
great_expectations Documentation

The Great Expectations Team

Aug 11, 2020

CONTENTS

1	Introduction	3
1.1	What is Great Expectations?	3
1.2	Why would I use Great Expectations?	3
1.3	Key features	4
1.4	Workflow advantages	4
1.5	What does Great Expectations NOT do?	5
1.6	How do I get started?	5
2	Tutorials	7
2.1	Quick start	7
2.2	Getting started	8
2.3	Contribute to Great Expectations	23
3	How-to guides	25
3.1	Configuring Data Contexts	25
3.2	Configuring Datasources	27
3.3	Configuring metadata stores	41
3.4	Creating Batches	49
3.5	Creating and editing Expectations	51
3.6	Validation	81
3.7	Configuring Data Docs	99
3.8	Migrating between versions	102
3.9	Miscellaneous	111
3.10	How-to guides: Spare parts	132
4	Reference	155
4.1	Glossary of Expectations	155
4.2	Core concepts	157
4.3	Supporting resources	187
4.4	Changelog	188
4.5	Reference: Spare parts	210
5	Community resources	215
5.1	Get in touch with the Great Expectations team	215
5.2	Ask a question	215
5.3	File a bug report or feature request	215
5.4	Contribute code or documentation	215
6	Contributing	217
6.1	Setting up your dev environment	217
6.2	Contribution checklist	219

6.3	Making changes directly through Github	221
6.4	Testing	223
6.5	Levels of maturity	226
6.6	Style Guide	226
6.7	Miscellaneous	229
7	Index	233
7.1	API Reference	233
	Python Module Index	663
	Index	667

Great Expectations is a leading tool for *validating*, *documenting*, and *profiling* your data to maintain quality and improve communication between teams. Head over to the *Introduction* to learn more, or jump straight to our *Getting started* guide.

Attention: Welcome to our brand new docs site! We are actively improving the docs and migrating content to this new format, but there are still some places under construction. Feel free to ask questions on our [slack channel](#) or [discussion forum](#)!

INTRODUCTION

Always know what to expect from your data.

1.1 What is Great Expectations?

Great Expectations helps teams save time and promote analytic integrity by offering a unique approach to automated testing: pipeline tests. Pipeline tests are applied to data (instead of code) and at batch time (instead of compile or deploy time). Pipeline tests are like unit tests for datasets: they help you guard against upstream data changes and monitor data quality.

Software developers have long known that automated testing is essential for managing complex codebases. Great Expectations brings the same discipline, confidence, and acceleration to data science and engineering teams.

1.2 Why would I use Great Expectations?

To get more done with data, faster. Teams use Great Expectations to

- Save time during data cleaning and munging.
- Accelerate ETL and data normalization.
- Streamline analyst-to-engineer handoffs.
- Streamline knowledge capture and requirements gathering from subject-matter experts.
- Monitor data quality in production data pipelines and data products.
- Automate verification of new data deliveries from vendors and other teams.
- Simplify debugging data pipelines if (when) they break.
- Codify assumptions used to build models when sharing with other teams or analysts.
- Develop rich, shared data documentation in the course of normal work.
- Make implicit knowledge explicit.
- ... and much more

1.3 Key features

Expectations

Expectations are the workhorse abstraction in Great Expectations. Like assertions in traditional python unit tests, Expectations provide a flexible, declarative language for describing expected behavior. Unlike traditional unit tests, Great Expectations applies Expectations to data instead of code.

Great Expectations currently supports native execution of Expectations in three environments: pandas, SQL (including distributed environments like BigQuery and Redshift), and Spark. This approach follows the philosophy of “take the compute to the data.” Future releases of Great Expectations will extend this functionality to other frameworks, such as dask.

Automated data profiling

Writing pipeline tests from scratch can be tedious and overwhelming. Great Expectations jump starts the process by providing powerful tools for *automated data profiling*. This provides the double benefit of helping you explore data faster, and capturing knowledge for future documentation and testing.

Data Contexts and Data Sources

... allow you to configure connections to your *Datasources*, using names you’re already familiar with: “the ml_training_results bucket in S3,” “the Users table in Redshift.” Great Expectations provides convenience libraries to introspect most common data stores (Ex: SQL databases, data directories and S3 buckets.) We are also working to integrate with pipeline execution frameworks (Ex: Airflow, dbt, Dagster, Prefect). The Great Expectations framework lets you fetch, validate, profile, and document your data in a way that’s meaningful within your existing infrastructure and work environment.

Tooling for validation

Evaluating Expectations against data is just one step in a typical validation workflow. Checkpoints make the followup steps simple, too: storing validation results to a shared bucket, summarizing results and posting notifications to slack, handling differences between warnings and errors, etc.

Great Expectations also provides robust concepts of Batches and Runs. Although we sometimes talk informally about validating “dataframes” or “tables,” it’s much more common to validate batches of new data—subsets of tables, rather than whole tables. Data Contexts provide simple, universal syntax to generate, fetch, and validate Batches of data from any of your DataSources.

Data Docs

Great Expectations can render Expectations to clean, human-readable documentation. Since docs are compiled from tests and you are running tests against new data as it arrives, your documentation is guaranteed to never go stale.

1.4 Workflow advantages

Most data science and data engineering teams end up building some form of pipeline testing, eventually. Unfortunately, many teams don’t get around to it until late in the game, long after early lessons from data exploration and model development have been forgotten.

In the meantime, data pipelines often become deep stacks of unverified assumptions. Mysterious (and sometimes embarrassing) bugs crop up more and more frequently. Resolving them requires painstaking exploration of upstream data, often leading to frustrating negotiations about data specs across teams.

It’s not unusual to see data teams grind to a halt for weeks (or even months!) to pay down accumulated pipeline debt. This work is never fun—after all, it’s just data cleaning: no new products shipped; no new insights kindled. Even

worse, it's re-cleaning old data that you thought you'd already dealt with. In our experience, servicing pipeline debt is one of the biggest productivity and morale killers on data teams.

We strongly believe that most of this pain is avoidable. We built Great Expectations to make it very, very simple to

1. set up and deploy your testing and documentation framework,
2. author Expectations through a combination of automated profiling and expert knowledge capture, and
3. systematically validate new data against them.

It's the best tool we know of for managing the complexity that inevitably grows within data pipelines. We hope it helps you as much as it's helped us.

Good night and good luck!

1.5 What does Great Expectations NOT do?

Great Expectations is NOT a pipeline execution framework.

We aim to integrate seamlessly with DAG execution tools like [Spark](#), [Airflow](#), [dbt](#), [Prefect](#), [Dagster](#), [Kedro](#), etc. We DON'T execute your pipelines for you.

Great Expectations is NOT a data versioning tool.

Great Expectations does not store data itself. Instead, it deals in metadata about data: Expectations, validation results, etc. If you want to bring your data itself under version control, check out tools like: [DVC](#) and [Quilt](#).

Great Expectations currently works best in a Python/Bash environment.

Great Expectations is Python-based. You can invoke it from the command line without using a Python programming environment, but if you're working in another ecosystem, other tools might be a better choice. If you're running in a pure R environment, you might consider [assertR](#) as an alternative. Within the TensorFlow ecosystem, [TFDV](#) fulfills a similar function as Great Expectations.

1.6 How do I get started?

Check out [Getting started](#) to set up your first deployment of Great Expectations, and learn important concepts along the way.

If you'd like to contribute to Great Expectations, please start [here](#).

If you're interested in a paid support contract or consulting services for Great Expectations, please see options [here](#)

For other questions and resources, please visit [Community resources](#).

TUTORIALS

These tutorials will teach you everything you need to know to get up and running with Great Expectations.

If you're the impatient type, head to [Quick start](#) to get going with no fuss or explanation.

If you're a new user of Great Expectations, check out [Getting started](#) to learn important concepts by setting your first deployment.

If you're interested in contributing to the open source project, thank you! Please check out [Contribute to Great Expectations](#) to learn how.

2.1 Quick start

Admonition from Mr. Dickens.

“What is before you is a fight with the world; and the sooner you begin it, the better.”

Off we go! Install Great Expectations:

```
pip install great_expectations
```

You can quickly try out Great Expectations using this guide: [How to quickly explore data using Expectations in a notebook](#).

To unlock more of the power of Great Expectations, you'll also need to configure a Data Context. From the root of the directory where you want to deploy Great Expectations:

```
great_expectations init
```

The CLI will guide you through all the steps to set up a basic deployment of Great Expectations.

After that, if you want to understand what just happened, check out [Getting started](#). Alternatively, you can [Customize your deployment](#).

2.2 Getting started

Welcome to Great Expectations! This tutorial will help you set up your first deployment of Great Expectations. We'll also introduce important concepts, with links to detailed material you can dig into later.

Please follow these steps to get started:

In Great Expectations, your *Data Context* manages boilerplate configuration. Using a Data Context is almost always the fastest way to get up and running, even though some teams don't need every component of a Data Context.

Follow these instructions to *Initialize a Data Context*.

Once you have a Data Context, you'll want to connect to data. In Great Expectations, *Datasources* simplify connections, by managing configuration and providing a consistent, cross-platform API for referencing data.

Check out *Connect to data* to learn how to configure your first Datasource.

Expectations are the workhorse abstraction in Great Expectations, a flexible, declarative language for describing the expected characteristics of data.

Click through to *Create your first Expectations*.

One of Great Expectations' core promises is that your tests and documentation will always stay in sync, because docs and tests are both *compiled from the same Expectations*.

To see how this works, follow *these instructions* to set up a local static website for your data documentation. Later, you'll be able to host the site remotely, or integrate content generated by Great Expectations into an metadata store.

In normal usage, the best way to validate data is with a Checkpoint. Checkpoints simplify deployment, by pre-specifying the data and Expectations that to validate at any given point in your data infrastructure, along with follow-up actions to trigger based on the results of validation.

Follow these instructions to *Set up your first Checkpoint*.

By this point, you'll have your first, working deployment of Great Expectations. The next step is to *Customize your deployment*.

Data Contexts make this modular, so that you can add or swap out one component at a time. Most of these changes are quick, incremental steps—so you can upgrade from a basic demo deployment to a full production deployment at your own pace, and be confident that your Data Context will continue to work at every step along the way.

Table of contents for Getting Started:

2.2.1 Initialize a Data Context

In Great Expectations, your *Data Context* manages boilerplate configuration. Using a Data Context is almost always the fastest way to get up and running, even though some teams don't need every component of a Data Context.

Install Great Expectations

If you haven't already, install Great Expectations.

We recommend deploying within a virtual environment. If you're not familiar with virtual environments, pip, jupyter notebooks, or git, you may want to check out the [Supporting resources](#) section before continuing.

```
pip install great_expectations
```

To install from a git branch, use the following command (replace `develop` below with the name of the branch you want to use):

```
git clone https://github.com/great-expectations/great_expectations.git
cd great_expectations/
git checkout develop
pip install -e .
```

To install from a git fork, use the following command (replace `great-expectations` below with the name of the fork, which is usually your github username):

```
pip install -e .
git clone https://github.com/great-expectations/great_expectations.git
pip install great_expectations/
```

If you intend to develop within the Great Expectations (e.g. to contribute back to the project), check out [Setting up your dev environment](#) in the contributor documentation.

Download example data

For this tutorial, we will use a simplified version of the National Provider Identifier (NPI) database. It's a public dataset released by the [Centers of Medicare and Medicaid Services](#), intended as an authoritative list of health care providers in the United States. NPI data is famously messy—a great place to see the value of data testing and documentation in action.

To avoid confusion during the tutorial, we recommend you set up the following directory structure before you download the data:

```
mkdir example_project
mkdir example_project/my_data
cd example_project
```

To download the NPI data using `wget`, please run:

```
wget https://superconductive-public.s3.amazonaws.com/data/npi/weekly/npidata_pfile_
  ↪ 20200511-20200517.csv.gz -O my_data
```

Alternatively, you can use `curl`:

```
curl https://superconductive-public.s3.amazonaws.com/data/npi/weekly/npidata_pfile_
  ↪ 20200511-20200517.csv.gz -o my_data/npidata_pfile_20200511-20200517.csv.gz
```

Finally, to unzip the data, please run:

```
gunzip my_data/npidata_pfile_20200511-20200517.csv.gz
```

Once unzipped, the data should be 22MB on disk.

Run great_expectations init

When you installed Great Expectations, you also installed the Great Expectations *command line interface (CLI)*. It provides helpful utilities for deploying and configuring DataContexts, plus a few other convenience methods.

To initialize your Great Expectations deployment for the project, run this command in the terminal from the `example_dickens_data_project/` directory.

```
great_expectations init
```

You should see this:

~ Always know what to expect from your data ~

Let's configure a new Data Context.

First, Great Expectations will create a new directory:

```
great_expectations
|-- great_expectations.yml
|-- expectations
|-- checkpoints
|-- notebooks
|-- plugins
|-- .gitignore
|-- uncommitted
    |-- config_variables.yml
    |-- documentation
    |-- validations
```

OK to proceed? [Y/n]:

Let's pause there for a moment.

Once you finish going through `init`, your `great_expectations/` directory will contain all of the important components of a Great Expectations deployment, in miniature:

- `great_expectations.yml` will contain the main configuration your deployment.
- The `expectations/` directory will store all your *Expectations* as JSON files. If you want to store them somewhere else, you can change that later.
- The `notebooks/` directory is for helper notebooks to interact with Great Expectations.
- The `plugins/` directory will hold code for any custom plugins you develop as part of your deployment.
- The `uncommitted/` directory contains files that shouldn't live in version control. It has a `.gitignore` configured to exclude all its contents from version control. The main contents of the directory are:
 - `uncommitted/config_variables.yml`, which will hold sensitive information, such as database credentials and other secrets.
 - `uncommitted/documentation`, which will contains *Data Docs* generated from Expectations, Validation Results, and other metadata.
 - `uncommitted/validations`, which will hold *Validation Results* generated by Great Expectations.

Back in your terminal, go ahead and hit `Enter` to proceed.

2.2.2 Connect to data

Once you have a `DataContext`, you'll want to connect to data. In Great Expectations, *Datasources* simplify connections, by managing configuration and providing a consistent, cross-platform API for referencing data.

Let's configure your first Datasource, by following the next steps in the CLI init flow:

```
Would you like to configure a Datasource? [Y/n]:

What data would you like Great Expectations to connect to?
  1. Files on a filesystem (for processing with Pandas or Spark)
  2. Relational database (SQL)
: 1

What are you processing your files with?
  1. Pandas
  2. PySpark
: 1

Enter the path (relative or absolute) of the root directory where the data files are
→stored.
: my_data

Give your new Datasource a short name.
[my_data__dir]:

Great Expectations will now add a new Datasource 'my_data__dir' to your deployment,
by adding this entry to your great_expectations.yml:

my_data__dir:
  data_asset_type:
    class_name: PandasDataset
    module_name: great_expectations.dataset
  batch_kwargs_generators:
    subdir_reader:
      class_name: SubdirReaderBatchKwargsGenerator
      base_directory: ../my_data
  class_name: PandasDatasource

Would you like to proceed? [Y/n]:
```

That's it! You just configured your first Datasource!

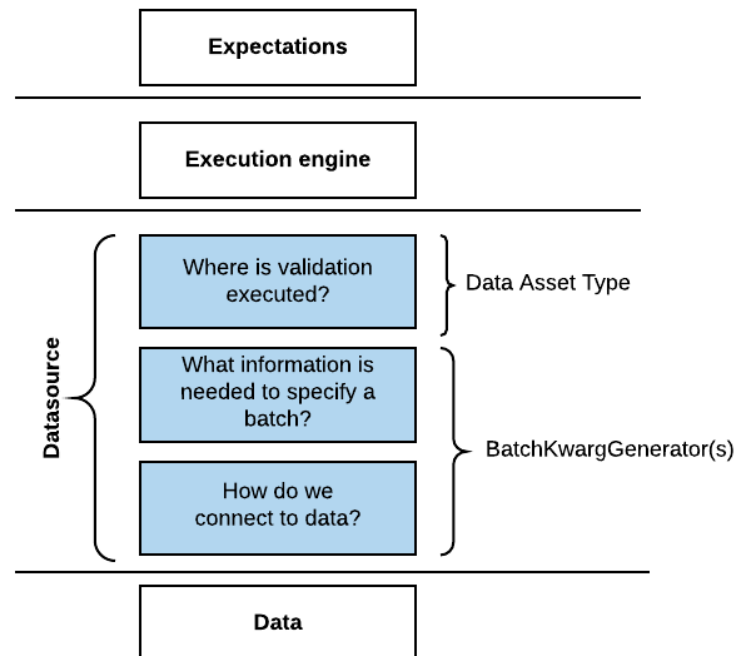
Before continuing, let's stop and unpack what just happened, and why.

Why Datasources?

Validation is the core operation in Great Expectations: “Validate X data against Y Expectations.”

Although the concept of data validation is simple, carrying it out can require complex engineering. This is because your Expectations and data might be stored in different places, and the computational resources for validation might live somewhere else entirely. The engineering cost of building the necessary connectors for validation has been one of the major things preventing data teams from testing their data.

Datasources solve this problem, by conceptually separating *what* you want to validate from *how* you want to validate it. Datasources give you full control over the process of bringing data and Expectations together, then abstract away that underlying complexity when you validate X data against Y Expectations.



We call the layer that handles the actual computation an Execution Engine. Currently, Great Expectations supports three Execution Engines: pandas, sqlalchemy, and pyspark. We plan to extend the library to support others in the future.

The layer that handles connecting to data is called a *Batch Kwarg Generator*. Not all Batch Kwarg Generators can be used with all Execution Engines. It's also possible to configure a Datasource without a Batch Kwarg Generator if you want to pass data in directly. However, Batch Kwarg Generators are required to get the most out of Great Expectations' *Data Docs* and *Profilers*.

You can read more about the inner workings of Datasources, Execution Engines, and Batch Kwarg Generators [here](#).

Attention: We plan to upgrade this configuration API with better names and more conceptual clarity prior to Great Expectations' 1.0 release. If at all possible, we will make those changes in a non-breaking way. If you have ideas, concerns or questions about this planned improvement, please join the public discussion [here](#).

Configuring Datasources

When you completed those last few steps in `great_expectations init`, you told Great Expectations that

1. You want to create a new Datasource called `my_data__dir`.
2. You want to use Pandas as your Execution Engine, hence `data_asset_type.class_name = PandasDataset`.
3. You want to create a BatchKwarg Generator called `subdir_reader` using the class `SubdirReaderBatchKwargsGenerator`.
4. This particular Generator connects to data in files within a local directory, specified here as `../my_data`.

Based on that information, the CLI added the following entry into your `great_expectations.yml` file, under the `datasources` header:

```
my_data__dir:
  data_asset_type:
    class_name: PandasDataset
    module_name: great_expectations.dataset
  batch_kwarg_generators:
    subdir_reader:
      class_name: SubdirReaderBatchKwargsGenerator
      base_directory: ../my_data
      class_name: PandasDatasource
```

In the future, you can modify or delete your configuration by editing your `great_expectations.yml` file directly. For instructions on how to configure various Datasources, check out [How-to guides for configuring Datasources](#).

For now, let's continue to [Create your first Expectations](#).

2.2.3 Create your first Expectations

Expectations are the workhorse abstraction in Great Expectations.

Each Expectation is a declarative, machine-verifiable assertion about the expected format, content, or behavior of your data. Great Expectations comes with *dozens of built-in Expectations*, and it's easy to *develop your own custom Expectations*, too.

Admonition from Mr. Dickens.

“Take nothing on its looks; take everything on evidence. There's no better rule.”

The CLI will help you create your first Expectations. You can accept the defaults by typing [Enter] twice:

```
Would you like to profile new Expectations for a single data asset within your new_
↳ Datasource? [Y/n]:

Would you like to:
  1. choose from a list of data assets in this datasource
  2. enter the path of a data file
: 1

Which data would you like to use?
  1. npidata_pfile_20200511-20200517 (file)
```

(continues on next page)

(continued from previous page)

```
Don't see the name of the data asset in the list above? Just type it
: 1

Name the new Expectation Suite [npidata_pfile_20200511-20200517.warning]:

Great Expectations will choose a couple of columns and generate expectations about
→them
to demonstrate some examples of assertions you can make about your data.

Great Expectations will store these expectations in a new Expectation Suite 'npidata_
→pfile_20200511-20200517.warning' here:

file:///home/ubuntu/example_project/great_expectations/expectations/npidata_pfile_
→20200511-20200517/warning.json

Would you like to proceed? [Y/n]:

Generating example Expectation Suite...

Done generating example Expectation Suite
```

What just happened?

You can create and edit Expectations using several different workflows. The CLI just used one of the quickest and simplest: scaffolding Expectations using an automated *Profiler*.

This Profiler connected to your data (using the Datasource you configured in the previous step), took a quick look at the contents, and produced an initial set of Expectations. These Expectations are not intended to be very smart. Instead, the goal is to quickly provide some good examples, so that you're not starting from a blank slate.

Later, you should also take a look at other workflows for *Creating and editing Expectations*. Creating and editing Expectations is a very active area of work in the Great Expectations community. Stay tuned for improvements over time.

Note: the Profiler also validated the source data using the new Expectations, producing a set of *Validation Results*. We'll explain why in the next step of the tutorial.

A first look at real Expectations

The newly profiled Expectations are stored in an Expectation Suite.

For now, they're stored in a JSON file in a subdirectory subdirectory of your `great_expectations/` folder. You can also configure Great Expectations to store Expectations to other locations, like S3, postgresql, etc. We'll come back to these options in the last step of the tutorial.

If you open up the suite in `great_expectations/expectations/something-something.json` in a text editor, you'll see:

```
{
  "data_asset_type": "Dataset",
  "expectation_suite_name": "npidata_pfile_20200511-20200517.warning",
  "expectations": [
    {
      "expectation_type": "expect_table_row_count_to_be_between",
      "kwargs": {
```

(continues on next page)

(continued from previous page)

```
        "max_value": 20884,
        "min_value": 17087
    },
    "meta": {
        "BasicSuiteBuilderProfiler": {
            "confidence": "very low"
        }
    }
},
{
    "expectation_type": "expect_table_column_count_to_equal",
    "kwargs": {
        "value": 330
    },
    "meta": {
        "BasicSuiteBuilderProfiler": {
            "confidence": "very low"
        }
    }
},
{
    "expectation_type": "expect_table_columns_to_match_ordered_list",
    "kwargs": {
        "column_list": [
            "NPI",
            "Entity Type Code",
            "Replacement NPI",
            "Employer Identification Number (EIN)",
            "Provider Organization Name (Legal Business Name)",
            "Provider Last Name (Legal Name)",
            "Provider First Name",
            "Provider Middle Name",
            "Provider Name Prefix Text",
            "Provider Name Suffix Text",
            "Provider Credential Text",
            ...
        ]
    }
}
```

There's a lot of information here. (This is good.)

Every Expectation in the file expresses a test that can be validated against data. (This is very good.)

We were able to generate all of this information very quickly. (Also good.)

However, most human beings find that dense JSON objects are very hard to read. (This is bad.)

In the next step of the tutorial, we'll show how to convert Expectations into more human-friendly formats: [Set up Data Docs](#).

2.2.4 Set up Data Docs

Data Docs translate *Expectations*, *Validation Results*, and other metadata into clean, human-readable documentation. Automatically compiling your data documentation from your data tests guarantees that your documentation will never go stale—a huge workflow change from the sprawling, chronically-out-of-date data wikis that many data teams otherwise have to maintain.

Go ahead and tell the CLI you want to build Data Docs:

```
Would you like to build Data Docs? [Y/n]:

The following Data Docs sites will be built:

- local_site: file:///home/ubuntu/example_project/great_expectations/uncommitted/
  ↳ data_docs/local_site/index.html

Would you like to proceed? [Y/n]:

Building Data Docs...

Done building Data Docs

Would you like to view your new Expectations in Data Docs? This will open a new
  ↳ browser window. [Y/n]:
```

What just happened?

In the previous step, our data Profiler generated a set of new Expectations. We just used Data Docs to compile those Expectations to HTML, which you can see here. This page is *prescriptive*: it describes how data *should* look:

When you open the window, you'll see a static website that looks like this:

In addition, your static site includes a set of pages for future data validations. These pages are *diagnostic*: they describe whether and how a specific batch of data differed from the ideal prescribed by a set of Expectations.

For now, your static site is built and stored locally. In the last step of the tutorial, we'll explain options for configuring, hosting and sharing it.

The Data Docs build chain

There's quite a bit going on under the hood here, but for the moment the details don't matter. You can think of Data Docs as an elaborate makefile. The steps in the makefile are called Renderers. Their inputs are Expectations, Validation Results, and other metadata. Their collective output is a static website.

Possibilities to keep in mind for the future:

- **The same inputs can be rendered more than once.** For example, you might want to use the same Expectation in a bullet point, and as an entry within a data dictionary on a different page.
- **You can build more than one target.** For example, if you share data outside your company, you might want one site to support internal engineering and analytics, and another site to support downstream clients consuming the data.
- **The build target for rendering does not need to be HTML.** For example, you can use Renderers to generate Slack messages and email notifications.
- **All components of the build chain are pluggable and extensible.** Like everything else in Great Expectations, you can customize and extend Renderers.



great_expectations

[Home](#) / [npidata_pfile_20200511-20200517.warning](#)

Expectation Suite

A collection of
Expectations defined for
batches of data.

Actions

[How to Edit This Suite](#)[Show Walkthrough](#)

Table of Contents

[Overview](#)[Entity Type Code](#)[NPI](#)[Provider Organization Name \(Legal Business Name\)](#)

Overview

Info

Expectation Suite Name	npidata_pfile_20200511-20200517.warning
Great Expectations Version	0.11.0

Table-Level Expectations

- Must have between **17087** and **20884** rows.
- Must have exactly **330** columns.
- Must have these columns in this order: **NPI**, **Entity Type Code**, **Replacement NPI**, **Employer Identification Number (EIN)**, **Provider Organization Name (Legal Business Name)**, **Provider Last Name (Legal Name)**, **Provider First Name**, **Provider Middle Name**, **Provider Name Prefix Text**, **Provider Name Suffix Text**, **Provider Credential Text**, **Provider Other Organization Name**, **Provider Other Organization Name Type Code**, **Provider Other Last Name**, **Provider Other First Name**, **Provider Other Middle Name**, **Provider Other Name Prefix Text**, **Provider Other Name Suffix Text**, **Provider Other Credential Text**, **Provider Other Last Name Type Code**, **Provider First Line Business Mailing Address**, **Provider Second Line Business Mailing Address**, **Provider Business Mailing Address City Name**, **Provider Business Mailing Address State Name**, **Provider Business Mailing Address Postal Code**, **Provider Business Mailing Address Country Code (If outside U.S.)**, **Provider Business Mailing Address Telephone Number**, **Provider Business Mailing Address Fax Number**, **Provider First Line Business Practice Location Address**, **Provider Second Line Business Practice Location Address**, **Provider Business Practice Location Address City Name**, **Provider Business Practice Location Address State Name**, **Provider Business Practice Location Address Postal Code**, **Provider Business Practice Location Address Country Code (If outside U.S.)**, **Provider Business Practice Location Address Telephone Number**, **Provider Business Practice Location Address Fax Number**, **Provider Enumeration Date**, **Last Update Date**, **NPI Deactivation Reason Code**, **NPI Deactivation Date**, **NPI Reactivation Date**, **Provider Gender Code**, **Authorized Official Last Name**, **Authorized Official First Name**, **Authorized Official Middle Name**, **Authorized Official Title or Position**, **Authorized Official Telephone Number**, **Healthcare Provider Taxonomy Code_1**, **Provider License Number_1**, **Provider License Number State Code_1**, **Healthcare Provider Primary Taxonomy Switch_1**, **Healthcare Provider Taxonomy Code_2**, **Provider License Number_2**, **Provider License Number State Code_2**, **Healthcare Provider Primary Taxonomy Switch_2**, **Healthcare Provider Taxonomy Code_3**, **Provider License Number_3**, **Provider License Number State Code_3**, **Healthcare Provider Primary Taxonomy Switch_3**, **Healthcare Provider Taxonomy Code_4**, **Provider License Number_4**, **Provider License Number State Code_4**, **Healthcare Provider Primary Taxonomy Switch_4**, **Healthcare Provider Taxonomy Code_5**, **Provider License Number_5**, **Provider License Number State Code_5**, **Healthcare Provider Primary Taxonomy Switch_5**, **Healthcare Provider Taxonomy Code_6**, **Provider License Number_6**, **Provider License Number State Code_6**, **Healthcare Provider Primary Taxonomy Switch_6**, **Healthcare Provider Taxonomy Code_7**, **Provider License Number_7**, **Provider License Number State Code_7**, **Healthcare Provider Primary Taxonomy Switch_7**, **Healthcare Provider Taxonomy Code_8**, **Provider License Number_8**, **Provider License Number State Code_8**, **Healthcare Provider Primary Taxonomy Switch_8**, **Healthcare Provider Taxonomy Code_9**, **Provider License Number_9**, **Provider License Number State Code_9**, **Healthcare Provider Primary Taxonomy Switch_9**, **Healthcare Provider Taxonomy Code_10**, **Provider License Number_10**, **Provider License Number State Code_10**, **Healthcare Provider Primary Taxonomy Switch_10**, **Healthcare Provider Taxonomy Code_11**, **Provider License Number_11**.



great_expectations

[Home](#)[/ npidata_pfile_20200511-20200517.warning](#) / [20200523T154958.899489Z](#) / [2020-05-23T15:49:58.899489+00:00](#) / [67b998dad0705e7fbb6ae324afdc7298](#)

Expectation Validation Result

Evaluates whether a batch
of data matches
expectations.

