Package 'datarobot'

November 19, 2018

Title 'DataRobot' Predictive Modeling API

Version 2.10.0

Description For working with the 'DataRobot' predictive modeling platform's API <https://www.datarobot.com/>.

Depends R (>= 3.1.1), methods, stats

Imports httr (>= 1.0), jsonlite (>= 1.0), yaml (>= 2.1.13), curl (>= 1.1)

License MIT + file LICENSE

LazyData true

Author Ron Pearson [aut], Zachary Deane-Mayer [aut], David Chudzicki [aut], Dallin Akagi [aut], Sergey Yurgenson [aut], Thakur Raj Anand [aut], Peter Hurford [aut]

Maintainer Peter Hurford <api-maintainer@datarobot.com>

Suggests knitr, testthat, lintr, stubthat, data.table, car, MASS, mlbench, beanplot, doBy, insuranceData, rmarkdown, ggplot2, colormap, modelwordcloud

VignetteBuilder car, MASS, mlbench, beanplot, doBy, insuranceData, rmarkdown, knitr, data.table, ggplot2, colormap, modelwordcloud

RoxygenNote 6.1.0

NeedsCompilation no

Repository CRAN

Date/Publication 2018-11-19 05:20:03 UTC

R topics documented:

`datarobot-package` *datarobot: 'DataRobot' Predictive Modeling API*

Description

For working with the 'DataRobot' predictive modeling platform's API <https://www.datarobot.com/>.

`ApplySchema` *Apply a schema to DataRobot objects (lists, frames)*

Description

Apply a schema to DataRobot objects (lists, frames)

Usage

```
ApplySchema(inList, schema, mask = NULL)
```

Arguments

`inList`	object. The DataRobot object to apply the schema to.
`schema`	list. The schema to apply.
`mask`	list. Search the schema and only apply values that match this with grep. Defaults to NULL, or no masking.

`as.data.frame` *DataRobot S3 object methods for R's generic as.data.frame function*

Description

These functions extend R's generic as.data.frame function to the DataRobot S3 object classes listOfBlueprints, listOfFeaturelists, listOfModels, and projectSummaryList.

Usage

```
## S3 method for class 'listOfBlueprints'
as.data.frame(x, row.names = NULL,
  optional = FALSE, ...)

## S3 method for class 'listOfFeaturelists'
as.data.frame(x, row.names = NULL,
  optional = FALSE, ...)

## S3 method for class 'listOfModels'
as.data.frame(x, row.names = NULL, optional = FALSE,
  simple = TRUE, ...)

## S3 method for class 'projectSummaryList'
as.data.frame(x, row.names = NULL,
  optional = FALSE, simple = TRUE, ...)

## S3 method for class 'listOfDataRobotPredictionDatasets'
as.data.frame(x, row.names = NULL,
  optional = FALSE, ...)
```

Arguments

`x`	S3 object to be converted into a dataframe.
`row.names`	character. Optional. Row names for the dataframe returned by the method.
`optional`	logical. Optional. If TRUE, setting row names and converting column names to syntactic names: see help for `make.names` function.
`...`	list. Additional optional parameters to be passed to the generic as.data.frame function (not used at present).
`simple`	logical. Optional. if TRUE (the default), a simplified dataframe is returned for objects of class listOfModels or projectSummaryList.

Details

All of the DataRobot S3 'listOf' class objects have relatively complex structures and are often easier to work with as dataframes. The methods described here extend R's generic as.data.frame

function to convert objects of these classes to convenient dataframes. For objects of class listOfBlueprints and listOfFeaturelists or objects of class listOfModels and projectSummaryList with simple = FALSE, the dataframes contain all information from the original S3 object. The default value simple = TRUE provides simpler dataframes for objects of class listOfModels and projectSummaryList.

Value

A dataframe containing some or all of the data from the original S3 object; see Details.

`AutopilotMode` *Autopilot modes*

Description

This is a list that contains the valid values for autopilot mode. If you wish, you can specify autopilot modes using the list values, e.g. AutopilotMode$FullAuto instead of typing the string 'auto'. This way you can benefit from autocomplete and not have to remember the valid optons.

Usage

```
AutopilotMode
```

Format

An object of class `list` of length 3.

`BlendMethods` *Blend methods*

Description

This is a list that contains the valid values for Blend methods

Usage

```
BlendMethods
```

Format

An object of class `list` of length 10.

BlueprintChartToGraphviz *Convert a blueprint chart into graphviz DOT format*

Description

Convert a blueprint chart into graphviz DOT format

Usage

```
BlueprintChartToGraphviz(blueprintChart)
```

Arguments

blueprintChart	list. The list returned by GetBlueprintChart function.

Value

Character string representation of chart in graphviz DOT language.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  model <- GetModel(projectId, modelId)
  blueprintId <- model$blueprintId
  blueprintChart <- GetBlueprintChart(projectId, blueprintId)
  BlueprintChartToGraphviz(blueprintChart)

## End(Not run)
```

CleanServerData *Reformat paginated data returned from the server.*

Description

Reformat paginated data returned from the server.

Usage

```
CleanServerData(serverData)
```

Arguments

serverData	list. Raw JSON parsed list returned from the server.

ConnectToDataRobot *Establish a connection to the DataRobot modeling engine*

Description

This function initializes a DataRobot session. If a (YAML) config file (with keys for endpoint and token) is placed at $HOME/.config/datarobot/drconfig.yaml, then we attempt to establish a connection to DataRobot when the package loads, so (if successful) this function does not need to be called.

Usage

```
ConnectToDataRobot(endpoint = NULL, token = NULL, username = NULL,
  password = NULL, userAgentSuffix = NULL, sslVerify = TRUE,
  configPath = NULL)
```

Arguments

endpoint	character. URL specifying the DataRobot server to be used. It depends on DataRobot modeling engine implementation (cloud-based, on-prem...) you are using. Contact your DataRobot admin for endpoint to use and to turn on API access to your account. The endpoint for DataRobot cloud accounts is https://app.datarobot.com/api/v2
token	character. DataRobot API access token. It is unique for each DataRobot modeling engine account and can be accessed using DataRobot webapp in Account profile section.
username	character. No longer supported.
password	character. No longer supported.
userAgentSuffix	character. Additional text that is appended to the User-Agent HTTP header when communicating with the DataRobot REST API. This can be useful for identifying different applications that are built on top of the DataRobot Python Client, which can aid debugging and help track usage.
sslVerify	logical. Whether to check the SSL certificate. Either TRUE to check (default), FALSE to not check.
configPath	character. Path to YAML config file specifying configuration (token and endpoint).

Details

The function creates the environment variables "DataRobot_URL" and "DataRobot_Token" used by other functions to access the DataRobot modeling engine.

Examples

```
## Not run:
  ConnectToDataRobot("https://app.datarobot.com/api/v2", "thisismyfaketoken")
  ConnectToDataRobot(configPath = "~/.config/datarobot/drconfig.yaml")

## End(Not run)
```

ConstructDurationString

Construct a valid string representing a duration in accordance with ISO8601

Description

A duration of six months, 3 days, and 12 hours could be represented as P6M3DT12H.

Usage

```
ConstructDurationString(years = 0, months = 0, days = 0, hours = 0,
  minutes = 0, seconds = 0)
```

Arguments

years	integer. The number of years in the duration.
months	integer. The number of months in the duration.
days	integer. The number of days in the duration.
hours	integer. The number of hours in the duration.
minutes	integer. The number of minutes in the duration.
seconds	integer. The number of seconds in the duration.

Value

The duration string, specified compatibly with ISO8601.

Examples

```
ConstructDurationString()
ConstructDurationString(days = 100)
ConstructDurationString(years = 10, months = 2, days = 5, seconds = 12)
```

`CreateBacktestSpecification` *Create a list describing backtest parameters*

Description

Uniquely defines a Backtest used in a DatetimePartitioning

Usage

```
CreateBacktestSpecification(index, gapDuration, validationStartDate,
  validationDuration)
```

Arguments

`index`	integer. The index of the backtest
`gapDuration`	character. The desired duration of the gap between training and validation data for the backtest in duration format (ISO8601).
`validationStartDate`	character. The desired start date of the validation data for this backtest (RFC 3339 format).
`validationDuration`	character. The desired end date of the validation data for this backtest in duration format (ISO8601).

Details

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. All durations should be specified with a duration string such as those returned by the ConstructDurationString helper function.

Value

list with backtest parameters

Examples

```
zeroDayDuration <- ConstructDurationString()
hundredDayDuration <- ConstructDurationString(days = 100)
CreateBacktestSpecification(index = 0,
                            gapDuration = zeroDayDuration,
                            validationStartDate = "1989-12-01",
                            validationDuration = hundredDayDuration)
```

`CreateDatetimePartitionSpecification` *Create a list describing datetime partition parameters*

Description

Uniquely defines a DatetimePartitioning for some project

Usage

```
CreateDatetimePartitionSpecification(datetimePartitionColumn,
  autopilotDataSelectionMethod = NULL, validationDuration = NULL,
  holdoutStartDate = NULL, holdoutDuration = NULL, disableHoldout = NULL,
  gapDuration = NULL, numberOfBacktests = NULL, backtests = NULL,
  useTimeSeries = FALSE, defaultToAPriori = FALSE,
  defaultToKnownInAdvance = FALSE, featureDerivationWindowStart = NULL,
  featureDerivationWindowEnd = NULL, featureSettings = NULL,
  treatAsExponential = NULL, differencingMethod = NULL,
  periodicities = NULL, forecastWindowStart = NULL,
  forecastWindowEnd = NULL)
```

Arguments

`datetimePartitionColumn`
: character. The name of the column whose values as dates are used to assign a row to a particular partition

`autopilotDataSelectionMethod`
: character. Optional. Whether models created by the autopilot should use "rowCount" or "duration" as their dataSelectionMethod

`validationDuration`
: character. Optional. The default validationDuration for the backtests

`holdoutStartDate`
: character. The start date of holdout scoring data (RFC 3339 format). If holdoutStartDuration is specified, holdoutDuration must also be specified.

`holdoutDuration`
: character. Optional. The duration of the holdout scoring data. If holdoutDuration is specified, holdoutStartDate must also be specified.

`disableHoldout`
: logical. Optional. Whether to suppress allocating the holdout fold. If set to TRUE, holdoutStartDate and holdoutDuration must not be specified.

`gapDuration`
: character. Optional. The duration of the gap between training and holdout scoring data.

`numberOfBacktests`
: integer. The number of backtests to use.

`backtests`
: list. List of BacktestSpecification the exact specification of backtests to use. The indexes of the specified backtests should range from 0 to numberOfBacktests - 1. If any backtest is left unspecified, a default configuration will be chosen.

`useTimeSeries` logical. 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.

`defaultToAPriori`
logical. Deprecated prior name of `defaultToKnownInAdvance`. Will be removed in v2.15.

`defaultToKnownInAdvance`
logical. Whether to default to treating features as known in advance. Defaults to FALSE. Only used for time series project. Known in advance features are expected to be known for dates in the future when making predictions (e.g., "is this a holiday").

`featureDerivationWindowStart`
integer. Optional. Offset into the past to define how far back relative to the forecast point the feature derivation window should start. Only used for time series projects. Expressed in terms of the `timeUnit` of the `datetimePartitionColumn`.

`featureDerivationWindowEnd`
integer. Optional. Offset into the past to define how far back relative to the forecast point the feature derivation window should end. Only used for time series projects. Expressed in terms of the `timeUnit` of the `datetimePartitionColumn`.

`featureSettings`
list. Optional. A list specifying settings for each feature.

`treatAsExponential`
string. Optional. Defaults to "auto". Used to specify whether to treat data as exponential trend and apply transformations like log-transform. Use values from `TreatAsExponential` enum.

`differencingMethod`
string. Optional. Defaults to "auto". Used to specify differencing method to apply of case if data is stationary. Use values from `DifferencingMethod`.

`periodicities` list. Optional. A list of periodicities for different times. Must be specified as a list of lists, where each list item specifies the ‘timeSteps‘ for a particular ‘timeUnit‘.

`forecastWindowStart`
integer. Optional. Offset into the future to define how far forward relative to the forceast point the forecaset window should start. Only used for time series projects. Expressed in terms of the `timeUnit` of the `datetimePartitionColumn`.

`forecastWindowEnd`
integer. Optional. Offset into the future to define how far forward relative to the forceast point the forecaset window should end. Only used for time series projects. Expressed in terms of the `timeUnit` of the `datetimePartitionColumn`.

Details

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 SetTarget via the partition parameter. To see the full partitioning based on the project dataset, GenerateDatetimePartition. All durations should be specified with a duration string such as those returned by the ConstructDurationString helper function.

Value

An S3 object of class 'partition' including the parameters required by the SetTarget function to generate a datetime partitioning of the modeling dataset.

Examples

```
CreateDatetimePartitionSpecification("date_col")
CreateDatetimePartitionSpecification("date",
                                     featureSettings = list(
                                       list("featureName" = "Product_offers",
                                            "knownInAdvance" = TRUE)))
partition <- CreateDatetimePartitionSpecification("dateColumn",
                                       treatAsExponential = TreatAsExponential$Always,
                                      differencingMethod = DifferencingMethod$Seasonal,
                                            periodicities = list(list("timeSteps" = 10,
                                                                      "timeUnit" = "HOUR"),
                                                                 list("timeSteps" = 600,
                                                                 "timeUnit" = "MINUTE"),
                                                                   list("timeSteps" = 7,
                                                                   "timeUnit" = "DAY")))
```

CreateDerivedFeatures *Derived Features*

Description

These functions request that new features be created as transformations of existing features and wait for the new feature to be created.

Usage

```
CreateDerivedFeatureAsCategorical(project, parentName, name = NULL,
  dateExtraction = NULL, replacement = NULL, maxWait = 600)

CreateDerivedFeatureAsText(project, parentName, name = NULL,
  dateExtraction = NULL, replacement = NULL, maxWait = 600)

CreateDerivedFeatureAsNumeric(project, parentName, name = NULL,
  dateExtraction = NULL, replacement = NULL, maxWait = 600)

CreateDerivedFeatureIntAsCategorical(project, parentName, name = NULL,
  dateExtraction = NULL, replacement = NULL, maxWait = 600)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
parentName	The name of the parent feature.

name	The name of the new feature.
dateExtraction	dateExtraction: The value to extract from the date column: 'year', 'yearDay', 'month', 'monthDay', 'week', or 'weekDay'. Required for transformation of a date column. Otherwise must not be provided.
replacement	The replacement in case of a failed transformation. Optional.
maxWait	The maximum time (in seconds) to wait for feature creation.

Value

Details for the created feature; same schema as the object returned from GetFeatureInfo.

`CreateFeaturelist` *Create a new featurelist in a DataRobot project*

Description

This function allows the user to create a new featurelist in a project by specifying its name and a list of variables to be included

Usage

```
CreateFeaturelist(project, listName, featureNames)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
listName	character. String identifying the new featurelist to be created.
featureNames	character. Vector listing the names of the variables to be included in the featurelist.

Details

DataRobot featurelists define the variables from the modeling dataset used in fitting each project model. Some functions (SetTarget, StartNewAutopilot) optionally accept a featurelist (and use a default featurelist if none is specified).

Value

A list with the following four elements describing the featurelist created:

featurelistId Character string giving the unique alphanumeric identifier for the new featurelist.

projectId Character string giving the projectId identifying the project to which the featurelist was added.

features Character vector with the names of the variables included in the new featurelist.

name Character string giving the name of the new featurelist.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  CreateFeaturelist(projectId, "myFeaturelist", c("feature1", "feature2", "otherFeature"))

## End(Not run)
```

`CreateGroupPartition` *Create a group-based S3 object of class partition for the SetTarget function*

Description

Group partitioning constructs data partitions such that all records with each level in the column specified by the parameter partitionKeyCols occur together in the same partition.

Usage

```
CreateGroupPartition(validationType, holdoutPct, partitionKeyCols,
  reps = NULL, validationPct = NULL)
```

Arguments

`validationType`	character. String specifying the type of partition generated, either "TVH" or "CV".
`holdoutPct`	integer. The percentage of data to be used as the holdout subset.
`partitionKeyCols`	list. List containing a single string specifying the name of the variable used in defining the group partition.
`reps`	integer. The number of cross-validation folds to generate; only applicable when validationType = "CV".
`validationPct`	integer. The percentage of data to be used as the validation subset.

Details

This function is one of several convenience functions provided to simplify the task of starting modeling projects with custom partitioning options. The other functions are `CreateRandomPartition`, `CreateStratifiedPartition`, and `CreateUserPartition`.

Value

An S3 object of class 'partition' including the parameters required by the SetTarget function to generate a group-based partitioning of the modeling dataset.

See Also

`CreateRandomPartition`, `CreateStratifiedPartition`, `CreateUserPartition`.

Examples

```
CreateGroupPartition(validationType = "CV",
                     holdoutPct = 20,
                     partitionKeyCols = list("groupId"),
                     reps = 5)
```

`CreateModelingFeaturelist`

This function allows the user to create a new featurelist in a project by specifying its name and a list of variables to be included

---

Description

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.

Usage

```
CreateModelingFeaturelist(project, listName, featureNames)
```

Arguments

`project`	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
`listName`	character. String identifying the new featurelist to be created.
`featureNames`	character. Vector listing the names of the variables to be included in the featurelist.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  CreateModelingFeaturelist(projectId, "myFeaturelist", c("feature1", "feature2"))

## End(Not run)
```

CreatePrimeCode *Create and validate the downloadable code for the ruleset associated with this model*

Description

Create and validate the downloadable code for the ruleset associated with this model

Usage

```
CreatePrimeCode(project, primeModelId, language)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
primeModelId	character. Id returned by GetPrimeModel(s) functions.
language	character. Programming language to use for downloadable code (see PrimeLanguage).

Value

job Id

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  CreatePrimeCode(projectId, modelId, "Python")

## End(Not run)
```

CreateRandomPartition *Create a random sampling-based S3 object of class partition for the SetTarget function*

Description

Random partitioning is supported for either Training/Validation/Holdout ("TVH") or cross-validation ("CV") splits. In either case, the holdout percentage (holdoutPct) must be specified; for the "CV" method, the number of cross-validation folds (reps) must also be specified, while for the "TVH" method, the validation subset percentage (validationPct) must be specified.

Usage

```
CreateRandomPartition(validationType, holdoutPct, reps = NULL,
  validationPct = NULL)
```

Arguments

validationType	character. String specifying the type of partition generated, either "TVH" or "CV".
holdoutPct	integer. The percentage of data to be used as the holdout subset.
reps	integer. The number of cross-validation folds to generate; only applicable when validationType = "CV".
validationPct	integer. The percentage of data to be used as the validation subset.

Details

This function is one of several convenience functions provided to simplify the task of starting modeling projects with custom partitioning options. The other functions are CreateGroupPartition, CreateStratifiedPartition, and CreateUserPartition.

Value

An S3 object of class partition including the parameters required by SetTarget to generate a random partitioning of the modeling dataset.

See Also

CreateStratifiedPartition, CreateGroupPartition, CreateUserPartition.

Examples

```
CreateRandomPartition(validationType = "CV", holdoutPct = 20, reps = 5)
```

CreateRatingTable *Creates and validates a new rating table from an uploaded CSV.*

Description

Creates and validates a new rating table from an uploaded CSV.

Usage

```
CreateRatingTable(project, parentModelId, file,
  ratingTableName = "Uploaded Rating Table")
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
parentModelId	integer. The id of the model to validate the rating table against.
file	character. The filename containing the rating table CSV to upload.
ratingTableName	character. Optional. The name of the rating table.

Value

An integer value that can be used as the JobId parameter in subsequent calls representing this job.

