\), 417](#)
- [get\(\) \(datarobot.models.PayoffMatrix class method\), 362](#)
- [get\(\) \(datarobot.models.prediction_explanations.PredictionExplanations class method\), 375](#)
- [get\(\) \(datarobot.models.PredictionDataset class method\), 368](#)
- [get\(\) \(datarobot.models.Predictions class method\), 378](#)
- [get\(\) \(datarobot.models.PredictJob class method\), 365](#)
- [get\(\) \(datarobot.models.PrimeModel class method\), 262](#)
- [get\(\) \(datarobot.models.Project class method\), 382](#)
- [get\(\) \(datarobot.models.RatingTable class method\), 410](#)
- [get\(\) \(datarobot.models.RatingTableModel class method\), 322](#)
- [get\(\) \(datarobot.models.reason_codes.ReasonCodesPage class method\), 416](#)
- [get\(\) \(datarobot.models.RelationshipsConfiguration method\), 428](#)
- [get\(\) \(datarobot.models.ServiceStats class method\), 210](#)
- [get\(\) \(datarobot.models.ServiceStatsOverTime class method\), 210](#)
- [get\(\) \(datarobot.models.ShapImpact class method\), 430](#)
- [get\(\) \(datarobot.models.ShapMatrix class method\), 376](#)
- [get\(\) \(datarobot.models.ShapMatrixJob class method\), 236](#)
- [get\(\) \(datarobot.models.TargetDrift class method\), 210](#)
- [get\(\) \(datarobot.models.training_predictions.TrainingPredictions class method\), 422](#)
- [get\(\) \(datarobot.models.TrainingPredictionsJob class method\), 235](#)

`get()` (*datarobot.models.visualai.Image* class method), 406

`get()` (*datarobot.PredictionExplanations* class method), 370

`get()` (*datarobot.PredictionExplanationsInitialization* class method), 369

`get()` (*datarobot.ReasonCodes* class method), 412

`get()` (*datarobot.ReasonCodesInitialization* class method), 411

`get_access_list()` (*datarobot.CalendarFile* class method), 157

`get_access_list()` (*datarobot.DataSource* method), 188

`get_access_list()` (*datarobot.DataStore* method), 185

`get_access_list()` (*datarobot.models.FeatureEngineeringGraph* method), 229

`get_access_list()` (*datarobot.models.Project* method), 403

`get_accuracy()` (*datarobot.Deployment* method), 207

`get_accuracy_over_time()` (*datarobot.Deployment* method), 207

`get_advanced_tuning_parameters()` (*datarobot.models.BlenderModel* method), 280

`get_advanced_tuning_parameters()` (*datarobot.models.DatetimeModel* method), 306

`get_advanced_tuning_parameters()` (*datarobot.models.Model* method), 258

`get_advanced_tuning_parameters()` (*datarobot.models.PrimeModel* method), 263

`get_advanced_tuning_parameters()` (*datarobot.models.RatingTableModel* method), 324

`get_all()` (*datarobot.models.ModelRecommendation* class method), 417

`get_all_as_dataframe()` (*datarobot.models.Predictions* method), 378

`get_all_as_dataframe()` (*datarobot.models.training_predictions.TrainingPredictions* method), 422