Actions

Validation Filter:

[Show All](#) [Failed Only](#)[How to Edit This Suite](#)[Show Walkthrough](#)

Table of Contents

[Overview](#)[Table-Level Expectations](#)[Entity Type Code](#)[NPI](#)[Provider Organization Name \(Legal Business Name\)](#)

Overview

Expectation Suite: [npidata_pfile_20200511-20200517.warning](#)Status: ✔ Succeeded

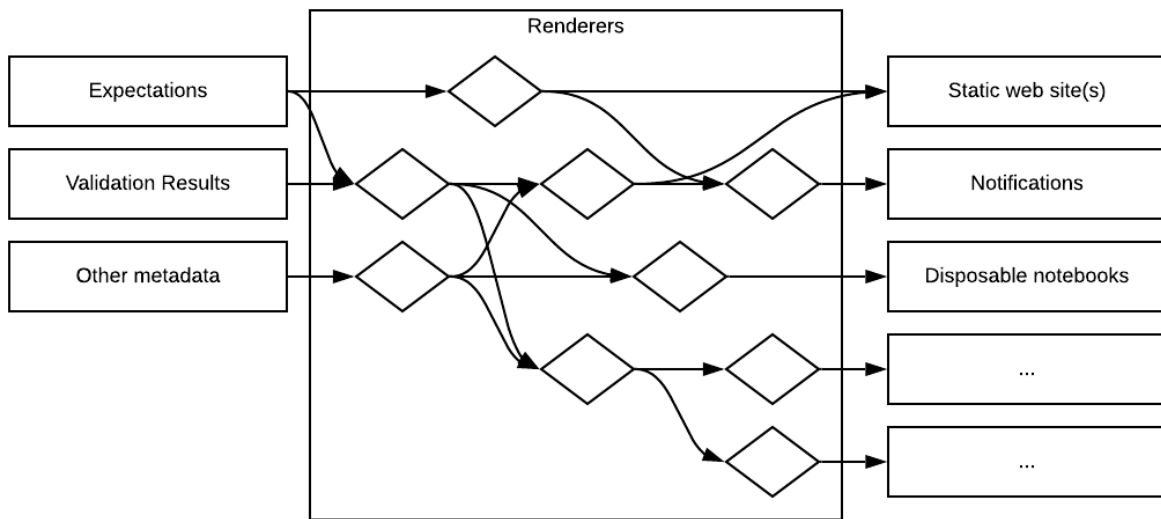
Statistics

Evaluated Expectations	14
Successful Expectations	14
Unsuccessful Expectations	0
Success Percent	100%

[Show more info...](#)

Table-Level Expectations

Status	Expectation	Observed Value
✔	Must have between 17087 and 20884 rows.	18986
✔	Must have exactly 330 columns.	330
	Must have these columns in this order: NPI , Entity Type Code , Replacement NPI , Employer Identification Number (EIN) , Provider Organization Name (Legal Business Name) , Provider Last Name (Legal Name) , Provider First Name , Provider Middle Name , Provider Name Prefix Text , Provider Name Suffix Text , Provider Credential Text , Provider Other Organization Name , Provider Other Organization Name Type Code , Provider Other Last Name .	['NPI', 'Entity Type Code', 'Replacement NPI', 'Employer Identification Number (EIN)', 'Provider Organization Name (Legal Business Name)', 'Provider Last Name (Legal Name)', 'Provider First Name', 'Provider Middle Name', 'Provider Name Prefix Text', 'Provider Name Suffix Text', 'Provider Credential Text', 'Provider Other Organization Name', 'Provider Other Organization Name Type Code', 'Provider Other Last Name', 'Provider Other First Name', 'Provider Other Middle Name', 'Provider Other Name']



This ability to translate between machine- and human-readable formats makes Great Expectations a powerful tool—not just for data testing, but also data communication and collaboration.

Attention: Great Expectations Renderers are still in beta. They’re very useful, but expect changes in appearance, behavior, and API.

In the next step, we will complete set up by showing you how to Validate data, review the results, and edit your Expectations. Incidentally, this step will use another kind of renderer: a disposable notebook generated specifically to support this workflow.

2.2.5 Set up your first Checkpoint

As we said earlier, validation the core operation of Great Expectations: “Validate X data against Y Expectations.”

In normal usage, the best way to validate data is with a Checkpoint. Checkpoints bring Batches of data together with corresponding *Expectation Suites* for validation. Configuring Checkpoints simplifies deployment, by pre-specifying the “X”s and “Y”s that you want to validate at any given point in your data infrastructure.

Let’s set up our first Checkpoint by running another CLI command:

```
great_expectations checkpoint new my_checkpoint npidata_pfile_20200511-20200517.
↪warning.json
```

`my_checkpoint` will be the name of your new Checkpoint. It will use `npidata_pfile_20200511-20200517.warning` as its primary Expectation Suite. (You can add other Expectation Suites later.)

From there, you can configure the Checkpoint using the CLI:

```
Heads up! This feature is Experimental. It may change. Please give us your feedback!

Would you like to:
  1. choose from a list of data assets in this datasource
```

(continues on next page)

(continued from previous page)

```

2. enter the path of a data file
: 1

Which data would you like to use?
1. npidata_pfile_20200511-20200517 (file)
Don't see the name of the data asset in the list above? Just type it
: 1
A checkpoint named `my_checkpoint` was added to your project!
- To edit this checkpoint edit the checkpoint file: /home/ubuntu/example_project/
→great_expectations/checkpoints/my_checkpoint.yml
- To run this checkpoint run `great_expectations checkpoint run my_checkpoint`

```

Let's pause there before continuing.

How Checkpoints work

Your new checkpoint file is in `my_checkpoint.yml`. With comments removed, it looks like this:

```

validation_operator_name: action_list_operator
batches:
- batch_kwargs:
    path: /home/ubuntu/example_project/great_expectations/../../my_data/npidata_pfile_
→20200511-20200517.csv
    datasource: my_data_dir
    data_asset_name: npidata_pfile_20200511-20200517
    expectation_suite_names: # one or more suites may validate against a single batch
- npidata_pfile_20200511-20200517.warning

```

Our newly configured Checkpoint knows how to load `npidata_pfile_20200511-20200517.csv` as a Batch, pair it with the `npidata_pfile_20200511-20200517.warning` Expectation Suite, and execute them both using a pre-configured *Validation Operator* called `action_list_operator`.

You don't need to worry much about the details of Validation Operators for now. They orchestrate the actual work of validating data and processing the results. After executing validation, the Validation Operator can kick off additional workflows through Validation Actions.

You can see the configuration for `action_list_operator` in your `great_expectations.yml` file. With comments removed, it looks like this:

```

action_list_operator:
  class_name: ActionListValidationOperator
  action_list:
    - name: store_validation_result
      action:
        class_name: StoreValidationResultAction
    - name: store_evaluation_params
      action:
        class_name: StoreEvaluationParametersAction
    - name: update_data_docs
      action:
        class_name: UpdateDataDocsAction
  # - name: send_slack_notification_on_validation_result
  #   action:
  #     class_name: SlackNotificationAction
  #     # put the actual webhook URL in the uncommitted/config_variables.yml file
  #     slack_webhook: ${validation_notification_slack_webhook}

```

(continues on next page)

(continued from previous page)

```
#     notify_on: all # possible values: "all", "failure", "success"
#     renderer:
#         module_name: great_expectations.render.renderer.slack_renderer
#         class_name: SlackRenderer
```

You can see that the `action_list` for your validation Operator contains three Validation Actions. After each run using this operator...

1. `store_validation_result`: store the *Validation Results*.
2. `store_evaluation_params`: store *Evaluation Parameters*.
3. `update_data_docs`: update your *Data Docs*.

A fourth action, `send_slack_notification_on_validation_result`, will trigger a notification in Slack. It's currently commented out. See [How to trigger Slack notifications as a Validation Action](#) to configure it.

For more examples of post-validation actions, please see the [How-to section for Validation](#).

How to run Checkpoints

Checkpoints can be run like applications from the command line or cron:

```
great_expectations checkpoint run my_checkpoint
```

You can also generate Checkpoint scripts that you can edit and run using python, or within data orchestration tools like airflow.

```
great_expectations checkpoint script my_checkpoint
```

Now that you know how to configure Checkpoints, let's proceed to the last step of the tutorial: [Customize your deployment](#).

2.2.6 Customize your deployment

At this point, you have your first, working deployment of Great Expectations. You've also been introduced to the foundational concepts in the library: Data Contexts, *Datasources*, *Expectations*, *Profilers*, *Data Docs*, *Validation*, and Checkpoints.

Congratulations! You're off to a very good start.

The next step is to customize your deployment by upgrading specific components of your deployment. Data Contexts make this modular, so that you can add or swap out one component at a time. Most of these changes are quick, incremental steps—so you can upgrade from a basic demo deployment to a full production deployment at your own pace and be confident that your Data Context will continue to work at every step along the way.

This last section of the [Getting started](#) tutorial is designed to present you with clear options for upgrading your deployment. For specific implementation steps, please check out the linked [How-to guides](#).

Components

Here's an overview of the components of a typical Great Expectations deployment:

- Great Expectations configs and metadata
 - *Options for storing Great Expectations configuration*
 - *Options for storing Expectations*
 - *Options for storing Validation Results*
- Integrations to related systems
 - *Additional DataSources and Generators*
 - *Options for hosting Data Docs*
 - *Additional Validation Operators and Actions*
 - *Options for triggering Validation*

Options for storing Great Expectations configuration

The simplest way to manage your Great Expectations configuration is usually by committing `great_expectations/great_expectations.yml` to git. However, it's not usually a good idea to commit credentials to source control. In some situations, you might need to deploy without access to source control (or maybe even a file system).

Here's how to handle each of those cases:

- How to use environment variables to populate credentials
- *How to populate credentials from a secrets store*
- *How to instantiate a Data Context without a yml file*

Options for storing Expectations

Many teams find it convenient to store Expectations in git. Essentially, this approach treats Expectations like test fixtures: they live adjacent to code and are stored within version control. git acts as a collaboration tool and source of record.

Alternatively, you can treat Expectations like configs, and store them in a blob store. Finally, you can store them in a database.

- *How to configure an Expectation store in Amazon S3*
- *How to configure an Expectation store in GCS*
- *How to configure an Expectation store in Azure blob storage*
- *How to configure an Expectation store to postgresql*

Options for storing Validation Results

By default, Validation Results are stored locally, in an uncommitted directory. This is great for individual work, but not good for collaboration. The most common pattern is to use a cloud-based blob store such as S3, GCS, or Azure blob store. You can also store Validation Results in a database.

- *How to configure a Validation Result store on a filesystem*
- *How to configure a Validation Result store in S3*
- *How to configure a Validation Result store in GCS*
- *How to configure a Validation Result store in Azure blob storage*
- *How to configure a Validation Result store to postgresql*

Additional DataSources and Generators

Great Expectations plugs into a wide variety of Datasources, and the list is constantly getting longer. If you have an idea for a Datasource not listed here, please speak up in [the public discussion forum](#).

- *How to configure a Pandas/filesystem Datasource*
- *How to configure a Pandas/S3 Datasource*
- *How to configure a Redshift Datasource*
- *How to configure a Snowflake Datasource*
- *How to configure a BigQuery Datasource*
- *How to configure a Databricks Azure Datasource*
- *How to configure an EMR Spark Datasource*
- *How to configure a Databricks AWS Datasource*
- *How to configure a self managed Spark Datasource*

Options for hosting Data Docs

By default, Data Docs are stored locally, in an uncommitted directory. This is great for individual work, but not good for collaboration. A better pattern is usually to deploy to a cloud-based blob store (S3, GCS, or Azure blob store), configured to share a static website.

- *How to host and share Data Docs on a filesystem*
- *How to host and share Data Docs on S3*
- *How to host and share Data Docs on Azure Blob Storage*
- *How to host and share Data Docs on GCS*

Additional Validation Operators and Actions

Most teams will want to configure various Validation Actions as part of their deployment.

- [How to re-render Data Docs as a Validation Action](#)
- [How to store Validation Results as a Validation Action](#)
- [How to trigger Slack notifications as a Validation Action](#)

Modifying *Validation Operators* themselves is more advanced work. You can learn how here.

- [How to configure a Validation Operator](#)
- [How to configure a WarningAndFailureExpectationSuitesValidationOperator](#)
- [How to configure an ActionListValidationOperator](#)
- [How to implement a custom Validation Operator](#)

Options for triggering Validation

There are two primary patterns for deploying Checkpoints. Sometimes Checkpoints are executed during data processing (e.g. as a task within Airflow). From this vantage point, they can control program flow. Sometimes Checkpoints are executed against materialized data. Great Expectations supports both patterns. There are also some rare instances where you may want to validate data without using a Checkpoint.

- [How to run a Checkpoint in Airflow](#)
- [How to run a Checkpoint in python](#)
- [How to run a Checkpoint in terminal](#)
- [How to validate data without a Checkpoint](#)

2.3 Contribute to Great Expectations

Welcome to the Great Expectations project! We're very glad you want to help out by contributing.

Our goal is to make your experience as great as possible. Please follow these steps to contribute:

Go to greatexpectations.io/slack and introduce yourself in the `#contributors` channel.

Follow [these instructions](#) to set up your dev environment.

Alternatively, for small changes that don't need to be tested locally (e.g. documentation changes), you can [make changes directly through Github](#).

Issues in GitHub are a great place to start. Check out the [help wanted](#) and [good first issue](#) labels. Comment to let everyone know you're working on it.

If there's no issue for what you want to work on, please create one. Add a comment to let everyone know that you're working on it. We prefer small, incremental commits, because it makes the thought process behind changes easier to review.

When your changes are ready, run through our [Contribution checklist](#) for pull requests.

Note that if it's your first contribution, there is a standard [Contributor license agreement \(CLA\)](#) to sign.

HOW-TO GUIDES

Warning: We are working on improving our How-to Guides, but many of them are just stubs. Please check the “spare parts” sections or feel free to review older versions of documentation while we migrate content.

3.1 Configuring Data Contexts

3.1.1 How to create a new Data Context with the CLI

We recommend that you create new Data Contexts by using the `great_expectations init` command in the directory where you want to deploy Great Expectations.

```
great_expectations init
```

Please see the [Getting Started](#) section for a more detailed tutorial on Data Contexts and getting a new project up and running.

3.1.2 How to use a YAML file or environment variables to populate credentials

This guide will explain how to use a YAML file and/or environment variables to populate credentials (or any value) in your `great_expectations.yml` project config.

Prerequisites: This how-to guide assumes you have already:

- [Set up a working deployment of Great Expectations](#)
-

Steps

1. Decide where you would like to save the desired credentials or config values - in a YAML file, environment variables, or a combination - then save the values. In most cases, we suggest using a config variables YAML file. YAML files make variables more visible, easily editable, and allow for modularization (e.g. one file for dev, another for prod).

Note:

- In the `great_expectations.yml` config file, environment variables take precedence over variables defined in a config variables YAML

- Environment variable substitution is also supported in the `great_expectations.yml` config file.

If using a YAML file, save desired credentials or config values to `great_expectations/uncommitted/config_variables.yml` or another YAML file of your choosing:

```
# great_expectations/uncommitted/config_variables.yml

my_postgres_db_yaml_creds:
  drivename: postgres
  host: 127.0.0.778
  port: '7987'
  username: administrator
  password: ${MY_DB_PW}
  database: postgres
```

If using environment variables, set values by entering `export ENV_VAR_NAME=env_var_value` in the terminal or adding the commands to your `~/ .bashrc` file:

```
export POSTGRES_DRIVENAME=postgres
export POSTGRES_HOST=localhost
export POSTGRES_PORT='5432'
export POSTGRES_USERNAME=postgres
export POSTGRES_PW=''
export POSTGRES_DB=postgres
export MY_DB_PW=password
```

2. If using a YAML file, set the `config_variables_file_path` key in your `great_expectations.yml` or leave the default.

```
# great_expectations/great_expectations.yml

config_variables_file_path: uncommitted/config_variables.yml
```

3. Replace credentials or other values in your `great_expectations.yml` with `${}`-wrapped variable names (i.e. `${ENVIRONMENT_VARIABLE}` or `${YAML_KEY}`).

```
# great_expectations/great_expectations.yml

datasources:
  my_postgres_db:
    class_name: SqlAlchemyDatasource
    data_asset_type:
      class_name: SqlAlchemyDataset
      module_name: great_expectations.dataset
    module_name: great_expectations.datasource
    credentials: ${my_postgres_db_yaml_creds}
  my_other_postgres_db:
    class_name: SqlAlchemyDatasource
    data_asset_type:
      class_name: SqlAlchemyDataset
      module_name: great_expectations.dataset
    module_name: great_expectations.datasource
    credentials:
      drivename: ${POSTGRES_DRIVENAME}
      host: ${POSTGRES_HOST}
      port: ${POSTGRES_PORT}
```

(continues on next page)

(continued from previous page)

```
username: ${POSTGRES_USERNAME}
password: ${POSTGRES_PW}
database: ${POSTGRES_DB}
```

Additional Notes

- The default `config_variables.yml` file located at `great_expectations/uncommitted/config_variables.yml` applies to deployments created using `great_expectations init`.

3.1.3 How to populate credentials from a secrets store

Admonition from Mr. Dickens

“Whether I shall turn out to be the hero of my own life, or whether that station will be held by anybody else, these pages must show.”

This guide is a stub. We all know that it will be useful, but no one has made time to write it yet.

If it would be useful to you, please comment with a +1 and feel free to add any suggestions or questions below.

If you want to be a real hero, we’d welcome a pull request. Please see [the Contributing tutorial](#) and [How to write a how to guide](#) to get started.

3.1.4 How to instantiate a Data Context without a yml file

Admonition from Mr. Dickens

“Whether I shall turn out to be the hero of my own life, or whether that station will be held by anybody else, these pages must show.”

This guide is a stub. We all know that it will be useful, but no one has made time to write it yet.

If it would be useful to you, please comment with a +1 and feel free to add any suggestions or questions below.

If you want to be a real hero, we’d welcome a pull request. Please see [the Contributing tutorial](#) and [How to write a how to guide](#) to get started.

3.2 Configuring Datasources

3.2.1 How to configure a BigQuery Datasource

This guide will help you add a BigQuery project (or a dataset) as a Datasource. This will allow you to validate tables and queries within this project. When you use a BigQuery Datasource, the validation is done in BigQuery itself. Your data is not downloaded.

Prerequisites: This how-to guide assumes you have already:

- *Set up a working deployment of Great Expectations*

- Followed the [Google Cloud library guide](#) for authentication
 - Installed the pybigquery package for the BigQuery sqlalchemy dialect (`pip install pybigquery`)
-

Steps

1. Run the following CLI command to begin the interactive Datasource creation process:

```
great_expectations datasource new
```

2. Choose “Big Query” from the list of database engines, when prompted.
3. Identify the connection string you would like Great Expectations to use to connect to BigQuery, using the examples below and the [PyBigQuery](#) documentation.

If you want Great Expectations to connect to your BigQuery project (without specifying a particular dataset), the URL should be:

```
bigquery://project-name
```

If you want Great Expectations to connect to a particular dataset inside your BigQuery project, the URL should be:

```
bigquery://project-name/dataset-name
```

If you want Great Expectations to connect to one of the Google’s public datasets, the URL should be:

```
bigquery://project-name/bigquery-public-data
```

5. Enter the connection string when prompted (and press Enter when asked “Would you like to proceed? [Y/n]:”).
6. Should you need to modify your connection string, you can manually edit the `great_expectations/uncommitted/config_variables.yml` file.

Additional notes

Environment variables can be used to store the SQLAlchemy URL instead of the file, if preferred - search documentation for “Managing Environment and Secrets”.

Additional resources

3.2.2 How to configure a Databricks AWS Datasource

Admonition from Mr. Dickens

“Whether I shall turn out to be the hero of my own life, or whether that station will be held by anybody else, these pages must show.”

This guide is a stub. We all know that it will be useful, but no one has made time to write it yet.

If it would be useful to you, please comment with a +1 and feel free to add any suggestions or questions below.

If you want to be a real hero, we'd welcome a pull request. Please see [the Contributing tutorial](#) and [How to write a how to guide](#) to get started.

3.2.3 How to configure a Databricks Azure Datasource

Admonition from Mr. Dickens

“Whether I shall turn out to be the hero of my own life, or whether that station will be held by anybody else, these pages must show.”

This guide is a stub. We all know that it will be useful, but no one has made time to write it yet.

If it would be useful to you, please comment with a +1 and feel free to add any suggestions or questions below.

If you want to be a real hero, we'd welcome a pull request. Please see [the Contributing tutorial](#) and [How to write a how to guide](#) to get started.

3.2.4 How to configure a Pandas/filesystem Datasource

This guide shows how to connect to a Pandas Datasource such that the data is accessible in the form of files on a local or NFS type of a filesystem.

Prerequisites: This how-to guide assumes you have already:

- [Set up a working deployment of Great Expectations](#)
-

Steps

To add a filesystem-backed Pandas datasource do this:

1. Run `datasource new`

From the command line, run:

```
great_expectations datasource new
```

2. Choose “Files on a filesystem (for processing with Pandas or Spark)”

```
What data would you like Great Expectations to connect to?
  1. Files on a filesystem (for processing with Pandas or Spark)
  2. Relational database (SQL)
: 1
```

3. Choose Pandas

```
What are you processing your files with?
  1. Pandas
  2. PySpark
: 1
```

4. Specify the directory path for data files

```
Enter the path (relative or absolute) of the root directory where the ↵
↵data files are stored.
: /path/to/directory/containing/your/data/files
```

5. Give your Datasource a name

When prompted, provide a custom name for your filesystem-backed Pandas data source, or hit Enter to accept the default.

```
Give your new Datasource a short name.
[my_data_files_dir]:
```

Great Expectations will now add a new Datasource 'my_data_files_dir' to your deployment, by adding this entry to your `great_expectations.yml`:

```
my_data_files_dir:
  data_asset_type:
    class_name: PandasDataset
    module_name: great_expectations.dataset
  batch_kwargs_generators:
    subdir_reader:
      class_name: SubdirReaderBatchKwargsGenerator
      base_directory: /path/to/directory/containing/your/data/files
      class_name: PandasDatasource

Would you like to proceed? [Y/n]:
```

6. Wait for confirmation

If all goes well, it will be followed by the message:

```
A new datasource 'my_data_files_dir' was added to your project.
```

If you run into an error, you will see something like:

```
Error: Directory '/nonexistent/path/to/directory/containing/your/data/
↵files' does not exist.

Enter the path (relative or absolute) of the root directory where the ↵
↵data files are stored.
:
```

In this case, please check your data directory path, permissions, etc. and try again.

7. Finally, if all goes well and you receive a confirmation on your Terminal screen, you can proceed with exploring the data sets in your new filesystem-backed Pandas data source.

Additional Notes

1. Relative path locations should be specified from the perspective of the directory, in which the

```
great_expectations datasource new
```

command is executed.

Comments

3.2.5 How to configure a Pandas/S3 Datasource

This guide shows how to connect to a Pandas Datasource such that the data is accessible in the form of files located on the AWS S3 service.

Prerequisites: This how-to guide assumes you have already:

- *Set up a working deployment of Great Expectations*

Steps

To add an S3-backed Pandas datasource do this:

1. Edit your `great_expectations/great_expectations.yml` file

Update your `datasources:` section to include a `PandasDatasource`.

```
datasources:
  pandas_s3:
    class_name: PandasDatasource
```

2. Load data from S3 using native S3 path-based Batch Kwargs.

Because Pandas provides native support for reading from S3 paths, this simple configuration will allow loading datasources from S3 using native S3 paths.

```
context = DataContext()
batch_kwargs = {
    "datasource": "pandas_s3",
    "path": "s3a://my_bucket/my_prefix/key.csv",
}
batch = context.get_batch(batch_kwargs, "existing_expectation_suite_name")
```

3. Optionally, configure a `BatchKwargsGenerator` that will allow you to generate Data Assets and Partitions from your S3 bucket.

Update your datasource configuration to include the new Batch Kwargs Generator:

```
datasources:
  pandas_s3:
    class_name: PandasDatasource
    batch_kwargs_generators:
      pandas_s3_generator:
        class_name: S3GlobReaderBatchKwargsGenerator
        bucket: your_s3_bucket # Only the bucket name here (i.e., no_
↪prefix)
    assets:
      your_first_data_asset_name:
        prefix: prefix_to_folder_containing_your_first_data_asset_
↪files/ # trailing slash is important
        regex_filter: .* # The regex filter will filter the results_
↪returned by S3 for the key and prefix to only those matching the regex
      your_second_data_asset_name:
        prefix: prefix_to_folder_containing_your_second_data_asset_
↪files/ # trailing slash is important
```

(continues on next page)

(continued from previous page)

```

        regex_filter: .* # The regex filter will filter the results_
↪returned by S3 for the key and prefix to only those matching the regex
        your_third_data_asset_name:
            prefix: prefix_to_folder_containing_your_third_data_asset_
↪files/ # trailing slash is important
            regex_filter: .* # The regex filter will filter the results_
↪returned by S3 for the key and prefix to only those matching the regex
        module_name: great_expectations.datasource
        data_asset_type:
            class_name: PandasDataset
            module_name: great_expectations.dataset

```

Update the configuration of the `assets:` section to reflect your project's data storage system. There is no limit on the number of data assets, but you should only keep the ones that are actually used in the configuration file (i.e., delete the unused ones from the above template and/or add as many as needed for your project).

Note: Multiple data sources can easily be configured in the Data Context by adding a new configuration block for each in the data sources section. Each data source name should be at the same level of indentation.

4. Optionally, run ``great_expectations suite scaffold`` to verify your new Datasource and BatchKwargs-Generator configurations.

Since you edited the Great Expectations configuration file, the updated configuration should be tested to make sure that no errors were introduced.

1. From the command line, run:

```
great_expectations suite scaffold name_of_new_expectation_suite
```

```

Select a datasource
  1. local_filesystem
  2. some_sql_db
  3. pandas_s3
: 3

```

Note: If `pandas_s3` is the only available data source, then you will not be offered a choice of the data source; in this case, the `pandas_s3` data source will be chosen automatically.

2. Choose to see “a list of data assets in this datasource”

```

Would you like to:
  1. choose from a list of data assets in this datasource
  2. enter the path of a data file
: 1

```

3. Verify that all your data assets appear in the list

```

Which data would you like to use?
  1. your_first_data_asset_name (file)
  2. your_second_data_asset_name (file)
  3. your_third_data_asset_name (file)
  Don't see the name of the data asset in the list above?_
↪Just type it
:

```

When you select the number corresponding to a data asset, a Jupyter notebook will open, pre-populated with the code for adding expectations to the expectation suite specified on the command line against the data set you selected.

Check the composition of the `batch_kwarg`s variable at the top of the notebook to make sure that the S3 file used appropriately corresponds to the data set you selected. Repeat this check for all data sets you configured. An inconsistency is likely due to an incorrect regular expression pattern in the respective data set configuration.

Additional Notes

1. Additional options are available for a more fine-grained customization of the S3-backed Pandas data sources.

```
delimiter: "/" # This is the delimiter for the bucket keys (paths inside the
↳ buckets). By default, it is "/".

boto3_options:
  endpoint_url: ${S3_ENDPOINT} # Uses the S3_ENDPOINT environment variable to
↳ determine which endpoint to use.

reader_options: # Note that reader options can be specified globally or per-
↳ asset.
  sep: ",",

max_keys: 100 # The maximum number of keys to fetch in a single request to S3,
↳ (default is 100).
```

2. Errors in generated BatchKwarg during configuration of the S3GlobReaderBatchKwargGenerator are likely due to an incorrect regular expression pattern in the respective data set configuration.
3. The default values of the various options satisfy the vast majority of scenarios. However, in certain cases, the developers may need to override them. For instance, `reader_options`, which can be specified globally and/or at the per-asset level, provide a mechanism for customizing the separator character inside CSV files.
4. Note that specifying the `--no-jupyter` flag on the command line will initialize the specified expectation suite in the `great_expectations/expectations` directory, but suppress the launching of the Jupyter notebook.

```
great_expectations suite scaffold name_of_new_expectation_suite --no-jupyter
```

If you resume editing the given expectation suite at a later time, please first verify that the `batch_kwarg`s contain the correct S3 path for the intended data source.

Comments

3.2.6 How to configure a Redshift Datasource

This guide shows how to connect to a Redshift Datasource.

Prerequisites: This how-to guide assumes you have already:

- *Set up a working deployment of Great Expectations*
-

Steps

To add a Redshift datasource, do this:

1. Install the required modules

If you haven't already, install these modules for connecting to Redshift.

```
pip install sqlalchemy  
  
pip install psycopg2  
  
# or if on macOS:  
pip install psycopg2-binary
```

2. Run `datasource new`

From the command line, run:

```
great_expectations datasource new
```

3. Choose “Relational database (SQL)”

```
What data would you like Great Expectations to connect to?  
  1. Files on a filesystem (for processing with Pandas or Spark)  
  2. Relational database (SQL)  
: 2
```

4. Choose Redshift

```
Which database backend are you using?  
  1. MySQL  
  2. Postgres  
  3. Redshift  
  4. Snowflake  
  5. BigQuery  
  6. other - Do you have a working SQLAlchemy connection string?  
: 3
```

5. Give your Datasource a name

When prompted, provide a custom name for your Redshift data source, or hit Enter to accept the default.

```
Give your new Datasource a short name.  
[my_redshift_db]:
```

6. Provide credentials

Next, you will be asked to supply the credentials for your Redshift instance:

```
Next, we will configure database credentials and store them in the `my_  
↪redshift_db` section  
of this config file: great_expectations/uncommitted/config_variables.yml:
```

```
What is the host for the Redshift connection? []: my-datawarehouse-name.  
↪abcdelqrstuw.us-east-1.redshift.amazonaws.com  
What is the port for the Redshift connection? [5439]:  
What is the username for the Redshift connection? []: myusername
```

(continues on next page)

(continued from previous page)

```
What is the password for the Redshift connection?:
What is the database name for the Redshift connection? []: my_database
What is sslmode name for the Redshift connection? [prefer]: prefer
```

Great Expectations will store these secrets privately on your machine. They will not be committed to git.

7. Wait to verify your connection

You will then see the following message on your terminal screen:

```
Attempting to connect to your database. This may take a moment...
```

If all goes well, it will be followed by the message:

```
Great Expectations connected to your database!
```

If you run into an error, you will see something like:

```
Cannot connect to the database.
- Please check your environment and the configuration you provided.
- Database Error: Cannot initialize datasource my_redshift_db, error:
↳ (psycopg2.OperationalError) could not connect to server: No such file
↳ or directory
  Is the server running locally and accepting
  connections on Unix domain socket "/tmp/.s.PGSQL.5439"?

(Background on this error at: http://sqlalche.me/e/e3q8)
Enter the credentials again? [Y/n]: n
```

In this case, please check your credentials, ports, firewall, etc. and try again.

8. Save your new configuration

Finally, you'll be asked to confirm that you want to save your configuration:

```
Great Expectations will now add a new Datasource 'my_redshift_db' to your
↳ deployment, by adding this entry to your great_expectations.yml:

my_redshift_db:
  credentials: ${my_redshift_db}
  data_asset_type:
    class_name: SQLAlchemyDataset
    module_name: great_expectations.dataset
    class_name: SQLAlchemyDatasource

The credentials will be saved in uncommitted/config_variables.yml under
↳ the key 'my_redshift_db'

Would you like to proceed? [Y/n]:
```

After this confirmation, you can proceed with exploring the data sets in your new Redshift Data-source.

Additional Notes

1. Depending on your Redshift cluster configuration, you may or may not need the `sslmode` parameter.
2. Should you need to modify your connection string, you can manually edit the `great_expectations/uncommitted/config_variables.yml` file.
3. You can edit the `great_expectations/uncommitted/config_variables.yml` file to accomplish the connection configuration without using the CLI. The entry would have the following format:

```
my_redshift_db:
  url: "postgresql+psycopg2://username:password@host:port/database_name?
  ↪sslmode=require"
```

Comments

3.2.7 How to configure a self managed Spark Datasource

Admonition from Mr. Dickens

“Whether I shall turn out to be the hero of my own life, or whether that station will be held by anybody else, these pages must show.”

This guide is a stub. We all know that it will be useful, but no one has made time to write it yet.

If it would be useful to you, please comment with a +1 and feel free to add any suggestions or questions below.

If you want to be a real hero, we’d welcome a pull request. Please see [the Contributing tutorial](#) and [How to write a how to guide](#) to get started.

3.2.8 How to configure a Snowflake Datasource

This guide shows how to connect to a Snowflake Datasource.

