

# Package ‘previsionio’

November 17, 2021

**Type** Package

**Title** 'Prevision.io' R SDK

**Version** 11.3.3

**Description** For working with the 'Prevision.io' AI model management platform's API <<https://prevision.io/>>.

**License** MIT + file LICENSE

**Encoding** UTF-8

**RoxygenNote** 7.1.2

**Imports** data.table, futile.logger, httr, jsonlite, Metrics, graphics,  
stats, utils, XML, magrittr, plotly

**Suggests** testthat

**Depends** R (>= 2.10)

**NeedsCompilation** no

**Author** Florian Laroumagne [aut, cre],  
Prevision.io Inc [cph]

**Maintainer** Florian Laroumagne <[florian.laroumagne@prevision.io](mailto:florian.laroumagne@prevision.io)>

**Repository** CRAN

**Date/Publication** 2021-11-17 09:20:02 UTC

## R topics documented:

create_connector . . . . .	4
create_dataframe_from_dataset . . . . .	5
create_dataset_embedding . . . . .	5
create_dataset_from_dataframe . . . . .	6
create_dataset_from_datasource . . . . .	6
create_dataset_from_file . . . . .	7
create_datasource . . . . .	7
create_deployment_api_key . . . . .	8
create_deployment_model . . . . .	9
create_deployment_predictions . . . . .	10

create_experiment . . . . .	10
create_experiment_version . . . . .	11
create_export . . . . .	14
create_exporter . . . . .	14
create_folder . . . . .	15
create_pipeline_trigger . . . . .	16
create_prediction . . . . .	16
create_project . . . . .	17
create_project_user . . . . .	18
delete_connector . . . . .	19
delete_dataset . . . . .	19
delete_datasource . . . . .	20
delete_deployment . . . . .	20
delete_experiment . . . . .	21
delete_exporter . . . . .	21
delete_folder . . . . .	22
delete_pipeline . . . . .	22
delete_prediction . . . . .	23
delete_project . . . . .	23
delete_project_user . . . . .	24
get_best_model_id . . . . .	24
get_connectors . . . . .	25
get_connector_id_from_name . . . . .	25
get_connector_info . . . . .	26
get_datasets . . . . .	26
get_dataset_embedding . . . . .	27
get_dataset_head . . . . .	27
get_dataset_id_from_name . . . . .	28
get_dataset_info . . . . .	28
get_datasources . . . . .	29
get_datasource_id_from_name . . . . .	29
get_datasource_info . . . . .	30
get_deployments . . . . .	30
get_deployment_api_keys . . . . .	31
get_deployment_app_logs . . . . .	31
get_deployment_id_from_name . . . . .	32
get_deployment_info . . . . .	32
get_deployment_predictions . . . . .	33
get_deployment_prediction_info . . . . .	33
get_deployment_usage . . . . .	34
get_experiments . . . . .	34
get_experiment_id_from_name . . . . .	35
get_experiment_info . . . . .	35
get_experiment_version_features . . . . .	36
get_experiment_version_id . . . . .	36
get_experiment_version_info . . . . .	37
get_experiment_version_models . . . . .	37
get_experiment_version_predictions . . . . .	38

get_exporters . . . . .	38
get_exporter_exports . . . . .	39
get_exporter_id_from_name . . . . .	39
get_exporter_info . . . . .	40
get_features_infos . . . . .	40
get_folder . . . . .	41
get_folders . . . . .	41
get_folder_id_from_name . . . . .	42
get_model_cv . . . . .	42
get_model_feature_importance . . . . .	43
get_model_hyperparameters . . . . .	43
get_model_infos . . . . .	44
get_pipelines . . . . .	44
get_pipeline_id_from_name . . . . .	45
get_pipeline_info . . . . .	45
get_prediction . . . . .	46
get_prediction_infos . . . . .	46
get_projects . . . . .	47
get_project_id_from_name . . . . .	47
get_project_info . . . . .	48
get_project_users . . . . .	48
helper_cv_classif_analysis . . . . .	49
helper_drift_analysis . . . . .	49
helper_optimal_prediction . . . . .	50
helper_plot_classif_analysis . . . . .	51
pause_experiment_version . . . . .	52
pio_download . . . . .	52
pio_init . . . . .	53
pio_list_to_df . . . . .	53
pio_request . . . . .	54
resume_experiment_version . . . . .	54
stop_experiment_version . . . . .	55
test_connector . . . . .	55
test_datasource . . . . .	56
test_deployment_type . . . . .	56
test_pipeline_type . . . . .	57
update_experiment_version_description . . . . .	57
update_project_user_role . . . . .	58

---

create_connector	<i>Create a new connector of a supported type (among: "SQL", "FTP", "SFTP", "S3", "GCP"). If check_if_exist is enabled, the function will check if a connector with the same name already exists. If yes, it will return a message and the information of the existing connector instead of creating a new one.</i>
------------------	---

---

### Description

Create a new connector of a supported type (among: "SQL", "FTP", "SFTP", "S3", "GCP"). If check\_if\_exist is enabled, the function will check if a connector with the same name already exists. If yes, it will return a message and the information of the existing connector instead of creating a new one.

### Usage

```
create_connector(
  project_id,
  type,
  name,
  host,
  port,
  username,
  password,
  google_credentials = NULL,
  check_if_exist = FALSE
)
```

### Arguments

project_id	id of the project, can be obtained with get_projects().
type	connector type.
name	connector name.
host	connector host.
port	connector port.
username	connector username.
password	connector password.
google_credentials	google credentials JSON (for GCP only).
check_if_exist	boolean (FALSE by default). If TRUE, makes extra checks to see if a connector with the same name is already existing.

### Value

list - parsed content of the connector.

---

`create_dataframe_from_dataset`

*Create a dataframe from a dataset\_id.*

---

**Description**

Create a dataframe from a dataset\_id.

**Usage**

```
create_dataframe_from_dataset(dataset_id)
```

**Arguments**

`dataset_id`      dataset id.

**Value**

data.frame - a R dataframe matching the dataset.

---

`create_dataset_embedding`

*Create a dataset embedding from a dataset\_id.*

---

**Description**

Create a dataset embedding from a dataset\_id.

**Usage**

```
create_dataset_embedding(dataset_id)
```

**Arguments**

`dataset_id`      dataset id.

**Value**

integer - 200 on success.

create\_dataset\_from\_dataframe

*Upload dataset from data frame.*

---

### **Description**

Upload dataset from data frame.

### **Usage**

```
create_dataset_from_dataframe(project_id, dataset_name, dataframe, zip = FALSE)
```

### **Arguments**

`project_id` id of the project, can be obtained with `get_projects()`.  
`dataset_name` given name of the dataset on the platform.  
`dataframe` data.frame to upload.  
`zip` is the temp file zipped before sending it to Prevision.io (default = FALSE).

### **Value**

list - parsed content of the dataset.

---

create\_dataset\_from\_datasource

*Create a dataset from an existing datasource.*

---

### **Description**

Create a dataset from an existing datasource.

### **Usage**

```
create_dataset_from_datasource(project_id, dataset_name, datasource_id)
```

### **Arguments**

`project_id` id of the project, can be obtained with `get_projects()`.  
`dataset_name` given name of the dataset on the platform.  
`datasource_id` datasource id.