`get_all_as_dataframe()` (*datarobot.PredictionExplanations* method), 372

`get_all_as_dataframe()` (*datarobot.ReasonCodes* method), 414

`get_all_confusion_charts()` (*datarobot.models.BlenderModel* method), 281

`get_all_confusion_charts()` (*datarobot.models.DatetimeModel* method), 307

`get_all_confusion_charts()` (*datarobot.models.Model* method), 255

`get_all_confusion_charts()` (*datarobot.models.PrimeModel* method), 265

`get_all_confusion_charts()` (*datarobot.models.RatingTableModel* method), 325

`get_all_features()` (*datarobot.Dataset* method), 194

`get_all_jobs()` (*datarobot.models.Project* method), 400

`get_all_lift_charts()` (*datarobot.models.BlenderModel* method), 282

`get_all_lift_charts()` (*datarobot.models.DatetimeModel* method), 308

`get_all_lift_charts()` (*datarobot.models.Model* method), 254

`get_all_lift_charts()` (*datarobot.models.PrimeModel* method), 265

`get_all_lift_charts()` (*datarobot.models.RatingTableModel* method), 326

`get_all_residuals_charts()` (*datarobot.models.BlenderModel* method), 282

`get_all_residuals_charts()` (*datarobot.models.DatetimeModel* method), 308

`get_all_residuals_charts()` (*datarobot.models.Model* method), 255

`get_all_residuals_charts()` (*datarobot.models.PrimeModel* method), 265

`get_all_residuals_charts()` (*datarobot.models.RatingTableModel* method), 326

`get_all_roc_curves()` (*datarobot.models.BlenderModel* method), 282

`get_all_roc_curves()` (*datarobot.models.DatetimeModel* method), 308

`get_all_roc_curves()` (*datarobot.models.Model* method), 256

`get_all_roc_curves()` (*datarobot.models.PrimeModel* method), 266

<code>get_all_roc_curves()</code> (<i>datarobot.models.RatingTableModel</i> method), 326	<code>get_cross_validation_scores()</code> (<i>datarobot.models.PrimeModel</i> method), 266
<code>get_as_dataframe()</code> (<i>datarobot.models.AccuracyOverTime</i> class method), 214	<code>get_cross_validation_scores()</code> (<i>datarobot.models.RatingTableModel</i> method), 326
<code>get_as_dataframe()</code> (<i>datarobot.models.ShapMatrix</i> method), 376	<code>get_datasets()</code> (<i>datarobot.models.Project</i> method), 390
<code>get_association_featurelists()</code> (<i>datarobot.models.Project</i> method), 393	<code>get_datetime_models()</code> (<i>datarobot.models.Project</i> method), 390
<code>get_association_id_settings()</code> (<i>datarobot.Deployment</i> method), 203	<code>get_default()</code> (<i>datarobot.models.compliance_doc_template.Compliance</i> class method), 158
<code>get_association_matrix_details()</code> (<i>datarobot.models.Project</i> method), 393	<code>get_details()</code> (<i>datarobot.Dataset</i> method), 194
<code>get_associations()</code> (<i>datarobot.models.Project</i> method), 393	<code>get_documents()</code> (<i>datarobot.models.Blueprint</i> method), 151
<code>get_blenders()</code> (<i>datarobot.models.Project</i> method), 401	<code>get_drift_tracking_settings()</code> (<i>datarobot.Deployment</i> method), 202
<code>get_blueprints()</code> (<i>datarobot.models.Project</i> method), 392	<code>get_feature_drift()</code> (<i>datarobot.Deployment</i> method), 206
<code>get_build_info()</code> (<i>datarobot.CustomModelVersionDependency</i> class method), 175	<code>get_feature_effect()</code> (<i>datarobot.models.BlenderModel</i> method), 337
<code>get_build_log()</code> (<i>datarobot.ExecutionEnvironmentVersion</i> method), 180	<code>get_feature_effect()</code> (<i>datarobot.models.DatetimeModel</i> method), 303
<code>get_calculated_prediction_intervals()</code> (<i>datarobot.models.DatetimeModel</i> method), 305	<code>get_feature_effect()</code> (<i>datarobot.models.Model</i> method), 249
<code>get_chart()</code> (<i>datarobot.models.Blueprint</i> method), 151	<code>get_feature_effect()</code> (<i>datarobot.models.PrimeModel</i> method), 266
<code>get_confusion_chart()</code> (<i>datarobot.models.BlenderModel</i> method), 282	<code>get_feature_effect()</code> (<i>datarobot.models.RatingTableModel</i> method), 327
<code>get_confusion_chart()</code> (<i>datarobot.models.DatetimeModel</i> method), 308	<code>get_feature_effect_metadata()</code> (<i>datarobot.models.BlenderModel</i> method), 283
<code>get_confusion_chart()</code> (<i>datarobot.models.Model</i> method), 255	<code>get_feature_effect_metadata()</code> (<i>datarobot.models.DatetimeModel</i> method), 301
<code>get_confusion_chart()</code> (<i>datarobot.models.PrimeModel</i> method), 266	<code>get_feature_effect_metadata()</code> (<i>datarobot.models.Model</i> method), 249
<code>get_confusion_chart()</code> (<i>datarobot.models.RatingTableModel</i> method), 326	<code>get_feature_effect_metadata()</code> (<i>datarobot.models.PrimeModel</i> method), 267
<code>get_cross_series_properties()</code> (<i>datarobot.models.Feature</i> method), 219	<code>get_feature_effect_metadata()</code> (<i>datarobot.models.RatingTableModel</i> method), 327
<code>get_cross_validation_scores()</code> (<i>datarobot.models.BlenderModel</i> method), 283	<code>get_feature_fit()</code> (<i>datarobot.models.BlenderModel</i> method), 284
<code>get_cross_validation_scores()</code> (<i>datarobot.models.DatetimeModel</i> method), 300	<code>get_feature_fit()</code> (<i>datarobot.models.DatetimeModel</i> method), 304
<code>get_cross_validation_scores()</code> (<i>datarobot.models.Model</i> method), 258	<code>get_feature_fit()</code> (<i>datarobot.models.Model</i>

`method)`, 250
`get_feature_fit()` (`datarobot.models.PrimeModel` `method`), 267
`get_feature_fit()` (`datarobot.models.RatingTableModel` `method`), 328
`get_feature_fit_metadata()` (`datarobot.models.BlenderModel` `method`), 284
`get_feature_fit_metadata()` (`datarobot.models.DatetimeModel` `method`), 302
`get_feature_fit_metadata()` (`datarobot.models.Model` `method`), 249
`get_feature_fit_metadata()` (`datarobot.models.PrimeModel` `method`), 268
`get_feature_fit_metadata()` (`datarobot.models.RatingTableModel` `method`), 328
`get_feature_impact()` (`datarobot.CustomInferenceImage` `method`), 166
`get_feature_impact()` (`datarobot.models.BlenderModel` `method`), 284
`get_feature_impact()` (`datarobot.models.DatetimeModel` `method`), 309
`get_feature_impact()` (`datarobot.models.Model` `method`), 247
`get_feature_impact()` (`datarobot.models.PrimeModel` `method`), 268
`get_feature_impact()` (`datarobot.models.RatingTableModel` `method`), 328
`get_featurelists()` (`datarobot.Dataset` `method`), 194
`get_featurelists()` (`datarobot.models.Project` `method`), 393
`get_features()` (`datarobot.Deployment` `method`), 201
`get_features()` (`datarobot.models.Project` `method`), 392
`get_features_used()` (`datarobot.models.BlenderModel` `method`), 285
`get_features_used()` (`datarobot.models.DatetimeModel` `method`), 309
`get_features_used()` (`datarobot.models.Model` `method`), 242
`get_features_used()` (`datarobot.models.PrimeModel` `method`), 269
`get_features_used()` (`datarobot.models.RatingTableModel` `method`), 329
`get_file()` (`datarobot.Dataset` `method`), 195
`get_frozen_child_models()` (`datarobot.models.BlenderModel` `method`), 285
`get_frozen_child_models()` (`datarobot.models.DatetimeModel` `method`), 310
`get_frozen_child_models()` (`datarobot.models.Model` `method`), 257
`get_frozen_child_models()` (`datarobot.models.PrimeModel` `method`), 269
`get_frozen_child_models()` (`datarobot.models.RatingTableModel` `method`), 329
`get_frozen_models()` (`datarobot.models.Project` `method`), 401
`get_histogram()` (`datarobot.models.DatasetFeature` `method`), 222
`get_leaderboard_ui_permalink()` (`datarobot.models.BlenderModel` `method`), 285
`get_leaderboard_ui_permalink()` (`datarobot.models.DatetimeModel` `method`), 310
`get_leaderboard_ui_permalink()` (`datarobot.models.Model` `method`), 243
`get_leaderboard_ui_permalink()` (`datarobot.models.PrimeModel` `method`), 269
`get_leaderboard_ui_permalink()` (`datarobot.models.Project` `method`), 402
`get_leaderboard_ui_permalink()` (`datarobot.models.RatingTableModel` `method`), 329
`get_lift_chart()` (`datarobot.models.BlenderModel` `method`), 285
`get_lift_chart()` (`datarobot.models.DatetimeModel` `method`), 310
`get_lift_chart()` (`datarobot.models.Model` `method`), 253
`get_lift_chart()` (`datarobot.models.PrimeModel` `method`), 269
`get_lift_chart()` (`datarobot.models.RatingTableModel` `method`), 329
`get_log()` (`datarobot.CustomModelTest` `method`), 172
`get_log()` (`datarobot.CustomModelVersionDependencyBuild` `method`), 176
`get_log_tail()` (`datarobot.CustomModelTest`