Prerequisites: This how-to guide assumes you have already:

- *Set up a working deployment of Great Expectations*
-

Steps

To add a Snowflake datasource, do this:

1. Install the required modules

If you haven’t already, install these modules for connecting to Snowflake.

```
pip install sqlalchemy
pip install snowflake-connector-python
pip install snowflake-sqlalchemy
```

2. Run `datasource new`

From the command line, run:

```
great_expectations datasource new
```

3. Choose “Relational database (SQL)”

```
What data would you like Great Expectations to connect to?
  1. Files on a filesystem (for processing with Pandas or Spark)
  2. Relational database (SQL)
: 2
```

4. Choose Snowflake

```
Which database backend are you using?
  1. MySQL
  2. Postgres
  3. Redshift
  4. Snowflake
  5. BigQuery
  6. other - Do you have a working SQLAlchemy connection string?
: 4
```

5. Give your Datasource a name

When prompted, provide a custom name for your Snowflake data source, or hit Enter to accept the default.

```
Give your new Datasource a short name.
[my_snowflake_db]:
```

6. Provide credentials

Next, you will be asked to supply the credentials for your Snowflake instance:

```
Next, we will configure database credentials and store them in the `my_
↪snowflake_db` section
of this config file: great_expectations/uncommitted/config_variables.yml:

What is the user login name for the snowflake connection? []: myusername
What is the password for the snowflake connection?:
What is the account name for the snowflake connection (include region --
↪ex 'ABCD.us-east-1')? [xyz12345.us-east-1]:
What is database name for the snowflake connection? (optional -- leave
↪blank for none) []: MY_DATABASE
What is schema name for the snowflake connection? (optional -- leave
↪blank for none) []: MY_SCHEMA
What is warehouse name for the snowflake connection? (optional -- leave
↪blank for none) []: MY_COMPUTE_WH
What is role name for the snowflake connection? (optional -- leave blank
↪for none) []: MY_ROLE
```

Great Expectations will store these secrets privately on your machine. They will not be committed to git.

7. Wait to verify your connection

You will then see the following message on your terminal screen:

```
Attempting to connect to your database. This may take a moment...
```

If all goes well, it will be followed by the message:

```
Great Expectations connected to your database!
```

If you run into an error, you will see something like:

```
Cannot connect to the database.
- Please check your environment and the configuration you provided.
- Database Error: Cannot initialize datasource my_snowflake_db, error:
↳ (snowflake.connector.errors.DatabaseError) 250001 (08001): Failed to
↳ connect to DB: oca29081.us-east-1.snowflakecomputing.com:443. Incorrect
↳ username or password was specified.

(Background on this error at: http://sqlalche.me/e/4xp6)
Enter the credentials again? [Y/n]:
```

In this case, please check your credentials, ports, firewall, etc. and try again.

8. Save your new configuration

Finally, you'll be asked to confirm that you want to save your configuration:

```
Great Expectations will now add a new Datasource 'my_snowflake_db' to
↳ your deployment, by adding this entry to your great_expectations.yml:

my_snowflake_db:
  credentials: ${my_snowflake_db}
  data_asset_type:
    class_name: SqlAlchemyDataset
    module_name: great_expectations.dataset
    class_name: SqlAlchemyDatasource

The credentials will be saved in uncommitted/config_variables.yml under
↳ the key 'my_snowflake_db'

Would you like to proceed? [Y/n]:
```

After this confirmation, you can proceed with exploring the data sets in your new Snowflake Data-source.

Additional Notes

1. When using the Snowflake dialect, *SqlAlchemyDataset* will create a **transient** table instead of a **temporary** table when passing in *query* Batch Kwargs or providing *custom_sql* to its constructor. Consequently, users **must** provide a *snowflake_transient_table* in addition to the *query* parameter. Any existing table with that name will be overwritten.
2. Should you need to modify your connection string, you can manually edit the `great_expectations/uncommitted/config_variables.yml` file.
3. You can edit the `great_expectations/uncommitted/config_variables.yml` file to accomplish the connection configuration without using the CLI. The entry would have the following format:

```
my_snowflake_db:
  url: "snowflake://<user_login_name>:<password>@<account_name>/<database_name>/
↳ <schema_name>?warehouse=<warehouse_name>&role=<role_name>"
```

Comments

3.2.9 How to configure a Spark/filesystem Datasource

This guide shows how to connect to a Spark Datasource such that the data is accessible in the form of files on a local or NFS type of a filesystem.

Prerequisites: This how-to guide assumes you have already:

- *Set up a working deployment of Great Expectations*

Steps

To add a filesystem-backed Spark datasource do this:

1. Run `datasource new`

From the command line, run:

```
great_expectations datasource new
```

2. Choose “Files on a filesystem (for processing with Pandas or Spark)”

```
What data would you like Great Expectations to connect to?
  1. Files on a filesystem (for processing with Pandas or Spark)
  2. Relational database (SQL)
: 1
```

3. Choose PySpark

```
What are you processing your files with?
  1. Pandas
  2. PySpark
: 2
```

4. Specify the directory path for data files

```
Enter the path (relative or absolute) of the root directory where the
↳ data files are stored.
: /path/to/directory/containing/your/data/files
```

5. Give your Datasource a name

When prompted, provide a custom name for your filesystem-backed Spark data source, or hit Enter to accept the default.

```
Give your new Datasource a short name.
[my_data_files_dir]:
```

Great Expectations will now add a new Datasource ‘my_data_files_dir’ to your deployment, by adding this entry to your `great_expectations.yml`:

```
my_data_files_dir:
  data_asset_type:
    class_name: SparkDFDataset
```

(continues on next page)

(continued from previous page)

```
module_name: great_expectations.dataset
spark_config: {}
batch_kwarg_generators:
  subdir_reader:
    class_name: SubdirReaderBatchKwargGenerator
    base_directory: /path/to/directory/containing/your/data/files
class_name: SparkDFDatasource

Would you like to proceed? [Y/n]:
```

6. Wait for confirmation

If all goes well, it will be followed by the message:

```
A new datasource 'my_data_files_dir' was added to your project.
```

If you run into an error, you will see something like:

```
Error: Directory '/nonexistent/path/to/directory/containing/your/data/
↳files' does not exist.

Enter the path (relative or absolute) of the root directory where the
↳data files are stored.
:
```

In this case, please check your data directory path, permissions, etc. and try again.

7. Finally, if all goes well and you receive a confirmation on your Terminal screen, you can proceed with exploring the data sets in your new filesystem-backed Spark data source.

Additional Notes

1. Relative path locations should be specified from the perspective of the directory, in which the

```
great_expectations datasource new
```

command is executed.

Comments

3.2.10 How to configure an EMR Spark Datasource

Admonition from Mr. Dickens

“Whether I shall turn out to be the hero of my own life, or whether that station will be held by anybody else, these pages must show.”

This guide is a stub. We all know that it will be useful, but no one has made time to write it yet.

If it would be useful to you, please comment with a +1 and feel free to add any suggestions or questions below.

If you want to be a real hero, we’d welcome a pull request. Please see [the Contributing tutorial](#) and [How to write a how to guide](#) to get started.

3.3 Configuring metadata stores

3.3.1 How to configure a Validation Result store in Azure blob storage

Unfortunately, this feature has not been implemented yet. If you would like to help contribute to Great Expectations, please start with our [Contributing](#) guide and don't be shy with questions!

Also, please reach out to us on [slack](#) if you would like to learn more, or have any questions.

3.3.2 How to configure a Validation Result store in GCS

Admonition from Mr. Dickens

“Whether I shall turn out to be the hero of my own life, or whether that station will be held by anybody else, these pages must show.”

This guide is a stub. We all know that it will be useful, but no one has made time to write it yet.

If it would be useful to you, please comment with a +1 and feel free to add any suggestions or questions below.

If you want to be a real hero, we'd welcome a pull request. Please see [the Contributing tutorial](#) and [How to write a how to guide](#) to get started.

3.3.3 How to configure a Validation Result store in S3

Admonition from Mr. Dickens

“Whether I shall turn out to be the hero of my own life, or whether that station will be held by anybody else, these pages must show.”

This guide is a stub. We all know that it will be useful, but no one has made time to write it yet.

If it would be useful to you, please comment with a +1 and feel free to add any suggestions or questions below.

If you want to be a real hero, we'd welcome a pull request. Please see [the Contributing tutorial](#) and [How to write a how to guide](#) to get started.

3.3.4 How to configure a Validation Result store on a filesystem

By default, Validation results are stored in the `great_expectations/uncommitted/validations/` directory. Since Validations may include examples of data (which could be sensitive or regulated) they should not be committed to a source control system.

This guide will help you configure a new storage location for Validations on your filesystem.

Prerequisites: This how-to guide assumes that you have already:

- Configured a [Data Context](#).
- Configured an [Expectation Suite](#).
- Configured a [Checkpoint](#).

- Determined a new storage location where you would like to store Validations. This can either be a local path, or a path to a secure network filesystem.
-

Steps

1. Open the `great_expectations.yml` file and look for the following lines.

```
validations_store_name: validations_store

stores:
  validations_store:
    class_name: ValidationsStore
    store_backend:
      class_name: TupleFilesystemStoreBackend
      base_directory: uncommitted/validations/
```

The configuration file tells Great Expectations to look for Validations in a store called `validations_store`. The `base_directory` is set to `uncommitted/validations` by default.

2. Update your configuration file using the example below. In our case, Validations store is being set to `shared_validations_filesystem_store`, but it can be any name you like. Also, the `base_directory` is being set to `uncommitted/shared_validations/`, but it can be set to any path accessible by Great Expectations. Copy existing Validation results to the new folder if necessary.

```
expectations_store_name: shared_validations_filesystem_store

stores:
  shared_validations_filesystem_store:
    class_name: ValidationsStore
    store_backend:
      class_name: TupleFilesystemStoreBackend
      base_directory: uncommitted/shared_validations/
```

3. Confirm that the Validation store has been updated by re-running a Checkpoint: `great_expectations checkpoint run check_1`. You can also visualize the results by re-building [Data Docs](#).

```
great_expectations checkpoint run check_1 # check_1 is the name of our Checkpoint

Validation Succeeded!
```

3.3.5 How to configure a Validation Result store to postgresql

Admonition from Mr. Dickens

“Whether I shall turn out to be the hero of my own life, or whether that station will be held by anybody else, these pages must show.”

This guide is a stub. We all know that it will be useful, but no one has made time to write it yet.

If it would be useful to you, please comment with a +1 and feel free to add any suggestions or questions below.

If you want to be a real hero, we’d welcome a pull request. Please see [the Contributing tutorial](#) and [How to write a how to guide](#) to get started.

3.3.6 How to configure an Expectation store in Amazon S3

By default, newly profiled Expectations are stored in JSON format in the `expectations/` subdirectory of your `great_expectations/` folder. This guide will help you configure Great Expectations to store them in an Amazon S3 bucket.

Prerequisites: This how-to guide assumes that you have already:

- Configured a *Data Context*.
- Configured an *Expectations Suite*.
- Installed `boto3` in your local environment.
- Identified the S3 bucket and prefix where Expectations will be stored.

Steps

1. Configure `boto3` to connect to the Amazon S3 bucket where Expectations will be stored.
2. Identify your Data Context Expectations Store

In your `great_expectations.yml`, look for the following lines. The configuration tells Great Expectations to look for Expectations in a store called `expectations_store`. The `base_directory` for `expectations_store` is set to `expectations/` by default.

```
expectations_store_name: expectations_store

stores:
  expectations_store:
    class_name: ExpectationsStore
    store_backend:
      class_name: TupleFilesystemStoreBackend
      base_directory: expectations/
```

3. Update your configuration file to include a new store for Expectations on S3.

In our case, the name is set to `expectations_S3_store`, but it can be any name you like. We also need to make some changes to the `store_backend` settings. The `class_name` will be set to `TupleS3StoreBackend`, `bucket` will be set to the address of your S3 bucket, and `prefix` will be set to the folder where Expectation files will be located.

Warning: If you are also storing *Validations in S3*, please ensure that the `prefix` values are disjoint and one is not a substring of the other.

```
expectations_store_name: expectations_S3_store

stores:
  expectations_S3_store:
    class_name: ExpectationsStore
    store_backend:
      class_name: TupleS3StoreBackend
      bucket: '<your_s3_bucket_name>'
      prefix: '<your_s3_bucket_folder_name>'
```

4. Copy existing Expectation JSON files to the S3 bucket. (This step is optional).

One way to copy Expectations into Amazon S3 is by using the `aws s3 sync` command. As mentioned earlier, the `base_directory` is set to `expectations/` by default. In the example below, two Expectations, `exp1` and `exp2` are copied to Amazon S3. Your output should look something like this:

```
aws s3 sync '<base_directory>' s3://'<your_s3_bucket_name>/' '<your_s3_
↪bucket_folder_name>'

upload: ./exp1.json to s3://'<your_s3_bucket_name>/' '<your_s3_bucket_
↪folder_name>'/exp1.json
upload: ./exp2.json to s3://'<your_s3_bucket_name>/' '<your_s3_bucket_
↪folder_name>'/exp2.json
```

5. **Confirm that the new Expectations store has been added by running** `great_expectations store list`.

Notice the output contains two Expectation stores: the original `expectations_store` on the local filesystem and the `expectations_S3_store` we just configured. This is ok, since Great Expectations will look for Expectations in the S3 bucket as long as we set the `expectations_name` variable to `expectations_S3_store`.

```
great_expectations store list

- name: expectations_store
  class_name: ExpectationsStore
  store_backend:
    class_name: TupleFilesystemStoreBackend
    base_directory: expectations/

- name: expectations_S3_store
  class_name: ExpectationsStore
  store_backend:
    class_name: TupleS3StoreBackend
    bucket: '<your_s3_bucket_name>'
    prefix: '<your_s3_bucket_folder_name>'
```

6. **Confirm that Expectations can be accessed from Amazon S3 by running** `great_expectations suite list`.

If you followed Step 4, The output should include the 2 Expectations we copied to Amazon S3: `exp1` and `exp2`. If you did not copy Expectations to the new Store, you will see a message saying no expectations were found.

```
great_expectations suite list

2 Expectation Suites found:
- exp1
- exp2
```


Additional resources

- Instructions on how to set up [boto3](#) with AWS can be found at [boto3's documentation site](#).

If it would be useful to you, please comment with a +1 and feel free to add any suggestions or questions below.

3.3.7 How to configure an Expectation store in Azure blob storage

Unfortunately, this feature has not been implemented yet. If you would like to help contribute to Great Expectations, please start with our [Contributing](#) guide and don't be shy with questions!

Also, please reach out to us on [slack](#) if you would like to learn more, or have any questions.

3.3.8 How to configure an Expectation store in GCS

Admonition from Mr. Dickens

“Whether I shall turn out to be the hero of my own life, or whether that station will be held by anybody else, these pages must show.”

This guide is a stub. We all know that it will be useful, but no one has made time to write it yet.

If it would be useful to you, please comment with a +1 and feel free to add any suggestions or questions below.

If you want to be a real hero, we'd welcome a pull request. Please see [the Contributing tutorial](#) and [How to write a how to guide](#) to get started.

3.3.9 How to configure an Expectation store in git

Admonition from Mr. Dickens

“Whether I shall turn out to be the hero of my own life, or whether that station will be held by anybody else, these pages must show.”

This guide is a stub. We all know that it will be useful, but no one has made time to write it yet.

If it would be useful to you, please comment with a +1 and feel free to add any suggestions or questions below.

If you want to be a real hero, we'd welcome a pull request. Please see [the Contributing tutorial](#) and [How to write a how to guide](#) to get started.

3.3.10 How to configure an Expectation store on a filesystem

By default, newly profiled Expectations are stored in JSON format in the `expectations/` subdirectory of your `great_expectations` folder. This guide will help you configure a new storage location for Expectations on your filesystem.

Prerequisites: This how-to guide assumes that you have already:

- Configured a [Data Context](#).
- Configured an [Expectation Suite](#).

- Determined a new storage location where you would like to store Expectations. This can either be a local path, or a path to a network filesystem.
-

Steps

1. First create a new folder where you would to store your Expectations, and move your existing Expectation files over to the new location. In our case, the name of the Expectations file is `npi_expectations` and the path to our new storage location is `/shared_expectations`.

```
# in the great_expectations folder
mkdir shared_expectations
mv expectations/npi_expectations.json shared_expectations/
```

2. Next open the `great_expectations.yml` file and look for the following lines.

```
expectations_store_name: expectations_store

stores:
  expectations_store:
    class_name: ExpectationsStore
    store_backend:
      class_name: TupleFilesystemStoreBackend
      base_directory: expectations/
```

The configuration file tells Great Expectations to look for Expectations in a Store called `expectations_store`. The `base_directory` for `expectations_store` is set to `expectations/` by default.

3. Update your configuration following the example below. This example would change the Store name to `shared_expectations_filesystem_store` with the `base_directory` set to `shared_expectations/`.

Paths are relative to the directory where `great_expectations.yml` is stored.

```
expectations_store_name: shared_expectations_filesystem_store

stores:
  shared_expectations_filesystem_store:
    class_name: ExpectationsStore
    store_backend:
      class_name: TupleFilesystemStoreBackend
      base_directory: shared_expectations/
```

4. Confirm that the location has been updated by running `great_expectations store list`.

```
great_expectations store list

3 Stores found:

- name: shared_expectations_filesystem_store
  class_name: ExpectationsStore
  store_backend:
    class_name: TupleFilesystemStoreBackend
    base_directory: shared_expectations/
```

5. Confirm that Expectations can be read from the new storage location by running `great_expectations suite list`.

```
great_expectations suite list
```

```
1 Expectation Suite found:
  - np_i_expectations
```

Additional Notes

- For best practices, we highly recommend that you store Expectations in a version-control system like Git. The JSON format of Expectations will allow for informative diff-statements and effective tracking of modifications. In the example below, 2 changes have been made to `np_i_expectations`. The Expectation ``expect_table_column_count_to_equal`` was been changed from 330 to 333 to 331.

```
git log -p np_i_expectations.json

commit cbc127fb27095364c3c1fc6f6e7f078369b07455
    changed expect_table_column_count_to_equal to 331

diff --git a/great_expectations/expectations/np_i_expectations.json b/great_
->expectations/expectations/np_i_expectations.json

--- a/great_expectations/expectations/np_i_expectations.json
+++ b/great_expectations/expectations/np_i_expectations.json
@@ -17,7 +17,7 @@
     {
       "expectation_type": "expect_table_column_count_to_equal",
       "kwargs": {
-        "value": 333
+        "value": 331
       }
     }
commit 05b3c8c1ed35d183bac1717d4877fe13bc574963
    changed expect_table_column_count_to_equal to 333

diff --git a/great_expectations/expectations/np_i_expectations.json b/great_
->expectations/expectations/np_i_expectations.json
--- a/great_expectations/expectations/np_i_expectations.json
+++ b/great_expectations/expectations/np_i_expectations.json
     {
       "expectation_type": "expect_table_column_count_to_equal",
       "kwargs": {
-        "value": 330
+        "value": 333
       }
     }
  }
```

3.3.11 How to configure an Expectation store to postgresql

Admonition from Mr. Dickens

“Whether I shall turn out to be the hero of my own life, or whether that station will be held by anybody else, these pages must show.”

This guide is a stub. We all know that it will be useful, but no one has made time to write it yet.

If it would be useful to you, please comment with a +1 and feel free to add any suggestions or questions below.

If you want to be a real hero, we'd welcome a pull request. Please see [the Contributing tutorial](#) and [How to write a how to guide](#) to get started.

This guide is

Find more discussion [here](#)

Thanks to contributors

- [@joe_gargery](#)
- [@jaggers](#)
- [@esthav2002](#)

3.3.12 How to configure a MetricsStore

Saving metrics during Validation makes it easy to construct a new data series based on observed dataset characteristics computed by Great Expectations. That data series can serve as the source for a dashboard or overall data quality metrics, for example.

Storing metrics is still a **beta** feature of Great Expectations, and we expect configuration and capability to evolve rapidly.

Adding a MetricsStore

A MetricStore is a special store that can store Metrics computed during Validation. A Metric store tracks the run_id of the validation and the expectation suite name in addition to the metric name and metric kwargs.

In most cases, a MetricStore will be configured as a SQL database. To add a MetricStore to your DataContext, add the following yaml block to the “stores” section:

```
stores:
  # ...
  metrics_store: # You can choose any name for your metric store
    class_name: MetricStore
    store_backend:
      class_name: DatabaseStoreBackend
      # These credentials can be the same as those used in a Datasource_
↵configuration
      credentials: ${my_store_credentials}
```

The next time your DataContext is loaded, it will connect to the database and initialize a table to store metrics if one has not already been created. See the [Metrics Reference](#) for more information on additional configuration options.

Configuring a Validation Action

Once a `MetricStore` is available, it is possible to configure a new `StoreMetricsAction` to save metrics during validation. Add the following yaml block to your DataContext validation operators configuration:

```
validation_operators:
  # ...
  action_list_operator:
    class_name: ActionListValidationOperator
    action_list:
      # ...
      - name: store_metrics
        action:
          class_name: StoreMetricsAction
          target_store_name: metrics_store # Keep the space before this hash so it's
↪not read as the name. This should match the name of the store configured above
          # Note that the syntax for selecting requested metrics will change in a
↪future release
          requested_metrics:
            "*": # The asterisk here matches *any* expectation suite name
              # use the 'kwargs' key to request metrics that are defined by kwargs,
              # for example because they are defined only for a particular column
              # - column:
              #   Age:
              #     - expect_column_min_to_be_between.result.observed_value
              - statistics.evaluated_expectations
              - statistics.successful_expectations
```

The `StoreMetricsValidationAction` processes an `ExpectationValidationResult` and stores Metrics to a configured Store. Now, when your operator is executed, the requested metrics will be available in your database!

```
context.run_validation_operator('action_list_operator', (batch_kwargs, expectation_
↪suite_name))
```

Note: To discuss with the Great Expectations community, please visit this topic in our community discussion forum: <https://discuss.greatexpectations.io/t/ge-with-databricks-delta/82/3>

3.4 Creating Batches

3.4.1 How to load a database table or a query result as a batch

Admonition from Mr. Dickens

“Whether I shall turn out to be the hero of my own life, or whether that station will be held by anybody else, these pages must show.”

This guide is a stub. We all know that it will be useful, but no one has made time to write it yet.

If it would be useful to you, please comment with a +1 and feel free to add any suggestions or questions below.

If you want to be a real hero, we’d welcome a pull request. Please see [the Contributing tutorial](#) and [How to write a how to guide](#) to get started.

3.4.2 How to load a Pandas dataframe as a Batch

Admonition from Mr. Dickens

“Whether I shall turn out to be the hero of my own life, or whether that station will be held by anybody else, these pages must show.”

This guide is a stub. We all know that it will be useful, but no one has made time to write it yet.

If it would be useful to you, please comment with a +1 and feel free to add any suggestions or questions below.

If you want to be a real hero, we’d welcome a pull request. Please see [the Contributing tutorial](#) and [How to write a how to guide](#) to get started.

3.4.3 How to load a Pandas dataframe as a Batch

Admonition from Mr. Dickens

“Whether I shall turn out to be the hero of my own life, or whether that station will be held by anybody else, these pages must show.”

This guide is a stub. We all know that it will be useful, but no one has made time to write it yet.

If it would be useful to you, please comment with a +1 and feel free to add any suggestions or questions below.

If you want to be a real hero, we’d welcome a pull request. Please see [the Contributing tutorial](#) and [How to write a how to guide](#) to get started.

3.4.4 How to load a Spark dataframe as a batch

Admonition from Mr. Dickens

“Whether I shall turn out to be the hero of my own life, or whether that station will be held by anybody else, these pages must show.”

This guide is a stub. We all know that it will be useful, but no one has made time to write it yet.

If it would be useful to you, please comment with a +1 and feel free to add any suggestions or questions below.

If you want to be a real hero, we’d welcome a pull request. Please see [the Contributing tutorial](#) and [How to write a how to guide](#) to get started.

3.5 Creating and editing Expectations

3.5.1 How to create a new Expectation Suite using `suite scaffold`

`great_expectations suite scaffold` helps you quickly create of *Expectation Suites* through an interactive development loop that combines *Profilers* and *Data Docs*.

Prerequisites: This how-to guide assumes you have already:

- *Set up a working deployment of Great Expectations*
-

Steps

1. Run `suite scaffold`

```
great_expectations suite scaffold my_suite
```

Much like the `suite new` and `suite edit` commands, you will be prompted interactively to choose some data from one of your datasources.

```
suite scaffold np_i_distributions --no-jupyter
Heads up! This feature is Experimental. It may change. Please give us
↳ your feedback!

Enter the path (relative or absolute) of a data file
: np_i.csv
To continue scaffolding this suite, run `jupyter notebook uncommitted/
↳ scaffold_np_i_distributions.ipynb`
```

Important: This command generates a disposable jupyter notebook in your `great_expectations/uncommitted` directory. The goal is to save you time by writing boilerplate code for you.

Because these notebooks can be generated at any time from the expectation suites (stored as JSON) you should **consider the notebook to be a disposable artifact**. You can delete it at any time.

Once you choose a data asset, the CLI will open up a jupyter notebook.

2. Within the notebook, choose columns and Expectations to scaffold.

Run the first cell in the notebook that loads the data. You don't need to worry about what's happening there.

The next code cell in the notebook presents you with a list of all the columns found in your selected data. Because the scaffolder is not very smart, you will want to edit this suite to tune the parameters and make any adjustments such as removing *Expectations* that don't make sense for your use case.

```
included_columns = [
    'crim',
    'zn',
    'indus',
    'chas',
    'nox',
```

(continues on next page)

(continued from previous page)

```
'rm',
'age',
# 'dis',
'rad',
# 'tax',
'ptratio',
# 'b',
# 'lstat',
# 'medv'
]
```

To select which columns you want to scaffold Expectations on, simply uncomment them to include them.

```
scaffold_config = {
    "included_columns": included_columns,
    # "excluded_columns": [],
    # "included_expectations": [],
    # "excluded_expectations": [],
}
```

3. Generate Data Docs and review the results there

Run the next few code cells to see the scaffolded suite in Data Docs.

You can iterate on included and excluded columns and Expectations to get closer to the Suite you want.

Additional notes

Important: The Suites generated by the `scaffold` command **are not meant to be production suites** - they are scaffolds to build upon.

Great Expectations will choose which Expectations **might make sense** for a column based on the type and cardinality of the data in each selected column.

You will definitely want to edit the Suite to hone it after scaffolding.

3.5.2 How to create a new Expectation Suite using the CLI

While you could hand-author an Expectation Suite by writing a JSON file, just like with other features it is easier to let *CLI* save you time and typos. Run this command in the root directory of your project (where the `init` command created the `great_expectations` subdirectory:

```
great_expectations suite new
```

This command prompts you to name your new Expectation Suite and to select a sample batch of data the suite will describe. Then it uses a sample of the selected data to add some initial expectations to the suite. The purpose of these is expectations is to provide examples of data assertions, and not to be meaningful. They are intended only a starting point for you to build upon.

The command concludes by saving the newly generated Expectation Suite as a JSON file and rendering the expectation suite into an HTML page in Data Docs.

3.5.3 How to create custom Expectations for pandas

Custom Expectations let you extend the logic for validating data to use any criteria you choose. This guide will show you how to extend the `PandasDataset` class with your own *Expectations*.

Prerequisites: This how-to guide assumes you have already:

- *Set up a working deployment of Great Expectations*
- Launched a generic notebook (e.g. `jupyter notebook`, `jupyter lab`, etc.)
- Obtained data that can be accessed from your notebook
- *Configured a pandas Datasource*

Steps

1. Import `great_expectations` and `PandasDataset` and `MetaPandasDataset`

```
import great_expectations as ge
from great_expectations.dataset import (
    PandasDataset,
    MetaPandasDataset,
)
```

`PandasDataset` is the parent class used for executing Expectations on pandas Dataframes. Most of the core Expectations are built using decorators defined in `MetaPandasDataset`. These decorators greatly streamline the task of extending Great Expectations with custom Expectation logic.

2. Define a class inheriting from `PandasDataset`

```
class MyCustomPandasDataset(PandasDataset):

    _data_asset_type = "MyCustomPandasDataset"
```

Setting the `_data_asset_type` is not strictly necessary, but can be helpful for tracking the lineage of instantiated Expectations and *Validation Results*.

3. Within your new class, define Expectations using decorators from `MetaPandasDataset`

`column_map_expectations` are Expectations that are applied to a single column, on a row-by-row basis. To learn about other Expectation types, please see *Other Expectation decorators* below.

The `@MetaPandasDataset.column_map_expectation` decorator wraps your custom function with all the business logic required to turn it into a fully-fledged Expectation. This spares you the hassle of defining logic to handle required arguments like `mostly` and `result_format`. Your custom function can focus exclusively on the business logic of passing or failing the Expectation.

In the simplest case, they could be as simple as one-line lambda functions.

```
@MetaPandasDataset.column_map_expectation
def expect_column_values_to_be_even(self, column):
    return column.map(lambda x: x%2==0)
```

To use the `column_map_expectation` decorator, your custom function must accept at least two arguments: `self` and `column`. When the user invokes your Expectation, they will pass a string containing the column name. The decorator will then fetch the appropriate column and pass all of

the non-null values to your function as a pandas Series. Your function must then return a Series of boolean values in the same order, with the same index.

Custom functions can also accept additional arguments:

```
@MetaPandasDataset.column_map_expectation
def expect_column_values_to_be_less_than(self, column, value):
    return column.map(lambda x: x<value)
```

Custom functions can have complex internal logic:

```
@MetaPandasDataset.column_map_expectation
def expect_column_value_word_counts_to_be_between(self, column, min_
↪value=None, max_value=None):
    def count_words(string):
        word_list = re.findall("(\\S+)", string)
        return len(word_list)

    word_counts = column.map(lambda x: count_words(str(x)))

    if min_value != None and max_value != None:
        return word_counts.map(lambda x: min_value <= x <= max_value)
    elif min_value != None and max_value == None:
        return word_counts.map(lambda x: min_value <= x)
    elif min_value == None and max_value != None:
        return word_counts.map(lambda x: x <= max_value)
    else:
        return word_counts.map(lambda x: True)
```

Custom functions can reference external modules and methods:

```
import pytz

@MetaPandasDataset.column_map_expectation
def expect_column_values_to_be_valid_timezones(self, column, timezone_
↪values=pytz.all_timezones):
    return column.map(lambda x: x in timezone_values)
```

By convention, column_map_expectations always start with expect_column_values... or expect_column_value... (Ex: expect_column_value_word_counts_to_be_between). Following this pattern is highly recommended, but not strictly required. If you want to confuse yourself with bad names, the package won't stop you.

4. Load some data

To make your new Expectations available for validation, you can instantiate a MyCustomPandasDataset as follows:

```
my_df = ge.read_csv("./data/Titanic.csv", dataset_
↪class=MyCustomPandasDataset)
```

You can also coerce an existing pandas DataFrame to your class using from_pandas:

```
my_pd_df = pd.read_csv("./data/Titanic.csv")
my_df = ge.from_pandas(my_pd_df, dataset_class=MyCustomPandasDataset)
```

As a third option:

Note: We're using the `read_csv` method to fetch data, instead of the more typical `DataContext.get_batch`. This is for convenience: it allows us to handle the full development loop for a custom Expectation within a notebook with a minimum of configuration.

In a moment, we'll demonstrate how to configure a Datasource to use `MyCustomPandasDataset` when calling `get_batch`.

5. Test your Expectations

At this point, you can test your new Expectations exactly like built-in Expectations. All out-of-the-box Expectations will still be available, plus your new methods.

returns

As mentioned previously, the `column_map_expectation` decorator extends the arguments to include other arguments, like `mostly`. Please see the module documentation for full details.

```
my_df.expect_column_values_to_be_even("Survived", mostly=.6)
```

returns

Often, the best development loop for custom Expectations is iterative: editing Expectations in `MyCustomPandasDataset`, then re-running the cells to load data and execute Expectations on data.

