Package 'datarobot'

July 1, 2020

Title 'DataRobot' Predictive Modeling API

Version 2.17.1

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

Depends R (>= 3.1.1), methods, stats

Imports httr (>= 1.2.0), jsonlite (>= 1.0), yaml (>= 2.1.13)

License MIT + file LICENSE

LazyData true

Encoding UTF-8

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

Maintainer Chester Ismay <api-maintainer@datarobot.com>

Suggests knitr, testthat, lintr, stubthat, data.table, MASS, mlbench, beanplot, doBy, insuranceData, rmarkdown, ggplot2, modelwordcloud, withr, memoise, rex

VignetteBuilder knitr

RoxygenNote 7.1.1

NeedsCompilation no

Repository CRAN

Date/Publication 2020-07-01 12:00:19 UTC

R topics documented:

datarobot-package 7
AddEureqaSolution 7
ApplySchema 8
as.data.frame 8
AutopilotMode 10
BatchFeaturesTypeTransform 10
BlendMethods 11

BlueprintChartToGraphviz . . . 12
CheckUrl . . . 12
CleanServerData . . . 13
ConnectToDataRobot . . . 13
ConstructDurationString . . . 14
CreateBacktestSpecification . . . 15
CreateCalendar . . . 16
CreateComplianceDocumentation . . . 17
CreateDataSource . . . 17
CreateDataStore . . . 18
CreateDatetimePartitionSpecification . . . 19
CreateDeployment . . . 22
CreateDerivedFeatures . . . 24
CreateFeaturelist . . . 25
CreateGroupPartition . . . 26
CreateModelingFeaturelist . . . 27
CreatePrimeCode . . . 28
CreateRandomPartition . . . 29
CreateRatingTable . . . 30
CreateStratifiedPartition . . . 31
CreateUserPartition . . . 32
CrossValidateModel . . . 33
cvMethods . . . 34
DataPartition . . . 34
DataPathFromDataArg . . . 35
DataSubset . . . 35
DeleteCalendar . . . 36
DeleteComplianceDocTemplate . . . 36
DeleteDataSource . . . 37
DeleteDataStore . . . 37
DeleteDeployment . . . 38
DeleteFeaturelist . . . 38
DeleteJob . . . 39
DeleteModel . . . 39
DeleteModelingFeaturelist . . . 40
DeleteModelJob . . . 41
DeletePredictionDataset . . . 41
DeletePredictionExplanations . . . 42
DeletePredictionExplanationsInitialization . . . 43
DeletePredictJob . . . 43
DeleteProject . . . 44
DeleteTransferableModel . . . 45
DifferencingMethod . . . 45
DownloadComplianceDocTemplate . . . 46
DownloadComplianceDocumentation . . . 47
DownloadPredictionExplanations . . . 48
DownloadPrimeCode . . . 49
DownloadRatingTable . . . 50

DownloadScoringCode . . . 50
DownloadSeriesAccuracy . . . 51
DownloadTimeSeriesFeatureDerivationLog . . . 52
DownloadTrainingPredictions . . . 53
DownloadTransferableModel . . . 54
ExpectHasKeys . . . 54
FeatureFromAsyncUrl . . . 55
GenerateDatetimePartition . . . 55
GetBlenderModel . . . 58
GetBlenderModelFromJobId . . . 59
GetBlueprint . . . 61
GetBlueprintChart . . . 62
GetBlueprintDocumentation . . . 62
GetCalendar . . . 63
GetCalendarFromProject . . . 64
GetComplianceDocTemplate . . . 64
GetConfusionChart . . . 65
GetCrossValidationScores . . . 67
GetDataSource . . . 67
GetDataStore . . . 68
GetDataStoreSchemas . . . 69
GetDataStoreTables . . . 70
GetDatetimeModel . . . 70
GetDatetimeModelFromJobId . . . 73
GetDatetimePartition . . . 74
GetDeployment . . . 76
GetDeploymentDriftTrackingSettings . . . 77
GetDriver . . . 78
GetFeatureAssociationMatrix . . . 79
GetFeatureAssociationMatrixDetails . . . 80
GetFeatureHistogram . . . 81
GetFeatureImpact . . . 82
GetFeatureImpactForJobId . . . 82
GetFeatureImpactForModel . . . 83
GetFeatureInfo . . . 84
GetFeaturelist . . . 86
GetFrozenModel . . . 87
GetFrozenModelFromJobId . . . 89
GetGeneralizedInsight . . . 90
GetJob . . . 90
GetLiftChart . . . 91
GetMissingValuesReport . . . 92
GetModel . . . 93
GetModelBlueprintChart . . . 95
GetModelBlueprintDocumentation . . . 95
GetModelCapabilities . . . 96
GetModelFromJobId . . . 97
GetModelingFeaturelist . . . 98

GetModelJob . . . 99
GetModelParameters . . . 100
GetModelRecommendation . . . 101
GetMultiSeriesProperties . . . 102
GetParetoFront . . . 103
GetPredictionDataset . . . 104
GetPredictionExplanations . . . 105
GetPredictionExplanationsInitialization . . . 107
GetPredictionExplanationsInitializationFromJobId . . . 108
GetPredictionExplanationsMetadata . . . 109
GetPredictionExplanationsMetadataFromJobId . . . 110
GetPredictionExplanationsRows . . . 111
GetPredictionExplanationsRowsAsDataFrame . . . 113
GetPredictions . . . 114
GetPredictJob . . . 116
GetPredictJobs . . . 117
GetPrimeEligibility . . . 118
GetPrimeFile . . . 118
GetPrimeFileFromJobId . . . 119
GetPrimeModel . . . 120
GetPrimeModelFromJobId . . . 121
GetProject . . . 122
GetProjectStatus . . . 123
GetRatingTable . . . 124
GetRatingTableFromJobId . . . 124
GetRatingTableModel . . . 125
GetRatingTableModelFromJobId . . . 126
GetRecommendedModel . . . 127
GetRocCurve . . . 127
GetRulesets . . . 128
GetSeriesAccuracy . . . 129
GetSeriesAccuracyForModel . . . 130
GetServerDataInRows . . . 131
GetTimeSeriesFeatureDerivationLog . . . 132
GetTrainingPredictionDataFrame . . . 133
GetTrainingPredictions . . . 133
GetTrainingPredictionsForModel . . . 134
GetTrainingPredictionsFromJobId . . . 135
GetTransferableModel . . . 135
GetTuningParameters . . . 137
GetValidMetrics . . . 138
GetWordCloud . . . 138
IsBlenderEligible . . . 139
IsId . . . 140
IsParameterIn . . . 140
JobStatus . . . 141
JobType . . . 142
ListBlueprints . . . 142

ListCalendars . . . 143
ListComplianceDocTemplates . . . 143
ListConfusionCharts . . . 144
ListDataSources . . . 145
ListDataStores . . . 145
ListDeployments . . . 146
ListDrivers . . . 147
ListFeatureInfo . . . 147
ListFeaturelists . . . 148
ListJobs . . . 149
ListLiftCharts . . . 150
ListModelFeatures . . . 151
ListModelingFeaturelists . . . 152
ListModelJobs . . . 153
ListModelRecommendations . . . 154
ListModels . . . 155
ListPredictionDatasets . . . 156
ListPredictionExplanationsMetadata . . . 157
ListPredictions . . . 158
ListPredictionServers . . . 159
ListPrimeFiles . . . 159
ListPrimeModels . . . 160
ListProjects . . . 161
ListRatingTableModels . . . 162
ListRatingTables . . . 163
ListRocCurves . . . 163
ListSharingAccess . . . 164
ListStarredModels . . . 165
ListTrainingPredictions . . . 166
ListTransferableModels . . . 166
MakeDataRobotRequest . . . 168
ModelReplacementReason . . . 169
PauseQueue . . . 169
PeriodicityMaxTimeStep . . . 170
PeriodicityTimeUnits . . . 170
plot.listOfModels . . . 170
PostgreSQLdrivers . . . 172
Predict . . . 173
predict.dataRobotModel . . . 174
PredictionDatasetFromAsyncUrl . . . 175
PrimeLanguage . . . 176
ProjectFromAsyncUrl . . . 176
ProjectStage . . . 176
RecommendedModelType . . . 177
ReformatMetrics . . . 177
RenameRatingTable . . . 178
ReplaceDeployedModel . . . 178
RequestApproximation . . . 179

RequestBlender . . . 180
RequestCrossSeriesDetection . . . 181
RequestFeatureImpact . . . 182
RequestFrozenDatetimeModel . . . 183
RequestFrozenModel . . . 184
RequestMultiSeriesDetection . . . 185
RequestNewDatetimeModel . . . 187
RequestNewModel . . . 188
RequestNewRatingTableModel . . . 190
RequestPredictionExplanations . . . 191
RequestPredictionExplanationsInitialization . . . 192
RequestPredictions . . . 193
RequestPrimeModel . . . 194
RequestSampleSizeUpdate . . . 195
RequestSeriesAccuracy . . . 196
RequestTrainingPredictions . . . 197
RequestTransferableModel . . . 198
RunInteractiveTuning . . . 199
ScaleoutModelingMode . . . 200
ScoreBacktests . . . 200
SeriesAggregationType . . . 201
SetPredictionThreshold . . . 201
SetTarget . . . 202
SetupProject . . . 205
SetupProjectFromDataSource . . . 206
SetupProjectFromHDFS . . . 207
Share . . . 208
SharingRole . . . 209
StarModel . . . 209
StartNewAutoPilot . . . 210
StartProject . . . 211
StartRetryWaiter . . . 214
StartTuningSession . . . 215
Stringify . . . 216
summary.dataRobotModel . . . 216
summary.listOfDataRobotTuningParameters . . . 218
TargetLeakageType . . . 219
TargetType . . . 219
TestDataStore . . . 219
TimeUnits . . . 220
ToggleStarForModel . . . 220
TreatAsExponential . . . 221
TryingToSubmitNull . . . 221
UnpauseQueue . . . 222
UnstarModel . . . 222
UpdateAccess . . . 223
UpdateCalendar . . . 224
UpdateComplianceDocTemplate . . . 224

UpdateDataSource . . . 225
UpdateDataStore . . . 226
UpdateDeploymentDriftTrackingSettings . . . 227
UpdateFeaturelist . . . 228
UpdateModelingFeaturelist . . . 228
UpdateProject . . . 229
UpdateTransferableModel . . . 229
UploadComplianceDocTemplate . . . 231
UploadData . . . 232
UploadPredictionDataset . . . 232
UploadPredictionDatasetFromDataSource . . . 234
UploadTransferableModel . . . 235
ValidateCalendar . . . 236
ValidateModel . . . 236
ValidateMultiSeriesProperties . . . 237
ValidateParameterIn . . . 237
ValidatePartition . . . 238
ValidateProject . . . 238
ValidateReplaceDeployedModel . . . 239
VariableTransformTypes . . . 239
ViewWebModel . . . 240
ViewWebProject . . . 240
WaitForAutopilot . . . 241
WaitForJobToComplete . . . 242

Index **243**

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

Description

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

`AddEureqaSolution` *Add a Eureqa solution to the list of models for the project.*

Description

Each Eureqa model contains multiple possible solutions (see `GetParetoFront`). However, only the best model is included in the leaderboard by default. To include other models, you can get them via `GetParetoFront` and then add them.

Usage

```
AddEureqaSolution(project, eureqaSolutionId)
```

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.
eureqaSolutionId	character. The solution ID of the Eureqa model to add.

Examples

```
## Not run:
  projectId <- "5b2827556523cd05bd1507a5"
  modelId <- "5b29406c6523cd0665685a8d"
  eureqaModel <- GetModel(projectId, modelId)
  paretoFront <- GetParetoFront(eureqModel)

## End(Not run)
```

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.

If simple = TRUE (the default), this method returns a dataframe with one row for each model and the following columns: projectName, projectId, created, fileName, target, targetType, positiveClass, metric, autopilotMode, stage, maxTrainPct, and holdoutUnlocked. If simple = FALSE, a dataframe is constructed from all elements of 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.

If simple = TRUE (the default), this method returns a dataframe with one row for each model and the following columns: modelType, expandedModel (constructed from modelType and processes from the listOfModels elements), modelId, blueprintId, featurelistName, featurelistId, samplePct, and the metrics validation value for projectMetric. If simple = FALSE, the method returns a complete dataframe with one row for each model and columns constructed from all fields in the original listOfModels object

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 options.

Usage

```
AutopilotMode
```

Format

An object of class `list` of length 3.

Details

`FullAuto` represents running the entire autopilot. `Quick` runs a quicker, abridged version of the autopilot that focuses on the most important models. `Manual` does not run the autopilot and instead leaves it to the user to select the algorithms to be run.

BatchFeaturesTypeTransform *Create new features by transforming the type of an existing ones.*

Description

Supports feature transformations, including:

- text to categorical
- text to numeric
- categorical to text
- categorical to numeric
- numeric to categorical

Usage

```
BatchFeaturesTypeTransform(
  project,
  parentNames,
  variableType,
  prefix = NULL,
  suffix = 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.
parentNames	character. Character vector of variable names to be transformed.
variableType	character. The new type that the columns should be converted to. See VariableTransformTypes.
prefix	character. Optional. The string to preface all the transformed features. Either prefix or suffix or both must be provided.
suffix	character. Optional. The string that will be appended at the end to all the transformed features. Either prefix or suffix or both must be provided.
maxWait	integer. Optional. The maximum amount of time (in seconds) to wait for DataRobot to finish processing the new column before providing a timeout error.

Value

a list of all the features, after transformation. See GetFeaturelist for details.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  BatchFeaturesTypeTransform(projectId,
                             parentNames = c("var1", "var2"),
                             variableType = VariableTransformTypes$Categorical,
                             suffix = "_transformed")

## End(Not run)
```

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 11.

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)
```

CheckUrl *Make sure the path is a reasonable URL*

Description

Make sure the path is a reasonable URL

Usage

```
CheckUrl(url)
```

Arguments

url	character. The URL to check.

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. To use DataRobot, you must connect to your account. This can be done in three ways:

- by passing an `endpoint` and `token` directly to `ConnectToDataRobot`
- by having a YAML config file in $HOME/.config/datarobot/drconfig.yaml
- by setting DATAROBOT_API_ENDPOINT and DATAROBOT_API_TOKEN environment variables

The three methods of authentication are given priority in that order (explicitly passing parameters to the function will trump a YAML config file, which will trump the environment variables.) If you have a YAML config file or environment variables set, you will not need to pass any parameters to `ConnectToDataRobot` in order to connect.

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).

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)
```

CreateCalendar *Create a calendar from an uploaded CSV.*

Description

Create a calendar from an uploaded CSV.

Usage

```
CreateCalendar(file, name = NULL, maxWait = 600)
```

Arguments

file	character. The filename containing the calendar CSV to upload.
name	character. Optional. The name of the calendar.
maxWait	integer. The maximum time (in seconds) to wait for the retrieve to complete.

Value

An S3 object of class "dataRobotCalendar"

Examples

```
## Not run:
   CreateCalendar("myRatingTable.csv", name = "myCalendar")

## End(Not run)
```

CreateComplianceDocumentation *Create compliance documentation from a model.*

Description

Note that if you're looking to download compliance documentation to a DOCX file, you can call `DownloadComplianceDocumentation` directly without using this function.

Usage

```
CreateComplianceDocumentation(model, templateId = 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.
`templateId`	character. Optional. The ID of the template to use in generating custom model documentation.

Value

An integer value that can be used as the jobId parameter in a subsequent call to `WaitForJobToComplete`.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  model <- GetModel(projectId, modelId)
  jobId <- CreateComplianceDocumentation(model) # optional step
  WaitForJobToComplete(projectId, jobId)        # optional step
  DownloadComplianceDocumentation(model)

## End(Not run)
```

CreateDataSource *Create a data source.*

Description

Create a data source.

Usage

```
CreateDataSource(
  type,
  canonicalName,
  dataStoreId,
  query = NULL,
  table = NULL,
  schema = NULL,
  partitionColumn = NULL,
  fetchSize = NULL
)
```

Arguments

type	character. The type of data source.
canonicalName	character. The user-friendly name of the data source.
dataStoreId	character. The ID of the data store to connect to.
query	character. A query to execute on the data store to get the data. Optional.
table	character. The specified database table. Optional.
schema	character. The specified database schema. Optional.
partitionColumn	character. The name of the partition column. Optional.
fetchSize	integer. a user specified fetch size in the range [1, 20000]. Optional. By default a fetchSize will be assigned to balance throughput and memory usage

Examples

```
## Not run:
 dataStoreId <- "5c1303269300d900016b41a7"
 CreateDataSource(type = "jdbc",
                  canonicalName = "Airline stats after 1995",
                  dataStoreId = dataStoreId,
                  query = 'SELECT * FROM airlines10mb WHERE "Year" >= 1995;')

## End(Not run)
```

CreateDataStore *Create a data store.*

Description

Create a data store.

Usage

```
CreateDataStore(type, canonicalName, driverId, jdbcUrl)
```

Arguments

type	character. The type of data store.
canonicalName	character. The user-friendly name of the data store.
driverId	character. The ID of the driver to use.
jdbcUrl	character. The full JDBC url.