`method)`, 172
`get_metrics()` (`datarobot.models.Project` method), 396
`get_missing_report_info()` (`datarobot.models.BlenderModel` method), 286
`get_missing_report_info()` (`datarobot.models.DatetimeModel` method), 310
`get_missing_report_info()` (`datarobot.models.Model` method), 256
`get_missing_report_info()` (`datarobot.models.PrimeModel` method), 269
`get_missing_report_info()` (`datarobot.models.RatingTableModel` method), 330
`get_model()` (`datarobot.models.ModelJob` class method), 343
`get_model()` (`datarobot.models.ModelRecommendation` method), 417
`get_model_blueprint_chart()` (`datarobot.models.BlenderModel` method), 286
`get_model_blueprint_chart()` (`datarobot.models.DatetimeModel` method), 310
`get_model_blueprint_chart()` (`datarobot.models.Model` method), 256
`get_model_blueprint_chart()` (`datarobot.models.PrimeModel` method), 270
`get_model_blueprint_chart()` (`datarobot.models.RatingTableModel` method), 330
`get_model_blueprint_documents()` (`datarobot.models.BlenderModel` method), 286
`get_model_blueprint_documents()` (`datarobot.models.DatetimeModel` method), 310
`get_model_blueprint_documents()` (`datarobot.models.Model` method), 256
`get_model_blueprint_documents()` (`datarobot.models.PrimeModel` method), 270
`get_model_blueprint_documents()` (`datarobot.models.RatingTableModel` method), 330
`get_model_jobs()` (`datarobot.models.Project` method), 401
`get_modeling_featurelists()` (`datarobot.models.Project` method), 394
`get_modeling_features()` (`datarobot.models.Project` method), 392
`get_models()` (`datarobot.models.Project` method), 389
`get_multiclass_feature_impact()` (`datarobot.models.BlenderModel` method), 286
`get_multiclass_feature_impact()` (`datarobot.models.DatetimeModel` method), 310
`get_multiclass_feature_impact()` (`datarobot.models.Model` method), 247
`get_multiclass_feature_impact()` (`datarobot.models.PrimeModel` method), 270
`get_multiclass_feature_impact()` (`datarobot.models.RatingTableModel` method), 330
`get_multiclass_lift_chart()` (`datarobot.models.BlenderModel` method), 286
`get_multiclass_lift_chart()` (`datarobot.models.DatetimeModel` method), 311
`get_multiclass_lift_chart()` (`datarobot.models.Model` method), 254
`get_multiclass_lift_chart()` (`datarobot.models.PrimeModel` method), 270
`get_multiclass_lift_chart()` (`datarobot.models.RatingTableModel` method), 330
`get_multiseries_properties()` (`datarobot.models.Feature` method), 219
`get_num_iterations_trained()` (`datarobot.models.BlenderModel` method), 287
`get_num_iterations_trained()` (`datarobot.models.DatetimeModel` method), 311
`get_num_iterations_trained()` (`datarobot.models.Model` method), 243
`get_num_iterations_trained()` (`datarobot.models.PrimeModel` method), 270
`get_num_iterations_trained()` (`datarobot.models.RatingTableModel` method), 331
`get_optimized()` (`datarobot.DatetimePartitioning` class method), 357
`get_or_request_feature_effect()` (`datarobot.models.BlenderModel` method), 287
`get_or_request_feature_effect()` (`datarobot.models.DatetimeModel` method),

