pandas-gbq Documentation

Release 0.13.0+2.ge177978

PyData Development Team

Feb 13, 2020

Contents

The `pandas_gbq` module provides a wrapper for Google's BigQuery analytics web service to simplify retrieving results from BigQuery tables using SQL-like queries. Result sets are parsed into a `pandas.DataFrame` with a shape and data types derived from the source table. Additionally, DataFrames can be inserted into new BigQuery tables or appended to existing tables.

Note: To use this module, you will need a valid BigQuery account. Use the BigQuery sandbox to try the service for free.

While BigQuery uses standard SQL syntax, it has some important differences from traditional databases both in functionality, API limitations (size and quantity of queries or uploads), and how Google charges for use of the service. BiqQuery is best for analyzing large sets of data quickly. It is not a direct replacement for a transactional database. Refer to the BigQuery Documentation for details on the service itself.

Contents:

CHAPTER 1

Installation

You can install pandas-gbq with `conda`, `pip`, or by installing from source.

1.1 Conda

```
$ conda install pandas-gbq --channel conda-forge
```

This installs pandas-gbq and all common dependencies, including `pandas`.

1.2 Pip

To install the latest version of pandas-gbq: from the

```
$ pip install pandas-gbq -U
```

This installs pandas-gbq and all common dependencies, including `pandas`.

1.3 Install from Source

```
$ pip install git+https://github.com/pydata/pandas-gbq.git
```

1.4 Dependencies

This module requires following additional dependencies:

- pydata-google-auth: Helpers for authentication to Google's API

- google-auth: authentication and authorization for Google's API
- google-auth-oauthlib: integration with oauthlib for end-user authentication
- google-cloud-bigquery: Google Cloud client library for BigQuery
- google-cloud-bigquery-storage: Google Cloud client library for BigQuery Storage API

Note: The dependency on google-cloud-bigquery is new in version 0.3.0 of `pandas-gbq`. Versions less than 0.3.0 required the following dependencies:

- httplib2: HTTP client (no longer required)
- google-api-python-client: Google's API client (no longer required, replaced by google-cloud-bigquery:)
- google-auth: authentication and authorization for Google's API
- google-auth-oauthlib: integration with oauthlib for end-user authentication
- google-auth-httplib2: adapter to use `httplib2` HTTP client with `google-auth` (no longer required)

CHAPTER 2

Introduction

The pandas-gbq package reads data from Google BigQuery to a `pandas.DataFrame` object and also writes `pandas.DataFrame` objects to BigQuery tables.

2.1 Authenticating to BigQuery

Before you begin, you must create a Google Cloud Platform project. Use the BigQuery sandbox to try the service for free.

If you do not provide any credentials, this module attempts to load credentials from the environment. If no credentials are found, pandas-gbq prompts you to open a web browser, where you can grant it permissions to access your cloud resources. These credentials are only used locally. See the *privacy policy* for details.

Learn about authentication methods in the *authentication guide*.

2.2 Reading data from BigQuery

Use the *`pandas_gbq.read_gbq()`* function to run a BigQuery query and download the results as a `pandas.DataFrame` object.

```
import pandas_gbq

# TODO: Set project_id to your Google Cloud Platform project ID.
# project_id = "my-project"

sql = """
SELECT country_name, alpha_2_code
FROM `bigquery-public-data.utility_us.country_code_iso`
WHERE alpha_2_code LIKE 'A%'
"""
df = pandas_gbq.read_gbq(sql, project_id=project_id)
```

By default, queries use standard SQL syntax. Visit the *reading tables guide* to learn about the available options.

2.2.1 Adjusting log vebosity

Because some requests take some time, this library will log its progress of longer queries. IPython & Jupyter by default attach a handler to the logger. If you're running in another process and want to see logs, or you want to see more verbose logs, you can do something like:

```
import logging
logger = logging.getLogger('pandas_gbq')
logger.setLevel(logging.DEBUG)
logger.addHandler(logging.StreamHandler())
```

2.3 Writing data to BigQuery

Use the *pandas_gbq.to_gbq()* function to write a pandas.DataFrame object to a BigQuery table.

```
import pandas
import pandas_gbq

# TODO: Set project_id to your Google Cloud Platform project ID.
# project_id = "my-project"

# TODO: Set table_id to the full destination table ID (including the
#       dataset ID).
# table_id = 'my_dataset.my_table'

df = pandas.DataFrame(
    {
        "my_string": ["a", "b", "c"],
        "my_int64": [1, 2, 3],
        "my_float64": [4.0, 5.0, 6.0],
        "my_bool1": [True, False, True],
        "my_bool2": [False, True, False],
        "my_dates": pandas.date_range("now", periods=3),
    }
)

pandas_gbq.to_gbq(df, table_id, project_id=project_id)
```

The destination table and destination dataset will automatically be created. By default, writes to BigQuery fail if the table already exists. Visit the *writing tables guide* to learn about the available options.

CHAPTER 3

Authentication

Before you begin, you must create a Google Cloud Platform project. Use the BigQuery sandbox to try the service for free.

pandas-gbq authenticates with the Google BigQuery service via OAuth 2.0. Use the `credentials` argument to explicitly pass in Google `Credentials`.

3.1 Default Authentication Methods

If the `credentials` parameter is not set, pandas-gbq tries the following authentication methods:

1. In-memory, cached credentials at `pandas_gbq.context.credentials`. See *`pandas_gbq.Context.credentials`* for details.

```
import pandas_gbq

credentials = ...  # From google-auth or pydata-google-auth library.

# Update the in-memory credentials cache (added in pandas-gbq 0.7.0).
pandas_gbq.context.credentials = credentials
pandas_gbq.context.project = "your-project-id"

# The credentials and project_id arguments can be omitted.
df = pandas_gbq.read_gbq("SELECT my_col FROM `my_dataset.my_table`")
```

2. Application Default Credentials via the `google.auth.default()` function.

Note: If pandas-gbq can obtain default credentials but those credentials cannot be used to query BigQuery, pandas-gbq will also try obtaining user account credentials.

A common problem with default credentials when running on Google Compute Engine is that the VM does not have sufficient scopes to query BigQuery.

---

3. User account credentials.

   pandas-gbq loads cached credentials from a hidden user folder on the operating system.

   Windows `%APPDATA%\pandas_gbq\bigquery_credentials.dat`

   Linux/Mac/Unix `~/.config/pandas_gbq/bigquery_credentials.dat`

   If pandas-gbq does not find cached credentials, it prompts you to open a web browser, where you can grant pandas-gbq permissions to access your cloud resources. These credentials are only used locally. See the privacy policy for details.