Examples

```
## Not run:
 CreateDataStore(type = "jdbc",
                 canonicalName = "Demo DB",
                 driverId = "57a7c978c808916f4a630f89",
                 jdbcUrl = "jdbc:postgresql://my.db.address.org:5432/my_db")

## End(Not run)
```

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,
  defaultToKnownInAdvance = FALSE,
  featureDerivationWindowStart = NULL,
  featureDerivationWindowEnd = NULL,
  featureSettings = NULL,
  treatAsExponential = NULL,
  differencingMethod = NULL,
  windowsBasisUnit = NULL,
  periodicities = NULL,
  forecastWindowStart = NULL,
```

```
  forecastWindowEnd = NULL,
  multiseriesIdColumns = NULL,
  useCrossSeries = NULL,
  aggregationType = NULL,
  crossSeriesGroupByColumns = NULL,
  calendar = 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 holdoutStartDate 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 behavior is to create an OTV project.

`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. For each feature you would like to set feature settings for, pass the following in a list:

  - featureName character. The name of the feature to set feature settings.
  - knownInAdvance logical. Optional. Whether or not the feature is known in advance. Used for time series only. Defaults to FALSE.
  - doNotDerive logical. Optional. If TRUE, no time series derived features (e.g., lags) will be automatically engineered from this feature. Used for time series only. Defaults to FALSE.

treatAsExponential
: character. 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
: character. Optional. Defaults to "auto". Used to specify differencing method to apply if data is stationary. Use values from DifferencingMethod.

windowsBasisUnit
: character. Optional. Indicates which unit is the basis for the feature derivation window and forecast window. Valid options are a time unit (see TimeUnit) or "ROW".

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‘. Should be "ROW" if windowsBasisUnit is "ROW".

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

multiseriesIdColumns
: list. A list of the names of multiseries id columns to define series

useCrossSeries
: logical. If TRUE, cross series features will be included. For details, see "Calculating features across series" in the Time Series section of the DataRobot user guide.

aggregationType
: character. Optional. The aggregation type to apply when creating cross series features. Must be either "total" or "average". See SeriesAggregationType.

crossSeriesGroupByColumns
: character. Optional. Column to split a cross series into further groups. For example, if every series is sales of an individual product, the cross series group

could be e product category with values like "men's clothing", "sports equipment", etc. Requires multiseries with `useCrossSeries` enabled.

`calendar` character. Optional. Either the calendar object or calendar id to use for this project.

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",
                                            "defaultToKnownInAdvance" = 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")))
```

`CreateDeployment` *Create a deployment.*

Description

Create a deployment.

Usage

```
CreateDeployment(
  model,
  label = "",
```

```
  description = "",
  defaultPredictionServerId = 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.
label	character. The name of the deployment.
description	character. Optional. A longer description of the deployment.
defaultPredictionServerId	character. The ID of the prediction server to connect to. Can also be a prediction server object.

Value

A DataRobotDeployment object containing:

- id character. The ID of the deployment.
- label character. The label of the deployment.
- description character. The description of the deployment.
- defaultPredictionServer list. Information on the default prediction server connected with the deployment. See `ListPredictionServers` for details.
- model dataRobotModel. The model associated with the deployment. See `GetModel` for details.
- capabilities list. Information on the capabilities of the deployment.
- predictionUsage list. Information on the prediction usage of the deployment.
- serviceHealth list. Information on the service health of the deployment.
- modelHealth list. Information on the model health of the deployment.
- accuracyHealth list. Information on the accuracy health of the deployment.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  model <- GetModel(projectId, modelId)
  predictionServer <- ListPredictionServers()[[1]]
  CreateDeployment(model,
                   label = "myDeployment",
                   description = "this is my deployment",
                   defaultPredictionServerId = predictionServer)

## End(Not run)
```

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)
```

`CrossValidateModel` *Run cross validation on a model.*

Description

Note that this runs cross validation on a model as-is. If you would like to run cross-validation on a model with new parameters, use `RequestNewModel` instead.

Usage

```
CrossValidateModel(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.

Details

Note that this is not implemented for prime models or datetime models.

Value

Job ID of the cross validation job.

Examples

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

## End(Not run)
```

`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 4.

Details

For All, all available data is used.

For ValidationAndHoldout, only data outside the training set is used.

For Holdout, only holdout data is used.

For AllBacktests, data is used from all backtest validation folds. This requires the model to have successfully scored all backtests. Backtests are available on datetime partitioned projects only.

DeleteCalendar *Delete a calendar*

Description

Delete a calendar

Usage

```
DeleteCalendar(calendarId)
```

Arguments

calendarId	character. The ID of the calendar to retrieve.

Examples

```
## Not run:
   calendarId <- "5da75da31fb4a45b8a815a53"
   DeleteCalendar(calendarId)

## End(Not run)
```

DeleteComplianceDocTemplate *Deletes a compliance doc template.*

Description

Note that default templates cannot be deleted.

Usage

```
DeleteComplianceDocTemplate(templateId)
```

Arguments

templateId	character. The ID of the template to update.

Value

Nothing returned, but deletes the compliance doc template.

Examples

```
## Not run:
  templateId <- "5cf85080d9436e5c310c796d"
  DeleteComplianceDocTemplate(templateId)

## End(Not run)
```

`DeleteDataSource` *Delete a data store.*

Description

Delete a data store.

Usage

```
DeleteDataSource(dataSourceId)
```

Arguments

`dataSourceId`	character. The ID of the data store to update.

Examples

```
## Not run:
 dataSourceId <- "5c1303269300d900016b41a7"
 DeleteDataSource(dataSourceId)

## End(Not run)
```

`DeleteDataStore` *Delete a data store.*

Description

Delete a data store.

Usage

```
DeleteDataStore(dataStoreId)
```

Arguments

`dataStoreId`	character. The ID of the data store to update.

Examples

```
## Not run:
 dataStoreId <- "5c1303269300d900016b41a7"
 DeleteDataStore(dataStoreId)

## End(Not run)
```

`DeleteDeployment` *Delete a deployment.*

Description

Delete a deployment.

Usage

```
DeleteDeployment(deploymentId)
```

Arguments

`deploymentId`	character. The ID of the deployment.

Examples

```
## Not run:
  deploymentId <- "5e319d2e422fbd6b58a5edad"
  DeleteDeployment(deploymentId)

## End(Not run)
```

`DeleteFeaturelist` *Delete a featurelist*

Description

Delete a featurelist

Usage

```
DeleteFeaturelist(featurelist)
```

Arguments

`featurelist`	list. The featurelist to delete.

Examples

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

## End(Not run)
```

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 <- ListModelJobs(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)
```

DeleteModelingFeaturelist

Delete a modeling featurelist

Description

Delete a modeling featurelist

Usage

```
DeleteModelingFeaturelist(featurelist)
```

Arguments

featurelist	list. The modeling featurelist to delete.

Examples

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

## 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 ListModelJobs.

Examples

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

`DeletePredictionExplanations` *Function to delete prediction explanations*

Description

This function deletes prediction explanations specified by project and predictionExplanationId.

Usage

```
DeletePredictionExplanations(project, predictionExplanationId)
```

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.
`predictionExplanationId`	character. Id of the prediction explanations.

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$id
  model <- GetModel(projectId, modelId)
  jobId <- RequestPredictionExplanations(model, datasetId)
 predictionExplanationId <- GetPredictionExplanationsMetadataFromJobId(projectId, jobId)$id
  DeletePredictionExplanations(projectId, predictionExplanationId)

## End(Not run)
```

DeletePredictionExplanationsInitialization *Delete the prediction explanations initialization for a model.*

Description

Delete the prediction explanations initialization for a model.

Usage

```
DeletePredictionExplanationsInitialization(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

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"
  model <- GetModel(projectId, modelId)
  DeletePredictionExplanationsInitialization(model)

## 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. The integer ID predictionJobId that is created by the call to RequestPredictions.

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 ListProjects; (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)
```

DeleteTransferableModel *Delete this imported model.*

Description

Delete this imported model.

Usage

```
DeleteTransferableModel(importId)
```

Arguments

importId	character. Id of the import.

Examples

```
## Not run:
  id <- UploadTransferableModel("model.drmodel")
  DeleteTransferableModel(id)

## End(Not run)
```

DifferencingMethod *Differencing method*

Description

Differencing method

Usage

```
DifferencingMethod
```

Format

An object of class `list` of length 4.

DownloadComplianceDocTemplate *Download a compliance doc template (in JSON format).*

Description

Download a compliance doc template (in JSON format).

Usage

```
DownloadComplianceDocTemplate(
  filename = "template.json",
  templateId = NULL,
  type = NULL
)
```

Arguments

filename	character. Filename of file to save the compliance doc template to.
templateId	character. Optional. The ID of the template to use in generating custom model documentation.
type	character. Optional. The type of compliance doc to get. Can be "normal" to retrieve the default template or "timeSeries" to get the default time series template.

Value

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

Examples

```
## Not run:
  DownloadComplianceDocTemplate("template.json")  # download the default template
  # download the default template
  DownloadComplianceDocTemplate("template.json", type = "normal")
  # download the default time series template
  DownloadComplianceDocTemplate("template.json" type = "timeSeries")
  templateId <- "5cf85080d9436e5c310c796d"
 DownloadComplianceDocTemplate(templateId) # Download a custom template for a specific ID.

## End(Not run)
```

DownloadComplianceDocumentation
Download compliance documentation (in DOCX format).

Description

This function will create the compliance documentation first if it has not already been created. To create compliance documentation without downloading it, use `CreateComplianceDocumentation`. You can then skip the create step in this function by using 'create = FALSE'.

Usage

```
DownloadComplianceDocumentation(
  model,
  filename,
  templateId = NULL,
  create = TRUE,
  maxWait = 600
)
```

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.
`filename`	character. Filename of file to save the compliance documentation to.
`templateId`	character. Optional. The ID of the template to use in generating custom model documentation.
`create`	logical. Should we create the compliance documentation prior to downloading?
`maxWait`	integer. How long to wait (in seconds) for compliance documentation creation before raising a timeout error? Default 600.

Value

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

Examples

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

## End(Not run)
```

DownloadPredictionExplanations *Function to download and save prediction explanations rows as csv file*

Description

Function to download and save prediction explanations rows as csv file

Usage

```
DownloadPredictionExplanations(
  project,
  predictionExplanationId,
  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.
predictionExplanationId	character. Id of the prediction explanations.
filename	character. Filename of file to save prediction explanations 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(projectId, modelId)
```

```
  jobId <- RequestPredictionExplanations(model, datasetId)
 predictionExplanationId <- GetPredictionExplanationsMetadataFromJobId(projectId, jobId)$id
  file <- file.path(tempdir(), "testPredictionExplanation.csv")
  DownloadPredictionExplanations(projectId, predictionExplanationId, file)

## End(Not run)
```

`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 acquired 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)
```

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. Unique alphanumeric identifier for the model of interest.
fileName	character. 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)
```

DownloadSeriesAccuracy

Download the series accuracy for a model, computing it if not already computed.

Description

Download the series accuracy for a model, computing it if not already computed.

Usage

```
DownloadSeriesAccuracy(model, filename, encoding = "UTF-8")
```

Arguments

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

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

Examples

```
## Not run:
  projectId <- "5984b4d7100d2b31c1166529"
  modelId <- "5984b4d7100d2b31c1166529"
  model <- GetModel(projectId, modelId)
  DownloadSeriesAccuracy(model, "seriesAccuracy.csv")

## End(Not run)
```

DownloadTimeSeriesFeatureDerivationLog

Download the time series feature derivation log as a text file.

Description

Download the time series feature derivation log as a text file.

Usage

```
DownloadTimeSeriesFeatureDerivationLog(project, file)
```

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.
`file`	character. The name or path of the file to download to.

Value

Nothing, but writes the output to the desired file.

See Also

GetTimeSeriesFeatureDerivationLog

Examples

```
## Not run:
 projectId <- "5984b4d7100d2b31c1166529"
 DownloadTimeSeriesFeatureDerivationLog(projectId, "featureLog.txt")

## 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)
```

DownloadTransferableModel *Download an transferable 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

```
DownloadTransferableModel(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 transferable model

Examples

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

- cvMethod. The type of validation scheme used for the project.
- projectId character. The id of the project this partitioning applies to.
- datetimePartitionColumn character. The name of the column whose values as dates are used to assign a row to a particular partition.
- dateFormat character. The format (e.g. " partition column was interpreted (compatible with strftime [https://docs.python.org/2/library/time.html#time.strftime]).
- autopilotDataSelectionMethod character. Whether models created by the autopilot use "rowCount" or "duration" as their dataSelectionMethod.
- validationDuration character. 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. The start date of the available training data for scoring the holdout.
- availableTrainingDuration character. 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. The end date of the available training data for scoring the holdout.
- primaryTrainingStartDate character. The start date of primary training data for scoring the holdout.
- primaryTrainingDuration character. 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. The end date of the primary training data for scoring the holdout.
- gapStartDate character. The start date of the gap between training and holdout scoring data.
- gapDuration character. 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. The end date of the gap between training and holdout scoring data.
- holdoutStartDate character. The start date of holdout scoring data.
- holdoutDuration character. 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. The end date of the holdout scoring data.
- numberOfBacktests integer. the number of backtests used.

- backtests data.frame. A data frame of partition backtest. Each element represent one backtest and has the 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 known in advance. Knon in advance 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 forecast point the forecast 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 forecast point the forecast window should end. Only used for time series projects. Expressed in terms of the `timeUnit` of the `datetimePartitionColumn`.
- featureSettings list. A list of lists specifying settings for each feature. For each feature you would like to set feature settings for, pass the following in a list:
  - featureName character. The name of the feature to set feature settings.
  - knownInAdvance logical. Optional. Whether or not the feature is known in advance. Used for time series only. Defaults to `FALSE`.
  - doNotDerive logical. Optional. If `TRUE`, no time series derived features (e.g., lags) will be automatically engineered from this feature. Used for time series only. Defaults to `FALSE`.
- treatAsExponential character. Specifies whether to treat data as exponential trend and apply transformations like log-transform. Uses values from from `TreatAsExponential`.
- differencingMethod character. Used to specify differencing method to apply if data is stationary. Use values from `DifferencingMethod`.
- windowsBasisUnit character. Indicates which unit is the basis for the feature derivation window and forecast window. Uses values from `TimeUnit` and the value "ROW".
- periodicities list. A list of periodicities for different times, specified as a list of lists, where each list item specifies the ‘timeSteps‘ for a particular ‘timeUnit‘. Will be
- 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`.
- validationRowCount integer. The number of rows in the validation set.
- multiseriesIdColumns list. A list of the names of multiseries id columns to define series.

- numberOfKnownInAdvanceFeatures integer. The number of known in advance features.
- useCrossSeriesFeatures logical. Whether or not cross series features are included.
- aggregationType character. The aggregation type to apply when creating cross series features. See `SeriesAggregationType`.
- calendarId character. The ID of the calendar used for this project, if any.

Examples

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

## 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 external 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)
```

GetCalendar *Retrieve a calendar*

Description

Retrieve a calendar

Usage

```
GetCalendar(calendarId)
```

Arguments

calendarId	character. The ID of the calendar to retrieve.

Value

An S3 object of class "dataRobotCalendar"

Examples

```
## Not run:
   calendarId <- "5da75da31fb4a45b8a815a53"
   GetCalendar(calendarId)

## End(Not run)
```

`GetCalendarFromProject` *Retrieve the calendar for a particular project.*

Description

Retrieve the calendar for a particular project.

Usage

```
GetCalendarFromProject(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 "dataRobotCalendar"

Examples

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

## End(Not run)
```

`GetComplianceDocTemplate` *Get a compliance doc template.*

Description

A custom compliance doc template can be retrieved using `templateId`. Default compliance doc templates that are built-in to DataRobot can be retrieved by using the `type` parameter. A type of NULL or "normal" will retrieve the default template. A type of "timeSeries" can be used to retrieve the default time series template.

Usage

```
GetComplianceDocTemplate(templateId = NULL, type = NULL)
```

Arguments

templateId	character. Optional. The ID of the template to use in generating custom model documentation.
type	character. Optional. The type of compliance doc to get. Can be "normal" to retrieve the default template or "timeSeries" to get the default time series template.

Value

An S3 object of class 'dataRobotComplianceDocTemplate' that contains:

- name character. The name of the compliance doc template.
- creatorUsername character. The name of the user who created the compliance doc template.
- orgId character. The ID of the organization of the creator user.
- creatorId character. The ID of the creator user.
- sections list. The list of sections that define the template.
- id character. The ID of the template.

Examples

```
## Not run:
  GetComplianceDocTemplate()  # get the default template
  GetComplianceDocTemplate(type = "normal")  # get the default template
  GetComplianceDocTemplate(type = "timeSeries")  # get the default time series template
  templateId <- "5cf85080d9436e5c310c796d"
  GetComplianceDocTemplate(templateId) # Get a custom template for a specific ID.

## 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,
  fallbackToParentInsights = FALSE
)
```

Arguments

model	dataRobotModel. A DataRobot model object like that returned by GetModel.
source	character. The data partition for which data would be returned. Default is DataPartition$VALIDATION. See DataPartition for details.
fallbackToParentInsights	logical. If TRUE, this will return the lift chart data for the model's parent if the lift chart is not available for the model and the model has a parent model.

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)
```

GetCrossValidationScores *Get cross validation scores*

Description

Get cross validation scores

Usage

```
GetCrossValidationScores(model, partition = NULL, metric = 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.
partition	numeric. Optional. The ID of the partition to filter results by.
metric	character. Optional. The name of the metric to filter results by.