### **Value**

list - parsed content of the dataset.

---

`create_dataset_from_file`*Upload dataset from file name.*

---

**Description**

Upload dataset from file name.

**Usage**

```
create_dataset_from_file(  
    project_id,  
    dataset_name,  
    file,  
    separator = ",",  
    decimal = "."  
)
```

**Arguments**

<code>project_id</code>	id of the project, can be obtained with <code>get_projects()</code> .
<code>dataset_name</code>	given name of the dataset on the platform.
<code>file</code>	path to the dataset.
<code>separator</code>	column separator in the file (default: ",")
<code>decimal</code>	decimal separator in the file (default: ".")

**Value**

list - parsed content of the dataset.

---

`create_datasource`*Create a new datasource If check\_if\_exist is enabled, the function will check if a datasource with the same name already exists. If yes, it will return a message and the information of the existing datasource instead of creating a new one.*

---

**Description**

Create a new datasource If `check_if_exist` is enabled, the function will check if a datasource with the same name already exists. If yes, it will return a message and the information of the existing datasource instead of creating a new one.

**Usage**

```

create_datasource(
    project_id,
    connector_id,
    name,
    path = "",
    database = "",
    table = "",
    bucket = "",
    request = "",
    check_if_exist = FALSE
)

```

**Arguments**

project_id	id of the project, can be obtained with get_projects().
connector_id	connector_id linked to the datasource.
name	datasource name.
path	datasource path (for SFTP & FTP connector).
database	datasource database (for SQL connector).
table	datasource table (for SQL connector).
bucket	datasource bucket (for S3 connector).
request	datasource request (for SQLconnector).
check_if_exist	boolean (FALSE by default). If TRUE, makes extra checks to see if a datasource with the same name is already existing.

**Value**

list - parsed content of the datasource.

---

create\_deployment\_api\_key

*Create a new API key for a deployed model.*

---

**Description**

Create a new API key for a deployed model.

**Usage**

```
create_deployment_api_key(deployment_id)
```

**Arguments**

deployment_id	id of the deployment to create an API key on, can be obtained with get_deployments().
---------------	---



**Value**

list - API key information.

---

```
create_deployment_model
```

*Create a new deployment for a model.*

---

**Description**

Create a new deployment for a model.

**Usage**

```
create_deployment_model(
  project_id,
  name,
  experiment_id,
  main_model_experiment_version_id,
  challenger_model_experiment_version_id = NULL,
  access_type = c("fine_grained", "private", "public"),
  description = NULL,
  main_model_id,
  challenger_model_id = NULL
)
```

**Arguments**

project_id	id of the project, can be obtained with <code>get_projects()</code> .
name	name of the deployment.
experiment_id	id of the experiment to deploy, can be obtained with <code>get_experiment_id_from_name()</code> .
main_model_experiment_version_id	id of the experiment_version to deploy, can be obtained with <code>get_experiment_version_id()</code> .
challenger_model_experiment_version_id	id of the challenger experiment_version to deploy, can be obtained with <code>get_experiment_version_id()</code> .
access_type	type of access of the deployment among "fine_grained" (project defined, default), "private" (instance) or "public" (everyone).
description	description of the deployment.
main_model_id	id of the model to deploy
challenger_model_id	id of the challenger model to deploy

**Value**

list - parsed content of the deployment.

---

```
create_deployment_predictions
```

*Create predictions on a deployed model using a dataset.*

---

### Description

Create predictions on a deployed model using a dataset.

### Usage

```
create_deployment_predictions(deployment_id, dataset_id)
```

### Arguments

`deployment_id` id of the deployment, can be obtained with `get_deployments()`.

`dataset_id` id of the dataset to predict, can be obtained with `get_dataset_id_from_name()`.

### Value

integer - 200 on success.

---

```
create_experiment
```

*Create a new experiment. If `check_if_exist` is enabled, the function will check if an experiment with the same name already exists. If yes, it will return a message and the information of the existing experiment instead of creating a new one.*

---

### Description

Create a new experiment. If `check_if_exist` is enabled, the function will check if an experiment with the same name already exists. If yes, it will return a message and the information of the existing experiment instead of creating a new one.

### Usage

```
create_experiment(
    project_id,
    name,
    provider,
    data_type,
    training_type,
    check_if_exist = FALSE
)
```

**Arguments**

project_id	id of the project in which we create the experiment.
name	name of the experiment.
provider	provider of the experiment ("prevision-auto-ml" or "external")
data_type	type of data ("tabular", "images" or "timeseries").
training_type	type of the training you want to achieve ("regression", "classification", "multi-classification", "clustering", "object-detection" or "text-similarity").
check_if_exist	boolean (FALSE by default). If TRUE, makes extra checks to see if an experiment with the same name is already existing.

**Value**

list - experiment information.

---

create\_experiment\_version

*Create a new version of an existing experiment.*

---

**Description**

Create a new version of an existing experiment.

**Usage**

```
create_experiment_version(
  experiment_id,
  dataset_id = NULL,
  target_column = NULL,
  holdout_dataset_id = NULL,
  id_column = NULL,
  drop_list = NULL,
  profile = NULL,
  experiment_description = NULL,
  metric = NULL,
  fold_column = NULL,
  normal_models = NULL,
  lite_models = NULL,
  simple_models = NULL,
  with_blend = NULL,
  weight_column = NULL,
  features_engineering_selected_list = NULL,
  features_selection_count = NULL,
  features_selection_time = NULL,
  folder_dataset_id = NULL,
  filename_column = NULL,
```

```

ymin = NULL,
ymax = NULL,
xmin = NULL,
xmax = NULL,
time_column = NULL,
start_dw = NULL,
end_dw = NULL,
start_fw = NULL,
end_fw = NULL,
group_list = NULL,
apriori_list = NULL,
content_column = NULL,
queries_dataset_id = NULL,
queries_dataset_content_column = NULL,
queries_dataset_id_column = NULL,
queries_dataset_matching_id_description_column = NULL,
top_k = NULL,
lang = NULL,
models_params = NULL,
name = NULL,
onnx_file = NULL,
yaml_file = NULL
)

```

### Arguments

experiment_id	id of the experiment that will host the new version.
dataset_id	id of the dataset used for the training phase.
target_column	name of the TARGET column.
holdout_dataset_id	id of the holdout dataset.
id_column	name of the id column.
drop_list	list of names of features to drop.
profile	chosen profil among "quick", "normal", "advanced".
experiment_description	experiment description.
metric	name of the metric to optimise.
fold_column	name of the fold column.
normal_models	list of (normal) models to select with full FE & hyperparameters search (among "LR", "RF", "ET", "XGB", "LGB", "NN", "CB").
lite_models	list of (lite) models to select with lite FE & default hyperparameters (among "LR", "RF", "ET", "XGB", "LGB", "NN", "CB", "NBC").
simple_models	list of simple models to select (among "LR", "DT").
with_blend	boolean, do we allow to include blend in the modelisation.
weight_column	name of the weight columns.