Examples

```
## Not run:
   projectId <- "5984b4d7100d2b31c1166529"
   modelId <- "5984b4d7100d2b31c1166529"
   CreateRatingTable(projectId, modelId, file = "myRatingTable.csv")

## End(Not run)
```

CreateStratifiedPartition

Create a stratified sampling-based S3 object of class partition for the SetTarget function

---

Description

Stratified partitioning is supported for binary classification problems and it randomly partitions the modeling data, keeping the percentage of positive class observations in each partition the same as in the original dataset. Stratified partitioning is supported for either Training/Validation/Holdout ("TVH") or cross-validation ("CV") splits. In either case, the holdout percentage (holdoutPct) must be specified; for the "CV" method, the number of cross-validation folds (reps) must also be specified, while for the "TVH" method, the validation subset percentage (validationPct) must be specified.

Usage

```
CreateStratifiedPartition(validationType, holdoutPct, reps = NULL,
  validationPct = NULL)
```

Arguments

validationType	character. String specifying the type of partition generated, either "TVH" or "CV".
holdoutPct	integer. The percentage of data to be used as the holdout subset.
reps	integer. The number of cross-validation folds to generate; only applicable when validationType = "CV".
validationPct	integer. The percentage of data to be used as the validation subset.

Details

This function is one of several convenience functions provided to simplify the task of starting modeling projects with custom partitioning options. The other functions are `CreateGroupPartition`, `CreateRandomPartition`, and `CreateUserPartition`.

Value

An S3 object of class 'partition' including the parameters required by the SetTarget function to generate a stratified partitioning of the modeling dataset.

See Also

`CreateGroupPartition`, `CreateRandomPartition`, `CreateUserPartition`.

Examples

```
CreateStratifiedPartition(validationType = "CV", holdoutPct = 20, reps = 5)
```

CreateUserPartition *Create a class partition object for use in the SetTarget function representing a user-defined partition.*

Description

Creates a list object used by the SetTarget function to specify either Training/Validation/Holdout (validationType = "TVH") or cross-validation (validationType = "CV") partitions of the modeling dataset based on the values included in a column from the dataset. In either case, the name of this data column must be specified (as userPartitionCol).

Usage

```
CreateUserPartition(validationType, userPartitionCol, cvHoldoutLevel = NULL,
  trainingLevel = NULL, holdoutLevel = NULL, validationLevel = NULL)
```

Arguments

validationType	character. String specifying the type of partition generated, either "TVH" or "CV".
userPartitionCol	character. String naming the data column from the modeling dataset containing the subset designations.
cvHoldoutLevel	character. Data value from userPartitionCol that identifies the holdout subset under the "CV" option.
trainingLevel	character. Data value from userPartitionCol that identifies the training subset under the "TVH" option.
holdoutLevel	character. Data value from userPartitionCol that identifies the holdout subset under both "TVH" and "CV" options. To specify that the project should not use a holdout you can omit this parameter or pass NA directly.
validationLevel	character. Data value from userPartitionCol that identifies the validation subset under the "TVH" option.

Details

For the "TVH" option of cvMethod, no cross-validation is used. Users must specify the trainingLevel and validationLevel; use of a holdoutLevel is always recommended but not required. If no holdoutLevel is used, then the column must contain exactly 2 unique values. If a holdoutLevel is used, the column must contain exactly 3 unique values.

For the "CV" option, each value in the column will be used to separate rows into cross-validation folds. Use of a holdoutLevel is optional; if not specified, then no holdout is used.

This function is one of several convenience functions provided to simplify the task of starting modeling projects with custom partitioning options. The other functions are `CreateGroupPartition`, `CreateRandomPartition`, and `CreateStratifiedPartition`.

Value

An S3 object of class 'partition' including the parameters required by the SetTarget function to generate a user-specified of the modeling dataset.

See Also

`CreateGroupPartition`, `CreateRandomPartition`, `CreateStratifiedPartition`.

Examples

```
CreateUserPartition(validationType = "CV", userPartitionCol = "TVHflag", cvHoldoutLevel = NA)
```

cvMethods *CV methods*

Description

This is a list that contains the valid values for CV methods

Usage

```
cvMethods
```

Format

An object of class `list` of length 5.

DataPartition *Data Partition methods*

Description

This is a list that contains the valid values for data partitions

Usage

```
DataPartition
```

Format

An object of class `list` of length 3.

DataPathFromDataArg *Get the data path.*

Description

Verifies that newdata is either an existing datafile or a dataframe If a dataframe, save as a CSV file If neither an existing datafile nor a dataframe, halt with error

Usage

```
DataPathFromDataArg(dataSource, saveFile = NULL)
```

Arguments

`dataSource`	object. The dataframe or path to CSV to get data for.
`saveFile`	character. Optional. A file name to write an autosaved dataframe to.

DataSubset *Data subset for training predictions*

Description

This is a list that contains the valid values for the `dataSubset` parameter found in `RequestTrainingPredictions`. If you wish, you can specify `dataSubset` using the list values here.

Usage

```
DataSubset
```

Format

An object of class `list` of length 3.

DeleteJob *Cancel a running job*

Description

Cancel a running job

Usage

```
DeleteJob(job)
```

Arguments

`job`	object. The job you want to cancel (one of the items in the list returned from `ListJobs`)

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  initialJobs <- GetModelJobs(project)
  job <- initialJobs[[1]]
  DeleteJob(job)

## End(Not run)
```

DeleteModel *Delete a specified DataRobot model*

Description

This function removes the model specified by the parameter model from its associated project.

Usage

```
DeleteModel(model)
```

Arguments

model	An S3 object of class dataRobotModel like that returned by the function GetModel, or each element of the list returned by the function ListModels.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  model <- GetModel(projectId, modelId)
  DeleteModel(model)

## End(Not run)
```

DeleteModelJob *Delete a model job from the modeling queue*

Description

This function deletes the modeling job specified by modelJobId from the DataRobot modeling queue.

Usage

```
DeleteModelJob(project, modelJobId)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
modelJobId	integer. Identifier for the modeling job to be deleted; can be obtained from the results returned by the function GetModelJobs.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  initialJobs <- GetModelJobs(project)
  job <- initialJobs[[1]]
  modelJobId <- job$modelJobId
  DeleteModelJob(projectId, modelJobId)

## End(Not run)
```

`DeletePredictionDataset` *Delete a specified prediction dataset*

Description

This function removes a prediction dataset

Usage

```
DeletePredictionDataset(project, datasetId)
```

Arguments

`project`	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
`datasetId`	The id of the dataset to delete

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  datasets <- ListPredictionDatasets(projectId)
  dataset <- datasets[[1]]
  datasetId <- dataset$id
  DeletePredictionDataset(projectId, datasetId)

## End(Not run)
```

DeletePredictJob *Function to delete one predict job from the DataRobot queue*

Description

This function deletes the predict job specified by predictJobId from the DataRobot queue.

Usage

```
DeletePredictJob(project, predictJobId)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
predictJobId	integer. Id identifying the prediction job created by the call to RequestPredictionsForDataset.

Value

Logical TRUE and displays a message to the user if the delete request was successful; otherwise, execution halts and an error message is displayed.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  initialJobs <- GetPredictJobs(project)
  job <- initialJobs[[1]]
  predictJobId <- job$predictJobId
  DeletePredictJob(projectId, predictJobId)

## End(Not run)
```

DeleteProject *Delete a specified element from the DataRobot project list*

Description

This function deletes the project defined by project, described under Arguments. This parameter may be obtained in several ways, including: (1), as one of the projectId elements of the list returned by GetProjectList; (2), as the S3 object returned by the GetProject function; or (3), as the list returned by the SetupProject function.

Usage

```
DeleteProject(project)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  DeleteProject(projectId)

## End(Not run)
```

DeleteReasonCodes *Function to delete reason codes*

Description

This function deletes reason codes specified by project and reasonCodeId

Usage

```
DeleteReasonCodes(project, reasonCodeId)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
reasonCodeId	character. id of the reason codes.

Value

Logical TRUE and displays a message to the user if the delete request was successful; otherwise an error message is displayed.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  datasets <- ListPredictionDatasets(projectId)
  dataset <- datasets[[1]]
  datasetId <- dataset$id
  model <- GetModel(model, datasetId)
  jobId <- RequestReasonCodes(model, datasetId)
  reasonCodeId <- GetReasonCodesMetadataFromJobId(projectId, jobId)$id
  DeleteReasonCodes(projectId, reasonCodeId)

## End(Not run)
```

DeleteReasonCodesInitialization
Delete the reason codes initialization for a model.

Description

Delete the reason codes initialization for a model.

Usage

```
DeleteReasonCodesInitialization(model)
```

Arguments

model	An S3 object of class dataRobotModel like that returned by the function GetModel, or each element of the list returned by the function ListModels.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  model <- GetModel(projectId, modelId)
  DeleteReasonCodesInitialization(model)

## End(Not run)
```

DeleteTransferrableModel
Delete this imported model.

Description

Delete this imported model.

Usage

```
DeleteTransferrableModel(importId)
```

Arguments

importId	character. Id of the import.

Examples

```
## Not run:
  id <- UploadTransferrableModel("model.drmodel")
  DeleteTransferrableModel(id)

## End(Not run)
```

`DifferencingMethod` *Differencing method*

Description

Differencing method

Usage

```
DifferencingMethod
```

Format

An object of class `list` of length 4.

`DownloadPrimeCode` *Download the code of DataRobot Prime model and save it to a file.*

Description

Training a model using a ruleset is a necessary prerequisite for being able to download the code for a ruleset.

Usage

```
DownloadPrimeCode(project, primeFileId, filepath)
```

Arguments

`project`	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
`primeFileId`	numeric. Prime file Id (can be aquired using ListPrimeFiles function)
`filepath`	character. The location to save the file to.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  primeFiles <- ListPrimeFiles(projectId)
  primeFile <- primeFiles[[1]]
  primeFileId <- primeFile$id
  file <- file.path(tempdir(), "primeCode.py")
  DownloadPrimeCode(projectId, primeFileId, file)

## End(Not run)
```

`DownloadRatingTable` *Download a rating table to a CSV.*

Description

Download a rating table to a CSV.

Usage

```
DownloadRatingTable(project, ratingTableId, filename)
```

Arguments

`project`	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
`ratingTableId`	character. The ID of the rating table.
`filename`	character. Filename of file to save the rating table to.

Value

Nothing returned, but downloads the file to the stated filename.

Examples

```
## Not run:
 projectId <- "5984b4d7100d2b31c1166529"
 ratingTableId <- "5984b4d7100d2b31c1166529"
 file <- file.path(tempdir(), "ratingTable.csv")
 DownloadRatingTable(projectId, ratingTableId, file)

## End(Not run)
```

DownloadReasonCodes *Function to download and save reason codes rows as csv file*

Description

Function to download and save reason codes rows as csv file

Usage

```
DownloadReasonCodes(project, reasonCodeId, filename, encoding = "UTF-8",
  excludeAdjustedPredictions = TRUE)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
reasonCodeId	character. id of the reason codes.
filename	character. Fileneme of file to save reason codes rows
encoding	character. Optional. Character string A string representing the encoding to use in the output file, defaults to 'UTF-8'.
excludeAdjustedPredictions	logical. Optional. Set to FALSE to include adjusted predictions, which are predictions adjusted by an exposure column. This is only relevant for projects that use an exposure column.

Value

Logical TRUE and displays a message to the user if the delete request was successful; otherwise an error message is displayed.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  datasets <- ListPredictionDatasets(projectId)
  dataset <- datasets[[1]]
  datasetId <- dataset$id
  model <- GetModel(model, datasetId)
  jobId <- RequestReasonCodes(model, datasetId)
  reasonCodeId <- GetReasonCodesMetadataFromJobId(projectId, jobId)$id
  file <- file.path(tempdir(), "testReasonCode.csv")
  DownloadReasonCodes(projectId, reasonCodeId, file)

## End(Not run)
```

DownloadScoringCode *Download scoring code JAR*

Description

Download scoring code JAR

Usage

```
DownloadScoringCode(project, modelId, fileName, sourceCode = FALSE)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
modelId	Character string: unique alphanumeric identifier for the model of interest.
fileName	Character string: File path where scoring code will be saved.
sourceCode	Logical (optional) : Set to True to download source code archive. It will not be executable.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  file <- file.path(tempdir(), "scoringCode.jar")
  DownloadScoringCode(projectId, modelId, file)

## End(Not run)
```

DownloadTrainingPredictions *Download training predictions on a specified data set.*

Description

Download training predictions on a specified data set.

Usage

```
DownloadTrainingPredictions(project, predictionId, filename,
  encoding = "UTF-8")
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
predictionId	character. ID of the prediction to retrieve training predictions for.
filename	character. Filename of file to save reason codes rows
encoding	character. Optional. Character string A string representing the encoding to use in the output file, defaults to 'UTF-8'.

Value

NULL, but will produce a CSV with a dataframe with out-of-fold predictions for the training data.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  predictions <- ListTrainingPredictions(projectId)
  predictionId <- predictions[[1]]$predictionId
  file <- file.path(tempdir(), "myTrainingPredictions.csv")
  DownloadTrainingPredictions(proejctId, predictionId, file)

## End(Not run)
```

DownloadTransferrableModel

Download an transferrable model file for use in an on-premise DataRobot standalone prediction environment.

Description

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.

Usage

```
DownloadTransferrableModel(project, modelId, modelFile)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
modelId	numeric. Unique alphanumeric identifier for the model of interest.
modelFile	character. File name to be use for tranferrable model

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  file <- file.path(tempdir(), "model.drmodel")
  DownloadTransferrableModel(projectId, modelId, file)

## End(Not run)
```

ExpectHasKeys *Make sure that the object has all of the keys specified. Also tests that there are not additional keys if* allowAdditional *is FALSE (default).*

Description

Make sure that the object has all of the keys specified. Also tests that there are not additional keys if allowAdditional is FALSE (default).

Usage

```
ExpectHasKeys(obj, keys, allowAdditional = FALSE)
```

Arguments

obj	object. A list, vector, or data.frame to check names.
keys	character. A vector of names of keys to check.
allowAdditional	logical. Should we allow there to be more keys than specified?

FeatureFromAsyncUrl *Retrieve a feature from the creation URL*

Description

If feature creation times out, the error message includes a URL corresponding to the creation task. That URL can be passed to this function (which will return the feature details when finished) to resume waiting for feature creation.

Usage

```
FeatureFromAsyncUrl(asyncUrl, maxWait = 600)
```

Arguments

asyncUrl	The temporary status URL
maxWait	The maximum time to wait (in seconds) for project creation before aborting.

`GenerateDatetimePartition` *Preview the full partitioning determined by a DatetimePartitioningSpecification*

Description

Based on the project dataset and the partitioning specification, inspect the full partitioning that would be used if the same specification were passed into SetTarget. This is not intended to be passed to SetTarget.

Usage

```
GenerateDatetimePartition(project, spec)
```

Arguments

`project`	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
`spec`	list. Datetime partition specification returned by `CreateDatetimePartitionSpecification`

Value

list describing datetime partition with following components

- projectId. Character string the id of the project this partitioning applies to.
- datetimePartitionColumn. Character string the name of the column whose values as dates are used to assign a row to a particular partition.
- dateFormat. Character string the format (e.g. " partition column was interpreted (compatible with strftime [https://docs.python.org/2/library/time.html#time.strftime]).
- autopilotDataSelectionMethod. Character string Whether models created by the autopilot use "rowCount" or "duration" as their dataSelectionMethod.
- validationDuration. Character string the validation duration specified when initializing the partitioning - not directly significant if the backtests have been modified, but used as the default validationDuration for the backtests.
- availableTrainingStartDate. Character string The start date of the available training data for scoring the holdout.
- availableTrainingDuration. Character string The duration of the available training data for scoring the holdout.
- availableTrainingRowCount. integer The number of rows in the available training data for scoring the holdout. Only available when retrieving the partitioning after setting the target.
- availableTrainingEndDate. Character string The end date of the available training data for scoring the holdout.
- primaryTrainingStartDate. Character string The start date of primary training data for scoring the holdout.

- primaryTrainingDuration. Character string The duration of the primary training data for scoring the holdout.
- primaryTrainingRowCount. integer The number of rows in the primary training data for scoring the holdout. Only available when retrieving the partitioning after setting the target.
- primaryTrainingEndDate. Character string The end date of the primary training data for scoring the holdout.
- gapStartDate. Character string The start date of the gap between training and holdout scoring data.
- gapDuration. Character string The duration of the gap between training and holdout scoring data.
- gapRowCount. integer The number of rows in the gap between training and holdout scoring data. Only available when retrieving the partitioning after setting the target.
- gapEndDate. Character string The end date of the gap between training and holdout scoring data.
- holdoutStartDate. Character string The start date of holdout scoring data.
- holdoutDuration. Character string The duration of the holdout scoring data.
- holdoutRowCount. integer The number of rows in the holdout scoring data. Only available when retrieving the partitioning after setting the target.
- holdoutEndDate. Character string The end date of the holdout scoring data.
- numberOfBacktests. integer the number of backtests used.
- backtests. data.frame of partition backtest. Each elemnet represent one backtest and has following components: index, availableTrainingStartDate, availableTrainingDuration, availableTrainingRowCount, availableTrainingEndDate, primaryTrainingStartDate, primaryTrainingDuration, primaryTrainingRowCount, primaryTrainingEndDate, gapStartDate, gapDuration, gapRowCount, gapEndDate, validationStartDate, validationDuration, validationRowCount, validationEndDate, totalRowCount.
- useTimeSeries logical. Whether the project is a time series project (if TRUE) or an OTV project which uses datetime partitioning (if FALSE).
- defaultToKnownInAdvance logical. Whether the project defaults to treating features as a priori. A priori features are time series features that are expected to be known for dates in the future when making predictions (e.g., "is this a holiday").
- featureDerivationWindowStart integer. Offset into the past to define how far back relative to the forecast point the feature derivation window should start. Only used for time series projects. Expressed in terms of the `timeUnit` of the `datetimePartitionColumn`.
- featureDerivationWindowEnd integer. Offset into the past to define how far back relative to the forecast point the feature derivation window should end. Only used for time series projects. Expressed in terms of the `timeUnit` of the `datetimePartitionColumn`.
- forecastWindowStart integer. Offset into the future to define how far forward relative to the forceast point the forecaset window should start. Only used for time series projects. Expressed in terms of the `timeUnit` of the `datetimePartitionColumn`.
- forecastWindowEnd integer. Offset into the future to define how far forward relative to the forceast point the forecaset window should end. Only used for time series projects. Expressed in terms of the `timeUnit` of the `datetimePartitionColumn`.

- totalRowCount. integer the number of rows in the project dataset. Only available when retrieving the partitioning after setting the target. Thus it will be null for GenerateDatetimePartition and populated for GetDatetimePartition.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  partitionSpec <- CreateDatetimePartitionSpecification("date_col")
  GenerateDatetimePartition(projectId, partitionSpec)

## End(Not run)
```

`GetAllLiftCharts` *Retrieve lift chart data for a model for all available data partitions (see DataPartition)*

Description

Retrieve lift chart data for a model for all available data partitions (see DataPartition)

Usage

```
GetAllLiftCharts(model)
```

Arguments

`model`	An S3 object of class dataRobotModel like that returned by the function GetModel, or each element of the list returned by the function ListModels.