Value

A list of lists with cross validation score data. Each list contains a series of lists for each model metric. Each model metric list contains the metric data for each fold.

Examples

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

## End(Not run)
```

GetDataSource *Returns information about a particular data source.*

Description

Returns information about a particular data source.

Usage

```
GetDataSource(dataSourceId)
```

Arguments

dataSourceId	character. The id of the data source

Value

A list containing information on the particular data source:

- className character. The Java class name of the driver.
- baseNames character. A vector of the file name(s) of the jar files.
- canonicalName character. The user-friendly name of the driver.
- id character. The dataSourceId of the driver.
- creator character. The userId of the user who created the driver.

Examples

```
## Not run:
 dataSourceId <- "57a7c978c808916f4a630f89"
 GetDataSource(dataSourceId)

## End(Not run)
```

GetDataStore *Returns information about a particular data store.*

Description

Returns information about a particular data store.

Usage

```
GetDataStore(dataStoreId)
```

Arguments

dataStoreId	character. The id of the data store.

Value

A list containing information on the particular data store:

- id character. The dataStoreId of the data store.
- canonicalName character. The user-friendly name of the data store.
- type character. The type of data store.
- updated datetime. A timestamp for the last time the data store was updated.
- creator character. The userId of the user who created the data store.
- params list. A list specifying the data store parameters.

Examples

```
## Not run:
 dataStoreId <- "5c1303269300d900016b41a7"
 GetDataStore(dataStoreId)

## End(Not run)
```

`GetDataStoreSchemas` *Get the schemas associated with a data store.*

Description

Get the schemas associated with a data store.

Usage

```
GetDataStoreSchemas(dataStoreId, username, password)
```

Arguments

`dataStoreId`	character. The ID of the data store to update.
`username`	character. The username to use for authentication to the database.
`password`	character. The password to use for authentication to the database. The password is encrypted at server side and never saved or stored.

Value

A list with the name of the catalog and the name of the schemas.

Examples

```
## Not run:
 dataStoreId <- "5c1303269300d900016b41a7"
 GetDataStoreSchemas(dataStoreId, username = "myUser", password = "mySecurePass129")

## End(Not run)
```

GetDataStoreTables *Get all tables associated with a data store.*

Description

Get all tables associated with a data store.

Usage

```
GetDataStoreTables(dataStoreId, username, password, schema = NULL)
```

Arguments

dataStoreId	character. The ID of the data store to update.
username	character. The username to use for authentication to the database.
password	character. The password to use for authentication to the database. The password is encrypted at server side and never saved or stored.
schema	character. The name of the schema to reference. Optional.

Value

A list with the name of the catalog and the name of the tables.

Examples

```
## Not run:
 dataStoreId <- "5c1303269300d900016b41a7"
 GetDataStoreTables(dataStoreId, username = "myUser", password = "mySecurePass129")

## End(Not run)
```

GetDatetimeModel *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

```
GetDatetimeModel(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. Unique alphanumeric identifier for the featurelist on which the model is based.
- processes character. Vector with components describing preprocessing; may include 'modelType'.
- featurelistName character. The name of the featurelist on which the model is based.
- projectId character. 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. A description of the model.
- metrics list. 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. The model category (e.g., blend, model).
- blueprintId character. The unique DataRobot blueprint identifier on which the model is based.
- modelId character. The unique alphanumeric model identifier.
- projectName character. Optional description of project defined by projectId.
- projectTarget character. The target variable predicted by all models in the project.
- projectMetric character. 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. 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 character. Only present for frozen models in datetime partitioned projects. If specified, the start date of the data used to train the model.
- trainingEndDate character. Only present for frozen models in datetime partitioned projects. If specified, the end date of the data used to train the model.

- backtests list. 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. 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. 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. The score against the holdout, if available and the holdout is unlocked, according to the project metric.
- holdoutStatus character. The status of the holdout score, e.g. "COMPLETED", "HOLDOUT_BOUNDARIES_EXCEEDED".
- effectiveFeatureDerivationWindowStart integer. Only available for time series projects. How many timeUnits into the past relative to the forecast point the user needs to provide history for at prediction time. This can differ from the ‘featureDerivationWindowStart‘ set on the project due to the differencing method and period selected, or if the model is a time series native model such as ARIMA. Will be a negative integer in time series projects and ‘NULL‘ otherwise.
- effectiveFeatureDerivationWindowEnd integer. Only available for time series projects. How many timeUnits into the past relative to the forecast point the feature derivation window should end. Will be a non-positive integer in time series projects and ‘NULL‘ otherwise.
- forecastWindowStart integer. Only available for time series projects. How many timeUnits into the future relative to the forecast point the forecast window should start. Note that this field will be the same as what is shown in the project settings. Will be a non-negative integer in time series projects and ‘NULL‘ otherwise. into the future relative to the forecast point the forecast window should end. Note that this field will be the same as what is shown in the project settings. Will be a non-negative integer in time series projects and ‘NULL‘ otherwise.
- windowsBasisUnit character. Only available for time series projects. Indicates which unit is the basis for the feature derivation window and the forecast window. Note that this field will be the same as what is shown in the project settings. In time series projects, will be either the detected time unit or "ROW", and ‘NULL‘ otherwise.

Examples

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

## 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 GetDatetimeModel

Examples

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

## 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 with following components

- cvMethod. The type of validation scheme used for the project.
- projectId character. The id of the project this partitioning applies to.
- datetimePartitionColumn character. The name of the column whose values as dates are used to assign a row to a particular partition.
- dateFormat character. The format (e.g. " partition column was interpreted (compatible with strftime [https://docs.python.org/2/library/time.html#time.strftime]).
- autopilotDataSelectionMethod character. Whether models created by the autopilot use "rowCount" or "duration" as their dataSelectionMethod.
- validationDuration character. 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. The start date of the available training data for scoring the holdout.
- availableTrainingDuration character. 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. The end date of the available training data for scoring the holdout.
- primaryTrainingStartDate character. The start date of primary training data for scoring the holdout.
- primaryTrainingDuration character. 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. The end date of the primary training data for scoring the holdout.
- gapStartDate character. The start date of the gap between training and holdout scoring data.
- gapDuration character. 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. The end date of the gap between training and holdout scoring data.
- holdoutStartDate character. The start date of holdout scoring data.
- holdoutDuration character. 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. The end date of the holdout scoring data.
- numberOfBacktests integer. the number of backtests used.
- backtests data.frame. A data frame of partition backtest. Each element represent one backtest and has the 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 known in advance. Knon in advance 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 forecast point the forecast 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 forecast point the forecast window should end. Only used for time series projects. Expressed in terms of the `timeUnit` of the `datetimePartitionColumn`.
- featureSettings list. A list of lists specifying settings for each feature. For each feature you would like to set feature settings for, pass the following in a list:
  - featureName character. The name of the feature to set feature settings.
  - knownInAdvance logical. Optional. Whether or not the feature is known in advance. Used for time series only. Defaults to `FALSE`.

  - doNotDerive logical. Optional. If TRUE, no time series derived features (e.g., lags) will be automatically engineered from this feature. Used for time series only. Defaults to FALSE.

- treatAsExponential character. Specifies whether to treat data as exponential trend and apply transformations like log-transform. Uses values from from TreatAsExponential.
- differencingMethod character. Used to specify differencing method to apply if data is stationary. Use values from DifferencingMethod.
- windowsBasisUnit character. Indicates which unit is the basis for the feature derivation window and forecast window. Uses values from TimeUnit and the value "ROW".
- periodicities list. A list of periodicities for different times, specified as a list of lists, where each list item specifies the ‘timeSteps‘ for a particular ‘timeUnit‘. Will be
- 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.
- validationRowCount integer. The number of rows in the validation set.
- multiseriesIdColumns list. A list of the names of multiseries id columns to define series.
- numberOfKnownInAdvanceFeatures integer. The number of known in advance features.
- useCrossSeriesFeatures logical. Whether or not cross series features are included.
- aggregationType character. The aggregation type to apply when creating cross series features. See SeriesAggregationType.
- calendarId character. The ID of the calendar used for this project, if any.

Examples

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

## End(Not run)
```

GetDeployment *Get information on a particular deployment.*

Description

Get information on a particular deployment.

Usage

```
GetDeployment(deploymentId)
```

Arguments

deploymentId	character. The ID of the deployment.

Value

A DataRobotDeployment object containing:

- id character. The ID of the deployment.
- label character. The label of the deployment.
- description character. The description of the deployment.
- defaultPredictionServer list. Information on the default prediction server connected with the deployment. See `ListPredictionServers` for details.
- model dataRobotModel. The model associated with the deployment. See `GetModel` for details.
- capabilities list. Information on the capabilities of the deployment.
- predictionUsage list. Information on the prediction usage of the deployment.
- serviceHealth list. Information on the service health of the deployment.
- modelHealth list. Information on the model health of the deployment.
- accuracyHealth list. Information on the accuracy health of the deployment.

Examples

```
## Not run:
  deploymentId <- "5e319d2e422fbd6b58a5edad"
  GetDeployment(deploymentId)

## End(Not run)
```

GetDeploymentDriftTrackingSettings

Get drift tracking settings for a deployment.

Description

Get drift tracking settings for a deployment.

Usage

```
GetDeploymentDriftTrackingSettings(deploymentId)
```

Arguments

`deploymentId`	character. The ID of the deployment.

Value

A list with the following information on drift tracking:

- associationId
- predictionIntervals list. A list with two keys:
  - enabled. ‘TRUE‘ if prediction intervals are enabled and ‘FALSE‘ otherwise.
  - percentiles list. A list of percentiles, if prediction intervals are enabled.
- targetDrift list. A list with one key, ‘enabled‘, which is ‘TRUE‘ if target drift is enabled, and ‘FALSE‘ otherwise.
- featureDrift list. A list with one key, ‘enabled‘, which is ‘TRUE‘ if feature drift is enabled, and ‘FALSE‘ otherwise.

Examples

```
## Not run:
  deploymentId <- "5e319d2e422fbd6b58a5edad"
  GetDeploymentDriftTrackingSettings(deploymentId)

## End(Not run)
```

GetDriver *Returns information about a particular driver.*

Description

Returns information about a particular driver.

Usage

```
GetDriver(driverId)
```

Arguments

driverId	character. The id of the driver.

Value

A list containing information on the particular driver:

- className character. The Java class name of the driver.
- baseNames character. A vector of the file name(s) of the jar files.
- canonicalName character. The user-friendly name of the driver.
- id character. The driverId of the driver.
- creator character. The userId of the user who created the driver.

Examples

```
## Not run:
 driverId <- "57a7c978c808916f4a630f89"
 GetDriver(driverId)

## End(Not run)
```

`GetFeatureAssociationMatrix` *Get pairwise feature association statistics for a project's informative features*

Description

Get pairwise feature association statistics for a project's informative features

Usage

```
GetFeatureAssociationMatrix(project, associationType, metric)
```

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.
`associationType`	character. The type of association, must be either "association" or "correlation".
`metric`	character. The specified association metric, must be one of "mutualInfo", "cramersV", "spearman", "pearson", or "tau".

Value

A list with two items:

- features data.frame. A data.frame containing the following info for each feature:
  - alphabeticSortIndex integer. A number representing the alphabetical order of this feature compared ot the other features in this dataset.
  - feature character. The name of the feature.
  - importanceSortIndex integer. A number ranking the importance of this feature compared to the other features in this dataset.
  - strengthSortIndex integer. A number ranking the strength of this feature compared to the other features in this dataset.
- strengths data.frame. A data.frame of pairwise strength data, with the following info:
  - feature1 character. The name of the first feature.
  - feature2 character. THe name of the second feature.
  - statistic numeric. Feature association statistics for ‘feature1‘ and ‘feature2‘.

Examples

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

## End(Not run)
```

`GetFeatureAssociationMatrixDetails` *Get a sample of the actual values used to measure the association between a pair of features.*

Description

Get a sample of the actual values used to measure the association between a pair of features.

Usage

```
GetFeatureAssociationMatrixDetails(project, feature1, feature2)
```

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.
`feature1`	character. The name of the first feature of interest.
`feature2`	character. The name of the second feature of interest.

Value

A list with the following info:

- features list. The names of ‘feature1‘ and ‘feature2‘.
- types list. The type of ‘feature1‘ and ‘feature2‘. Will be "C" for categorical and "N" for numeric.
- values data.frame. The values of the feature associations and the relative frequency of the datapoints in the sample.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  GetFeatureAssociationMatrix(projectId, "SepalWidth", "SepalLength")

## End(Not run)
```

`GetFeatureHistogram` *Retrieve histogram plot data for a specific feature*

Description

A histogram is a popular way of visual representation of a feature values distribution in a series of bins. For categorical features every bin represents exactly one of feature values plus the number of occurrences of that value. For numeric features every bin represents a range of values (low end inclusive, high end exclusive) plus the total number of occurrences of all values in this range. In addition to that, with every bin for categorical and numeric features there is also included a target feature average for values in that bin (though it can be missing if the feature is deemed uninformative, if the project target has not been selected yet using `SetTarget`, or if the project is a multiclass project).

Usage

```
GetFeatureHistogram(project, featureName, binLimit = 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.
`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.
`binLimit`	integer. Optional. Desired max number of histogram bins. The default is 60.

Value

list containing:

- count numeric. The number of values in this bin's range. If a project is using weights, the value is equal to the sum of weights of all feature values in the bin's range.
- target numeric. Average of the target feature for values in this bin. It may be NULL if the feature is deemed uninformative, if the target has not yet been set (see `SetTarget`), or if the project is multiclass.
- label character. The value of the feature if categorical, otherwise the low end of the bin range such that the difference between two consecutive bin labels is the length of the bin.

`GetFeatureImpact` *Get the feature impact for a model, requesting the feature impact if it is not already available.*

Description

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'.

Usage

```
GetFeatureImpact(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)`.

Details

Note that `GetFeatureImpact` will block for the duration of feature impact calculation. If you would prefer not to block the call, use `RequestFeatureImpact` to generate an async request for feature impact and then use `GetFeatureImpactForModel` or `GetFeatureImpactForJobId` to get the feature impact when it has been calculated. `GetFeatureImpactForJobId` will also block until the request is complete, whereas `GetFeatureImpactForModel` will error if the job is not complete yet.

`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 character. The name of the feature.
- impactNormalized numeric. The normalized impact score (largest value is 1).
- impactUnnormalized numeric. The unnormalized impact score.
- redundantWith character. A feature that makes this feature redundant, or NA if the feature is not redundant.

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'.

Feature impact also runs redundancy detection, which detects if some features are redundant with higher importance features. Note that some types of projects, like multiclass, do not run redundancy detection. This function will generate a warning if redundancy detection was not run.

Value

A data frame with the following columns:

- featureName character. The name of the feature.
- impactNormalized numeric. The normalized impact score (largest value is 1).
- impactUnnormalized numeric. The unnormalized impact score.
- redundantWith character. A feature that makes this feature redundant, or NA if the feature is not redundant.

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 numeric. 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 belongs 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.
- targetLeakage character. Whether a feature is considered to have target leakage or not. A value of "SKIPPED_DETECTION" indicates that target leakage detection was not run on the feature.

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 elements describing the requested featurelist:

- featurelistId character. The unique alphanumeric identifier for the featurelist.
- projectId character. The project to which the featurelist belongs.
- features character. The names of the variables included in the featurelist.
- name character. The name of the featurelist.
- created character. A timestamp of when the featurelist was created.
- isUserCreated logical. Whether or not the featurelist was created by a user (as opposed to DataRobot automation).
- numModels numeric. The number of models that currently use this 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.
- supportsMonotonicConstraints logical. Whether or not the model supports monotonic constraints.
- monotonicIncreasingFeaturelistId character. The ID of the featurelist specifying the features that are constrained to be monotonically increasing. Will be `NULL` if no increasing constraints are used.
- monotonicDecreasingFeaturelistId character. The ID of the featurelist specifying the features that are constrained to be monotonically decreasing. Will be `NULL` if no decreasing constraints are used.
- isStarred logical. Whether or not the model is starred.
- predictionThreshold numeric. For binary classification projects, the threshold used for predictions.
- predictionThresholdReadOnly logical. Whether or not the prediction threshold can be modified. Typically, the prediction threshold can no longer be modified once a model has a deployment created or predictions have been made with the dedicated prediction API.

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 <- ListModelJobs(project)
  job <- initialJobs[[1]]
  modelJobId <- job$modelJobId
  GetModelJobFromJobId(projectId, modelJobId)

## End(Not run)
```

GetGeneralizedInsight *An internal function to help fetch insights.*

Description

See GetLiftChart, GetRocCurve for details.

Usage

```
GetGeneralizedInsight(
  method,
  model,
  source = DataPartition$VALIDATION,
  fallbackToParentInsights = FALSE
)
```

Arguments

method	character. The API URL to use to get insight information.
model	dataRobotModel. A DataRobot model object like that returned by GetModel.
source	character. The data partition for which data would be returned. Default is DataPartition$VALIDATION. See DataPartition for details.
fallbackToParentInsights	logical. If TRUE, this will return the lift chart data for the model's parent if the lift chart is not available for the model and the model has a parent model.