3.2 Authenticating with a Service Account

Using service account credentials is particularly useful when working on remote servers without access to user input.

Create a service account key via the service account key creation page in the Google Cloud Platform Console. Select the JSON key type and download the key file.

To use service account credentials, set the `credentials` parameter to the result of a call to:

- **`google.oauth2.service_account.Credentials.from_service_account_file()`**, which accepts a file path to the JSON file.

```
from google.oauth2 import service_account
import pandas_gbq

credentials = service_account.Credentials.from_service_account_file(
    'path/to/key.json',
)
df = pandas_gbq.read_gbq(sql, project_id="YOUR-PROJECT-ID",␣
↪credentials=credentials)
```

- **`google.oauth2.service_account.Credentials.from_service_account_info()`**, which accepts a dictionary corresponding to the JSON file contents.

```
from google.oauth2 import service_account
import pandas_gbq

credentials = service_account.Credentials.from_service_account_info(
    {
        "type": "service_account",
        "project_id": "YOUR-PROJECT-ID",
        "private_key_id": "6747200734a1f2b9d8d62fc0b9414c5f2461db0e",
        "private_key": "-----BEGIN PRIVATE KEY-----\nM...I==\n-----END␣
↪PRIVATE KEY-----\n",
        "client_email": "service-account@YOUR-PROJECT-ID.iam.gserviceaccount.
↪com",
        "client_id": "12345678900001",
        "auth_uri": "https://accounts.google.com/o/oauth2/auth",
        "token_uri": "https://accounts.google.com/o/oauth2/token",
        "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/
↪certs",
        "client_x509_cert_url": "https://www.googleapis.com/...iam.
↪gserviceaccount.com"
    },
)
df = pandas_gbq.read_gbq(sql, project_id="YOUR-PROJECT-ID",␣
↪credentials=credentials)
```

(continues on next page)

(continued from previous page)

Use the `with_scopes()` method to use authorize with specific OAuth2 scopes, which may be required in queries to federated data sources such as Google Sheets.

```
credentials = ...
credentials = credentials.with_scopes(
    [
        'https://www.googleapis.com/auth/drive',
        'https://www.googleapis.com/auth/cloud-platform',
    ],
)
df = pandas_gbq.read_gbq(..., credentials=credentials)
```

See the Getting started with authentication on Google Cloud Platform guide for more information on service accounts.

3.3 Authenticating with a User Account

Use the pydata-google-auth library to authenticate with a user account (i.e. a G Suite or Gmail account). The `pydata_google_auth.get_user_credentials()` function loads credentials from a cache on disk or initiates an OAuth 2.0 flow if cached credentials are not found.

```
import pandas_gbq
import pydata_google_auth

SCOPES = [
    'https://www.googleapis.com/auth/cloud-platform',
    'https://www.googleapis.com/auth/drive',
]

credentials = pydata_google_auth.get_user_credentials(
    SCOPES,
    # Set auth_local_webserver to True to have a slightly more convienient
    # authorization flow. Note, this doesn't work if you're running from a
    # notebook on a remote sever, such as over SSH or with Google Colab.
    auth_local_webserver=True,
)

df = pandas_gbq.read_gbq(
    "SELECT my_col FROM `my_dataset.my_table`",
    project_id='YOUR-PROJECT-ID',
    credentials=credentials,
)
```

Warning: Do not store credentials on disk when using shared computing resources such as a GCE VM or Colab notebook. Use the `pydata_google_auth.cache.NOOP` cache to avoid writing credentials to disk.

```
import pydata_google_auth.cache

credentials = pydata_google_auth.get_user_credentials(
    SCOPES,
    # Use the NOOP cache to avoid writing credentials to disk.
    cache=pydata_google_auth.cache.NOOP,
)
```

Additional information on the user credentials authentication mechanism can be found in the Google Cloud authentication guide.

CHAPTER 4

Reading Tables

Use the *pandas_gbq.read_gbq()* function to run a BigQuery query and download the results as a `pandas.DataFrame` object.

```
import pandas_gbq

# TODO: Set project_id to your Google Cloud Platform project ID.
# project_id = "my-project"

sql = """
SELECT country_name, alpha_2_code
FROM `bigquery-public-data.utility_us.country_code_iso`
WHERE alpha_2_code LIKE 'A%'
"""
df = pandas_gbq.read_gbq(sql, project_id=project_id)
```

Note: A project ID is optional if it can be inferred during authentication, but it is required when authenticating with user credentials. You can find your project ID in the Google Cloud console.

You can define which column from BigQuery to use as an index in the destination DataFrame as well as a preferred column order as follows:

```
data_frame = pandas_gbq.read_gbq(
    'SELECT * FROM `test_dataset.test_table`',
    project_id=projectid,
    index_col='index_column_name',
    col_order=['col1', 'col2', 'col3'])
```

4.1 Querying with legacy SQL syntax

The `dialect` argument can be used to indicate whether to use BigQuery's `'legacy'` SQL or BigQuery's `'standard'` SQL. The default value is `'standard'`.

```
sql = """
SELECT country_name, alpha_2_code
FROM [bigquery-public-data:utility_us.country_code_iso]
WHERE alpha_2_code LIKE 'Z%'
"""
df = pandas_gbq.read_gbq(
    sql,
    project_id=project_id,
    # Set the dialect to "legacy" to use legacy SQL syntax. As of
    # pandas-gbq version 0.10.0, the default dialect is "standard".
    dialect="legacy",
)
```

- Standard SQL reference
- Legacy SQL reference

4.2 Inferring the DataFrame's dtypes

The *`read_gbq()`* method infers the pandas dtype for each column, based on the BigQuery table schema.

BigQuery Data Type	dtype
FLOAT	float
TIMESTAMP	`DatetimeTZDtype` with `unit='ns'` and `tz='UTC'`
DATETIME	datetime64[ns]
TIME	datetime64[ns]
DATE	datetime64[ns]

4.3 Improving download performance

Use the BigQuery Storage API to download large (>125 MB) query results more quickly (but at an increased cost) by setting `use_bqstorage_api` to `True`.

1. Enable the BigQuery Storage API on the project you are using to run queries.

   Enable the API.

2. Ensure you have the bigquery.readsessions.create permission. to create BigQuery Storage API read sessions. This permission is provided by the bigquery.user role.