Value

data.frame with the following components:

- binWeight. Numeric: weight of the bin. For weighted projects, the sum of the weights of all rows in the bin; otherwise, the number of rows in the bin.
- actual. Numeric: sum of actual target values in bin.
- predicted. Numeric: sum of predicted target values in bin.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  model <- GetModel(projectId, modelId)
  GetAllLiftCharts(model)

## End(Not run)
```

GetAllReasonCodesRowsAsDataFrame
Retrieve all reason codes rows and return them as a data frame

Description

There are some groups of columns whose appearance depends on the exact contents of the project dataset. For classification projects, columns "classNLabel", 'classNProbability", "classNLabel", "classNProbability" will appear corresponding to each class within the target; these columns will not appear for regression projects. Columns like "reasonNLabel" will appear corresponding to each included reason code in the row. In both cases, the value of N will start at 1 and count up.

Usage

```
GetAllReasonCodesRowsAsDataFrame(project, reasonCodeId,
  excludeAdjustedPredictions = TRUE)
```

Arguments

`project`	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
`reasonCodeId`	character. id of the reason codes.
`excludeAdjustedPredictions`	logical. Optional. Set to FALSE to include adjusted predictions, which are predictions adjusted by an exposure column. This is only relevant for projects that use an exposure column.

Value

data frame with following colums:

- rowId. Integer row id from prediction dataset.
- prediction. Numeric the output of the model for this row (numric prediction for regression problem, predicted class for classification problem).
- class1Label. Character string Label of class 0. Available only for classification problem.
- class1Probability. Numeric Predicted probability of class 0. Available only for classification problem.
- class2Label. Character string Label of class 1. Available only for classification
- class2Probability. Numeric Predicted probability of class 1. Available only for classification problem.
- reason1FeatureName. Character string the name of the feature contributing to the prediction.
- reason1FeatureValue. the value the feature took on for this row.
- reason1QualitativeStrength. Numeric how strongly the feature affected the prediction.
- reason1Strength. Character string a human-readable description of how strongly the feature affected the prediction (e.g. '+++', '–', '+').

- reason1Label. Character string describes what output was driven by this reason code. For regression projects, it is the name of the target feature. For classification projects, it is theclass whose probability increasing would correspond to a positive strength of this.
- reasonNFeatureName. Character string the name of the feature contributing to the prediction.
- reasonNFeatureValue. the value the feature took on for this row.
- reasonNQualitativeStrength. Numeric how strongly the feature affected the prediction.
- reasonNStrength. Character string a human-readable description of how strongly the feature affected the prediction (e.g. '+++', '–', '+').
- reasonNLabel. Character string describes what output was driven by this reason code. For regression projects, it is the name of the target feature. For classification projects, it is theclass whose probability increasing would correspond to a positive strength of this.
- reasonNFeatureName. Character string the name of the feature contributing to the prediction.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  datasets <- ListPredictionDatasets(projectId)
  dataset <- datasets[[1]]
  datasetId <- dataset$id
  model <- GetModel(model, datasetId)
  jobId <- RequestReasonCodes(model, datasetId)
  reasonCodeId <- GetReasonCodesMetadataFromJobId(projectId, jobId)$id
  GetReasonCodesRowsAsDataFrame(projectId, reasonCodeId)

## End(Not run)
```

GetAllRocCurves *Retrieve ROC curve data for a model for all available data partitions (see DataPartition)*

Description

Retrieve ROC curve data for a model for all available data partitions (see DataPartition)

Usage

```
GetAllRocCurves(model)
```

Arguments

model	An S3 object of class dataRobotModel like that returned by the function GetModel, or each element of the list returned by the function ListModels.

Value

list of lists where each list is renamed as the data partitions source and returns the following components:

- source. Character: data partitions for which ROC curve data is returned (see DataPartition).
- negativeClassPredictions. Numeric: example predictions for the negative class for each
- rocPoints. data.frame: each row represents pre-calculated metrics (accuracy, f1_score, false_negative_score, true_negative_score, true_positive_score, false_positive_score, true_negative_rate, false_positive_rate, true_positive_rate, matthews_correlation_coefficient, positive_predictive_value, negative_predictive_value, threshold) associated with different thresholds for the ROC curve.
- positiveClassPredictions. Numeric: example predictions for the positive class for each data partition source.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  model <- GetModel(projectId, modelId)
  GetAllRocCurves(model)

## End(Not run)
```

GetBlenderModel *Retrieve the details of a specified blender model*

Description

This function returns a DataRobot S3 object of class dataRobotModel for the model defined by project and modelId.

Usage

```
GetBlenderModel(project, modelId)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
modelId	character. Unique alphanumeric identifier for the blender model of interest.

Value

An S3 object of class 'dataRobotBlenderModel', which is a list with the following components:

- featurelistId. Character string: unique alphanumeric identifier for the = featurelist on which the model is based.
- processes. Character vector with components describing preprocessing; may include modelType.
- featurelistName. Character string giving the name of the featurelist on which the model is based.
- projectId. Character string giving the unique alphanumeric identifier for the project.
- samplePct. Numeric: percentage of the dataset used to form the training dataset
- isFrozen. Logical : is model created with frozen tuning parameters.
- modelType. Character string describing the model type.
- metrics. List with one element for each valid metric associated with the model. Each element is a list with elements for each possible evaluation type (holdout, validation, and crossValidation).
- modelCategory. Character string giving model category (e.g., blend, model).
- blueprintId. Character string giving the unique DataRobot blueprint identifier on which the model is based.
- modelIds. Character string giving the unique alphanumeric model identifier of blended models.
- blenderMethod. Character string describing blender method.
- modelId. Character string giving the unique alphanumeric blender model identifier.
- projectName. Character string: name of project defined by projectId.
- projectTarget. Character string defining the target variable predicted by all models in the project.
- projectMetric. Character string defining the fitting metric optimized by all project models.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  GetBlenderModel(projectId, modelId)

## End(Not run)
```

GetBlenderModelFromJobId

Retrieve a new or updated blender model defined by modelJobId

Description

The function RequestBlender initiates the creation of new blender models in a DataRobot project. It submits requests to the DataRobot modeling engine and returns an integer-valued modelJobId. The GetBlenderModelFromJobId function polls the modeling engine until the model has been built or a specified time limit is exceeded, returning an S3 object of class 'dataRobotBlenderModel' when the model is available.

Usage

```
GetBlenderModelFromJobId(project, modelJobId, maxWait = 600)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
modelJobId	integer. The integer returned by RequestBlender.
maxWait	integer. The maximum time (in seconds) to wait for the model job to

Details

Motivation for this function is the fact that some models - e.g., very complex machine learning models fit to large datasets - may take a long time to complete. Splitting the model creation request from model retrieval in these cases allows the user to perform other interactive R session tasks between the time the model creation/update request is made and the time the final model is available.

Value

An S3 object of class 'dataRobotBlenderModel' summarizing all available information about the model. It is a list with the following components:

- featurelistId. Character string: unique alphanumeric identifier for the featurelist on which the model is based.
- processes. Character vector with components describing preprocessing; may include modelType.
- featurelistName. Character string giving the name of the featurelist on which the model is based.
- projectId. Character string giving the unique alphanumeric identifier for the project.
- samplePct. Numeric: percentage of the dataset used to form the training dataset for model fitting.

- trainingRowCount. Integer. 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 `trainingDuration` or `trainingStartDate` and `trainingEndDate` was used to
- isFrozen. Logical : is model created with frozen tuning parameters.
- modelType. Character string describing the model type.
- metrics. List with one element for each valid metric associated with the model. Each element is a list with elements for each possible evaluation type (holdout,
- modelCategory. Character string giving model category (e.g., blend, model).
- blueprintId. Character string giving the unique DataRobot blueprint identifier on which the model is based.
- modelIds. Character string giving the unique alphanumeric model identifier of blended models.
- blenderMethod. Character string describing blender method.
- id. Character string giving the unique alphanumeric blender model identifier.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelsToBlend <- c("5996f820af07fc605e81ead4", "59a5ce3301e9f0296721c64c")
  blendJobId <- RequestBlender(projectId, modelId, "GLM")
  GetBlenderModelFromJobId(projectId, blendJobId)

## End(Not run)
```

`GetBlueprint` *Retrieve a blueprint*

Description

Retrieve a blueprint

Usage

```
GetBlueprint(project, blueprintId)
```

Arguments

`project`	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
`blueprintId`	character. Id of blueprint to retrieve.

Value

List with the following four components:

projectId Character string giving the unique DataRobot project identifier

processes List of character strings, identifying any preprocessing steps included in the blueprint

blueprintId Character string giving the unique DataRobot blueprint identifier

modelType Character string, specifying the type of model the blueprint builds

blueprintCategory Character string. Describes the category of the blueprint and the kind of model it produces.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  model <- GetModel(projectId, modelId)
  blueprintId <- model$blueprintId
  GetBlueprint(projectId, blueprintId)

## End(Not run)
```

GetBlueprintChart *Retrieve a blueprint chart*

Description

A Blueprint chart can be used to understand data flow in blueprint.

Usage

```
GetBlueprintChart(project, blueprintId)
```

Arguments

`project`	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
`blueprintId`	character. Id of blueprint to retrieve.

Value

List with the following two components:

- nodes. list each element contains information about one node of a blueprint : id and label.
- edges. Two column matrix, identifying blueprint nodes connections.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  model <- GetModel(projectId, modelId)
  blueprintId <- model$blueprintId
  GetBlueprintChart(projectId, blueprintId)

## End(Not run)
```

`GetBlueprintDocumentation` *Get documentation for tasks used in the blueprint*

Description

Get documentation for tasks used in the blueprint

Usage

```
GetBlueprintDocumentation(project, blueprintId)
```

Arguments

`project`	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
`blueprintId`	character. Id of blueprint to retrieve.

Value

list with following components

task Character string name of the task described in document

description Character string task description

title Character string title of document

parameters List of parameters that task can received in human-readable format with following components: name, type, description

links List of exteranl lines used in document with following components: name, url

references List of references used in document with following components: name, url

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  model <- GetModel(projectId, modelId)
  blueprintId <- model$blueprintId
  GetBlueprintDocumentation(projectId, blueprintId)

## End(Not run)
```

`GetConfusionChart` *Retrieve a model's confusion chart for a specified source.*

Description

Retrieve a model's confusion chart for a specified source.

Usage

```
GetConfusionChart(model, source = DataPartition$VALIDATION)
```

Arguments

`model`	An S3 object of class dataRobotModel like that returned by the function GetModel, or each element of the list returned by the function ListModels.
`source`	character. The source to pull the confusion chart from. See `DataPartition` for options for sources. Defaults to `DataPartition$VALIDATION`.

Value

data.frame with the following components:

- source character. The name of the source of the confusion chart. Will be a member of `DataPartition`.
- data list. The data for the confusion chart, containing:
  - classes character. A vector containing the names of all the classes.
  - confusionMatrix matrix. A matrix showing the actual versus the predicted class values.
  - classMetrics list. A list detailing further metrics for each class:
    * wasActualPercentages data.frame. A dataframe detailing the actual percentage distribution of the classes.
    * wasPredictedPercentages data.frame. A dataframe detailing the predicted distribution of the classes.
    * f1 numeric. The F1 score for the predictions of the class.
    * recall numeric. The recall score for the predictions of the class.
    * precision numeric. The precision score for the predictions of the class.
    * actualCount integer. The actual count of values for the class.
    * predictedCount integer. The predicted count of values for the class.
    * className character. A vector containing the name of the class.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  GetModel(projectId, modelId)
  GetConfusionChart(modelId, source = DataPartition$VALIDATION)

## End(Not run)
```

`GetDatetimeModelFromJobId` *Retrieve a new or updated datetime model defined by modelJobId*

Description

The functions RequestNewDatatimeModel and RequestFrozenDatetimeModel initiate the creation of new models in a DataRobot project. Both functions submit requests to the DataRobot modeling engine and return an integer-valued modelJobId. The GetDatetimeModelFromJobId function polls the modeling engine until the model has been built or a specified time limit is exceeded, returning an S3 object of class 'dataRobotDatetimeModel' when the model is available.

Usage

```
GetDatetimeModelFromJobId(project, modelJobId, maxWait = 600)
```

Arguments

`project`	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
`modelJobId`	The integer returned by either RequestNewDatetimeModel
`maxWait`	Integer, The maximum time (in seconds) to wait for the model job to complete

Details

Motivation for this function is the fact that some models - e.g., very complex machine learning models fit to large datasets - may take a long time to complete. Splitting the model creation request from model retrieval in these cases allows the user to perform other interactive R session tasks between the time the model creation/update request is made and the time the final model is available.

Value

An S3 object of class 'dataRobotDatetimeModel' summarizing all available information about the model. See GetDatetimeModelObject

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  initialJobs <- GetModelJobs(project)
  job <- initialJobs[[1]]
  modelJobId <- job$modelJobId
  GetDatetimeModelFromJobId(projectId, modelJobId)

## End(Not run)
```

`GetDatetimeModelObject` *Retrieve the details of a specified datetime model.*

Description

This function returns a DataRobot S3 object of class dataRobotDatetimeModel for the model defined by project and modelId.

Usage

```
GetDatetimeModelObject(project, modelId)
```

Arguments

`project`	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
`modelId`	character. Unique alphanumeric identifier for the model of interest.

Details

If the project does not use datetime partitioning an error will occur.

Value

An S3 object of class 'dataRobotDatetimeModel', which is a list with the following components:

- featurelistId. Character string: unique alphanumeric identifier for the featurelist on which the model is based.
- processes. Character vector with components describing preprocessing; may include modelType.
- featurelistName. Character string giving the name of the featurelist on which the model is based.
- projectId. Character string giving the unique alphanumeric identifier for the project.
- samplePct. Numeric: percentage of the dataset used to form the training dataset for model fitting.

- isFrozen. Logical : is model created with frozen tuning parameters.
- modelType. Character string describing the model type.
- metrics. List with one element for each valid metric associated with the model. Each element is a list with elements for each possible evaluation type (holdout, validation, and crossValidation).
- modelCategory. Character string giving model category (e.g., blend, model).
- blueprintId. Character string giving the unique DataRobot blueprint identifier on which the model is based.
- modelId. Character string giving the unique alphanumeric model identifier.
- projectName. Character string: optional description of project defined by projectId.
- projectTarget. Character string defining the target variable predicted by all models in the project.
- projectMetric. Character string defining the fitting metric optimized by all project models.
- trainingRowCount. Integer. 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 `trainingDuration` or `trainingStartDate` and `trainingEndDate` was used to determine that instead.
- trainingDuration. Character string 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.
- trainingStartDate. Charcter string or none only present for frozen models in datetime partitioned projects. If specified, the start date of the data used to train the model.
- trainingEndDate. Charcter string or none only present for frozen models in datetime partitioned projects. If specified, the end date of the data used to train the model.
- backtests. list 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.
- dataSelectionMethod. Character string which of trainingRowCount, trainingDuration, or trainingStartDate and trainingEndDate were used to determine the data used to fit the model. One of 'rowCount', 'duration', or 'selectedDateRange'.
- trainingInfo. list describes which data was used to train on when scoring the holdout and making predictions. trainingInfo will have the following keys: 'holdoutTrainingStartDate', 'holdoutTrainingDuration', 'holdoutTrainingRowCount', 'holdoutTrainingEndDate', 'predictionTrainingStartDate', 'predictionTrainingDuration', 'predictionTrainingRowCount', 'predictionTrainingEndDate'. Start and end dates will be datetime string, durations will be duration strings, and rows will be integers.
- holdoutScore. numeric or none the score against the holdout, if available and the holdout is unlocked, according to the project metric.
- holdoutStatus. Character string the status of the holdout score, e.g. "COMPLETED", "HOLDOUT_BOUNDARIES_EXCEEDED".

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  GetDatetimeModelObject(projectId, modelId)

## End(Not run)
```

`GetDatetimePartition` *Retrieve the DatetimePartitioning from a project*

Description

Only available if the project has already set the target as a datetime project.

Usage

```
GetDatetimePartition(project)
```

Arguments

`project`	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.

Value

list describing datetime partition. See `GenerateDatetimePartition`.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  GetDatetimePartition(projectId)

## End(Not run)
```

`GetFeatureImpactForJobId`
Retrieve completed Feature Impact results given a job ID

Description

This will wait for the Feature Impact job to be completed (giving an error if the job is not a Feature Impact job and an error if the job errors).

Usage

```
GetFeatureImpactForJobId(project, jobId, maxWait = 600)
```

Arguments

project	character. The project the Feature Impact is part of.
jobId	character. The ID of the job (e.g. as returned from RequestFeatureImpact)
maxWait	integer. The maximum time (in seconds) to wait for the model job to complete

Value

A data frame with the following columns:

featureName The name of the feature

impactNormalized The normalized impact score (largest value is 1)

impactUnnormalized The unnormalized impact score

Examples

```
## Not run:
  model <- ListModels(project)[[1]]
  featureImpactJobId <- RequestFeatureImpact(model)
  featureImpact <- GetFeatureImpactForJobId(project, featureImpactJobId)

## End(Not run)
```

GetFeatureImpactForModel

Retrieve completed Feature Impact results given a model

Description

This will only succeed if the Feature Impact computation has completed.

Usage

```
GetFeatureImpactForModel(model)
```

Arguments

model	character. The model for which you want to retrieve Feature Impact.

Details

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. Elsewhere this technique is sometimes called 'Permutation Importance'.

Value

A data frame with the following columns:

featureName The name of the feature

impactNormalized The normalized impact score (largest value is 1)

impactUnnormalized The unnormalized impact score

Examples

```
## Not run:
  model <- ListModels(project)[[1]]
  featureImpactJobId <- RequestFeatureImpact(model)
  # Note: This will only work after the feature impact job has completed. Use
  #       GetFeatureImpactFromIobId to automatically wait for the job.\
  featureImpact <- GetFeatureImpactForModel(model)

## End(Not run)
```

GetFeatureInfo *Details about a feature*

Description

Details about a feature

Usage

```
GetFeatureInfo(project, featureName)
```

Arguments

`project`	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
`featureName`	Name of the feature to retrieve. Note: DataRobot renames some features, so the feature name may not be the one from your original data. You can use ListFeatureInfo to list the features and check the name.

Value

A named list which contains:

- id nummeric. feature id. Note that throughout the API, features are specified using their names, not this ID.
- name character. The name of the feature.
- featureType character. Feature type: 'Numeric', 'Categorical', etc.
- importance numeric. numeric measure of the strength of relationship between the feature and target (independent of any model or other features).
- lowInformation logical. Whether the feature has too few values to be informative.
- uniqueCount numeric. The number of unique values in the feature.
- naCount numeric. The number of missing values in the feature.
- dateFormat character. The format of the feature if it is date-time feature.
- projectId character. Character id of the project the feature belonges to.
- max. The maximum value in the dataset, formatted in the same format as the data.
- min. The minimum value in the dataset, formatted in the same format as the data.
- mean. The arithmetic mean of the dataset, formatted in the same format as the data.
- median. The median of the dataset, formatted in the same format as the data.
- stdDev. The standard deviation of the dataset, formatted in the same format as the data.
- timeSeriesEligible logical. Whether this feature can be used as the datetime partition column in a time series project.
- timeSeriesEligibilityReason character. Why the feature is ineligible for the datetime partition column in a time series project, "suitable" when it is eligible.
- timeStep numeric. 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. NULL for features that are not time series eligible.
- timeUnit character. For time series eligible features, the time unit covered by a single time step, e.g. "HOUR", or NULL for features that are not time series eligible.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  GetFeatureInfo(projectId, "myFeature")

## End(Not run)
```

GetFeaturelist *Retrieve a specific featurelist from a DataRobot project*

Description

This function returns information about and the contents of a specified featurelist from a specified project.

Usage

```
GetFeaturelist(project, featurelistId)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
featurelistId	Unique alphanumeric identifier for the featurelist to be retrieved.

Details

DataRobot featurelists define the variables from the modeling dataset used in fitting each project model. In most cases, the same featurelist is used in fitting all project models, but models can be fit using alternative featurelists using the RequestNewModel function. To do this, featurelistId is required, and this is one of the elements returned by the GetFeaturelist function.