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 character. Model job status; an element of JobStatus, e.g. JobStatus$Queue.
- url character. URL to request more detail about the job.
- id character. The job id.
- jobType character. See JobType for valid values.
- projectId character. The project that contains the model.
- isBlocked logical. If TRUE, the job is blocked (cannot be executed) until its dependencies are resolved.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  initialJobs <- ListModelJobs(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,
  fallbackToParentInsights = FALSE
)
```

Arguments

model	dataRobotModel. A DataRobot model object like that returned by GetModel.
source	character. The data partition for which data would be returned. Default is DataPartition$VALIDATION. See DataPartition for details.
fallbackToParentInsights	logical. If TRUE, this will return the lift chart data for the model's parent if the lift chart is not available for the model and the model has a parent model.

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)
```

GetMissingValuesReport

Get a report on missing values for the model.

Description

The missing values report is a list of items, one per feature, sorted by missing count in descending order. Each item in the report contains details on the number of missing values for that feature and how they were handled by the model.

Usage

```
GetMissingValuesReport(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

A list containing:

- feature character. The name of the feature.
- type character. Feature type (numeric or categorical).
- missingCount numeric. The number of missing values in the training data for that feature.
- missingPercentage numeric. The percentage of missing values in the training data for the feature.

- tasks list. A list of information on each task that was applied to that feature to handle missing values. This information contains:
  - id character. The id of the node in the model blueprint chart for this task. (See GetBlueprintChart for more information on blueprint charts.)
  - name character. The name of the task.
  - descriptions character. Aggregated information about how the task handles missing values.

Examples

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

## 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.
- supportsMonotonicConstraints logical. Whether or not the model supports monotonic constraints.
- monotonicIncreasingFeaturelistId character. The ID of the featurelist specifying the features that are constrained to be monotonically increasing. Will be `NULL` if no increasing constraints are used.
- monotonicDecreasingFeaturelistId character. The ID of the featurelist specifying the features that are constrained to be monotonically decreasing. Will be `NULL` if no decreasing constraints are used.
- isStarred logical. Whether or not the model is starred.
- predictionThreshold numeric. For binary classification projects, the threshold used for predictions.
- predictionThresholdReadOnly logical. Whether or not the prediction threshold can be modified. Typically, the prediction threshold can no longer be modified once a model has a deployment created or predictions have been made with the dedicated prediction API.

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 external links 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)
```

GetModelCapabilities *Get supported capabilities for a model, e.g., whether it has a word cloud.*

Description

Get supported capabilities for a model, e.g., whether it has a word cloud.

Usage

```
GetModelCapabilities(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

Returns

- supportsBlending logical. Whether or not the model supports blending. See `RequestBlender`.
- supportsMonotonicConstraints logical. Whether or not the model supports monotonic constraints. See `RequestModel`
- hasWordCloud logical. Whether or not the model has a word cloud. See `GetWordCloud`.
- eligibleForPrime logical. Whether or not the model is eligible for Prime. See `CreatePrimeCode`.
- hasParameters logical. Whether or not the model has parameters. See `GetModelParameters`.

Examples

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

## 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 <- ListModelJobs(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 elements describing the requested featurelist:

- featurelistId character. The unique alphanumeric identifier for the featurelist.
- projectId character. The project to which the featurelist belongs.
- features character. The names of the variables included in the featurelist.
- name character. The name of the featurelist.
- created character. A timestamp of when the featurelist was created.
- isUserCreated logical. Whether or not the featurelist was created by a user (as opposed to DataRobot automation).
- numModels numeric. The number of models that currently use this 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 character. Model job status; an element of `JobStatus`, e.g. `JobStatus$Queue`.
- processes list. List of character vectors describing any preprocessing applied.
- projectId character. 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. What kind of model this is - `prime` for DataRobot Prime models, /codeblend for blender models, and /codemodel for other models.
- featurelistId character. Id of the featurelist used in fitting the model.
- blueprintId character. Id of the DataRobot blueprint on which the model is based.
- modelJobId character. Id of the job.
- isBlocked logical. If TRUE, the job is blocked (cannot be executed) until its dependencies are resolved.

Examples

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

## 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)
```

GetModelRecommendation

Retrieve a model recommendation from DataRobot for your project.

Description

Model recommendations are only generated when you run full Autopilot. One of them (the most accurate individual, non-blender model) will be prepared for deployment. In the preparation process, DataRobot will: (1) calculate feature impact for the selected model and use it to generate a reduced feature list, (2) retrain the selected model on the reduced featurelist, (3) will replace the recommended model with the new model if performance is improved on the reduced featurelist, (4) will retrain the model on a higher sample size, and (5) will replace the recommended model with the higher sample size model if it is more accurate.

Usage

```
GetModelRecommendation(project, type = RecommendedModelType$FastAccurate)
```

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.
type	character. The type of recommendation to retrieve. See RecommendedModelType for available options. Defaults to RecommendedModelType$FastAccurate.

Value

A list containing information about the recommended model:

- modelId character. The model ID of the recommended model.
- projectId character. The project ID of the project the recommendations were made for.
- recommendationType character. The type of recommendation being made.

Examples

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

## End(Not run)
```

GetMultiSeriesProperties

Retrieve time series properties for a potential multiseries datetime partition column

---

Description

Multiseries time series projects use multiseries id columns to model multiple distinct series within a single project. This function returns the time series properties (time step and time unit) of this column if it were used as a datetime partition column with the specified multiseries id columns, running multiseries detection automatically if it had not previously been successfully ran.

Usage

```
GetMultiSeriesProperties(
  project,
  dateColumn,
  multiseriesIdColumns,
  crossSeriesGroupByColumns = 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.
dateColumn	character. The name of the column containing the date that defines the time series.
multiseriesIdColumns	character. Optional. The Series ID to demarcate the series. If not specified, DataRobot will attempt to automatically infer the series ID.

crossSeriesGroupByColumns
: character. Optional. Column to split a cross series into further groups. For example, if every series is sales of an individual product, the cross series group could be e product category with values like "men's clothing", "sports equipment", etc. Requires multiseries with `useCrossSeries` enabled.

maxWait
: integer. if a multiseries detection task is run, the maximum amount of time to wait for it to complete before giving up.

Value

A named list which contains:

- timeSeriesEligible logical. Whether or not the series is eligible to be used for time series.
- crossSeriesEligible logical. Whether or not the cross series group by column is eligible for cross-series modeling. Will be NULL if no cross series group by column is used.
- crossSeriesEligibilityReason character. The type of cross series eligibility (or ineligibility).
- 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.
- timeStep integer. Expected difference in time units between rows in the data. Will be NULL for features that are not time series eligible.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  GetMultiSeriesProperties(projectId,
                           dateColumn = "myFeature",
                           multiseriesIdColumns = "Store")

## End(Not run)
```

GetParetoFront *Pareto Front data for a Eureqa model*

Description

The Eureqa algorithm generates millions and millions of equations. Eureqa takes the best bits from the best initial models and splices them randomly into the next generation. After enough mixing, the models can achieve good accuracy. There are usually many equations at every complexity level, but they aren't exposed. The models that are displayed are the "Pareto-optimal" models. That means that for any given complexity score, it shows the model with the best error metric on the training data out of all the modes. After that, for each remaining model, if there a strictly better model, throw out the strictly-worse model. A Pareto Front are those "Pareto-optimal" models that are generated at various complexity scores.

Usage

```
GetParetoFront(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:

- projectId character. the id of the project the model belongs to
- errorMetric character. Eureqa error-metric identifier used to compute error metrics for this search. Note that Eureqa error metrics do NOT correspond 1:1 with DataRobot error metrics – the available metrics are not the same, and even equivalent metrics may be computed slightly differently.
- hyperparameters list. A list of the various hyperparameters that could be used. By default there are none.
- targetType character. Indicating what kind of modeling is being done in this project Options are: "Regression", "Binary" (Binary classification), "Multiclass" (Multiclass classification)
- solutions list. List of pareto points. Every pareto point contains a dictionary with keys:
  - eureqaSolutionId character. ID of this solution
  - complexity numeric. Complexity score for this solution. Complexity score is a function of the mathematical operators used in the current solution. The Complexity calculation can be tuned via model hyperparameters.
  - error numeric. Error for the current solution, as computed by Eureqa using the "error_metric" error metric.
  - expression character. String specifying the Eureqa model equation.
  - expression_annotated character. Eureqa model equation string with variable names tagged for easy identification.

Examples

```
## Not run:
  projectId <- "5b2827556523cd05bd1507a5"
  modelId <- "5b29406c6523cd0665685a8d"
  model <- GetModel(projectId, modelId)
  GetParetoFront(model)

## End(Not run)
```

GetPredictionDataset *Retrieve data on a prediction dataset*

Description

Retrieve data on a prediction dataset

Usage

```
GetPredictionDataset(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	character. The ID of the prediction dataset.

Value

Data for a particular prediction dataset:

- 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"
  datasetId <- "5cd36e6e77a90f79a28ba414"
  GetPredictionDataset(projectId, datasetId)

## End(Not run)
```

GetPredictionExplanations

Get prediction explanations

Description

A streamlined workflow to both generate and retrieve prediction explanations for a model.

Usage

```
GetPredictionExplanations(
  model,
  dataset,
  maxExplanations = NULL,
  thresholdLow = NULL,
  thresholdHigh = NULL,
  batchSize = NULL,
  maxWait = 600,
  excludeAdjustedPredictions = TRUE
)
```

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.
`dataset`	object. Either (1) the prediction dataset object of class `dataRobotPredictionDataset`, (2) a data.frame containing the prediction data, (3) the datasetID of the prediction dataset, (4) a file path to the data, or (5) a URL to the data. References the dataset of predictions used to get prediction explanations for.
`maxExplanations`	integer. Optional. The maximum number of prediction explanations to supply per row of the dataset, default: 3.
`thresholdLow`	numeric. Optional. The lower threshold, below which a prediction must score in order for prediction explanations to be computed for a row in the dataset. If neither `threshold_high` nor `threshold_low` is specified, prediction explanations will be computed for all rows.
`thresholdHigh`	numeric. Optional. The high threshold, above which a prediction must score in order for prediction explanations to be computed. If neither `threshold_high` nor `threshold_low` is specified, prediction explanations will be computed for all rows.
`batchSize`	integer. Optional. Maximum number of prediction explanations rows to retrieve per request
`maxWait`	integer. The maximum time (in seconds) to wait for the model job to complete.
`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 columns:

- rowId integer. Row id from prediction dataset.
- prediction numeric. The output of the model for this row (numeric prediction for regression problem, predicted class for classification problem).

- class1Label character. Label of class 0. Available only for classification problem.
- class1Probability numeric. Predicted probability of class 0. Available only for classification problem.
- class2Label character. Label of class 1. Available only for classification
- class2Probability numeric. Predicted probability of class 1. Available only for classification problem.
- explanation1FeatureName character. The name of the feature contributing to the prediction.
- explanation1FeatureValue character. the value the feature took on for this row.
- explanation1QualitativeStrength numeric. How strongly the feature affected the prediction.
- explanation1Strength character. A human-readable description of how strongly the feature affected the prediction (e.g. '+++', '–', '+').
- explanation1Label character. Describes what output was driven by this prediction explanation. For regression projects, it is the name of the target feature. For classification projects, it is the class whose probability increasing would correspond to a positive strength of this.
- explanationNFeatureName character. The name of the feature contributing to the prediction.
- explanationNFeatureValue character. The value the feature took on for this row.
- explanationNQualitativeStrength numeric. How strongly the feature affected the prediction.
- explanationNStrength character. A human-readable description of how strongly the feature affected the prediction (e.g. '+++', '–', '+').
- explanationNLabel character. Describes what output was driven by this prediction explanation. For regression projects, it is the name of the target feature. For classification projects, it is the class whose probability increasing would correspond to a positive strength of this.
- explanationNFeatureName. Character string the name of the feature contributing to the prediction.

Examples

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

## End(Not run)
```

`GetPredictionExplanationsInitialization` *Retrieve the prediction explanations initialization for a model.*

Description

Prediction explanations initializations are a prerequisite for computing prediction explanations, and include a sample what the computed prediction explanations for a prediction dataset would look like.

Usage

```
GetPredictionExplanationsInitialization(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 belongs to.
- modelId character. The unique alphanumeric model identifier.
- predictionExplanationsSample list. List with sample of prediction explanations. Each element of the list is information about prediction explanations for one data row. For more information see GetPredictionExplanationsRows.

Examples

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

## End(Not run)
```

GetPredictionExplanationsInitializationFromJobId

Retrieve the prediction explanations initialization for a model using jobId

---

Description

Prediction explanations initializations are a prerequisite for computing prediction explanations, and include a sample what the computed prediction explanations for a prediction dataset would look like.

Usage

```
GetPredictionExplanationsInitializationFromJobId(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 prediction explanations job (returned for example by RequestPredictionExplanationsInitialization.)
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 belongs to.
- modelId character. The unique alphanumeric model identifier.
- predictionExplanationsSample list. List with sample of prediction explanations. Each element of the list is information about prediction explanations for one data row.

Examples

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

## End(Not run)
```

GetPredictionExplanationsMetadata

Retrieve metadata for specified prediction explanations

Description

Retrieve metadata for specified prediction explanations

Usage

```
GetPredictionExplanationsMetadata(project, predictionExplanationId)
```

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.
predictionExplanationId	character. Id of the prediction explanations.

Value

A named list which contains prediction explanation metadata:

- id character. ID of the record and prediction explanations computation result.
- projectId character. ID of the project the model belongs to.
- modelId character. ID of the model prediction explanations initialization is for.
- datasetId character. ID of the prediction dataset prediction explanations were computed for.
- maxExplanations integer. Maximum number of prediction explanations to supply per row of the dataset.
- thresholdLow numeric. The low threshold, below which a prediction must score in order for prediction explanations to be computed for a row in the dataset.
- thresholdHigh numeric. The high threshold, above which a prediction must score in order for prediction explanations to be computed for a row in the dataset.
- numColumns integer. The number of columns prediction explanations were computed for.
- finishTime. Numeric timestamp referencing when computation for these prediction explanations finished.
- predictionExplanationsLocation character. Where to retrieve the prediction explanations.

Examples

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

## End(Not run)
```

GetPredictionExplanationsMetadataFromJobId

Retrieve the prediction explanations metadata for a model using jobId

---

Description

Retrieve the prediction explanations metadata for a model using jobId

Usage

```
GetPredictionExplanationsMetadataFromJobId(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 (return for example by RequestPredictionExplanations).
maxWait	integer. The maximum time (in seconds) to wait for the model job to complete.

Value

A named list which contains prediction explanation metadata. For more information see GetPredictionExplanationsMetad

Examples

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

## End(Not run)
```

GetPredictionExplanationsRows

Retrieve all prediction explanations rows

Description

Retrieve all prediction explanations rows

Usage

```
GetPredictionExplanationsRows(
  project,
  predictionExplanationId,
  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.
predictionExplanationId	character. Id of the prediction explanations.

`batchSize` integer. Optional. Maximum number of prediction explanations 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 prediction explanations, each element corresponds to a row of the prediction dataset and has following components.

- 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.
- predictionExplanations. list containing
  - label. described what output was driven by this prediction explanation. For regression projects, it is the name of the target feature. For classification projects, it is the class whose probability increasing would correspond to a positive strength of this prediction explanation.
  - feature. 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
  - qualitativeStrength. 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$id
  model <- GetModel(projectId, modelId)
  jobId <- RequestPredictionExplanations(model, datasetId)
 predictionExplanationId <- GetPredictionExplanationsMetadataFromJobId(projectId, jobId)$id
  GetPredictionExplanationsRows(projectId, predictionExplanationId)

## End(Not run)
```

GetPredictionExplanationsRowsAsDataFrame *Retrieve all prediction explanations 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 "explanationNLabel" will appear corresponding to each included prediction explanation in the row. In both cases, the value of N will start at 1 and count up.

Usage

```
GetPredictionExplanationsRowsAsDataFrame(
  project,
  predictionExplanationId,
  excludeAdjustedPredictions = TRUE,
  batchSize = 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.
`predictionExplanationId`	character. Id of the prediction explanations.
`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.
`batchSize`	integer. Optional. Maximum number of prediction explanations rows to retrieve per request

Value

data frame with following columns:

- rowId integer. Row id from prediction dataset.
- prediction numeric. The output of the model for this row (numeric prediction for regression problem, predicted class for classification problem).
- class1Label character. Label of class 0. Available only for classification problem.
- class1Probability numeric. Predicted probability of class 0. Available only for classification problem.

- class2Label character. Label of class 1. Available only for classification
- class2Probability numeric. Predicted probability of class 1. Available only for classification problem.
- explanation1FeatureName character. The name of the feature contributing to the prediction.
- explanation1FeatureValue character. the value the feature took on for this row.
- explanation1QualitativeStrength numeric. How strongly the feature affected the prediction.
- explanation1Strength character. A human-readable description of how strongly the feature affected the prediction (e.g. '+++', '–', '+').
- explanation1Label character. Describes what output was driven by this prediction explanation. For regression projects, it is the name of the target feature. For classification projects, it is the class whose probability increasing would correspond to a positive strength of this.
- explanationNFeatureName character. The name of the feature contributing to the prediction.
- explanationNFeatureValue character. The value the feature took on for this row.
- explanationNQualitativeStrength numeric. How strongly the feature affected the prediction.
- explanationNStrength character. A human-readable description of how strongly the feature affected the prediction (e.g. '+++', '–', '+').
- explanationNLabel character. Describes what output was driven by this prediction explanation. For regression projects, it is the name of the target feature. For classification projects, it is the class whose probability increasing would correspond to a positive strength of this.
- explanationNFeatureName. 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$id
  model <- GetModel(projectId, modelId)
  jobId <- RequestPredictionExplanations(model, datasetId)
 predictionExplanationId <- GetPredictionExplanationsMetadataFromJobId(projectId, jobId)$id
  GetPredictionExplanationsRowsAsDataFrame(projectId, predictionExplanationId)

## End(Not run)
```

`GetPredictions` *Retrieve model predictions*

Description

This function can be used to retrieve predictions from a project and either (1) a `predictionId` specifying the ID for the predictions desired (use `ListPredictions` to see available predictionIds for individual prediction sets) or (2) a `predictionJobId` that comes from a call to `RequestPredictions`. This function will then return the predictions generated for the model and data.

