Gretel Client

Gretel.ai

Apr 02, 2024

CONTENTS:

CHAPTER

ONE

GETTING STARTED

The following command will install the latest stable Gretel CLI and Python SDK

```
pip install gretel-client
```

To install the latest development version, you may run

```
pip install git+https://github.com/gretelai/gretel-python-client@main
```

To configure the CLI, run

```
gretel configure
```

CHAPTER

TWO

SYSTEM REQUIREMENTS

The Gretel CLI and python SDKs require Python version 3.9 or greater. Docker is required for local training and generation jobs.

For more information please refer to the Gretel Environment Setup docs.

CHAPTER

THREE

CLIENT SDKS

The `gretel-client` package also ships with a set of Python Client SDKs that may be used to interact with Gretel APIs using a familiar pythonic interface. For more information on how to use these SDKs, please refer to the following links

- Projects SDK Reference

CHAPTER

FOUR

MODULES

4.1 Gretel CLI

The `gretel-client` package will automatically install the Gretel CLI. To ensure the CLI was installed correctly, you may run the following command from your terminal

```
gretel --help
```

For more information how to setup your CLI environment, see Environment Setup.

You may also refer to the CLI Tutorial docs for a list of guides detailing how to use the CLI.

4.2 High-level SDK Interface

The SDK's high-level interface is centered around the `Gretel` object, which serves as the main entry point for interacting with Gretel's APIs, models, and artifacts:

```
from gretel_client import Gretel

# connect to your Gretel account
gretel = Gretel(api_key="prompt")

# train a deep generative model from scratch and update the config via kwargs
trained = gretel.submit_train(
    base_config="tabular-actgan",
    data_source="https://gretel-public-website.s3-us-west-2.amazonaws.com/datasets/
↪USAdultIncome5k.csv",
    params={"epochs": 500},
)

# view synthetic data quality scores
print(trained.report)

# display the full report in your browser
trained.report.display_in_browser()

# fetch and inspect the synthetic data used in the report
df_report_synth = trained.fetch_report_synthetic_data()
print(df_report_synth.head())
```

(continues on next page)

(continued from previous page)

```
# generate synthetic data from a trained model
generated = gretel.submit_generate(trained.model_id, num_records=1000)

# inspect the synthetic data
print(generated.synthetic_data.head())
```

Module Reference

4.2.1 Interface

class gretel_client.gretel.interface.**Gretel**(*, *project_name: str | None = None*, *project_display_name: str | None = None*, *session: ClientConfig | None = None*, ***session_kwargs*)

High-level interface for interacting with Gretel's APIs.

To bound an instance of this class to a Gretel project, provide a project name at instantiation or use the *set_project* method. If a job is submitted (via a *submit_** method) without a project set, a randomly-named project will be created and set as the current project.

Parameters

- **project_name** (*str*) – Name of new or existing project. If a new project name is given, it will be created at instantiation. If no name given, a new randomly-named project will be created with the first job submission.
- **project_display_name** (*str*) – Project display name. If *None*, will use the project name. This argument is only used when creating a new project.
- **session** (ClientConfig) – Client session to use. If set, no `session_kwargs` may be specified.
- ****session_kwargs** – kwargs for your Gretel session. See options below.

Keyword Arguments

- **api_key** (*str*) – Your Gretel API key. If set to "prompt" and no API key is found on the system, you will be prompted for the key.
- **endpoint** (*str*) – Specifies the Gretel API endpoint. This must be a fully qualified URL. The default is "https://api.gretel.cloud".
- **default_runner** (*str*) – Specifies the runner mode. Must be one of "cloud", "local", "manual", or "hybrid". The default is "cloud".
- **artifact_endpoint** (*str*) – Specifies the endpoint for project and model artifacts. Defaults to "cloud" for running in Gretel Cloud. If working in hybrid mode, set to the URL of your artifact storage bucket.
- **cache** (*str*) – Valid options are "yes" or "no". If set to "no", the session configuration will not be written to disk. If set to "yes", the session configuration will be written to disk only if one doesn't already exist. The default is "no".
- **validate** (*bool*) – If *True*, will validate the login credentials at instantiation. The default is *False*.
- **clear** (*bool*) – If *True*, existing Gretel credentials will be removed. The default is *False.*

fetch_generate_job_results(*model_id: str*, *record_id: str*) → *GenerateJobResults*

Fetch the results object from a Gretel generate job.

Parameters

- **model_id** – The Gretel model ID.
- **record_id** – The Gretel record handler ID.

Raises

GretelProjectNotSetError – If a project has not been set.

Returns

Job results including the model object, record handler, and synthetic data.

fetch_model(*model_id: str*) → *Model*

Fetch a Gretel model using its ID.

You must set a project before calling this method.

Parameters

model_id – The Gretel model ID.

Raises

GretelProjectNotSetError – If a project has not been set.

Returns

The Gretel model object.

fetch_train_job_results(*model_id: str*) → *TrainJobResults*

Fetch the results object from a Gretel training job.

You must set a project before calling this method.

Parameters

model_id – The Gretel model ID.

Raises

GretelProjectNotSetError – If a project has not been set.

Returns

Job results including the model object, report, logs, and final config.

get_project(***kwargs*) → *Project*

Returns the current Gretel project.

If a project has not been set, a new one will be created. The optional kwargs are the same as those available for the *set_project* method.

run_tuner(*tuner_config: str | Path | dict*, ***, *data_source: str | Path | _DataFrameT*, *n_trials: int = 5*, *n_jobs: int = 1*, *use_temporary_project: bool = False*, *verbose_logging: bool = False*, ***non_default_config_settings*)

Run a hyperparameter tuning experiment with Gretel Tuner.

Parameters

- **tuner_config** – The config as a yaml file path, yaml string, or dict.
- **data_source** – Training data source as a file path or pandas DataFrame.
- **n_trials** – Number of trials to run.
- **n_jobs** – Number of parallel jobs to run locally. Note each job will spin up a Gretel worker.