At this point, your custom Expectations work—but only within a notebook. Next, let's configure them to work from within a Datasource in your Data Context.

6. Save your `MyCustomPandasDataset` class to a Plugin module

The simplest way to do this is to create a new, single-file python module within your `great_expectations/plugins/` directory. Name it something like `custom_pandas_dataset.py`. Copy the full contents of your `MyCustomPandasDataset` class into this file. Make sure to include any required imports, too.

When you instantiate a Data Context, Great Expectations automatically adds `plugins/` to the python namespace, so your class can be imported as `custom_pandas_dataset.MyCustomPandasDataset`. For more information, please see [Plugins](#).

7. Configure your Datasource(s)

Now, open your `great_expectations.yml` file. Assuming that you've previously [configured a pandas Datasource](#), you should see a configuration block similar to this, under the `datasources` key:

```
my_data_dir:
  module_name: great_expectations.datasource
  class_name: PandasDatasource

  data_asset_type:
    module_name: great_expectations.dataset
    class_name: PandasDataset

  batch_kwargs_generators:
    subdir_reader:
      class_name: SubdirReaderBatchKwargsGenerator
      base_directory: ../my_data
```

In the `data_asset_type` section, replace `module_name` and `class_name` with names for your module and class:

```
data_asset_type:
  module_name: custom_pandas_dataset
  class_name: MyCustomPandasDataset
```

Now, any time you load data through the `my_data_dir` Datasource, it will be loaded as a `MyCustomPandasDataset`, with all of your new Expectations available.

If you have other `PandasDatasources` in your configuration, you may want to switch them to use your new `data_asset_type`, too.

8. Test loading a new Batch through the DataContext

You can test this configuration as follows:

```

context = ge.DataContext()
context.create_expectation_suite("my_new_suite")
my_batch = context.get_batch({
    "path": "my_data/Titanic.csv",
    "datasource": "my_data_dir"
}, "my_new_suite")

my_batch.expect_column_values_to_be_even("Age")

```

Executing this Expectation should return something like:

```

{
  "result": {
    "element_count": 1313,
    "missing_count": 557,
    "missing_percent": 42.421934501142424,
    "unexpected_count": 344,
    "unexpected_percent": 26.199543031226202,
    "unexpected_percent_nonmissing": 45.5026455026455,
    "partial_unexpected_list": [
      29.0,
      25.0,
      0.92,
      ...,
      59.0,
      45.0
    ]
  },
  "success": false,
  "meta": {},
  "exception_info": null
}

```

Additional notes

Other Expectation decorators

Aside from `column_map_expectations`, there are several other types of Expectations you can create. Please see the module docs for [MetaPandasDataset](#) for details.

Additional resources

Here's a single code block containing all the notebook code in this article:

```

import re
import pytz

class MyCustomPandasDataset(PandasDataset):

    _data_asset_type = "MyCustomPandasDataset"

    @MetaPandasDataset.column_map_expectation
    def expect_column_values_to_be_even(self, column):
        return column.map(lambda x: x%2==0)

```

(continues on next page)

(continued from previous page)

```

@MetaPandasDataset.column_map_expectation
def expect_column_value_most_common_characters_to_be(self, column, values):
    return column.map(lambda x: set(get_most_common_characters(x)) == set(values))

@MetaPandasDataset.column_map_expectation
def expect_column_value_word_counts_to_be_between(self, column, min_value=None,
↳max_value=None):
    def count_words(string):
        word_list = re.findall("\\S+", string)
        return len(word_list)

    word_counts = column.map(lambda x: count_words(str(x)))

    if min_value != None and max_value != None:
        return word_counts.map(lambda x: min_value <= x <= max_value)
    elif min_value != None and max_value == None:
        return word_counts.map(lambda x: min_value <= x)
    elif min_value == None and max_value != None:
        return word_counts.map(lambda x: x <= max_value)
    else:
        return word_counts.map(lambda x: True)

@MetaPandasDataset.column_map_expectation
def expect_column_values_to_be_valid_timezones(self, column, timezone_values=pytz.
↳all_timezones):
    return column.map(lambda x: x in timezone_values)

#Instantiate the class in several different ways
my_df = ge.read_csv("my_data/Titanic.csv", dataset_class=MyCustomPandasDataset)

my_other_df = pd.read_csv("my_data/Titanic.csv")
ge.from_pandas(my_other_df, dataset_class=MyCustomPandasDataset)

my_other_df = ge.read_csv("my_data/Titanic.csv")
ge.from_pandas(my_other_df, dataset_class=MyCustomPandasDataset)

# Run Expectations in assertions so that they can be used as tests for this guide
assert my_df.expect_column_values_to_be_in_set("Sex", value_set=["Male", "Female"]).
↳success == False
assert my_df.expect_column_values_to_be_even("Survived").success == False
assert my_df.expect_column_values_to_be_even("Survived", mostly=.6).success == True
assert my_df.expect_column_value_word_counts_to_be_between("Name", 3, 5).success ==
↳False
assert my_df.expect_column_value_word_counts_to_be_between("Name", 3, 5, mostly=.9).
↳success == True
assert my_df.expect_column_values_to_be_valid_timezones("Name", mostly=.9).success ==
↳False

```

Comments

3.5.4 How to create custom Expectations for Spark

Admonition from Mr. Dickens

“Whether I shall turn out to be the hero of my own life, or whether that station will be held by anybody else, these pages must show.”

This guide is a stub. We all know that it will be useful, but no one has made time to write it yet.

If it would be useful to you, please comment with a +1 and feel free to add any suggestions or questions below.

If you want to be a real hero, we’d welcome a pull request. Please see [the Contributing tutorial](#) and [How to write a how to guide](#) to get started.

3.5.5 How to create custom Expectations for SQLAlchemy

Admonition from Mr. Dickens

“Whether I shall turn out to be the hero of my own life, or whether that station will be held by anybody else, these pages must show.”

This guide is a stub. We all know that it will be useful, but no one has made time to write it yet.

If it would be useful to you, please comment with a +1 and feel free to add any suggestions or questions below.

If you want to be a real hero, we’d welcome a pull request. Please see [the Contributing tutorial](#) and [How to write a how to guide](#) to get started.

3.5.6 How to define Expectations that span multiple tables

Admonition from Mr. Dickens

“Whether I shall turn out to be the hero of my own life, or whether that station will be held by anybody else, these pages must show.”

This guide is a stub. We all know that it will be useful, but no one has made time to write it yet.

If it would be useful to you, please comment with a +1 and feel free to add any suggestions or questions below.

If you want to be a real hero, we’d welcome a pull request. Please see [the Contributing tutorial](#) and [How to write a how to guide](#) to get started.

3.5.7 How to edit an Expectation Suite using a disposable notebook

Editing *Expectations* in a notebook is usually much more convenient than editing them as raw JSON objects. You can evaluate them against real data, examine the results, and calibrate parameters. Often, you also learn about your data in the process.

To simplify this workflow, the CLI command `suite edit` takes a named Expectation Suite and uses it to *generate* an equivalent Jupyter notebook. You can then use this notebook as a disposable interface to explore data and edit your Expectations.

Prerequisites: This how-to guide assumes you have already:

- *Set up a working deployment of Great Expectations*
- Created an Expectation Suite, possibly using the *great_expectations suite scaffold* or *great_expectations suite new* commands

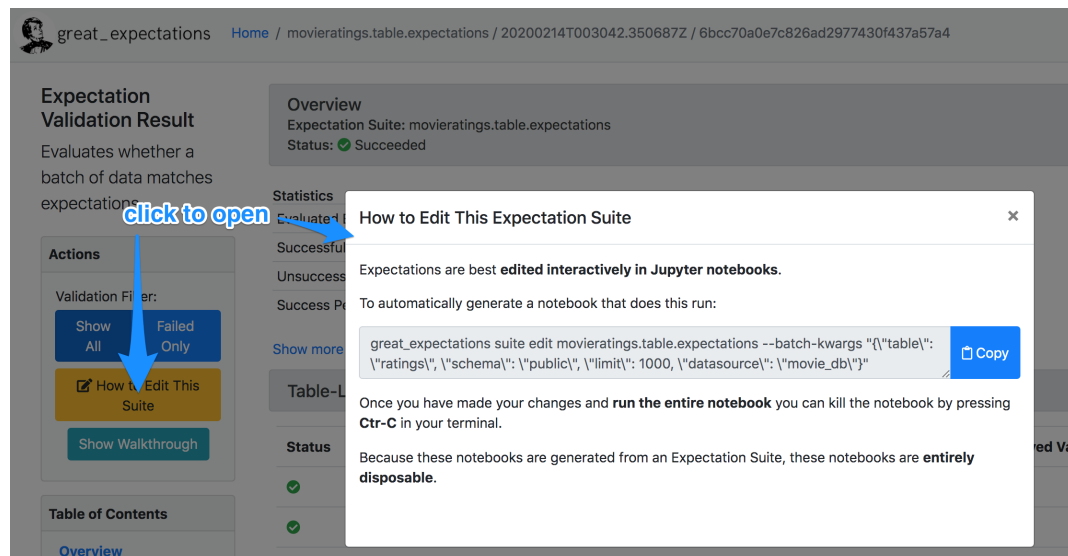
Steps

1. Generate a notebook.

To edit a suite called `movieratings.ratings` you could run the CLI command:

```
great_expectations suite edit movieratings.ratings
```

For convenience, the *Data Docs* page for each Expectation Suite has the CLI command syntax for you. Simply press the “How to Edit This Suite” button, and copy/paste the CLI command into your terminal.



2. Run the boilerplate code at the top of the notebook.

The first code cell at the top of the notebook contains boilerplate code to fetch your Expectation Suite, plus a Batch of data to test it on.

Run this cell to get started.

3. Run (and edit) any Expectations in the “Create & Edit Expectations” section:

The middle portion of the notebook contains a code cell to reproduce each Expectation in the Suite.

- If you re-run these cells with no changes, you will re-validate all of your Expectations against the chosen Batch. At the end of your notebook, the Expectation Suite will contain exactly the same Expectations that you started out with.
- If you edit and run any of the Expectations, the new parameters will replace the old ones.
- If you don't run an Expectation, it will be left out of the Suite.

Until you run the final cell in the notebook, any changes will only affect the Expectation Suite in memory. They won't yet be saved for later.

4. Save your Expectations and review them in Data Docs

At the bottom of the notebook, there's a cell labeled "Save & Review Your Expectations." It will:

1. Save your Expectations back to the configured Expectation Store.
 2. Validate the batch, using a Validation Operator. (This will automatically store Validation Results.)
 3. Rebuild Data Docs.
 4. Open your Data Docs to the appropriate validation page.
- #### 5. Delete the notebook

In general, these Jupyter notebooks should not be kept in source control. In almost all cases, it's better to treat the Expectations as the source of truth, and delete the notebook to avoid confusion. (You can always auto-generate another one later.)

The notebook will be stored in the `great_expectations/uncommitted` directory. You can remove it like so:

```
rm great_expectations/uncommitted/edit_movieratings.ratings.ipynb
```

Content

3.5.8 How to quickly explore data using Expectations in a notebook

Building *Expectations* as you conduct exploratory data analysis is a great way to ensure that your insights about data processes and pipelines remain part of your team's knowledge.

This guide will help you quickly get a taste of Great Expectations, without even setting up a *Data Context*. All you need is a notebook and some data.

Prerequisites: This how-to guide assumes you have already:

- Installed Great Expectations (e.g. `pip install great_expectations`)
- Have access to a notebook (e.g. `jupyter notebook`, `jupyter lab`, etc.)
- Obtained a sample of data to use for exploration

Note: Unlike most how-to guides, these instructions do *not* assume that you have already configured a Data Context by running `great_expectations init`. Once you're comfortable with these basic concepts, you will almost certainly want to unlock the full power of Great Expectations by configuring a Data Context. Please check out the instructions in the [Getting started](#) tutorial when you're ready to start.

Steps

All of these steps take place within your notebook:

1. Import Great Expectations.

```
import great_expectations as ge
```

2. Load some data.

The simplest way to do this is with `read_csv`.

```
my_df = ge.read_csv("my_data_directory/titanic.csv")
```

This method behaves exactly the same as `pandas.read_csv`, so you can add parameters to parse your file:

```
my_df = ge.read_csv(
    "my_data_directory/my_messy_data.csv",
    sep="\t",
    skiprows=3
)
```

Similarly wrapped versions of other pandas methods (`read_excel`, `read_table`, `read_parquet`, `read_pickle`, `read_json`, etc.) are also available. Please see the `great_expectations.utils` module for details.

If you wish to load data from somewhere else (e.g. from a SQL database or blob store), please fetch a copy of the data locally. Alternatively, you can [configure a Data Context with Datasources](#), which will allow you to take advantage of more of Great Expectations' advanced features.

As another option, if you have already instantiated a `pandas.DataFrame`, you can use `from_pandas`:

```
my_df = ge.from_pandas(
    my_pandas_dataframe
)
```

This method will convert your boring old pandas `DataFrame` into a new and exciting `great_expectations PandasDataset`. The two classes are absolutely identical, except that `PandasDataset` has access to Great Expectations' methods.

3. Explore your data and add Expectations.

Each of the methods in step 1 will produce `my_df`, a `PandasDataset`. `PandasDataset` is a subclass of `pandas.DataFrame`, which means that you can use all of pandas' normal methods on it.

```
my_df.head()
my_df.Sex.value_counts()
my_df[my_df.Sex=="male"].head()
# etc., etc.
```

In addition, `my_df` has access to a wide array of Expectations. You can see the full list [here](#). By convention, every Expectation method name starts with the name `expect_...`, so you can quickly access the full list with tab-based autocomplete:

When you invoke an Expectation, it will immediately be validated against your data. The returned object will contain the result and a list of unexpected values. This instant feedback helps you zero in on unexpected data very quickly, taking a lot of the guesswork out of data exploration.

Hint: it's common to encounter data issues where most cases match, but you can't guarantee 100% adherence. In these cases, consider using a `mostly` parameter. This parameter is an option for all Expectations that are applied on a row-by-row basis, and allows you to control the level of wiggle room you want built into your data validation.

Fig. 1: Note how `success` switches from `false` to `true` once `mostly=.99` is added.

4. Review your Expectations.

As you run Expectations in your notebook, `my_df` will build up a running list of Expectations. By default, Great Expectations will recognize and replace duplicate Expectations, so that only the most recent version is stored. (See [Determining duplicate results](#) below for details.)

You can get the config file for your Expectations by running:

```
my_df.get_expectation_suite()
```

which will return an `ExpectationSuite` object.

By default, `get_expectation_suite()` only returns Expectations with `success=True` on their most recent validation. You can override this behavior with:

```
my_df.get_expectation_suite(discard_failed_expectations=False)
```

5. Save your Expectation Suite.

Expectation Suites can be serialized as JSON objects, so you can save your Expectation Suite like this:

```
import json

with open( "my_expectation_file.json", "w") as my_file:
    my_file.write(
        json.dumps(my_df.get_expectation_suite().to_json_dict())
    )
```

As you develop more Expectation Suites, you'll probably want some kind of system for naming and organizing them, not to mention matching them up with data, validating them, and keeping track of validation results.

When you get to this stage, we recommend following the *Getting started* tutorial to set up a *Data Context*. You can get through the basics in less than half an hour, and setting up a Data Context will unlock many additional power tools within Great Expectations.

Additional notes

Adding notes and metadata

You can also add notes and structured metadata to Expectations:

```
>> my_df.expect_column_values_to_match_regex(
    "Name",
    "^[A-Za-z\\, \\(\\)\\'|]+$",
    meta = {
        "notes": "A simple experimental regex for name matching.",
        "source": "max@company.com"
    }
)
```

Determining duplicate results

As a general rule,

- If a given Expectation has no `column` parameters, it will replace another Expectation(s) of the same type.

Example:

```
expect_table_row_count_to_equal(100)
```

will overwrite

```
expect_table_row_count_to_equal(200)
```

- If a given Expectation has one or more `column` parameters, it will replace another Expectation(s) of the same type with the same column parameter(s).

Example:

```
expect_column_values_to_be_between(
    column="percent_agree",
    min_value=0,
```

(continues on next page)

(continued from previous page)

```
max_value=100,
)
```

will overwrite

```
expect_column_values_to_be_between(
    column="percent_agree",
    min_value=10,
    max_value=90,
)
```

or

```
expect_column_values_to_be_between(
    column="percent_agree",
    min_value=0,
    max_value=100,
    mostly=.80,
)
```

but not

```
expect_column_values_to_be_between(
    column="percent_agreement",
    min_value=0,
    max_value=100,
    mostly=.80,
)
```

and not

```
expect_column_mean_to_be_between(
    column="percent",
    min_value=65,
    max_value=75,
)
```

Additional resources

- *Glossary of Expectations*

Comments

Spare parts:

Warning: Articles in Spare Parts are leftovers from the previous version of Great Expectations' documentation. Some content may be outdated or incorrect. We are actively working to move the relevant parts of this documentation into the structured How-To guides above.

If you'd like to help, please reach out in the [#contributors-contributing](#) channel in [slack](#).

3.5.9 How to create a new Expectation Suite using the CLI

While you could hand-author an Expectation Suite by writing a JSON file, just like with other features it is easier to let *CLI* save you time and typos. Run this command in the root directory of your project (where the `init` command created the `great_expectations` subdirectory):

```
great_expectations suite new
```

This command prompts you to name your new Expectation Suite and to select a sample batch of data the suite will describe. Then it uses a sample of the selected data to add some initial expectations to the suite. The purpose of these is expectations is to provide examples of data assertions, and not to be meaningful. They are intended only a starting point for you to build upon.

The command concludes by saving the newly generated Expectation Suite as a JSON file and rendering the expectation suite into an HTML page in Data Docs.

3.5.10 How to create custom Expectations

This document provides examples that walk through several methods for building and deploying custom expectations. Most of the core Great Expectations expectations are built using expectation decorators, and using decorators on existing logic can make bringing custom integrations into your pipeline tests easy.

Using Expectation Decorators

Under the hood, `great_expectations` evaluates similar kinds of expectations using standard logic, including:

- *column_map_expectations*, which apply their condition to each value in a column independently of other values
- *column_aggregate_expectations*, which apply their condition to an aggregate value or values from the column

In general, if a column is empty, a *column_map_expectation* will return `True` (vacuously), whereas a *column_aggregate_expectation* will return `False` (since no aggregate value could be computed).

Adding an expectation about element counts to a set of expectations is usually therefore very important to ensure the overall set of expectations captures the full set of constraints you expect.

High-level decorators

High-level decorators may modify the type of arguments passed to their decorated methods and do significant computation or bookkeeping to augment the work done in an expectation. That makes implementing many common types of operations using high-level decorators very easy.

To use the high-level decorators (e.g. `column_map_expectation` or `column_aggregate_expectation`):

1. Create a subclass from the dataset class of your choice
2. Define custom functions containing your business logic
3. Use the *column_map_expectation* and *column_aggregate_expectation* decorators to turn them into full Expectations. Note that each dataset class implements its own versions of `@column_map_expectation` and `@column_aggregate_expectation`, so you should consult the documentation of each class to ensure you are returning the correct information to the decorator.

Note: following Great Expectations patterns for extending `great_expectations` is highly recommended, but not strictly required. If you want to confuse yourself with bad names, the package won't stop you.

For example, in Pandas:

```
from great_expectations.dataset import PandasDataset, MetaPandasDataset

class CustomPandasDataset(PandasDataset):

    _data_asset_type = "CustomPandasDataset"

    @MetaPandasDataset.column_map_expectation
    def expect_column_values_to_equal_2(self, column):
        return column.map(lambda x: x==2)

    @MetaPandasDataset.column_aggregate_expectation
    def expect_column_mode_to_equal_0(self, column):
        mode = column.mode[0]
        return {
            "success": mode == 0,
            "result": {
                "observed_value": mode,
            }
        }
```

For SQLAlchemyDataset and SparkDFDataset, the decorators work slightly differently. See the respective Meta-class docstrings for more information; the example below shows SQLAlchemy implementations of the same custom expectations.

```
import sqlalchemy as sa
from great_expectations.dataset import SQLAlchemyDataset, MetaSQLAlchemyDataset

class CustomSQLAlchemyDataset(SQLAlchemyDataset):

    _data_asset_type = "CustomSQLAlchemyDataset"

    @MetaSQLAlchemyDataset.column_map_expectation
    def expect_column_values_to_equal_2(self, column):
        return (sa.column(column) == 2)

    @MetaSQLAlchemyDataset.column_aggregate_expectation
    def expect_column_mode_to_equal_0(self, column):
        mode_query = sa.select([
            sa.column(column).label('value'),
            sa.func.count(sa.column(column)).label('frequency')
        ]).select_from(self._table).group_by(sa.column(column)).order_by(sa.desc(sa.
        ↪column('frequency')))

        mode = self.engine.execute(mode_query).scalar()
        return {
            "success": mode == 0,
            "result": {
                "observed_value": mode,
            }
        }
```

Using the base Expectation decorator

When the high-level decorators do not provide sufficient granularity for controlling your expectation's behavior, you need to use the base expectation decorator, which will handle storing and retrieving your expectation in an expectation suite, and facilitate validation using your expectation. You will need to explicitly declare the parameters.

1. Create a subclass from the dataset class of your choice
2. Write the whole expectation yourself
3. Decorate it with the `@expectation` decorator, declaring the parameters you will use.

This is more complicated, since you have to handle all the logic of additional parameters and output formats. Pay special attention to proper formatting of *result_format*.

```
from great_expectations.data_asset import DataAsset
from great_expectations.dataset import PandasDataset

class CustomPandasDataset(PandasDataset):

    _data_asset_type = "CustomPandasDataset"

    @DataAsset.expectation(["column", "mostly"])
    def expect_column_values_to_equal_1(self, column, mostly=None):
        not_null = self[column].notnull()

        result = self[column][not_null] == 1
        unexpected_values = list(self[column][not_null][result==False])

        if mostly:
            #Prevent division-by-zero errors
            if len(not_null) == 0:
                return {
                    "success":True,
                    "result": {
                        'unexpected_list':unexpected_values,
                        'unexpected_index_list':self.index[result],
                    }
                }

            percent_equaling_1 = float(sum(result))/len(not_null)
            return {
                "success" : percent_equaling_1 >= mostly,
                "result": {
                    "unexpected_list" : unexpected_values[:20],
                    "unexpected_index_list" : list(self.index[result==False])[:20],
                }
            }
        else:
            return {
                "success" : len(unexpected_values) == 0,
                "result": {
                    "unexpected_list" : unexpected_values[:20],
                    "unexpected_index_list" : list(self.index[result==False])[:20],
                }
            }
    }
```

A similar implementation for SQLAlchemy would also import the base decorator:


```

import sqlalchemy as sa
from great_expectations.data_asset import DataAsset
from great_expectations.dataset import SqlAlchemyDataset

import numpy as np
import scipy.stats as stats
import scipy.special as special

if sys.version_info.major >= 3 and sys.version_info.minor >= 5:
    from math import gcd
else:
    from fractions import gcd

class CustomSqlAlchemyDataset(SqlAlchemyDataset):

    _data_asset_type = "CustomSqlAlchemyDataset"

    @DataAsset.expectation(["column_A", "column_B", "p_value", "mode"])
    def expect_column_pair_histogram_ks_2samp_test_p_value_to_be_greater_than(
        self,
        column_A,
        column_B,
        p_value=0.05,
        mode='auto'
    ):
        """Execute the two sample KS test on two columns of data that are expected to
        be **histograms** with
        aligned values/points on the CDF. """
        LARGE_N = 10000 # 'auto' will attempt to be exact if n1,n2 <= LARGE_N

        # We will assume that these are already HISTOGRAMS created as a check_dataset
        # either of binned values or of (ordered) value counts
        rows = sa.select([
            sa.column(column_A).label("col_A_counts"),
            sa.column(column_B).label("col_B_counts")
        ]).select_from(self._table).fetchall()

        cols = [col for col in zip(*rows)]
        cdf1 = np.array(cols[0])
        cdf2 = np.array(cols[1])
        n1 = cdf1.sum()
        n2 = cdf2.sum()
        cdf1 = cdf1 / n1
        cdf2 = cdf2 / n2

        # This code is taken verbatim from scipy implementation,
        # skipping the searchsorted (using sqlalchemy check asset as a view)
        # https://github.com/scipy/scipy/blob/v1.3.1/scipy/stats/stats.py#L5385-L5573
        cddiffs = cdf1 - cdf2
        minS = -np.min(cddiffs)
        maxS = np.max(cddiffs)
        alt2Dvalue = {'less': minS, 'greater': maxS, 'two-sided': max(minS, maxS)}
        d = alt2Dvalue[alternative]
        g = gcd(n1, n2)
        n1g = n1 // g
        n2g = n2 // g
        prob = -np.inf

```

(continues on next page)

(continued from previous page)

```

original_mode = mode
if mode == 'auto':
    if max(n1, n2) <= LARGE_N:
        mode = 'exact'
    else:
        mode = 'asyp'
elif mode == 'exact':
    # If lcm(n1, n2) is too big, switch from exact to asyp
    if n1g >= np.iinfo(np.int).max / n2g:
        mode = 'asyp'
        warnings.warn(
            "Exact ks_2samp calculation not possible with samples sizes "
            "%d and %d. Switching to 'asyp' " % (n1, n2), RuntimeWarning)

saw_fp_error = False
if mode == 'exact':
    lcm = (n1 // g) * n2
    h = int(np.round(d * lcm))
    d = h * 1.0 / lcm
    if h == 0:
        prob = 1.0
    else:
        try:
            if alternative == 'two-sided':
                if n1 == n2:
                    prob = stats._compute_prob_outside_square(n1, h)
                else:
                    prob = 1 - stats._compute_prob_inside_method(n1, n2, g, h)
            else:
                if n1 == n2:
                    # prob = binom(2n, n-h) / binom(2n, n)
                    # Evaluating in that form incurs roundoff errors
                    # from special.binom. Instead calculate directly
                    prob = 1.0
                    for j in range(h):
                        prob = (n1 - j) * prob / (n1 + j + 1)
                else:
                    num_paths = stats._count_paths_outside_method(n1, n2, g,
↪h)

                    bin = special.binom(n1 + n2, n1)
                    if not np.isfinite(bin) or not np.isfinite(num_paths) or
↪num_paths > bin:
                        raise FloatingPointError()
                    prob = num_paths / bin

        except FloatingPointError:
            # Switch mode
            mode = 'asyp'
            saw_fp_error = True
            # Can't raise warning here, inside the try
    finally:
        if saw_fp_error:
            if original_mode == 'exact':
                warnings.warn(
                    "ks_2samp: Exact calculation overflowed. "
                    "Switching to mode=%s" % mode, RuntimeWarning)
            else:

```

(continues on next page)

(continued from previous page)

```

        if prob > 1 or prob < 0:
            mode = 'asyp'
            if original_mode == 'exact':
                warnings.warn(
                    "ks_2samp: Exact calculation incurred large"
                    " rounding error. Switching to mode=%s" % mode,
                    RuntimeWarning)

    if mode == 'asyp':
        # The product n1*n2 is large. Use Smirnov's asymptotic formula.
        if alternative == 'two-sided':
            en = np.sqrt(n1 * n2 / (n1 + n2))
            # Switch to using kstwo.sf() when it becomes available.
            # prob = distributions.kstwo.sf(d, int(np.round(en)))
            prob = distributions.kstwobign.sf(en * d)
        else:
            m, n = max(n1, n2), min(n1, n2)
            z = np.sqrt(m*n/(m+n)) * d
            # Use Hodges' suggested approximation Eqn 5.3
            expt = -2 * z**2 - 2 * z * (m + 2*n)/np.sqrt(m*n*(m+n))/3.0
            prob = np.exp(expt)

    prob = (0 if prob < 0 else (1 if prob > 1 else prob))

    return {
        "success": prob > p_value,
        "result": {
            "observed_value": prob,
            "details": {
                "ks_2samp_statistic": d
            }
        }
    }
}

```

Using custom expectations

Let's suppose you've defined *CustomPandasDataset* in a module called *custom_dataset.py*. You can instantiate a dataset with your custom expectations simply by adding *dataset_class=CustomPandasDataset* in *ge.read_csv*.

Once you do this, all the functionality of your new expectations will be available for uses.

```

>> import great_expectations as ge
>> from custom_dataset import CustomPandasDataset

>> my_df = ge.read_csv("my_data_file.csv", dataset_class=CustomPandasDataset)

>> my_df.expect_column_values_to_equal_1("all_twos")
{
    "success": False,
    "unexpected_list": [2, 2, 2, 2, 2, 2, 2, 2, 2]
}

```

A similar approach works for the command-line tool.

```
>> great_expectations validation csv \
    my_data_file.csv \
    my_expectations.json \
    dataset_class=custom_dataset.CustomPandasDataset
```

Using custom expectations with a Datasource

To use custom expectations in a datasource or DataContext, you need to define the custom DataAsset in the datasource configuration or batch_kwarg for a specific batch. Following the same example above, let's suppose you've defined *CustomPandasDataset* in a module called *custom_dataset.py*. You can configure your datasource to return instances of your custom DataAsset type by declaring that as the *data_asset_type* for the datasource to build.

If you are working a DataContext, simply placing *custom_dataset.py* in your configured plugin directory will make it accessible, otherwise, you need to ensure the module is on the import path.

Once you do this, all the functionality of your new expectations will be available for use. For example, you could use the datasource snippet below to configure a PANDASDataSource that will produce instances of your new CustomPandasDataset in a DataContext. Note the use of standard python dot notation to import.

```
datasources:
  my_datasource:
    class_name: PANDASDataSource
    data_asset_type:
      module_name: custom_module.custom_dataset
      class_name: CustomPandasDataset
    generators:
      default:
        class_name: SubdirReaderBatchKwargGenerator
        base_directory: /data
        reader_options:
          sep: \t
```

Note that we need to have added our **custom_dataset.py** to a directory called **custom_module** as in the directory structure below.

```
great_expectations
├── .gitignore
├── datasources
├── expectations
├── great_expectations.yml
├── notebooks
│   ├── pandas
│   ├── spark
│   └── sql
├── plugins
│   └── custom_module
│       └── custom_dataset.py
├── uncommitted
│   ├── config_variables.yml
│   ├── data_docs
│   │   └── local_site
│   ├── samples
│   └── validations
```

```
>> import great_expectations as ge
>> context = ge.DataContext()
>> my_df = context.get_batch(
    "my_datasource/default/my_file",
    "warning",
    context.yield_batch_kwargs("my_datasource/default/my_file"))

>> my_df.expect_column_values_to_equal_1("all_twos")
{
    "success": False,
    "unexpected_list": [2, 2, 2, 2, 2, 2, 2, 2]
}
```

3.5.11 How to create Expectations

This tutorial covers the workflow of creating and editing expectations.

The tutorial assumes that you have created a new Data Context (project), as covered here: [tutorial_init](#).

Creating expectations is an opportunity to blend contextual knowledge from subject-matter experts and insights from profiling and performing exploratory analysis on your dataset.

Once the initial setup of Great Expectations is complete, the workflow looks like a loop over the following steps:

1. Data team members capture and document their shared understanding of their data as expectations.
2. As new data arrives in the pipeline, Great Expectations evaluates it against these expectations.
3. If the observed properties of the data are found to be different from the expected ones, the team responds by rejecting (or fixing) the data, updating the expectations, or both.

For a broader understanding of the typical workflow read this article: [typical_workflow](#).