DataRobot featurelists define the variables from the modeling dataset used in fitting each project model. In most cases, the same featurelist is used in fitting all project models, but models can be fit using alternative featurelists using the RequestNewModel function. To do this, featurelistId is required, and this is one of the elements returned by the GetFeaturelist function.

Value

A list with the following four elements describing the requested featurelist:

- featurelistId. Character string giving the unique alphanumeric identifier for the featurelist.
- projectId. Character string identifying the project to which the featurelist belongs.
- features. Character vector with the names of the variables included in the featurelist.
- name. Character string giving the name of the featurelist.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  featureList <- CreateFeaturelist(projectId, "myFeaturelist", c("feature1", "feature2"))
  featurelistId <- featureList$featurelistId
  GetFeaturelist(projectId, featurelistId)

## End(Not run)
```

`GetFrozenModel` *Retrieve the details of a specified frozen model*

Description

This function returns a DataRobot S3 object of class dataRobotFrozenModel for the model defined by project and modelId. GetModel also can be used to retrieve some information about frozen model, however then some frozen specific information (parentModelId) will not be returned

Usage

```
GetFrozenModel(project, modelId)
```

Arguments

`project`	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
`modelId`	Unique alphanumeric identifier for the model of interest.

Details

The S3 object returned by this function is required by the functions DeleteModel, ListModelFeatures, and RequestSampleSizeUpdate.

Value

An S3 object of class 'dataRobotModel', which is a list with the following components:

- featurelistId. Character string: unique alphanumeric identifier for the featurelist on which the model is based.
- processes. Character vector with components describing preprocessing; may include modelType.
- featurelistName. Character string giving the name of the featurelist on which the model is based.
- projectId. Character string giving the unique alphanumeric identifier for the project.
- samplePct. Numeric or NULL. The percentage of the project dataset used in training the model. If the project uses datetime partitioning, the `samplePct` will be NULL. See `trainingRowCount`, `trainingDuration`, and `trainingStartDate` and `trainingEndDate` instead.
- trainingRowCount. Integer. 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 `trainingDuration` or `trainingStartDate` and `trainingEndDate` was used to determine that instead.
- isFrozen. Logical : is model created with frozen tuning parameters.
- modelType. Character string describing the model type.

- metrics. List with one element for each valid metric associated with the model. Each element is a list with elements for each possible evaluation type (holdout, validation, and crossValidation).
- modelCategory. Character string giving model category (e.g., blend, model).
- blueprintId. Character string giving the unique DataRobot blueprint identifier on which the model is based.
- modelId. Character string giving the unique alphanumeric model identifier.
- projectName. Character string: optional description of project defined by projectId.
- projectTarget. Character string defining the target variable predicted by all models in the project.
- projectMetric. Character string defining the fitting metric optimized by all project models.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  GetFrozenModel(projectId, modelId)

## End(Not run)
```

`GetFrozenModelFromJobId`

Retrieve a frozen model defined by modelJobId

Description

The function RequestFrozenModel initiate the creation of frozen models in a DataRobot project. RequestFrozenModel function submit requests to the DataRobot modeling engine and return an integer-valued modelJobId. The GetFrozenModelFromJobId function polls the modeling engine until the model has been built or a specified time limit is exceeded, returning an S3 object of class 'dataRobotFrozenModel' when the model is available.

Usage

```
GetFrozenModelFromJobId(project, modelJobId, maxWait = 600)
```

Arguments

`project`	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
`modelJobId`	integer. The integer returned by either `RequestNewModel` or `RequestSampleSizeUpdate`.
`maxWait`	integer. The maximum time (in seconds) to wait for the model job to complete.

Details

Motivation for this function is the fact that some models - e.g., very complex machine learning models fit to large datasets - may take a long time to complete. Splitting the model creation request from model retrieval in these cases allows the user to perform other interactive R session tasks between the time the model creation/update request is made and the time the final model is available.

GetModelFromJobId also can be used to retrieve some information about frozen model, however then some frozen specific information (parentModelId) will not be returned.

Value

An S3 object of class 'dataRobotFrozenModel' summarizing all available information about the model.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  initialJobs <- GetModelJobs(project)
  job <- initialJobs[[1]]
  modelJobId <- job$modelJobId
  GetModelJobFromJobId(projectId, modelJobId)

## End(Not run)
```

GetJob *Request information about a job*

Description

Request information about a job

Usage

```
GetJob(project, jobId)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
jobId	Character string specifying the job id

Value

list with following elements:

status job status; an element of JobStatus, e.g. JobStatus$Queue

url Character string: URL to request more detail about the job

id Character string specifying the job id

jobType Job type. See JobType for valid values

projectId Character string specifying the project that contains the model

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  initialJobs <- GetModelJobs(project)
  job <- initialJobs[[1]]
  jobId <- job$modelJobId
  GetJob(projectId, jobId)

## End(Not run)
```

GetLiftChart *Retrieve lift chart data for a model for a data partition (see DataPartition)*

Description

Retrieve lift chart data for a model for a data partition (see DataPartition)

Usage

```
GetLiftChart(model, source = DataPartition$VALIDATION)
```

Arguments

model	An S3 object of class dataRobotModel like that returned by the function GetModel, or each element of the list returned by the function ListModels.
source	Data partition for which lift chart data would be returned. Default is DataPartition$VALIDATION (see DataPartition)

Value

data.frame with the following components:

- binWeight. Numeric: weight of the bin. For weighted projects, the sum of the weights of all rows in the bin; otherwise, the number of rows in the bin.
- actual. Numeric: sum of actual target values in bin.
- predicted. Numeric: sum of predicted target values in bin.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  model <- GetModel(projectId, modelId)
  GetLiftChart(model, source = DataPartition$VALIDATION)

## End(Not run)
```

`GetModel` *Retrieve the details of a specified model*

Description

This function returns a DataRobot S3 object of class dataRobotModel for the model defined by project and modelId.

Usage

```
GetModel(project, modelId)
```

Arguments

`project`	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
`modelId`	character. Unique alphanumeric identifier for the model of interest.

Details

The S3 object returned by this function is required by the functions DeleteModel, ListModelFeatures, and RequestSampleSizeUpdate.

Value

An S3 object of class 'dataRobotModel', which is a list with the following components:

- featurelistId. Character string: unique alphanumeric identifier for the featurelist on which the model is based.
- processes. Character vector with components describing preprocessing; may include modelType.
- featurelistName. Character string giving the name of the featurelist on which the model is based.
- projectId. Character string giving the unique alphanumeric identifier for the project.
- samplePct. Numeric or NULL. The percentage of the project dataset used in training the model. If the project uses datetime partitioning, the `samplePct` will be NULL. See `trainingRowCount`, `trainingDuration`, and `trainingStartDate` and `trainingEndDate` instead.

- trainingRowCount. Integer. 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 `trainingDuration` or `trainingStartDate` and `trainingEndDate` was used to determine that instead.
- isFrozen. Logical : is model created with frozen tuning parameters.
- modelType. Character string describing the model type.
- metrics. List with one element for each valid metric associated with the model. Each element is a list with elements for each possible evaluation type (holdout, validation, and crossValidation).
- modelCategory. Character string giving model category (e.g., blend, model).
- blueprintId. Character string giving the unique DataRobot blueprint identifier on which the model is based.
- modelId. Character string giving the unique alphanumeric model identifier.
- projectName. Character string: optional description of project defined by projectId.
- projectTarget. Character string defining the target variable predicted by all models in the project.
- projectMetric. Character string defining the fitting metric optimized by all project models.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  GetModel(projectId, modelId)

## End(Not run)
```

`GetModelBlueprintChart`

Retrieve a model blueprint chart

Description

A model blueprint is a "pruned down" blueprint representing what was actually run for the model. This is solely the branches of the blueprint that were executed based on the featurelist.

Usage

```
GetModelBlueprintChart(project, modelId)
```

Arguments

`project`	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
`modelId`	character. Unique alphanumeric identifier for the model of interest.

Value

List with the following two components:

- nodes. list each element contains information about one node of a blueprint : id and label.
- edges. Two column matrix, identifying blueprint nodes connections.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  GetModelBlueprintChart(projectId, modelId)

## End(Not run)
```

GetModelBlueprintDocumentation

Get documentation for tasks used in the model blueprint

Description

A model blueprint is a "pruned down" blueprint representing what was actually run for the model. This is solely the branches of the blueprint that were executed based on the featurelist.

Usage

```
GetModelBlueprintDocumentation(project, modelId)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
modelId	character. Unique alphanumeric identifier for the model of interest.

Value

list with following components

task Character string name of the task described in document

description Character string task description

title Character string title of document

parameters List of parameters that task can received in human-readable format with following components: name, type, description

links List of exteranl lins used in document with following components: name, url

references List of references used in document with following components: name, url

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  GetModelBlueprintDocumentation(projectId, modelId)

## End(Not run)
```

`GetModelFromJobId` *Retrieve a new or updated model defined by modelJobId*

Description

The functions RequestNewModel and RequestSampleSizeUpdate initiate the creation of new models in a DataRobot project. Both functions submit requests to the DataRobot modeling engine and return an integer-valued modelJobId. The GetModelFromJobId function polls the modeling engine until the model has been built or a specified time limit is exceeded, returning an S3 object of class 'dataRobotModel' when the model is available.

Usage

```
GetModelFromJobId(project, modelJobId, maxWait = 600)
```

Arguments

`project`	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
`modelJobId`	The integer returned by either RequestNewModel or RequestSampleSizeUpdate.
`maxWait`	Integer, The maximum time (in seconds) to wait for the model job to complete

Details

Motivation for this function is the fact that some models - e.g., very complex machine learning models fit to large datasets - may take a long time to complete. Splitting the model creation request from model retrieval in these cases allows the user to perform other interactive R session tasks between the time the model creation/update request is made and the time the final model is available.

Value

An S3 object of class 'dataRobotModel' summarizing all available information about the model.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  initialJobs <- GetModelJobs(project)
  job <- initialJobs[[1]]
  modelJobId <- job$modelJobId
  GetModelJobFromJobId(projectId, modelJobId)

## End(Not run)
```

GetModelingFeaturelist

Retrieve a specific modeling featurelist from a DataRobot project

---

Description

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.

Usage

```
GetModelingFeaturelist(project, featurelistId)
```

Arguments

`project`	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
`featurelistId`	Unique alphanumeric identifier for the featurelist to be retrieved.

Value

A list with the following four elements describing the requested featurelist:

- featurelistId. Character string giving the unique alphanumeric identifier for the featurelist.
- projectId. Character string identifying the project to which the featurelist belongs.
- features. Character vector with the names of the variables included in the featurelist.
- name. Character string giving the name of the featurelist.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  featureList <- CreateModelingFeaturelist(projectId, "myFeaturelist", c("feature1", "feature2"))
  featurelistId <- featureList$featurelistId
  GetModelingFeaturelist(projectId, featurelistId)

## End(Not run)
```

`GetModelJob` *Request information about a single model job*

Description

Request information about a single model job

Usage

```
GetModelJob(project, modelJobId)
```

Arguments

`project`	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
`modelJobId`	Character string specifying the job id

Value

list with following elements:

- status. Model job status; an element of JobStatus, e.g. JobStatus$Queue.
- processes. List of character vectors describing any preprocessing applied.
- projectId. Character string giving the unique identifier for the project.
- samplePct. Numeric: the percentage of the dataset used for model building.
- trainingRowCount. Integer. The number of rows of the project dataset used in training the model.
- modelType. Character string specifying the model this job builds.
- modelCategory. Character string: what kind of model this is - 'prime' for DataRobot Prime models, 'blend' for blender models, and 'model' for other models.
- featurelistId. Character string: id of the featurelist used in fitting the model.
- blueprintId. Character string: id of the DataRobot blueprint on which the model is based.
- modelJobId. Character: id of the job.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  initialJobs <- GetModelJobs(project)
  job <- initialJobs[[1]]
  modelJobId <- job$modelJobId
  GetModelJob(projectId, modelJobId)

## End(Not run)
```

`GetModelJobs` *Retrieve status of Autopilot modeling jobs that are not complete*

Description

This function requests information on DataRobot Autopilot modeling tasks that are not complete, for one of three reasons: the task is running and has not yet completed; the task is queued and has not yet been started; or, the task has terminated due to an error.

Usage

```
GetModelJobs(project, status = NULL)
```

Arguments

`project`	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
`status`	character. The status of the desired jobs: one of JobStatus$Queue, JobStatus$InProgress, orJobStatus$Error. If NULL (default), queued and inprogress jobs are returned.

Details

The jobStatus variable specifies which of the three groups of modeling tasks is of interest. Specifically, if jobStatus has the value 'inprogress', the request returns information about modeling tasks that are running but not yet complete; if jobStatus has the value 'queue', the request returns information about modeling tasks that are scheduled to run but have not yet started; if jobStatus has the value 'error', the request returns information about modeling tasks that have terminated due to an error. By default, jobStatus is NULL, which means jobs with status "inprogress" or "queue" are returned, but not those with status "error".

Value

A list of lists with one element for each modeling task in the group being queried; if there are no tasks in the class being queried, an empty list is returned. If the group is not empty, a list is returned with the following nine elements:

- status. Prediction job status; an element of JobStatus, e.g. JobStatus$Queue.

- processes. List of character vectors describing any preprocessing applied.
- projectId. Character string giving the unique identifier for the project.
- samplePct. Numeric: the percentage of the dataset used for model building.
- modelType. Character string specifying the model type.
- modelCategory. Character string: what kind of model this is - 'prime' for DataRobot Prime models, 'blend' for blender models, and 'model' for other models.
- featurelistId. Character string: id of the featurelist used in fitting the model.
- blueprintId. Character string: id of the DataRobot blueprint on which the model is based.
- modelJobId. Character: id of the job.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  GetModelJobs(projectId)

## End(Not run)
```

`GetModelParameters` *Retrieve model parameters*

Description

Retrieve model parameters

Usage

```
GetModelParameters(project, modelId)
```

Arguments

`project`	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
`modelId`	character. Unique alphanumeric identifier for the model of interest.

Value

List with the following components:

- parameters. List of model parameters that are related to the whole model with following components: name, value.
- derivedFeatures. List containing preprocessing information about derived features with following components: originalFeature, derivedFeature, type, coefficient, transformations and stageCoefficients. 'transformations' is a list itself with components: name and value. 'stageCoefficients' is also a list with components: stage and coefficient. It contains coefficients for each stage of multistage models and is empty list for single stage models.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  GetModelParameters(projectId, modelId)

## End(Not run)
```

`GetPredictions` *Retrieve model predictions from predictJobId*

Description

This function is called with a project descriptor and an integer predictJobId, obtained from an earlier call to `RequestPredictionsForDataset`. It returns the predictions generated for the model and data specified in this prior function call.

Usage

```
GetPredictions(project, predictJobId, type = "response",
  classPrefix = "class_", maxWait = 600)
```

Arguments

`project`	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
`predictJobId`	integer. Id identifying the prediction job created by the call to `RequestPredictionsForDataset`.
`type`	character. String specifying the type of response for binary classifiers; see Details.
`classPrefix`	character. For multiclass projects returning prediction probabilities, this prefix is prepended to each class in the header of the dataframe. Defaults to "class_".
`maxWait`	integer. The maximum time (in seconds) to wait for the prediction job to complete.

Details

The contents of the return vector depends on both the modeling task - binary classification, multiclass classification, or regression - and the value of the type parameter. For regression tasks, the type parameter is ignored and a vector of numerical predictions of the response variable is returned.

For binary classification tasks, either a vector of predicted responses is returned if type has the value "response" (the default), or a vector of probabilities for the positive class is returned, if type is "probability".

For multiclass classification tasks, "response" will return the predicted class and "probability" will return the probability of each class.

This function will error if the requested job has errored, or if it isn't complete within maxWait seconds.

Value

Vector of predictions, depending on the modeling task ("Binary", "Multiclass", or "Regression") and the value of the type parameter; see Details.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  initialJobs <- GetPredictJobs(project)
  job <- initialJobs[[1]]
  predictJobId <- job$predictJobId
  GetPredictions(projectId, predictJobId)

## End(Not run)
```

GetPredictJob *Request information about a predict job*

Description

Request information about a predict job

Usage

```
GetPredictJob(project, predictJobId)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
predictJobId	Character string specifying the job id

Value

list with following elements:

status Prediction job status; an element of JobStatus, e.g. JobStatus$Queue

predictJobId Character string specifying the job id

modelId Character string specifying the model from which predictions have been requested

projectId Character string specifying the project that contains the model

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  initialJobs <- GetPredictJobs(project)
  job <- initialJobs[[1]]
  predictJobId <- job$predictJobId
  GetPredictJob(projectId, predictJobId)

## End(Not run)
```

`GetPredictJobs` *Function to list all prediction jobs in a project*

Description

Function to list all prediction jobs in a project

Usage

```
GetPredictJobs(project, status = NULL)
```

Arguments

`project`	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
`status`	character. The status of the desired jobs: one of JobStatus$Queue, JobStatus$InProgress, orJobStatus$Error. If NULL (default), queued and inprogress jobs are returned.

Value

Dataframe with one row for each prediction job in the queue, with the following columns:

status Prediction job status; one of JobStatus$Queue, JobStatus$InProgress, or JobStatus$Error

predictJobId Character string specifying the job id

modelId Character string specifying the model from which predictions have been requested

projectId Character string specifying the project that contains the model

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  GetPredictJobs(projectId)

## End(Not run)
```

GetPrimeEligibility *Check if model can be approximated with DataRobot Prime*

Description

Check if model can be approximated with DataRobot Prime

Usage

```
GetPrimeEligibility(project, modelId)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
modelId	character. Unique alphanumeric identifier for the model of interest.

Value

list with two members:

- canMakePrime logical. TRUE if model can be approximated using DataRobot Prime, FALSE if model can not be approximated.
- message character. Provides information why model may not be approximated with DataRobot Prime.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  GetPrimeEligibility(projectId, modelId)

## End(Not run)
```

GetPrimeFile *Retrieve a specific Prime file from a DataRobot project*

Description

This function returns information about specified Prime file from a specified project.

Usage

```
GetPrimeFile(project, primeFileId)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
primeFileId	numeric. Unique alphanumeric identifier for the primeFile to be retrieved.

Value

List with following elements:

language Character string. Code programming language

isValid logical flag indicating if code passed validation

rulesetId Integer identifier for the ruleset

parentModelId Unique alphanumeric identifier for the parent model

projectId Unique alphanumeric identifier for the project

id Unique alphanumeric identifier for the Prime file

modelId Unique alphanumeric identifier for the model

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  primeFiles <- ListPrimeFiles(projectId)
  primeFile <- primeFiles[[1]]
  primeFileId <- primeFile$id
  GetPrimeFile(projectId, primeFileId)

## End(Not run)
```

GetPrimeFileFromJobId *Retrieve a specific Prime file from a DataRobot project for corresponding jobId*

Description

Retrieve a specific Prime file from a DataRobot project for corresponding jobId

Usage

```
GetPrimeFileFromJobId(project, jobId, maxWait = 600)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
jobId	numeric. Unique integer identifier (return for example by RequestPrimeModel)
maxWait	numeric. maximum time to wait (in sec) before job completed.