- **use_temporary_project** – If True, will create a temporary project for the tuning experiment. The project will be deleted when the experiment is complete. If False, will use the current project.
- **verbose_logging** – If True, will print all logs from submitted Gretel jobs.
- ****non_default_config_settings** – Config settings to override in the given tuner config. The kwargs must follow the same nesting format as the yaml config file. See example below.

Raises

ImportError – If the Gretel Tuner is not installed.

Returns

Tuner results dataclass with the best config, best model id, study object, and trial data as attributes.

Example:

```
from gretel_client import Gretel
gretel = Gretel(api_key="prompt")

yaml_config_string = '''
base_config: "tabular-actgan"
metric: synthetic_data_quality_score
params:
    epochs:
        fixed: 50
    batch_size:
        choices: [500, 1000]
privacy_filters:
    similarity:
        choices: ["medium", "high"]
'''

data_source="https://gretel-public-website.s3-us-west-2.amazonaws.com/datasets/
↪USAdultIncome5k.csv"

results = gretel.run_tuner(
    tuner_config=yaml_config_string,
    data_source=data_source,
    n_trials=2,
    params={
        "batch_size": {"choices": [50, 100]},
        "generator_lr": {"log_range": [0.001, 0.01]}
    },
    privacy_filters={"similarity": {"choices": [None, "medium", "high"]}},
)

print(f"Best config: {results.best_config}")

# generate data with best model
generated = gretel.submit_generate(results.best_model_id, num_records=100)
```

set_project(*name: str | None = None*, *desc: str | None = None*, *display_name: str | None = None*)

Set the current Gretel project.

If a project with the given name does not exist, it will be created. If the name is not unique, the user id will be appended to the name.

Parameters

- **name** – Name of new or existing project. If None, will create one.
- **desc** – Project description.
- **display_name** – Project display name. If None, will use project name.

Raises

ApiException – If an error occurs while creating the project.

submit_generate(*model_id: str*, ***, *num_records: int | None = None*, *seed_data: str | Path | _DataFrameT | None = None*, *wait: bool = True*, *fetch_data: bool = True*, *verbose_logging: bool = False*, ***generate_kwargs*) → *GenerateJobResults*

Submit a Gretel model generate job.

Only one of *num_records* or *seed_data* can be provided. The former will generate a complete synthetic dataset, while the latter will conditionally generate synthetic data based on the seed data.

Parameters

- **model_id** – The Gretel model ID.
- **num_records** – Number of records to generate.
- **seed_data** – Seed data source as a file path or pandas DataFrame.
- **wait** – If True, wait for the job to complete before returning.
- **fetch_data** – If True, fetch the synthetic data as a DataFrame.
- **verbose_logging** – If True, will print all logs from the job.

Raises

GretelJobSubmissionError – If the combination of arguments is invalid.

Returns

Job results including the model object, record handler, and synthetic data.

Examples:

```
# Generate a synthetic dataset with 1000 records.
from gretel_client import Gretel
gretel = Gretel(project_name="my-project")
generated = gretel.submit_generate(model_id, num_records=100)

# Conditionally generate synthetic examples of a rare class.
import pandas pd
from gretel_client import Gretel
gretel = Gretel(project_name="my-project")
df_seed = pd.DataFrame(["rare_class"] * 1000, columns=["field_name"])
generated = gretel.submit_generate(model_id, seed_data=df_seed)
```

submit_train(*base_config: str | Path | dict*, ***, *data_source: str | Path | _DataFrameT | None*, *job_label: str | None = None*, *wait: bool = True*, *verbose_logging: bool = False*, ***non_default_config_settings*) → *TrainJobResults*

Submit a Gretel model training job.

Training jobs are configured by updating a base config, which can be given as a dict, yaml file path, yaml string, or as the name of one of the Gretel base config files (without the extension) listed here: https://github.com/gretelai/gretel-blueprints/tree/main/config_templates/gretel/synthetics

Parameters

- **base_config** – Base config name, yaml file path, yaml string, or dict.
- **data_source** – Training data source as a file path or pandas DataFrame.
- **job_label** – Descriptive label to append to job the name.
- **wait** – If True, wait for the job to complete before returning.
- **verbose_logging** – If True, will print all logs from the job.
- ****non_default_config_settings** – Config settings to override in the template. The format is *section={"setting": "value"}*, where *section* is the name of a yaml section within the specific model settings, e.g. *params* or *privacy_filters*. If the parameter is not nested within a section, pass it directly as a keyword argument.

Returns

Job results including the model object, report, logs, and final config.

Example:

```
from gretel_client import Gretel

data_source="https://gretel-public-website.s3-us-west-2.amazonaws.com/datasets/
↪USAdultIncome5k.csv"

gretel = Gretel(project_name="my-project")
trained = gretel.submit_train(
    base_config="tabular-actgan",
    data_source=data_source,
    params={"epochs": 100, "generator_dim": [128, 128]},
    privacy_filters={"similarity": "high", "outliers": None},
)
```

4.2.2 Job Results

class gretel_client.gretel.job_results.**GenerateJobResults**(*project: Project*, *model: Model*, *record_handler: RecordHandler*, *synthetic_data_link: str | None = None*, *synthetic_data: _DataFrameT | None = None*)

Dataclass for the results from a Gretel data generation job.

refresh()

Refresh the generate job results attributes.

wait_for_completion()

Wait for the model to finish generating data.

class gretel_client.gretel.job_results.**GretelJobResults**(*project: Project*, *model: Model*)

Base class for Gretel jobs.

class gretel_client.gretel.job_results.**TrainJobResults**(*project: Project*, *model: Model*, *model_config: dict | None = None*, *report: GretelReport | None = None*, *model_logs: List[dict] | None = None*)

Dataclass for the results from a Gretel model training job.

fetch_report_synthetic_data() → _DataFrameT

Fetch synthetic data generated for the report and return as a DataFrame.