Expectations are grouped into Expectations Suites. An Expectation Suite combines multiple expectations into an overall description of a dataset. For example, a team can group all the expectations about the `rating` table in the movie ratings database into an Expectation Suite and call it “`movieratings.table.expectations`”.

Each Expectation Suite is saved as a JSON file in the `great_expectations/expectations` subdirectory of the Data Context. Users check these files into version control each time they are updated, in the same way they treat their source files.

The lifecycle of an Expectation Suite starts with creating it. Then it goes through a loop of Review and Edit as the team’s understanding of the data described by the suite evolves.

We will describe the Create, Review and Edit steps in brief:

Create an Expectation Suite

Expectation Suites are saved as JSON files, so you *could* create a new suite by writing a file directly. However the preferred way is to let the CLI save you time and typos. If you cannot use the *CLI* in your environment (e.g., in a Databricks cluster), you can create and edit an Expectation Suite in a notebook. Jump to this section for details: [Jupyter Notebook for Creating and Editing Expectation Suites](#).

To continue with the *CLI*, run this command in the root directory of your project (where the `init` command created the `great_expectations` subdirectory:

```
great_expectations suite new
```

This command prompts you to name your new Expectation Suite and to select a sample batch of the dataset the suite will describe. Then it profiles the selected sample and adds some initial expectations to the suite. The purpose of these expectations is to provide examples of what properties of data can be described using Great Expectations. They are only a starting point that the user builds on.

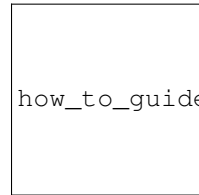
The command concludes by saving the newly generated Expectation Suite as a JSON file and rendering the expectation suite into an HTML page in the Data Docs website of the Data Context.

Review an Expectation Suite

Data Docs is a feature of Great Expectations that creates data documentation by compiling expectations and validation results into HTML.

Data Docs produces a visual data quality report of what you expect from your data, and how the observed properties of your data differ from your expectations. It helps to keep your entire team on the same page as data evolves.

Reviewing expectations is best done in Data Docs:



how_to_guides/creating_and_editing_expectations/spare_parts/./images/sample_e_s_view.png

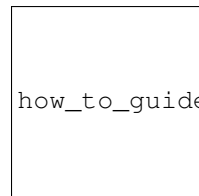
Edit an Expectation Suite

The best interface for editing an Expectation Suite is a Jupyter notebook.

Editing an Expectation Suite means adding expectations, removing expectations, and modifying the arguments of existing expectations.

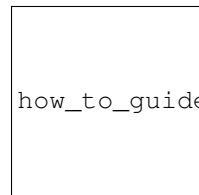
For every expectation type there is a Python method that sets its arguments, evaluates this expectation against a sample batch of data and adds it to the Expectation Suite.

Take a look at the screenshot below. It shows the HTML view and the Python method for the same expectation (`expect_column_distinct_values_to_be_in_set`) side by side:



how_to_guides/creating_and_editing_expectations/spare_parts/./images/exp_html_python_side

The *CLI* provides a command that, given an Expectation Suite, generates a Jupyter notebook to edit it. It takes care of generating a cell for every expectation in the suite and of getting a sample batch of data. The HTML page for each Expectation Suite has the CLI command syntax in order to make it easier for users.



The generated Jupyter notebook can be discarded, since it is auto-generated.

To understand this auto-generated notebook in more depth, jump to this section: [Jupyter Notebook for Creating and Editing Expectation Suites](#).

Jupyter Notebook for Creating and Editing Expectation Suites

If you used the `CLI suite new` command to create an Expectation Suite and then the `suite edit` command to edit it, then the CLI generated a notebook in the `great_expectations/uncommitted/` folder for you. There is no need to check this notebook in to version control. Next time you decide to edit this Expectation Suite, use the `CLI` again to generate a new notebook that reflects the expectations in the suite at that time.

If you do not use the `CLI`, create a new notebook in the `great_expectations/notebooks/`` folder in your project.

1. Setup

```
from datetime import datetime
import great_expectations as ge
import great_expectations.jupyter_ux
from great_expectations.data_context.types.resource_identifiers import _
↳ ValidationResultIdentifier

# Data Context is a GE object that represents your project.
# Your project's great_expectations.yml contains all the config
# options for the project's GE Data Context.
context = ge.data_context.DataContext()

# Create a new empty Expectation Suite
# and give it a name
expectation_suite_name = "ratings.table.warning" # this is just an example
context.create_expectation_suite(
    expectation_suite_name)
```

If an expectation suite with this name already exists for this data_asset, you will get an error. If you would like to overwrite this expectation suite, set `overwrite_existing=True`.

2. Load a batch of data to create Expectations

Select a sample batch of the dataset the suite will describe.

`batch_kwargs` provide detailed instructions for the Datasource on how to construct a batch. Each Datasource accepts different types of `batch_kwargs` - regardless of Datasource type, a Datasource name must always be provided:

pandas

A pandas datasource can accept `batch_kwargs` that describe either a path to a file or an existing DataFrame. For example, if the data asset is a collection of CSV files in a folder that are processed with Pandas, then a batch could be one of these files. Here is how to construct `batch_kwargs` that specify a particular file to load:

```
batch_kwargs = {
    'path': "PATH_OF_THE_FILE_YOU_WANT_TO_LOAD",
    'datasource': "DATASOURCE_NAME"
}
```

To instruct `get_batch` to read CSV files with specific options (e.g., not to interpret the first line as the header or to use a specific separator), add them to the `batch_kwargs` under the “reader_options” key.

See the complete list of options for Pandas `read_csv`.

`batch_kwargs` might look like the following:

```
{
  "path": "/data/npidata/npidata_pfile_20190902-20190908.csv",
  "datasource": "files_datasource",
  "reader_options": {
    "sep": "|"
  }
}
```

If you already loaded the data into a Pandas DataFrame called `df`, you could use following `batch_kwargs` to instruct the datasource to use your DataFrame as a batch:

```
batch_kwargs = {
  'dataset': df,
  'datasource': 'files_datasource'
}
```

pyspark

A pyspark datasource can accept `batch_kwargs` that describe either a path to a file or an existing DataFrame. For example, if the data asset is a collection of CSV files in a folder that are processed with Pandas, then a batch could be one of these files. Here is how to construct `batch_kwargs` that specify a particular file to load:

```
batch_kwargs = {
  'path': "PATH_OF_THE_FILE_YOU_WANT_TO_LOAD",
  'datasource': "DATASOURCE_NAME"
}
```

To instruct `get_batch` to read CSV files with specific options (e.g., not to interpret the first line as the header or to use a specific separator), add them to the `batch_kwargs` under the “reader_options” key.

See the complete list of options for `Spark DataFrameReader`

SQLAlchemy

A SQLAlchemy datasource can accept `batch_kwargs` that instruct it load a batch from a table, a view, or a result set of a query:

If you would like to validate an entire table (or a view) in your database’s default schema:

```
batch_kwargs = {
  'table': "YOUR TABLE NAME",
  'datasource': "DATASOURCE_NAME"
}
```

If you would like to validate an entire table or view from a non-default schema in your database:

```
batch_kwargs = {
  'table': "YOUR TABLE NAME",
  'schema': "YOUR SCHEMA",
  'datasource': "DATASOURCE_NAME"
}
```

If you would like to validate using a query to construct a temporary table:


```
batch_kwargs = {
    'query': 'SELECT YOUR_ROWS FROM YOUR_TABLE',
    'datasource': "DATASOURCE_NAME"
}
```

The DataContext's `get_batch` method is used to load a batch of a data asset:

```
batch = context.get_batch(batch_kwargs, expectation_suite_name)
batch.head()
```

Calling this method asks the Context to get a batch of data and attach the expectation suite `expectation_suite_name` to it. The `batch_kwargs` argument specifies which batch of the data asset should be loaded.

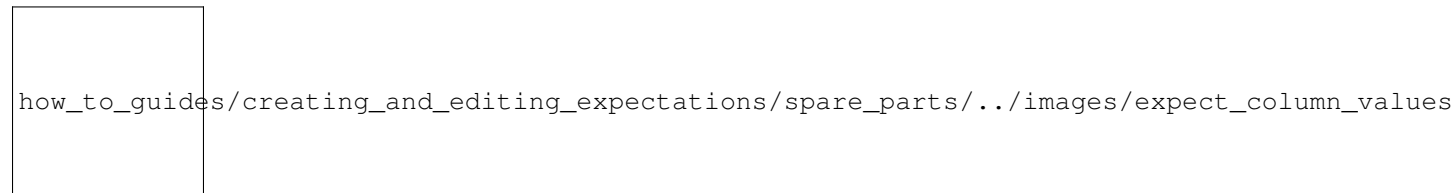
3. Author Expectations

Now that you have a batch of data, you can call `expect` methods on the data asset in order to check whether this expectation is true for this batch of data.

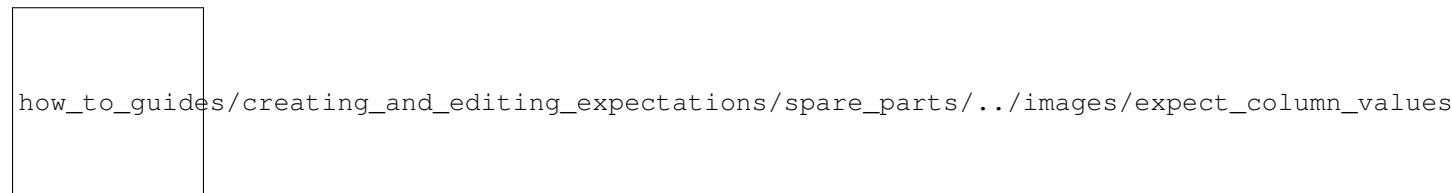
For example, to check whether it is reasonable to expect values in the column “NPI” to never be empty, call: `batch.expect_column_values_to_not_be_null('NPI')`

Some expectations can be created from your domain expertise; for example we might expect that most entries in the NPI database use the title “Dr.” instead of “Ms.”, or we might expect that every row should use a unique value in the ‘NPI’ column.

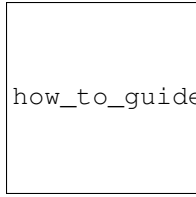
Here is how we can add an expectation that expresses that knowledge:



Other expectations can be created by examining the data in the batch. For example, suppose you want to protect a pipeline against improper values in the “Provider Other Organization Name Type Code” column. Even if you don’t know exactly what the “improper” values are, you can explore the data by trying some values to check if the data in the batch meets your expectation:



Validating the expectation against the batch resulted in failure - there are some values in the column that do not meet the expectation. The “`partial_unexpected_list`” key in the result dictionary contains examples of non-conforming values. Examining these examples shows that some titles are not in the expected set. Adjust the `value_set` and rerun the expectation method:



how_to_guides/creating_and_editing_expectations/spare_parts/../../images/expect_column_values

This time validation was successful - all values in the column meet the expectation.

Although you called `expect_column_values_to_be_in_set` twice (with different argument values), only one expectation of type `expect_column_values_to_be_in_set` will be created for the column - the latest call overrides all the earlier ones. By default, only expectations that were true on their last run are saved.

How do I know which types of expectations I can add?

- *Tab-complete* the partially typed `expect_` method name to see available expectations.
- In Jupyter, you can also use *shift-tab* to see the docstring for each expectation, including the parameters it takes and to get more information about the expectation.
- Visit the [Glossary of Expectations](#) for a complete list of expectations that are currently part of the great expectations vocabulary. Here is a short preview of the glossary:



how_to_guides/creating_and_editing_expectations/spare_parts/../../images/glossary_of_expectat

4. Finalize

Data Docs compiles Expectations and Validations into HTML documentation. By default the HTML website is hosted on your local filesystem. When you are working in a team, the website can be hosted in the cloud (e.g., on S3) and serve as the shared source of truth for the team working on the data pipeline.

To view the expectation suite you just created as HTML, rebuild the data docs and open the website in the browser:

```
# save the Expectation Suite (by default to a JSON file in great_expectations/
↳ expectations folder
batch.save_expectation_suite(discard_failed_expectations=False)

# This step is optional, but useful - evaluate the expectations against the current_
↳ batch of data
run_id = {
    "run_name": "some_string_that_uniquely_identifies_this_run",
    "run_time": datetime.now(datetime.timezone.utc)
}
results = context.run_validation_operator("action_list_operator", assets_to_
↳ validate=[batch], run_id=run_id)
expectation_suite_identifier = list(results["details"].keys())[0]
validation_result_identifier = ValidationResultIdentifier(
    expectation_suite_identifier=expectation_suite_identifier,
    batch_identifier=batch.batch_kwargs.to_id(),
    run_id=run_id
)

# Update the Data Docs site to display the new Expectation Suite
# and open the site in the browser
context.build_data_docs()
context.open_data_docs(validation_result_identifier)
```

3.5.12 How to create Expectations between tables

This tutorial covers the workflow of creating and editing [:ref:`Expectations`](#) that relate to data stored in different tables or datasets.

Unfortunately, no Expectations natively support cross-table comparisons today (but we hope to in the future!). Consequently, there are two available paths:

1. Use “check assets”, where you create a new table that joins the two tables; or
2. Use [:ref:`Evaluation Parameters`](#) to supply relevant metrics to expectations.

Check asset Pattern

See [How to use check assets to validate data](#) for more information on the check asset pattern.

Use Evaluation Parameters

To compare two tables using evaluation parameters, you create Expectations for each table, and reference a property from one **:ref: `Validation Result <Validation Results>`_** in the other. To use Evaluation Parameters, both assets need to be validated during the same run.

```
table_1 = context.get_batch(table_1_batch_kwargs, expectation_suite_name='table_1.
↳warning')
table_2 = context.get_batch(table_2_batch_kwargs, expectation_suite_name='table_2.
↳warning')
# Create an expectation that will always pass, but will produce a metric_
↳corresponding to the true observed value
table_1.expect_column_unique_value_count_to_be_between('id', min_value=0, max_
↳value=None)

# Reference the value from the first
table_2.expect_table_row_count_to_equal(value={
    "$PARAMETER": "urn:great_expectations:validations:table_1.warning:expect_column_
↳unique_value_count_to_be_between.result.observed_value:col=id"
})

# Now, validation of both assets within the same run will support a form of cross-
↳table comparison
results = context.run_validation_operator("action_list_operator", assets_to_
↳validate=[table_1, table_2])
```

3.5.13 How to use check assets to validate data

While Great Expectations has nearly *50 built in expectations*, the need for complex data assertions is common.

Check assets are a **design pattern** that enables even more complex and fine-grained data tests, such as:

assertions on slices of the data For example: How many visits occurred last week?

assertions across logical tables For example: Does my visits table join with my patients table?

A check asset is a slice of data that is only created for validation purposes and may not feed into pipeline or analytical output. It helps express your expectations against a data object that more naturally reflects the meaning of the expectation. For example, a Check Asset for event data might be built as an aggregate rollup binning events by the hour in which they occurred. Then, you can easily express expectations about how many events should happen in the day versus the night.

These check assets should be built in your pipeline's native transformation language. For example, if your pipeline is primarily SQL, create an additional table or view that slices the data so that you can use the built in expectations found here: *Glossary of Expectations*.

Using a check asset introduces a new node into your data pipeline. You should clearly name the expectations about a check asset in a way that makes it easy to understand how it is used in the pipeline, for example by creating an **:ref: `Expectation Suite`_** with the name `event_data.time_rollup_check.warning`.

Postgres SQL example.

Let's suppose we have a `visits` table and we want to make an assertion about the typical number of visits in the last 30 days.

1. Create a new table with an obvious name like `visits.last_30_days` that is populated with the following query:

```
SELECT *
FROM visits
WHERE visit_date > current_date - interval '30' day;
```

2. Create a new Expectation Suite for this new table. Again, we recommend using an obvious name such as `visits.last_30_days.warning`.

3. Add an Expectation as follows:

```
batch.expect_table_row_count_to_be_between(min_value=2000, max_value=5000)
```

3.6 Validation

3.6.1 How to add a Validation Operator

This guide will help you add a new instance of a *Validation Operator*. Validation Operators give you the ability to encode business logic around validation, such as validating multiple batches of data together, differentiating between warnings and errors, and kicking off actions based on the results of validation.

As a general rule, Validation Operators should be invoked from within Checkpoints. Separating out the configuration for Validation Operators and Checkpoints can help make Operator code reusable.

Prerequisites: This how-to guide assumes you have already:

- *Set up a working deployment of Great Expectations.*
 - Created at least one Expectation Suite.
 - Created at least one *Checkpoint*. You will need it in order to test that your new Validation Operator is working.
-

Steps

The snippet below shows a portion of your `great_expectations.yml` configuration after you perform the following steps. The steps will explain each line in this snippet.

```
1 validation_operators:
2   action_list_operator:
3     class_name: ActionListValidationOperator
4     action_list:
5       - name: store_validation_result
6         action:
7           class_name: StoreValidationResultAction
8       - name: store_evaluation_params
9         action:
10          class_name: StoreEvaluationParametersAction
```

(continues on next page)

(continued from previous page)

```
11 - name: update_data_docs
12   action:
13     class_name: UpdateDataDocsAction
14 # Next Validation Operator was added manually
15 my_second_validation_operator:
16   class_name: ActionListValidationOperator
17   action_list:
18     - name: store_validation_result
19       action:
20         class_name: StoreValidationResultAction
```

1. Find the `validation_operators` section in your `great_expectations` config file.

Open your project's `great_expectations.yml` configuration file and navigate to the `validation_operators` section (line 1 in the snippet). This section contains the Validation Operator instance `action_list_operator` that was automatically created by the `great_expectations init` CLI command.

2. Add a new Validation Operator block.

Add a new block after the existing Validation Operator instances. The name of the block is the name you are giving to the new Validation Operator instance (line 15 in the snippet). These names must be unique within a project.

3. Pick a `class_name`.

Add a `class_name` attribute in the new block you added in the previous step (line 16 in the snippet). The value is the name of the class that implements the Validation Operator that you are adding. This can be one of the classes that are included in Great Expectations or a class that you implemented. This example adds another instance of `great_expectations.validation_operators.validation_operators.ActionListValidationOperator`.

Note:

- If you are adding a custom Validation Operator, you will have to add a `module_name` attribute in addition to `class_name`. You will find more details about custom Validation Operators in this [guide](#).
-

4. Configure additional fields.

Consult the reference documentation of the class that implements the Validation Operator you are adding for additional properties (required or optional) that are specific to that class. The snippet above configured one such property specific to `ActionListValidationOperator`.

5. Test your configuration.

Test that your new Validation Operator is configured correctly:

1. Open the configuration file of a Checkpoint you created earlier and replace the value of `validation_operator_name` with the value from Step 2 above. The details of Checkpoint configuration can be found in this [guide](#).
2. Run the Checkpoint and verify that no errors are thrown. You can run the Checkpoint from the CLI as explained [here](#) or from Python, as explained [here](#).

Additional notes

Two Validation Operator classes are currently shipped with Great Expectations:

- `ActionListValidationOperator` invokes a configurable list of actions on every Validation Result validation result. Firing a Slack notification and updating Data Docs are examples of these actions.
- `WarningAndFailureExpectationSuitesValidationOperator` extends the class above and allows to group Expectation Suites into two groups - critical and warning.

Additional resources

3.6.2 How to add validations, data, or suites to a Checkpoint

This guide will help you add validations, data or suites to an existing Checkpoint. This is useful if you want to aggregate individual validations (across suites or datasources) into a single Checkpoint.

Prerequisites: This how-to guide assumes you have already:

- *Set up a working deployment of Great Expectations*
 - You have an existing Expectation Suite
 - You have an *existing Checkpoint*
-

Steps

1. First, open your existing Checkpoint in a text editor. It will look similar to this:

```
validation_operator_name: action_list_operator
batches:
  - batch_kwargs:
      path: /home/me/my_project/source_files/npi.csv
      datasource: files_datasource
      reader_method: read_csv
    expectation_suite_names:
      - npi.warning
```

2. To add a second suite (in this example we add `npi.critical`) to your Checkpoint modify the file to look like this:

```
validation_operator_name: action_list_operator
batches:
  - batch_kwargs:
      path: /home/me/my_project/source_files/npi.csv
      datasource: files_datasource
      reader_method: read_csv
    expectation_suite_names:
      - npi.warning
      - npi.critical
```

3. To add a second validation of a batch of data (in this case a table named `npi` from a datasource named `data_lake`) to your Checkpoint modify the file to look like this:

```
validation_operator_name: action_list_operator
batches:
  - batch_kwargs:
      path: /home/me/my_project/source_files/npi.csv
      datasource: files_datasource
      reader_method: read_csv
    expectation_suite_names:
      - npi.warning
      - another_suite
  - batch_kwargs:
      table: npi
      datasource: data_lake
    expectation_suite_names:
      - npi.warning
```

Additional notes

Tip: This is a good way to aggregate validations in a complex pipeline. You could use this feature to **validate multiple source files before and after their ingestion into your data lake**.

If it would be useful to you, please comment with a +1 and feel free to add any suggestions or questions below.

If you want to be a real hero, we'd welcome a pull request. Please see [the Contributing tutorial](#) and [How to write a how to guide](#) to get started.

3.6.3 How to create a new Checkpoint

This guide will help you create a new Checkpoint, which allows you to couple an Expectation Suite with a data set and quickly run a validation.

Prerequisites: This how-to guide assumes you have already:

- *Set up a working deployment of Great Expectations*
 - Created an Expectation Suite
-

Steps

1. First, run the CLI command below.

```
great_expectations checkpoint new my_checkpoint my_suite
```

2. Next, you will be prompted to select a data asset you want to couple with the Expectation Suite.
3. You will then see a message that indicates the checkpoint has been added to your project.

```
A checkpoint named `my_checkpoint` was added to your project!
- To edit this checkpoint edit the checkpoint file: /home/ubuntu/my_project/great_
  ↳ expectations/checkpoints/my_checkpoint.yml
- To run this checkpoint run `great_expectations checkpoint run my_checkpoint`
```

Additional Resources

- [Check out the detailed tutorial on Checkpoints](#)

3.6.4 How to deploy a scheduled Checkpoint with cron

This guide will help you deploy a scheduled Checkpoint with cron.

Prerequisites: This how-to guide assumes you have already:

- [Set up a working deployment of Great Expectations](#)
 - You have [created a Checkpoint](#)
-

1. First, verify that your Checkpoint is runnable via shell:

```
great_expectations checkpoint run my_checkpoint
```

2. Next, to prepare for editing the cron file, you'll need the full path of the project's great_expectations directory.
3. Next, get full path to the great_expectations executable by running:

```
which great_expectations
/full/path/to/your/environment/bin/great_expectations
```

4. Next, open the cron schedule in a text editor. On most operating systems, `crontab -e` will open your cron file in an editor.
5. To run the Checkpoint `my_checkpoint` every morning at 0300, add the following line in the text editor that opens:

```
0 3 * * * /full/path/to/your/environment/bin/great_expectations checkpoint run_
  ↳ ratings --directory /full/path/to/my_project/great_expectations/
```

6. Finally save the text file and exit the text editor.

Additional notes

The five fields correspond to the minute, hour, day of the month, month and day of the week.

It is critical that you have full paths to both the `great_expectations` executable in your project's environment and the full path to the project's `great_expectations/` directory.

If you have not used cron before, we suggest searching for one of the many excellent cron references on the web.

3.6.5 How to implement a custom Validation Operator

Admonition from Mr. Dickens

“Whether I shall turn out to be the hero of my own life, or whether that station will be held by anybody else, these pages must show.”

This guide is a stub. We all know that it will be useful, but no one has made time to write it yet.

If it would be useful to you, please comment with a +1 and feel free to add any suggestions or questions below.

If you want to be a real hero, we'd welcome a pull request. Please see [the Contributing tutorial](#) and [How to write a how to guide](#) to get started.

3.6.6 How to implement custom notifications

Admonition from Mr. Dickens

“Whether I shall turn out to be the hero of my own life, or whether that station will be held by anybody else, these pages must show.”

This guide is a stub. We all know that it will be useful, but no one has made time to write it yet.

If it would be useful to you, please comment with a +1 and feel free to add any suggestions or questions below.

If you want to be a real hero, we'd welcome a pull request. Please see [the Contributing tutorial](#) and [How to write a how to guide](#) to get started.

3.6.7 How to re-render Data Docs as a Validation Action

Admonition from Mr. Dickens

“Whether I shall turn out to be the hero of my own life, or whether that station will be held by anybody else, these pages must show.”

This guide is a stub. We all know that it will be useful, but no one has made time to write it yet.

If it would be useful to you, please comment with a +1 and feel free to add any suggestions or questions below.

If you want to be a real hero, we'd welcome a pull request. Please see [the Contributing tutorial](#) and [How to write a how to guide](#) to get started.

3.6.8 How to run a Checkpoint in Airflow

This guide will help you run a Great Expectations checkpoint in Apache Airflow, which allows you to trigger validation of a data asset using an Expectation Suite directly within an Airflow DAG.

- *Set up a working deployment of Great Expectations*
- *Created an Expectation Suite*
- *Created a checkpoint for that Expectation Suite and a data asset*
- Created an Airflow DAG file

Using checkpoints is the most straightforward way to trigger a validation run from within Airflow. The following sections describe two alternative approaches to accomplishing this.

Running a checkpoint with a BashOperator

You can use a simple *BashOperator* in Airflow to trigger the checkpoint run. The following snippet shows an Airflow task for an Airflow DAG named *dag* that triggers the run of a checkpoint we named *my_checkpoint*:

```
validation_task = BashOperator( task_id='validation_task',      bash_command='great_expectations
                              checkpoint run my_checkpoint', dag=dag
)
```

Running the *checkpoint script* output with a PythonOperator

Another option is to use the output of the *checkpoint script* command and paste it into a method that is called from a PythonOperator in the DAG. This gives you more fine-grained control over how to respond to validation results:

1. Run *checkpoint script*

```
great_expectations checkpoint script my_checkpoint ... A python script was created that runs the
checkpoint named: my_checkpoint
```

- The script is located in *great_expectations/uncommitted/my_checkpoint.py*
- The script can be run with *python great_expectations/uncommitted/my_checkpoint.py*

2. Navigate to the generated Python script and copy the content
3. Create a method in your Airflow DAG file and call it from a PythonOperator:

```
def run_checkpoint(): # paste content from the checkpoint script here

task_run_checkpoint = PythonOperator( task_id='run_checkpoint',
                                     python_callable=run_checkpoint, dag=dag,
)
```

Additional Resources

- [Check out the detailed tutorial on Checkpoints](#)

3.6.9 How to run a Checkpoint in python

This guide will help you run a Checkpoint in python. This is useful if your pipeline environment or orchestration engine does not have shell access.

Prerequisites: This how-to guide assumes you have already:

- [Set up a working deployment of Great Expectations](#)
 - You have [created a Checkpoint](#)
-

Steps

If it would be useful to you, please comment with a +1 and feel free to add any suggestions or questions below.

If you want to be a real hero, we'd welcome a pull request. Please see [the Contributing tutorial](#) and [How to write a how to guide](#) to get started.

1. First, generate the python with the command:

```
great_expectations checkpoint script my_checkpoint
```

2. Next, you will see a message about where the python script was created like:

```
A python script was created that runs the checkpoint named: `my_checkpoint`
- The script is located in `great_expectations/uncommitted/run_my_checkpoint.py`
- The script can be run with `python great_expectations/uncommitted/run_my_
↪checkpoint.py`
```

3. Next, open the script which should look like this:

```
"""
This is a basic generated Great Expectations script that runs a checkpoint.

A checkpoint is a list of one or more batches paired with one or more
Expectation Suites and a configurable Validation Operator.

Checkpoints can be run directly without this script using the
`great_expectations checkpoint run` command. This script is provided for those
who wish to run checkpoints via python.

Data that is validated is controlled by BatchKwargs, which can be adjusted in
the checkpoint file: great_expectations/checkpoints/my_checkpoint.yml.

Data are validated by use of the `ActionListValidationOperator` which is
configured by default. The default configuration of this Validation Operator
saves validation results to your results store and then updates Data Docs.

This makes viewing validation results easy for you and your team.

Usage:
```

(continues on next page)

(continued from previous page)

```

- Run this file: `python great_expectations/uncommitted/run_my_checkpoint.py`.
- This can be run manually or via a scheduler such as cron.
- If your pipeline runner supports python snippets you can paste this into your
  pipeline.
"""
import sys

from great_expectations import DataContext

# checkpoint configuration
context = DataContext("/home/ubuntu/my_project/great_expectations")
checkpoint = context.get_checkpoint("my_checkpoint")

# load batches of data
batches_to_validate = []
for batch in checkpoint["batches"]:
    batch_kwargs = batch["batch_kwargs"]
    for suite_name in batch["expectation_suite_names"]:
        suite = context.get_expectation_suite(suite_name)
        batch = context.get_batch(batch_kwargs, suite)
        batches_to_validate.append(batch)

# run the validation operator
results = context.run_validation_operator(
    checkpoint["validation_operator_name"],
    assets_to_validate=batches_to_validate,
    # TODO prepare for new RunID - checkpoint name and timestamp
    # run_id=RunID(checkpoint)
)

# take action based on results
if not results["success"]:
    print("Validation Failed!")
    sys.exit(1)

print("Validation Succeeded!")
sys.exit(0)

```

4. This python can then be invoked directly using `python great_expectations/uncommitted/run_my_checkpoint.py` or the python code can be embedded in your pipeline.

3.6.10 How to run a Checkpoint in terminal

This guide will help you run a Checkpoint in a terminal.

Prerequisites: This how-to guide assumes you have already:

- *Set up a working deployment of Great Expectations*
 - *Created a Checkpoint*
-

Steps

1. Checkpoints can be run like applications from the command line by running:

```
great_expectations checkpoint run my_checkpoint  
Validation Failed!
```

2. Next, observe the output which will tell you if all validations passed or failed.

Additional notes

This command will return posix status codes and print messages as follows:

Situation	Return code	Message
all validations passed	0	Validation Succeeded!
one or more validation failed	1	Validation Failed!

If you want to be a real hero, we'd welcome a pull request. Please see [the Contributing tutorial](#) and [How to write a how to guide](#) to get started.

3.6.11 How to store Validation Results as a Validation Action

Admonition from Mr. Dickens

“Whether I shall turn out to be the hero of my own life, or whether that station will be held by anybody else, these pages must show.”

This guide is a stub. We all know that it will be useful, but no one has made time to write it yet.

If it would be useful to you, please comment with a +1 and feel free to add any suggestions or questions below.

If you want to be a real hero, we'd welcome a pull request. Please see [the Contributing tutorial](#) and [How to write a how to guide](#) to get started.

3.6.12 How to trigger Slack notifications as a Validation Action

This guide will help you trigger Slack notifications as a *Validation Action*. It will allow you to send a Slack message including information about a Validation Result, including whether or not the Validation succeeded.