3. **Install the `google-cloud-bigquery-storage` and `pyarrow` packages.**

   With pip:

   ```
   pip install --upgrade google-cloud-bigquery-storage pyarrow
   ```

   With conda:

```
conda install -c conda-forge google-cloud-bigquery-storage
```

4. Set `use_bqstorage_api` to `True` when calling the *`read_gbq()`* function. As of the `google-cloud-bigquery` package, version 1.11.1 or later,the function will fallback to the BigQuery API if the BigQuery Storage API cannot be used, such as with small query results.

4.4 Advanced configuration

You can specify the query config as parameter to use additional options of your job. Refer to the JobConfiguration REST resource reference for details.

```
configuration = {
   'query': {
     "useQueryCache": False
   }
}
data_frame = read_gbq(
    'SELECT * FROM `test_dataset.test_table`',
    project_id=projectid,
    configuration=configuration)
```

CHAPTER 5

Writing Tables

Use the *pandas_gbq.to_gbq()* function to write a `pandas.DataFrame` object to a BigQuery table.

```
import pandas
import pandas_gbq

# TODO: Set project_id to your Google Cloud Platform project ID.
# project_id = "my-project"

# TODO: Set table_id to the full destination table ID (including the
#       dataset ID).
# table_id = 'my_dataset.my_table'

df = pandas.DataFrame(
    {
        "my_string": ["a", "b", "c"],
        "my_int64": [1, 2, 3],
        "my_float64": [4.0, 5.0, 6.0],
        "my_bool1": [True, False, True],
        "my_bool2": [False, True, False],
        "my_dates": pandas.date_range("now", periods=3),
    }
)

pandas_gbq.to_gbq(df, table_id, project_id=project_id)
```

The destination table and destination dataset will automatically be created if they do not already exist.

5.1 Writing to an Existing Table

Use the `if_exists` argument to dictate whether to `'fail'`, `'replace'` or `'append'` if the destination table already exists. The default value is `'fail'`.

For example, assume that `if_exists` is set to `'fail'`. The following snippet will raise a `TableCreationError` if the destination table already exists.

```
import pandas_gbq
pandas_gbq.to_gbq(
    df, 'my_dataset.my_table', project_id=projectid, if_exists='fail',
)
```

If the `if_exists` argument is set to `'append'`, the destination dataframe will be written to the table using the defined table schema and column types. The dataframe must contain fields (matching name and type) currently in the destination table.

5.2 Inferring the Table Schema

The *`to_gbq()`* method infers the BigQuery table schema based on the dtypes of the uploaded `DataFrame`.

dtype	BigQuery Data Type
i (integer)	INTEGER
b (boolean)	BOOLEAN
f (float)	FLOAT
O (object)	STRING
S (zero-terminated bytes)	STRING
U (Unicode string)	STRING
M (datetime)	TIMESTAMP

If the data type inference does not suit your needs, supply a BigQuery schema as the `table_schema` parameter of *`to_gbq()`*.

5.3 Troubleshooting Errors

If an error occurs while writing data to BigQuery, see Troubleshooting BigQuery Errors.

CHAPTER 6

API Reference

Note: Only functions and classes which are members of the `pandas_gbq` module are considered public. Submodules and their members are considered private.

read_gbq(query[, project_id, index_col, ...])	Load data from Google BigQuery using google-cloud-python
to_gbq(dataframe, destination_table[, ...])	Write a DataFrame to a Google BigQuery table.
context	Storage for objects to be used throughout a session.
Context()	Storage for objects to be used throughout a session.

pandas_gbq.**read_gbq**(*query*, *project_id=None*, *index_col=None*, *col_order=None*, *reauth=False*, *auth_local_webserver=False*, *dialect=None*, *location=None*, *configuration=None*, *credentials=None*, *use_bqstorage_api=False*, *max_results=None*, *verbose=None*, *private_key=None*, *progress_bar_type='tqdm'*)

Load data from Google BigQuery using google-cloud-python

The main method a user calls to execute a Query in Google BigQuery and read results into a pandas DataFrame.

This method uses the Google Cloud client library to make requests to Google BigQuery, documented here.

See the *How to authenticate with Google BigQuery* guide for authentication instructions.

Parameters

query [str] SQL-Like Query to return data values.

project_id [str, optional] Google BigQuery Account project ID. Optional when available from the environment.

index_col [str, optional] Name of result column to use for index in results DataFrame.

col_order [list(str), optional] List of BigQuery column names in the desired order for results DataFrame.

reauth [boolean, default False] Force Google BigQuery to re-authenticate the user. This is useful if multiple accounts are used.

auth_local_webserver [boolean, default False] Use the local webserver flow instead of the console flow when getting user credentials.

New in version 0.2.0.

dialect [str, default 'standard'] Note: The default value changed to 'standard' in version 0.10.0.

SQL syntax dialect to use. Value can be one of:

'legacy' Use BigQuery's legacy SQL dialect. For more information see BigQuery Legacy SQL Reference.

'standard' Use BigQuery's standard SQL, which is compliant with the SQL 2011 standard. For more information see BigQuery Standard SQL Reference.

location [str, optional] Location where the query job should run. See the BigQuery locations documentation for a list of available locations. The location must match that of any datasets used in the query.

New in version 0.5.0.

configuration [dict, optional] Query config parameters for job processing. For example:

configuration = {'query': {'useQueryCache': False}}

For more information see BigQuery REST API Reference.

credentials [google.auth.credentials.Credentials, optional] Credentials for accessing Google APIs. Use this parameter to override default credentials, such as to use Compute Engine `google.auth.compute_engine.Credentials` or Service Account `google.oauth2.service_account.Credentials` directly.

New in version 0.8.0.

use_bqstorage_api [bool, default False] Use the BigQuery Storage API to download query results quickly, but at an increased cost. To use this API, first enable it in the Cloud Console. You must also have the bigquery.readsessions.create permission on the project you are billing queries to.

Note: Due to a *known issue in the ''google-cloud-bigquery*' package <https://github.com/googleapis/google-cloud-python/pull/7633>'__ (fixed in version 1.11.0), you must write your query results to a destination table. To do this with `read_gbq`, supply a `configuration` dictionary.

This feature requires the `google-cloud-bigquery-storage` and `pyarrow` packages.

This value is ignored if `max_results` is set.

New in version 0.10.0.

max_results [int, optional] If set, limit the maximum number of rows to fetch from the query results.