Note: This method requires the *pandas* package to be installed.

refresh()

Refresh the training job results attributes.

wait_for_completion()

Wait for the model to finish training.

4.2.3 Artifact Fetching

class gretel_client.gretel.artifact_fetching.**GretelReport**(*as_dict: dict*, *as_html: str*)

Dataclass for a Gretel synthetic data quality report.

display_in_browser()

Display the HTML report in a browser.

display_in_notebook()

Display the HTML report in a notebook.

save_html(*save_path: str | Path*)

Save the HTML report to a file at the given path.

class gretel_client.gretel.artifact_fetching.**ReportType**(*value*)

The kind of report to fetch.

gretel_client.gretel.artifact_fetching.**fetch_final_model_config**(*model: Model*) → dict

Fetch the final model configuration from a model training job.

Parameters
model – The Gretel model object.

Returns
The final training configuration as a dict.

gretel_client.gretel.artifact_fetching.**fetch_model_logs**(*model: Model*) → List[dict]

Fetch the logs from training a Gretel model.

Parameters
model – The Gretel model object.

Returns
A list of log messages.

gretel_client.gretel.artifact_fetching.**fetch_model_report**(*model: Model*, *report_type: ReportType = ReportType.SQS*) → *GretelReport*

Fetch the quality report from a model training job.

Parameters

- **model** – The Gretel model object.

- **report_type** – The type of report to fetch. One of "sqs" or "text".

Returns
The Gretel report object.

gretel_client.gretel.artifact_fetching.**fetch_synthetic_data**(*record_handler:* RecordHandler) → _DataFrameT

Fetch synthetic data from a model generate job.

This function requires the *pandas* package to be installed.

Parameters
record_handler – A RecordHandler object from the model.

Raises
ImportError – If the *pandas* package is not installed.

Returns
A pandas DataFrame containing the synthetic data.

4.3 Projects SDK

You may use the Projects SDK to programmatically interact with Gretel APIs using a familiar python interface.

The example below ties together a number of concepts to train a synthetic model and then generate data from the model.

```
import pandas as pd

from gretel_client import create_project, poll

project = create_project()

# create a synthetic model using a default synthetic config from
#   https://github.com/gretelai/gretel-blueprints/blob/main/config_templates/gretel/
↪synthetics/default.yml
#
#   Providing a data_source will override the datasource from the template. If the data␣
↪source is a local
#   file, then it will automatically be uploaded to Gretel Cloud as part of the␣
↪submission step
model = project.create_model_obj(
    model_config="synthetics/default",
    data_source="https://gretel-public-website.s3.us-west-2.amazonaws.com/datasets/
↪USAdultIncome5k.csv",
)

# submit the model to Gretel Cloud for training
model.submit()

# wait for the model to training
poll(model)

# read out a preview data from the synthetic model
pd.read_csv(model.get_artifact_link("data_preview"), compression="gzip")
```

Module Reference

4.3.1 Projects

High level API for interacting with a Gretel Project

class gretel_client.projects.projects.**Project**(*, *name: str*, *project_id: str*, *project_guid: str | None = None*, *desc: str | None = None*, *display_name: str | None = None*, *runner_mode: str | RunnerMode | None = None*, *session: ClientConfig | None = None*, *cluster_guid: str | None = None*)

Represents a Gretel project. In general you should not init this class directly, but always make use of the factory methods `get_project`, `create_project`, `get_or_create_project` etc.

Parameters

- **name** – The unique name of the project. This is either set by you or auto managed by Gretel
- **project_id** – The unique project id of your project. This is managed by gretel and never changes.
- **desc** – A short description of the project
- **display_name** – The main display name used in the Gretel Console for your project
- **session** – The client session to use for API interactions, or `None` to use the default session.

property artifacts: List[dict]

Returns a list of project artifacts.

property as_dict: dict

Returns a dictionary representation of the project.

create_model_obj(*model_config: str | Path | dict*, *data_source: str | _DataFrameT | None = None*, *ref_data: str | Dict[str, str] | List[str] | Tuple[str] | _DataFrameT | List[_DataFrameT] | None = None*) → *Model*

Creates a new model object. This will not submit the model to Gretel's cloud API. Please refer to the `Model` docs for more information detailing how to submit the model.

Parameters

- **model_config** – Specifies a model config. For more information about model configs, please refer to our doc site, https://docs.gretel.ai/reference/model-configurations.
- **data_source** – Defines the model data_source. If the model_config already has a data_source defined, this property will override the existing one.
- **ref_data** – An Optional str, dict, dataframe or list of reference data sources to upload for the job.

delete(**args*, ***kwargs*)

Deletes a project. After the project has been deleted, functions relying on a project will fail with a `GretelProjectError` exception.

Note: Deleting projects is asynchronous. It may take a few seconds for the project to be deleted by Gretel services.

delete_artifact(*key: str*)

Deletes a project artifact.

Parameters

key – Artifact key to delete.

get_artifact_handle(*key: str*) → BinaryIO

Returns a reference to a remote artifact that can be used to read binary data within a context manager

```
>>> with job.get_artifact_handle("report_json") as file:
...   print(file.read())
```

Parameters
key – Artifact key to download.

Returns
a file like object

get_artifact_link(*key: str*) → str

Returns a link to download a project artifact.

Parameters
key – Project artifact key to generate download url for.

Returns
A signed URL that may be used to download the given project artifact.

get_console_url() → str

Returns web link to access the project from Gretel's console.

get_model(*model_id: str*) → *Model*

Lookup and return a Project `Model` by it's `model_id`.

Parameters
model_id – The `model_id` to lookup

info() → dict

Return details about the project.

search_models(*factory: ~typing.Type[~gretel_client.projects.projects.MT] = <class 'gretel_client.projects.models.Model'>*, *limit: int = 100*, *model_name: str = ''*) → Iterator[MT]

Search for project models.