features_engineering_selected_list	list of feature engineering to select (among "Counter", "Date", "freq", "text_tfidf", "text_word2vec", "text_embedding", "tenc", "poly", "pca", "kmean").
features_selection_count	number of features to keep after the feature selection process.
features_selection_time	time budget in minutes of the feature selection process.
folder_dataset_id	id of the dataset folder (images).
filename_column	name of the file name path (images).
ymin	name of the column matching the lower y value of the image (object detection).
ymax	name of the column matching the higher y value of the image (object detection).
xmin	name of the column matching the lower x value of the image (object detection).
xmax	name of the column matching the higher x value of the image (object detection).
time_column	name of column containing the timestamp (time series).
start_dw	value of the start of derivative window (time series), should be a strict negative integer.
end_dw	value of the end of derivative window (time series), should be a negative integer greater than start_dw.
start_fw	value of the start of forecast window (time series), should be a strict positive integer.
end_fw	value of the end of forecast window (time series), should be a strict positive integer greater than start_fw.
group_list	list of name of feature that describes groups (time series).
apriori_list	list of name of feature that are a priori (time series).
content_column	content column name (text-similarity).
queries_dataset_id	id of the dataset containing queries (text-similarity).
queries_dataset_content_column	name of the column containing queries in the query dataset (text-similarity).
queries_dataset_id_column	name of the ID column in the query dataset (text-similarity).
queries_dataset_matching_id_description_column	name of the column matching id in the description dataset (text-similarity).
top_k	top k individual to find (text-similarity).
lang	lang of the text (text-similarity).
models_params	parameters of the model (text-similarity).
name	name of the external model (external model).
onnx_file	path to the onnx file (external model).
yaml_file	path to the yaml file (external model).

**Value**

list - experiment information.

---

create_export	<i>Export data using an existing exporter and the resource to export</i>
---------------	--

---

**Description**

Export data using an existing exporter and the resource to export

**Usage**

```
create_export(exporter_id, type, dataset_id = NULL, prediction_id = NULL)
```

**Arguments**

exporter_id	id of the exporter, can be obtained with get_exporters().
type	type of data to export among \"dataset\", \"validation-prediction\" or \"deployment-prediction\"
dataset_id	id of the dataset to export (only for type == \"dataset\")
prediction_id	id of the prediction to export (only for type == \"validation_prediction\" or type == \"deployment-prediction\")

**Value**

list - parsed content of the export.

---

create_exporter	<i>Create a new exporter</i>
-----------------	------------------------------

---

**Description**

Create a new exporter

**Usage**

```
create_exporter(
  project_id,
  connector_id,
  name,
  description = "",
  filepath = "",
  file_write_mode = "timestamp",
  database = "",
  table = "",
  database_write_mode = "append",
  bucket = ""
)
```

**Arguments**

project_id	id of the project, can be obtained with get_projects().
connector_id	connector_id linked to the exporter.
name	exporter name.
description	description of the exporter.
filepath	exporter path (for SFTP & FTP connector).
file_write_mode	writing type when exporting a file (for SFT & FTP connector, among \"timestamp\", \"safe\" or \"replace\")
database	exporter database (for SQL connector).
table	exporter table (for SQL connector).
database_write_mode	writing type when exporting data within a database (for SQL connector, among \"append\" or \"replace\").
bucket	exporter bucket (for S3 connector).

**Value**

list - parsed content of the exporter.

---

create_folder	<i>Upload folder from a local file.</i>
---------------	---

---

**Description**

Upload folder from a local file.

**Usage**

```
create_folder(project_id, folder_name, file)
```

**Arguments**

project_id	id of the project, can be obtained with get_projects().
folder_name	given name of the folder on the platform.
file	path to the folder.

**Value**

list - parsed content of the folder.

---

`create_pipeline_trigger`*Trigger an existing pipeline run.*

---

**Description**

Trigger an existing pipeline run.

**Usage**

```
create_pipeline_trigger(pipeline_id)
```

**Arguments**

`pipeline_id` id of the pipeline run to trigger, can be obtained with `get_pipelines()`.

**Value**

integer - 200 on success.

---

`create_prediction`*Create a prediction on a specified experiment\_version*

---

**Description**

Create a prediction on a specified `experiment_version`

**Usage**

```
create_prediction(  
  experiment_version_id,  
  dataset_id = NULL,  
  folder_dataset_id = NULL,  
  confidence = FALSE,  
  best_single = FALSE,  
  model_id = NULL,  
  queries_dataset_id = NULL,  
  queries_dataset_content_column = NULL,  
  queries_dataset_id_column = NULL,  
  queries_dataset_matching_id_description_column = NULL,  
  top_k = NULL  
)
```



**Arguments**

experiment_version_id	id of the experiment_version, can be obtained with get_experiment_version_id().
dataset_id	id of the dataset to start the prediction on, can be obtained with get_datasets().
folder_dataset_id	id of the folder dataset to start prediction on, can be obtained with get_folders(). Only usefull for images use cases.
confidence	boolean. If enable, confidence interval will be added to predictions.
best_single	boolean. If enable, best single model (non blend) will be used for making predictions other wise, best model will be used unless if model_id is fed.
model_id	id of the model to start the prediction on. If provided, it will overwrite the "best single" params.
queries_dataset_id	id of the dataset containing queries (text-similarity).
queries_dataset_content_column	name of the content column in the queries dataset (text-similarity).
queries_dataset_id_column	name of the id column in the queries dataset (text-similarity).
queries_dataset_matching_id_description_column	name of the column matching the id (text-similarity).
top_k	number of class to retrieve (text-similarity).

**Value**

list - parsed prediction information.

---

create_project	<i>Create a new project. If check_if_exist is enabled, the function will check if a project with the same name already exists. If yes, it will return a message and the information of the existing project instead of creating a new one.</i>
----------------	--

---

**Description**

Create a new project. If check\_if\_exist is enabled, the function will check if a project with the same name already exists. If yes, it will return a message and the information of the existing project instead of creating a new one.

**Usage**

```
create_project(
    name,
    description = NULL,
    color = "#a748f5",
    check_if_exist = FALSE
)
```

**Arguments**

name	name of the project.
description	description of the project.
color	color of the project among <code>\"#4876be\"</code> , <code>\"#4ab6eb\"</code> , <code>\"#49cf7d\"</code> , <code>\"#dc8218\"</code> , <code>\"#ecba35\"</code> , <code>\"#f45b69\"</code> , <code>\"#a748f5\"</code> , <code>\"#b34ca2\"</code> or <code>\"#2fe6d0\"</code> ( <code>\"#a748f5\"</code> by default).
check_if_exist	boolean (FALSE by default). If TRUE, makes extra checks to see if a project with the same name is already existing.

**Value**

list - information of the created project.

---

`create_project_user`    *Add user in and existing project.*

---

**Description**

Add user in and existing project.

**Usage**

```
create_project_user(project_id, user_mail, user_role)
```

**Arguments**

project_id	id of the project, can be obtained with <code>get_projects()</code> .
user_mail	email of the user to be add.
user_role	role to grand to the user among "admin", "contributor", "viewer" or "end_user".

**Value**

list - information of project's users.

---

`delete_connector`      *Delete an existing connector.*

---

**Description**

Delete an existing connector.

**Usage**

`delete_connector(connector_id)`

**Arguments**

`connector_id`      id of the connector to be deleted, can be obtained with `get_connectors()`.

**Value**

integer - 200 on success.

---

`delete_dataset`      *Delete an existing dataset.*

---

**Description**

Delete an existing dataset.