New in version 0.12.0.

progress_bar_type (Optional[str]): If set, use the tqdm library to display a progress bar while the data downloads. Install the `tqdm` package to use this feature. Possible values of `progress_bar_type` include:

None No progress bar.

'tqdm' Use the `tqdm.tqdm()` function to print a progress bar to `sys.stderr`.

'tqdm_notebook' Use the `tqdm.tqdm_notebook()` function to display a progress bar as a Jupyter notebook widget.

'tqdm_gui' Use the `tqdm.tqdm_gui()` function to display a progress bar as a graphical dialog box.

verbose [None, deprecated] Deprecated in Pandas-GBQ 0.4.0. Use the logging module to adjust verbosity instead.

private_key [str, deprecated] Deprecated in pandas-gbq version 0.8.0. Use the `credentials` parameter and `google.oauth2.service_account.Credentials.from_service_account_info()` or `google.oauth2.service_account.Credentials.from_service_account_file()` instead.

Returns

df: DataFrame DataFrame representing results of query.

pandas_gbq.**to_gbq**(*dataframe*, *destination_table*, *project_id=None*, *chunksize=None*, *reauth=False*, *if_exists='fail'*, *auth_local_webserver=False*, *table_schema=None*, *location=None*, *progress_bar=True*, *credentials=None*, *verbose=None*, *private_key=None*)

Write a DataFrame to a Google BigQuery table.

The main method a user calls to export pandas DataFrame contents to Google BigQuery table.

This method uses the Google Cloud client library to make requests to Google BigQuery, documented here.

See the *How to authenticate with Google BigQuery* guide for authentication instructions.

Parameters

dataframe [pandas.DataFrame] DataFrame to be written to a Google BigQuery table.

destination_table [str] Name of table to be written, in the form `dataset.tablename`.

project_id [str, optional] Google BigQuery Account project ID. Optional when available from the environment.

chunksize [int, optional] Number of rows to be inserted in each chunk from the dataframe. Set to `None` to load the whole dataframe at once.

reauth [bool, default False] Force Google BigQuery to re-authenticate the user. This is useful if multiple accounts are used.

if_exists [str, default 'fail'] Behavior when the destination table exists. Value can be one of:

'fail' If table exists, do nothing.

'replace' If table exists, drop it, recreate it, and insert data.

'append' If table exists, insert data. Create if does not exist.

auth_local_webserver [bool, default False] Use the local webserver flow instead of the console flow when getting user credentials.

New in version 0.2.0.

table_schema [list of dicts, optional] List of BigQuery table fields to which according DataFrame columns conform to, e.g. `[{'name': 'col1', 'type': 'STRING'},...]`.

- If `table_schema` is provided, it may contain all or a subset of DataFrame columns. If a subset is provided, the rest will be inferred from the DataFrame dtypes.

- If `table_schema` is **not** provided, it will be generated according to dtypes of DataFrame columns. See Inferring the Table Schema. for a description of the schema inference.

See *BigQuery API documentation on valid column names <https://cloud.google.com/bigquery/docs/schemas#column_names>__*.

New in version 0.3.1.

location [str, optional] Location where the load job should run. See the BigQuery locations documentation for a list of available locations. The location must match that of the target dataset.

New in version 0.5.0.

progress_bar [bool, default True] Use the library *tqdm* to show the progress bar for the upload, chunk by chunk.

New in version 0.5.0.

credentials [google.auth.credentials.Credentials, optional] Credentials for accessing Google APIs. Use this parameter to override default credentials, such as to use Compute Engine `google.auth.compute_engine.Credentials` or Service Account `google.oauth2.service_account.Credentials` directly.

New in version 0.8.0.

verbose [bool, deprecated] Deprecated in Pandas-GBQ 0.4.0. Use the logging module to adjust verbosity instead.

private_key [str, deprecated] Deprecated in pandas-gbq version 0.8.0. Use the `credentials` parameter and `google.oauth2.service_account.Credentials.from_service_account_info()` or `google.oauth2.service_account.Credentials.from_service_account_file()` instead.

pandas_gbq.**context = <pandas_gbq.gbq.Context object>**

Storage for objects to be used throughout a session.

A Context object is initialized when the `pandas_gbq` module is imported, and can be found at *pandas_gbq.context*.

class pandas_gbq.**Context**

Storage for objects to be used throughout a session.

A Context object is initialized when the `pandas_gbq` module is imported, and can be found at *pandas_gbq.context*.

Attributes

credentials Credentials to use for Google APIs.

dialect Default dialect to use in *pandas_gbq.read_gbq()*.

project Default project to use for calls to Google APIs.

credentials

Credentials to use for Google APIs.

These credentials are automatically cached in memory by calls to *pandas_gbq.read_gbq()* and *pandas_gbq.to_gbq()*. To manually set the credentials, construct an `google.auth.credentials.Credentials` object and set it as the context credentials as demonstrated in the example below. See auth docs for more information on obtaining credentials.

Returns

google.auth.credentials.Credentials

Examples

Manually setting the context credentials:

```
>>> import pandas_gbq
>>> from google.oauth2 import service_account
>>> credentials = service_account.Credentials.from_service_account_file(
...     '/path/to/key.json',
... )
>>> pandas_gbq.context.credentials = credentials
```

dialect

Default dialect to use in *pandas_gbq.read_gbq()*.

Allowed values for the BigQuery SQL syntax dialect:

'legacy' Use BigQuery's legacy SQL dialect. For more information see BigQuery Legacy SQL Reference.

'standard' Use BigQuery's standard SQL, which is compliant with the SQL 2011 standard. For more information see BigQuery Standard SQL Reference.

Returns

str

Examples

Setting the default syntax to standard:

```
>>> import pandas_gbq
>>> pandas_gbq.context.dialect = 'standard'
```

project

Default project to use for calls to Google APIs.

Returns

str

Examples

Manually setting the context project:

```
>>> import pandas_gbq
>>> pandas_gbq.context.project = 'my-project'
```

CHAPTER 7

Contributing to pandas-gbq

Table of contents:

7.1 Where to start?

All contributions, bug reports, bug fixes, documentation improvements, enhancements and ideas are welcome.

If you are simply looking to start working with the *pandas-gbq* codebase, navigate to the GitHub "issues" tab and start looking through interesting issues.

Or maybe through using *pandas-gbq* you have an idea of your own or are looking for something in the documentation and thinking 'this can be improved'... you can do something about it!

Feel free to ask questions on the mailing list.