Parameters

- **factory** – Determines what type of Model representation is returned. If `Model` is passed, a `Model` will be returned. If `dict` is passed, a dictionary representation of the search results will be returned.
- **limit** – Limits the number of project models to return
- **model_name** – Name of the model to try and match on (partial match)

upload_artifact(*artifact_path: Path | str | _DataFrameT*, *_validate: bool = True*, *_artifacts_handler: ArtifactsHandler | None = None*) → str

Resolves and uploads the data source specified in the model config.

Returns
A Gretel artifact key.

gretel_client.projects.projects.**create_or_get_unique_project**(*, *name: str*, *desc: str | None = None*, *display_name: str | None = None*, *session:* ClientConfig *| None = None*) → *Project*

Helper function that provides a consistent experience for creating and fetching a project with the same name. Given a name of a project, this helper will fetch the current user's ID and use that as a suffix in order to create a project name unique to that user. Once the project is created, if it can be fetched, it will not be re-created over and over again.

Parameters

- **name** – The name of the project, which will have the User's ID appended to it automatically.
- **desc** – Description of the project.
- **display_name** – If None, the display name will be set equal to the value of `name` _before_ the user ID is appended.
- **session** – The client session to use, or `None` to use the default client session.

Note: The `desc` and `display_name` parameters will only be used when the project is first created. If the project already exists, these params will have no affect.

---

gretel_client.projects.projects.**create_project**(*, *name: str | None = None*, *desc: str | None = None*, *display_name: str | None = None*, *session: ClientConfig | None = None*, *runner_mode: str | RunnerMode | None = None*, *hybrid_environment_guid: str | None = None*) → *Project*

Excplit project creation. This function will simply call the API endpoint and will raise any HTTP exceptions upstream.

gretel_client.projects.projects.**get_project**(*, *name: str | None = None*, *create: bool = False*, *desc: str | None = None*, *display_name: str | None = None*, *runner_mode: str | RunnerMode | None = None*, *session: ClientConfig | None = None*) → *Project*

Used to get or create a Gretel project.

Parameters

- **name** – The unique name of the project. This is either set by you or auto managed by Gretel.
- **create** – If create is set to True the function will create the project if it doesn't exist.
- **project_id** – The unique project id of your project. This is managed by gretel and never changes.
- **desc** – A short description of the project
- **display_name** – The main display name used in the Gretel Console for your project
- **session** – the client session to use, or `None` to use the default session.

Returns

A project instance.

gretel_client.projects.projects.**search_projects**(*limit: int = 200*, *query: str | None = None*, *, *session: ClientConfig | None = None*) → List[*Project*]

Searches for project.

Parameters

- **limit** – The max number of projects to return.

- **query** – String filter applied to project names.
- **session** – Can be used to override local Gretel config.

Returns
A list of projects.

gretel_client.projects.projects.**tmp_project**(*, *session:* ClientConfig | *None* = *None*)

A temporary project context manager. Create a new project that can be used inside of a "with" statement for temporary purposes. The project will be deleted from Gretel Cloud when the scope is exited.

Parameters
session – the client session to use, or `None` to use the default session.

Example:

```
with tmp_project() as proj:
    model = proj.create_model_obj()
```

4.3.2 Models

Classes and methods for working with Gretel Models

class gretel_client.projects.models.**Model**(*project: None*, *model_config: str | Path | dict | None = None*, *model_id: str | None = None*)

Represents a Gretel Model. This class can be used to train new models or run and lookup existing ones.

property artifact_types: List[str]

Returns a list of artifact types associated with the model.

property billing_details: dict

Get billing details for the current job.

cancel()

Cancels the active job.

property container_image: str

Return the container image for the job.

create_record_handler_obj(*data_source: str | _DataFrameT | None = None*, *params: dict | None = None*, *ref_data: str | Dict[str, str] | List[str] | Tuple[str] | _DataFrameT | List[_DataFrameT] | None = None*) → *RecordHandler*

Creates a new record handler for the model.

Parameters

- **data_source** – A data source to upload to the record handler.
- **params** – Any custom params for the record handler. These params are specific to the upstream model.

property data_source: str | _DataFrameT | None

Retrieves the configured data source from the model config.

If the model config has a local data_source we'll try and resolve that path relative to the location of the model config.

delete() → dict | None

Deletes the remote model.

download_artifacts(*target_dir: str | Path*)

Given a target directory, either as a string or a Path object, attempt to enumerate and download all artifacts associated with this Job

Parameters
target_dir – The target directory to store artifacts in. If the directory does not exist, it will be created for you.

property errors

Return any errors associated with the model.

property external_data_source: bool

Returns `True` if the data source is external to Gretel Cloud. If the data source is a Gretel Artifact, returns `False`.

property external_ref_data: bool

Returns `True` if the data refs are external to Gretel Cloud. If the data refs are Gretel Artifacts, returns `False`.

get_artifact_handle(*artifact_key: str*) → BinaryIO

Returns a reference to a remote artifact that can be used to read binary data within a context manager

```
>>> with job.get_artifact_handle("report_json") as file:
...   print(file.read())
```

Parameters
artifact_key – Artifact type to download.

Returns
a file like object

get_artifact_link(*artifact_key: str*) → str

Retrieves a signed S3 link that will download the specified artifact type.

Parameters
artifact_key – Artifact type to download.

get_artifacts() → Iterator[Tuple[str, str]]

List artifact links for all known artifact types.

get_artifacts_by_artifact_types(*artifact_types: List[str]*) → Iterator[Tuple[str, str]]

List artifact links for all known artifact types.

get_record_handlers() → Iterator[*RecordHandler*]

Returns a list of record handlers associated with the model.

get_report_summary(*report_path: str | None = None*) → dict | None

Return a summary of the job results :param report_path: If a report_path is passed, that report

will be used for the summary. If no report path is passed, the function will check for a cloud report artifact.

property instance_type: str

Returns CPU or GPU based on the model being trained.

property is_cloud_model

Returns `True` if the model was created to run in Gretel's Cloud. `False` otherwise.

property logs

Returns run logs for the job.

property model_config: dict

Returns the model config used to create the model.

property model_type: str

Returns the type of model. Eg synthetics, transforms or classify.

property name: str | None

Gets the name of the model. If no name is specified, a random name will be selected when the model is submitted to the backend.