**Usage**

`delete_dataset(dataset_id)`

**Arguments**

`dataset_id`      id of the dataset, can be obtained with `get_datasets()`.

**Value**

integer - 204 on success.

---

delete\_datasource      *Delete a datasource*

---

**Description**

Delete a datasource

**Usage**

```
delete_datasource(datasource_id)
```

**Arguments**

datasource\_id    id of the datasource to be deleted, can be obtained with get\_datasources().

**Value**

integer - 200 on success.

---

delete\_deployment      *Delete an existing deployment.*

---

**Description**

Delete an existing deployment.

**Usage**

```
delete_deployment(deployment_id)
```

**Arguments**

deployment\_id    id of the deployment, can be obtained with get\_deployments().

**Value**

integer - 204 on success.

---

`delete_experiment`      *Delete a experiment on the platform.*

---

**Description**

Delete a experiment on the platform.

**Usage**

`delete_experiment(experiment_id)`

**Arguments**

`experiment_id`    id of the experiment, can be obtained with `get_experiments()`.

**Value**

integer - 204 on success.

---

`delete_exporter`      *Delete an exporter*

---

**Description**

Delete an exporter

**Usage**

`delete_exporter(exporter_id)`

**Arguments**

`exporter_id`      id of the exporter to be deleted, can be obtained with `get_exporters()`.

**Value**

integer - 204 on success.

---

delete_folder	<i>Delete an existing folder.</i>
---------------	-----------------------------------

---

**Description**

Delete an existing folder.

**Usage**

```
delete_folder(folder_id)
```

**Arguments**

folder\_id      id of the folder to be deleted.

**Value**

integer - 200 on success.

---

delete_pipeline	<i>Delete an existing pipeline</i>
-----------------	------------------------------------

---

**Description**

Delete an existing pipeline

**Usage**

```
delete_pipeline(pipeline_id, type)
```

**Arguments**

pipeline\_id      id of the pipeline to be retrieved, can be obtained with get\_pipelines().  
type              type of the pipeline to be retrieved among "component", "template", "run".

**Value**

integer - 204 on success.

---

delete\_prediction      *Delete a prediction.*

---

**Description**

Delete a prediction.

**Usage**

```
delete_prediction(prediction_id)
```

**Arguments**

prediction\_id      id of the prediction to be deleted, can be obtained with get\_experiment\_version\_predictions().

**Value**

integer - 204 on success.

list of predictions of experiment\_id.

---

delete\_project      *Delete an existing project.*

---

**Description**

Delete an existing project.

**Usage**

```
delete_project(project_id)
```

**Arguments**

project\_id      id of the project, can be obtained with get\_projects().

**Value**

integer - 204 on success.

---

delete\_project\_user     *Delete user in and existing project.*

---

### Description

Delete user in and existing project.

### Usage

```
delete_project_user(project_id, user_id)
```

### Arguments

project\_id     id of the project, can be obtained with get\_projects().  
 user\_id        user\_id of the user to be delete, can be obtained with get\_project\_users().

### Value

integer - 200 on success.

---

get\_best\_model\_id     *Get the model\_id that provide the best predictive performance given experiment\_version\_id. If include\_blend is false, it will return the model\_id from the best "non blended" model.*

---

### Description

Get the model\_id that provide the best predictive performance given experiment\_version\_id. If include\_blend is false, it will return the model\_id from the best "non blended" model.

### Usage

```
get_best_model_id(experiment_version_id, include_blend = TRUE)
```

### Arguments

experiment\_version\_id     id of the experiment\_version, can be obtained with get\_experiment\_version\_id().  
 include\_blend     boolean, indicating if you want to retrieve the best model among blended models too.

### Value

character - model\_id.



---

get_connectors	<i>Get information of all connectors available for a given project_id.</i>
----------------	--

---

**Description**

Get information of all connectors available for a given project\_id.

**Usage**

```
get_connectors(project_id)
```

**Arguments**

project\_id      id of the project, can be obtained with get\_projects().

**Value**

list - parsed content of all connectors for the supplied project\_id.

---

get_connector_id_from_name	<i>Get a connector_id from a connector_name for a given project_id. If duplicated name, the first connector_id that match it is retrieved.</i>
----------------------------	--

---

**Description**

Get a connector\_id from a connector\_name for a given project\_id. If duplicated name, the first connector\_id that match it is retrieved.

**Usage**

```
get_connector_id_from_name(project_id, connector_name)
```

**Arguments**

project\_id      id of the project, can be obtained with get\_projects(project\_id).  
connector\_name   name of the connector we are searching its id from.

**Value**

character - id of the connector if found.

---

get\_connector\_info      *Get information about connector from its id.*

---

**Description**

Get information about connector from its id.

**Usage**

```
get_connector_info(connector_id)
```

**Arguments**

connector\_id      id of the connector to be retrieved, can be obtained with get\_connectors().

**Value**

list - parsed content of the connector.

---

get\_datasets              *Get information of all datasets available for a given project\_id.*

---

**Description**

Get information of all datasets available for a given project\_id.

**Usage**

```
get_datasets(project_id)
```

**Arguments**

project\_id      id of the project, can be obtained with get\_projects().

**Value**

list - parsed content of all datasets for the supplied project\_id.

---

`get_dataset_embedding` *Get a dataset embedding from a dataset\_id.*

---

**Description**

Get a dataset embedding from a `dataset_id`.

**Usage**

```
get_dataset_embedding(dataset_id)
```

**Arguments**

`dataset_id`      dataset id.

**Value**

integer - 200 on success.

---

`get_dataset_head`      *Show the head of a dataset from its id.*

---

**Description**

Show the head of a dataset from its id.

**Usage**

```
get_dataset_head(dataset_id)
```

**Arguments**

`dataset_id`      id of the dataset, can be obtained with `get_datasets()`.

**Value**

data.frame - head of the dataset.

---

get\_dataset\_id\_from\_name

*Get a dataset\_id from a dataset\_name. If duplicated name, the first dataset\_id that match it is retrieved.*

---

### **Description**

Get a dataset\_id from a dataset\_name. If duplicated name, the first dataset\_id that match it is retrieved.

### **Usage**

```
get_dataset_id_from_name(project_id, dataset_name)
```

### **Arguments**

project\_id      id of the project, can be obtained with get\_projects().

dataset\_name    name of the dataset we are searching its id from. Can be obtained with get\_datasets().

### **Value**

character - id of the dataset if found.

---

get\_dataset\_info      *Get a dataset from its id.*

---

### **Description**

Get a dataset from its id.

### **Usage**

```
get_dataset_info(dataset_id)
```

### **Arguments**

dataset\_id      id of the dataset, can be obtained with get\_datasets().

### **Value**

list - parsed content of the dataset.

---

get_datasources	<i>Get information of all data sources available for a given project_id.</i>
-----------------	--

---

**Description**

Get information of all data sources available for a given project\_id.

**Usage**

```
get_datasources(project_id)
```

**Arguments**

project\_id      id of the project, can be obtained with get\_projects().

**Value**

list - parsed content of all data\_sources for the supplied project\_id.