Usage

```
GetPredictions(
  project,
  predictId,
  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.
predictId	character or integer. Either can be the character id of the predictionId associated with the prediction or the integer predictionJobId that is created by the call to RequestPredictions.
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 the modeling task - binary classification, multiclass classification, or regression; whether or not the underlying data is time series, multiseries, cross-series, or not time series; and the value of the ‘type‘ parameter. For non-time-series 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. You can also fetch the raw dataframe of prediction values using raw.

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

For time series tasks, ‘type = "raw"‘ will return more detailed information on the time series prediction. This will also include any prediction intervals if requested.

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:
  # Retrieve by predictJobID
  dataset <- UploadPredictionDataset(project, diamonds_small)
  model <- ListModels(project)[[1]]
  modelId <- model$modelId
  predictJobId <- RequestPredictions(project, modelId, dataset$id)
  predictions <- GetPredictions(project, predictJobId)
  # Retrieve by predictionID
  predictions <- ListPredictions(project)
  predictions <- GetPredictions(project, predictions$predictionId[[1]])

## 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 <- ListModelJobs(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 information about specified DataRobot Prime model

Examples

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

## End(Not run)
```

`GetProject` *Retrieve details 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)
```

`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)
```

GetRecommendedModel *Retrieve the model object that DataRobot recommends for your project.*

Description

See GetModelRecommendation for details.

Usage

```
GetRecommendedModel(project, type = RecommendedModelType$FastAccurate)
```

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.
type	character. The type of recommendation to retrieve. See RecommendedModelType for available options. Defaults to RecommendedModelType$FastAccurate.

Value

The model object corresponding with that recommendation

Examples

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

## 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,
  fallbackToParentInsights = FALSE
)
```

Arguments

model	dataRobotModel. A DataRobot model object like that returned by GetModel.
source	character. The data partition for which data would be returned. Default is DataPartition$VALIDATION. See DataPartition for details.
fallbackToParentInsights	logical. If TRUE, this will return the lift chart data for the model's parent if the lift chart is not available for the model and the model has a parent model.

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 positive 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 will 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)
```

`GetSeriesAccuracy` *Get the computed series accuracy for a model, computing it if not already computed.*

Description

Get the computed series accuracy for a model, computing it if not already computed.

Usage

```
GetSeriesAccuracy(model, maxWait = 600)
```

Arguments

`model`	character. The model for which you want to compute Feature Impact, e.g. from the list of models returned by `ListModels(project)`.
`maxWait`	integer. How long (in seconds) to wait for series accuracy computation before raising a timeout error? Default 600.

Value

data.frame with items:

- multiseriesId character. The ID of the series.
- rowCount integer. The number of rows in the series.
- multiseriesValues character. The name of the series.
- duration character. The duration of the series.
- validationScore numeric. The validation score for the series.
- backtestingScore numeric. The score on backtests for the series. See `ScoreBacktests`.
- holdoutScore numeric. The score for the series on the holdout set.

Examples

```
## Not run:
  projectId <- "5984b4d7100d2b31c1166529"
  modelId <- "5984b4d7100d2b31c1166529"
  model <- GetModel(projectId, modelId)
  seriesAccuracy <- GetSeriesAccuracy(model)

## End(Not run)
```

`GetSeriesAccuracyForModel` *Get the series accuracy associated with a particular model.*

Description

This will not work if you have not separately computed series accuracy via `RequestSeriesAccuracy`. See `GetSeriesAccuracy` for a function that will get series accuracy and also compute it automatically if it has not already been compute.

Usage

```
GetSeriesAccuracyForModel(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

data.frame with items:

- multiseriesId character. The ID of the series.
- rowCount integer. The number of rows in the series.
- multiseriesValues character. The name of the series.
- duration character. The duration of the series.
- validationScore numeric. The validation score for the series.
- backtestingScore numeric. The score on backtests for the series. See `ScoreBacktests`.
- holdoutScore numeric. The score for the series on the holdout set.

Examples

```
## Not run:
  projectId <- "5984b4d7100d2b31c1166529"
  modelId <- "5984b4d7100d2b31c1166529"
  model <- GetModel(projectId, modelId)
  jobId <- RequestSeriesAccuracy(projectId, modelId)
  WaitForJobToComplete(projectId, jobId)
  seriesAccuracy <- GetSeriesAccuracyForModel(model)

## 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.

`GetTimeSeriesFeatureDerivationLog` *Retrieve the time series feature derivation log content*

Description

The Time Series Feature Derivation Log provides details about the feature generation process for a time series project. It includes information about which features are generated and their priority, as well as the detected properties of the time series data such as whether the series is stationary, and periodicities detected.

Usage

```
GetTimeSeriesFeatureDerivationLog(project, offset = NULL, limit = 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.
`offset`	integer. Optional. Default is 0. This many results will be skipped.
`limit`	integer. Optional. Defaults to 100. At most this many results are returned. To specify no limit, use 0. The default may change without notice.

Details

This route is only supported for time series projects that have finished partitioning. The time series feature log will include information about:

- Detected stationarity of the series (e.g. "Series detected as non-stationary")
- Detected presence of multiplicative trend in the series (e.g., "Multiplicative trend detected")
- Any periodicities (e.g., "Detected periodicities: 7 day")
- Maximum number of feature to be generated (e.g., "Maximum number of feature to be generated is 1440")
- Window sizes used in rolling statistics / lag extractors (e.g., "The window sizes chosen to be: 2 months") (because the time step is 1 month and Feature Derivation Window is 2 months)
- Features that are specified as known-in-advance (e.g., "Variables treated as known in advance: holiday")
- Details about why certain variables are transformed in the input data (e.g., "Generating variable "y (log)" from "y" because multiplicative trend is detected")
- Details about features generated as time series features, and their priority (e.g., "Generating feature "date (actual)" from "date" (priority: 1)")

Value

Returns the feature log output

Examples

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

## End(Not run)
```

`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

Training predictions are the internal out-of-fold predictions for data that was used to train the model. These predictions are especially useful for creating stacked models or blenders.

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)
```

`GetTrainingPredictionsForModel`

Get training predictions for a particular model.

Description

Training predictions are the internal out-of-fold predictions for data that was used to train the model. These predictions are especially useful for creating stacked models or blenders.

Usage

```
GetTrainingPredictionsForModel(model, dataSubset = "all", maxWait = 600)
```

Arguments

`model`	dataRobotModel. The model to get training predictions for.
`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.
`maxWait`	integer. The maximum time (in seconds) to wait for the model job to complete.

Examples

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

## 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	integer. Unique integer identifier (return for example by RequestPredictionExplanations).
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(projectId, modelId)
  jobId <- RequestTrainingPredictions(model, dataSubset = "all")
  trainingPredictions <- GetTrainingPredictionsFromJobId(projectId, jobId)

## End(Not run)
```

GetTransferableModel *Retrieve imported model info using import id*

Description

Retrieve imported model info using import id

Usage

```
GetTransferableModel(importId)
```

Arguments

importId	character. Id of the import.

Value

A list describing uploaded transferable 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 <- UploadTransferableModel("model.drmodel")
  GetTransferableModel(id)

## End(Not run)
```

`GetTuningParameters` *Retrieve data on tuning parameters for a particular model.*

Description

Retrieve data on tuning parameters for a particular model.

Usage

```
GetTuningParameters(model)
```

Arguments

`model`	dataRobotModel. A DataRobot model object to get tuning parameters for.

Value

A list detailing the following about each tuning parameter:

- currentValue character. The current searched values of that parameter.
- defaultValue character. The default value of that parameter.
- parameterId character. A unique ID for that particular parameter.
- parameterName character. The name of the tuning parameter.
- taskName character. The name of the task the parameter is for.
- constraints list. A list describing constraints on the possible values for the parameter. Will be one of `int` or `float` specifying a `min` and `max` value, or will be `select` and will specify possible values from a list of choices. `int` and `float` correspond with integer and floating-point parameter spaces respectively. It is possible for a parameter to be multiple types. Lastly, some parameters will also have a `supportsGridSearch` logical for whether or not that parameter can be grid searched or not.

Examples

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

## 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. 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)
```

IsBlenderEligible *Check whether individual models can be blended together*

Description

Check whether individual models can be blended together

Usage

```
IsBlenderEligible(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	list. A list of model ids corresponding to the models to check.
blendMethod	character. The blender method to check. See BlendMethods.

Value

List with:

- blendable logical. Whether or not the models can be blended.
- reason character. An explanation for why the models cannot be blended, if not blendable. Otherwise "".

Examples

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

## End(Not run)
```

IsId *Checks if an id is a valid DataRobot ID (24 character string)*

Description

Checks if an id is a valid DataRobot ID (24 character string)

Usage

```
IsId(id)
```

Arguments

id	character. An ID to test whether it is a valid DataRobot ID.

IsParameterIn *Check if a parameter is in a list of possibilities.*

Description

Check if a parameter is in a list of possibilities.

Usage

```
IsParameterIn(
  paramValue,
  paramPossibilities,
  allowNULL = TRUE,
  paramName = NULL
)
```

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.
paramName	character. The name of the parameter to check.

Value

TRUE if paramValue is valid, otherwise returns an error message.

Examples

```
## Not run:
  IsParameterIn("all", DataSubset)

## 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 options.

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 10.

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)
```

ListCalendars *List all available calendars.*

Description

List all available calendars.

Usage

```
ListCalendars()
```

Value

A list of S3 objects of class "dataRobotCalendar"

Examples

```
## Not run:
   ListCalendars()

## End(Not run)
```

ListComplianceDocTemplates *Retrieve information about all compliance doc templates.*

Description

Retrieve information about all compliance doc templates.

Usage

```
ListComplianceDocTemplates(namePart = NULL, limit = NULL, offset = NULL)
```

Arguments

namePart	character. Return only compliance doc templates that have a name that contains this string.
limit	integer. Return only this many compliance doc templates.
offset	integer. Skip this many compliance doc templates before returning.

Value

list of available compliance doc templates. Contains:

- name character. The name of the compliance doc template.
- creatorUsername character. The name of the user who created the compliance doc template.
- orgId character. The ID of the organization of the creator user.
- creatorId character. The ID of the creator user.
- sections list. The list of sections that define the template.
- id character. The ID of the template.

Examples

```
## Not run:
 # Get all compliance doc templates
 ListComplianceDocTemplates()
 Get the first three compliance doc templates with names that contain "foo".
 ListComplianceDocTemplates(namePart = "foo", limit = 3)

## 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, fallbackToParentInsights = FALSE)
```

Arguments

`model`	dataRobotModel. A DataRobot model object like that returned by `GetModel`.
`fallbackToParentInsights`	logical. If TRUE, this will return the lift chart data for the model's parent if the lift chart is not available for the model and the model has a parent model.

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)
```

`ListDataSources` *Returns a dataframe with information on available data sources.*

Description

Returns a dataframe with information on available data sources.

Usage

```
ListDataSources()
```

Value

data.frame containing information on possible data sources.

Examples

```
## Not run:
 ListDataSources()

## End(Not run)
```

`ListDataStores` *Returns a dataframe with information on available data stores.*

Description

Returns a dataframe with information on available data stores.

Usage

```
ListDataStores()
```

Value

data.frame containing information on possible data stores.

Examples

```
## Not run:
 ListDataStores()

## End(Not run)
```

`ListDeployments` *List all current model deployments.*

Description

List all current model deployments.

Usage

```
ListDeployments()
```

Value

A list of DataRobotDeployment objects containing:

- id character. The ID of the deployment.
- label character. The label of the deployment.
- description character. The description of the deployment.
- defaultPredictionServer list. Information on the default prediction server connected with the deployment. See `ListPredictionServers` for details.
- model dataRobotModel. The model associated with the deployment. See `GetModel` for details.
- capabilities list. Information on the capabilities of the deployment.
- predictionUsage list. Information on the prediction usage of the deployment.
- serviceHealth list. Information on the service health of the deployment.
- modelHealth list. Information on the model health of the deployment.
- accuracyHealth list. Information on the accuracy health of the deployment.

Examples

```
## Not run:
  ListDeployments()

## End(Not run)
```

ListDrivers *Returns a dataframe with information on available drivers.*

Description

Returns a dataframe with information on available drivers.

Usage

```
ListDrivers()
```

Value

data.frame containing information on possible drivers.

Examples

```
## Not run:
 ListDrivers()

## 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 numeric. 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 belongs 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.
- targetLeakage character. Whether a feature is considered to have target leakage or not. A value of "SKIPPED_DETECTION" indicates that target leakage detection was not run on the feature.

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 character. Model job status; an element of `JobStatus`, e.g. `JobStatus$Queue`.
- url character. URL to request more detail about the job.
- id character. The job id.
- jobType character. See `JobType` for valid values.
- projectId character. The project that contains the model.
- isBlocked logical. If TRUE, the job is blocked (cannot be executed) until its dependencies are resolved.

Examples

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

## End(Not run)
```

`ListLiftCharts` *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

```
ListLiftCharts(model, fallbackToParentInsights = FALSE)
```

Arguments

`model`	dataRobotModel. A DataRobot model object like that returned by `GetModel`.
`fallbackToParentInsights`	logical. If TRUE, this will return the lift chart data for the model's parent if the lift chart is not available for the model and the model has a parent model.

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)
  ListLiftCharts(model)

## 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)
```

`ListModelJobs` *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

```
ListModelJobs(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.
- isBlocked logical. If TRUE, the job is blocked (cannot be executed) until its dependencies are resolved.

Examples

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

## End(Not run)
```

`ListModelRecommendations` *Retrieve information about model recommendation made by DataRobot for your project.*

Description

DataRobot will help pick out a few models from your project that meet certain criteria, such as being the most accurate model or being a model that captures a good blend of both prediction speed and model accuracy.

Usage

```
ListModelRecommendations(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 list containing information about each recommendation made by DataRobot, containing:

- modelId character. The model ID of the recommended model.
- projectId character. The project ID of the project the recommendations were made for.
- recommendationType character. The type of recommendation being made.

Examples

```
## Not run:
 projectId <- "5984b4d7100d2b31c1166529"
 ListModelRecommendations(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 ListProjects; (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, orderBy = NULL, filter = 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.
`orderBy`	character. Optional. A vector of keys to order the list by. You can order by `metric` or `samplePct`. If the sort attribute is preceded by a hyphen, models will be sorted in descending order, otherwise in ascending order. Multiple sort attributes can be included as a comma-delimited string or in a vector.
`filter`	list. Optional. A named list of parameters to search a model by, such as `name`, `samplePct`, or `isStarred`.

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)
  ListModels(projectId, orderBy=c("samplePct", "-metric"))
  ListModels(projectId, filter=list("sample_pct__gt" = 64, "name" = "Ridge"))
  ListModels(projectId, filter=list("isStarred" = TRUE))

## 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)
```

ListPredictionExplanationsMetadata *Retrieve metadata for prediction explanations in specified project*

Description

Retrieve metadata for prediction explanations in specified project

Usage

```
ListPredictionExplanationsMetadata(
  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 prediction explanations computed for this model will be returned.
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 prediction explanations in the project. Each element of list is metadata for one prediction explanations (for format see GetPredictionExplanationsMetadata).

Examples

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

## End(Not run)
```

`ListPredictions` *Fetch all computed predictions for a project.*

Description

This function itemizes the predictions available for a given project, model, and/or dataset. Note that this function does not actually return the predictions. Use `GetPredictions(projectId,predictionId)` to get the predictions for a particular set of predictions.

Usage

```
ListPredictions(project, modelId = NULL, datasetId = 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`	numeric. Optional. Filter returned predictions to only be predictions made against the model specified by this model ID.
`datasetId`	numeric. Optional. Filter returned predictions to only be predictions made against the prediction dataset specified by this dataset ID.

Value

A data.frame specifying:

- projectId character. The ID of the project the predictions were made in.
- datasetId character. The dataset ID of the dataset used to make predictions
- modelId character. The model ID of the model used to make predictions.
- predictionId character. The unique ID corresponding to those predictions. Use `GetPredictions(projectId,predicti` to fetch the individual predictions.
- includesPredictionIntervals logical. Whether or not the predictions include prediction intervals. See `Predict` for details.
- predictionIntervalsSize integer. Optional. The size, in percent, of prediction intervals or NULL if there are no intervals. See `Predict` for details.