[303](#)
`get_or_request_feature_effect()`
 (*datarobot.models.Model* method), [250](#)
`get_or_request_feature_effect()`
 (*datarobot.models.PrimeModel* method), [271](#)
`get_or_request_feature_effect()`
 (*datarobot.models.RatingTableModel* method), [331](#)
`get_or_request_feature_fit()`
 (*datarobot.models.BlenderModel* method), [287](#)
`get_or_request_feature_fit()`
 (*datarobot.models.DatetimeModel* method), [304](#)
`get_or_request_feature_fit()`
 (*datarobot.models.Model* method), [251](#)
`get_or_request_feature_fit()`
 (*datarobot.models.PrimeModel* method), [271](#)
`get_or_request_feature_fit()`
 (*datarobot.models.RatingTableModel* method), [331](#)
`get_or_request_feature_impact()`
 (*datarobot.models.BlenderModel* method), [288](#)
`get_or_request_feature_impact()`
 (*datarobot.models.DatetimeModel* method), [311](#)
`get_or_request_feature_impact()`
 (*datarobot.models.Model* method), [248](#)
`get_or_request_feature_impact()`
 (*datarobot.models.PrimeModel* method), [271](#)
`get_or_request_feature_impact()`
 (*datarobot.models.RatingTableModel* method), [332](#)
`get_parameter_names()`
 (*datarobot.models.advanced_tuning.AdvancedTuningSession* method), [341](#)
`get_parameters()` (*datarobot.models.advanced_tuning.AdvancedTuningSession* method), [341](#)
`get_parameters()` (*datarobot.models.BlenderModel* method), [288](#)
`get_parameters()` (*datarobot.models.DatetimeModel* method), [312](#)
`get_parameters()` (*datarobot.models.Model* method), [253](#)
`get_parameters()` (*datarobot.models.PrimeModel* method), [271](#)
`get_parameters()` (*datarobot.models.RatingTableModel* method), [332](#)
`get_pareto_front()`
 (*datarobot.models.BlenderModel* method), [288](#)
`get_pareto_front()`
 (*datarobot.models.DatetimeModel* method), [312](#)
`get_pareto_front()` (*datarobot.models.Model* method), [255](#)
`get_pareto_front()` (*datarobot.models.PrimeModel* method), [271](#)
`get_pareto_front()` (*datarobot.models.RatingTableModel* method), [332](#)
`get_predict_jobs()` (*datarobot.models.Project* method), [401](#)
`get_prediction_explanations_page()`
 (*datarobot.PredictionExplanations* method), [373](#)
`get_prediction_intervals_settings()`
 (*datarobot.Deployment* method), [205](#)
`get_prediction_warning_settings()`
 (*datarobot.Deployment* method), [204](#)
`get_predictions()` (*datarobot.models.PredictJob* class method), [365](#)
`get_predictions_data_collection_settings()`
 (*datarobot.Deployment* method), [203](#)
`get_prime_eligibility()`
 (*datarobot.models.BlenderModel* method), [288](#)
`get_prime_eligibility()`
 (*datarobot.models.DatetimeModel* method), [312](#)
`get_prime_eligibility()`
 (*datarobot.models.Model* method), [251](#)
`get_prime_eligibility()`
 (*datarobot.models.PrimeModel* method), [272](#)
`get_prime_eligibility()`
 (*datarobot.models.RatingTableModel* method), [332](#)
`get_prime_files()` (*datarobot.models.Project* method), [390](#)
`get_prime_models()` (*datarobot.models.Project* method), [390](#)
`get_projects()` (*datarobot.Dataset* method), [195](#)
`get_rating_table_models()`
 (*datarobot.models.Project* method), [403](#)
`get_rating_tables()` (*datarobot.models.Project* method), [403](#)
`get_reason_codes_page()`
 (*datarobot.ReasonCodes* method), [415](#)
`get_recommendation()`
 (*datarobot.models.ModelRecommendation* class method), [417](#)
`get_relationships_configuration()`

[\(datarobot.models.Project method\), 405](#)
[get_residuals_chart\(\) \(datarobot.models.BlenderModel method\), 288](#)
[get_residuals_chart\(\) \(datarobot.models.DatetimeModel method\), 312](#)
[get_residuals_chart\(\) \(datarobot.models.Model method\), 254](#)
[get_residuals_chart\(\) \(datarobot.models.PrimeModel method\), 272](#)
[get_residuals_chart\(\) \(datarobot.models.RatingTableModel method\), 332](#)
[get_result\(\) \(datarobot.models.FeatureImpactJob method\), 238](#)
[get_result\(\) \(datarobot.models.Job method\), 233](#)
[get_result\(\) \(datarobot.models.ModelJob method\), 343](#)
[get_result\(\) \(datarobot.models.PredictJob method\), 366](#)
[get_result\(\) \(datarobot.models.ShapMatrixJob method\), 236](#)
[get_result\(\) \(datarobot.models.TrainingPredictionsJob method\), 235](#)
[get_result_when_complete\(\) \(datarobot.models.FeatureImpactJob method\), 239](#)
[get_result_when_complete\(\) \(datarobot.models.Job method\), 234](#)
[get_result_when_complete\(\) \(datarobot.models.ModelJob method\), 344](#)
[get_result_when_complete\(\) \(datarobot.models.PredictJob method\), 367](#)
[get_result_when_complete\(\) \(datarobot.models.ShapMatrixJob method\), 237](#)
[get_result_when_complete\(\) \(datarobot.models.TrainingPredictionsJob method\), 236](#)
[get_roc_curve\(\) \(datarobot.models.BlenderModel method\), 288](#)
[get_roc_curve\(\) \(datarobot.models.DatetimeModel method\), 312](#)
[get_roc_curve\(\) \(datarobot.models.Model method\), 255](#)
[get_roc_curve\(\) \(datarobot.models.PrimeModel method\), 272](#)
[get_roc_curve\(\) \(datarobot.models.RatingTableModel method\), 332](#)
[get_rows\(\) \(datarobot.PredictionExplanations method\), 371](#)
[get_rows\(\) \(datarobot.ReasonCodes method\), 413](#)
[get_rulesets\(\) \(datarobot.models.BlenderModel method\), 289](#)
[get_rulesets\(\) \(datarobot.models.DatetimeModel method\), 313](#)
[get_rulesets\(\) \(datarobot.models.Model method\), 251](#)
[get_rulesets\(\) \(datarobot.models.PrimeModel method\), 272](#)
[get_rulesets\(\) \(datarobot.models.RatingTableModel method\), 333](#)
[get_series_accuracy_as_dataframe\(\) \(datarobot.models.DatetimeModel method\), 300](#)
[get_service_stats\(\) \(datarobot.Deployment method\), 205](#)
[get_service_stats_over_time\(\) \(datarobot.Deployment method\), 206](#)
[get_status\(\) \(datarobot.models.BatchPredictionJob method\), 150](#)
[get_status\(\) \(datarobot.models.Project method\), 396](#)
[get_supported_capabilities\(\) \(datarobot.models.BlenderModel method\), 289](#)
[get_supported_capabilities\(\) \(datarobot.models.DatetimeModel method\), 313](#)
[get_supported_capabilities\(\) \(datarobot.models.Model method\), 243](#)
[get_supported_capabilities\(\) \(datarobot.models.PrimeModel method\), 273](#)
[get_supported_capabilities\(\) \(datarobot.models.RatingTableModel method\), 333](#)
[get_target_drift\(\) \(datarobot.Deployment method\), 206](#)
[get_task_names\(\) \(datarobot.models.advanced_tuning.AdvancedTuning method\), 341](#)
[get_word_cloud\(\) \(datarobot.models.BlenderModel method\), 289](#)
[get_word_cloud\(\) \(datarobot.models.DatetimeModel method\), 313](#)
[get_word_cloud\(\) \(datarobot.models.Model method\), 256](#)
[get_word_cloud\(\) \(datarobot.models.PrimeModel method\), 273](#)
[get_word_cloud\(\) \(datarobot.models.RatingTableModel method\), 333](#)
[GroupCV \(class in datarobot\), 346](#)
[GroupTVH \(class in datarobot\), 347](#)
[Image \(class in datarobot.models.visualai\), 406](#)