---

get_datasource_id_from_name	<i>Get a datasource_id from a datasource_name If duplicated name, the first datasource_id that match it is retrieved</i>
-----------------------------	--

---

**Description**

Get a datasource\_id from a datasource\_name If duplicated name, the first datasource\_id that match it is retrieved

**Usage**

```
get_datasource_id_from_name(project_id, datasource_name)
```

**Arguments**

project\_id      id of the project, can be obtained with get\_projects().  
datasource\_name      name of the datasource we are searching its id from. Can be obtained with get\_datasources().

**Value**

character - id of the datasource if found.

---

get\_datasource\_info    *Get a datasource from its id.*

---

**Description**

Get a datasource from its id.

**Usage**

```
get_datasource_info(datasource_id)
```

**Arguments**

datasource\_id    id of the data\_sources to be retrieved, can be obtained with get\_datasources().

**Value**

list - parsed content of the data\_sources.

---

get\_deployments    *Get information of all deployments of a given type available for a given project\_id.*

---

**Description**

Get information of all deployments of a given type available for a given project\_id.

**Usage**

```
get_deployments(project_id, type)
```

**Arguments**

project\_id    id of the project, can be obtained with get\_projects().

type    type of the deployment to retrieve among "model" or "app".

**Value**

list - parsed content of all deployments of the given type for the supplied project\_id.

---

get\_deployment\_api\_keys

*Get API keys for a deployed model.*

---

**Description**

Get API keys for a deployed model.

**Usage**

```
get_deployment_api_keys(deployment_id)
```

**Arguments**

deployment\_id id of the deployment to get API keys, can be obtained with get\_deployments().

**Value**

data.frame - API keys available for deployment\_id.

---

get\_deployment\_app\_logs

*Get logs from a deployed app.*

---

**Description**

Get logs from a deployed app.

**Usage**

```
get_deployment_app_logs(deployment_id, log_type)
```

**Arguments**

deployment\_id id of the deployment to get the log, can be obtained with get\_deployments().

log\_type type of logs we want to get among "build", "deploy" or "run".

**Value**

list - logs from deployed apps.

---

get\_deployment\_id\_from\_name

*Get a deployment\_id from a name and type for a given project\_id. If duplicated name, the first deployment\_id that match it is retrieved.*

---

### **Description**

Get a deployment\_id from a name and type for a given project\_id. If duplicated name, the first deployment\_id that match it is retrieved.

### **Usage**

```
get_deployment_id_from_name(project_id, name, type)
```

### **Arguments**

project_id	id of the project, can be obtained with get_projects().
name	name of the deployment we are searching its id from.
type	type of the deployment to be retrieved among "model" or "app".

### **Value**

character - id of the connector if found.

---

get\_deployment\_info *Get information about a deployment from its id.*

---

### **Description**

Get information about a deployment from its id.

### **Usage**

```
get_deployment_info(deployment_id)
```

### **Arguments**

deployment_id	id of the deployment to be retrieved, can be obtained with get_deployments().
---------------	---

### **Value**

list - parsed content of the deployment.



---

`get_deployment_predictions`

*Get listing of predictions related to a deployment\_id.*

---

**Description**

Get listing of predictions related to a deployment\_id.

**Usage**

`get_deployment_predictions(deployment_id)`

**Arguments**

`deployment_id` id of the deployment, can be obtained with `get_deployments()`.

**Value**

list - predictions available for a deployed model.

---

`get_deployment_prediction_info`

*Get information related to predictions of a prediction\_id.*

---

**Description**

Get information related to predictions of a prediction\_id.

**Usage**

`get_deployment_prediction_info(prediction_id)`

**Arguments**

`prediction_id` id of the prediction returned by `create_deployment_predictions` or that can be obtained with `get_deployment_predictions()`.

**Value**

list - prediction information for a deployed model.

---

`get_deployment_usage` *Get usage (calls, errors and response time) of the last version of a deployed model.*

---

**Description**

Get usage (calls, errors and response time) of the last version of a deployed model.

**Usage**

```
get_deployment_usage(deployment_id, usage_type)
```

**Arguments**

`deployment_id` id of the deployment to get usage, can be obtained with `get_deployments()`.  
`usage_type` type of usage to get, among "calls", "errors", "response\_time".

**Value**

list - plotly object.

---

`get_experiments` *Get information of all experiments available for a given project\_id.*

---

**Description**

Get information of all experiments available for a given `project_id`.

**Usage**

```
get_experiments(project_id)
```

**Arguments**

`project_id` id of the project, can be obtained with `get_projects()`.

**Value**

list - parsed content of all experiments for the supplied `project_id`.

---

get\_experiment\_id\_from\_name

*Get a experiment\_id from a experiment\_name If duplicated name, the first experiment\_id that match it is retrieved.*

---

### **Description**

Get a experiment\_id from a experiment\_name If duplicated name, the first experiment\_id that match it is retrieved.

### **Usage**

```
get_experiment_id_from_name(project_id, experiment_name)
```

### **Arguments**

project\_id      id of the project, can be obtained with get\_projects().

experiment\_name      name of the experiment we are searching its id from. Can be obtained with get\_experiments().

### **Value**

character - id matching the experiment\_name if found.

---

get\_experiment\_info      *Get a experiment from its experiment\_id.*

---

### **Description**

Get a experiment from its experiment\_id.

### **Usage**

```
get_experiment_info(experiment_id)
```

### **Arguments**

experiment\_id      id of the experiment, can be obtained with get\_experiments().

### **Value**

list - parsed content of the experiment.

---

`get_experiment_version_features`*Get features information related to a experiment\_version\_id.*

---

**Description**

Get features information related to a experiment\_version\_id.

**Usage**

```
get_experiment_version_features(experiment_version_id)
```

**Arguments**

experiment\_version\_id

id of the experiment\_version, can be obtained with get\_experiment\_version\_id().

**Value**

list - parsed content of the experiment\_version features information.

---

`get_experiment_version_id`*Get a experiment version id from a experiment\_id and its version number.*

---

**Description**

Get a experiment version id from a experiment\_id and its version number.

**Usage**

```
get_experiment_version_id(experiment_id, version_number = 1)
```

**Arguments**

experiment\_id id of the experiment, can be obtained with get\_experiments().

version\_number number of the version of the experiment. 1 by default

**Value**

character - experiment version id.

---

get\_experiment\_version\_info

*Get a experiment\_version info from its experiment\_version\_id*

---

**Description**

Get a experiment\_version info from its experiment\_version\_id

**Usage**

get\_experiment\_version\_info(experiment\_version\_id)

**Arguments**

experiment\_version\_id

id of the experiment\_version, can be obtained with get\_experiment\_version\_id().

**Value**

list - parsed content of the experiment\_version.

---

get\_experiment\_version\_models

*Get a model list related to a experiment\_version\_id.*

---

**Description**

Get a model list related to a experiment\_version\_id.

**Usage**

get\_experiment\_version\_models(experiment\_version\_id)

**Arguments**

experiment\_version\_id

id of the experiment\_version, can be obtained with get\_experiment\_version\_id().

**Value**

list - parsed content of models attached to experiment\_version\_id.

---

`get_experiment_version_predictions`*Get a list of prediction from a experiment\_version\_id.*

---

**Description**

Get a list of prediction from a experiment\_version\_id.