7.2 Bug reports and enhancement requests

Bug reports are an important part of making *pandas-gbq* more stable. Having a complete bug report will allow others to reproduce the bug and provide insight into fixing it. Because many versions of *pandas-gbq* are supported, knowing version information will also identify improvements made since previous versions. Trying the bug-producing code out on the *master* branch is often a worthwhile exercise to confirm the bug still exists. It is also worth searching existing bug reports and pull requests to see if the issue has already been reported and/or fixed.

Bug reports must:

1. Include a short, self-contained Python snippet reproducing the problem. You can format the code nicely by using GitHub Flavored Markdown

```
>>> from pandas_gbq import gbq
>>> df = gbq.read_gbq(...)
...
```

2. Include the full version string of *pandas-gbq*.

```
>>> import pandas_gbq
>>> pandas_gbq.__version__
...
```

3. Explain why the current behavior is wrong/not desired and what you expect instead.

The issue will then show up to the *pandas-gbq* community and be open to comments/ideas from others.

7.3 Working with the code

Now that you have an issue you want to fix, enhancement to add, or documentation to improve, you need to learn how to work with GitHub and the *pandas-gbq* code base.

7.3.1 Version control, Git, and GitHub

To the new user, working with Git is one of the more daunting aspects of contributing to *pandas-gbq*. It can very quickly become overwhelming, but sticking to the guidelines below will help keep the process straightforward and mostly trouble free. As always, if you are having difficulties please feel free to ask for help.

The code is hosted on GitHub. To contribute you will need to sign up for a free GitHub account. We use Git for version control to allow many people to work together on the project.

Some great resources for learning Git:

- the GitHub help pages.
- the NumPy's documentation.
- Matthew Brett's Pydagogue.

7.3.2 Getting started with Git

GitHub has instructions for installing git, setting up your SSH key, and configuring git. All these steps need to be completed before you can work seamlessly between your local repository and GitHub.

7.3.3 Forking

You will need your own fork to work on the code. Go to the pandas-gbq project page and hit the `Fork` button. You will want to clone your fork to your machine:

```
git clone git@github.com:your-user-name/pandas-gbq.git pandas-gbq-yourname
cd pandas-gbq-yourname
git remote add upstream git://github.com/pydata/pandas-gbq.git
```

This creates the directory *pandas-gbq-yourname* and connects your repository to the upstream (main project) *pandas-gbq* repository.

The testing suite will run automatically on CircleCI once your pull request is submitted. However, if you wish to run the test suite on a branch prior to submitting the pull request, then CircleCI needs to be hooked up to your GitHub repository. Instructions for doing so are here.

7.3.4 Creating a branch

You want your master branch to reflect only production-ready code, so create a feature branch for making your changes. For example:

```
git branch shiny-new-feature
git checkout shiny-new-feature
```

The above can be simplified to:

```
git checkout -b shiny-new-feature
```

This changes your working directory to the shiny-new-feature branch. Keep any changes in this branch specific to one bug or feature so it is clear what the branch brings to *pandas-gbq*. You can have many shiny-new-features and switch in between them using the git checkout command.

To update this branch, you need to retrieve the changes from the master branch:

```
git fetch upstream
git rebase upstream/master
```

This will replay your commits on top of the latest pandas-gbq git master. If this leads to merge conflicts, you must resolve these before submitting your pull request. If you have uncommitted changes, you will need to `stash` them prior to updating. This will effectively store your changes and they can be reapplied after updating.

7.3.5 Install in Development Mode

It's helpful to install pandas-gbq in development mode so that you can use the library without reinstalling the package after every change.

Conda

Create a new conda environment and install the necessary dependencies

```
$ conda create -n my-env --channel conda-forge  \
      pandas \
      google-auth-oauthlib \
      google-api-python-client \
      google-auth-httplib2
$ source activate my-env
```

Install pandas-gbq in development mode

```
$ python setup.py develop
```

Pip & virtualenv

Skip this section if you already followed the conda instructions.

Create a new virtual environment.

```
$ virtualenv env
$ source env/bin/activate
```

You can install pandas-gbq and its dependencies in development mode via pip.

```
$ pip install -e .
```

7.4 Contributing to the code base

Code Base:

7.4.1 Code standards

Writing good code is not just about what you write. It is also about *how* you write it. During testing on Travis-CI, several tools will be run to check your code for stylistic errors. Generating any warnings will cause the test to fail. Thus, good style is a requirement for submitting code to *pandas-gbq*.

In addition, because a lot of people use our library, it is important that we do not make sudden changes to the code that could have the potential to break a lot of user code as a result, that is, we need it to be as *backwards compatible* as possible to avoid mass breakages.

Python (PEP8)

pandas-gbq uses the PEP8 standard. There are several tools to ensure you abide by this standard. Here are *some* of the more common `PEP8` issues:

- we restrict line-length to 79 characters to promote readability
- passing arguments should have spaces after commas, e.g. `foo(arg1, arg2, kw1='bar')`

CircleCI will run the 'black' code formatting tool and report any stylistic errors in your code. Therefore, it is helpful before submitting code to run the formatter yourself:

```
pip install black
black .
```

Backwards Compatibility

Please try to maintain backward compatibility. If you think breakage is required, clearly state why as part of the pull request. Also, be careful when changing method signatures and add deprecation warnings where needed.

7.4.2 Test-driven development/code writing

pandas-gbq is serious about testing and strongly encourages contributors to embrace test-driven development (TDD). This development process "relies on the repetition of a very short development cycle: first the developer writes an (initially failing) automated test case that defines a desired improvement or new function, then produces the minimum amount of code to pass that test." So, before actually writing any code, you should write your tests. Often the test

can be taken from the original GitHub issue. However, it is always worth considering additional use cases and writing corresponding tests.

Adding tests is one of the most common requests after code is pushed to *pandas-gbq*. Therefore, it is worth getting in the habit of writing tests ahead of time so this is never an issue.

Like many packages, *pandas-gbq* uses pytest.

Running the test suite

The tests can then be run directly inside your Git clone (without having to install *pandas-gbq*) by typing:

```
pytest tests/unit
pytest tests/system.py
```

The tests suite is exhaustive and takes around 20 minutes to run. Often it is worth running only a subset of tests first around your changes before running the entire suite.

The easiest way to do this is with:

```
pytest tests/path/to/test.py -k regex_matching_test_name
```

Or with one of the following constructs:

```
pytest tests/[test-module].py
pytest tests/[test-module].py::[TestClass]
pytest tests/[test-module].py::[TestClass]::[test_method]
```

For more, see the pytest documentation.