Getter
Returns the model name.

Setter
Sets the model name.

peek_report(*report_path: str | None = None*) → dict | None

Return a summary of the job results.

Parameters
report_path – If a report_path is passed, that report will be used for the summary. If no report path is passed, the function will check for a cloud based artifact.

poll_logs_status(*wait: int = -1*, *callback: Callable | None = None*) → Iterator[LogStatus]

Returns an iterator that may be used to tail the logs of a running Model.

Parameters

- **wait** – The time in seconds to wait before closing the iterator. If wait is -1 (WAIT_UNTIL_DONE), the iterator will run until the model has reached a "completed" or "error" state.
- **callback** – This function will be executed on every polling loop. A callback is useful for checking some external state that is working on a Job.

property print_obj: dict

Returns a printable object representation of the job.

project: *Project*

Project associated with the job.

property ref_data: RefData

Retrieves configured ref data from the model config. If there are local ref data sources we will try and resolve that path relative to the location of the model config.

refresh()

Update internal state of the job by making an API call to Gretel Cloud.

property runner_mode: str

Returns the runner_mode of the job. May be one of `hybrid`, `manual` or `cloud`.

property status: Status

The status of the job. Is one of `gretel_client.projects.jobs.Status`.

submit(*runner_mode: str | RunnerMode | None = None*, *dry_run: bool = False*) → Job

Submit this Job to the Gretel Cloud API.

Parameters

- **runner_mode** – Determines where to run the model. If not specified, the runner mode of the project (if configured) is used, otherwise the default runner mode of the session is used.
- **dry_run** – If set to True the model config will be submitted for validation, but won't be run. Ignored for record handlers.

submit_cloud(*dry_run: bool = False*) → Job

Submit this Job to the Gretel Cloud API be scheduled for running in Gretel Cloud.

Returns
The response from the Gretel API.

submit_hybrid(*dry_run: bool = False*) → Job

Submit this Job to the Gretel Cloud API to be scheduled for running in a hybrid deployment.

Returns
The response from the Gretel API.

submit_local(*dry_run: bool = False*) → Job

Submit this Job to the Gretel Cloud API to be scheduled for running in a local container.

Returns
The response from the Gretel API.

submit_manual(*dry_run: bool = False*) → Job

Submit this Job to the Gretel Cloud API, which will create the job metadata but no runner will be started. The `Model` instance can now be passed into a dedicated runner.

Returns
The response from the Gretel API.

property traceback: str | None

Returns the traceback associated with any job errors.

upload_data_source(*_validate: bool = True*, *_artifacts_handler: CloudArtifactsHandler | HybridArtifactsHandler | None = None*) → str | None

Resolves and uploads the data source specified in the model config.

If the data source is already a Gretel artifact, the artifact will not be uploaded.

Returns
A Gretel artifact key.

upload_ref_data(*_validate: bool = True*, *_artifacts_handler: ArtifactsHandler | None = None*) → RefData

Resolves and uploads ref data sources specificed in the model config.

If the ref data are already Gretel artifacts, we'll return the ref data as-is.

Returns
A `RefData` instance that contains the new Gretel artifact values.

validate_data_source()

Tests that the attached data source is a valid CSV or JSON file. If the data source is a Gretel cloud artifact OR a hybrid artifact and the runner mode is hybrid, data validation will be skipped.

Raises

- *DataSourceError* – file can't be opened.
- *DataValidationError* – the data isn't valid CSV or JSON.

worker_key: str | None

Worker key used to launch the job.

gretel_client.projects.models.**read_model_config**(*model_config: str | Path | dict*, ***, *base_url: str = 'https://raw.githubusercontent.com/gretelai/gretel-blueprints/main/config_templates/gretel'*) → dict

Load a Gretel configuration into a dictionary.

Parameters

- **model_config** – This argument may be a string to a file on disk or a Gretel configuration template string such as "synthetics/default". First, this function will treat string input as a location on disk and attempt to read the file and parse it as YAML or JSON. If this is successful, a dict of the config is returned. If the provided *model_config* str is not a file on disk, the function will attempt to resolve the config as a shortcut-path from URL provided by *base_url.*
- **base_url** – A base HTTP URL that should be use to construct a fully qualified path to a configuration template. This URL will be used to resolve a config shortcut string to the fully qualified URL.

4.3.3 Records

class gretel_client.projects.records.**RecordHandler**(*model:* Model, ***, *record_id: str = None*, *data_source: DataSourceTypes | None = None*, *params: dict | None = None*, *ref_data: RefDataTypes | None = None*)

Manages a model's record handler. After a model has been created and trained, a record handler may be used to "run" the model.

Parameters

- **model** – The model to generate a record handler for
- **record_id** – The id of an existing record handler.

property artifact_types: List[str]

Returns a list of valid artifacts for the record handler.

property billing_details: dict

Get billing details for the current job.

cancel()

Cancels the active job.

property container_image: str

Return the container image for the job.

property data_source: str | _DataFrameT | None

Returns the data source with which the record handler was configured, if any.

If the record handler has been submitted, returns the resolved artifact ID. Otherwise, returns the originally-supplied data_source argument.

delete()

Deletes the record handler.

download_artifacts(*target_dir: str | Path*)

Given a target directory, either as a string or a Path object, attempt to enumerate and download all artifacts associated with this Job

Parameters
target_dir – The target directory to store artifacts in. If the directory does not exist, it will be created for you.

property errors

Return any errors associated with the model.

property external_data_source: bool

Returns `True` if the data source is external to Gretel Cloud. If the data source is a Gretel Artifact, returns `False`.

property external_ref_data: bool

Returns `True` if the data refs are external to Gretel Cloud. If the data refs are Gretel Artifacts, returns `False`.

get_artifact_handle(*artifact_key: str*) → BinaryIO

Returns a reference to a remote artifact that can be used to read binary data within a context manager

```
>>> with job.get_artifact_handle("report_json") as file:
...   print(file.read())
```

Parameters
artifact_key – Artifact type to download.