Examples

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

## End(Not run)
```

ListPredictionServers *List all available prediction servers.*

Description

List all available prediction servers.

Usage

```
ListPredictionServers()
```

Value

A list of DataRobotPredictionServer objects containing:

- id character. The ID of the prediction server.
- url character. The URL of the prediction server.
- dataRobotKey character. The key used to access the prediction server.

Examples

```
## Not run:
  ListPredictionServers()

## 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)
```

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

Description

This function returns an S3 object of class projectSummaryList that describes all (optionally filtered) 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

```
ListProjects(filter = NULL)
```

Arguments

`filter`	list. Optional. A named list that can be used to specify various filters. Currently 'projectName' is supported which will filter returned projects for projects with names containing the specified string.

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). Use `ProjectStage` to get a list of options.
- 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:
  ListProjects()
  ListProjects(filter = list("projectName" = "TimeSeries"))

## 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)
```

ListRocCurves *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

```
ListRocCurves(model, fallbackToParentInsights = FALSE)
```

Arguments

model	dataRobotModel. A DataRobot model object like that returned by GetModel.
fallbackToParentInsights	logical. If TRUE, this will return the lift chart data for the model's parent if the lift chart is not available for the model and the model has a parent model.

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)
  ListRocCurves(model)

## End(Not run)
```

ListSharingAccess *List information about which users have what kinds of access to a shared object.*

Description

Note that currently only data sources and data stores can be shared with this API.

Usage

```
ListSharingAccess(object, batchSize = NULL)
```

Arguments

object	object. The shared object to inspect access for.
batchSize	integer. The number of requests per page to expect.

Value

A list specifying information on access:

- username character. The name of the user with access.
- userId character. The ID of the user with access.
- role character. The type of access granted. See SharingRole for options.
- canShare logical. Whether the user can further share access.

Examples

```
## Not run:
 dataStoreId <- "5c1303269300d900016b41a7"
 dataStore <- GetDataStore(dataStoreId)
 ListSharingAccess(dataStore)

## End(Not run)
```

`ListStarredModels` *List all the starred models in a project.*

Description

Star models and add them to this list using `StarModel` or `ToggleStarForModel`. Unstar models and remove them from this list using `UnstarModel` or `ToggleStarForModel`

Usage

```
ListStarredModels(project, orderBy = 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.
`orderBy`	character. Optional. A vector of keys to order the list by. You can order by `metric` or `samplePct`. If the sort attribute is preceded by a hyphen, models will be sorted in descending order, otherwise in ascending order. Multiple sort attributes can be included as a comma-delimited string or in a vector.

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"
   ListStarredModels(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)
```

ListTransferableModels

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

```
ListTransferableModels(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 transferable 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:
  ListTransferableModels()

## End(Not run)
```

`MakeDataRobotRequest` *Make a HTTP request*

Description

Make a HTTP request

Usage

```
MakeDataRobotRequest(
  requestMethod,
  routeString,
  addUrl = TRUE,
  returnRawResponse = TRUE,
  as = "json",
  simplifyDataFrame = TRUE,
  body = NULL,
  query = NULL,
  timeout = DefaultHTTPTimeout,
  encode = NULL,
  followLocation = TRUE,
  filename = NULL,
  stopOnError = TRUE
)
```

Arguments

`requestMethod`	function. A function from httr (e.g., 'httr::GET', 'httr::POST') to use.
`routeString`	character. The path to make the request on.
`addUrl`	logical. Should the endpoint be prepended to the routeString? (Default TRUE).
`returnRawResponse`	logical. Whether to return the raw httr response object (as opposed to postprocessing and returning the content of that object, which is the default.)
`as`	character. What should the resulting data be interpreted as? (default "json"). Use "file" to download as a file (see `filename`).
`simplifyDataFrame`	logical. Whether to invoke `jsonlite::simplifyDataFrame`.
`body`	list. The body of the request for POST.
`query`	list. The query parameters for GET.
`timeout`	numeric. How many seconds before the request times out?
`encode`	character. What should the body be encoded as for the JSON request?
`followLocation`	logical. Should HTTR follow the location if provided? (Default TRUE).
`filename`	character. The path of the file to download to, if it is a download request.
`stopOnError`	logical. If there is an error, should it be raised as a fatal R error? (Default TRUE).

ModelReplacementReason *Model replacement reason*

Description

Model replacement reason

Usage

```
ModelReplacementReason
```

Format

An object of class `list` of length 6.

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

Same as time units, but kept for backwards compatibility.

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.

`Predict` *Retrieve model predictions*

Description

This function can be used to predict with a particular model.

Usage

```
Predict(
  model,
  predictionDataset,
  classPrefix = "class_",
  maxWait = 600,
  forecastPoint = NULL,
  predictionsStartDate = NULL,
  predictionsEndDate = NULL,
  type = "response",
  includePredictionIntervals = FALSE,
  predictionIntervalsSize = 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.
`predictionDataset`	object. Either a dataframe of data to predict on or a DataRobot prediction dataset object of class `dataRobotPredictionDataset`.
`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.
`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.
`predictionsStartDate`	datetime. Optional. Only specified in time series projects. The start date for bulk predictions. This parameter should be provided in conjunction `predictionsEndDate`. Can't be provided with `forecastPoint` parameter.
`predictionsEndDate`	datetime. Optional. Only specified in time series projects. The end date for bulk predictions. This parameter should be provided in conjunction `predictionsStartDate`. Can't be provided with `forecastPoint` parameter.
`type`	character. String specifying the type of response for binary classifiers; see Details.

includePredictionIntervals
: logical. Optional. Should prediction intervals bounds should be part of predictions? Only available for time series projects. See "Details" for more info.

predictionIntervalsSize
: numeric. Optional. Size of the prediction intervals, in percent. Only available for time series projects. See "Details" for more info.

Details

The contents of the return vector depends on the modeling task - binary classification, multiclass classification, or regression; whether or not the underlying data is time series, multiseries, cross-series, or not time series; and the value of the /codetype parameter. For non-time-series regression tasks, the type parameter is ignored and a vector of numerical predictions of the response variable is returned.

This function will error if the requested job has errored, or if it does not complete within `maxWait` seconds.

See `RequestPredictions` and `GetPredictions` for more details.

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:
   trainIndex <- sample(nrow(iris) * 0.7)
   trainIris <- iris[trainIndex, ]
   testIris <- iris[-trainIndex, ]
   project <- StartProject(trainIris, "iris", target = "Petal_Width", wait = TRUE)
   model <- GetRecommendedModel(project)
   predictions <- Predict(model, testIris)

   # Or, if prediction intervals are desired (datetime only)
   model <- GetRecommendedModel(datetimeProject)
   predictions <- Predict(datetimeModel,
                          dataset,
                          includePredictionIntervals = TRUE,
                          predictionIntervalsSize = 100,
                          type = "raw")

## End(Not run)
```

`predict.dataRobotModel`
: *Retrieve model predictions using R's default S3* `predict` *method.*

Description

Retrieve model predictions using R's default S3 `predict` method.

Usage

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

Arguments

`object`	dataRobotModel. The object of class `dataRobotModel` to predict with.
`...`	list. Additional arguments to pass to `Predict`

See Also

Predict

Examples

```
## Not run:
   trainIndex <- sample(nrow(iris) * 0.7)
   trainIris <- iris[trainIndex, ]
   testIris <- iris[-trainIndex, ]
   project <- StartProject(trainIris, "iris", target = "Petal_Width", wait = TRUE)
   model <- GetRecommendedModel(project)
   predictions <- predict(model, testIris)

## End(Not run)
```

`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.

ProjectStage *Project stage*

Description

Project stage

Usage

```
ProjectStage
```

Format

An object of class `list` of length 4.

RecommendedModelType *Recommended model type values*

Description

MostAccurate retrieves the most accurate model based on validation or cross-validation results. In most cases, this will be a blender model.

Usage

```
RecommendedModelType
```

Format

An object of class list of length 3.

Details

FastAccurate retrieves the most accurate individual model (not blender) that passes set guidelines for prediction speed. If no models meet the prediction speed guideline, this will not retrieve anything.

RecommendedForDeployment retrieves the most accurate individual model. This model will have undergone specific pre-preparations to be deployment ready. See GetModelRecommendation for details.

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)
```

ReplaceDeployedModel *Replace a model in a deployment with another model.*

Description

Replace a model in a deployment with another model.

Usage

```
ReplaceDeployedModel(
  deploymentId,
  newModelId,
  replacementReason,
  maxWait = 600
)
```

Arguments

deploymentId	character. The ID of the deployment.
newModelId	character. The ID of the model to use in the deployment. This model will replace the old model. You can also pass a dataRobotModel object.
replacementReason	character. Optional. The reason for replacing the deployment. See ModelReplacementReason for a list of reasons.
maxWait	integer. How long to wait (in seconds) for the computation to complete before returning a timeout error? (Default 600 seconds)

Value

A DataRobotDeployment object containing:

- id character. The ID of the deployment.
- label character. The label of the deployment.
- description character. The description of the deployment.
- defaultPredictionServer list. Information on the default prediction server connected with the deployment. See ListPredictionServers for details.
- model dataRobotModel. The model associated with the deployment. See GetModel for details.
- capabilities list. Information on the capabilities of the deployment.
- predictionUsage list. Information on the prediction usage of the deployment.
- serviceHealth list. Information on the service health of the deployment.
- modelHealth list. Information on the model health of the deployment.
- accuracyHealth list. Information on the accuracy health of the deployment.

Examples

```
## Not run:
  deploymentId <- "5e319d2e422fbd6b58a5edad"
  newModelId <- "5996f820af07fc605e81ead4"
  ReplaceDeployedModel(deploymentId, newModelId, ModelReplacementReason$Other)

## End(Not run)
```

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

Description

This function will create several rulesets that approximate the specified model. The code used in the approximation can be downloaded to be run locally. Currently 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 rulesets 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 PrimeFileId 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)
```

RequestCrossSeriesDetection

Format a cross series with group by columns.

Description

Call this function to request the project be formatted as a cross series project with a group by column.

Usage

```
RequestCrossSeriesDetection(
  project,
  dateColumn,
  multiseriesIdColumns = NULL,
  crossSeriesGroupByColumns = 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.
dateColumn	character. The name of the column containing the date that defines the time series.

`multiseriesIdColumns`
: character. Optional. The Series ID to demarcate the series. If not specified, DataRobot will attempt to automatically infer the series ID.

`crossSeriesGroupByColumns`
: character. Optional. Column to split a cross series into further groups. For example, if every series is sales of an individual product, the cross series group could be e product category with values like "men's clothing", "sports equipment", etc. Requires multiseries with `useCrossSeries` enabled.

`maxWait`
: integer. The maximum time (in seconds) to wait for the model job to complete.

Details

Note that this function no longer needs to be called directly, but is called indirectly as a part of `SetTarget` (which itself is called indirectly as part of `StartProject`) when you pass a /code-crossSeriesGroupByColumn using `CreateDatetimePartitionSpecification`.

Value

A named list which contains:

- timeSeriesEligible logical. Whether or not the series is eligible to be used for time series.
- crossSeriesEligible logical. Whether or not the cross series group by column is eligible for cross-series modeling. Will be NULL if no cross series group by column is used.
- crossSeriesEligibilityReason character. The type of cross series eligibility (or ineligibility).
- 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.
- timeStep integer. Expected difference in time units between rows in the data. Will be NULL for features that are not time series eligible.

`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 <- GetDatetimeModel(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)
```

RequestMultiSeriesDetection *Format a multiseries.*

Description

Call this function to request the project be formatted as a multiseries project, with the dateColumn specifying the time series.

Usage

```
RequestMultiSeriesDetection(
  project,
  dateColumn,
  multiseriesIdColumns = 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.
dateColumn	character. The name of the column containing the date that defines the time series.
multiseriesIdColumns	character. Optional. The Series ID to demarcate the series. If not specified, DataRobot will attempt to automatically infer the series ID.
maxWait	integer. The maximum time (in seconds) to wait for the model job to complete.

Details

Note that as of v2.13 this function no longer needs to be called directly, but is called indirectly as a part of `SetTarget` (which itself is called indirectly as part of `StartProject`) when you pass a multiseries partition using `CreateDatetimePartitionSpecification`.

Value

A named list which contains:

- timeSeriesEligible logical. Whether or not the series is eligible to be used for time series.
- crossSeriesEligible logical. Whether or not the cross series group by column is eligible for cross-series modeling. Will be NULL if no cross series group by column is used.
- crossSeriesEligibilityReason character. The type of cross series eligibility (or ineligibility).
- 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.
- timeStep integer. Expected difference in time units between rows in the data. Will be NULL for features that are not time series eligible.

`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,
  monotonicIncreasingFeaturelistId = NULL,
  monotonicDecreasingFeaturelistId = 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.
`monotonicIncreasingFeaturelistId`	character. Optional. The id of the featurelist that defines the set of features with a monotonically increasing relationship to the target. If `NULL` (default), the default for the project will be used (if any). Note that currently there is no way to create a model without monotonic constraints if there was a project-level default set. If desired, the featurelist itself can also be passed as this parameter.
`monotonicDecreasingFeaturelistId`	character. Optional. The id of the featurelist that defines the set of features with a monotonically decreasing relationship to the target. If `NULL`, the default for the project will be used (if any). If empty (i.e., `""`), no such constraints are enforced. Also, if desired, the featurelist itself can be passed as this parameter.

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)
```

RequestPredictionExplanations *Request prediction explanations computation for a specified model and dataset.*

Description

In order to create PredictionExplanations for a particular model and dataset, you must first: Compute feature impact for the model via `RequestFeatureImpact()` Compute a PredictionExplanationsInitialization for the model via `RequestPredictionExplanationsInitialization()` Compute predictions for the model and dataset via'`RequestPredictions()` After prediction explanations are requested information about them can be accessed using the functions `GetPredictionExplanationsMetadataFrom` and `GetPredictionExplanationsMetadata`. Prediction explanations themselves can be accessed using the functions `GetPredictionExplanationsRows`, `GetPredictionExplanationsRowsAsDataFrame`, and `DownloadPredictionExplanations`.

Usage

```
RequestPredictionExplanations(
  model,
  datasetId,
  maxExplanations = 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. ID of the prediction dataset for which prediction explanations are requested.
`maxExplanations`	integer. Optional. The maximum number of prediction explanations to supply per row of the dataset, default: 3.
`thresholdLow`	numeric. Optional. The lower threshold, below which a prediction must score in order for prediction explanations to be computed for a row in the dataset. If neither `threshold_high` nor `threshold_low` is specified, prediction explanations will be computed for all rows.
`thresholdHigh`	numeric. Optional. The high threshold, above which a prediction must score in order for prediction explanations to be computed. If neither `threshold_high` nor `threshold_low` is specified, prediction explanations will be computed for all rows.

Details

thresholdHigh and thresholdLow are optional filters applied to speed up computation. When at least one is specified, only the selected outlier rows will have prediction explanations computed. Rows are considered to be outliers if their predicted value (in case of regression projects) or probability of being the positive class (in case of classification projects) is less than threshold_low or greater than thresholdHigh. If neither is specified, prediction explanations 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(projectId, modelId)
  RequestPredictionExplanations(model, datasetId)

## End(Not run)
```

RequestPredictionExplanationsInitialization

Request prediction explanations initialization for specified model

---

Description

Prediction explanations initializations are a prerequisite for computing prediction explanations, and include a sample of what the computed prediction explanations for a prediction dataset would look like.

Usage

```
RequestPredictionExplanationsInitialization(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)
  RequestPredictionExplanationsInitialization(model)

## End(Not run)
```

`RequestPredictions` *Request predictions from a model against a previously uploaded dataset*

Description

Prediction intervals can now be returned for predictions with datetime models. Use 'includePredictionIntervals = TRUE' in calls to /codePredict or /codeRequestPredictions. For each model, prediction intervals estimate the range of values DataRobot expects actual values of the target to fall within. They are similar to a confidence interval of a prediction, but are based on the residual errors measured during the backtesting for the selected model.

Usage

```
RequestPredictions(
  project,
  modelId,
  datasetId,
  includePredictionIntervals = NULL,
  predictionIntervalsSize = 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`	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`)
`includePredictionIntervals`	logical. Optional. Should prediction intervals bounds should be part of predictions? Only available for time series projects. See "Details" for more info.
`predictionIntervalsSize`	numeric. Optional. Size of the prediction intervals, in percent. Only available for time series projects. See "Details" for more info.

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$modelId
  predictJobId <- RequestPredictions(project, modelId, dataset$id)
  predictions <- GetPredictions(project, predictJobId)

  # Or, if prediction intervals are desired (datetime only)
  predictJobId <- RequestPredictions(datetimeProject,
                                     DatetimeModelId,
                                     includePredictionIntervals = TRUE,
                                     predictionIntervalsSize = 100)
  predictions <- GetPredictions(datetimeProject, predictJobId, type = "raw")

## 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 ruleset 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)
```

`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)
```

`RequestSeriesAccuracy` *Compute the series accuracy for a model.*

Description

Note that you can call `GetSeriesAccuracy` without calling this function, and the series accuracy will be requested automatically.