Value

List with following elements:

language Character string. Code programming language

isValid logical flag indicating if code passed validation

rulesetId Integer identifier for the ruleset

parentModelId Unique alphanumeric identifier for the parent model

projectId Unique alphanumeric identifier for the project

id Unique alphanumeric identifier for the Prime file

modelId Unique alphanumeric identifier for the model

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  initialJobs <- GetModelJobs(project)
  job <- initialJobs[[1]]
  modelJobId <- job$modelJobId
  GetPrimeFileFromJobId(projectId, modelJobId)

## End(Not run)
```

GetPrimeModel *Retrieve information about specified DataRobot Prime model.*

Description

This function requests the DataRobot Prime model information for the DataRobot project specified by the project argument, and modelId.

Usage

```
GetPrimeModel(project, modelId)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
modelId	character. Unique alphanumeric identifier for the model of interest.

Details

The function returns list containing information about specified DataRobot Prime model.

Value

list containing information about specified DataRobot Prime model.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  GetPrimeModel(projectId, modelId)

## End(Not run)
```

`GetPrimeModelFromJobId`

Retrieve information about specified DataRobot Prime model using corresponding jobId.

Description

Retrieve information about specified DataRobot Prime model using corresponding jobId.

Usage

```
GetPrimeModelFromJobId(project, jobId, maxWait = 600)
```

Arguments

`project`	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
`jobId`	Unique integer identifier (return for example by RequestPrimeModel)
`maxWait`	maximum time to wait (in sec) before job completed

Value

list containing informatione about specified DataRobot Prime model

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  initialJobs <- GetModelJobs(project)
  job <- initialJobs[[1]]
  modelJobId <- job$modelJobId
  GetPrimeModelFromJobId(projectId, modelJobId)

## End(Not run)
```

`GetProject` *Retrieve dtails about a specified DataRobot modeling project*

Description

Returns a list of details about the DataRobot modeling project specified by project.

Usage

```
GetProject(project)
```

Arguments

`project`	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.

Value

An S3 object of class 'dataRobotProject', consisting of the following elements:

- projectId. Character string giving the unique project identifier.
- projectName. Character string giving the name assigned to the project.
- fileName. Character string giving the name of the modeling dataset for the project.
- stage. Character string describing the stage of the DataRobot Autopilot.
- autopilotMode. Numeric: 0 for fully automatic mode; 1 for semi-automatic mode; 2 for manual mode.
- created. Character string representation of the project creation time and date.
- target. Name of the target variable from fileName.
- metric. Character string specifying the metric optimized by all project models.
- partition. A 7-element list describing the data partitioning for model fitting and cross validation.
- recommender. A 3-element list with information specific to recommender models.
- advancedOptions. A 4-element list with advanced option specifications.
- positiveClass. Character string: name of positive class for binary response models.
- maxTrainPct. 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 a project.
- maxTrainRows. 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.
- scaleoutMaxTrainPct. 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 `maxTrainPct`, in which case only scaleout models can be trained up to this point.
- scaleoutMaxTrainRows. The maximum number of rows that can be used to successfully train a scaleout model without going into the validation data. May exceed `maxTrainRows`, in which case only scaleout models can be trained up to this point.

- holdoutUnlocked. A logical flag indicating whether the holdout dataset has been used for model evaluation.
- targetType. Character string specifying the type of modeling problem (e.g., regression or binary classification).

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  GetProject(projectId)

## End(Not run)
```

`GetProjectList` *Retrieve a list of all DataRobot projects*

Description

This function returns an S3 object of class projectSummaryList that describes all DataRobot modeling projects available to the user. This list may be converted into a dataframe with the as.data.frame method for this class of S3 objects.

Usage

```
GetProjectList()
```

Value

An S3 object of class 'projectSummaryList', consisting of the following elements:

- projectId. List of character strings giving the unique DataRobot identifier for each project.
- projectName. List of character strings giving the user-supplied project names.
- fileName. List of character strings giving the name of the modeling dataset for each project.
- stage. List of character strings specifying each project's Autopilot stage (e.g., 'aim' is necessary to set target).
- autopilotMode. List of integers specifying the Autopilot mode (0 = fully automatic, 1 = semi-automatic, 2 = manual).
- created. List of character strings giving the project creation time and date.
- target. List of character strings giving the name of the target variable for each project.
- metric. List of character strings identifying the fitting metric optimized for each project.
- partition. Dataframe with one row for each project and 12 columns specifying partitioning details.
- recommender. Dataframe with one row for each project and 3 columns characterizing recommender projects.

- advancedOptions. Dataframe with one row for each project and 4 columns specifying values for advanced option parameters.
- positiveClass. Character string identifying the positive target class for binary classification projects.
- maxTrainPct. 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 a project.
- maxTrainRows. 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.
- scaleoutMaxTrainPct. 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 `maxTrainPct`, in which case only scaleout models can be trained up to this point.
- scaleoutMaxTrainRows. The maximum number of rows that can be used to successfully train a scaleout model without going into the validation data. May exceed `maxTrainRows`, in which case only scaleout models can be trained up to this point.
- holdoutUnlocked. Logical flag indicating whether holdout subset results have been computed.
- targetType. Character string giving the type of modeling project (e.g., regression or binary classification).

Examples

```
## Not run:
  GetProjectList()

## End(Not run)
```

`GetProjectStatus` *Request Autopilot status for a specified DataRobot project*

Description

This function polls the DataRobot Autopilot for the status of the project specified by the project parameter.

Usage

```
GetProjectStatus(project)
```

Arguments

`project`	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.

Value

List with the following three components:

autopilotDone Logical flag indicating whether the Autopilot has completed

stage Character string specifying the Autopilot stage

stageDescription Character string interpreting the Autopilot stage value

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  GetProjectStatus(projectId)

## End(Not run)
```

GetRatingTable *Retrieve a single rating table.*

Description

Retrieve a single rating table.

Usage

```
GetRatingTable(project, ratingTableId)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
ratingTableId	character. The ID of the rating table.

Value

An S3 object of class 'dataRobotRatingTable' summarizing all available information about the rating table.

Examples

```
## Not run:
 projectId <- "5984b4d7100d2b31c1166529"
 ratingTableId <- "5984b4d7100d2b31c1166529"
 GetRatingTable(projectId, ratingTableId)

## End(Not run)
```

GetRatingTableFromJobId *Get a rating table from the rating table job metadata.*

Description

Get a rating table from the rating table job metadata.

Usage

```
GetRatingTableFromJobId(project, ratingTableJobId, maxWait = 600)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
ratingTableJobId	integer. The job ID returned by CreateRatingTable.
maxWait	integer. The maximum time (in seconds) to wait for the retrieve to complete.

Value

An S3 object of class 'dataRobotRatingTable' summarizing all available information about the rating table.

Examples

```
## Not run:
   projectId <- "5984b4d7100d2b31c1166529"
   modelId <- "5984b4d7100d2b31c1166529"
  ratingTableJobId <- CreateRatingTable(projectId, modelId, dataSource = "myRatingTable.csv")
   GetRatingTableFromJobId(projectId, ratingTableJobId)

## End(Not run)
```

GetRatingTableModel *Retrieve information about specified model with a rating table.*

Description

Retrieve information about specified model with a rating table.

Usage

```
GetRatingTableModel(project, modelId)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
modelId	character. Unique alphanumeric identifier for the model of interest.

Value

list containing information about specified model with a rating table.

Examples

```
## Not run:
 projectId <- "5984b4d7100d2b31c1166529"
 modelId <- "5984b4d7100d2b31c1166529"
 GetRatingTableModel(projectId, modelId)

## End(Not run)
```

GetRatingTableModelFromJobId

Retrieve a new or updated rating table model defined by a job ID.

Description

Retrieve a new or updated rating table model defined by a job ID.

Usage

```
GetRatingTableModelFromJobId(project, ratingTableModelJobId, maxWait = 600)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
ratingTableModelJobId	integer. The ID returned by `RequestNewRatingTableModel`.
maxWait	integer. The maximum time (in seconds) to wait for the retrieve to complete.

Value

An S3 object of class 'dataRobotRatingTableModel' summarizing all available information about the model.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  ratingTableId <- "5984b4d7100d2b31c1166529"
  ratingTableModelJobId <- RequestNewModel(projectId, ratingTableId)
  GetRatingTableModelFromJobId(project, ratingTableModelJobId)

## End(Not run)
```

GetReasonCodesInitialization

Retrieve the reason codes initialization for a model.

Description

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.

Usage

```
GetReasonCodesInitialization(model)
```

Arguments

model	An S3 object of class dataRobotModel like that returned by the function GetModel, or each element of the list returned by the function ListModels.

Value

A named list which contains:

- projectId. Character id of the project the feature belonges to.
- modelId. Character string giving the unique alphanumeric model identifier.
- reasonCodesSample. list which contains sample of reason codes. Each element of the list is information about reason codes for one data row. For more information see GetReasonCodesRows.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  model <- GetModel(projectId, modelId)
  GetReasonCodesInitialization(model)

## End(Not run)
```

GetReasonCodesInitializationFromJobId *Retrieve the reason codes initialization for a model using jobId*

Description

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.

Usage

```
GetReasonCodesInitializationFromJobId(project, jobId, maxWait = 600)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
jobId	integer. Unique integer identifier pointing to the reason codes job (returned for example by RequestReasonCodesInitialization.)
maxWait	Integer, The maximum time (in seconds) to wait for the model job to complete

Value

A named list which contains:

- projectId. Character id of the project the feature belonges to.
- modelId. Character string giving the unique alphanumeric model identifier.
- reasonCodesSample. list which contains sample of reason codes. Each element of the list is information about reason codes for one data row. For more information see GetReasonCodesRows.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  model <- GetModel(projectId, modelId)
  jobId <- RequestReasonCodesInitialization(model)
  GetReasonCodesInitializationFromJobId(projectId, jobId)

## End(Not run)
```

GetReasonCodesMetadata

Retrieve metadata for specified reason codes

Description

Retrieve metadata for specified reason codes

Usage

```
GetReasonCodesMetadata(project, reasonCodeId)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
reasonCodeId	character. id of the reason codes.

Value

A named list which contains reason code metadata:

- id. Character string id of the record and reason codes computation result.
- projectId. Character string id of the project the model belongs to.
- modelId. Character string id of the model reason codes initialization is for.
- datasetId. Character string id of the prediction dataset reason codes were computed for.
- maxCodes. Integer maximum number of reason codes to supply per row of the dataset.
- thresholdLow. Numeric the low threshold, below which a prediction must score in order for reason codes to be computed for a row in the dataset.
- thresholdHigh. Numeric the high threshold, above which a prediction must score in order for reason codes to be computed for a row in the dataset.
- numColumns. Integer the number of columns reason codes were computed for.
- finishTime. Numeric timestamp referencing when computation for these reason codes finished.
- reasonCodesLocation. Character string where to retrieve the reason codes.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  datasets <- ListPredictionDatasets(projectId)
  dataset <- datasets[[1]]
  datasetId <- dataset$id
  model <- GetModel(model, datasetId)
```

```
  jobId <- RequestReasonCodes(model, datasetId)
  reasonCodeId <- GetReasonCodesMetadataFromJobId(projectId, jobId)$id
  GetReasonCodesMetadata(projectId, reasonCodeId)

## End(Not run)
```

`GetReasonCodesMetadataFromJobId` *Retrieve the reason codes metadata for a model using jobId*

Description

Retrieve the reason codes metadata for a model using jobId

Usage

```
GetReasonCodesMetadataFromJobId(project, jobId, maxWait = 600)
```

Arguments

`project`	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
`jobId`	Unique integer identifier (return for example by `RequestReasonCodes`).
`maxWait`	Integer, The maximum time (in seconds) to wait for the model job to complete.

Value

A named list which contains reason code metadata. For more information see `GetReasonCodesMetadata`.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  datasets <- ListPredictionDatasets(projectId)
  dataset <- datasets[[1]]
  datasetId <- dataset$id
  model <- GetModel(model, datasetId)
  jobId <- RequestReasonCodes(model, datasetId)
  GetReasonCodesMetadataFromJobId(projectId, jobId)

## End(Not run)
```

GetReasonCodesRows *Retrieve all reason codes rows*

Description

Retrieve all reason codes rows

Usage

```
GetReasonCodesRows(project, reasonCodeId, batchSize = NULL,
  excludeAdjustedPredictions = TRUE)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
reasonCodeId	character. id of the reason codes.
batchSize	integer. Optional. Maximum number of reason codes rows to retrieve per request
excludeAdjustedPredictions	logical. Optional. Set to FALSE to include adjusted predictions, which are predictions adjusted by an exposure column. This is only relevant for projects that use an exposure column.

Value

list of raw reason codes, each element corresponds to a row of the prediction dataset

- rowId. Character string row Id.
- prediction. prediction for the row.
- predictionValues. list containing
  - 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.
- adjustedPrediction. adjusted predictions, if they are not excluded.
- adjustedPredictionValues. Similar to predictionValues, but for adjusted predictions, if they are not excluded.
- reasonCodes. list contaning
  - 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
  - feature. the name of the feature contributing to the prediction.
  - featureValue. the value the feature took on for this row

- strength. the amount this feature's value affected the prediction
- qualitativateStrength. a human-readable description of how strongly the feature affected the prediction (e.g. '+++', '–', '+').

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  datasets <- ListPredictionDatasets(projectId)
  dataset <- datasets[[1]]
  datasetId <- dataset$id
  model <- GetModel(model, datasetId)
  jobId <- RequestReasonCodes(model, datasetId)
  reasonCodeId <- GetReasonCodesMetadataFromJobId(projectId, jobId)$id
  GetReasonCodesRows(projectId, reasonCodeId)

## End(Not run)
```

GetRocCurve *Retrieve ROC curve data for a model for a particular data partition (see DataPartition)*

Description

Retrieve ROC curve data for a model for a particular data partition (see DataPartition)

Usage

```
GetRocCurve(model, source = DataPartition$VALIDATION)
```

Arguments

model	An S3 object of class dataRobotModel like that returned by the function GetModel, or each element of the list returned by the function ListModels.
source	character. Data partition to retrieve ROC curve data.Default is

Value

list with the following components:

- source. Character: data partition for which ROC curve data is returned (see DataPartition).
- negativeClassPredictions. Numeric: example predictions for the negative class.
- rocPoints. data.frame: each row represents pre-calculated metrics (accuracy, f1_score, false_negative_score, true_negative_score, true_positive_score, false_positive_score, true_negative_rate, false_positive_rate, true_positive_rate, matthews_correlation_coefficient, positive_predictive_value, negative_predictive_value, threshold) associated with different thresholds for the ROC curve.
- positiveClassPredictions. Numeric: example predictions for the positve class.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  model <- GetModel(projectId, modelId)
  GetRocCurve(model)

## End(Not run)
```

`GetRulesets` *List the rulesets approximating a model generated by DataRobot Prime*

Description

This function wll return list of rulesets that could be used to approximate the specified model. Rulesets are created using the RequestApproximation function. If model hasn't been approximated yet, will return empty list

Usage

```
GetRulesets(project, modelId)
```

Arguments

`project`	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
`modelId`	Unique alphanumeric identifier for the model of interest.

Value

A list of lists with one element for each ruleset. If there are no rulesets created for a model then an empty list is returned If the group is not empty,a list is returned with the following elements:

- projectId. Character string giving the unique identifier for the project.
- rulesetId. Integer number giving the identifier for the ruleset.
- score. Score of ruleset (using project leaderboard metric).
- parentModelId. Character string giving the unique identifier for the parent model.
- ruleCount. integer: number of rules in ruleset.
- modelId. Character string giving the unique identifier for a model using the ruleset. May be NULL if no model using the ruleset has been created yet.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  GetRulesets(projectId, modelId)

## End(Not run)
```

`GetServerDataInRows` *Handle server side pagination.*

Description

Handle server side pagination.

Usage

```
GetServerDataInRows(serverData, batchSize = 50)
```

Arguments

`serverData`	list. Raw JSON parsed list returned from the server.
`batchSize`	integer. The number of requests per page to expect.

`GetTrainingPredictionDataFrame`
Simplify the training prediction rows into a tidy format dataframe.

Description

Simplify the training prediction rows into a tidy format dataframe.

Usage

```
GetTrainingPredictionDataFrame(rows)
```

Arguments

`rows`	data.frame. The dataframe to tidy.

GetTrainingPredictions

Retrieve training predictions on a specified data set.

Description

Retrieve training predictions on a specified data set.

Usage

```
GetTrainingPredictions(project, predictionId)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
predictionId	character. ID of the prediction to retrieve training predictions for.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  predictions <- ListTrainingPredictions(projectId)
  predictionId <- predictions[[1]]$id
  trainingPredictions <- GetTrainingPredictions(projectId, predictionId)

## End(Not run)
```

GetTrainingPredictionsFromJobId

Retrieve the training predictions for a model using a job id.

Description

Retrieve the training predictions for a model using a job id.

Usage

```
GetTrainingPredictionsFromJobId(project, jobId, maxWait = 600)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
jobId	Unique integer identifier (return for example by RequestReasonCodes).
maxWait	Integer, The maximum time (in seconds) to wait for the model job to complete.

Value

A dataframe with out-of-fold predictions for the training data.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  model <- GetModel(model, datasetId)
  jobId <- RequestTrainingPredictions(model, dataSubset = "all")
  trainingPredictions <- GetTrainingPredictionsFromJobId(projectId, jobId)

## End(Not run)
```

GetTransferrableModel *Retrieve imported model info using import id*

Description

Retrieve imported model info using import id

Usage

```
GetTransferrableModel(importId)
```

Arguments

importId	character. Id of the import.

Value

A list describing uploaded transferrable model with the following components:

- note. Character string Manually added node about this imported model.
- datasetName. Character string Filename of the dataset used to create the project the model belonged to.
- modelName. Character string Model type describing the model generated by DataRobot.
- displayName. Character string Manually specified human-readable name of the imported model.
- target. Character string The target of the project the model belonged to prior to export.
- projectName. Character string Name of the project the model belonged to prior to export.
- importedByUsername. Character string Username of the user who imported the model.
- importedAt. Character string The time the model was imported.
- version. Numeric Project version of the project the model belonged to.
- projectId. Character id of the project the model belonged to prior to export.

- featurelistName. Character string Name of the featurelist used to train the model.
- createdByUsername. Character string Username of the user who created the model prior to export.
- importedById. Character string id of the user who imported the model.
- id. Character string id of the import.
- createdById. Character string id of the user who created the model prior to export.
- modelId. Character string original id of the model prior to export.
- originUrl. Character string URL.

Examples

```
## Not run:
  id <- UploadTransferrableModel("model.drmodel")
  GetTransferrableModel(id)

## End(Not run)
```

`GetValidMetrics` *Retrieve the valid fitting metrics for a specified project and target*

Description

For the response variable defined by the character string target and the project defined by the parameter project, return the vector of metric names that can be specified for fitting models in this project. This function is intended for use after SetupProject has been run but before SetTarget, allowing the user to specify valid non-default values for the metric parameter.

Usage

```
GetValidMetrics(project, target)
```

Arguments

`project`	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
`target`	character. String giving the name of the response variable to be predicted by all project models.

Value

Character vector containing the names of the metric values that are valid for a subsequent call to the SetTarget function.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  GetValidMetrics(projectId, "targetFeature")

## End(Not run)
```

`GetWordCloud` *Retrieve word cloud data for a model.*

Description

Retrieve word cloud data for a model.

Usage

```
GetWordCloud(project, modelId, excludeStopWords = FALSE)
```

Arguments

`project`	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
`modelId`	Character string: unique alphanumeric identifier for the model of interest.
`excludeStopWords`	Logical (optional) : Set to True if you want stopwords filtered out the response.