Returns
a file like object

get_artifact_link(*artifact_key: str*) → str

Retrieves a signed S3 link that will download the specified artifact type.

Parameters
artifact_key – Artifact type to download.

get_artifacts() → Iterator[Tuple[str, str]]

List artifact links for all known artifact types.

get_artifacts_by_artifact_types(*artifact_types: List[str]*) → Iterator[Tuple[str, str]]

List artifact links for all known artifact types.

get_report_summary(*report_path: str | None = None*) → dict | None

Return a summary of the job results :param report_path: If a report_path is passed, that report

will be used for the summary. If no report path is passed, the function will check for a cloud report artifact.

property instance_type: str

Return CPU or GPU based on the record handler's run requirements.

property logs

Returns run logs for the job.

property model_type: str

Returns the parent model type of the record handler.

property params: dict | None

Returns the params with which the record handler was configured, if any.

peek_report(*report_path: str | None = None*) → dict | None

Return a summary of the job results.

Parameters

report_path – If a report_path is passed, that report will be used for the summary. If no report path is passed, the function will check for a cloud based artifact.

poll_logs_status(*wait: int = -1*, *callback: Callable | None = None*) → Iterator[LogStatus]

Returns an iterator that may be used to tail the logs of a running Model.

Parameters

- **wait** – The time in seconds to wait before closing the iterator. If wait is -1 (WAIT_UNTIL_DONE), the iterator will run until the model has reached a "completed" or "error" state.
- **callback** – This function will be executed on every polling loop. A callback is useful for checking some external state that is working on a Job.

property print_obj: dict

Returns a printable object representation of the job.

project: *Project*

Project associated with the job.

property ref_data: str | Dict[str, str] | List[str] | Tuple[str] | _DataFrameT | List[_DataFrameT] | None

Returns the ref_data with which the record handler was configured, if any.

refresh()

Update internal state of the job by making an API call to Gretel Cloud.

property runner_mode: str

Returns the runner_mode of the job. May be one of `hybrid`, `manual` or `cloud`.

property status: Status

The status of the job. Is one of `gretel_client.projects.jobs.Status`.

submit(*runner_mode: str | RunnerMode | None = None*, *dry_run: bool = False*) → Job

Submit this Job to the Gretel Cloud API.

Parameters

- **runner_mode** – Determines where to run the model. If not specified, the runner mode of the project (if configured) is used, otherwise the default runner mode of the session is used.
- **dry_run** – If set to True the model config will be submitted for validation, but won't be run. Ignored for record handlers.

submit_cloud(*dry_run: bool = False*) → Job

Submit this Job to the Gretel Cloud API be scheduled for running in Gretel Cloud.

Returns

The response from the Gretel API.

submit_hybrid(*dry_run: bool = False*) → Job

Submit this Job to the Gretel Cloud API to be scheduled for running in a hybrid deployment.

Returns
The response from the Gretel API.

submit_local(*dry_run: bool = False*) → Job

Submit this Job to the Gretel Cloud API to be scheduled for running in a local container.

Returns
The response from the Gretel API.

submit_manual(*dry_run: bool = False*) → Job

Submit this Job to the Gretel Cloud API, which will create the job metadata but no runner will be started. The `Model` instance can now be passed into a dedicated runner.

Returns
The response from the Gretel API.

property traceback: str | None

Returns the traceback associated with any job errors.

upload_data_source(*_validate: bool = True*, *_artifacts_handler: CloudArtifactsHandler | HybridArtifactsHandler | None = None*) → str | None

Resolves and uploads the data source specified in the model config.

If the data source is already a Gretel artifact, the artifact will not be uploaded.

Returns
A Gretel artifact key.

upload_ref_data(*_validate: bool = True*, *_artifacts_handler: ArtifactsHandler | None = None*) → RefData

Resolves and uploads ref data sources specificed in the model config.

If the ref data are already Gretel artifacts, we'll return the ref data as-is.

Returns
A `RefData` instance that contains the new Gretel artifact values.

worker_key: str | None

Worker key used to launch the job.

4.3.4 Docker

Classes for running a Gretel Job as a local container

class gretel_client.projects.docker.**ContainerRun**(*job: Job*)

Runs a Gretel Job from a local container.

Parameters
job – Job to run as docker container.

graceful_shutdown()

Attempts to gracefully shutdown the container run.

is_ok()

Checks to see if the container is ok.

Raises
ContainerRunError` if there is a problem with the container –

start()

Run job via a local container. This method is async and will return after the job has started.

If you wish to block until the container has finished, the wait method may be used.

wait(*timeout: int = 30*)

Blocks until a running container has completed. If the container hasn't started yet, we wait until a timeout interval is reached.

Parameters

timeout – The time in seconds to wait for a container to start. If the timeout is reached, the function will return.

4.3.5 Exceptions

exception gretel_client.projects.exceptions.**ContainerRunError**

There was a problem running the job from a local container.

exception gretel_client.projects.exceptions.**DataSourceError**

Indicates there is a problem reading the data source.

exception gretel_client.projects.exceptions.**DataValidationError**

Indicates there is a problem validating the structure of the data source.

exception gretel_client.projects.exceptions.**DockerEnvironmentError**

The host docker environment isn't configured correctly.

exception gretel_client.projects.exceptions.**GretelJobNotFound**(*job: None*)

The job could not be found. May be either a model or record handler.

exception gretel_client.projects.exceptions.**GretelProjectError**

Problems with a Gretel Project.

exception gretel_client.projects.exceptions.**GretelResourceNotFound**

Baseclass for errors locating remote Gretel resources.