Usage

```
RequestSeriesAccuracy(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

Job ID for the async job associated with the computation.

Examples

```
## Not run:
  projectId <- "5984b4d7100d2b31c1166529"
  modelId <- "5984b4d7100d2b31c1166529"
  model <- GetModel(projectId, modelId)
  jobId <- RequestSeriesAccuracy(projectId, modelId)
  WaitForJobToComplete(projectId, jobId)

## 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)
```

RequestTransferableModel

Request generation of an transferable 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 DownloadTransferableModel for that.

Description

Request generation of an transferable 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 DownloadTransferableModel for that.

Usage

```
RequestTransferableModel(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")
  DownloadTransferableModel(projectObject, modelId, file)

## End(Not run)
```

`RunInteractiveTuning` *Run an interactive model tuning session.*

Description

The advanced tuning feature allows you to manually set model parameters and override the DataRobot default selections. It is generally available for Eureqa models. To use this feature with other model types, contact your CFDS for more information.

Usage

```
RunInteractiveTuning(model)
```

Arguments

`model`	dataRobotModel. A DataRobot model object to get tuning parameters for.

Details

This function runs an interactive session to iterate you through individual arguments for each tunable hyperparameter, presenting you with the defaults and other available information. You can set each parameter one at a time, skipping ones you don't intend to set. At the end, it will return a job ID that can be used to get the tuned model.

Note that sometimes you may see the exact same parameter more than once. These are for different parts of the blueprint that use the same parameter (e.g., one hot encoding for text and then one hot encoding for numeric). They are listed in the order they are found in the blueprint but unfortunately more user-facing information cannot be provided.

Value

A job ID that can be used to get the tuned model.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  myXGBModel <- GetModel(projectId, modelId)
  tuningJob <- RunInteractiveTuning(myXGBModel)
  tunedModel <- GetModelFromJobId(projectId, tuningJob)

## 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.

Details

For `Disabled`, no scaleout models will run in autopilot or be available in blueprints.

For `RepositoryOnly`, scaleout models will be available in blueprints and can be run manually, but will not run in autopilot.

For `Auopilot`, scaleout models will run during autopilot and also be available in blueprints.

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, wait = FALSE)
```

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.
`wait`	logical. If TRUE, wait until job completion.

Value

job ID of pending job if `wait` is FALSE. Use `WaitForJobToComplete` to await job completion. If `wait` is TRUE, will wait until completion and return `NULL`. Upon completion, all available backtests will have scores computed.

Examples

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

## End(Not run)
```

SeriesAggregationType *Series aggregation type*

Description

For details, see "Calculating features across series" in the Time Series section of the DataRobot user guide.

Usage

```
SeriesAggregationType
```

Format

An object of class `list` of length 2.

SetPredictionThreshold *Set a custom prediction threshold for binary classification models.*

Description

The prediction threshold is used by a binary classification model when deciding between the positive and negative class.

Usage

```
SetPredictionThreshold(model, threshold)
```

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.
`threshold`	numeric. The threshold to use when deciding between the positive and negative class. Should be between 0 and 1 inclusive.

Details

Note: This feature can only can be used when `PredictionThresholdReadOnly` is `FALSE`. Models typically cannot have their prediction threshold modified if they have been used to set a deployment or predictions have been made with the dedicated prediction API.

Value

Returns NULL but updates the model in place.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  model <- GetModel(projectId, modelId)
  SetPredictionThreshold(model, threshold = 0.6)

## 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,
  monotonicIncreasingFeaturelistId = NULL,
  monotonicDecreasingFeaturelistId = NULL,
  onlyIncludeMonotonicBlueprints = FALSE,
  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.

`monotonicIncreasingFeaturelistId`
: character. Optional. The id of the featurelist that defines the set of features with a monotonically increasing relationship to the target. If `NULL` (default), no such constraints are enforced. When specified, this will set a default for the project that can be overridden at model submission time if desired. The featurelist itself can also be passed as this parameter.

`monotonicDecreasingFeaturelistId`
: character. Optional. The id of the featurelist that defines the set of features with a monotonically decreasing relationship to the target. If `NULL` (default), no such constraints are enforced. When specified, this will set a default for the project that can be overridden at model submission time if desired. The featurelist itself can also be passed as this parameter.

`onlyIncludeMonotonicBlueprints`
: logical. Optional. When TRUE, only blueprints that support enforcing monotonic constraints will be available in the project or selected for the autopilot.

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)
```

`SetupProjectFromDataSource`

Create a project from a data source.

Description

Create a project from a data source.

Usage

```
SetupProjectFromDataSource(
  dataSourceId,
  username,
  password,
  projectName = NULL,
  maxWait = 60 * 60
)
```

Arguments

`dataSourceId`	character. The ID of the data source to create a project from.
`username`	character. The username to use for authentication to the database.
`password`	character. The password to use for authentication to the database.
`projectName`	character. Optional. String specifying a project name. The password is encrypted at server side and never saved or stored.
`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

project object for the created project.

Examples

```
## Not run:
 dataSourceId <- "5c1303269300d900016b41a7"
 SetupProjectFromDataSource(dataSourceId, username = "username", password = "hunter1",
                            projectName = "My Project")

## End(Not run)
```

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

Description

This function is deprecated. Use `SetupProjectFromDataSource` instead.

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.

Details

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.

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)
```

Share *Share a shareable object with a particular user.*

Description

See SharingRole for more details on available access levels that can be granted to a user. Set role to NULL to revoke access to a particular user.

Usage

```
Share(object, username, role = "default", canShare = NULL)
```

Arguments

object	object. The shared object to inspect access for.
username	character. The name of the user to share the object with.
role	character. The role (access level) to give that user. See SharingRole.
canShare	logical. Is the user allowed to further reshare?

Examples

```
## Not run:
 dataStoreId <- "5c1303269300d900016b41a7"
 dataStore <- GetDataStore(dataStoreId)
 # Grant access to a particular user.
 Share(dataStore, "foo@foo.com")
 # Grant access in a Read Only role.
 Share(dataStore, "foo@foo.com", role = SharingRole$ReadOnly)
 # Revoke access
 Share(dataStore, "foo@foo.com", role = NULL)

## End(Not run)
```

SharingRole *Sharing role*

Description

This is a list that contains the valid values for granting access to other users (see `Share`). If you wish, you can specify access roles using the list values, e.g., `SharingRole$ReadWrite` instead of typing the string "READ_WRITE". This way you can benefit from autocomplete and not have to remember the valid options.

Usage

```
SharingRole
```

Format

An object of class `list` of length 6.

Details

`Owner` allows any action including deletion.

`ReadWrite` or `Editor` allows modifications to the state, e.g., renaming and creating data sources from a data store, but *not* deleting the entity.

`ReadOnly` or `Consumer` - for data sources, enables creating projects and predictions; for data stores, allows viewing them only.

StarModel *Star a model.*

Description

Star a model.

Usage

```
StarModel(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

the model object, but now starred

Examples

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

## 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)
```

StartProject *Start a project, set the target, and run autopilot.*

Description

This function is a convenient shorthand to start a project and set the target. See SetupProject and SetTarget.

Usage

```
StartProject(
  dataSource,
  projectName = NULL,
  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,
  monotonicIncreasingFeaturelistId = NULL,
  monotonicDecreasingFeaturelistId = NULL,
  onlyIncludeMonotonicBlueprints = FALSE,
  workerCount = NULL,
  wait = FALSE,
  checkInterval = 20,
  timeout = NULL,
  username = NULL,
  password = NULL,
  verbosity = 1,
  maxWait = 600
)
```

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.

`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.

`monotonicIncreasingFeaturelistId`
: character. Optional. The id of the featurelist that defines the set of features with a monotonically increasing relationship to the target. If `NULL` (default), no such constraints are enforced. When specified, this will set a default for the project that can be overridden at model submission time if desired. The featurelist itself can also be passed as this parameter.

`monotonicDecreasingFeaturelistId`
: character. Optional. The id of the featurelist that defines the set of features with a monotonically decreasing relationship to the target. If `NULL` (default), no such constraints are enforced. When specified, this will set a default for the project that can be overridden at model submission time if desired. The featurelist itself can also be passed as this parameter.

`onlyIncludeMonotonicBlueprints`
: logical. Optional. When TRUE, only blueprints that support enforcing monotonic constraints will be available in the project or selected for the autopilot.

`workerCount`
: integer. The number of workers to run (default 2). Use `"max"` to set to the maximum number of workers available.

`wait`
: logical. If `TRUE`, invokes `WaitForAutopilot` to block execution until the autopilot is complete.

`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.

`username`
: character. The username to use for authentication to the database.

`password`
: character. The password to use for authentication to the database.

`verbosity`
: numeric. Optional. 0 is silent, 1 or more displays information about progress. Default is 1.

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"
  StartProject(iris,
               projectName = "iris",
               target = "Species",
               targetType = TargetType$Multiclass)

## 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. Maximum 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)

`StartTuningSession` *Create a function to initiate hyperparameter tuning for a particular model.*

Description

The advanced tuning feature allows you to manually set model parameters and override the DataRobot default selections.

Usage

```
StartTuningSession(model)
```

Arguments

`model`	dataRobotModel. A DataRobot model object to get tuning parameters for.

Value

A function that can be used to tune the model. The function will take `model`, the model object to tune, and will have individual arguments for each tunable hyperparameter that are each set to the default value for that hyperparameter. Furthermore, the function takes `tuningDescription` which can be used to describe the hyperparameter tuning taking place for future reference. The function itself will return a job ID that can be used to get the tuned model.

See Also

RunInteractiveTuning

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
  modelId <- "5996f820af07fc605e81ead4"
  myXGBModel <- GetModel(projectId, modelId)
  RunTune <- StartTuningSession(myXGBModel)
  tuningJob <- RunTune(myXGBModel, colsample_bytree = 0.4, colsample_bylevel = 0.8)
  tunedModel <- GetModelFromJobId(projectId, tuningJob)

## End(Not run)
```

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. The summary of object has the following components: modelType, expandedModel (constructed from modelType and processes), modelId, blueprintId, and projectId.

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 <- ListProjects()
```

```
  summary(projectSummary)

## End(Not run)
```

---

summary.listOfDataRobotTuningParameters
Summarize the list of tuning parameters available for a model.

Description

Summarize the list of tuning parameters available for a model.

Usage

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

Arguments

object	list. The list of tuning parameters to summarize.
...	list. Extra parameters that are ignored. Used to allow S3 inheritance to work.

Value

A data.frame detailing the following about each tuning parameter:

- name character. The name of the tuning parameter.
- current character. The current searched values of that parameter.
- default character. The default value of that parameter.
- constraint character. A short description of the possible values that parameter can take.

Examples

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

## End(Not run)
```

TargetLeakageType *Target leakage report values*

Description

Target leakage report values

Usage

```
TargetLeakageType
```

Format

An object of class `list` of length 4.

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.

TestDataStore *Test the database connection to the data store.*

Description

Test the database connection to the data store.

Usage

```
TestDataStore(dataStoreId, username, password)
```

Arguments

dataStoreId	character. The ID of the data store to update.
username	character. The username to use for authentication to the database.
password	character. The password to use for authentication to the database. The password is encrypted at server side and never saved or stored.

Value

TRUE if successful, otherwise it will error.

Examples

```
## Not run:
 dataStoreId <- "5c1303269300d900016b41a7"
 TestDataStore(dataStoreId, username = "myUser", password = "mySecurePass129")

## End(Not run)
```

TimeUnits *Time units*

Description

Time units

Usage

```
TimeUnits
```

Format

An object of class `list` of length 8.

ToggleStarForModel *Star a model if it is unstarred, otherwise unstar the model.*

Description

Star a model if it is unstarred, otherwise unstar the model.

Usage

```
ToggleStarForModel(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

the model object, but now starred if unstarred or unstarred if starred.

Examples

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

## End(Not run)
```

TreatAsExponential *Treat as exponential*

Description

Treat as exponential

Usage

```
TreatAsExponential
```

Format

An object of class list of length 3.

TryingToSubmitNull *Checks to see if we are trying to submit 'NULL' as a value.*

Description

Checks to see if we are trying to submit 'NULL' as a value.

Usage

```
TryingToSubmitNull(body)
```

Arguments

body	list. The body to check for NULL.

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)
```

UnstarModel *Unstar a model.*

Description

Unstar a model.

Usage

```
UnstarModel(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

the model object, but now unstarred

Examples

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

## End(Not run)
```

`UpdateAccess` *Update access to a particular object.*

Description

Update access to a particular object.

Usage

```
UpdateAccess(object, access)
```

Arguments

`object`	object. The shared object to inspect access for.
`access`	dataRobotAccessList. A list specifying access given to all users. See `ListSharingAccess`.

Examples

```
## Not run:
 dataStoreId <- "5c1303269300d900016b41a7"
 dataStore <- GetDataStore(dataStoreId)
 access <- ListSharingAccess(dataStore)
 # Remove access from the first user and grant it to foo@foo.com instead.
 access[[1]]$username <- "foo@foo.com"
 UpdateAccess(dataStore, access)
 # Change access to a Read Only role.
 access[[1]]$role <- SharingRole$ReadOnly
 UpdateAccess(dataStore, access)

## End(Not run)
```

UpdateCalendar *Update a calendar*

Description

Currently supports changing the name of a calendar.

Usage

```
UpdateCalendar(calendarId, name = NULL)
```

Arguments

calendarId	character. The ID of the calendar to retrieve.
name	character. The new name to name the calendar.

Value

An S3 object of class "dataRobotCalendar"

Examples

```
## Not run:
   calendarId <- "5da75da31fb4a45b8a815a53"
   UpdateCalendar(calendarId, name = "New name for calendar")

## End(Not run)
```

UpdateComplianceDocTemplate *Update the name or sections of an existing doc template.*

Description

Note that default templates cannot be updated.

Usage

```
UpdateComplianceDocTemplate(templateId, name = NULL, sections = NULL)
```

Arguments

templateId	character. The ID of the template to update.
name	character. Optional. A new name to identify the compliance doc template by.
sections	list. Optional. Section definitions for the compliance template.

Value

The updated compliance doc template object.

Examples

```
## Not run:
sections <- list(list("title" = "Missing Values Report",
                      "highlightedText" = "NOTICE",
                      "regularText" = paste("This dataset had a lot of Missing Values."
                                            "See the chart below: {{missingValues}}"),
                      "type" = "user"),
                 list("title" = "Blueprints",
                     "regularText" = "{{blueprintDiagram}} /n Blueprint for this model",
                      "type" = "user"))
  templateId <- "5cf85080d9436e5c310c796d"
  UpdateComplianceDocTemplate(templateId, name = "newName", sections = sections)

## End(Not run)
```

`UpdateDataSource` *Update a data store.*

Description

Update a data store.

Usage

```
UpdateDataSource(
  dataSourceId,
  canonicalName = NULL,
  dataStoreId = NULL,
  query = NULL,
  table = NULL,
  schema = NULL,
  partitionColumn = NULL,
  fetchSize = NULL
)
```

Arguments

`dataSourceId`	character. The ID of the data store to update.
`canonicalName`	character. The user-friendly name of the data source.
`dataStoreId`	character. The ID of the data store to connect to.
`query`	character. A query to execute on the data store to get the data. Optional.
`table`	character. The specified database table. Optional.

schema	character. The specified database schema. Optional.
partitionColumn	character. The name of the partition column. Optional.
fetchSize	integer. a user specified fetch size in the range [1, 20000]. Optional. By default a fetchSize will be assigned to balance throughput and memory usage

Examples

```
## Not run:
 dataSourceId <- "5c1303269300d900016b41a7"
 UpdateDataSource(dataSourceId, canonicalName = "Different Name")

## End(Not run)
```

`UpdateDataStore` *Update a data store.*

Description

Update a data store.

Usage

```
UpdateDataStore(
  dataStoreId,
  canonicalName = NULL,
  driverId = NULL,
  jdbcUrl = NULL
)
```

Arguments

dataStoreId	character. The ID of the data store to update.
canonicalName	character. The user-friendly name of the data store.
driverId	character. The ID of the driver to use.
jdbcUrl	character. The full JDBC url.

Examples

```
## Not run:
 dataStoreId <- "5c1303269300d900016b41a7"
 UpdateDataStore(dataStoreId, canonicalName = "Different Name")

## End(Not run)
```

`UpdateDeploymentDriftTrackingSettings` *Update drift tracking settings for a deployment.*

Description

Update drift tracking settings for a deployment.

Usage

```
UpdateDeploymentDriftTrackingSettings(
  deploymentId,
  targetDriftEnabled = NULL,
  featureDriftEnabled = NULL,
  maxWait = 600
)
```

Arguments

`deploymentId`	character. The ID of the deployment.
`targetDriftEnabled`	logical. Optional. Set to TRUE to enable target drift. Set to FALSE to disable.
`featureDriftEnabled`	logical. Optional. Set to TRUE to enable feature drift. Set to FALSE to disable.
`maxWait`	integer. How long to wait (in seconds) for the computation to complete before returning a timeout error? (Default 600 seconds)

Value

A list with the following information on drift tracking:

- associationId
- predictionIntervals list. A list with two keys:
  - enabled. 'TRUE' if prediction intervals are enabled and 'FALSE' otherwise.
  - percentiles list. A list of percentiles, if prediction intervals are enabled.
- targetDrift list. A list with one key, 'enabled', which is 'TRUE' if target drift is enabled, and 'FALSE' otherwise.
- featureDrift list. A list with one key, 'enabled', which is 'TRUE' if feature drift is enabled, and 'FALSE' otherwise.

Examples

```
## Not run:
  deploymentId <- "5e319d2e422fbd6b58a5edad"
  UpdateDeploymentDriftTrackingSettings(deploymentId, targetDriftEnabled = TRUE)

## End(Not run)
```

UpdateFeaturelist *Update a featurelist*

Description

Updates a featurelist to change the name or description.

Usage

```
UpdateFeaturelist(featurelist, listName = NULL, description = NULL)
```

Arguments

featurelist	list. The featurelist to delete.
listName	character. String identifying the new featurelist to be created.
description	character. A user-friendly description to give a featurelist.

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.

UpdateModelingFeaturelist *Update a modeling featurelist*

Description

Updates a modeling featurelist to change the name or description.

Usage

```
UpdateModelingFeaturelist(featurelist, listName = NULL, description = NULL)
```

Arguments

featurelist	list. The modeling featurelist to delete.
listName	character. String identifying the new featurelist to be created.
description	character. A user-friendly description to give a featurelist.

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	character. Updated value for the projectName parameter associated with the project.
workerCount	integer. The number of workers to run (default 2). Use "max" to set to the maximum number of workers available.
holdoutUnlocked	logical. Either NULL (default) or 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)
```

UpdateTransferableModel *Update the display name or note for an imported model.*

Description

Update the display name or note for an imported model.