Testing on multiple Python versions

pandas-gbq uses nox to automate testing in multiple Python environments. First, install nox.

```
$ pip install --upgrade nox-automation
```

To run tests in all versions of Python, run *nox* from the repository's root directory.

Running Google BigQuery Integration Tests

You will need to create a Google BigQuery private key in JSON format in order to run Google BigQuery integration tests on your local machine and on CircleCI. The first step is to create a service account. Grant the service account permissions to run BigQuery queries and to create datasets and tables.

To run the integration tests locally, set the following environment variables before running `pytest`:

1. `GBQ_PROJECT_ID` with the value being the ID of your BigQuery project.
2. `GBQ_GOOGLE_APPLICATION_CREDENTIALS` with the value being the *path* to the JSON key that you downloaded for your service account.

Integration tests are skipped in pull requests because the credentials that are required for running Google BigQuery integration tests are configured in the CircleCI web interface and are only accessible from the pydata/pandas-gbq repository. The credentials won't be available on forks of pandas-gbq. Here are the steps to run gbq integration tests on a forked repository:

1. Go to CircleCI and sign in with your GitHub account.

2. Switch to your personal account in the top-left organization switcher.
3. Use the "Add projects" tab to enable CircleCI for your fork.
4. Click on the gear icon to edit your CircleCI build, and add two environment variables:
   - `GBQ_PROJECT_ID` with the value being the ID of your BigQuery project.
   - `SERVICE_ACCOUNT_KEY` with the value being the base64-encoded *contents* of the JSON key that you downloaded for your service account.

   Keep the contents of these variables confidential. These variables contain sensitive data and you do not want their contents being exposed in build logs.
5. Your branch should be tested automatically once it is pushed. You can check the status by visiting your Travis branches page which exists at the following location: https://circleci.com/gh/your-username/pandas-gbq . Click on a build job for your branch.

7.4.3 Documenting your code

Changes should be reflected in the release notes located in `doc/source/changelog.rst`. This file contains an ongoing change log. Add an entry to this file to document your fix, enhancement or (unavoidable) breaking change. Make sure to include the GitHub issue number when adding your entry (using `` GH#1234 `` where *1234* is the issue/pull request number).

If your code is an enhancement, it is most likely necessary to add usage examples to the existing documentation. Further, to let users know when this feature was added, the `versionadded` directive is used. The sphinx syntax for that is:

```
.. versionadded:: 0.1.3
```

This will put the text *New in version 0.1.3* wherever you put the sphinx directive. This should also be put in the docstring when adding a new function or method.

7.5 Contributing your changes to *pandas-gbq*

7.5.1 Committing your code

Keep style fixes to a separate commit to make your pull request more readable.

Once you've made changes, you can see them by typing:

```
git status
```

If you have created a new file, it is not being tracked by git. Add it by typing:

```
git add path/to/file-to-be-added.py
```

Doing 'git status' again should give something like:

```
# On branch shiny-new-feature
#
#       modified:   /relative/path/to/file-you-added.py
#
```

Finally, commit your changes to your local repository with an explanatory message. *pandas-gbq* uses a convention for commit message prefixes and layout. Here are some common prefixes along with general guidelines for when to use them:

- ENH: Enhancement, new functionality
- BUG: Bug fix
- DOC: Additions/updates to documentation
- TST: Additions/updates to tests
- BLD: Updates to the build process/scripts
- PERF: Performance improvement
- CLN: Code cleanup

The following defines how a commit message should be structured. Please reference the relevant GitHub issues in your commit message using GH1234 or #1234. Either style is fine, but the former is generally preferred:

- a subject line with < *80* chars.
- One blank line.
- Optionally, a commit message body.

Now you can commit your changes in your local repository:

```
git commit -m
```

7.5.2 Combining commits

If you have multiple commits, you may want to combine them into one commit, often referred to as "squashing" or "rebasing". This is a common request by package maintainers when submitting a pull request as it maintains a more compact commit history. To rebase your commits:

```
git rebase -i HEAD~#
```

Where # is the number of commits you want to combine. Then you can pick the relevant commit message and discard others.

To squash to the master branch do:

```
git rebase -i master
```

Use the `s` option on a commit to `squash`, meaning to keep the commit messages, or `f` to `fixup`, meaning to merge the commit messages.

Then you will need to push the branch (see below) forcefully to replace the current commits with the new ones:

```
git push origin shiny-new-feature -f
```

7.5.3 Pushing your changes

When you want your changes to appear publicly on your GitHub page, push your forked feature branch's commits:

```
git push origin shiny-new-feature
```

Here `origin` is the default name given to your remote repository on GitHub. You can see the remote repositories:

```
git remote -v
```

If you added the upstream repository as described above you will see something like:

```
origin  git@github.com:yourname/pandas-gbq.git (fetch)
origin  git@github.com:yourname/pandas-gbq.git (push)
upstream        git://github.com/pydata/pandas-gbq.git (fetch)
upstream        git://github.com/pydata/pandas-gbq.git (push)
```

Now your code is on GitHub, but it is not yet a part of the *pandas-gbq* project. For that to happen, a pull request needs to be submitted on GitHub.

7.5.4 Review your code

When you're ready to ask for a code review, file a pull request. Before you do, once again make sure that you have followed all the guidelines outlined in this document regarding code style, tests, performance tests, and documentation. You should also double check your branch changes against the branch it was based on:

1. Navigate to your repository on GitHub – https://github.com/your-user-name/pandas-gbq
2. Click on `Branches`
3. Click on the `Compare` button for your feature branch
4. Select the `base` and `compare` branches, if necessary. This will be `master` and `shiny-new-feature`, respectively.

7.5.5 Finally, make the pull request

If everything looks good, you are ready to make a pull request. A pull request is how code from a local repository becomes available to the GitHub community and can be looked at and eventually merged into the master version. This pull request and its associated changes will eventually be committed to the master branch and available in the next release. To submit a pull request:

1. Navigate to your repository on GitHub
2. Click on the `Pull Request` button
3. You can then click on `Commits` and `Files Changed` to make sure everything looks okay one last time
4. Write a description of your changes in the `Preview Discussion` tab
5. Click `Send Pull Request`.

This request then goes to the repository maintainers, and they will review the code. If you need to make more changes, you can make them in your branch, push them to GitHub, and the pull request will be automatically updated. Pushing them to GitHub again is done by:

```
git push -f origin shiny-new-feature
```

This will automatically update your pull request with the latest code and restart the Travis-CI tests.