Value

data.frame with the following components:

ngram Character string: word or ngram value

coefficient Numerical: value from [-1.0, 1.0] range, describes effect of this ngram on the target. A large negative value means a strong effect toward the negative class in classification projects and a smaller predicted target value in regression projects. A large positive value means a strong effect toward the positive class and a larger predicted target value respectively

count Integer: number of rows in the training sample where this ngram appears

frequency Numerical: value from (0.0, 1.0] range, frequency of this ngram relative to the most frequent ngram

isStopword Logical: true for ngrams that DataRobot evaluates as stopwords

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  GetWordCloud(projectId, modelId)

## End(Not run)
```

JobStatus *Job statuses*

Description

This is a list that contains the valid values for job status when querying the list of jobs mode. If you wish, you can specify job status modes using the list values, e.g. JobStatus$InProgress instead of typing the string 'inprogress'. This way you can benefit from autocomplete and not have to remember the valid optons.

Usage

```
JobStatus
```

Format

An object of class `list` of length 5.

JobType *Job type*

Description

This is a list that contains the valid values for job type when querying the list of jobs.

Usage

```
JobType
```

Format

An object of class `list` of length 9.

ListBlueprints *Retrieve the list of available blueprints for a project*

Description

This function returns the list of available blueprints for a specified modeling project, as an S3 object of class listOfBlueprints; see Value.

Usage

```
ListBlueprints(project)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.

Value

An S3 object of class 'listOfBlueprints', a list with one element for each recommended blueprint in the associated project. For more information see GetBlueprint()

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  ListBlueprints(projectId)

## End(Not run)
```

ListConfusionCharts *Returns all available confusion charts for the model.*

Description

Note that the confusion chart for `source = "crossValidation"` will not be available unless cross validation has been run for that model. Also, the confusion chart for `source = "holdout"` will not be available unless the holdout has been unlocked for the project.

Usage

```
ListConfusionCharts(model)
```

Arguments

model	An S3 object of class dataRobotModel like that returned by the function GetModel, or each element of the list returned by the function ListModels.

Value

A list of all confusion charts for the model, one for each partition type found in `DataPartition`.

Examples

```
## Not run:
  modelId <- "5996f820af07fc605e81ead4"
  ListConfusionCharts(modelId)

## End(Not run)
```

`ListFeatureInfo` *Details about all features for this project*

Description

Details about all features for this project

Usage

```
ListFeatureInfo(project)
```

Arguments

`project`	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.

Value

A named list which contains:

- id nummeric. feature id. Note that throughout the API, features are specified using their names, not this ID.
- name character. The name of the feature.
- featureType character. Feature type: 'Numeric', 'Categorical', etc.
- importance numeric. numeric measure of the strength of relationship between the feature and target (independent of any model or other features).
- lowInformation logical. Whether the feature has too few values to be informative.
- uniqueCount numeric. The number of unique values in the feature.
- naCount numeric. The number of missing values in the feature.
- dateFormat character. The format of the feature if it is date-time feature.
- projectId character. Character id of the project the feature belonges to.
- max. The maximum value in the dataset, formatted in the same format as the data.
- min. The minimum value in the dataset, formatted in the same format as the data.

- mean. The arithmetic mean of the dataset, formatted in the same format as the data.
- median. The median of the dataset, formatted in the same format as the data.
- stdDev. The standard deviation of the dataset, formatted in the same format as the data.
- timeSeriesEligible logical. Whether this feature can be used as the datetime partition column in a time series project.
- timeSeriesEligibilityReason character. Why the feature is ineligible for the datetime partition column in a time series project, "suitable" when it is eligible.
- timeStep numeric. 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. NULL for features that are not time series eligible.
- timeUnit character. For time series eligible features, the time unit covered by a single time step, e.g. "HOUR", or NULL for features that are not time series eligible.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  ListFeatureInfo(projectId)

## End(Not run)
```

ListFeaturelists *Retrieve all featurelists associated with a project*

Description

This function returns an S3 object of class listOfFeaturelists that describes all featurelists (i.e., lists of modeling variables) available for the project specified by the project parameter. This list may be converted to a dataframe with the as.data.frame method for objects of class listOfFeaturelists.

Usage

```
ListFeaturelists(project)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.

Value

An S3 object of class 'listOfFeaturelists', which is a list of dataframes: each element of the list corresponds to one featurelist associated with the project, and each dataframe has one row and the following four columns:

- featurelistId. Unique alphanumeric identifier for the featurelist.
- projectId. Unique alphanumeric project identifier.
- features. Comma-separated character string listing the variables included in the featurelist.
- name. Character string giving the name of the featurelist.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  ListFeaturelists(projectId)

## End(Not run)
```

ListJobs *Retrieve information about jobs*

Description

This function requests information about the jobs that go through the DataRobot queue.

Usage

```
ListJobs(project, status = NULL)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
status	character. The status of the desired jobs: one of JobStatus$Queue, JobStatus$InProgress, orJobStatus$Error. If NULL (default), queued and inprogress jobs are returned.

Value

A list of lists with one element for each job. The named list for each job contains:

status Model job status; an element of JobStatus, e.g. JobStatus$Queue

url Character string: URL to request more detail about the job

id Character string specifying the job id

jobType Job type. See JobType for valid values

projectId Character string specifying the project that contains the model

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  ListJobs(projectId)

## End(Not run)
```

`ListModelFeatures` *Returns the list of features (i.e., variables) on which a specified model is based*

Description

This function returns the list of features (typically, response variable and raw covariates) used in building the model specified by model, an S3 object of class 'dataRobotModel'.

Usage

```
ListModelFeatures(model)
```

Arguments

`model`	An S3 object of class dataRobotModel like that returned by the function GetModel, or each element of the list returned by the function ListModels.

Value

A character vector of feature names, with one component for each model feature.

Examples

```
## Not run:
  modelId <- "5996f820af07fc605e81ead4"
  ListModelFeatures(modelId)

## End(Not run)
```

`ListModelingFeaturelists` *Retrieve all modeling featurelists associated with a project*

Description

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.

Usage

```
ListModelingFeaturelists(project)
```

Arguments

`project`	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.

Value

An S3 object of class 'listOfFeaturelists', which is a list of dataframes: each element of the list corresponds to one featurelist associated with the project, and each dataframe has one row and the following four columns:

- featurelistId. Unique alphanumeric identifier for the featurelist.
- projectId. Unique alphanumeric project identifier.
- features. Comma-separated character string listing the variables included in the featurelist.
- name. Character string giving the name of the featurelist.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  ListModelingFeaturelists(projectId)

## End(Not run)
```

ListModels *Retrieve all available model information for a DataRobot project*

Description

This function requests the model information for the DataRobot project specified by the project argument, described under Arguments. This parameter may be obtained in several ways, including: (1), from the projectId element of the list returned by GetProjectList; (2), as the object returned by the GetProject function; or (3), as the list returned by the SetupProject function. The function returns an S3 object of class 'listOfModels'.

Usage

```
ListModels(project)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.

Value

An S3 object of class listOfModels, which may be characterized using R's generic summary function or converted to a dataframe with the as.data.frame method.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  ListModels(projectId)

## End(Not run)
```

ListPredictionDatasets *Retrieve all prediction datasets associated with a project*

Description

This function returns an S3 object of class listDataRobotPredictionDataset that describes all prediction datasets available for the project specified by the project parameter. This list may be converted to a dataframe with the as.data.frame method for objects of class listDataRobotPredictionDataset.

Usage

```
ListPredictionDatasets(project)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.

Value

An S3 object of class 'listDataRobotPredictionDataset', which is a list of dataframes: each element of the list corresponds to one prediction dataset associated with the project, and each dataframe has one row and the following columns:

- id character. The unique alphanumeric identifier for the dataset.
- numColumns numeric. Number of columns in dataset.
- name character. Name of dataset file.
- created character. time of upload.
- projectId character. String giving the unique alphanumeric identifier for the project.
- numRows numeric. Number of rows in dataset.
- forecastPoint. The point relative to which predictions will be generated, based on the forecast window of the project. Only specified in time series projects, otherwise will be NULL.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  ListPredictionDatasets(projectId)

## End(Not run)
```

ListPrimeFiles *List all downloadable code files from DataRobot Prime for the project*

Description

Training a model using a ruleset is a necessary prerequisite for being able to download the code for a ruleset.

Usage

```
ListPrimeFiles(project, parentModelId = NULL, modelId = NULL)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
parentModelId	numeric. Optional. Filter for only those prime files approximating this parent model.
modelId	numeric. Optional. Filter for only those prime files with code for this prime model.

Value

List of lists. Each element of the list corresponds to one Prime file available to download. The elements of this list have the same format as the return value of GetPrimeFile.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  ListPrimeFiles(projectId)

## End(Not run)
```

`ListPrimeModels` *Retrieve information about all DataRobot Prime models for a DataRobot project*

Description

This function requests the DataRobot Prime models information for the DataRobot project specified by the project argument, described under Arguments.

Usage

```
ListPrimeModels(project)
```

Arguments

`project`	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.

Details

The function returns data.frame containing information about each DataRobot Prime model in a project (one row per Prime model)

Value

data.frame containing information about each DataRobot Prime model in a project (one row per Prime model).

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  ListPrimeModels(projectId)

## End(Not run)
```

ListRatingTableModels *Retrieve information about all DataRobot models with a rating table.*

Description

Retrieve information about all DataRobot models with a rating table.

Usage

```
ListRatingTableModels(project)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.

Value

data.frame containing information about each model with a rating table in a project (one row per model with a rating table).

Examples

```
## Not run:
 projectId <- "5984b4d7100d2b31c1166529"
 ListRatingTableModels(projectId)

## End(Not run)
```

ListRatingTables *Retrieve information about all rating tables.*

Description

Retrieve information about all rating tables.

Usage

```
ListRatingTables(project)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.

Value

data.frame containing information about each rating table in a project (one row per model with a rating table).

Examples

```
## Not run:
 projectId <- "5984b4d7100d2b31c1166529"
 ListRatingTables(projectId)

## End(Not run)
```

`ListReasonCodesMetadata` *Retrieve metadata for reason codes in specified project*

Description

Retrieve metadata for reason codes in specified project

Usage

```
ListReasonCodesMetadata(project, modelId = NULL, limit = NULL,
  offset = NULL)
```

Arguments

`project`	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
`modelId`	character. Optional. If specified, only reason codes computed for this
`limit`	integer. Optional. At most this many results are returned, default: no limit
`offset`	integer. This many results will be skipped, default: 0

Value

List of metadata for all reason codes in the project. Each element of list is metadata for one reason codes (for format see `GetReasonCodesMetadata`).

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  ListReasonCodesMetadata(projectId)

## End(Not run)
```

ListTrainingPredictions *Retrieve information about all training prediction datasets in a project.*

Description

Retrieve information about all training prediction datasets in a project.

Usage

```
ListTrainingPredictions(project)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.

Value

data.frame containing information about each training prediction.

Examples

```
## Not run:
 projectId <- "5984b4d7100d2b31c1166529"
 ListTrainingPredictions(projectId)

## End(Not run)
```

ListTransferrableModels *Retrieve information about all imported models This function returns a data.frame that describes all imported models*

Description

Retrieve information about all imported models This function returns a data.frame that describes all imported models

Usage

```
ListTransferrableModels(limit = NULL, offset = NULL)
```

Arguments

limit	integer. The number of records to return. The server will use a (possibly finite) default if not specified.
offset	integer. The number of records to skip.

Value

A data.frame describing uploaded transferrable model with the following components:

- note. Character string Manually added node about this imported model.
- datasetName. Character string Filename of the dataset used to create the project the model belonged to.
- modelName. Character string Model type describing the model generated by DataRobot.
- displayName. Character string Manually specified human-readable name of the imported model.
- target. Character string The target of the project the model belonged to prior to export.
- projectName. Character string Name of the project the model belonged to prior to export.
- importedByUsername. Character string Username of the user who imported the model.
- importedAt. Character string The time the model was imported.
- version. Numeric Project version of the project the model belonged to.
- projectId. Character id of the project the model belonged to prior to export.
- featurelistName. Character string Name of the featurelist used to train the model.
- createdByUsername. Character string Username of the user who created the model prior to export.
- importedById. Character string id of the user who imported the model.
- id. Character string id of the import.
- createdById. Character string id of the user who created the model prior to export.
- modelId. Character string original id of the model prior to export.
- originUrl. Character string URL.

Examples

```
## Not run:
  ListTransferrableModels()

## End(Not run)
```

PauseQueue *Pause the DataRobot modeling queue*

Description

This function pauses the DataRobot modeling queue for a specified project

Usage

```
PauseQueue(project)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  PauseQueue(projectId)

## End(Not run)
```

PeriodicityMaxTimeStep *Periodicity max time step*

Description

Periodicity max time step

Usage

```
PeriodicityMaxTimeStep
```

Format

An object of class `numeric` of length 1.

PeriodicityTimeUnits *Periodicity time units*

Description

Periodicity time units

Usage

```
PeriodicityTimeUnits
```

Format

An object of class `list` of length 8.

plot.listOfModels *Plot method for DataRobot S3 objects of class listOfModels*

Description

Method for R's generic plot function for DataRobot S3 objects of class listOfModels. This function generates a horizontal barplot as described under Details.

Usage

```
## S3 method for class 'listOfModels'
plot(x, y, metric = NULL, pct = NULL,
  selectRecords = NULL, orderDecreasing = NULL, textSize = 0.8,
  textColor = "black", borderColor = "blue", xpos = NULL, ...)
```

Arguments

x	S3 object of class listOfModels to be plotted.
y	Not used; included for conformance with plot() generic function parameter requirements.
metric	character. Optional. Defines the metric to be used in constructing the barplot. If NULL (the default), the validation set value for the project fitting metric is used; otherwise, this value must name one of the elements of the metrics list associated with each model in x.
pct	integer. Optional. Specifies a samplePct value used in selecting models to include in the barplot summary. If NULL (the default), all project models are included. Note, however, that this list of models is intersected with the list of models defined by the selectRecords parameter, so that only those models identified by both selectRecords and pct appear in the plot.

`selectRecords` integer. Optional. A vector that specifies the individual elements of the list x to be included in the barplot summary. If NULL (the default), all models are included. Note, however, that this list of models is intersected with the list of models defined by the pct parameter, so that only those models identified by both selectRecords and pct appear in the plot.

`orderDecreasing` logical. Optional. If TRUE, the barplot is built from the bottom up in decreasing order of the metric values; if FALSE, the barplot is built in increasing order of metric values. The default is NULL, which causes the plot to be generated in the order in which the models appear in the list x.

`textSize` numeric. Optional. Multiplicative scaling factor for the model name labels on the barplot.

`textColor` character. Optional. If character, this parameter specifies the text color used in labelling all models in the barplot; if a character vector, it specifies one color for each model in the plot.

`borderColor` character. Optional. Specifies the border color for all bars in the barplot, surrounding a transparent background.

`xpos` numeric. Optional. Defines the horizontal position of the center of all text labels on the plot. The default is NULL, which causes all text to be centered in the plot; if xpos is a single number, all text labels are centered at this position; if xpos is a vector, it specifies one center position for each model in the plot.

`...` list. Optional. Additional named parameters to be passed to R's barplot function used in generating the plot

Details

This function generates a horizontal barplot with one bar for each model characterized in the 'listOfModels' object x. The length of each bar is specified by the value of metric; if this parameter is specified as NULL (the default), the project fitting metric is used, as determined by the projectMetric value from the first element of x. Text is added to each bar in the plot, centered at the position specified by the xpos parameter, based on the value of the modelType element of each model in the list x. The size and color of these text labels may be controlled with the textSize and textColor parameters. The order in which these models appear on the plot is controlled by the choice of metric and the value of the orderDecreasing parameter, and subsets of the models appearing in the list x may be selected via the pct and selectRecords parameters.

Value

None. This function is called for its side-effect of generating a plot.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  plot(ListModels(projectId))

## End(Not run)
```

PostgreSQLdrivers *PostgreSQL drivers*

Description

This is a list that contains the valid values for PostgreSQL drivers.

Usage

```
PostgreSQLdrivers
```

Format

An object of class `list` of length 2.

PredictionDatasetFromAsyncUrl *Retrieve prediction dataset info from the dataset creation URL*

Description

If dataset creation times out, the error message includes a URL corresponding to the creation task. That URL can be passed to this function (which will return the completed dataset info details when finished) to resume waiting for creation.

Usage

```
PredictionDatasetFromAsyncUrl(asyncUrl, maxWait = 600)
```

Arguments

`asyncUrl`	The temporary status URL
`maxWait`	The maximum time to wait (in seconds) for creation before aborting.

PrimeLanguage *Prime Language*

Description

This is a list that contains the valid values for downloadable code programming languages.

Usage

```
PrimeLanguage
```

Format

An object of class `list` of length 2.

ProjectFromAsyncUrl *Retrieve a project from the project-creation URL*

Description

If project creation times out, the error message includes a URL corresponding to the project creation task. That URL can be passed to this function (which will return the completed project details when finished) to resume waiting for project creation.

Usage

```
ProjectFromAsyncUrl(asyncUrl, maxWait = 600)
```

Arguments

`asyncUrl`	The temporary status URL
`maxWait`	The maximum time to wait (in seconds) for project creation before aborting.

ReformatMetrics *replace NULL in $metrics list elements with NA*

Description

replace NULL in $metrics list elements with NA

Usage

```
ReformatMetrics(metricsList)
```

Arguments

`metricsList`	list. List of metrics to reformat.

RenameRatingTable *Renames a rating table to a different name.*

Description

Renames a rating table to a different name.

Usage

```
RenameRatingTable(project, ratingTableId, ratingTableName)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
ratingTableId	character. The ID of the rating table.
ratingTableName	character. The new name for the rating table.

Value

An S3 object of class 'dataRobotRatingTable' summarizing all available information about the renamed rating table.

Examples

```
## Not run:
  projectId <- "5984b4d7100d2b31c1166529"
  ratingTableId <- "5984b4d7100d2b31c1166529"
  RenameRatingTable(projectId, ratingTableId, "Renamed Table")

## End(Not run)
```

RequestApproximation *Request an approximation of a model using DataRobot Prime*

Description

This function wll create several rulesets that approximate the specified model. The code used in the approximation can be downloaded to be run locally. Currenly only Python and Java downloadable code is available

Usage

```
RequestApproximation(project, modelId)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
modelId	character. Unique alphanumeric identifier for the model of interest.

Details

General workflow of creating and downloading Prime code may look like following: RequestApproximation - create several rulestes that approximate the specified model GetRulesets - list all rulesests created for the parent model RequestPrimeModel - create Prime model for specified ruleset (use one of rulesets return by GetRulests) GetPrimeModelFromJobId - get PrimeModelId using JobId returned by RequestPrimeModel CreatePrimeCode - create code for one of available Prime models GetPrimeFileFromJobId - get PrimeFilelId using JobId returned by CreatePrimeCode DownloadPrimeCode - download specified Prime code file

Value

job Id

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  RequestApproximation(projectId, modelId)

## End(Not run)
```

RequestBlender *Submit a job for creating blender model. Upon success, the new job will be added to the end of the queue.*

Description

This function requests the creation of a blend of several models in specified DataRobot project. The function also allows the user to specify method used for blending. This function returns an integer modelJobId value, which can be used by the GetBlenderModelFromJobId function to return the full

Usage

```
RequestBlender(project, modelIds, blendMethod)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
modelIds	character. Vector listing the model Ids to be blended.
blendMethod	character. Parameter specifying blending method. See acceptable values within BlendMethods.

Value

An integer value that can be used as the modelJobId parameter in subsequent calls to the GetBlenderModelFromJobId function.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelsToBlend <- c("5996f820af07fc605e81ead4", "59a5ce3301e9f0296721c64c")
  RequestBlender(projectId, modelId, "GLM")

## End(Not run)
```

`RequestFeatureImpact` *Request Feature Impact to be computed.*

Description

This adds a Feature Impact job to the project queue.

Usage

```
RequestFeatureImpact(model)
```

Arguments

`model`	character. The model for which you want to compute Feature Impact, e.g. from the list of models returned by `ListModels(project)`.