**Usage**

```
get_experiment_version_predictions(  
    experiment_version_id,  
    generating_type = "user"  
)
```

**Arguments**

experiment\_version\_id

id of the experiment\_version, can be obtained with get\_experiment\_version\_id().

generating\_type

can be "user" (= user predictions) or "auto" (= hold out predictions).

**Value**

list - parsed prediction list items.

---

`get_exporters`*Get information of all exporters available for a given project\_id.*

---

**Description**

Get information of all exporters available for a given project\_id.

**Usage**

```
get_exporters(project_id)
```

**Arguments**

project\_id

id of the project, can be obtained with get\_projects().

**Value**

list - parsed content of all exporters for the supplied project\_id.

---

get\_exporter\_exports    *Get all exports done from an exporter\_id*

---

**Description**

Get all exports done from an exporter\_id

**Usage**

```
get_exporter_exports(exporter_id)
```

**Arguments**

exporter\_id    id of the exporter to retrieve information, can be obtained with get\_exporters().

**Value**

list - list of exports of the supplied exporter\_id.

---

get\_exporter\_id\_from\_name

*Get a exporter\_id from a exporter\_name. If duplicated name, the first exporter\_id that match it is retrieved*

---

**Description**

Get a exporter\_id from a exporter\_name. If duplicated name, the first exporter\_id that match it is retrieved

**Usage**

```
get_exporter_id_from_name(project_id, exporter_name)
```

**Arguments**

project\_id    id of the project, can be obtained with get\_projects().

exporter\_name    name of the exporter we are searching its id from. Can be obtained with get\_exporters().

**Value**

character - id of the exporter if found.

---

get\_exporter\_info      *Get an exporter from its id.*

---

**Description**

Get an exporter from its id.

**Usage**

```
get_exporter_info(exporter_id)
```

**Arguments**

exporter\_id      id of the exporter to be retrieved, can be obtained with get\_exporters().

**Value**

list - parsed content of the exporter.

---

get\_features\_infos      *Get information of a given feature related to a experiment\_version\_id.*

---

**Description**

Get information of a given feature related to a experiment\_version\_id.

**Usage**

```
get_features_infos(experiment_version_id, feature_name)
```

**Arguments**

experiment\_version\_id      id of the experiment\_version, can be obtained with get\_experiment\_version\_id().

feature\_name      name of the feature to retrieve information.

**Value**

list - parsed content of the specific feature.



---

get_folder	<i>Get a folder from its id.</i>
------------	----------------------------------

---

**Description**

Get a folder from its id.

**Usage**

```
get_folder(folder_id)
```

**Arguments**

folder\_id      id of the image folder, can be obtained with get\_folders().

**Value**

list - parsed content of the folder.

---

get_folders	<i>Get information of all image folders available for a given project_id.</i>
-------------	---

---

**Description**

Get information of all image folders available for a given project\_id.

**Usage**

```
get_folders(project_id)
```

**Arguments**

project\_id      id of the project, can be obtained with get\_projects().

**Value**

list - parsed content of all folders.

---

get\_folder\_id\_from\_name

*Get a folder\_id from a folder\_name. If duplicated name, the first folder\_id that match it is retrieved.*

---

### **Description**

Get a folder\_id from a folder\_name. If duplicated name, the first folder\_id that match it is retrieved.

### **Usage**

```
get_folder_id_from_name(project_id, folder_name)
```

### **Arguments**

project\_id      id of the project, can be obtained with get\_projects().  
folder\_name     name of the folder we are searching its id from. Can be obtained with get\_folders().

### **Value**

character - id of the folder if found.

---

get\_model\_cv

*Get the cross validation file from a specific model.*

---

### **Description**

Get the cross validation file from a specific model.

### **Usage**

```
get_model_cv(model_id)
```

### **Arguments**

model\_id        id of the model to get the CV, can be obtained with get\_experiment\_version\_models().

### **Value**

data.frame - cross validation data coming from model\_id.

---

`get_model_feature_importance`*Get feature importance corresponding to a model\_id.*

---

**Description**

Get feature importance corresponding to a model\_id.

**Usage**

```
get_model_feature_importance(model_id, mode = "raw")
```

**Arguments**

model_id	id of the model, can be obtained with <code>get_experiment_models()</code> .
mode	character indicating the type of feature importance among "raw" (default) or "engineered".

**Value**

data.frame - dataset of the model's feature importance.

---

`get_model_hyperparameters`*Get hyperparameters corresponding to a model\_id.*

---

**Description**

Get hyperparameters corresponding to a model\_id.

**Usage**

```
get_model_hyperparameters(model_id)
```

**Arguments**

model_id	id of the model, can be obtained with <code>experimentModels(experiment_id)</code> .
----------	--

**Value**

list - parsed content of the model's hyperparameters.

---

get_model_infos	<i>Get model information corresponding to a model_id.</i>
-----------------	---

---

**Description**

Get model information corresponding to a model\_id.

**Usage**

```
get_model_infos(model_id)
```

**Arguments**

model\_id          id of the model, can be obtained with get\_experiment\_models().

**Value**

list - parsed content of the model.

---

get_pipelines	<i>Get information of all pipelines of a given type available for a given project_id.</i>
---------------	---

---

**Description**

Get information of all pipelines of a given type available for a given project\_id.

**Usage**

```
get_pipelines(project_id, type)
```

**Arguments**

project\_id      id of the project, can be obtained with get\_projects().  
type            type of the pipeline to retrieve among "component", "template", or "run".

**Value**

list - parsed content of all pipelines of the given type for the supplied project\_id.

---

get\_pipeline\_id\_from\_name

*Get a pipeline\_id from a pipeline\_name and type for a given project\_id. If duplicated name, the first pipeline\_id that match it is retrieved.*

---

### Description

Get a pipeline\_id from a pipeline\_name and type for a given project\_id. If duplicated name, the first pipeline\_id that match it is retrieved.

### Usage

```
get_pipeline_id_from_name(project_id, name, type)
```

### Arguments

project_id	id of the project, can be obtained with get_projects().
name	name of the pipeline we are searching its id from.
type	type of the pipeline to be retrieved among "component", "template", "run".

### Value

character - id of the connector if found.

---

get\_pipeline\_info      *Get information about a pipeline from its id and its type.*

---

### Description

Get information about a pipeline from its id and its type.

### Usage

```
get_pipeline_info(pipeline_id, type)
```

### Arguments

pipeline_id	id of the pipeline to be retrieved, can be obtained with get_pipelines().
type	type of the pipeline to be retrieved among "component", "template", "run".

### Value

list - parsed content of the pipeline.

---

get_prediction	<i>Get a specific prediction from a prediction_id. Wait up until time_out is reached and wait wait_time between each retry.</i>
----------------	---

---

**Description**

Get a specific prediction from a prediction\_id. Wait up until time\_out is reached and wait wait\_time between each retry.

**Usage**

```
get_prediction(prediction_id, prediction_type, time_out = 3600, wait_time = 10)
```

**Arguments**

prediction_id	id of the prediction to be retrieved, can be obtained with get_experiment_version_predictions().
prediction_type	type of prediction among "validation" (not deployed model) and "deployment" (deployed model).
time_out	maximum number of seconds to wait for the prediction. 3 600 by default.
wait_time	number of seconds to wait between each retry. 10 by default.