7.5.6 Delete your merged branch (optional)

Once your feature branch is accepted into upstream, you'll probably want to get rid of the branch. First, merge upstream master into your branch so git knows it is safe to delete your branch:

```
git fetch upstream
git checkout master
git merge upstream/master
```

Then you can just do:

```
git branch -d shiny-new-feature
```

Make sure you use a lower-case `-d`, or else git won't warn you if your feature branch has not actually been merged.

The branch will still exist on GitHub, so to delete it there do:

```
git push origin --delete shiny-new-feature
```

CHAPTER 8

Changelog

8.1 0.13.1 / 2020-02-13

- Fix `AttributeError` with BQ Storage API to download empty results. (GH#299)

8.2 0.13.0 / 2019-12-12

- Raise `NotImplementedError` when the deprecated `private_key` argument is used. (GH#301)

8.3 0.12.0 / 2019-11-25

- Add `max_results` argument to *`read_gbq()`*. Use this argument to limit the number of rows in the results DataFrame. Set `max_results` to 0 to ignore query outputs, such as for DML or DDL queries. (GH#102)
- Add `progress_bar_type` argument to *`read_gbq()`*. Use this argument to display a progress bar when downloading data. (GH#182)

8.3.1 Dependency updates

- Update the minimum version of `google-cloud-bigquery` to 1.11.1. (GH#296)

8.3.2 Documentation

- Add code samples to introduction and refactor howto guides. (GH#239)

8.4 0.11.0 / 2019-07-29

- **Breaking Change:** Python 2 support has been dropped. This is to align with the pandas package which dropped Python 2 support at the end of 2019. (GH#268)

8.4.1 Enhancements

- Ensure `table_schema` argument is not modified inplace. (GH#278)

8.4.2 Implementation changes

- Use object dtype for `STRING`, `ARRAY`, and `STRUCT` columns when there are zero rows. (GH#285)

8.4.3 Internal changes

- Populate `user-agent` with `pandas` version information. (GH#281)
- Fix `pytest.raises` usage for latest pytest. Fix warnings in tests. (GH#282)
- Update CI to install nightly packages in the conda tests. (GH#254)

8.5 0.10.0 / 2019-04-05

- **Breaking Change:** Default SQL dialect is now `standard`. Use `pandas_gbq.context.dialect` to override the default value. (GH#195, GH#245)

8.5.1 Documentation

- Document *BigQuery data type to pandas dtype conversion* for `read_gbq`. (GH#269)

8.5.2 Dependency updates

- Update the minimum version of `google-cloud-bigquery` to 1.9.0. (GH#247)
- Update the minimum version of `pandas` to 0.19.0. (GH#262)

8.5.3 Internal changes

- Update the authentication credentials. **Note:** You may need to set `reauth=True` in order to update your credentials to the most recent version. This is required to use new functionality such as the BigQuery Storage API. (GH#267)
- Use `to_dataframe()` from `google-cloud-bigquery` in the `read_gbq()` function. (GH#247)

8.5.4 Enhancements

- Fix a bug where pandas-gbq could not upload an empty DataFrame. (GH#237)
- Allow `table_schema` in `to_gbq()` to contain only a subset of columns, with the rest being populated using the DataFrame dtypes (GH#218) (contributed by @johnpaton)
- Read `project_id` in `to_gbq()` from provided `credentials` if available (contributed by @daureg)
- `read_gbq` uses the timezone-aware `DatetimeTZDtype(unit='ns', tz='UTC')` dtype for BigQuery `TIMESTAMP` columns. (GH#269)
- Add `use_bqstorage_api` to `read_gbq()`. The BigQuery Storage API can be used to download large query results (>125 MB) more quickly. If the BQ Storage API can't be used, the BigQuery API is used instead. (GH#133, GH#270)

8.6 0.9.0 / 2019-01-11

- Warn when deprecated `private_key` parameter is used (GH#240)
- **New dependency** Use the `pydata-google-auth` package for authentication. (GH#241)

8.7 0.8.0 / 2018-11-12

8.7.1 Breaking changes

- **Deprecate** `private_key` parameter to *`pandas_gbq.read_gbq()`* and *`pandas_gbq.to_gbq()`* in favor of new `credentials` argument. Instead, create a credentials object using `google.oauth2.service_account.Credentials.from_service_account_info()` or `google.oauth2.service_account.Credentials.from_service_account_file()`. See the *authentication how-to guide* for examples. (GH#161, GH#231)

8.7.2 Enhancements

- Allow newlines in data passed to `to_gbq`. (GH#180)
- Add `pandas_gbq.context.dialect` to allow overriding the default SQL syntax dialect. (GH#195, GH#235)
- Support Python 3.7. (GH#197, GH#232)

8.7.3 Internal changes

- Migrate tests to CircleCI. (GH#228, GH#232)

8.8 0.7.0 / 2018-10-19

- *int* columns which contain *NULL* are now cast to *float*, rather than *object* type. (GH#174)
- *DATE*, *DATETIME* and *TIMESTAMP* columns are now parsed as pandas' *timestamp* objects (GH#224)

- Add *pandas_gbq.Context* to cache credentials in-memory, across calls to `read_gbq` and `to_gbq`. (GH#198, GH#208)
- Fast queries now do not log above `DEBUG` level. (GH#204) With BigQuery's release of clustering querying smaller samples of data is now faster and cheaper.
- Don't load credentials from disk if reauth is `True`. (GH#212) This fixes a bug where pandas-gbq could not refresh credentials if the cached credentials were invalid, revoked, or expired, even when `reauth=True`.
- Catch RefreshError when trying credentials. (GH#226)

8.8.1 Internal changes

- Avoid listing datasets and tables in system tests. (GH#215)
- Improved performance from eliminating some duplicative parsing steps (GH#224)

8.9 0.6.1 / 2018-09-11

- Improved `read_gbq` performance and memory consumption by delegating `DataFrame` construction to the Pandas library, radically reducing the number of loops that execute in python (GH#128)
- Reduced verbosity of logging from `read_gbq`, particularly for short queries. (GH#201)
- Avoid `SELECT 1` query when running `to_gbq`. (GH#202)

8.10 0.6.0 / 2018-08-15

- Warn when `dialect` is not passed in to `read_gbq`. The default dialect will be changing from 'legacy' to 'standard' in a future version. (GH#195)
- Use general float with 15 decimal digit precision when writing to local CSV buffer in `to_gbq`. This prevents numerical overflow in certain edge cases. (GH#192)