- ImageActivationMap (class in `datarobot.models.visualai`), 408
- ImageEmbedding (class in `datarobot.models.visualai`), 407
- ImportedModel (class in `datarobot.models`), 319
- InteractionFeature (class in `datarobot.models`), 223
- iterate() (`datarobot.Dataset` class method), 193
- iterate_all_features() (`datarobot.Dataset` method), 194
- iterate_rows() (`datarobot.models.training_predictions.TrainingPredictions` class method), 422
- J**
- Job (class in `datarobot.models`), 233
- L**
- LiftChart (class in `datarobot.models.lift_chart`), 239
- list() (`datarobot.CalendarFile` class method), 155
- list() (`datarobot.CustomInferenceImage` class method), 165
- list() (`datarobot.CustomInferenceModel` class method), 167
- list() (`datarobot.CustomModelTest` class method), 171
- list() (`datarobot.CustomModelVersion` class method), 174
- list() (`datarobot.DataDriver` class method), 181
- list() (`datarobot.Dataset` class method), 192
- list() (`datarobot.DataSource` class method), 186
- list() (`datarobot.DataStore` class method), 182
- list() (`datarobot.Deployment` class method), 199
- list() (`datarobot.ExecutionEnvironment` class method), 177
- list() (`datarobot.ExecutionEnvironmentVersion` class method), 179
- list() (`datarobot.ExternalLiftChart` class method), 216
- list() (`datarobot.ExternalRocCurve` class method), 217
- list() (`datarobot.ExternalScores` class method), 215
- list() (`datarobot.models.compliance_doc_template.ComplianceDocTemplate` class method), 159
- list() (`datarobot.models.Credential` class method), 161
- list() (`datarobot.models.FeatureDrift` class method), 211
- list() (`datarobot.models.FeatureEngineeringGraph` class method), 228
- list() (`datarobot.models.ImportedModel` class method), 320
- list() (`datarobot.models.PayoffMatrix` class method), 362
- list() (`datarobot.models.Predictions` class method), 378
- list() (`datarobot.models.Project` class method), 386
- list() (`datarobot.models.ShapMatrix` class method), 376
- list() (`datarobot.models.training_predictions.TrainingPredictions` class method), 422
- list() (`datarobot.models.visualai.DuplicateImage` class method), 407
- list() (`datarobot.models.visualai.ImageActivationMap` class method), 408
- list() (`datarobot.models.visualai.ImageEmbedding` class method), 408
- list() (`datarobot.models.visualai.SampleImage` class method), 406
- list() (`datarobot.PredictionExplanations` class method), 371
- list() (`datarobot.PredictionServer` class method), 379
- list() (`datarobot.ReasonCodes` class method), 413
- list_by_status() (`datarobot.models.BatchPredictionJob` class method), 150
- M**
- metric_baselines (`datarobot.models.Accuracy` attribute), 213
- metric_values (`datarobot.models.Accuracy` attribute), 212
- MissingValuesReport (class in `datarobot.models.missing_report`), 240
- Model (class in `datarobot.models`), 241
- ModelBlueprintChart (class in `datarobot.models`), 152
- ModelingFeature (class in `datarobot.models`), 220
- ModelingFeaturelist (class in `datarobot.models`), 231
- ModelJob (class in `datarobot.models`), 342
- ModelRecommendation (class in `datarobot.models`), 417
- models() (`datarobot.models.visualai.ImageActivationMap` class method), 408
- models() (`datarobot.models.visualai.ImageEmbedding` class method), 407
- modify() (`datarobot.Dataset` method), 193
- most_frequent() (`datarobot.models.word_cloud.WordCloud` method), 424
- most_important() (`datarobot.models.word_cloud.WordCloud` method), 424
- N**
- ngrams_per_class() (`datarobot.models.word_cloud.WordCloud` method), 424