The context property may be overridden to provide additional information that may be helpful in correctly addressing the Gretel resource. Generally, the context should follow a simple key, value dictionary pattern.

exception gretel_client.projects.exceptions.**MaxConcurrentJobsException**(*status=None*, *reason=None*, *http_resp=None*)

Maximum concurrent jobs limit imposed by the server.

For backwards compatibility this exception subclasses the ApiException generated by OpenAPI.

exception gretel_client.projects.exceptions.**ModelConfigError**

Could not read or load the specified model configuration.

exception gretel_client.projects.exceptions.**ModelError**

There was a problem creating the model.

exception gretel_client.projects.exceptions.**ModelNotFoundError**(*job: None*)

Failed to lookup the remote model.

exception gretel_client.projects.exceptions.**RecordHandlerError**

There was a problem run the model.

exception gretel_client.projects.exceptions.**RecordHandlerNotFound**(*job: None*)

Failed to lookup the remote record handler.

exception gretel_client.projects.exceptions.**WaitTimeExceeded**

Thrown when the wait time specified by the user has expired.

4.4 Client Config

class gretel_client.config.**ClientConfig**(**args*, ***kwargs*)

get_api(*api_interface: Type[T]*, *max_retry_attempts: int = 5*, *backoff_factor: float = 1*, ***, *default_headers: dict[str, str] | None = None*) → T

Instantiates and configures an api client for a given component interface.

Parameters

- **api_interface** – The api interface to instantiate
- **max_retry_attempts** – The number of times to retry a failed api request.
- **backoff_factor** – A back factor to apply between retry attempts. A base factor of 2 will applied to this value to determine the time between attempts.

property masked: dict

Returns a masked representation of the config object.

class gretel_client.config.**Context**(*client_metrics: 'dict[str, str]'*, *job_provenance: 'dict[str, str]'*)

gretel_client.config.**DEFAULT_GRETEL_ARTIFACT_ENDPOINT = 'cloud'**

Default artifact endpoint

gretel_client.config.**DEFAULT_GRETEL_ENDPOINT = 'https://api.gretel.cloud'**

Default gretel endpoint

class gretel_client.config.**DefaultClientConfig**(**args*, ***kwargs*)

Holds Gretel client configuration details. This can be instantiated from a file or environment.

api_key: str | None = None

Gretel API key.

artifact_endpoint: str = ''

Artifact endpoint.

default_project_name: str | None = None

Default Gretel project name.

default_runner: str = 'cloud'

Default runner

endpoint: str = ''

Gretel API endpoint.

preview_features: str = 'disabled'

Preview features enabled

update_default_project(*project_id: str*)

Updates the default project.

Parameters

project_name – The name or id of the project to set.

class gretel_client.config.**DelegatingClientConfig**(**args*, ***kwargs*)

gretel_client.config.**GRETEL = 'gretel'**

Gretel application name

gretel_client.config.**GRETEL_API_KEY = 'GRETEL_API_KEY'**

Env variable to configure Gretel api key.

gretel_client.config.**GRETEL_ARTIFACT_ENDPOINT = 'GRETEL_ARTIFACT_ENDPOINT'**

Env variable name to configure artifact endpoint. Defaults to DEFAULT_GRETEL_ARTIFACT_ENDPOINT.

gretel_client.config.**GRETEL_CONFIG_FILE = 'GRETEL_CONFIG_FILE'**

Env variable name to override default configuration file location

gretel_client.config.**GRETEL_ENDPOINT = 'GRETEL_ENDPOINT'**

Env variable name to configure default Gretel endpoint. Defaults to DEFAULT_GRETEL_ENDPOINT.

gretel_client.config.**GRETEL_PREVIEW_FEATURES = 'GRETEL_PREVIEW_FEATURES'**

Env variable to manage preview features

gretel_client.config.**GRETEL_PROJECT = 'GRETEL_PROJECT'**

Env variable name to select default project

gretel_client.config.**GRETEL_RUNNER_MODE = 'GRETEL_RUNNER_MODE'**

Env variable name to set the default runner mode

class gretel_client.config.**GretelApiRetry**(**args*, ***kwargs*)

Custom retry logic for calling Gretel Cloud APIs.

increment(*method=None*, *url=None*, *response=None*, *error=None*, *_pool=None*, *_stacktrace=None*)

Return a new Retry object with incremented retry counters.

Parameters

- **response** (HTTPResponse) – A response object, or None, if the server did not return a response.
- **error** (*Exception*) – An error encountered during the request, or None if the response was received successfully.

Returns

A new Retry object.

exception gretel_client.config.**GretelClientConfigurationError**

class gretel_client.config.**PreviewFeatures**(*value*)

Manage preview feature configurations

class gretel_client.config.**RunnerMode**(*value*)

An enumeration.

class gretel_client.config.**TaggedClientConfig**(**args*, ***kwargs*)

get_api(*api_interface: Type[T]*, *max_retry_attempts: int = 5*, *backoff_factor: float = 1*, ***, *default_headers: dict[str, str] | None = None*) → T

Instantiates and configures an api client for a given component interface.

Parameters

- **api_interface** – The api interface to instantiate
- **max_retry_attempts** – The number of times to retry a failed api request.
- **backoff_factor** – A back factor to apply between retry attempts. A base factor of 2 will applied to this value to determine the time between attempts.

gretel_client.config.**clear_gretel_config**()

Removes any Gretel configuration files from the host file system.