Prerequisites: This how-to guide assumes that you have already:

- Configured a Slack app with the Webhook function enabled (See Additional Resources below for more information on setting up a new Slack app).
 - Obtained the Webhook address for your Slack app.
 - Identified the Slack channel that messages will be sent to.
-

Steps

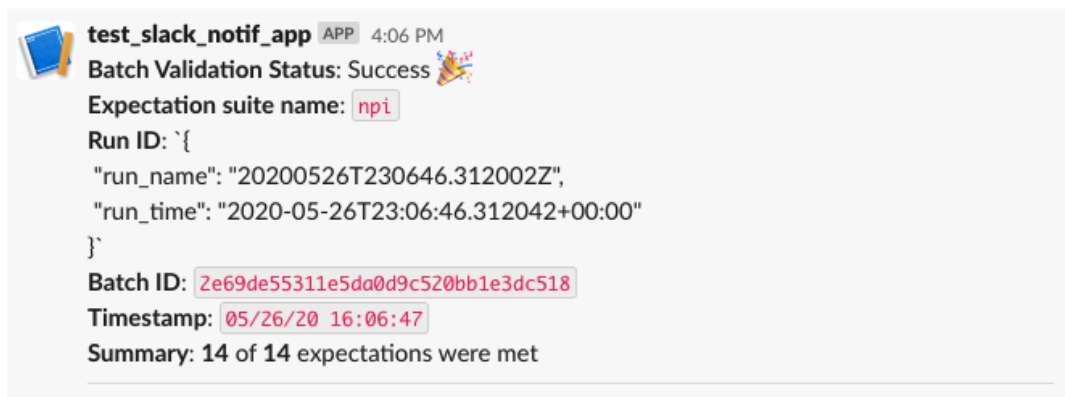
1. Open `uncommitted/config_variables.yml` file and add `validation_notification_slack_webhook` variable by adding the following line:

```
validation_notification_slack_webhook: [address to web hook]
```

2. Open `great_expectations.yml` and add `send_slack_notification_on_validation_result` action to `validation_operators`. Make sure the following section exists in the `great_expectations.yml` file.

```
validation_operators:
  action_list_operator:
    # To learn how to configure sending Slack notifications during evaluation
    # (and other customizations), read: https://docs.greatexpectations.io/en/
    ↪latest/reference/validation_operators/action_list_validation_operator.html
    class_name: ActionListValidationOperator
    action_list:
      #-----
      # here is what you will be adding
      #-----
      - name: send_slack_notification_on_validation_result # name can be set to any_
      ↪value
        action:
          class_name: SlackNotificationAction
          # put the actual webhook URL in the uncommitted/config_variables.yml file
          slack_webhook: ${validation_notification_slack_webhook}
          notify_on: all # possible values: "all", "failure", "success"
          renderer:
            module_name: great_expectations.render.renderers.slack_renderer
            class_name: SlackRenderer
```

3. Run Validation checkpoint to receive Slack notification on the success or failure of validation suite. If successful, you should receive a Slack message that looks like this:



Additional resources

- Instructions on how to set up a Slack app with webhook can be found in the documentation for the [Slack API](#)

3.6.13 How to validate data without a Checkpoint

Admonition from Mr. Dickens

“Whether I shall turn out to be the hero of my own life, or whether that station will be held by anybody else, these pages must show.”

This guide is a stub. We all know that it will be useful, but no one has made time to write it yet.

If it would be useful to you, please comment with a +1 and feel free to add any suggestions or questions below.

If you want to be a real hero, we’d welcome a pull request. Please see [the Contributing tutorial](#) and [How to write a how to guide](#) to get started.

Spare parts:

Warning: Articles in Spare Parts are leftovers from the previous version of Great Expectations’ documentation. Some content may be outdated or incorrect. We are actively working to move the relevant parts of this documentation into the structured How-To guides above.

If you’d like to help, please reach out in the `#contributors-contributing` channel in [slack](#).

3.6.14 How to implement a custom Validation Operator

If you wish to implement some validation handling logic that is not supported by the operators included in Great Expectations, follow these steps:

- Extend the `great_expectations.validation_operators.ValidationOperator` base class
- Implement the `run` method in your new class
- Put your class in the `plugins` directory of your project (see `plugins_directory` property in the `great_expectations.yml` configuration file)

Once these steps are complete, your new Validation Operator can be configured and used.

If you think that the business logic of your operator can be useful to other data practitioners, please consider contributing it to Great Expectations.

3.6.15 Validate Data

Expectations describe Data Assets. Data Assets are composed of Batches. Validation checks Expectations against a Batch of data. Expectation Suites combine multiple Expectations into an overall description of a Batch.

Validation = checking if a Batch of data from a Data Asset X conforms to all Expectations in Expectation Suite Y. Expectation Suite Y is a collection of Expectations that you created that specify what a valid Batch of Data Asset X should look like.

To run Validation you need a **Batch** of data. To get a **Batch** of data you need:

- to provide `batch_kwargs` to a [Data Context](#)

- to specify an **Expectation Suite** to validate against

This tutorial will explain each of these objects, show how to obtain them, execute validation and view its result.

0. Open Jupyter Notebook

This tutorial assumes that:

- you ran `great_expectations init`
- your current directory is the root of the project where you ran `great_expectations init`

You can either follow the tutorial with the sample National Provider Identifier (NPI) dataset (processed with Pandas) referenced in the `great_expectations init` tutorial, or you can execute the same steps on your project with your own data.

If you get stuck, find a bug or want to ask a question, go to [our Slack](#) - this is the best way to get help from the contributors and other users.

Validation is typically invoked inside the code of a data pipeline (e.g., an Airflow operator). This tutorial uses a Jupyter notebook as a validation playground.

The `great_expectations init` command created a `great_expectations/notebooks/` folder in your project. The folder contains example notebooks for pandas, Spark and SQL datasources.

If you are following this tutorial using the NPI dataset, open the pandas notebook. If you are working with your dataset, see the instructions for your datasource:

pandas

```
jupyter notebook great_expectations/notebooks/pandas/validation_playground.ipynb
```

pyspark

```
jupyter notebook great_expectations/notebooks/spark/validation_playground.ipynb
```

SQLAlchemy

```
jupyter notebook great_expectations/notebooks/sql/validation_playground.ipynb
```

1. Get a DataContext Object

A `DataContext` represents a Great Expectations project. It organizes Datasources, notification settings, data documentation sites, and storage and access for Expectation Suites and Validation Results. The `DataContext` is configured via a `yml` file stored in a directory called `great_expectations`; the configuration file as well as managed Expectation Suites should be stored in version control.

Instantiating a `DataContext` loads your project configuration and all its resources.

```
context = ge.data_context.DataContext()
```

To read more about `DataContexts`, see: [DataContexts](#)

2. Choose an Expectation Suite

The `context` instantiated in the previous section has a convenience method that lists all Expectation Suites created in a project:

```
for expectation_suite_id in context.list_expectation_suites():
    print(expectation_suite_id.expectation_suite_name)
```

Choose the Expectation Suite you will use to validate a Batch of data:

```
expectation_suite_name = "warning"
```

3. Load a batch of data you want to validate

Expectations describe Batches of data - Expectation Suites combine multiple Expectations into an overall description of a Batch. Validation checks a Batch against an Expectation Suite.

For example, a Batch could be the most recent day of log data. For a database table, a Batch could be the data in that table at a particular time.

In order to validate a Batch of data, you will load it as a Great Expectations *Dataset*.

Batches are obtained by using a Data Context's `get_batch` method, which accepts `batch_kwargs` and `expectation_suite_name` as arguments.

Calling this method asks the Context to get a Batch of data using the provided `batch_kwargs` and attach the Expectation Suite `expectation_suite_name` to it.

The `batch_kwargs` argument is a dictionary that specifies a batch of data - it contains all the information necessary for a Data Context to obtain a batch of data from a *Datasource*. The keys of a `batch_kwargs` dictionary will vary depending on the type of Datasource and how it generates Batches, but will always have a `datasource` key with the name of a Datasource. To list the Datasources configured in a project, you may use a Data Context's `list_datasources` method.

pandas

A Pandas Datasource generates Batches from Pandas DataFrames or CSV files. A Pandas Datasource can accept `batch_kwargs` that describe either a path to a file or an existing DataFrame:

```
# list datasources of the type PandasDatasource in your project
[datasource['name'] for datasource in context.list_datasources() if datasource['class_
↪name'] == 'PandasDatasource']
datasource_name = # TODO: set to a datasource name from above

# If you would like to validate a file on a filesystem:
batch_kwargs = {'path': "YOUR_FILE_PATH", 'datasource': datasource_name}

# If you already loaded the data into a Pandas Data Frame:
batch_kwargs = {'dataset': "YOUR_DATAFRAME", 'datasource': datasource_name}

batch = context.get_batch(batch_kwargs, expectation_suite_name)
batch.head()
```

pyspark

A Spark Datasource generates Batches from Spark DataFrames or CSV files. A Spark Datasource can accept `batch_kwargs` that describe either a path to a file or an existing DataFrame:

```
# list datasources of the type SparkDFDatasource in your project
[datasource['name'] for datasource in context.list_datasources() if datasource['class_
↳name'] == 'SparkDFDatasource']
datasource_name = # TODO: set to a datasource name from above

# If you would like to validate a file on a filesystem:
batch_kwargs = {'path': "YOUR_FILE_PATH", 'datasource': datasource_name}
# To customize how Spark reads the file, you can add options under reader_options key_
↳in batch_kwargs (e.g., header='true')

# If you already loaded the data into a PySpark Data Frame:
batch_kwargs = {'dataset': "YOUR_DATAFRAME", 'datasource': datasource_name}

batch = context.get_batch(batch_kwargs, expectation_suite_name)
batch.head()
```

SQLAlchemy

A SQLAlchemy Datasource generates Batches from tables, views and query results. A SQLAlchemy Datasource can accept batch_kwargs that instruct it load a batch from a table, a view, or a result set of a query:

```
# list datasources of the type SQLAlchemyDatasource in your project
[datasource['name'] for datasource in context.list_datasources() if datasource['class_
↳name'] == 'SQLAlchemyDatasource']
datasource_name = # TODO: set to a datasource name from above

# If you would like to validate an entire table or view in your database's default_
↳schema:
batch_kwargs = {'table': "YOUR_TABLE", 'datasource': datasource_name}

# If you would like to validate an entire table or view from a non-default schema in_
↳your database:
batch_kwargs = {'table': "YOUR_TABLE", "schema": "YOUR_SCHEMA", 'datasource':_
↳datasource_name}

# If you would like to validate the result set of a query:
# batch_kwargs = {'query': 'SELECT YOUR_ROWS FROM YOUR_TABLE', 'datasource':_
↳datasource_name}

batch = context.get_batch(batch_kwargs, expectation_suite_name)
batch.head()
```

The examples of batch_kwargs above can also be the outputs of “Generators” used by Great Expectations. You can read about the default Generators’ behavior and how to implement additional Generators in this article: [Batch Kwargs Generators](#).

4. Validate the batch

When Great Expectations is integrated into a data pipeline, the pipeline calls GE to validate a specific batch (an input to a pipeline's step or its output).

Validation evaluates the Expectations of an Expectation Suite against the given Batch and produces a report that describes observed values and any places where Expectations are not met. To validate the Batch of data call the `validate()` method on the batch:

```
validation_result = batch.validate()
```

The `validation_result` object has detailed information about every Expectation in the Expectation Suite that was used to validate the Batch: whether the Batch met the Expectation and even more details if it did not. You can read more about the result object's structure here: [Validation Results](#).

You can print this object out:

```
print(json.dumps(validation_result, indent=4))
```

Here is what a part of this object looks like:



how_to_guides/validation/spare_parts/../../images/validation_playground_result_json.png

Don't panic! This blob of JSON is meant for machines. [Data Docs](#) are a compiled HTML view of both expectation suites and validation results that is far more suitable for humans. You will see how easy it is to build them in the next sections.

5. Validation Operators

The `validate()` method evaluates one Batch of data against one Expectation Suite and returns a dictionary of Validation Results. This is sufficient when you explore your data and get to know Great Expectations.

When deploying Great Expectations in a real data pipeline, you will typically discover these additional needs:

- Validating a group of Batches that are logically related (e.g. Did all my Salesforce integrations work last night?).
- Validating a Batch against several Expectation Suites (e.g. Did my nightly clickstream event job have any **critical** failures I need to deal with ASAP or **warnings** I should investigate later?).
- Doing something with the Validation Results (e.g., saving them for a later review, sending notifications in case of failures, etc.).

Validation Operators provide a convenient abstraction for both bundling the validation of multiple Expectation Suites and the actions that should be taken after the validation. See the [Validation Operators And Actions Introduction](#) for more information.

An instance of `action_list_operator` operator is configured in the default `great_expectations.yml` configuration file. `ActionListValidationOperator` validates each Batch in the list that is passed as `assets_to_validate` argument to its `run` method against the Expectation Suite included within that Batch and then invokes a list of configured actions on every Validation Result.

Below is the operator's configuration snippet in the `great_expectations.yml` file:

```
action_list_operator:
  class_name: ActionListValidationOperator
  action_list:
    - name: store_validation_result
      action:
        class_name: StoreValidationResultAction
    - name: store_evaluation_params
      action:
        class_name: StoreEvaluationParametersAction
    - name: update_data_docs
      action:
        class_name: UpdateDataDocsAction
    - name: send_slack_notification_on_validation_result
      action:
        class_name: SlackNotificationAction
        # put the actual webhook URL in the uncommitted/config_variables.yml file
        slack_webhook: ${validation_notification_slack_webhook}
        notify_on: all # possible values: "all", "failure", "success"
        renderer:
          module_name: great_expectations.render.renderers.slack_renderer
          class_name: SlackRenderer
```

We will show how to use the two most commonly used actions that are available to this operator:

Save Validation Results

The DataContext object provides a configurable `validations_store` where GE can store validation_result objects for subsequent evaluation and review. By default, the DataContext stores results in the `great_expectations/uncommitted/validations` directory. To specify a different directory or use a remote store such as s3 or gcs, edit the stores section of the DataContext configuration object:

```
stores:
  validations_store:
    class_name: ValidationsStore
    store_backend:
      class_name: TupleS3StoreBackend
      bucket: my_bucket
      prefix: my_prefix
```

Removing the `store_validation_result` action from the `action_list_operator` configuration will disable automatically storing validation_result objects.

Send a Slack Notification

The last action in the action list of the Validation Operator above sends notifications using a user-provided callback function based on the validation result.

```
- name: send_slack_notification_on_validation_result
  action:
    class_name: SlackNotificationAction
    # put the actual webhook URL in the uncommitted/config_variables.yml file
    slack_webhook: ${validation_notification_slack_webhook}
    notify_on: all # possible values: "all", "failure", "success"
    renderer:
      module_name: great_expectations.render.renderer.slack_renderer
      class_name: SlackRenderer
```

GE includes a slack-based notification in the base package. To enable a slack notification for results, simply specify the slack webhook URL in the `uncommitted/config_variables.yml` file:

```
validation_notification_slack_webhook: https://slack.com/your_webhook_url
```

Running the Validation Operator

Before running the Validation Operator, create a `run_id`. A `run_id` links together validations of different data assets, making it possible to track “runs” of a pipeline and follow data assets as they are transformed, joined, annotated, enriched, or evaluated. The `run_id` must be of type `RunIdentifier`, with optional `run_name` and `run_time` instantiation arguments (or a dictionary with these keys). The `run_name` can be any string (this could come from your pipeline runner, e.g. Airflow run id). The `run_time` can be either a dateutil parsable string or a datetime object. Note - any provided datetime will be assumed to be a UTC time. If no instantiation arguments are given, `run_name` will be `None` and `run_time` will default to the current UTC datetime.

```
run_id = {
  "run_name": "some_string_that_uniquely_identifies_this_run", # insert your own run_
  ↪name here
  "run_time": datetime.now(datetime.timezone.utc)
}
```

When you integrate validation in your pipeline, your pipeline runner probably has a run id that can be inserted here to make smoother integration.

Finally, run the Validation Operator:

```
results = context.run_validation_operator(  
    "action_list_operator",  
    assets_to_validate=[batch],  
    run_id=run_id)
```

6. View the Validation Results in Data Docs

Data Docs compiles raw Great Expectations objects including Expectations and Validations into structured documents such as HTML documentation. By default the HTML website is hosted on your local filesystem. When you are working in a team, the website can be hosted in the cloud (e.g., on S3) and serve as the shared source of truth for the team working on the data pipeline.

Read more about the capabilities and configuration of Data Docs here: [Data Docs](#).

One of the actions executed by the validation operator in the previous section rendered the validation result as HTML and added this page to the Data Docs site.

You can open the page programmatically and examine the result:

```
context.open_data_docs()
```

Congratulations!

Now you know how to validate a Batch of data.

What is next? This is a collection of tutorials that walk you through a variety of useful Great Expectations workflows: [Tutorials](#).

3.7 Configuring Data Docs

3.7.1 How to host and share Data Docs on a filesystem

Admonition from Mr. Dickens

“Whether I shall turn out to be the hero of my own life, or whether that station will be held by anybody else, these pages must show.”

This guide is a stub. We all know that it will be useful, but no one has made time to write it yet.

If it would be useful to you, please comment with a +1 and feel free to add any suggestions or questions below.

If you want to be a real hero, we’d welcome a pull request. Please see [the Contributing tutorial](#) and [How to write a how to guide](#) to get started.

3.7.2 How to host and share Data Docs on Azure Blob Storage

Admonition from Mr. Dickens

“Whether I shall turn out to be the hero of my own life, or whether that station will be held by anybody else, these pages must show.”

This guide is a stub. We all know that it will be useful, but no one has made time to write it yet.

If it would be useful to you, please comment with a +1 and feel free to add any suggestions or questions below.

If you want to be a real hero, we’d welcome a pull request. Please see [the Contributing tutorial](#) and [How to write a how to guide](#) to get started.

3.7.3 How to host and share Data Docs on GCS

Admonition from Mr. Dickens

“Whether I shall turn out to be the hero of my own life, or whether that station will be held by anybody else, these pages must show.”

This guide is a stub. We all know that it will be useful, but no one has made time to write it yet.

If it would be useful to you, please comment with a +1 and feel free to add any suggestions or questions below.

If you want to be a real hero, we’d welcome a pull request. Please see [the Contributing tutorial](#) and [How to write a how to guide](#) to get started.

3.7.4 How to host and share Data Docs on S3

Admonition from Mr. Dickens

“Whether I shall turn out to be the hero of my own life, or whether that station will be held by anybody else, these pages must show.”

This guide is a stub. We all know that it will be useful, but no one has made time to write it yet.

If it would be useful to you, please comment with a +1 and feel free to add any suggestions or questions below.

If you want to be a real hero, we’d welcome a pull request. Please see [the Contributing tutorial](#) and [How to write a how to guide](#) to get started.

3.7.5 How to publish Data Docs to S3

In this tutorial we will cover publishing a data docs site directly to S3. Publishing a site this way makes reviewing and acting on validation results easy in a team, and provides a central location to review the expectations currently configured for data assets under test.

Configuring data docs requires three simple steps:

1. Configure an S3 bucket.

Modify the bucket name and region for your situation.

```
> aws s3api create-bucket --bucket data-docs.my_org --region us-east-1
{
  "Location": "/data-docs.my_org"
}
```

Configure your bucket policy to enable appropriate access. **IMPORTANT:** your policy should provide access only to appropriate users; data-docs can include critical information about raw data and should generally **not** be publicly accessible. The example policy below **enforces IP-based access**.

Modify the bucket name and IP addresses below for your situation.

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Sid": "Allow only based on source IP",
    "Effect": "Allow",
    "Principal": "*",
    "Action": "s3:GetObject",
    "Resource": [
      "arn:aws:s3:::data-docs.my_org",
      "arn:aws:s3:::data-docs.my_org/*"
    ],
    "Condition": {
      "IpAddress": {
        "aws:SourceIp": [
          "192.168.0.1/32",
          "2001:db8:1234:1234::/64"
        ]
      }
    }
  ]
}
```

Modify the policy above and save it to a file called *ip-policy.json* in your local directory. Then, run:

```
> aws s3api put-bucket-policy --bucket data-docs.my_org --policy file://ip-policy.json
```

2. Edit your *great_expectations.yml* file to change the *data_docs_sites* configuration for the site you will publish. **Add the `s3_site` section below existing site configuration.**

```
# ... additional configuration above
data_docs_sites:
  local_site:
    class_name: SiteBuilder
    store_backend:
      class_name: TupleFilesystemStoreBackend
```

(continues on next page)

(continued from previous page)

```
base_directory: uncommitted/data_docs/local_site/
s3_site:
  class_name: SiteBuilder
  store_backend:
    class_name: TupleS3StoreBackend
    bucket: data-docs.my_org # UPDATE the bucket name here to match the bucket you_
→configured above.
# ... additional configuration below
```

3. Build your documentation:

```
> great_expectations docs build
Building...
```

You're now ready to visit the site! Your site will be available at the following URL: http://data-docs.my_org.s3.amazonaws.com/index.html

Additional Resources

Optionally, you may wish to update static hosting settings for your bucket to enable AWS to automatically serve your index.html file or a custom error file:

```
> aws s3 website s3://data-docs.my_org/ --index-document index.html
```

For more information on static site hosting in AWS, see the following:

- [AWS Website Hosting](#)
- [AWS Static Site Access Permissions](#)
- [AWS Website configuration](#)

To discuss with the Great Expectations community, please visit this topic in our community discussion forum:

- <https://discuss.greatexpectations.io/t/ge-with-databricks-delta/82/3>

3.8 Migrating between versions

While we are committed to keeping Great Expectations as stable as possible, sometimes breaking changes are necessary to maintain our trajectory. This is especially true as the library has evolved from just a data quality tool to a more capable framework including data docs and profiling in addition to validation.

Great Expectations provides a warning when the currently-installed version is different from the version stored in the expectation suite.

Since expectation semantics are usually consistent across versions, there is little change required when upgrading great expectations, with some exceptions noted here.

3.8.1 How to use the project check-config command

To facilitate this substantial config format change, starting with version 0.8.0 we introduced `project check-config` to sanity check your config files. From your project directory, run:

```
great_expectations project check-config
```

This can be used at any time and will grow more robust and helpful as our internal config typing system improves.

You will most likely be prompted to install a new template. Rest assured that your original yaml file will be archived automatically for you. Even so, it's in your source control system already, right? ;-)

3.8.2 Upgrading to 0.11.x

The 0.11.0 release has several breaking changes related to `run_id` and `ValidationMetric` objects. Existing projects that have Expectation Suite Validation Results or configured evaluation parameter stores with `DatabaseStoreBackend` backends must be migrated.

In addition, `ValidationOperator.run` now returns an instance of new type, `ValidationOperatorResult` (instead of a dictionary). If your code uses output from `Validation Operators`, it must be updated.

run_id and ValidationMetric Changes

`run_id` is now typed using the new `RunIdentifier` class, with optional `run_name` and `run_time` instantiation arguments. The `run_name` can be any string (this could come from your pipeline runner, e.g. Airflow run id). The `run_time` can be either a `dateutil` parsable string or a `datetime` object. Note - any provided `datetime` will be assumed to be a UTC time. If no instantiation arguments are provided, `run_name` will be `None` (and appear as “__none__” in stores) and `run_time` will default to the current UTC `datetime`. This change affects all Great Expectations classes that have a `run_id` attribute as well as any functions or methods that accept a `run_id` argument.

`data_asset_name` (if available) is now added to `batch_kwargs` by `batch_kwargs_generators`. Because of this newly exposed key in `batch_kwargs`, `ValidationMetric` and associated `ValidationMetricIdentifier` objects now have a `data_asset_name` attribute.

The affected classes that are relevant to existing projects are `ValidationResultIdentifier` and `ValidationMetricIdentifier`, as well as any configured stores that rely on these classes for keys, namely stores of type `ValidationsStore` (and subclasses) or `EvaluationParameterStore` (and other subclasses of `MetricStore`). In addition, because Expectation Suite Validation Result json objects have a `run_id` key, existing validation result json files must be updated with a new typed `run_id`.

Migrating Your 0.10.x Project

Before performing any of the following migration steps, please make sure you have appropriate backups of your project.

Great Expectations has a CLI Upgrade Helper that helps automate all or most of the migration process (affected stores with database backends will still have to be migrated manually). The CLI tool makes use of a new class called `UpgradeHelperV11`. For reference, the `UpgradeHelperV11` class is located at `great_expectations.cli.upgrade_helpers.upgrade_helper_v11`.

To use the CLI Upgrade Helper, enter the following command: `great_expectations project upgrade`

The Upgrade Helper will check your project and guide you through the upgrade process.

Note: The following instructions detail the steps required to upgrade your project manually. The migration steps are written in the order they should be completed. They are also provided in the event that the Upgrade Helper is unable to complete a fully automated upgrade and some user intervention is required.

0. Code That Uses Great Expectations

If you are using any Great Expectations methods that accept a `run_id` argument, you should update your code to pass in the new `RunIdentifier` type (or a dictionary with `run_name` and `run_time` keys). For now, methods with a `run_id` parameter will continue to accept strings. In this case, the provided `run_id` string will be converted to a `RunIdentifier` object, acting as the `run_name`. If the `run_id` string can also be parsed as a datetime, it will also be used for the `run_time` attribute, otherwise, the current UTC time is used. All times are assumed to be UTC times.

If your code uses output from Validation Operators, it must be updated to handle the new `ValidationOperatorResult` type.

1. Expectation Suite Validation Result JSONs

Each existing Expectation Suite Validation Result JSON in your project should be updated with a typed `run_id`. The `run_id` key is found under the top-level `meta` key. You can use the current `run_id` string as the new `run_name` (or select a different one). If the current `run_id` is already a datetime string, you can also use it for the `run_time` as well, otherwise, we suggest using the last modified datetime of the validation result.

Note: Subsequent migration steps will make use of this new `run_time` when generating new paths/keys for validation result jsons and their Data Docs html pages. Please ensure the `run_time` in these paths/keys match the `run_time` in the corresponding validation result. Similarly, if you decide to use a different value for `run_name` instead of reusing an existing `run_id` string, make sure this is reflected in the new paths/keys.

For example, an existing validation result json with `run_id="my_run"` should be updated to look like the following:

```
{
  "meta": {
    "great_expectations.__version__": "0.10.8",
    "expectation_suite_name": "diabetic_data.warning",
    "run_id": {
      "run_name": "my_run",
      "run_time": "20200507T065044.404158Z"
    },
    ...
  },
  ...
}
```

2. Stores and their Backends

Stores rely on special identifier classes to serve as keys when getting or setting values. When the signature of an identifier class changes, any existing stores that rely on that identifier must be updated. Specifically, the structure of that store's backend must be modified to conform to the new identifier signature.

For example, in a v0.10.x project, you might have an Expectation Suite Validation Result with the following `ValidationResultIdentifier`:

```
v10_identifier = ValidationResultIdentifier(
    expectation_suite_identifier=ExpectationSuiteIdentifier(expectation_suite_name="my_
↳suite_name"),
    run_id="my_string_run_id",
    batch_identifier="some_batch_identifier"
)
```

A configured `ValidationsStore` with a `TupleFilesystemStoreBackend` (and default config) would use this identifier to generate the following filepath for writing the validation result to a file (and retrieving it at a later time):

```
v10_filepath = "great_expectations/uncommitted/validations/my_suite_name/my_string_
↳run_id/some_batch_identifier.json"
```

In a v0.11.x project, the `ValidationResultIdentifier` and corresponding filepath would look like the following:

```
v11_identifier = ValidationResultIdentifier(
    expectation_suite_identifier=ExpectationSuiteIdentifier(expectation_suite_name="my_
↳suite_name"),
    run_id=RunIdentifier(run_name="my_string_run_name", run_time="2020-05-08T20:51:18.
↳077262"),
    batch_identifier="some_batch_identifier"
)
v11_filepath = "great_expectations/uncommitted/validations/my_suite_name/my_string_
↳run_name/2020-05-08T20:51:18.077262/some_batch_identifier.json"
```

When migrating to v0.11.x, you would have to move all existing validation results to new filepaths. For a particular validation result, you might move the file like this:

```
os.makedirs(v11_filepath, exist_ok=True) # create missing directories from v11_
↳filepath
shutil.move(v10_filepath, v11_filepath) # move validation result json file
```

The following sections detail the changes you must make to existing store backends.

2a. Validations Store Backends

For validations stores with backends of type `TupleFilesystemStoreBackend`, `TupleS3StoreBackend`, or `TupleGCSStoreBackend`, rename paths (or object keys) of all existing Expectation Suite Validation Result json files:

Before:

```
great_expectations/uncommitted/validations/my_suite_name/my_run_id/some_batch_
↳identifier.json
```

After:

```
great_expectations/uncommitted/validations/my_suite_name/my_run_id/my_run_time/batch_
↳ identifier.json
```

For validations stores with backends of type `DatabaseStoreBackend`, perform the following database migration:

- add string column with name `run_name`; copy values from `run_id` column
- add string column with name `run_time`; fill with appropriate dateutil parsable values
- delete `run_id` column

2b. Evaluation Parameter Store Backends

If you have any configured evaluation parameter stores that use a `DatabaseStoreBackend` backend, you must perform the following migration for each database backend:

- add string column with name `data_asset_name`; fill with appropriate values or use “__none__”
- add string column with name `run_name`; copy values from `run_id` column
- add string column with name `run_time`; fill with appropriate dateutil parsable values
- delete `run_id` column

2c. Data Docs Validations Store Backends

Note: If you are okay with rebuilding your Data Docs sites, you can skip the migration steps in this section. Instead, you should run the following CLI command, but **only after** you have completed the above migration steps: `great_expectations docs clean --all && great_expectations docs build`.

For Data Docs sites with store backends of type `TupleFilesystemStoreBackend`, `TupleS3StoreBackend`, or `TupleGCSSStoreBackend`, rename paths (or object keys) of all existing Expectation Suite Validation Result html files:

Before:

```
great_expectations/uncommitted/data_docs/my_site_name/validations/my_suite_name/my_
↳ run_id/some_batch_identifier.html
```

After:

```
great_expectations/uncommitted/data_docs/my_site_name/validations/my_suite_name/my_
↳ run_id/my_run_time/batch_identifier.html
```

3.8.3 How to upgrade to 0.10.x

In the 0.10.0 release, there are several breaking changes to the `DataContext` API.

Most are related to the clarified naming `BatchKwargsGenerators`.

So, if you are using methods on the data context that used to have an argument named `generators`, you will need to update that code to use the more precise name `batch_kwargs_generators`.

For example, in the method `DataContext.get_available_data_asset_names` the parameter `generator_names` is now `batch_kwargs_generator_names`.

If you are using `BatchKwargsGenerators` in your project config, follow these steps to upgrade your existing Great Expectations project: * Edit your `great_expectations.yml` file and change the key `generators` to `batch_kwargs_generators`.

- Run a simple command such as: `great_expectations datasource list` and ensure you see a list of datasources.

3.8.4 How to upgrade to 0.9.x

In the 0.9.0 release, there are several changes to the DataContext API.

Follow these steps to upgrade your existing Great Expectations project:

- In the terminal navigate to the parent of the `great_expectations` directory of your project.
- Run this command:

```
great_expectations project check-config
```

- For every item that needs to be renamed the command will display a message that looks like this: The class name 'X' has changed to 'Y'. Replace all occurrences of X with Y in your project's `great_expectations.yml` config file.
- After saving the config file, rerun the `check-config` command.
- Depending on your configuration, you will see 3-6 of these messages.
- The command will display this message when done: Your config file appears valid!.
- Rename your Expectation Suites to make them compatible with the new naming. Save this Python code snippet in a file called `update_project.py`, then run it using the command: `python update_project.py PATH_TO_GE_CONFIG_DIRECTORY:`