O

`open_leaderboard_browser()`
(*datarobot.models.Project* method), 403

`open_model_browser()`
(*datarobot.models.BlenderModel* method), 289

`open_model_browser()`
(*datarobot.models.DatetimeModel* method), 313

`open_model_browser()` (*datarobot.models.Model* method), 243

`open_model_browser()`
(*datarobot.models.PrimeModel* method), 273

`open_model_browser()`
(*datarobot.models.RatingTableModel* method), 333

P

ParetoFront (class in *datarobot.models.pareto_front*), 345

`pause_autopilot()` (*datarobot.models.Project* method), 397

PayoffMatrix (class in *datarobot.models*), 361

`percent_changes` (*datarobot.models.Accuracy* attribute), 213

Periodicity (class in *datarobot*), 352

PredictionDataset (class in *datarobot.models*), 367

PredictionExplanations (class in *datarobot*), 370

PredictionExplanationsInitialization (class in *datarobot*), 369

PredictionExplanationsPage (class in *datarobot.models.prediction_explanations*), 374

PredictionExplanationsRow (class in *datarobot.models.prediction_explanations*), 373

Predictions (class in *datarobot.models*), 377

PredictionServer (class in *datarobot*), 379

PredictJob (class in *datarobot.models*), 364

`prep_payload()` (*datarobot.DatetimePartitioningSpecification* method), 350

PrimeFile (class in *datarobot.models*), 380

PrimeModel (class in *datarobot.models*), 261

Project (class in *datarobot.models*), 381

R

RandomCV (class in *datarobot*), 346

RandomTVH (class in *datarobot*), 346

RatingTable (class in *datarobot.models*), 410

RatingTableModel (class in *datarobot.models*), 321

ReasonCodes (class in *datarobot*), 412

ReasonCodesInitialization (class in *datarobot*), 411

ReasonCodesPage (class in *datarobot.models.reason_codes*), 416

ReasonCodesRow (class in *datarobot.models.reason_codes*), 415

`refresh()` (*datarobot.CustomInferenceImage* method), 166

`refresh()` (*datarobot.CustomInferenceModel* method), 170

`refresh()` (*datarobot.CustomModelTest* method), 172

`refresh()` (*datarobot.CustomModelVersion* method), 175

`refresh()` (*datarobot.CustomModelVersionDependencyBuild* method), 176

`refresh()` (*datarobot.ExecutionEnvironment* method), 178

`refresh()` (*datarobot.ExecutionEnvironmentVersion* method), 180

`refresh()` (*datarobot.models.FeatureImpactJob* method), 239

`refresh()` (*datarobot.models.Job* method), 234

`refresh()` (*datarobot.models.ModelJob* method), 344

`refresh()` (*datarobot.models.PredictJob* method), 367

`refresh()` (*datarobot.models.Project* method), 387

`refresh()` (*datarobot.models.ShapMatrixJob* method), 236

`refresh()` (*datarobot.models.TrainingPredictionsJob* method), 235

RelationshipsConfiguration (class in *datarobot.models*), 425

`rename()` (*datarobot.models.Project* method), 402

`rename()` (*datarobot.models.RatingTable* method), 411

`replace()` (*datarobot.models.FeatureEngineeringGraph* method), 227

`replace()` (*datarobot.models.RelationshipsConfiguration* method), 428

`replace_model()` (*datarobot.Deployment* method), 200

`request_approximation()`
(*datarobot.models.BlenderModel* method), 290

`request_approximation()`
(*datarobot.models.DatetimeModel* method), 313

`request_approximation()`
(*datarobot.models.Model* method), 251

`request_approximation()`
(*datarobot.models.RatingTableModel* method), 334

`request_download_validation()`
(*datarobot.models.PrimeModel* method), 262

`request_external_test()`

<code>(datarobot.models.BlenderModel method),</code>	<code>334</code>
<code>290</code>	<code>request_frozen_datetime_model()</code>
<code>request_external_test()</code>	<code>(datarobot.models.BlenderModel method),</code>
<code>(datarobot.models.DatetimeModel method),</code>	<code>291</code>
<code>313</code>	<code>request_frozen_datetime_model()</code>
<code>request_external_test()</code>	<code>(datarobot.models.DatetimeModel method),</code>
<code>(datarobot.models.Model method),</code>	<code>314</code>
<code>248</code>	<code>request_frozen_datetime_model()</code>
<code>request_external_test()</code>	<code>(datarobot.models.Model method),</code>
<code>(datarobot.models.PrimeModel method),</code>	<code>252</code>
<code>273</code>	<code>request_frozen_datetime_model()</code>
<code>request_external_test()</code>	<code>(datarobot.models.RatingTableModel method),</code>
<code>(datarobot.models.RatingTableModel method),</code>	<code>335</code>
<code>334</code>	<code>request_frozen_model()</code>
<code>request_feature_effect()</code>	<code>(datarobot.models.BlenderModel method),</code>
<code>(datarobot.models.BlenderModel method),</code>	<code>292</code>
<code>290</code>	<code>request_frozen_model()</code>
<code>request_feature_effect()</code>	<code>(datarobot.models.Model method),</code>
<code>(datarobot.models.DatetimeModel method),</code>	<code>252</code>
<code>302</code>	<code>request_frozen_model()</code>
<code>request_feature_effect()</code>	<code>(datarobot.models.RatingTableModel method),</code>
<code>(datarobot.models.Model method),</code>	<code>336</code>
<code>249</code>	<code>request_model()</code>
<code>request_feature_effect()</code>	<code>(datarobot.models.Ruleset method),</code>
<code>(datarobot.models.PrimeModel method),</code>	<code>380</code>
<code>273</code>	<code>request_predictions()</code>
<code>request_feature_effect()</code>	<code>(datarobot.models.BlenderModel method),</code>
<code>(datarobot.models.RatingTableModel method),</code>	<code>292</code>
<code>334</code>	<code>request_predictions()</code>
<code>request_feature_fit()</code>	<code>(datarobot.models.DatetimeModel method),</code>