If any Gretel related environment variables exist, this will also remove them from the current processes.

gretel_client.config.**configure_session**(*config: str | ClientConfig | None = None*, ***, *api_key: str | None = None*, *default_runner: str | RunnerMode | None = None*, *endpoint: str | None = None*, *artifact_endpoint: str | None = None*, *cache: str = 'no'*, *validate: bool = False*, *clear: bool = False*)

Updates client config for the session

Parameters

- **config** – The config to update. This config takes precedence over other parameters such as `api_key` or `endpoint`.
- **api_key** – Configures your Gretel API key. If `api_key` is set to "prompt" and no Api Key is found on the system, `getpass` will be used to prompt for the key.
- **default_runner** – Specifies the runner mode. Must be one of "cloud", "local", "manual", or "hybrid".
- **endpoint** – Specifies the Gretel API endpoint. This must be a fully qualified URL.
- **artifact_endpoint** – Specifies the endpoint for project and model artifacts.
- **cache** – Valid options include "yes" and "no". If cache is "no" the session configuration will not be written to disk. If cache is "yes", session configuration will be written to disk only if a configuration doesn't exist.
- **validate** – If set to `True` this will check that login credentials are valid.
- **clear** – If set to `True` any existing Gretel credentials will be removed from the host.

Raises

GretelClientConfigurationError` if validate=True and credential – are invalid.

gretel_client.config.**get_session_config**() → *ClientConfig*

Return the session's client config

gretel_client.config.**write_config**(*config:* ClientConfig, *config_path: str | Path | None = None*) → Path

Writes a Gretel client config to disk.

Parameters

- **config** – The client config to write
- **config_path** – Path to write the config to. If not path is provided, the default `$HOME/.gretel/config.json` path is used.

4.5 Quality Report

class gretel_client.evaluation.quality_report.**QualityReport**(*, *project:* Project | *None* = *None*, *name: str* | *None* = *None*, *data_source: str* | *Path* | *_DataFrameT*, *ref_data: Path* | *str* | *_DataFrameT*, *output_dir: str* | *Path* | *None* = *None*, *runner_mode:* RunnerMode | *None* = *None*, *record_count: int* | *None* = *5000*, *correlation_column_count: int* | *None* = *75*, *column_count: int* | *None* = *250*, *mandatory_columns: List[str]* | *None* = *[]*, *session:* ClientConfig | *None* = *None*)

Represents a Quality Report. This class can be used to create a report.

Parameters

- **project** – Optional project associated with the report. If no project is passed, a temp project (*gretel_client.projects.projects.tmp_project*) will be used.
- **data_source** – Data source used for the report.
- **ref_data** – Reference data used for the report.
- **output_dir** – Optional directory path to write the report to. If the directory does not exist, the path will be created for you.
- **runner_mode** – Determines where to run the model. See *gretel_client.config.RunnerMode* for a list of valid modes. Manual mode is not explicitly supported.
- **record_count** – Number of rows to use from the data sets, 5000 by default. A value of 0 means "use as many rows/columns as possible." We still attempt to maintain parity between the data sets for "fair" comparisons, i.e. we will use min(len(train), len(synth)), e.g.
- **correlation_column_count** – Similar to record_count, but for number of columns used for correlation calculations.
- **column_count** – Similar to record_count, but for number of columns used for all other calculations.
- **mandatory_columns** – Use in conjuction with correlation_column_count and column_count. The columns listed will be included in the sample of columns. Any additional requested columns will be selected randomly.
- **session** – The client session to use, or `None` to use the session associated with the project (if any), or the default session otherwise.

property as_dict: Dict[str, Any]

Returns a dictionary representation of the report.

property as_html: str

Returns a HTML representation of the report.

data_source: DataSourceTypes

Data source used for the report.

output_dir: Path | None

Optional directory path to write the report to. If the directory does not exist, the path will be created for you.

peek() → Dict[str, Any] | None

Returns a dictionary representation of the top level report scores.

project: *Project* | None

Optional project associated with the report. If no project is passed, a temp project (`tmp_project`) will be used.

ref_data: RefDataTypes

Reference data used for the report.

runner_mode: *RunnerMode*

Determines where to run the model. See `RunnerMode` for a list of valid modes. Manual mode is not explicitly supported.

test_data: RefDataTypes

Additional optional test data used for MQS reports.

4.6 Helpers

gretel_client.helpers.**do_api_call**(*method: str*, *path: str*, *query_params: dict | None = None*, *body: dict | None = None*, *headers: dict | None = None*, ***, *session:* ClientConfig | *None = None*) → dict

Make a direct API call to Gretel Cloud.

Parameters

- **method** – "get", "post", etc
- **path** – The full path to make the request to, any path params must be already included. Example: "/users/me"
- **query_params** – Optional URL based query parameters
- **body** – An optional JSON payload to send
- **headers** – Any custom headers that need to bet set.
- **session** – the session to use, or `None` to use the default session.

Note: This function will automatically inject the appropriate API hostname and authentication from the Gretel configuration.

---

gretel_client.helpers.**poll**(*job: Job*, *wait: int = -1*, *verbose: bool = True*) → Callable

Polls a `Model` or `RecordHandler`.

Parameters

- **job** – The job to poll.
- **wait** – The time to wait for the job to complete.
- **verbose** – `False` uses new quiet polling, defaults to `True`.

gretel_client.helpers.**submit_docker_local**(*job: Job*, *, *output_dir: str | Path | None = None*, *in_data: str | Path | None = None*, *ref_data: Dict[str, str | Path] | None = None*, *model_path: str | Path | None = None*) → *ContainerRun*

Run a *Job* from a local docker container.

While the Job is running, the *submit_docker_local* function will block and periodically send back status updates as the Job progresses.

Note: Please ensure the Job has not already been submitted. If the Job has already been submitted, the run will fail.

Parameters

- **job** – The job to run. May be either a `Model` or `RecordHandler`.
- **output_dir** – A directory path to write the output to. If the directory does not exist, the path will be created for you. If no path is specified, the current working directory is used.
- **in_data** – Input data path.
- **ref_data** – Reference data path or dict where values are reference data path
- **model_path** – If you are running a `RecordHandler`, this is the path to the model that is being ran.

Returns

A `ContainerRun` that can be used to manage the lifecycle of the associated local docker container.

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

g

INDEX

E

F

G

I

L

M

N

O

P

Q

R

S

T

U

V

W