```
#!/usr/bin/env python3
import sys
import os
import json
import uuid
import shutil

def update_validation_result_name(validation_result):
    data_asset_name = validation_result["meta"].get("data_asset_name")
    if data_asset_name is None:
        print("    No data_asset_name in this validation result. Unable to update it.")
        return
    data_asset_name_parts = data_asset_name.split("/")
    if len(data_asset_name_parts) != 3:
        print("    data_asset_name in this validation result does not appear to be_
normalized. Unable to update it.")
        return
    expectation_suite_suffix = validation_result["meta"].get("expectation_suite_name")
    if expectation_suite_suffix is None:
        print("    No expectation_suite_name found in this validation result. Unable_
to update it.")
        return
    expectation_suite_name = ".".join(
        data_asset_name_parts +
        [expectation_suite_suffix]
    )
    validation_result["meta"]["expectation_suite_name"] = expectation_suite_name
    try:
        del validation_result["meta"]["data_asset_name"]
    except KeyError:
```

(continues on next page)

(continued from previous page)

```

        pass
def update_expectation_suite_name(expectation_suite):
    data_asset_name = expectation_suite.get("data_asset_name")
    if data_asset_name is None:
        print("    No data_asset_name in this expectation suite. Unable to update it.
↪")
        return
    data_asset_name_parts = data_asset_name.split("/")
    if len(data_asset_name_parts) != 3:
        print("    data_asset_name in this expectation suite does not appear to be_
↪normalized. Unable to update it.")
        return
    expectation_suite_suffix = expectation_suite.get("expectation_suite_name")
    if expectation_suite_suffix is None:
        print("    No expectation_suite_name found in this expectation suite. Unable_
↪to update it.")
        return
    expectation_suite_name = ".".join(
        data_asset_name_parts +
        [expectation_suite_suffix]
    )
    expectation_suite["expectation_suite_name"] = expectation_suite_name
    try:
        del expectation_suite["data_asset_name"]
    except KeyError:
        pass
def update_context_dir(context_root_dir):
    # Update expectation suite names in expectation suites
    expectations_dir = os.path.join(context_root_dir, "expectations")
    for subdir, dirs, files in os.walk(expectations_dir):
        for file in files:
            if file.endswith(".json"):
                print("Migrating suite located at: " + str(os.path.join(subdir,
↪file)))
                with open(os.path.join(subdir, file), 'r') as suite_fp:
                    suite = json.load(suite_fp)
                    update_expectation_suite_name(suite)
                with open(os.path.join(subdir, file), 'w') as suite_fp:
                    json.dump(suite, suite_fp)
    # Update expectation suite names in validation results
    validations_dir = os.path.join(context_root_dir, "uncommitted", "validations")
    for subdir, dirs, files in os.walk(validations_dir):
        for file in files:
            if file.endswith(".json"):
                print("Migrating validation_result located at: " + str(os.path.
↪join(subdir, file)))
                try:
                    with open(os.path.join(subdir, file), 'r') as suite_fp:
                        suite = json.load(suite_fp)
                        update_validation_result_name(suite)
                    with open(os.path.join(subdir, file), 'w') as suite_fp:
                        json.dump(suite, suite_fp)
                try:
                    run_id = suite["meta"].get("run_id")
                    es_name = suite["meta"].get("expectation_suite_name").split(".")
↪")
                    filename = "converted__" + str(uuid.uuid1()) + ".json"

```

(continues on next page)

(continued from previous page)

```

        os.makedirs(os.path.join(
            context_root_dir, "uncommitted", "validations",
            *es_name, run_id
        ), exist_ok=True)
        shutil.move(os.path.join(subdir, file),
                    os.path.join(
                        context_root_dir, "uncommitted", "validations
↪",
                        *es_name, run_id, filename
                    )
                )
    except OSError as e:
        print("    Unable to move validation result; file has been_
↪updated to new "
            "format but not moved to new store location.")
    except KeyError:
        pass # error will have been generated above
    except json.decoder.JSONDecodeError:
        print("    Unable to process file: error reading JSON.")
if __name__ == "__main__":
    if len(sys.argv) < 2:
        print("Please provide a path to update.")
        sys.exit(-1)
    path = str(os.path.abspath(sys.argv[1]))
    print("About to update context dir for path: " + path)
    update_context_dir(path)

```

- Rebuild Data Docs:

```
great_expectations docs build
```

- This project has now been migrated to 0.9.0. Please see the list of changes below for more detailed information.

CONFIGURATION CHANGES:

- FixedLengthTupleXXXX stores are renamed to TupleXXXX stores; they no longer allow or require (or allow) a `key_length` to be specified, but they do allow `filepath_prefix` and/or `filepath_suffix` to be configured as an alternative to an the `filepath_template`.
- ExtractAndStoreEvaluationParamsAction is renamed to StoreEvaluationParametersAction; a new StoreMetricsAction is available as well to allow DataContext-configured metrics to be saved.
- The InMemoryEvaluationParameterStore is replaced with the EvaluationParameterStore; EvaluationParameterStore and MetricsStore can both be configured to use DatabaseStoreBackend instead of the InMemoryStoreBackend.
- The `type` key can no longer be used in place of `class_name` in configuration. Use `class_name` instead.
- BatchKwargsGenerators are more explicitly named; we avoid use of the term “Generator” because it is ambiguous. All existing BatchKwargsGenerators have been renamed by substituting “BatchKwargsGenerator” for “Generator”; for example GlobReaderGenerator is now GlobReaderBatchKwargsGenerator.
- ReaderMethod is no longer an enum; it is a string of the actual method to be invoked (e.g. `read_csv` for pandas). That change makes it easy to specify arbitrary reader_methods via `batch_kwargs` (including `read_pickle`), BUT existing configurations using enum-based `reader_method` in `batch_kwargs` will need to update their code. For example, a pandas datasource would use `reader_method: read_csv`` instead of `reader_method: csv`

CODE CHANGES:

- DataAssetName and name normalization have been completely eliminated, which causes several related changes to code using the DataContext.
 - data_asset_name is **no longer** a parameter in the create_expectation_suite, get_expectation_suite, or get_batch commands; expectation suite names exist in an independent namespace.
 - batch_kwarg alone now define the batch to be received, and the datasource name **must** be included in batch_kwarg as the “datasource” key.
 - **A generator name is therefore no longer required to get data or define an expectation suite.**
 - The BatchKwargGenerators API has been simplified; *build_batch_kwarg* should be the entrypoint for all cases of using a generator to get batch_kwarg, including when explicitly specifying a partition, limiting the number of returned rows, accessing saved kwarg, or using any other BatchKwargGenerator feature. BatchKwargGenerators *must* be attached to a specific datasource to be instantiated.
 - This tutorial uses the latest API for validating data: *Validate Data*
- **Database store tables are not compatible** between versions and require a manual migration; the new default table names are: *ge_validations_store*, *ge_expectations_store*, *ge_metrics*, and *ge_evaluation_parameters*. The Validations Store uses a three-part compound primary key consisting of run_id, expectation_suite_name, and batch_identifier; Expectations Store uses the expectation_suite_name as its only key. Both Metrics and Evaluation Parameters stores use run_id, expectation_suite_name, metric_id, and metric_kwarg_id to form a compound primary key.
- The term “batch_fingerprint” is no longer used, and has been replaced with “batch_markers”. It is a dictionary that, like batch_kwarg, can be used to construct an ID.
- *get_data_asset_name* and *save_data_asset_name* are removed.
- There are numerous under-the-scenes changes to the internal types used in GreatExpectations. These should be transparent to users.

3.8.5 How to upgrade to 0.8.x

In the 0.8.0 release, our DataContext config format has changed dramatically to enable new features including extensibility.

Some specific changes:

- New top-level keys:
 - *expectations_store_name*
 - *evaluation_parameter_store_name*
 - *validations_store_name*
- Deprecation of the *type* key for configuring objects (replaced by *class_name* (and *module_name* as well when ambiguous).
- Completely new *SiteBuilder* configuration. See *data_docs_reference*.

BREAKING:

- **top-level `validate` has a new signature**, that offers a variety of different options for specifying the DataAsset class to use during validation, including *data_asset_class_name* / *data_asset_module_name* or *data_asset_class*
- Internal class name changes between alpha versions: - InMemoryEvaluationParameterStore - ValidationsStore - ExpectationsStore - ActionListValidationOperator
- Several modules are now refactored into different names including all datasources

- InMemoryBatchKwargs use the key dataset instead of df to be more explicit

Pre-0.8.x configuration files `great_expectations.yml` are not compatible with 0.8.x. Run `great_expectations project check-config` - it will offer to create a new config file. The new config file will not have any customizations you made, so you will have to copy these from the old file.

If you run into any issues, please ask for help on [Slack](#).

3.8.6 How to upgrade to 0.7.x

In version 0.7, GE introduced several new features, and significantly changed the way `DataContext` objects work:

- A *DataContexts* object manages access to expectation suites and other configuration in addition to data assets. It provides a flexible but opinionated structure for creating and storing configuration and expectations in version control.
- When upgrading from prior versions, the new *Datasources* objects provide the same functionality that compute-environment-specific data context objects provided before, but with significantly more flexibility.
- The term “autoinspect” is no longer used directly, having been replaced by a much more flexible *Profiling* feature.

3.9 Miscellaneous

Warning: This doc is spare parts: leftover pieces of old documentation. It’s potentially helpful, but may be incomplete, incorrect, or confusing.

3.9.1 How to use the Great Expectations command line interface (CLI)

After reading this guide, you will know:

- How to create a Great Expectations project
- How to add new datasources
- How to add and edit expectation suites
- How to build and open Data Docs

The Great Expectations command line is organized using a **<NOUN> <VERB>** syntax. This guide is organized by nouns (datasource, suite, docs) then verbs (new, list, edit, etc).

Basics

There are a few commands that are critical to your everyday usage of Great Expectations. This is a list of the most common commands you’ll use in order of how much you’ll probably use them:

- `great_expectations suite edit`
- `great_expectations suite new`
- `great_expectations suite list`
- `great_expectations suite delete`
- `great_expectations docs build`

- `great_expectations docs clean`
- `great_expectations checkpoint new`
- `great_expectations checkpoint list`
- `great_expectations checkpoint run`
- `great_expectations checkpoint script`
- `great_expectations datasource list`
- `great_expectations datasource profile`
- `great_expectations datasource delete`
- `great_expectations validation-operator run`
- `great_expectations init`

You can get a list of Great Expectations commands available to you by typing `great_expectations --help`. Each noun command and each verb sub-command has a description, and should help you find the thing you need.

Note: All Great Expectations commands have help text. As with most posix utilities, you can try adding `--help` to the end. For example, by running `great_expectations suite new --help` you'll see help output for that specific command.

```
$ great_expectations --help
Usage: great_expectations [OPTIONS] COMMAND [ARGS]...

Welcome to the great_expectations CLI!

Most commands follow this format: great_expectations <NOUN> <VERB>
The nouns are: datasource, docs, project, suite
Most nouns accept the following verbs: new, list, edit

In particular, the CLI supports the following special commands:

- great_expectations init : create a new great_expectations project
- great_expectations datasource profile : profile a datasource
- great_expectations docs build : compile documentation from expectations

Options:
  --version          Show the version and exit.
  -v, --verbose      Set great_expectations to use verbose output.
  --help            Show this message and exit.

Commands:
  datasource  datasource operations
  docs        data docs operations
  init        initialize a new Great Expectations project
  project     project operations
  suite       expectation suite operations
```

great_expectations init

To add Great Expectations to your project run the `great_expectations init` command in your project directory. This will run you through a very short interactive experience to connect to your data, show you some sample expectations, and open Data Docs.

Note: You can install the Great Expectations python package by typing `pip install great_expectations`, if you don't have it already.

```
$ great_expectations init
...
```

After this command has completed, you will have the entire Great Expectations directory structure with all the code you need to get started protecting your pipelines and data.

great_expectations docs

great_expectations docs build

The `great_expectations docs build` command builds your Data Docs site. You'll use this any time you want to view your expectations and validations in a web browser.

```
$ great_expectations docs build
Building Data Docs...
The following Data Docs sites were built:
- local_site:
  file:///Users/dickens/my_pipeline/great_expectations/uncommitted/data_docs/local_
  ↪site/index.html
```

great_expectations docs build --site_name <YOUR_SITE>

By default, `great_expectations docs build` command builds all the Data Docs sites in a project. If you wish to only build once site you may use the `--site_name` argument and pass in the name of the site from your configuration.

```
$ great_expectations docs build --site_name s3_site
Building Data Docs...
The following Data Docs sites were built:
- s3_site: https://s3.amazonaws.com/my-ge-bucket/index.html
```

great_expectations docs clean

The `great_expectations docs clean` command deletes your Data Docs site. Specify `site_name` to delete specific site or `all` to delete all. To rebuild, just use the build command.

```
$ great_expectations docs clean --site-name local_site

$ great_expectations docs clean --all=y
```

great_expectations suite

All command line operations for working with expectation suites are here.

great_expectations suite list

Running `great_expectations suite list` gives a list of available expectation suites in your project:

```
$ great_expectations suite list
3 expectation suites found:
  customer_requests.warning
  customer_requests.critical
  churn_model_input
```

great_expectations suite new

Attention: In the next major release `suite new` command will no longer create a sample suite. Instead, `suite new` will create an empty suite. Additionally the `--empty` flag will be deprecated. The existing behavior of automatic creation of a demo suite is now in the command `suite demo`.

Create a new expectation suite. Just as writing SQL queries is far better with access to data, so are writing expectations. These are best written interactively against some data.

To this end, this command interactively helps you choose some data, creates the new suite, adds sample expectations to it, and opens up Data Docs.

Important: The sample suites generated **are not meant to be production suites** - they are examples only.

Great Expectations will choose a couple of columns and generate expectations about them to demonstrate some examples of assertions you can make about your data.

```
$ great_expectations suite new
Enter the path (relative or absolute) of a data file
: data/npi.csv

Name the new expectation suite [npi.warning]:

Great Expectations will choose a couple of columns and generate expectations about
↳ them
to demonstrate some examples of assertions you can make about your data.

Press Enter to continue
:

Generating example Expectation Suite...
Building Data Docs...
The following Data Docs sites were built:
- local_site:
  file:///Users/dickens/Desktop/great_expectations/uncommitted/data_docs/local_site/
  ↳ index.html
A new Expectation suite 'npi.warning' was added to your project
```

To edit this suite you can click the **How to edit** button in Data Docs, or run the command: `great_expectations suite edit np_i.warning`. This will generate a jupyter notebook and allow you to add, remove or adjust any expectations in the sample suite.

Important: Great Expectations generates working jupyter notebooks when you make new suites and edit existing ones. This saves you tons of time by avoiding all the necessary boilerplate.

Because these notebooks can be generated at any time from the expectation suites (stored as JSON) you should **consider the notebooks to be entirely disposable artifacts**.

They are put in your `great_expectations/uncommitted` directory and you can delete them at any time.

Because they can expose actual data, we strongly suggest leaving them in the `uncommitted` directory to avoid potential data leaks into source control.

```
great_expectations suite new --suite <SUITE_NAME>
```

If you already know the name of the suite you want to create you can skip one of the interactive prompts and specify the suite name directly.

```
$ great_expectations suite new --suite np_i.warning
Enter the path (relative or absolute) of a data file
: data/np_i.csv
... (same as above)
```

```
great_expectations suite new --empty
```

Attention: In the next major release `suite new` command will no longer create a sample suite. Instead, `suite new` will create an empty suite. Therefore the `--empty` flag will be deprecated.

If you prefer to skip the example expectations and start writing expectations in a new empty suite directly in a jupyter notebook, add the `--empty` flag.

```
$ great_expectations suite new --empty
Enter the path (relative or absolute) of a data file
: data/np_i.csv

Name the new expectation suite [np_i.warning]: np_i.warning
A new Expectation suite 'np_i.warning' was added to your project
Because you requested an empty suite, we'll open a notebook for you now to edit it!
If you wish to avoid this you can add the `--no-jupyter` flag.

[I 14:55:15.992 NotebookApp] Serving notebooks from local directory: /Users/dickens/
↳ Desktop/great_expectations/uncommitted
... (jupyter opens)
```

`great_expectations suite delete --suite <SUITE_NAME>`

If you already know the name of the suite you want to delete you can skip one of the interactive prompts and specify the suite name directly.

```
$ great_expectations suite delete --suite np1.warning
Enter the path (relative or absolute) of a data file
: data/np1.csv
... (same as above)
```

`great_expectations suite new --empty --no-jupyter`

If you prefer to disable Great Expectations from automatically opening the generated jupyter notebook, add the `--no-jupyter` flag.

```
$ great_expectations suite new --empty --no-jupyter

Enter the path (relative or absolute) of a data file
: data/np1.csv

Name the new expectation suite [np1.warning]: np1.warning
A new Expectation suite 'np1.warning' was added to your project
To continue editing this suite, run jupyter notebook /Users/taylor/Desktop/great_
↳ expectations/uncommitted/np1.warning.ipynb
```

You can then run jupyter.

`great_expectations suite edit`

Edit an existing expectation suite. Just as writing SQL queries is far better with access to data, so are authoring expectations. These are best authored interactively against some data. This best done in a jupyter notebook.

Note: BatchKwargs define what data to use during editing.

- When suites are created through the CLI, the original batch_kwargs are stored in a piece of metadata called a citation.
 - The edit command uses the most recent batch_kwargs as a way to know what data should be used for the interactive editing experience.
 - It is often desirable to edit the suite on a different chunk of data.
 - To do this you can edit the batch_kwargs in the generated notebook.
-

To this end, this command interactively helps you choose some data, generates a working jupyter notebook, and opens up that notebook in jupyter.

```
$ great_expectations suite edit np1.warning
[I 15:22:18.809 NotebookApp] Serving notebooks from local directory: /Users/dickens/
↳ Desktop/great_expectations/uncommitted
... (jupyter runs)
```


Important: Great Expectations generates working jupyter notebooks when you make new suites and edit existing ones. This saves you tons of time by avoiding all the necessary boilerplate.

Because these notebooks can be generated at any time from the expectation suites (stored as JSON) you should **consider the notebooks to be entirely disposable artifacts**.

They are put in your `great_expectations/uncommitted` directory and you can delete them at any time.

Because they can expose actual data, we strongly suggest leaving them in the `uncommitted` directory to avoid potential data leaks into source control.

```
great_expectations suite edit <SUITE_NAME> --no-jupyter
```

If you prefer to disable Great Expectations from automatically opening the generated jupyter notebook, add the `--no-jupyter` flag.

```
$ great_expectations suite edit np1.warning --no-jupyter
To continue editing this suite, run jupyter notebook /Users/dickens/Desktop/great_
↳ expectations/uncommitted/np1.warning.ipynb
```

You can then run jupyter.

```
great_expectations suite scaffold <SUITE_NAME>
```

To facilitate fast creation of suites this command helps you write boilerplate using simple heuristics. Much like the `suite new` and `suite edit` commands, you will be prompted interactively to choose some data from one of your datasources.

Important: The suites generated here **are not meant to be production suites** - they are scaffolds to build upon.

Great Expectations will choose which expectations **might make sense** for a column based on the type and cardinality of the data in each selected column.

You will definitely want to edit the suite to hone it after scaffolding.

To create a new suite called “`np1_distribution`” in a project that has a single files-based `PandasDatasource`:

```
$ great_expectations suite scaffold np1_distribution
Heads up! This feature is Experimental. It may change. Please give us your feedback!

Enter the path (relative or absolute) of a data file
: np1.csv
...jupyter opens
```

You’ll then see jupyter open a scaffolding notebook. Run the first cell in the notebook that loads the data. You don’t need to worry about what’s happening there.

The next code cell in the notebook presents you with a list of all the columns found in your selected data. To select which columns you want to scaffold expectations on, simply uncomment them to include them.

Run the next few code cells to see the scaffolded suite in Data Docs.

You may keep the scaffold notebook open and iterate on the included and excluded columns and expectations to get closer to the kind of suite you want.

Important: Great Expectations generates working jupyter notebooks. This saves you tons of time by avoiding all the necessary boilerplate.

Because these notebooks can be generated at any time from the expectation suites (stored as JSON) you should **consider the notebooks to be entirely disposable artifacts**.

They are put in your `great_expectations/uncommitted` directory and you can delete them at any time.

Because the scaffolder is not very smart, you will want to edit this suite to tune the parameters and make any adjustments such as removing expectations that don't make sense for your use case.

great_expectations suite scaffold <SUITE_NAME> --no-jupyter

If you wish to skip opening the scaffolding notebook in jupyter you can use this optional flag.

The notebook will be created in your `great_expectations/uncommitted` directory.

```
suite scaffold np_i_distributions --no-jupyter
Heads up! This feature is Experimental. It may change. Please give us your feedback!

Enter the path (relative or absolute) of a data file
: np_i.csv
To continue scaffolding this suite, run `jupyter notebook uncommitted/scaffold_np_i_
↳distributions.ipynb`
```

great_expectations suite demo

Create a sample expectation suite. Just as writing SQL queries is far better with access to data, so are writing expectations. These are best written interactively against some data.

To this end, this command interactively helps you choose some data, creates the new suite, adds sample expectations to it, and opens up Data Docs.

Important: The sample suites generated **are not meant to be production suites** - they are examples only.

Great Expectations will choose a couple of columns and generate expectations about them to demonstrate some examples of assertions you can make about your data.

```
$ great_expectations suite demo
Enter the path (relative or absolute) of a data file
: data/np_i.csv

Name the new expectation suite [np_i.warning]:

Great Expectations will choose a couple of columns and generate expectations about_
↳them
to demonstrate some examples of assertions you can make about your data.

Press Enter to continue
:

Generating example Expectation Suite...
```

(continues on next page)

(continued from previous page)

```
Building Data Docs...
The following Data Docs sites were built:
- local_site:
  file:///Users/dickens/Desktop/great_expectations/uncommitted/data_docs/local_site/
  → index.html
A new Expectation suite 'npi.warning' was added to your project
```

To edit this suite you can click the **How to edit** button in Data Docs, or run the command: `great_expectations suite edit npi.warning`. This will generate a jupyter notebook and allow you to add, remove or adjust any expectations in the sample suite.

Important: Great Expectations generates working jupyter notebooks when you make new suites and edit existing ones. This saves you tons of time by avoiding all the necessary boilerplate.

Because these notebooks can be generated at any time from the expectation suites (stored as JSON) you should **consider the notebooks to be entirely disposable artifacts**.

They are put in your `great_expectations/uncommitted` directory and you can delete them at any time.

Because they can expose actual data, we strongly suggest leaving them in the `uncommitted` directory to avoid potential data leaks into source control.

great_expectations validation-operator

All command line operations for working with *validation operators* are here.

great_expectations validation-operator list

Running `great_expectations validation-operator list` gives a list of validation operators configured in your project:

```
$ great_expectations validation-operator list
... (YOUR VALIDATION OPERATORS)
```

great_expectations validation-operator run

There are two modes to run this command:

1. Interactive (good for development):

Specify the name of the validation operator using the `--name` argument and the name of the expectation suite using the `--suite` argument.

The cli will help you specify the batch of data that you want to validate interactively.

If you want to call a validation operator to validate one batch of data against one expectation suite, you can invoke this command:

```
great_expectations validation-operator run --name <VALIDATION_OPERATOR_NAME>
--suite <SUITE_NAME>
```

Use the `--name` argument to specify the name of the validation operator you want to run. This has to be the name of one of the validation operators configured in your project. You can list the names by calling the `great_expectations validation-operator list` command or by examining the `validation_operators` section in your project's `great_expectations.yml` config file.

Use the `--suite` argument to specify the name of the expectation suite you want the validation operator to validate the batch of data against. This has to be the name of one of the expectation suites that exist in your project. You can look up the names by calling the `suite list` command.

The command will help you specify the batch of data that you want the validation operator to validate interactively.

```
$ great_expectations validation-operator --name action_list_operator --suite np1.
↪warning

Let us help you specify the batch of data you want the validation operator to_
↪validate.

Enter the path (relative or absolute) of a data file
: data/np1_small.csv
Validation Succeeded!
```

2. Non-interactive (good for production):

If you want run a validation operator non-interactively, use the `--validation_config_file` argument to specify the path of the validation configuration JSON file.

```
great_expectations validation-operator run ----validation_config_file
<VALIDATION_CONFIG_FILE_PATH>
```

This file can be used to instruct a validation operator to validate multiple batches of data and use multiple expectation suites to validate each batch.

Note: A validation operator can validate multiple batches of data and use multiple expectation suites to validate each batch. For example, you might want to validate 3 source files, with 2 tiers of suites each, one for a warning suite and one for a critical stop-the-presses hard failure suite.

This command exits with 0 if the validation operator ran and the “success” attribute in its return object is True. Otherwise, the command exits with 1.

Tip: This is an excellent way to use call Great Expectations from within your pipeline if your pipeline code can run shell commands.

A validation config file specifies the name of the validation operator in your project and the list of batches of data that you want the operator to validate. Each batch is defined using `batch_kwargs`. The `expectation_suite_names` attribute for each batch specifies the list of names of expectation suites that the validation operator should use to validate the batch.

Here is an example validation config file:

```
{
  "validation_operator_name": "action_list_operator",
  "batches": [
    {
      "batch_kwargs": {
```

(continues on next page)

(continued from previous page)

```

    "path": "/Users/me/projects/my_project/data/data.csv",
    "datasource": "my_filesystem_datasource",
    "reader_method": "read_csv"
  },
  "expectation_suite_names": ["suite_one", "suite_two"]
},
{
  "batch_kwargs": {
    "query": "SELECT * FROM users WHERE status = 1",
    "datasource": "my_redshift_datasource"
  },
  "expectation_suite_names": ["suite_three"]
}
]
}

```

```

$ great_expectations validation-operator run --validation_config_file my_val_config.
↪ json
Validation Succeeded!

```

great_expectations datasource

All command line operations for working with *datasources* are here. A datasource is a connection to data and a processing engine. Examples of a datasource are: - csv files processed in pandas or Spark - a relational database such as Postgres, Redshift or BigQuery

great_expectations datasource list

This command displays a list of your datasources and their types. These can be found in your `great_expectations/great_expectations.yml` config file.

```

$ great_expectations datasource list
[{'name': 'files_datasource', 'class_name': 'PandasDatasource'}]

```

great_expectations datasource new

This interactive command helps you connect to your data.

```

$ great_expectations datasource list
What data would you like Great Expectations to connect to?
1. Files on a filesystem (for processing with Pandas or Spark)
2. Relational database (SQL)
: 1

What are you processing your files with?
1. Pandas
2. PySpark
: 1

Enter the path (relative or absolute) of the root directory where the data files are
↪ stored.

```

(continues on next page)

(continued from previous page)

```
: data

Give your new data source a short name.
[data__dir]: np_i_drops
A new datasource 'np_i_drops' was added to your project.
```

If you are using a database you will be guided through a series of prompts that collects and verifies connection details and credentials.

`great_expectations datasource delete <datasource_name>`

This command deletes specified datasources.

```
$ great_expectations datasource delete files_datasource
```

`great_expectations datasource profile`

For details on profiling, see this [reference document](#)

Caution: Profiling is a beta feature and is not guaranteed to be stable. YMMV

`great_expectations checkpoint`

A checkpoint is a bundle of one or more batches of data with one or more Expectation Suites. A checkpoint can be as simple as one batch of data paired with one Expectation Suite. A checkpoint can be as complex as many batches of data across different datasources paired with one or more Expectation Suites each.

Tip: Checkpoints are an ideal way to embed Great Expectations into your pipeline or use Great Expectations adjacent to your pipeline. If you have shell access in your pipeline you can use the `checkpoint run` command. If you do not have shell access or prefer a python script you can use the `checkpoint script` command to generate a python file to run a checkpoint.

`great_expectations checkpoint new <CHECKPOINT> <SUITE>`

Interactively configure a new checkpoint.

A checkpoint is stored in a yaml file in the `great_expectations/checkpoints/` directory in your project. After creation, a checkpoint can be run with the `great_expectations checkpoint run` command.

Similar to other commands, this command interactively helps you choose some data for your checkpoint.

If your project has a suite named `np_i.warning` and you wish to create a checkpoint called `source_tables` in your project you run this as follows:

```
$ great_expectations checkpoint new source_tables np_i.warning
...(Interactively choose some data)
A checkpoint named `source_tables` was added to your project!
```

(continues on next page)

(continued from previous page)

- To edit this checkpoint edit the checkpoint file: `great_expectations/checkpoints/source_tables.yml`
- To run this checkpoint run ``great_expectations checkpoint run source_tables``

The new checkpoint file (`great_expectations/checkpoints/source_tables.yml`) will look like this:

```
# This checkpoint was created by the command `great_expectations checkpoint new`.
#
# It can be run with the `great_expectations checkpoint run` command.
# You can edit this file to add batches of data and expectation suites.
#
# For more details please see
# https://docs.greatexpectations.io/en/latest/command_line.html#great-expectations-
# checkpoint-new-checkpoint-suite
validation_operator_name: action_list_operator
# Batches are a list of batch_kwarg paired with a list of one or more suite
# names. A checkpoint can have one or more batches. This makes deploying
# Great Expectations in your pipelines easy!
batches:
  - batch_kwarg:
      path: /Users/me/pipeline/source_files/mpi.csv
      datasource: files_datasource
      reader_method: read_csv
      expectation_suite_names: # one or more suites may validate against a single batch
        - mpi.warning
```

You can edit this file to add batches of data and expectation suites across your project. To find out more see this guide: *How to add validations, data, or suites to a Checkpoint*

If you are using a SQL datasource you will be guided through a series of prompts that helps you choose a table or write a SQL query.

Tip: A custom SQL query can be very handy if for example you wanted to validate all records in a table with timestamps.

For example, imagine you want to protect a machine learning model that looks at insurance claims from last 90 days to predict costs with a checkpoint named `cost_model_protection`. If you have built a suite called `cost_model_assumptions` and have a postgres datasource with a `claims` table with an `claim_timestamp` column and you wanted to validate claims that occurred in the last 90 days you might do something like:

```
$ great_expectations checkpoint new cost_model_protection cost_model_assumptions
Heads up! This feature is Experimental. It may change. Please give us your feedback!

Which table would you like to use? (Choose one)
1. claims (table)
Do not see the table in the list above? Just type the SQL query
: SELECT * FROM claims WHERE claim_timestamp > now() - interval '90 day';
A checkpoint named `cost_model_protection` was added to your project!
  - To edit this checkpoint edit the checkpoint file: great_expectations/checkpoints/
  ↳ cost_model_protection.yml
  - To run this checkpoint run `great_expectations checkpoint run cost_model_
  ↳ protection`
```

This checkpoint can then be run nightly before your model makes cost predictions!

great_expectations checkpoint list

List the checkpoints found in the project.

```
$ great_expectations checkpoint list
Found 1 checkpoint.
- my_checkpoint
```

great_expectations checkpoint run <CHECKPOINT>

Run an existing checkpoint.

Tip: This is an ideal way to embed Great Expectations into your pipeline or use it adjacent to your pipeline if you have shell access in your pipeline. If you do not have shell access or prefer a python script you can use the `checkpoint script` command to generate a python file to run a checkpoint.

This command will return posix status codes and print messages as follows:

Situation	Return code	Message
all validations passed	0	Validation Succeeded!
one or more validation failed	1	Validation Failed!

If there is a checkpoint named `source_tables` in your project you can run it as follows:

```
$ great_expectations checkpoint run source_tables
Validation Succeeded!
```

great_expectations checkpoint script <CHECKPOINT>

Create an executable script for deploying a checkpoint via python.

This is provided as a convenience for teams whose pipeline deployments may not have shell access or may wish to have more flexibility when running a checkpoint.

To generate a script, you must first have an existing checkpoint. To make a new checkpoint use `great_expectations checkpoint new`.