**Value**

data.frame - predictions coming from prediction\_id.

---

get_prediction_infos	<i>Get a information about a prediction_id.</i>
----------------------	---

---

**Description**

Get a information about a prediction\_id.

**Usage**

```
get_prediction_infos(prediction_id)
```

**Arguments**

prediction_id	id of the prediction to be retrieved, can be obtained with get_experiment_version_predictions().
---------------	--

**Value**

list - parsed prediction information.

---

get_projects	<i>Retrieves all projects.</i>
--------------	--------------------------------

---

**Description**

Retrieves all projects.

**Usage**

```
get_projects()
```

**Value**

list - list of existing projects.

---

get\_project\_id\_from\_name

*Get a project\_id from a project\_name If duplicated name, the first project\_id that match it is retrieved.*

---

**Description**

Get a project\_id from a project\_name If duplicated name, the first project\_id that match it is retrieved.

**Usage**

```
get_project_id_from_name(project_name)
```

**Arguments**

project\_name     name of the project we are searching its id from. Can be obtained with get\_projects().

**Value**

character - project\_id of the project\_name if found.

---

get\_project\_info      *Get a project from its project\_id.*

---

**Description**

Get a project from its project\_id.

**Usage**

```
get_project_info(project_id)
```

**Arguments**

project\_id      id of the project, can be obtained with get\_projects().

**Value**

list - information of the project.

---

get\_project\_users      *Get users from a project.*

---

**Description**

Get users from a project.

**Usage**

```
get_project_users(project_id)
```

**Arguments**

project\_id      id of the project, can be obtained with get\_projects().

**Value**

list - information of project's users.



---

 helper\_cv\_classif\_analysis

*Get metrics on a CV file retrieved by Prevision.io for a binary classification use case*

---

### Description

Get metrics on a CV file retrieved by Prevision.io for a binary classification use case

### Usage

```
helper_cv_classif_analysis(actual, predicted, fold, thresh = NULL, step = 1000)
```

### Arguments

actual	target coming from the cross Validation dataframe retrieved by Prevision.io
predicted	prediction coming from the cross Validation dataframe retrieved by Prevision.io
fold	fold number coming from the cross Validation dataframe retrieved by Prevision.io
thresh	threshold to use. If not provided optimal threshold given F1 score will be computed
step	number of iteration done to find optimal thresh (1000 by default = 0.1% resolution per fold)

### Value

data.frame - metrics computed between actual and predicted vectors.

---

 helper\_drift\_analysis *[BETA] Return a data.frame that contains features, a boolean indicating if the feature may have a different distribution between the submitted datasets (if p-value < threshold), their exact p-value and the test used to compute it.*


---

### Description

[BETA] Return a data.frame that contains features, a boolean indicating if the feature may have a different distribution between the submitted datasets (if p-value < threshold), their exact p-value and the test used to compute it.

### Usage

```
helper_drift_analysis(dataset_1, dataset_2, p_value = 0.05, features = NULL)
```

**Arguments**

dataset_1	the first data set
dataset_2	the second data set
p_value	a p-value that will be the decision criteria for deciding if a feature is suspicious 5% by default
features	a vector of features names that should be tested. If NULL, only the intersection of the names() will be kept

**Value**

vector - a vector of suspicious features.

---

helper\_optimal\_prediction

*[BETA] Compute the optimal prediction for each rows in a data frame, for a given model, a list of actionable features and a number of samples for each features to be tested.*

---

**Description**

[BETA] Compute the optimal prediction for each rows in a data frame, for a given model, a list of actionable features and a number of samples for each features to be tested.

**Usage**

```
helper_optimal_prediction(
  project_id,
  experiment_id,
  model_id,
  df,
  actionable_features,
  nb_sample,
  maximize,
  zip = FALSE,
  version = 1
)
```

**Arguments**

project_id	id of the project containing the use case.
experiment_id	id of the experiment to be predicted on.
model_id	id of the model to be predicted on.
df	a data frame to be predicted on.
actionable_features	a list of actionable_features features contained in the names of the data frame.

nb_sample	a vector of number of sample for each actionable_features features.
maximize	a boolean indicating if we maximize or minimize the predicted target.
zip	a boolean indicating if the data frame to predict should be zipped prior sending to the instance.
version	version of the use case we want to make the prediction on.

**Value**

data.frame - optimal vector and the prediction associated with for each rows in the original data frame.

---

helper\_plot\_classif\_analysis

*Plot RECALL, PRECISION & F1 SCORE versus top n predictions for a binary classification use case*

---

**Description**

Plot RECALL, PRECISION & F1 SCORE versus top n predictions for a binary classification use case

**Usage**

```
helper_plot_classif_analysis(actual, predicted, top, compute_every_n = 1)
```

**Arguments**

actual	true value (0 or 1 only)
predicted	prediction vector (probability)
top	top individual to analyse
compute_every_n	compute indicators every n individuals (1 by default)

**Value**

data.frame - metrics computed between actual and predicted vectors.

---

pause\_experiment\_version

*Pause a running experiment\_version on the platform.*

---

### **Description**

Pause a running experiment\_version on the platform.

### **Usage**

```
pause_experiment_version(experiment_version_id)
```

### **Arguments**

experiment\_version\_id

id of the experiment\_version, can be obtained with get\_experiment\_version\_id().

### **Value**

integer - 200 on success.

---

pio\_download

*Download resources according specific parameters.*

---

### **Description**

Download resources according specific parameters.

### **Usage**

```
pio_download(endpoint, tempFile)
```

### **Arguments**

endpoint            end of the url of the API call.

tempFile            temporary file to download.

### **Value**

list - response from the request.

---

pio_init	<i>Initialization of the connection to your instance Prevision.io.</i>
----------	--

---

**Description**

Initialization of the connection to your instance Prevision.io.

**Usage**

```
pio_init(token, url)
```

**Arguments**

token	your master token, can be found on your instance on the "API KEY" page.
url	the url of your instance.

**Value**

list - url and token needed for connecting to the Prevision.io environment.

**Examples**

```
## Not run: pio_init('eyJhbGciOiJIUz', 'https://xxx.prevision.io')
```

---

pio_list_to_df	<i>Convert a list returned from APIs to a dataframe. Only working for consistent list (same naming and number of columns).</i>
----------------	--

---

**Description**

Convert a list returned from APIs to a dataframe. Only working for consistent list (same naming and number of columns).

**Usage**

```
pio_list_to_df(list)
```

**Arguments**

list	named list coming from an API call.
------	-------------------------------------

**Value**

data.frame - cast a consistent list to a data.frame.

---

pio_request	<i>Request the platform. Thanks to an endpoint, the url and the API, you can create request.</i>
-------------	--

---

**Description**

Request the platform. Thanks to an endpoint, the url and the API, you can create request.

**Usage**

```
pio_request(endpoint, method, data = NULL, upload = FALSE)
```

**Arguments**

endpoint	end of the url of the API call.
method	the method needed according the API (Available: POST, GET, DELETE).
data	object to upload when using method POST.
upload	used parameter when uploading dataset (for encoding in API call), don't use it.

**Value**

list - response from the request.

**Examples**

```
## Not run: pio_request(paste0('/jobs/', experiment$jobId), DELETE)
```

---

resume_experiment_version	<i>Resume a paused experiment_version on the platform.</i>
---------------------------	--

---

**Description**

Resume a paused experiment\_version on the platform.