<code>(datarobot.models.BlenderModel method),</code>	<code>315</code>
<code>290</code>	<code>request_predictions()</code>
<code>request_feature_fit()</code>	<code>(datarobot.models.Model method),</code>
<code>(datarobot.models.DatetimeModel method),</code>	<code>246</code>
<code>303</code>	<code>request_predictions()</code>
<code>request_feature_fit()</code>	<code>(datarobot.models.PrimeModel method),</code>
<code>(datarobot.models.Model method),</code>	<code>274</code>
<code>250</code>	<code>request_predictions()</code>
<code>request_feature_fit()</code>	<code>(datarobot.models.RatingTableModel method),</code>
<code>(datarobot.models.PrimeModel method),</code>	<code>336</code>
<code>274</code>	<code>request_training_predictions()</code>
<code>request_feature_fit()</code>	<code>(datarobot.models.BlenderModel method),</code>
<code>(datarobot.models.RatingTableModel method),</code>	<code>293</code>
<code>334</code>	<code>request_training_predictions()</code>
<code>request_feature_impact()</code>	<code>(datarobot.models.DatetimeModel method),</code>
<code>(datarobot.models.BlenderModel method),</code>	<code>300</code>
<code>290</code>	<code>request_training_predictions()</code>
<code>request_feature_impact()</code>	<code>(datarobot.models.Model method),</code>
<code>(datarobot.models.DatetimeModel method),</code>	<code>257</code>
<code>314</code>	<code>request_training_predictions()</code>
<code>request_feature_impact()</code>	<code>(datarobot.models.PrimeModel method),</code>
<code>(datarobot.models.Model method),</code>	<code>275</code>
<code>248</code>	<code>request_training_predictions()</code>
<code>request_feature_impact()</code>	<code>(datarobot.models.RatingTableModel method),</code>
<code>(datarobot.models.PrimeModel method),</code>	<code>337</code>
<code>274</code>	<code>request_transferable_export()</code>
<code>request_feature_impact()</code>	<code>(datarobot.models.BlenderModel method),</code>
<code>(datarobot.models.RatingTableModel method),</code>	<code>293</code>
	<code>request_transferable_export()</code>

(*datarobot.models.DatetimeModel* method), 316
 request_transferable_export() (*datarobot.models.Model* method), 252
 request_transferable_export() (*datarobot.models.PrimeModel* method), 276
 request_transferable_export() (*datarobot.models.RatingTableModel* method), 337
 retrain() (*datarobot.models.BlenderModel* method), 294
 retrain() (*datarobot.models.DatetimeModel* method), 301
 retrain() (*datarobot.models.Model* method), 245
 retrain() (*datarobot.models.PrimeModel* method), 276
 retrain() (*datarobot.models.RatingTableModel* method), 338
 RocCurve (class in *datarobot.models.roc_curve*), 418
 Ruleset (class in *datarobot.models*), 380
 run() (*datarobot.models.advanced_tuning.AdvancedTuningSession* method), 342

S

SampleImage (class in *datarobot.models.visualai*), 406
 schemas() (*datarobot.DataStore* method), 184
 score() (*datarobot.models.BatchPredictionJob* class method), 145
 score_azure() (*datarobot.models.BatchPredictionJob* class method), 149
 score_backtests() (*datarobot.models.DatetimeModel* method), 299
 score_from_existing() (*datarobot.models.BatchPredictionJob* class method), 150
 score_gcp() (*datarobot.models.BatchPredictionJob* class method), 149
 score_s3() (*datarobot.models.BatchPredictionJob* class method), 149
 score_to_file() (*datarobot.models.BatchPredictionJob* class method), 148
 SecondaryDatasetConfigurations (class in *datarobot.models*), 424
 sections_to_json_file() (*datarobot.models.compliance_doc_template.ComplianceDocTemplate* method), 159
 ServiceStats (class in *datarobot.models*), 209
 ServiceStatsOverTime (class in *datarobot.models*), 209
 set_parameter() (*datarobot.models.advanced_tuning.AdvancedTuningSession* method), 341
 set_prediction_threshold() (*datarobot.models.BlenderModel* method), 294
 set_prediction_threshold() (*datarobot.models.DatetimeModel* method), 316
 set_prediction_threshold() (*datarobot.models.Model* method), 260
 set_prediction_threshold() (*datarobot.models.PrimeModel* method), 277
 set_prediction_threshold() (*datarobot.models.RatingTableModel* method), 338
 set_project_description() (*datarobot.models.Project* method), 402
 set_target() (*datarobot.models.Project* method), 387
 set_worker_count() (*datarobot.models.Project* method), 402
 ShapImpact (class in *datarobot.models*), 429
 ShapMatrix (class in *datarobot.models*), 375
 ShapMatrixJob (class in *datarobot.models*), 236
 share() (*datarobot.CalendarFile* class method), 156
 share() (*datarobot.DataSource* method), 188
 share() (*datarobot.DataStore* method), 185
 share() (*datarobot.models.FeatureEngineeringGraph* method), 229
 share() (*datarobot.models.Project* method), 403
 SharingAccess (class in *datarobot*), 418
 Solution (class in *datarobot.models.pareto_front*), 345
 star_model() (*datarobot.models.BlenderModel* method), 294
 star_model() (*datarobot.models.DatetimeModel* method), 316
 star_model() (*datarobot.models.Model* method), 260
 star_model() (*datarobot.models.PrimeModel* method), 277
 star_model() (*datarobot.models.RatingTableModel* method), 338
 start() (*datarobot.models.Project* class method), 385
 start_advanced_tuning_session() (*datarobot.models.BlenderModel* method), 294
 start_advanced_tuning_session() (*datarobot.models.DatetimeModel* method), 317
 start_advanced_tuning_session() (*datarobot.models.Model* method), 260
 start_advanced_tuning_session() (*datarobot.models.PrimeModel* method), 277
 start_advanced_tuning_session()