8.11 0.5.0 / 2018-06-15

- Project ID parameter is optional in `read_gbq` and `to_gbq` when it can inferred from the environment. Note: you must still pass in a project ID when using user-based authentication. (GH#103)
- Progress bar added for `to_gbq`, through an optional library *tqdm* as dependency. (GH#162)
- Add location parameter to `read_gbq` and `to_gbq` so that pandas-gbq can work with datasets in the Tokyo region. (GH#177)

8.11.1 Documentation

- Add *authentication how-to guide*. (GH#183)
- Update *Contributing to pandas-gbq* guide with new paths to tests. (GH#154, GH#164)

8.11.2 Internal changes

- Tests now use *nox* to run in multiple Python environments. (GH#52)
- Renamed internal modules. (GH#154)
- Refactored auth to an internal auth module. (GH#176)
- Add unit tests for `get_credentials()`. (GH#184)

8.12 0.4.1 / 2018-04-05

- Only show `verbose` deprecation warning if Pandas version does not populate it. (GH#157)

8.13 0.4.0 / 2018-04-03

- Fix bug in *read_gbq* when building a dataframe with integer columns on Windows. Explicitly use 64bit integers when converting from BQ types. (GH#119)
- Fix bug in *read_gbq* when querying for an array of floats (GH#123)
- Fix bug in *read_gbq* with configuration argument. Updates *read_gbq* to account for breaking change in the way `google-cloud-python` version 0.32.0+ handles query configuration API representation. (GH#152)
- Fix bug in *to_gbq* where seconds were discarded in timestamp columns. (GH#148)
- Fix bug in *to_gbq* when supplying a user-defined schema (GH#150)
- **Deprecate** the `verbose` parameter in *read_gbq* and *to_gbq*. Messages use the logging module instead of printing progress directly to standard output. (GH#12)

8.14 0.3.1 / 2018-02-13

- Fix an issue where Unicode couldn't be uploaded in Python 2 (GH#106)
- Add support for a passed schema in :func:`to_gbq` instead inferring the schema from the passed `DataFrame` with `DataFrame.dtypes` (GH#46)
- Fix an issue where a dataframe containing both integer and floating point columns could not be uploaded with `to_gbq` (GH#116)
- `to_gbq` now uses `to_csv` to avoid manually looping over rows in a dataframe (should result in faster table uploads) (GH#96)

8.15 0.3.0 / 2018-01-03

- Use the google-cloud-bigquery library for API calls. The `google-cloud-bigquery` package is a new dependency, and dependencies on `google-api-python-client` and `httplib2` are removed. See the installation guide for more details. (GH#93)
- Structs and arrays are now named properly (GH#23) and BigQuery functions like `array_agg` no longer run into errors during type conversion (GH#22).

- `to_gbq()` now uses a load job instead of the streaming API. Remove `StreamingInsertError` class, as it is no longer used by `to_gbq()`. (GH#7, GH#75)

8.16 0.2.1 / 2017-11-27

- `read_gbq()` now raises `QueryTimeout` if the request exceeds the `query.timeoutMs` value specified in the BigQuery configuration. (GH#76)
- Environment variable `PANDAS_GBQ_CREDENTIALS_FILE` can now be used to override the default location where the BigQuery user account credentials are stored. (GH#86)
- BigQuery user account credentials are now stored in an application-specific hidden user folder on the operating system. (GH#41)

8.17 0.2.0 / 2017-07-24

- Drop support for Python 3.4 (GH#40)
- The dataframe passed to `` `.to_gbq(...., if_exists='append')` `` needs to contain only a subset of the fields in the BigQuery schema. (GH#24)
- Use the google-auth library for authentication because `oauth2client` is deprecated. (GH#39)
- `read_gbq()` now has a `auth_local_webserver` boolean argument for controlling whether to use web server or console flow when getting user credentials. Replaces *–noauth_local_webserver* command line argument. (GH#35)
- `read_gbq()` now displays the BigQuery Job ID and standard price in verbose output. (GH#70 and GH#71)

8.18 0.1.6 / 2017-05-03

- All gbq errors will simply be subclasses of `ValueError` and no longer inherit from the deprecated `PandasError`.

8.19 0.1.4 / 2017-03-17

- `InvalidIndexColumn` will be raised instead of `InvalidColumnOrder` in `read_gbq()` when the index column specified does not exist in the BigQuery schema. (GH#6)

8.20 0.1.3 / 2017-03-04

- Bug with appending to a BigQuery table where fields have modes (NULLABLE,REQUIRED,REPEATED) specified. These modes were compared versus the remote schema and writing a table via `to_gbq()` would previously raise. (GH#13)

8.21 0.1.2 / 2017-02-23

Initial release of transfered code from pandas

Includes patches since the 0.19.2 release on pandas with the following:

- `read_gbq()` now allows query configuration preferences pandas-GH#14742
- `read_gbq()` now stores `INTEGER` columns as `dtype=object` if they contain `NULL` values. Otherwise they are stored as `int64`. This prevents precision lost for integers greather than 2**53. Furthermore `FLOAT` columns with values above 10**4 are no longer casted to `int64` which also caused precision loss pandas-GH#14064, and pandas-GH#14305

CHAPTER 9

Privacy

This package is a PyData project and is subject to the NumFocus privacy policy. Your use of Google APIs with this module is subject to each API's respective terms of service.

9.1 Google account and user data

9.1.1 Accessing user data

The `pandas_gbq` module accesses Google Cloud Platform resources from your local machine. Your machine communicates directly with the Google APIs.

The *`read_gbq()`* function can read and write BigQuery data (and other data such as Google Sheets or Cloud Storage, via the federated query feature) through the BigQuery query interface via queries you supply.

The *`to_gbq()`* method can write data you supply to a BigQuery table.

9.1.2 Storing user data

By default, your credentials are stored to a local file, such as `~/.config/pandas_gbq/bigquery_credentials.dat`. See the *Authenticating with a User Account* guide for details. All user data is stored on your local machine. **Use caution when using this library on a shared machine**.

9.1.3 Sharing user data

The pandas-gbq library only communicates with Google APIs. No user data is shared with PyData, NumFocus, or any other servers.

9.2 Policies for application authors

Do not use the default client ID when using the pandas-gbq library from an application, library, or tool. Per the Google User Data Policy, your application must accurately represent itself when authenticating to Google API servcies.

CHAPTER 10

Indices and tables

- genindex
- modindex
- search

Index