Value

A job ID (character)

Examples

```
## Not run:
  model <- ListModels(project)[[1]]
  featureImpactJobId <- RequestFeatureImpact(model)
  featureImpact <- GetFeatureImpactForJobId(project, featureImpactJobId)

## End(Not run)
```

`RequestFrozenDatetimeModel` *Train a new frozen datetime model with parameters from the specified model*

Description

Requires that this model belongs to a datetime partitioned project. If it does not, an error will occur when submitting the job

Usage

```
RequestFrozenDatetimeModel(model, trainingRowCount = NULL,
  trainingDuration = NULL, trainingStartDate = NULL,
  trainingEndDate = NULL, timeWindowSamplePct = NULL)
```

Arguments

`model`	An S3 object of class dataRobotModel like that returned by the function GetModel, or each element of the list returned by the function ListModels.
`trainingRowCount`	integer. (optional) the number of rows of data that should be used to train the model.
`trainingDuration`	character. string (optional) a duration string specifying what time range the data used to train the model should span.
`trainingStartDate`	character. string(optional) the start date of the data to train to model on (" be used.
`trainingEndDate`	character. string(optional) the end date of the data to train the model on (" will be used.
`timeWindowSamplePct`	integer. (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

Details

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 to trainingRowCount and trainingDuration, frozen datetime models may be trained on an exact date range. Only one of trainingRowCount, trainingDuration, or trainingStartDate and trainingEndDate should be specified. Models specified using trainingStartDate and trainingEndDate are the only ones that can be trained into the holdout data (once the holdout is unlocked).

Value

An integer value that can be used as the modelJobId parameter in subsequent calls to the GetDatetimeModelFromJobId function.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  model <- GetDatetimeModelObject(modelId)
  RequestFrozenDatetimeModel(model)

## End(Not run)
```

`RequestFrozenModel` *Train a new frozen model with parameters from specified model*

Description

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.

Usage

```
RequestFrozenModel(model, samplePct = NULL, trainingRowCount = NULL)
```

Arguments

`model`	An S3 object of class dataRobotModel like that returned by the function GetModel, or each element of the list returned by the function ListModels.
`samplePct`	Numeric, specifying the percentage of the training dataset to be used in building the new model
`trainingRowCount`	integer. The number of rows to use to train the requested model.

Details

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, 'samplePct' and 'trainingRowCount' are assumed to be in terms of rows of the minority class.

Note : For datetime partitioned projects, use "RequestFrozenDatetimeModel' instead

Value

An integer value that can be used as the modelJobId parameter in subsequent calls to the GetModelFromJobId function.

An integer value that can be used as the modelJobId parameter in subsequent calls to the GetModelFromJobId function.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  model <- GetModel(projectId, modelId)
  RequestFrozenModel(model, samplePct = 10)

## End(Not run)
```

RequestNewDatetimeModel

Adds a new datetime model of the type specified by the blueprint to a DataRobot project

---

Description

This function requests the creation of a new datetime model in the DataRobot modeling project defined by the project parameter. The function also allows the user to specify alternatives to the project default for featurelist, samplePct, and scoringType. This function returns an integer modelJobId value, which can be used by the GetDatetimeModelFromJobId function to return the full model object.

Usage

```
RequestNewDatetimeModel(project, blueprint, featurelist = NULL,
  trainingRowCount = NULL, trainingDuration = NULL,
  timeWindowSamplePct = NULL)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
blueprint	list. A list with at least the following two elements: blueprintId and projectId. Note that the individual elements of the list returned by ListBlueprints are admissible values for this parameter.
featurelist	list. A list that contains the element featurelistId that specifies the featurelist to be used in building the model; if not specified (i.e., for the default value NULL), the project default (Informative Features) is used.

trainingRowCount
: integer. Optional, the number of rows of data that should be used to train the model. If specified, trainingDuration may not be specified.

trainingDuration
: character. String (optional) a duration string specifying what time range the data used to train the model should span. If specified, trainingRowCount may not be specified.

timeWindowSamplePct
: integer. 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.

Details

Motivation for this function is the fact that some models - e.g., very complex machine learning models fit to large datasets - may take a long time to complete. Splitting the model creation request from model retrieval in these cases allows the user to perform other interactive R session tasks between the time the model creation/update request is made and the time the final model is available.

Value

An integer value that can be used as the modelJobId parameter in subsequent calls to the GetDatetimeModelFromJobId function.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  blueprints <- ListBlueprints(projectId)
  blueprint <- blueprints[[1]]
  RequestNewDatetimeModel(projectId, blueprint)

## End(Not run)
```

RequestNewModel *Adds a new model of type specified by blueprint to a DataRobot project*

Description

This function requests the creation of a new model in the DataRobot modeling project defined by the project parameter. The function also allows the user to specify alternatives to the project default for featurelist, samplePct, and scoringType. This function returns an integer modelJobId value, which can be used by the GetModelFromJobId function to return the full model object.

Usage

```
RequestNewModel(project, blueprint, featurelist = NULL, samplePct = NULL,
  trainingRowCount = NULL, scoringType = NULL)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
blueprint	list. A list with at least the following two elements: blueprintId and projectId. Note that the individual elements of the list returned by ListBlueprints are admissible values for this parameter.
featurelist	list. A list that contains the element featurelistId that specifies the featurelist to be used in building the model; if not specified (i.e., for the default value NULL), the project default (Informative Features) is used.
samplePct	numeric. The percentage of the training dataset to be used in building the new model; if not specified (i.e., for the default value NULL), the maxTrainPct value for the project is used. Value should be between 0 and 100.
trainingRowCount	integer. The number of rows to use to train the requested model.
scoringType	character. String specifying the scoring type; default is validation set scoring, but cross-validation averaging is also possible.

Details

Motivation for this function is the fact that some models - e.g., very complex machine learning models fit to large datasets - may take a long time to complete. Splitting the model creation request from model retrieval in these cases allows the user to perform other interactive R session tasks between the time the model creation/update request is made and the time the final model is available.

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, 'samplePct' and 'trainingRowCount' are assumed to be in terms of rows of the minority class.

Note : For datetime partitioned projects, use `RequestNewDatetimeModel` instead

Value

An integer value that can be used as the modelJobId parameter in subsequent calls to the GetModelFromJobId function.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  blueprints <- ListBlueprints(projectId)
  blueprint <- blueprints[[1]]
  RequestNewModel(projectId, blueprint)

## End(Not run)
```

RequestNewRatingTableModel

Create a new model from a rating table.

Description

Create a new model from a rating table.

Usage

```
RequestNewRatingTableModel(project, ratingTableId)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
ratingTableId	character. The ID of the rating table.

Value

An integer value that can be used as the modelJobId parameter in subsequent calls to the GetModelFromJobId function.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  ratingTableId <- "5984b4d7100d2b31c1166529"
  RequestNewModel(projectId, ratingTableId)

## End(Not run)
```

RequestPredictionsForDataset

Request predictions against a previously uploaded dataset

Description

Request predictions against a previously uploaded dataset

Usage

```
RequestPredictionsForDataset(project, modelId, datasetId)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
modelId	numeric. The ID of the model to use to make predictions
datasetId	numeric. The ID of the dataset to make predictions against (as uploaded from UploadPredictionDataset)

Value

predictJobId to be used by GetPredictions function to retrieve the model predictions.

Examples

```
## Not run:
  dataset <- UploadPredictionDataset(project, diamonds_small)
  model <- ListModels(project)[[1]]
  modelId <- model$modelId
  predictJobId <- RequestPredictionsForDataset(project, modelId, dataset$id)
  predictions <- GetPredictions(project, predictJobId)

## End(Not run)
```

RequestPrimeModel *Request training for a DataRobot Prime model using a specified ruleset*

Description

Training a model using a ruleset is a necessary prerequisite for being able to download the code for a ruleset.

Usage

```
RequestPrimeModel(project, ruleset)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
ruleset	list. A list specifying rulest parameters (see GetRulesets)

Value

job Id

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  rulesets <- GetRulesets(projectId, modelId)
  ruleset <- rulesets[[1]]
  RequestPrimeModel(projectId, ruleset)

## End(Not run)
```

RequestReasonCodes *Request reason codes computation for a specified model and dataset.*

Description

In order to create ReasonCodes for a particular model and dataset, you must first: Compute feature impact for the model via `RequestFeatureImpact()` Compute a ReasonCodesInitialization for the model via `RequestReasonCodesInitialization()` Compute predictions for the model and dataset via `RequestPredictionsForDataset()` After reason codes are requested information about them can be accessed using the functions `GetReasonCodesMetadataFromJobId` and `GetReasonCodesMetadata` And reason codes themselves can be accessed using the functions `GetReasonCodesRows`, `GetAllReasonCodesRowsAsDataFrame`, `DownloadReasonCodes`

Usage

```
RequestReasonCodes(model, datasetId, maxCodes = NULL, thresholdLow = NULL,
  thresholdHigh = NULL)
```

Arguments

`model`	An S3 object of class dataRobotModel like that returned by the function GetModel, or each element of the list returned by the function ListModels.
`datasetId`	Character string. Id of the prediction dataset for which reason codes are requested
`maxCodes`	integer (optional) The maximum number of reason codes to supply per row of the dataset, default: 3.
`thresholdLow`	numeric (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.
`thresholdHigh`	numeric (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.

Details

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.

Value

job Id

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  datasets <- ListPredictionDatasets(projectId)
  dataset <- datasets[[1]]
  datasetId <- dataset$id
  model <- GetModel(model, datasetId)
  RequestReasonCodes(model, datasetId)

## End(Not run)
```

RequestReasonCodesInitialization

Request reason codes initialization for specified model

Description

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.

Usage

```
RequestReasonCodesInitialization(model)
```

Arguments

model	An S3 object of class dataRobotModel like that returned by the function GetModel, or each element of the list returned by the function ListModels.

Value

job Id

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  model <- GetModel(projectId, modelId)
  RequestReasonCodesInitialization(model)

## End(Not run)
```

`RequestSampleSizeUpdate` *Refits an existing model to a different fraction of the training dataset*

Description

This function requests a refit of the model defined by the model parameter to the same training dataset used in building it originally, but with a different fraction of the data, specified by the samplePct parameter. The function returns an integer value that may be used with the function GetModelFromJobId to retrieve the model after fitting is complete.

Usage

```
RequestSampleSizeUpdate(model, samplePct = NULL, trainingRowCount = NULL)
```

Arguments

`model`	An S3 object of class dataRobotModel like that returned by the function GetModel, or each element of the list returned by the function ListModels.
`samplePct`	Numeric, specifying the percentage of the training dataset to be used in building the new model.
`trainingRowCount`	integer. The number of rows to use to train the requested model.

Details

Motivation for this function is the fact that some models - e.g., very complex machine learning models fit to large datasets - may take a long time to complete. Splitting the model creation request from model retrieval in these cases allows the user to perform other interactive R session tasks between the time the model creation/update request is made and the time the final model is available.

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, 'samplePct' and 'trainingRowCount' are assumed to be in terms of rows of the minority class.

Value

Integer, value to be used as the modelJobId parameter in calling the function GetModelFromJobId to retrieve the updated model.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  model <- GetModel(projectId, modelId)
  RequestSampleSizeUpdate(model, samplePct = 100)

## End(Not run)
```

RequestTrainingPredictions

Request training predictions for a specific model.

Description

Request training predictions for a specific model.

Usage

```
RequestTrainingPredictions(model, dataSubset)
```

Arguments

`model`	An S3 object of class dataRobotModel like that returned by the function GetModel, or each element of the list returned by the function ListModels.
`dataSubset`	character. What data subset would you like to predict on? Possible options are included in `DataSubset`. Possible options are: • `DataSubset$All` will use all available data. • `DataSubset$ValidationAndHoldout` will use all data except the training set. • `DataSubset$Holdout` will use only holdout data.

Value

job Id

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  model <- GetModel(projectId, modelId)
  RequestTrainingPredictions(model, dataSubset = DataSubset$All)

## End(Not run)
```

`RequestTransferrableModel`

Request generation of an transferrable 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 DownloadTransferrableModel for that.

Description

Request generation of an transferrable 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 DownloadTransferrableModel for that.

Usage

```
RequestTransferrableModel(project, modelId)
```

Arguments

`project`	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
`modelId`	numeric. Unique alphanumeric identifier for the model of interest.

Value

jobId

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  jobId <- RequestTransferrrableModel(projectId, modelId)
  WaitForJobToComplete(projectId, jobId)
  file <- file.path(tempdir(), "model.drmodel")
  DownloadTransferrableModel(projectObject, modelId, file)
```

```
## End(Not run)
```

`ScaleoutModelingMode` *Scaleout modeling modes*

Description

This is a list that contains the valid values for the `scaleoutModelingMode` parameter found in `SetTarget`. If you wish, you can specify `scaleoutModelingMode` using the list values here, e.g. ScaleoutModelingMode$Autopilot instead of "Autopilot".

Usage

```
ScaleoutModelingMode
```

Format

An object of class `list` of length 3.

`ScoreBacktests` *Compute the scores for all available backtests.*

Description

Some backtests may be unavailable if the model is trained into their validation data.

Usage

```
ScoreBacktests(model)
```

Arguments

`model`	An S3 object of class dataRobotModel like that returned by the function GetModel, or each element of the list returned by the function ListModels.

Value

Integer a job tracking the backtest computation. When it is complete, all available backtests will have scores computed. Function WaitForJobToComplete can be used to wait for the job completion

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  model <- GetModel(projectId, modelId)
  ScoreBacktests(model)

## End(Not run)
```

SetTarget *Set the target variable (and by default, start the DataRobot Autopilot)*

Description

This function sets the target variable for the project defined by project, starting the process of building models to predict the response variable target. Both of these parameters - project and target - are required and they are sufficient to start a modeling project with DataRobot default specifications for the other 10 optional parameters.

Usage

```
SetTarget(project, target, metric = NULL, weights = NULL,
  partition = NULL, mode = NULL, seed = NULL, targetType = NULL,
  positiveClass = NULL, blueprintThreshold = NULL, responseCap = NULL,
  featurelistId = NULL, smartDownsampled = NULL,
  majorityDownsamplingRate = NULL, scaleoutModelingMode = NULL,
  accuracyOptimizedBlueprints = NULL, offset = NULL, exposure = NULL,
  eventsCount = NULL, maxWait = 600)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
target	character. String giving the name of the response variable to be predicted by all project models.
metric	character. Optional. String specifying the model fitting metric to be optimized; a list of valid options for this parameter, which depends on both project and target, may be obtained with the function GetValidMetrics.
weights	character. Optional. String specifying the name of the column from the modeling dataset to be used as weights in model fitting.
partition	partition. Optional. S3 object of class 'partition' whose elements specify a valid partitioning scheme. See help for functions CreateGroupPartition, CreateRandomPartition, CreateStratifiedPartition, CreateUserPartition and CreateDatetimePartitionSpecification.

`mode` character. Optional. Specifies the autopilot mode used to start the modeling project; valid options are "auto" (fully automatic, the current DataRobot default, obtained when mode = NULL), "manual" and "quick"

`seed` integer. Optional. Seed for the random number generator used in creating random partitions for model fitting.

`targetType` character. Optional. Used to specify the targetType to use for a project. Valid options are "Binary", "Multiclass", "Regression". Set to "Multiclass" to enable multiclass modeling. Otherwise, it can help to disambiguate, i.e. telling DataRobot how to handle a numeric target with a few unique values that could be used for either multiclass or regression. See `TargetType` for an easier way to keep track of the options.

`positiveClass` character. Optional. Target variable value corresponding to a positive response in binary classification problems.

`blueprintThreshold`
integer. Optional. The maximum time (in hours) that any modeling blueprint is allowed to run before being excluded from subsequent autopilot stages.

`responseCap` numeric. Optional. Floating point value, between 0.5 and 1.0, specifying a capping limit for the response variable. The default value NULL corresponds to an uncapped response, equivalent to responseCap = 1.0.

`featurelistId` numeric. Specifies which feature list to use. If NULL (default), a default featurelist is used.

`smartDownsampled`
logical. Optional. Whether to use smart downsampling to throw away excess rows of the majority class. Only applicable to classification and zero-boosted regression projects.

`majorityDownsamplingRate`
numeric. Optional. Floating point value, between 0.0 and 100.0. The percentage 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.

`scaleoutModelingMode`
character. Optional. Specifies the behavior of Scaleout models for the project. Possible options are in `ScaleoutModelingMode`.

- `ScaleoutModelingMode$Disabled` will prevent scaleout models from running during autopilot and will prevent Scaleout models from showing up in blueprints.
- `ScaleoutModelingMode$RepositoryOnly` will prevent scaleout models from running during autopilot, but will make them available in blueprints to run manually.
- `ScaleoutModelingMode$Autopilot` will run scaleout models during autopilot and will make them available in blueprints.

Note that scaleout models are only supported in the Hadoop environment with the correct corresponding user permission set.

`accuracyOptimizedBlueprints`
logical. Optional. When enabled, accuracy optimized blueprints will run in autopilot for the project. These are longer-running model blueprints that provide increased accuracy over normal blueprints that run during autopilot.

offset	character. Optional. Vector of the names of the columns containing the offset of each row.
exposure	character. Optional. The name of a column containing the exposure of each row.
eventsCount	character. Optional. The name of a column specifying the events count.
maxWait	integer. Specifies how many seconds to wait for the server to finish analyzing the target and begin the modeling process. If the process takes longer than this parameter specifies, execution will stop (but the server will continue to process the request).