**Usage**

```
resume_experiment_version(experiment_version_id)
```

**Arguments**

experiment_version_id	id of the experiment_version, can be obtained with get_experiment_version_id().
-----------------------	---

**Value**

integer - 200 on success.

---

`stop_experiment_version`

*Stop a running or paused experiment\_version on the platform.*

---

**Description**

Stop a running or paused experiment\_version on the platform.

**Usage**

```
stop_experiment_version(experiment_version_id)
```

**Arguments**

`experiment_version_id`

id of the experiment\_version, can be obtained with `get_experiment_version_id()`.

**Value**

integer - 200 on success.

---

`test_connector`

*Test an existing connector.*

---

**Description**

Test an existing connector.

**Usage**

```
test_connector(connector_id)
```

**Arguments**

`connector_id` id of the connector to be tested, can be obtained with `get_connectors()`.

**Value**

integer - 200 on success.

---

test\_datasource      *Test a datasource*

---

**Description**

Test a datasource

**Usage**

```
test_datasource(datasource_id)
```

**Arguments**

datasource\_id    id of the datasource to be tested, can be obtained with get\_datasources().

**Value**

integer - 200 on success.

---

test\_deployment\_type    *Check if a type of a deployment is supported*

---

**Description**

Check if a type of a deployment is supported

**Usage**

```
test_deployment_type(type)
```

**Arguments**

type              type of the deployment among "model" or "app".

**Value**

no return value, called for side effects.



---

test\_pipeline\_type      *Check if a type of a pipeline is supported*

---

**Description**

Check if a type of a pipeline is supported

**Usage**

```
test_pipeline_type(type)
```

**Arguments**

type                    type of the pipeline among "component", "template", "run".

**Value**

no return value, called for side effects.

---

update\_experiment\_version\_description  
*Update the description of a given experiment\_version\_id.*

---

**Description**

Update the description of a given experiment\_version\_id.

**Usage**

```
update_experiment_version_description(experiment_version_id, description = "")
```

**Arguments**

experiment\_version\_id  
                          id of the experiment\_version, can be obtained with get\_experiment\_version\_id().

description      Description of the experiment.

**Value**

integer - 200 on success.

---

update\_project\_user\_role

*Update user role in and existing project.*

---

**Description**

Update user role in and existing project.

**Usage**

```
update_project_user_role(project_id, user_id, user_role)
```

**Arguments**

project_id	id of the project, can be obtained with get_projects().
user_id	user_id of the user to be delete, can be obtained with get_project_users().
user_role	role to grand to the user among "admin", "contributor", "viewer" and "end_user".

**Value**

list - information of project's users.

# Index

[create\\_connector](#), 4  
[create\\_dataframe\\_from\\_dataset](#), 5  
[create\\_dataset\\_embedding](#), 5  
[create\\_dataset\\_from\\_dataframe](#), 6  
[create\\_dataset\\_from\\_datasource](#), 6  
[create\\_dataset\\_from\\_file](#), 7  
[create\\_datasource](#), 7  
[create\\_deployment\\_api\\_key](#), 8  
[create\\_deployment\\_model](#), 9  
[create\\_deployment\\_predictions](#), 10  
[create\\_experiment](#), 10  
[create\\_experiment\\_version](#), 11  
[create\\_export](#), 14  
[create\\_exporter](#), 14  
[create\\_folder](#), 15  
[create\\_pipeline\\_trigger](#), 16  
[create\\_prediction](#), 16  
[create\\_project](#), 17  
[create\\_project\\_user](#), 18

[delete\\_connector](#), 19  
[delete\\_dataset](#), 19  
[delete\\_datasource](#), 20  
[delete\\_deployment](#), 20  
[delete\\_experiment](#), 21  
[delete\\_exporter](#), 21  
[delete\\_folder](#), 22  
[delete\\_pipeline](#), 22  
[delete\\_prediction](#), 23  
[delete\\_project](#), 23  
[delete\\_project\\_user](#), 24

[get\\_best\\_model\\_id](#), 24  
[get\\_connector\\_id\\_from\\_name](#), 25  
[get\\_connector\\_info](#), 26  
[get\\_connectors](#), 25  
[get\\_dataset\\_embedding](#), 27  
[get\\_dataset\\_head](#), 27  
[get\\_dataset\\_id\\_from\\_name](#), 28  
[get\\_dataset\\_info](#), 28

[get\\_datasets](#), 26  
[get\\_datasource\\_id\\_from\\_name](#), 29  
[get\\_datasource\\_info](#), 30  
[get\\_datasources](#), 29  
[get\\_deployment\\_api\\_keys](#), 31  
[get\\_deployment\\_app\\_logs](#), 31  
[get\\_deployment\\_id\\_from\\_name](#), 32  
[get\\_deployment\\_info](#), 32  
[get\\_deployment\\_prediction\\_info](#), 33  
[get\\_deployment\\_predictions](#), 33  
[get\\_deployment\\_usage](#), 34  
[get\\_deployments](#), 30  
[get\\_experiment\\_id\\_from\\_name](#), 35  
[get\\_experiment\\_info](#), 35  
[get\\_experiment\\_version\\_features](#), 36  
[get\\_experiment\\_version\\_id](#), 36  
[get\\_experiment\\_version\\_info](#), 37  
[get\\_experiment\\_version\\_models](#), 37  
[get\\_experiment\\_version\\_predictions](#), 38  
[get\\_experiments](#), 34  
[get\\_exporter\\_exports](#), 39  
[get\\_exporter\\_id\\_from\\_name](#), 39  
[get\\_exporter\\_info](#), 40  
[get\\_exporters](#), 38  
[get\\_features\\_infos](#), 40  
[get\\_folder](#), 41  
[get\\_folder\\_id\\_from\\_name](#), 42  
[get\\_folders](#), 41  
[get\\_model\\_cv](#), 42  
[get\\_model\\_feature\\_importance](#), 43  
[get\\_model\\_hyperparameters](#), 43  
[get\\_model\\_infos](#), 44  
[get\\_pipeline\\_id\\_from\\_name](#), 45  
[get\\_pipeline\\_info](#), 45  
[get\\_pipelines](#), 44  
[get\\_prediction](#), 46  
[get\\_prediction\\_infos](#), 46  
[get\\_project\\_id\\_from\\_name](#), 47  
[get\\_project\\_info](#), 48

get\_project\_users, [48](#)  
get\_projects, [47](#)

helper\_cv\_classif\_analysis, [49](#)  
helper\_drift\_analysis, [49](#)  
helper\_optimal\_prediction, [50](#)  
helper\_plot\_classif\_analysis, [51](#)

pause\_experiment\_version, [52](#)  
pio\_download, [52](#)  
pio\_init, [53](#)  
pio\_list\_to\_df, [53](#)  
pio\_request, [54](#)

resume\_experiment\_version, [54](#)

stop\_experiment\_version, [55](#)

test\_connector, [55](#)  
test\_datasource, [56](#)  
test\_deployment\_type, [56](#)  
test\_pipeline\_type, [57](#)

update\_experiment\_version\_description,  
[57](#)  
update\_project\_user\_role, [58](#)