Usage

```
UpdateTransferableModel(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 transferable 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 <- UploadTransferableModel("model.drmodel")
  UpdateTransferableModel(id, displayName = "NewName", note = "This is my note.")

## End(Not run)
```

UploadComplianceDocTemplate *Upload a compliance doc template.*

Description

The structure of the compliance doc template can be specified by either a file specified by `filename` or by specifying it with a list via `sections`.

Usage

```
UploadComplianceDocTemplate(name, filename = NULL, sections = NULL)
```

Arguments

`name`	character. A name to identify the compliance doc template by.
`filename`	character. Optional. Filename of file to save the compliance doc template to.
`sections`	list. Optional. Section definitions for the compliance template.

Value

Nothing returned, but uploads the compliance doc template.

Examples

```
## Not run:
 ## Create a compliance documentation template from uploading a file
 DownloadComplianceDocTemplate("template.json")
 # Edit template.json in your favorite editor
 UploadComplianceDocTemplate("myTemplate", "template.json")

## Create a compliance documentation template from a list.
sections <- list(list("title" = "Missing Values Report",
                      "highlightedText" = "NOTICE",
                      "regularText" = paste("This dataset had a lot of Missing Values.",
                                            "See the chart below: {{missingValues}}"),
                      "type" = "user"),
                 list("title" = "Blueprints",
                     "regularText" = "{{blueprintDiagram}} /n Blueprint for this model",
                      "type" = "user"))

## 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,
  predictionsStartDate = NULL,
  predictionsEndDate = NULL,
  relaxKIAFeaturesCheck = 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.
predictionsStartDate	datetime. Optional. Only specified in time series projects. The start date for bulk predictions. This parameter should be provided in conjunction predictionsEndDate. Can't be provided with forecastPoint parameter.
predictionsEndDate	datetime. Optional. Only specified in time series projects. The end date for bulk predictions. This parameter should be provided in conjunction predictionsStartDate. Can't be provided with forecastPoint parameter.
relaxKIAFeaturesCheck	logical. For Time Series projects only. If True, missing values in the known in advance features are allowed in the forecast window at the prediction time. If omitted or FALSE, missing values are not allowed.
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.
- dataQualityWarnings list. A list of available warnings about potential problems in the uploaded prediction dataset. Will be empty if there are no warnings.

Examples

```
## Not run:
  projectId <- "59a5af20c80891534e3c2bde"
```

```
  UploadPredictionDataset(projectId, iris)

## End(Not run)
```

`UploadPredictionDatasetFromDataSource` *Upload a prediction dataset from a data source.*

Description

Upload a prediction dataset from a data source.

Usage

```
UploadPredictionDatasetFromDataSource(
  project,
  dataSourceId,
  username,
  password,
  forecastPoint = NULL,
  maxWait = 600,
  relaxKIAFeaturesCheck = 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.
`dataSourceId`	character. The id of the data source
`username`	character. The username to use for authentication to the database.
`password`	character. The password to use for authentication to the database. The password is encrypted at server side and never saved or stored.
`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.
`relaxKIAFeaturesCheck`	logical. For Time Series projects only. If True, missing values in the known in advance features are allowed in the forecast window at the prediction time. If omitted or FALSE, missing values are not allowed.

Examples

```
## Not run:
 dataSourceId <- "5c1303269300d900016b41a7"
 TestDataStore(dataSourceId, username = "myUser", password = "mySecurePass129")

## End(Not run)
```

`UploadTransferableModel` *Import a previously exported model for predictions.*

Description

Import a previously exported model for predictions.

Usage

```
UploadTransferableModel(modelFile, maxWait = 600)
```

Arguments

`modelFile`	character. Path to binary transferable model file.
`maxWait`	integer. Specifies how many seconds to wait for upload to finish.

Value

A list describing uploaded transferable 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:
  UploadTransferableModel("model.drmodel")

## End(Not run)
```

`ValidateCalendar` *Get a calendar id from a calendar object.*

Description

Get a calendar id from a calendar object.

Usage

```
ValidateCalendar(calendar)
```

Arguments

`calendar`	object. Either list with calendarId element or calendarId value

---

`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.

ValidateMultiSeriesProperties *Validate that the multiseries properties indicate a successful multiseries setup.*

Description

Validate that the multiseries properties indicate a successful multiseries setup.

Usage

```
ValidateMultiSeriesProperties(properties, error = TRUE)
```

Arguments

properties	list. List of multiseries properties.
error	logical. Should an error be raised if there is an issue?

Value

TRUE if all properties verify, otherwise FALSE or raises error.

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)
```

`ValidatePartition` *Checks if a partition is valid.*

Description

Checks if a partition is valid.

Usage

```
ValidatePartition(validationType, partition, reps = NULL, validationPct = NULL)
```

Arguments

`validationType`	character. The type of partition to validate.
`partition`	partition. The partition object.
`reps`	numeric. The number of repetitions for a CV validation.
`validationPct`	numeric. The size of the validation set for TVH validation.

`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

ValidateReplaceDeployedModel

Validate a potential deployment model replacement.

Description

Validate a potential deployment model replacement.

Usage

```
ValidateReplaceDeployedModel(deploymentId, newModelId)
```

Arguments

deploymentId	character. The ID of the deployment.
newModelId	character. The ID of the model to use in the deployment. This model will replace the old model. You can also pass a dataRobotModel object.

Value

A validation report with:

- status character. Either PASSED or FAILED depending on whether all checks passed or not.
- message character. A message explaining the status failure, if any.
- checks list. A list of each check and the individual status.

Examples

```
## Not run:
  deploymentId <- "5e319d2e422fbd6b58a5edad"
  newModelId <- "5996f820af07fc605e81ead4"
  ValidateReplaceDeployedModel(deploymentId, newModelId)

## End(Not run)
```

VariableTransformTypes

Types of variable transformations

Description

Types of variable transformations

Usage

```
VariableTransformTypes
```

Format

An object of class `list` of length 4.

`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

∗ **datasets**
AutopilotMode, 10
BlendMethods, 11
cvMethods, 34
DataPartition, 34
DataSubset, 35
DifferencingMethod, 45
JobStatus, 141
JobType, 142
ModelReplacementReason, 169
PeriodicityMaxTimeStep, 170
PeriodicityTimeUnits, 170
PostgreSQLdrivers, 172
PrimeLanguage, 176
ProjectStage, 176
RecommendedModelType, 177
ScaleoutModelingMode, 200
SeriesAggregationType, 201
SharingRole, 209
TargetLeakageType, 219
TargetType, 219
TimeUnits, 220
TreatAsExponential, 221
VariableTransformTypes, 239

AddEureqaSolution, 7
ApplySchema, 8
as.data.frame, 8
AutopilotMode, 10

BatchFeaturesTypeTransform, 10
BlendMethods, 11
BlueprintChartToGraphviz, 12

CheckUrl, 12
CleanServerData, 13
ConnectToDataRobot, 13
ConstructDurationString, 14
CreateBacktestSpecification, 15
CreateCalendar, 16
CreateComplianceDocumentation, 17
CreateDataSource, 17
CreateDataStore, 18
CreateDatetimePartitionSpecification, 19
CreateDeployment, 22
CreateDerivedFeatureAsCategorical *(*CreateDerivedFeatures*)*, 24
CreateDerivedFeatureAsNumeric *(*CreateDerivedFeatures*)*, 24
CreateDerivedFeatureAsText *(*CreateDerivedFeatures*)*, 24
CreateDerivedFeatureIntAsCategorical *(*CreateDerivedFeatures*)*, 24
CreateDerivedFeatures, 24
CreateFeaturelist, 25
CreateGroupPartition, 26, *29*, *31*, *33*
CreateModelingFeaturelist, 27
CreatePrimeCode, 28
CreateRandomPartition, *27*, 29, *31*, *33*
CreateRatingTable, 30
CreateStratifiedPartition, *27*, *29*, 31, *33*
CreateUserPartition, *27*, *29*, *31*, 32
CrossValidateModel, 33
cvMethods, 34

DataPartition, 34
DataPathFromDataArg, 35
datarobot *(*datarobot-package*)*, 7
datarobot-package, 7
DataSubset, 35
DeleteCalendar, 36
DeleteComplianceDocTemplate, 36
DeleteDataSource, 37
DeleteDataStore, 37
DeleteDeployment, 38
DeleteFeaturelist, 38
DeleteJob, 39
DeleteModel, 39
DeleteModelingFeaturelist, 40

DeleteModelJob, 41
DeletePredictionDataset, 41
DeletePredictionExplanations, 42
DeletePredictionExplanationsInitialization, 43
DeletePredictJob, 43
DeleteProject, 44
DeleteTransferableModel, 45
DifferencingMethod, 45
DownloadComplianceDocTemplate, 46
DownloadComplianceDocumentation, 47
DownloadPredictionExplanations, 48
DownloadPrimeCode, 49
DownloadRatingTable, 50
DownloadScoringCode, 50
DownloadSeriesAccuracy, 51
DownloadTimeSeriesFeatureDerivationLog, 52
DownloadTrainingPredictions, 53
DownloadTransferableModel, 54

ExpectHasKeys, 54

FeatureFromAsyncUrl, 55

GenerateDatetimePartition, 55
GetBlenderModel, 58
GetBlenderModelFromJobId, 59
GetBlueprint, 61
GetBlueprintChart, 62, *93*
GetBlueprintDocumentation, 62
GetCalendar, 63
GetCalendarFromProject, 64
GetComplianceDocTemplate, 64
GetConfusionChart, 65
GetCrossValidationScores, 67
GetDataSource, 67
GetDataStore, 68
GetDataStoreSchemas, 69
GetDataStoreTables, 70
GetDatetimeModel, 70
GetDatetimeModelFromJobId, 73
GetDatetimePartition, 74
GetDeployment, 76
GetDeploymentDriftTrackingSettings, 77
GetDriver, 78
GetFeatureAssociationMatrix, 79
GetFeatureAssociationMatrixDetails, 80
GetFeatureHistogram, 81
GetFeatureImpact, 82
GetFeatureImpactForJobId, 82
GetFeatureImpactForModel, 83
GetFeatureInfo, 84
GetFeaturelist, 86
GetFrozenModel, 87
GetFrozenModelFromJobId, 89
GetGeneralizedInsight, 90
GetJob, 90
GetLiftChart, 91
GetMissingValuesReport, 92
GetModel, 93
GetModelBlueprintChart, 95
GetModelBlueprintDocumentation, 95
GetModelCapabilities, 96
GetModelFromJobId, 97
GetModelingFeaturelist, 98
GetModelJob, 99
GetModelParameters, 100
GetModelRecommendation, 101
GetMultiSeriesProperties, 102
GetParetoFront, 103
GetPredictionDataset, 104
GetPredictionExplanations, 105
GetPredictionExplanationsInitialization, 107
GetPredictionExplanationsInitializationFromJobId, 108
GetPredictionExplanationsMetadata, 109
GetPredictionExplanationsMetadataFromJobId, 110
GetPredictionExplanationsRows, 111
GetPredictionExplanationsRowsAsDataFrame, 113
GetPredictions, 114
GetPredictJob, 116
GetPredictJobs, 117
GetPrimeEligibility, 118
GetPrimeFile, 118
GetPrimeFileFromJobId, 119
GetPrimeModel, 120
GetPrimeModelFromJobId, 121
GetProject, 122
GetProjectStatus, 123
GetRatingTable, 124
GetRatingTableFromJobId, 124
GetRatingTableModel, 125
GetRatingTableModelFromJobId, 126

GetRecommendedModel, 127
GetRocCurve, 127
GetRulesets, 128
GetSeriesAccuracy, 129
GetSeriesAccuracyForModel, 130
GetServerDataInRows, 131
GetTimeSeriesFeatureDerivationLog, *52*, 132
GetTrainingPredictionDataFrame, 133
GetTrainingPredictions, 133
GetTrainingPredictionsForModel, 134
GetTrainingPredictionsFromJobId, 135
GetTransferableModel, 135
GetTuningParameters, 137
GetValidMetrics, 138
GetWordCloud, 138

IsBlenderEligible, 139
IsId, 140
IsParameterIn, 140

JobStatus, 141
JobType, 142

ListBlueprints, 142
ListCalendars, 143
ListComplianceDocTemplates, 143
ListConfusionCharts, 144
ListDataSources, 145
ListDataStores, 145
ListDeployments, 146
ListDrivers, 147
ListFeatureInfo, 147
ListFeaturelists, 148
ListJobs, 149
ListLiftCharts, 150
ListModelFeatures, 151
ListModelingFeaturelists, 152
ListModelJobs, 153
ListModelRecommendations, 154
ListModels, 155
ListPredictionDatasets, 156
ListPredictionExplanationsMetadata, 157
ListPredictions, 158
ListPredictionServers, 159
ListPrimeFiles, 159
ListPrimeModels, 160
ListProjects, 161
ListRatingTableModels, 162
ListRatingTables, 163
ListRocCurves, 163
ListSharingAccess, 164
ListStarredModels, 165
ListTrainingPredictions, 166
ListTransferableModels, 166

MakeDataRobotRequest, 168
ModelReplacementReason, 169

PauseQueue, 169
PeriodicityMaxTimeStep, 170
PeriodicityTimeUnits, 170
plot.listOfModels, 170
PostgreSQLdrivers, 172
Predict, 173
predict.dataRobotModel, 174
PredictionDatasetFromAsyncUrl, 175
PrimeLanguage, 176
ProjectFromAsyncUrl, 176
ProjectStage, 176

RecommendedModelType, 177
ReformatMetrics, 177
RenameRatingTable, 178
ReplaceDeployedModel, 178
RequestApproximation, 179
RequestBlender, 180
RequestCrossSeriesDetection, 181
RequestFeatureImpact, 182
RequestFrozenDatetimeModel, 183
RequestFrozenModel, 184
RequestMultiSeriesDetection, 185
RequestNewDatetimeModel, 187
RequestNewModel, 188
RequestNewRatingTableModel, 190
RequestPredictionExplanations, 191
RequestPredictionExplanationsInitialization, 192
RequestPredictions, 193
RequestPrimeModel, 194
RequestSampleSizeUpdate, 195
RequestSeriesAccuracy, 196
RequestTrainingPredictions, 197
RequestTransferableModel, 198
RunInteractiveTuning, 199

ScaleoutModelingMode, 200

ScoreBacktests, 200
SeriesAggregationType, 201
SetPredictionThreshold, 201
SetTarget, 202
SetupProject, 205
SetupProjectFromDataSource, 206
SetupProjectFromHDFS, 207
Share, 208
SharingRole, 209
StarModel, 209
StartNewAutoPilot, 210
StartProject, 211
StartRetryWaiter, 214
StartTuningSession, 215
Stringify, 216
summary.dataRobotModel, 216
summary.dataRobotProject
(summary.dataRobotModel), 216
summary.listOfBlueprints
(summary.dataRobotModel), 216
summary.listOfDataRobotTuningParameters, 218
summary.listOfFeaturelists
(summary.dataRobotModel), 216
summary.listOfModels
(summary.dataRobotModel), 216
summary.projectSummaryList
(summary.dataRobotModel), 216

TargetLeakageType, 219
TargetType, 219
TestDataStore, 219
TimeUnits, 220
ToggleStarForModel, 220
TreatAsExponential, 221
TryingToSubmitNull, 221

UnpauseQueue, 222
UnstarModel, 222
UpdateAccess, 223
UpdateCalendar, 224
UpdateComplianceDocTemplate, 224
UpdateDataSource, 225
UpdateDataStore, 226
UpdateDeploymentDriftTrackingSettings, 227
UpdateFeaturelist, 228
UpdateModelingFeaturelist, 228
UpdateProject, 229
UpdateTransferableModel, 229
UploadComplianceDocTemplate, 231
UploadData, 232
UploadPredictionDataset, 232
UploadPredictionDatasetFromDataSource, 234
UploadTransferableModel, 235

ValidateCalendar, 236
ValidateModel, 236
ValidateMultiSeriesProperties, 237
ValidateParameterIn, 237
ValidatePartition, 238
ValidateProject, 238
ValidateReplaceDeployedModel, 239
VariableTransformTypes, 239
ViewWebModel, 240
ViewWebProject, 240

WaitForAutopilot, 241
WaitForJobToComplete, 242