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  SetTarget(projectId, "targetFeature")
  SetTarget(projectId, "targetFeature", metric = "LogLoss")
  SetTarget(projectId, "targetFeature", mode = AutopilotMode$Manual)
  SetTarget(projectId, "targetFeature", targetType = TargetType$Multiclass)

## End(Not run)
```

SetupProject *Function to set up a new DataRobot project*

Description

This function uploads a modeling dataset defined by the dataSource parameter and allows specification of the optional project name projectName. The dataSource parameter can be either the name of a CSV file or a dataframe; in the latter case, it is saved as a CSV file whose name is described in the Details section. This function returns the projectName specified in the calling sequence, the unique alphanumeric identifier projectId for the new project, the name of the modeling dataset uploaded to create this project, and the project creation time and date.

Usage

```
SetupProject(dataSource, projectName = NULL, maxWait = 60 * 60)
```

Arguments

dataSource	object. Either (a) the name of a CSV file, (b) a dataframe or (c) url to a publicly available file; in each case, this parameter identifies the source of the data from which all project models will be built. See Details.
projectName	character. Optional. String specifying a project name.
maxWait	integer. The maximum time to wait for each of two steps: (1) The initial project creation request, and (2) data processing that occurs after receiving the response to this initial request.

Details

The DataRobot modeling engine requires a CSV file containing the data to be used in fitting models, and this has been implemented here in two ways. The first and simpler is to specify dataSource as the name of this CSV file, but for the convenience of those who wish to work with dataframes, this function also provides the option of specifying a dataframe, which is then written to a CSV file and uploaded to the DataRobot server. In this case, the file name is either specified directly by the user through the saveFile parameter, or indirectly from the name of the dataSource dataframe if saveFile = NULL (the default). In this second case, the file name consists of the name of the dataSource dataframe with the string csvExtension appended.

Value

This function returns a list with the following four components:

projectName The name assigned to the DataRobot project

projectId The unique alphanumeric project identifier for this DataRobot project

fileName The name of the CSV modeling file uploaded for this project

created Character string containing the time and date of project creation

Examples

```
## Not run:
  SetupProject(iris, "dr-iris")

## End(Not run)
```

SetupProjectFromHDFS *Function to set up a new DataRobot project using datasource on a WebHDFS server*

Description

This function returns the projectName specified in the calling sequence, the unique alphanumeric identifier projectId for the new project, the name of the modeling dataset uploaded to create this project, and the project creation time and date.

Usage

```
SetupProjectFromHDFS(url, port = NULL, projectName = NULL, maxWait = 60 *
  60)
```

Arguments

url	character. The location of the WebHDFS file, both server and full path. Per the DataRobot specification, must begin with hdfs://
port	integer. Optional. The port to use. If not specified, will default to the server default (50070).
projectName	character. Optional. String specifying a project name.
maxWait	integer. The maximum time to wait for each of two steps: (1) The initial project creation request, and (2) data processing that occurs after receiving the response to this initial request.

Value

This function returns a list with the following four components:

projectName The name assigned to the DataRobot project

projectId The unique alphanumeric project identifier for this DataRobot project

fileName The name of the CSV modeling file uploaded for this project

created Character string containing the time and date of project creation

Examples

```
## Not run:
  SetupProjectFromHDFS(url = 'hdfs://path/to/data',
                       port = 12345,
                       projectName = 'dataProject')

## End(Not run)
```

SetupProjectFromMySQL *Function to set up a new DataRobot project using data from MySQL table*

Description

This function returns the projectName specified in the calling sequence, the unique alphanumeric identifier projectId for the new project, the name of the modeling dataset uploaded to create this project, and the project creation time and date.

Usage

```
SetupProjectFromMySQL(server, database, table, user, port = NULL,
  prefetch = NULL, projectName = NULL, password = NULL,
  encryptedPassword = NULL, maxWait = 60 * 60)
```

Arguments

server	character. The address of the MySQL server
database	character. The name of the database to use
table	character. The name of the table to fetch
user	character. The username to use to access the database
port	integer. Optional. The port to reach the MySQL server. If not specified, will use the default specified by DataRobot (3306).
prefetch	integer. Optional. If specified, specifies the number of rows to stream at a time from the database. If not specified, fetches all results at once. This is an optimization for reading from the database
projectName	character. Optional. String specifying a project name.
password	character. Optional. The plaintext password to be used to access MySQL database. Will be first encrypted with DataRobot. Only use this or encryptedPassword, not both.
encryptedPassword	character. Optional. The encrypted password to be used to access MySQL database. Only use this or password, not both.
maxWait	integer. The maximum time to wait for each of two steps: (1) The initial project creation request, and (2) data processing that occurs after receiving the response to this initial request.

Value

This function returns a list with the following four components:

projectName The name assigned to the DataRobot project

projectId The unique alphanumeric project identifier for this DataRobot project

fileName The name of the CSV modeling file uploaded for this project

created String containing the time and date of project creation

Examples

```
## Not run:
  SetupProjectFromMySQL(server = 'myServer',
                        database = 'myDatabase',
                        table = 'myTable',
                        user = 'sqlUser',
                        port = '12345')

## End(Not run)
```

SetupProjectFromOracle *Function to set up a new DataRobot project using data from Oracle table*

Description

This function returns the projectName specified in the calling sequence, the unique alphanumeric identifier projectId for the new project, the name of the modeling dataset uploaded to create this project, and the project creation time and date.

Usage

```
SetupProjectFromOracle(dbq, table, username, fetchBufferSize = NULL,
  projectName = NULL, password = NULL, encryptedPassword = NULL,
  maxWait = 60 * 60)
```

Arguments

dbq	character. tnsnames.ora entry in host:port/sid format
table	character. The name of the table to fetch.
username	character. The username to use to access the database
fetchBufferSize	integer. Optional. If specified, specifies the size of buffer that will be used to stream data from the database. Otherwise will use DataRobot default value.
projectName	character. Optional String specifying a project name.
password	character. Optional. The plaintext password to be used to access MySQL database. Will be first encrypted with DataRobot. Only use this or encryptedPassword, not both.
encryptedPassword	character. Optional. The encrypted password to be used to access MySQL database. Only use this or password, not both.
maxWait	integer. The maximum time to wait for each of two steps: (1) The initial project creation request, and (2) data processing that occurs after receiving the response to this initial request.

Value

This function returns a list with the following four components:

projectName The name assigned to the DataRobot project

projectId The unique alphanumeric project identifier for this DataRobot project

fileName The name of the CSV modeling file uploaded for this project

created Character string containing the time and date of project creation

Examples

```
## Not run:
  SetupProjectFromOracle(dbq = 'localhost:4001/sid',
                         table = 'myTable',
                         user = 'oracleUser')

## End(Not run)
```

`SetupProjectFromPostgreSQL` *Function to set up a new DataRobot project using data from PostgreSQL table*

Description

This function returns the projectName specified in the calling sequence, the unique alphanumeric identifier projectId for the new project, the name of the modeling dataset uploaded to create this project, and the project creation time and date.

Usage

```
SetupProjectFromPostgreSQL(server, database, table, username, port = NULL,
  driver = NULL, fetch = NULL, useDeclareFetch = NULL,
  projectName = NULL, password = NULL, encryptedPassword = NULL,
  maxWait = 60 * 60)
```

Arguments

`server`	character. The address of the MySQL server
`database`	character. The name of the database to use
`table`	character. The name of the table to fetch
`username`	character. The username to use to access the database
`port`	integer. Optional. The port to reach the PostgreSQL server. If not specified, will use the default specified by DataRobot (5432).
`driver`	character. Optional. Specify ODBC driver to use. If not specified - use DataRobot default. See the values within datarobot.enums.POSTGRESQL_DRIVER
`fetch`	integer. Optional. If specified, specifies the number of rows to stream at a time from the database. If not specified, fetches all results at once. This is an optimization for reading from the database
`useDeclareFetch`	logical. Optional. On TRUE, server will fetch result as available using DB cursor. On FALSE it will try to retrieve entire result set - not recommended for big tables. If not specified - use the default specified by DataRobot.
`projectName`	numeric. Optional. String specifying a project name.

`password` character. Optional. The plaintext password to be used to access MySQL database. Will be first encrypted with DataRobot. Only use this or `encryptedPassword`, not both.

`encryptedPassword` character. Optional. The encrypted password to be used to access

`maxWait` integer. The maximum time to wait for each of two steps: (1) The initial project creation request, and (2) data processing that occurs after receiving the response to this initial request.

Value

This function returns a list with the following four components:

projectName The name assigned to the DataRobot project

projectId The unique alphanumeric project identifier for this DataRobot project

fileName The name of the CSV modeling file uploaded for this project

created Character string containing the time and date of project creation

Examples

```
## Not run:
  SetupProjectFromPostgreSQL(server = 'myServer',
                             database = 'myDatabase',
                             table = 'myTable',
                             user = 'postgres',
                             port = '12345')

## End(Not run)
```

StartNewAutoPilot *Starts autopilot on provided featurelist. Only one autopilot can be running at the time. That's why any ongoing autopilot on different featurelist will be halted - modelling jobs in queue would not be affected but new jobs would not be added to queue by halted autopilot.*

Description

There is an error if autopilot is currently running on or has already finished running on the provided featurelist and also if project's target was not selected (via SetTarget).

Usage

```
StartNewAutoPilot(project, featurelistId, mode = AutopilotMode$FullAuto)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
featurelistId	numeric. Specifies which feature list to use.
mode	character. The desired autopilot mode. Currently only AutopilotMode$FullAuto is supported.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  featureList <- CreateFeaturelist(projectId, "myFeaturelist", c("feature1", "feature2"))
  featurelistId <- featureList$featurelistId
  StartNewAutoPilot(projectId, featurelistId)

## End(Not run)
```

StartRetryWaiter *Creates a waiter function that can be used in a loop while trying some task many times. The waiter sleeps while waiting to try again, with sleep times determined by exponential back-off.*

Description

Creates a waiter function that can be used in a loop while trying some task many times. The waiter sleeps while waiting to try again, with sleep times determined by exponential back-off.

Usage

```
StartRetryWaiter(timeout = NULL, delay = 0.1, maxdelay = 1)
```

Arguments

timeout	integer. How long (in seconds) to keep trying before timing out (NULL means no timeout)
delay	integer. Initial delay between tries (in seconds).
maxdelay	integer. Maximim delay (in seconds) between tries.

Value

function which gets the waiter status. This function returns a list with these items: /itemize /item index numeric. How many times we have waited. /item secondsWaited numeric. How long (in seconds) since we started the timer. /item stillTrying logical. Whether we should keep trying or give up (logical)

Stringify *Convert a function into a single string for DataRobot*

Description

Convert a function into a single string for DataRobot

Usage

```
Stringify(functionToConvert, dputFile = tempfile())
```

Arguments

functionToConvert	function. The function to convert to a string.
dputFile	character. Optional. A filepath to sink dput into.

summary.dataRobotModel *DataRobot S3 object methods for R's generic summary function*

Description

These functions extend R's generic summary function to the DataRobot S3 object classes dataRobotModel, dataRobotProject, listOfBlueprints, listOfFeaturelists, listOfModels, and projectSummaryList.

Usage

```
## S3 method for class 'dataRobotModel'
summary(object, ...)

## S3 method for class 'dataRobotProject'
summary(object, ...)

## S3 method for class 'listOfBlueprints'
summary(object, nList = 6, ...)

## S3 method for class 'listOfFeaturelists'
summary(object, nList = 6, ...)

## S3 method for class 'listOfModels'
summary(object, nList = 6, ...)

## S3 method for class 'projectSummaryList'
summary(object, nList = 6, ...)
```

Arguments

object	The S3 object to be summarized.
...	list. Not currently used.
nList	integer. For the 'listOf' class objects, the first nList elements of the list are summarized in the dataframe in the second element of the list returned by the function.

Value

An object-specific summary: for objects of class dataRobotModel and dataRobotProject, this summary is a character vector giving key characteristics of the model or project, respectively; for the other object classes, the value is a two-element list where the first element is a brief summary character string and the second element is a more detailed dataframe with nList elements.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  model <- GetModel(projectId, modelId)
  summary(model)

## End(Not run)
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  project <- GetProject(projectId)
  summary(project)

## End(Not run)
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  blueprints <- ListBlueprints(projectId)
  summary(blueprints)

## End(Not run)
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  featureList <- CreateFeaturelist(projectId, "myFeaturelist", c("feature1", "feature2"))
  summary(featureList)

## End(Not run)
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  models <- ListModels(projectId)
  summary(models)

## End(Not run)
## Not run:
  projectSummary <- GetProjectList()
  summary(projectSummary)
```

```
## End(Not run)
```

`TargetType` *Target Type modes*

Description

This is a list that contains the valid values for the Target Types

Usage

```
TargetType
```

Format

An object of class `list` of length 3.

`TreatAsExponential` *Treat as exponential*

Description

Treat as exponential

Usage

```
TreatAsExponential
```

Format

An object of class `list` of length 3.

UnpauseQueue *Re-start the DataRobot modeling queue*

Description

This function unpauses the modeling queue for a specified DataRobot project.

Usage

```
UnpauseQueue(project)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  UnpauseQueue(projectId)

## End(Not run)
```

UpdateProject *Update parameters for an existing project*

Description

This function updates parameters for the project defined by project.

Usage

```
UpdateProject(project, newProjectName = NULL, workerCount = NULL,
  holdoutUnlocked = NULL)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
newProjectName	Updated value for the projectName parameter associated with the project.
workerCount	Integer; sets the number of workers requested for the associated project.
holdoutUnlocked	Either NULL (the default) or logical TRUE; if TRUE, this function requests the DataRobot Autopilot to unlock the holdout data subset.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  UpdateProject(projectId, newProjectName = "cooler Project")
  UpdateProject(projectId, workerCount = 20)
  UpdateProject(projectId, holdoutUnlocked = TRUE)

## End(Not run)
```

`UpdateTransferrableModel` *Update the display name or note for an imported model.*

Description

Update the display name or note for an imported model.

Usage

```
UpdateTransferrableModel(importId, displayName = NULL, note = NULL)
```

Arguments

`importId`	character. Id of the import.
`displayName`	character. The new display name.
`note`	character. The new note.

Value

A list describing uploaded transferrable model with the following components:

- note. Character string Manually added node about this imported model.
- datasetName. Character string Filename of the dataset used to create the project the model belonged to.
- modelName. Character string Model type describing the model generated by DataRobot.
- displayName. Character string Manually specified human-readable name of the imported model.
- target. Character string The target of the project the model belonged to prior to export.
- projectName. Character string Name of the project the model belonged to prior to export.
- importedByUsername. Character string Username of the user who imported the model.
- importedAt. Character string The time the model was imported.
- version. Numeric Project version of the project the model belonged to.
- projectId. Character id of the project the model belonged to prior to export.
- featurelistName. Character string Name of the featurelist used to train the model.

- createdByUsername. Character string Username of the user who created the model prior to export.
- importedById. Character string id of the user who imported the model.
- id. Character string id of the import.
- createdById. Character string id of the user who created the model prior to export.
- modelId. Character string original id of the model prior to export.
- originUrl. Character string URL.

Examples

```
## Not run:
  id <- UploadTransferrableModel("model.drmodel")
  UpdateTransferrableModel(id, displayName = "NewName", note = "This is my note.")

## End(Not run)
```

`UploadData` *Upload a data source.*

Description

Takes either a file path and returns output for POST that specifies the file object via form upload. This function is meant to facilitate uploading CSV data sources into DataRobot, such as through `SetupProject`.

Usage

```
UploadData(filePath)
```

Arguments

`filePath`	character. The file to upload.

Value

An httr object specifying the form upload content of the file path.

See Also

SetupProject

`UploadPredictionDataset` *Function to upload new data to a DataRobot project for predictions*

Description

The DataRobot prediction engine requires a CSV file containing the data to be used in prediction, and this has been implemented here in two ways. The first and simpler is to specify dataSource as the name of this CSV file, but for the convenience of those who wish to work with dataframes, this function also provides the option of specifying a dataframe, which is then written to a CSV file and uploaded to the DataRobot server.

Usage

```
UploadPredictionDataset(project, dataSource, forecastPoint = NULL,
  maxWait = 600)
```

Arguments

`project`	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
`dataSource`	object. Either (a) the name of a CSV file (b) a dataframe or (c) url to publicly available file; in each case, this parameter identifies the source of the data for which predictions will be calculated.
`forecastPoint`	character. Optional. The point relative to which predictions will be generated, based on the forecast window of the project. Only specified in time series projects.
`maxWait`	integer. The maximum time (in seconds) to wait for each of two steps: (1) The initial dataset upload request, and (2) data processing that occurs after receiving the response to this initial request.

Value

list with the following components:

- id character. The unique alphanumeric identifier for the dataset.
- numColumns numeric. Number of columns in dataset.
- name character. Name of dataset file.
- created character. time of upload.
- projectId character. String giving the unique alphanumeric identifier for the project.
- numRows numeric. Number of rows in dataset.
- forecastPoint character. The point relative to which predictions will be generated, based on the forecast window of the project. Only specified in time series projects, otherwise will be NULL.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  UploadPredictionDataset(projectId, iris)

## End(Not run)
```

`UploadTransferrableModel`

Import a previously exported model for predictions.

Description

Import a previously exported model for predictions.

Usage

```
UploadTransferrableModel(modelFile, maxWait = 600)
```

Arguments

`modelFile`	character. Path to binary transeffable model file.
`maxWait`	integer. Specifies how many seconds to wait for upload to finish.

Value

A list describing uploaded transferrable model with the following components:

- note. Character string Manually added node about this imported model.
- datasetName. Character string Filename of the dataset used to create the project the model belonged to.
- modelName. Character string Model type describing the model generated by DataRobot.
- displayName. Character string Manually specified human-readable name of the imported model.
- target. Character string The target of the project the model belonged to prior to export.
- projectName. Character string Name of the project the model belonged to prior to export.
- importedByUsername. Character string Username of the user who imported the model.
- importedAt. Character string The time the model was imported.
- version. Numeric Project version of the project the model belonged to.
- projectId. Character id of the project the model belonged to prior to export.
- featurelistName. Character string Name of the featurelist used to train the model.
- createdByUsername. Character string Username of the user who created the model prior to export.

- importedById. Character string id of the user who imported the model.
- id. Character string id of the import.
- createdById. Character string id of the user who created the model prior to export.
- modelId. Character string original id of the model prior to export.
- originUrl. Character string URL.

Examples

```
## Not run:
  UploadTransferrableModel("model.drmodel")

## End(Not run)
```

`ValidateModel` *Validate that model belongs to class 'dataRobotModel' and includes projectId and modelId.*

Description

Validate that model belongs to class 'dataRobotModel' and includes projectId and modelId.

Usage

```
ValidateModel(model)
```

Arguments

`model`	An S3 object of class dataRobotModel like that returned by the function GetModel, or each element of the list returned by the function ListModels.

`ValidateParameterIn` *Ensure a parameter is valid*

Description

A valid parameter `paramValue` is either NULL or in the space of `paramPossibilites`.

Usage

```
ValidateParameterIn(paramValue, paramPossibilities, allowNULL = TRUE)
```

Arguments

paramValue	object. The parameter value to check.
paramPossibilities	vector. A vector of possible values for the parameter.
allowNULL	logical. Whether or not to allow NULL as a possibility.

Value

TRUE if paramValue is valid, otherwise it raises an error.

Examples

```
## Not run:
  ValidateParameterIn("all", DataSubset)

## End(Not run)
```

ValidateProject *Get a projectId from a project object.*

Description

Get a projectId from a project object.

Usage

```
ValidateProject(project)
```

Arguments

project	object. Either list with projectId element or projectId value

---

ViewWebModel *Retrieve a DataRobot web page that displays detailed model information*

Description

This function brings up a web page that displays detailed model information like that available from the standard DataRobot user interface (e.g., graphical representations of model structures).

Usage

```
ViewWebModel(model)
```

Arguments

model	An S3 object of class dataRobotModel like that returned by the function GetModel, or each element of the list returned by the function ListModels.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  model <- GetModel(projectId, modelId)
  ViewWebModel(model)

## End(Not run)
```

ViewWebProject *Retrieve a DataRobot web page that displays detailed project information*

Description

This function brings up a web page that displays detailed project information like that available from the standard DataRobot user interface.

Usage

```
ViewWebProject(project)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  ViewWebProject(projectId)

## End(Not run)
```

`WaitForAutopilot` *This function periodically checks whether Autopilot is finished and returns only after it is.*

Description

This function periodically checks whether Autopilot is finished and returns only after it is.

Usage

```
WaitForAutopilot(project, checkInterval = 20, timeout = NULL,
  verbosity = 1)
```

Arguments

`project`	character. The project for which you want to wait until autopilot is finished.
`checkInterval`	numeric. Optional. Maximum wait (in seconds) between checks that Autopilot is finished. Defaults to 20.
`timeout`	numeric. Optional. Time (in seconds) after which to give up (Default is no timeout). There is an error if Autopilot is not finished before timing out.
`verbosity`	numeric. Optional. 0 is silent, 1 or more displays information about progress. Default is 1.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  WaitForAutopilot(projectId)

## End(Not run)
```

`WaitForJobToComplete` *Wait for specified job to complete*

Description

Wait for specified job to complete

Usage

```
WaitForJobToComplete(project, jobId, maxWait = 600)
```

Arguments

project	character. Either (1) a character string giving the unique alphanumeric identifier for the project, or (2) a list containing the element projectId with this identifier.
jobId	integer identifier (returned for example by RequestPrimeModel)
maxWait	maximum time to wait (in seconds) for the job to complete

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  blueprints <- ListBlueprints(projectId)
  blueprint <- blueprints[[1]]
  jobId <- RequestNewModel(projectId, blueprint)
  WaitForJobToComplete(projectId, jobId)

## End(Not run)
```

Index