(*datarobot.models.RatingTableModel* method), 338

start_autopilot() (*datarobot.models.Project* method), 397

start_build() (*datarobot.CustomModelVersionDependencyBuild* class method), 176

StratifiedCV (class in *datarobot*), 346

StratifiedTVH (class in *datarobot*), 347

submit_actuals() (*datarobot.Deployment* method), 202

T

tables() (*datarobot.DataStore* method), 185

TargetDrift (class in *datarobot.models*), 210

test() (*datarobot.DataStore* method), 184

to_dataframe() (*datarobot.DatetimePartitioning* method), 359

to_dataframe() (*datarobot.helpers.partitioning_methods.Backtest* method), 361

to_dataset() (*datarobot.DatasetDetails* method), 197

to_graphviz() (*datarobot.models.BlueprintChart* method), 152

to_graphviz() (*datarobot.models.ModelBlueprintChart* method), 153

to_specification() (*datarobot.DatetimePartitioning* method), 358

to_specification() (*datarobot.helpers.partitioning_methods.Backtest* method), 360

train() (*datarobot.models.BlenderModel* method), 295

train() (*datarobot.models.Model* method), 244

train() (*datarobot.models.Project* method), 398

train() (*datarobot.models.RatingTableModel* method), 339

train_datetime() (*datarobot.models.BlenderModel* method), 296

train_datetime() (*datarobot.models.DatetimeModel* method), 317

train_datetime() (*datarobot.models.Model* method), 245

train_datetime() (*datarobot.models.Project* method), 399

train_datetime() (*datarobot.models.RatingTableModel* method), 340

TrainingPredictions (class in *datarobot.models.training_predictions*), 420

TrainingPredictionsIterator (class in *datarobot.models.training_predictions*), 419

TrainingPredictionsJob (class in *datarobot.models*), 234

U

un_delete() (*datarobot.Dataset* class method), 192

unlock_holdout() (*datarobot.models.Project* method), 402

unpause_autopilot() (*datarobot.models.Project* method), 397

unstar_model() (*datarobot.models.BlenderModel* method), 296

unstar_model() (*datarobot.models.DatetimeModel* method), 318

unstar_model() (*datarobot.models.Model* method), 260

unstar_model() (*datarobot.models.PrimeModel* method), 277

unstar_model() (*datarobot.models.RatingTableModel* method), 340

update() (*datarobot.CustomInferenceModel* method), 169

update() (*datarobot.CustomModelVersion* method), 175

update() (*datarobot.DataDriver* method), 182

update() (*datarobot.Dataset* method), 193

update() (*datarobot.DatasetFeaturelist* method), 230

update() (*datarobot.DataSource* method), 187

update() (*datarobot.DataStore* method), 183

update() (*datarobot.Deployment* method), 200

update() (*datarobot.ExecutionEnvironment* method), 178

update() (*datarobot.models.compliance_doc_template.ComplianceDocTemplate* method), 159

update() (*datarobot.models.FeatureEngineeringGraph* method), 228

update() (*datarobot.models.Featurelist* method), 231

update() (*datarobot.models.ImportedModel* method), 320

update() (*datarobot.models.ModelingFeaturelist* method), 233

update() (*datarobot.models.PayoffMatrix* class method), 363

update_association_id_settings() (*datarobot.Deployment* method), 203

update_drift_tracking_settings() (*datarobot.Deployment* method), 203

update_name() (*datarobot.CalendarFile* class method), 155

update_prediction_intervals_settings() (*datarobot.Deployment* method), 205

update_prediction_warning_settings() (*datarobot.Deployment* method), 204

update_predictions_data_collection_settings() (*datarobot.Deployment* method), 204

upload_dataset() (*datarobot.models.Project* method), 390

upload_dataset_from_data_source()

(*datarobot.models.Project* method), 391

UserCV (*class in datarobot*), 346

UserTVH (*class in datarobot*), 346

V

validate_replacement_model()
(*datarobot.Deployment* method), 201

W

wait_for_async_model_creation() (*in module datarobot.models.modeljob*), 342

wait_for_async_predictions() (*in module datarobot.models.predict_job*), 364

wait_for_autopilot() (*datarobot.models.Project* method), 402

wait_for_completion()
(*datarobot.models.FeatureImpactJob* method), 239

wait_for_completion() (*datarobot.models.Job* method), 234

wait_for_completion()
(*datarobot.models.ModelJob* method), 344

wait_for_completion()
(*datarobot.models.PredictJob* method), 367

wait_for_completion()
(*datarobot.models.ShapMatrixJob* method), 237

wait_for_completion()
(*datarobot.models.TrainingPredictionsJob* method), 236

WordCloud (*class in datarobot.models.word_cloud*), 423