```
$ great_expectations checkpoint script cost_model_protection
Heads up! This feature is Experimental. It may change. Please give us your feedback!
A python script was created that runs the checkpoint named: `cost_model_protection`
- The script is located in `great_expectations/uncommitted/run_cost_model_
↪protection.py`
- The script can be run with `python great_expectations/uncommitted/run_cost_model_
↪protection.py`
```

The generated script looks like this:

```
"""
This is a basic generated Great Expectations script that runs a checkpoint.

A checkpoint is a list of one or more batches paired with one or more
```

(continues on next page)

(continued from previous page)

Expectation Suites and a configurable Validation Operator.

Checkpoints can be run directly without this script using the ``great_expectations checkpoint run`` command. This script is provided for those who wish to run checkpoints via python.

Data that is validated is controlled by BatchKwargs, which can be adjusted in the checkpoint file: `great_expectations/checkpoints/cost_model_protection.yml`.

Data are validated by use of the ``ActionListValidationOperator`` which is configured by default. The default configuration of this Validation Operator saves validation results to your results store and then updates Data Docs.

This makes viewing validation results easy for you and your team.

Usage:

- Run this file: ``python great_expectations/uncommitted/run_cost_model_protection.py``.
- This can be run manually or via a scheduler such as cron.
- If your pipeline runner supports python snippets you can paste this into your pipeline.

```
"""
import sys

from great_expectations import DataContext

# checkpoint configuration
context = DataContext("/Users/taylor/Desktop/demo/great_expectations")
checkpoint = context.get_checkpoint("cost_model_protection")

# load batches of data
batches_to_validate = []
for batch in checkpoint["batches"]:
    batch_kwargs = batch["batch_kwargs"]
    for suite_name in batch["expectation_suite_names"]:
        suite = context.get_expectation_suite(suite_name)
        batch = context.get_batch(batch_kwargs, suite)
        batches_to_validate.append(batch)

# run the validation operator
results = context.run_validation_operator(
    checkpoint["validation_operator_name"],
    assets_to_validate=batches_to_validate,
)

# take action based on results
if not results["success"]:
    print("Validation Failed!")
    sys.exit(1)

print("Validation Succeeded!")
sys.exit(0)
```

This script can be run by invoking it with:

```
$ python great_expectations/uncommitted/run_cost_model_protection.py
Validation Succeeded!
```

Just like the built in command `great_expectations checkpoint run`, this posix-compatible script exits with a status of 0 if validation is successful and a status of 1 if validation failed.

A failure will look like:

```
$ python great_expectations/uncommitted/run_cost_model_protection.py
Validation Failed!
```

Shell autocompletion for the CLI

If you want to enable autocompletion for the Great Expectations CLI, you can execute following commands in your shell (or add them to your `.bashrc/.zshrc` or `~/.config/fish/completions/`):

```
$ eval "$(_GREAT_EXPECTATIONS_COMPLETE=source_bash great_expectations) "
```

for bash

```
$ eval "$(_GREAT_EXPECTATIONS_COMPLETE=source_zsh great_expectations) "
```

for zsh, and

```
$ eval (env _GREAT_EXPECTATIONS_COMPLETE=source_fish great_expectations)
```

for fish (you'll have to create a `~/.config/fish/completions/great_expectations.fish` file).

Alternatively, if you don't want the `eval` command to slow down your shell startup time, you can instead add the commands as a script to your shell profile. For more info, see the official [Click documentation](#).

Miscellaneous

- `great_expectations project check-config` checks your `great_expectations/great_expectations.yml` for validity. This is handy for occasional Great Expectations version migrations.

Acknowledgements

This article was heavily inspired by the phenomenal Rails Command Line Guide https://guides.rubyonrails.org/command_line.html.

3.9.2 How to add comments to a page on docs.greatexpectations.io

Many of the pages in Great Expectations' documentation allow comments through our discussion forum. Here's how to enable comments, specifically on *How-to guides*.

Prerequisites: This how-to guide assumes you have already:

- *Written a new How-to Guide*
-

Steps

1. Create and publish an article on the [Great Expectations discussion forum](#).
 - Please use the title of your How-to Guide as the title of the article (Ex: “How to configure an ActionListValidationOperator”).
 - Please copy and paste this text into the body of the article, and replace the link with your guide’s link.

```
This article is for comments to: https://docs.greatexpectations.io/en/
↳latest/how_to_guides/{some_path}/{your_guide_name}.html

Please comment +1 if this How-to Guide is important to you.
```

2. After publishing the article, find the `topic_id` at the end of the article’s URL: (Ex: 219 in `/t/how-to-configure-an-actionlistvalidationoperator/219`). Please add this code to the bottom of your Guide, and replace `{topic_id}` with the real id.

Additional Notes

The `sphinxcontrib-discourse` plugin used for this integration requires a verified domain, in this case `docs.greatexpectations.io`. During local development and PR review, you will see something like this:

Comments

Error Embedding



Referer:

The referer was either not sent, or did not match any of the following hosts:

- `docs.greatexpectations.io`

Don’t be alarmed. This is normal. Your comment box will appear as soon as the page goes live on `docs.greatexpectations.io`.

[Start Discussion](#)

0 replies

After at least one comment has been added, it will look like this.

1 reply


A

abegong

13h

+1 Love this idea. 🙌

Continue Discussion



Additional Resources

- <https://github.com/pdavide/sphinxcontrib-discourse>

Comments

Warning: This doc is a stub.

3.9.3 TEMPLATE: How to {stub}

Admonition from Mr. Dickens

“Whether I shall turn out to be the hero of my own life, or whether that station will be held by anybody else, these pages must show.”

This guide is a stub. We all know that it will be useful, but no one has made time to write it yet.

If it would be useful to you, please comment with a +1 and feel free to add any suggestions or questions below.

If you want to be a real hero, we’d welcome a pull request. Please see *the Contributing tutorial* and *How to write a how to guide* to get started.

Attention: This doc is a template. Please do not attempt to follow these instructions.

3.9.4 TEMPLATE: How to {do something}

This guide will help you {do something.} {That something is important or useful, because of some reason.}

Prerequisites: This how-to guide assumes you have already:

- *Set up a working deployment of Great Expectations*
- You have adopted a puppy

Steps

1. First, do something bashy.

```
something --foo bar
```

2. Next, do something with yaml.

```
site_section_builders:
  expectations: # Look, comments work, too!
    class_name: DefaultSiteSectionBuilder
    source_store_name: expectations_store
    renderer:
      module_name: great_expectations.render.renderer
      class_name: ExpectationSuitePageRenderer
```

3. Next, try a python snippet or two.

```
batch.expect_table_row_count_to_be_between(min_value=1000, max_value=4000)
```

Here's a pinch of connecting text.

```
batch.expect_table_row_count_to_be_between(min_value=2000, max_value=5000)
```

Additional notes

How-to guides are not about teaching or explanation. They are about providing clear, bite-sized replication steps. If you **must** include a longer explanation, it should go in this section.

Additional resources

- [Links in RST](#) are a pain.

Comments

3.9.5 How to write a how to guide

This guide shows how to create a new how-to guide in Great Expectations. By writing guides with consistent structure and styling, you can get your PRs approved faster and make the Great Expectations docs discoverable, useful, and maintainable.

Steps

1. Copy the *How-to guide template file* to the appropriate subdirectory of docs/how_to_guides/, and rename it.
2. Write a title and purpose paragraph.
3. Decide whether you're writing a *code-heavy* or *process-heavy* guide, and *adjust your formatting* appropriately.
4. Fill out the Prerequisites info box (see *How-to guide template file*). The header of the info box says: "This how-to guide assumes you have already:". Place each prerequisite under its own bullet and phrase it using the style in the template: "done something".

5. Fill in the Steps section, making sure to include bash, yml, and code snippets as appropriate.
 - Hint: it's often most efficient to run through the technical steps in a notebook or at the command line before starting to write. Then you can simply copy-paste content into code blocks, and write headers and text to connect them.
 - After you finish writing, you should definitely follow your own steps from start to finish at least once, to make sure there aren't any gaps.
6. If needed, add content to Additional Notes and/or Additional Resources. These sections supplement the article with information that would be distracting to include in Steps. It's fine for them to be empty.
7. Enable comments for your How-to guide, by following [these instructions](#).
8. Scan your article to make sure it follows the [Style Guide](#). If you're not familiar with the Style Guide, that's okay: your PR reviewer will also check for style and let you know if we find any issues.
9. Submit your PR!

Additional Notes

Purpose of a how-to guide

The purpose of a how-to guide is to *replicate*, NOT to *teach or explain*. Teaching and explaining Great Expectations concepts are covered in the [Core concepts](#) reference section.

- Assume that the user has already successfully run `great_expectations init` and has a working deployment of Great Expectations.
- Assume that the user is already familiar with core concepts in Great Expectations: Expectations, Data Contexts, Validation, Datasources, etc. etc. You don't need to spend any time explaining these things.
- If you're integrating with another system, assume that the user is familiar with that system. ie. If you're writing the "How to configure a Snowflake Datasource," don't spend any words explaining Snowflake or any of its core concepts.
- If there are important, non-obvious principles for how Great Expectations relates to other systems, you may include them in the guide. If they are short (1-2 sentences, max), they can go in the Steps section. Otherwise, please put them in Additional Notes.

Remember, the goal is to help users successfully replicate specific steps as simply as possible. Surprisingly often, it turns out to best to not include explanation at all, since it can distract from the main purpose of the guide. If you feel you must include it, shorter is better.

Structure of a how-to guide

With rare exceptions, How-to guides follow this structure:

1. Title
2. Purpose
3. Steps
4. Additional Notes (optional)
5. Additional Resources (optional)

Title: "How to X". See the [Style Guide](#) for specific guidance on how to phrase and format titles.

Purpose paragraph: A single, short paragraph to state the purpose of the guide, and motivate it if necessary.

“This guide will help you publish an Data Docs site directly to S3. Publishing a site this way makes reviewing and acting on Validation Results easy in a team, and provides a central location to review Expectations.”

Sometimes motivation can be a simple statement of purpose:

“This guide will help you connect to a MongoDB Datasource.”

If the user has data in Mongo and wants to configure a Datasource, no additional justification is needed.

Steps: Steps describe the golden path steps for successful replication.

- Most steps will include `inline code`, such as a bash command, or an example yml snippet or two.
- Snippets should be as short as possible, but no shorter. In general, you can think of the snippet like a diff: what needs to change to accomplish this step?
- Steps should be linear. “Do A, then B, then C.” Avoid complex loops and/or branching. If loops or branching are needed, it is likely a sign that the scope of the guide is too big. In that case, consider options for splitting it into more than one how-to guide.

Additional notes: This section covers errata that would be distracting to include in Steps. It’s fine for it to be empty.

Additional resources: Additional resources, usually external (i.e. not within the Great Expectations documentation) and usually shown as a list. To avoid link rot, please use this section sparingly, and prefer links to stable, well-maintained resources.

Code-heavy vs process-heavy guides

Broadly speaking, there are two kinds of How-to Guides: code-heavy and process-heavy. All guides are about following a specific sequence of steps. In code-heavy guides, most or all of the steps are expressed in technical syntax: code snippets, JSON or YAML objects, CLI commands, etc. In process-heavy guides, many of the steps are things that must be done manually.

Most guides are code-heavy. When writing a guide that could go either way, please prefer code-heavy, since they tend to make for better replication. (This guide happens to be process-heavy, because it’s about writing.)

Indentation, bolding, and code blocks

For code-heavy guides

- Treat the first sentence of each step like a header.
 - Use short, complete, imperative sentences: (“Paste the YAML snippet into your config file”, “Run `great_expectations init`”)
 - Header text should be **bold**.
 - Avoid [links](#) or `inline code` in headers, since RST files do not support nesting them within bolded text. If your header must include text that would normally be a link or inline code, please repeat it in the body text, and use a link or code block there.
- Indent content within steps.
- Any time the user needs to do something, it should be in a code block.
 - Please follow this convention even if the text in the code block is somewhat redundant against the text of the step.
 - Clear, sequential code blocks are easy for the eye to follow. They encourage a health copy-and-modify development pattern.

- All of these styles are modeled in the *How-to guide template file*. If you use that template as your guide, you'll be off to a very good start.

For process-heavy guides

- Do not separate headers or bold first sentences.
- Avoid big blocks of text without visual cues for how to read it. Indentation and sub-bullets are your friends.
- When including a code block, please follow the same conventions as for code-heavy guides.
- All of these styles are modeled in the this .rst file.

Additional Resources

- [Links in RST](#) are a pain.

3.10 How-to guides: Spare parts

These guides are leftover pieces of old documentation. They're potentially helpful, but may be incomplete, incorrect, or confusing.

Once each file is tidied up and rehabilitated, it can be moved to another section. Some of these article may turn into more than one guides elsewhere.

Warning: These docs are leftover pieces of old documentation. They're potentially helpful, but may be incomplete, incorrect, or confusing.

Warning: This doc is spare parts: leftover pieces of old documentation. It's potentially helpful, but may be incomplete, incorrect, or confusing.

3.10.1 Data Context Reference

A *DataContexts* manages assets for a project.

Configuration

The DataContext configuration file (`great_expectations.yml`) provides fine-grained control over several core features available to the DataContext to assist in managing resources used by Great Expectations. Key configuration areas include specifying Datasources, Data Docs, Validation Operators, and Stores used to manage access to resources such as expectation suites, validation results, profiling results, evaluation parameters and plugins.

This file is intended to be committed to your repo.

Datasources

Datasources tell Great Expectations where your data lives and how to get it.

Using the *CLI* command `great_expectations datasource new` is the easiest way to add a new datasource.

The *datasources* section declares which *Datasources* objects should be available in the DataContext. Each datasource definition should include the *class_name* of the datasource, generators, and any other relevant configuration information. For example, the following simple configuration supports a Pandas-based pipeline:

```
datasources:
  pipeline:
    class_name: PandasDatasource
    generators:
      default:
        class_name: InMemoryGenerator
```

The following configuration demonstrates a more complicated configuration for reading assets from S3 into pandas. It will access the amazon public NYC taxi data and provides access to two assets: 'taxi-green' and 'taxi-fhv' which represent two public datasets available from the resource.

```
datasources:
  nyc_taxi:
    class_name: PandasDatasource
    generators:
      s3:
        class_name: S3GlobReaderBatchKwargsGenerator
        bucket: nyc-tlc
        delimiter: '/'
        reader_options:
          sep: ','
          engine: python
        assets:
          taxi-green:
            prefix: trip data/
            regex_filter: 'trip data/green.*\.csv'
          taxi-fhv:
            prefix: trip data/
            regex_filter: 'trip data/fhv.*\.csv'
    data_asset_type:
      class_name: PandasDataset
```

Here is an example for a SQL based pipeline:

```
datasources:
  edw:
    class_name: SqlAlchemyDatasource
    credentials: ${data_warehouse}
    data_asset_type:
      class_name: SqlAlchemyDataset
    generators:
      default:
        class_name: TableBatchKwargsGenerator
```

Note the `credentials` key references a corresponding key in the `config_variables.yml` file which is not in source control that would look like this:

```
data_warehouse:
  drivename: postgres
  host: warehouse.ourcompany.biz
  port: '5432'
  username: bob
  password: 1234
  database: prod
```

Note that the `datasources` section *includes* all defined generators as well as specifying their names. See [Using custom expectations with a Datasource](#) for more information about configuring datasources to use custom expectations.

Data Asset Names

Data asset names consist of three parts, a `datasource`, `generator`, and `generator asset`. `DataContext` functions will attempt to “normalize” a `data_asset_name` if they are provided with only a string, by splitting on the delimiter character (by default `/`) and then attempting to identify an unambiguous name. `DataContext` searches through names that already have expectation suites first, then considers names provided by generators.

For example:

```
# Returns a normalized name with string representation my_datasource/my_generator/my_
↪asset if
# my_datasource and my_generator uniquely provide an asset called my_asset
context.normalize_data_asset_name("my_asset")
```

Data Docs

The [Data Docs](#) section defines how individual sites should be built and deployed. See the detailed documentation for more information.

Stores

A `DataContext` requires three stores to function properly: an `expectations_store`, `validations_store`, and `evaluation_parameter_store`. Consequently a minimal store configuration for a `DataContext` would include the following:

```
expectations_store_name: expectations_store
validations_store_name: validations_store
evaluation_parameter_store_name: evaluation_parameter_store

stores:
  expectations_store:
    class_name: ExpectationsStore
    store_backend:
      class_name: TupleFilesystemStoreBackend
      base_directory: expectations/
  validations_store:
    class_name: ValidationsStore
    store_backend:
      class_name: TupleFilesystemStoreBackend
      base_directory: uncommitted/validations/
  evaluation_parameter_store:
    class_name: EvaluationParameterStore
```

The *expectations_store* provides access to *expectations_suite* objects, using the *DataContext*'s namespace; the *validations_store* does the same for validations. See *Evaluation Parameters* for more information on the evaluation parameters store.

Stores can be referenced in other objects in the *DataContext*. They provide a common API for accessing data independently of the backend where it is stored. For example, on a team that uses S3 to store expectation suites and validation results, updating the configuration to use cloud storage requires only changing the store *class_name* and providing the bucket/prefix combination:

```
expectations_store_name: expectations_store
validations_store_name: validations_store
evaluation_parameter_store_name: evaluation_parameter_store

stores:
  expectations_store:
    class_name: ExpectationsStore
    store_backend:
      class_name: TupleS3StoreBackend
      base_directory: expectations/
      bucket: ge.my_org.com
      prefix:
    validations_store:
      class_name: ValidationsStore
      store_backend:
        class_name: TupleS3StoreBackend
        bucket: ge.my_org.com
        prefix: common_validations
    evaluation_parameter_store:
      class_name: EvaluationParameterStore
```

GE uses *boto3* to access AWS, so credentials simply need to be available in any standard place searched by that library. You may also specify keyword arguments for *boto3* to use in the *boto3_options* key of the *store_backend* configuration.

Validation Operators

See the *validation_operators* for more information regarding configuring and using validation operators.

Managing Environment and Secrets

Values can be injected in a *DataContext* configuration file by using the `${var}` syntax to specify a variable to be substituted.

The injected value can come from three sources:

1. A config variables file
2. Environment variables
3. A dictionary passed to the *DataContext* constructor.

Each source above will override variables set in a previous source.

Config Variables File

DataContext accepts a parameter called `config_variables_file_path` which can include a file path from which variables to substitute should be read. The file needs to define top-level keys which are available to substitute into a DataContext configuration file. Keys from the config variables file can be defined to represent complex types such as a dictionary or list, which is often useful for configuring database access.

Variable substitution enables: 1) keeping secrets out of source control & 2) environment-based configuration changes such as staging vs prod.

When GE encounters substitution syntax (like `my_key: ${my_value}` or `my_key: $my_value`) in the `great_expectations.yml` config file it will attempt to replace the value of `my_key` with the value from an environment variable `my_value` or a corresponding key read from the file specified using `config_variables_file_path`, which is located in `uncommitted/config_variables.yml` by default. This is an example of a `config_variables.yml` file:

```
prod_credentials:
  type: postgresql
  host: secure_server
  port: 5432
  username: username
  password: sensitive_password
  database: ge

dev_credentials:
  type: postgresql
  host: localhost
  port: 5432
  username: dev
  password: dev
  database: ge
```

If the substitution value comes from the config variables file, it can be a simple (non-nested) value or a nested value such as a dictionary. If it comes from an environment variable, it must be a simple value.

Environment Variable Substitution

Environment variables will be substituted into a DataContext config with higher priority than values from the config variables file.

Note: Substitution of environment variables is currently only supported in the `great_expectations.yml`, but not in a config variables file. See [\[this Discuss post\]\(https://discuss.greatexpectations.io/t/environment-variable-substitution-is-not-working-for-me-when-connecting-ge-to-my-database/72\)](https://discuss.greatexpectations.io/t/environment-variable-substitution-is-not-working-for-me-when-connecting-ge-to-my-database/72) for an example.

Passing Values To DataContext

A dictionary of values can be passed to a DataContext when it is instantiated. These values will override both values from the config variables file and from environment variables.

```
data_context = DataContext.create(runtime_environment={
    'name_to_replace': 'replacement_value'
})
```

Default Out of Box Config File

Should you need a clean config file you can run `great_expectation init` in a new directory or use this template:

```
# Welcome to Great Expectations! Always know what to expect from your data.
#
# Here you can define datasources, batch kwargs generators, integrations and
# more. This file is intended to be committed to your repo. For help with
# configuration please:
#   - Read our docs: https://docs.greatexpectations.io/en/latest/reference/data\_
↪context\_reference.html#configuration
#   - Join our slack channel: http://greatexpectations.io/slack

config_version: 2

# Datasources tell Great Expectations where your data lives and how to get it.
# You can use the CLI command `great_expectations datasource new` to help you
# add a new datasource. Read more at https://docs.greatexpectations.io/en/latest/
↪features/datasource.html
datasources: {}
  edw:
    class_name: SqlAlchemyDatasource
    credentials: ${edw}
    data_asset_type:
      class_name: SqlAlchemyDataset
    generators:
      default:
        class_name: TableBatchKwargsGenerator

# This config file supports variable substitution which enables: 1) keeping
# secrets out of source control & 2) environment-based configuration changes
# such as staging vs prod.
#
# When GE encounters substitution syntax (like `my_key: ${my_value}` or
# `my_key: $my_value`) in the config file it will attempt to replace the value
# of `my_key` with the value from an environment variable `my_value` or a
# corresponding key read from the file specified using
# `config_variables_file_path`. Environment variables take precedence.
#
# If the substitution value comes from the config variables file, it can be a
# simple (non-nested) value or a nested value such as a dictionary. If it comes
# from an environment variable, it must be a simple value. Read more at:
# https://docs.greatexpectations.io/en/latest/reference/data\_context\_reference.html
↪#managing-environment-and-secrets
config_variables_file_path: uncommitted/config_variables.yml

# The plugins_directory will be added to your python path for custom modules
# used to override and extend Great Expectations.
plugins_directory: plugins/

# Validation Operators are customizable workflows that bundle the validation of
# one or more expectation suites and subsequent actions. The example below
# stores validations and send a slack notification. To read more about
# customizing and extending these, read: https://docs.greatexpectations.io/en/latest/
↪features/validation\_operators\_and\_actions.html
validation_operators:
  action_list_operator:
```

(continues on next page)

(continued from previous page)

```

# To learn how to configure sending Slack notifications during evaluation
# (and other customizations), read: https://docs.greatexpectations.io/en/latest/
→reference/validation_operators/perform_action_list_validation_operator.html
class_name: ActionListValidationOperator
action_list:
  - name: store_validation_result
    action:
      class_name: StoreValidationResultAction
  - name: store_evaluation_params
    action:
      class_name: StoreEvaluationParametersAction
  - name: update_data_docs
    action:
      class_name: UpdateDataDocsAction
  - name: send_slack_notification_on_validation_result
    action:
      class_name: SlackNotificationAction
      slack_webhook: ${validation_notification_slack_webhook}
      notify_on: all
      renderer:
        module_name: great_expectations.render.renderers.slack_renderer
        class_name: SlackRenderer

stores:
# Stores are configurable places to store things like Expectations, Validations
# Data Docs, and more. These are for advanced users only - most users can simply
# leave this section alone.
#
# Three stores are required: expectations, validations, and
# evaluation_parameters, and must exist with a valid store entry. Additional
# stores can be configured for uses such as data_docs, validation_operators, etc.
expectations_store:
  class_name: ExpectationsStore
  store_backend:
    class_name: TupleFilesystemStoreBackend
    base_directory: expectations/
validations_store:
  class_name: ValidationsStore
  store_backend:
    class_name: TupleFilesystemStoreBackend
    base_directory: uncommitted/validations/
evaluation_parameter_store:
  # Evaluation Parameters enable dynamic expectations. Read more here:
  # https://docs.greatexpectations.io/en/latest/reference/evaluation\_parameters.html
  class_name: EvaluationParameterStore
expectations_store_name: expectations_store
validations_store_name: validations_store
evaluation_parameter_store_name: evaluation_parameter_store

data_docs_sites:
# Data Docs make it simple to visualize data quality in your project. These
# include Expectations, Validations & Profiles. The are built for all
# Datasources from JSON artifacts in the local repo including validations &
# profiles from the uncommitted directory. Read more at https://docs.greatexpectations.io/en/latest/features/data\_docs.html
→greatexpectations.io/en/latest/features/data_docs.html
local_site:
  class_name: SiteBuilder
  store_backend:

```

(continues on next page)

(continued from previous page)

```
class_name: TupleFilesystemStoreBackend
base_directory: uncommitted/data_docs/local_site/
```

3.10.2 Usage Statistics

To help us improve the tool, by default we track event data when certain Data Context-enabled commands are run. The usage statistics include things like the OS and python version, and which GE features are used. You can see the exact schemas for all of our messages [here](#).

While we hope you'll leave them on, you can easily disable usage statistics for a Data Context by adding the following to your data context configuration:

```
anonymous_usage_statistics:
  data_context_id: <randomly-generated-uuid>
  enabled: false
```

You can also disable usage statistics system-wide by setting the `GE_USAGE_STATS` environment variable to `FALSE` or adding the following code block to a file called `great_expectations.conf` located in `/etc/` or `~/`.

```
[anonymous_usage_statistics]
enabled=FALSE
```

As always, please reach out [on Slack](#) if you have any questions or comments.

Warning: This doc is spare parts: leftover pieces of old documentation. It's potentially helpful, but may be incomplete, incorrect, or confusing.

3.10.3 Data Docs Reference

Data Docs make it simple to visualize data quality in your project. These include Expectations, Validations & Profiles. They are built for all Datasources from JSON artifacts in the local repo including validations & profiles from the uncommitted directory.

Users have full control over configuring Data Documentation for their project - they can modify the pre-configured site (or remove it altogether) and add new sites with a configuration that meets the project's needs. The easiest way to add a new site to the configuration is to copy the "local_site" configuration block in `great_expectations.yml`, give the copy a new name and modify the details as needed.

Data Docs Site Configuration

The default Data Docs site configuration looks like this:

```
data_docs_sites:
  local_site:
    class_name: SiteBuilder
    store_backend:
      class_name: TupleFilesystemStoreBackend
      base_directory: uncommitted/data_docs/local_site/
    site_index_builder:
      class_name: DefaultSiteIndexBuilder
```

Here is an example of a site configuration from `great_expectations.yml` with defaults defined explicitly:

```
data_docs_sites:
  local_site: # site name
    module_name: great_expectations.render.renderers.site_builder
    class_name: SiteBuilder
    store_backend:
      class_name: TupleFilesystemStoreBackend
      base_directory: uncommitted/data_docs/local_site/
    site_index_builder:
      class_name: DefaultSiteIndexBuilder
    site_section_builders:
      expectations: # if empty, or one of ['0', 'None', 'False', 'false', 'FALSE',
↳ 'none', 'NONE'], section not rendered
        class_name: DefaultSiteSectionBuilder
        source_store_name: expectations_store
        renderer:
          module_name: great_expectations.render.renderers
          class_name: ExpectationSuitePageRenderer

      validations: # if empty, or one of ['0', 'None', 'False', 'false', 'FALSE',
↳ 'none', 'NONE'], section not rendered
        class_name: DefaultSiteSectionBuilder
        source_store_name: validations_store
        run_name_filter:
          ne: profiling # exclude validations with run_name "profiling" - reserved_
↳ for profiling results
        renderer:
          module_name: great_expectations.render.renderers
          class_name: ValidationResultsPageRenderer

      profiling: # if empty, or one of ['0', 'None', 'False', 'false', 'FALSE', 'none
↳ ', 'NONE'], section not rendered
        class_name: DefaultSiteSectionBuilder
        source_store_name: validations_store
        run_name_filter:
          eq: profiling # include ONLY validations with run_name "profiling" - _
↳ reserved for profiling results
        renderer:
          module_name: great_expectations.render.renderers
          class_name: ProfilingResultsPageRenderer
```

`validations_store` in the example above specifies the name of a store configured in the *stores* section. Validation and profiling results from that store will be included in the documentation. The optional `run_name_filter` attribute allows to include (eq for exact match) or exclude (ne) validation results with a particular run name.

Limiting Validation Results

If you would like to limit rendered Validation Results to the *n* most-recent, you may do so by setting the *validation_results_limit* key in your Data Docs configuration:

```
data_docs_sites:
  local_site:
    class_name: SiteBuilder
    show_how_to_buttons: true
    store_backend:
```

(continues on next page)

(continued from previous page)

```

class_name: TupleFilesystemStoreBackend
base_directory: uncommitted/data_docs/local_site/
site_index_builder:
  class_name: DefaultSiteIndexBuilder
  validation_results_limit: 5

```

Automatically Publishing Data Docs

It is possible to directly publish (continuously updating!) data docs sites to a shared location such as a static site hosted in S3 by simply updating the store_backend configuration in the site configuration. If we modify the configuration in the example above to adjust the store backend to an S3 bucket of our choosing, our *SiteBuilder* will automatically save the resulting site to that bucket.

```

store_backend:
  class_name: TupleS3StoreBackend
  bucket: data-docs.my_org.org
  prefix:

```

See the tutorial on *publishing data docs to S3* for more information.

More advanced configuration

It is possible to extend renderers and views and customize the particular class used to render any of the objects in your documentation. In this more advanced configuration, a “CustomTableContentBlockRenderer” is used only for the validations renderer, and no profiling results are rendered at all.

```

data_docs_sites:
  # Data Docs make it simple to visualize data quality in your project. These
  # include Expectations, Validations & Profiles. They are built for all
  # Datasources from JSON artifacts in the local repo including validations &
  # profiles from the uncommitted directory. Read more at
  # https://docs.greatexpectations.io/en/latest/features/data_docs.html
  local_site:
    class_name: SiteBuilder
    store_backend:
      class_name: TupleFilesystemStoreBackend
      base_directory: uncommitted/data_docs/local_site/
    site_section_builders:
      validations: # if empty, or one of ['0', 'None', 'False', 'false', 'FALSE',
→ 'none', 'NONE'], section not rendered
      class_name: DefaultSiteSectionBuilder
      source_store_name: validations_store
      run_name_filter:
        ne: profiling
      renderer:
        module_name: great_expectations.render.renderer
        class_name: ValidationResultsPageRenderer
      column_section_renderer:
        class_name: ValidationResultsColumnSectionRenderer
      table_renderer:
        module_name: custom_renderers.custom_table_content_block
        class_name: CustomTableContentBlockRenderer

```

(continues on next page)

(continued from previous page)

```
profiling: # if empty, or one of ['0', 'None', 'False', 'false', 'FALSE', 'none', 'NONE'], section not rendered
```

To support that custom renderer, we need to ensure the implementation is available in our `plugins/` directory. Note that we can use a subdirectory and standard python submodule notation, but that we need to include an `__init__.py` file in our `custom_renderers` package.

```
plugins/
├── custom_renderers
│   ├── __init__.py
│   └── custom_table_content_block.py
└── additional_ge_plugin.py
```

Building Data Docs

Using the CLI

The *Great Expectations CLI* can build comprehensive Data Docs from expectation suites available to the configured context and validations available in the `great_expectations/uncommitted` directory.

```
great_expectations docs build
```

When called without additional arguments, this command will render all the Data Docs sites specified in `great_expectations.yml` configuration file into HTML and open them in a web browser.

The command will print out the locations of `index.html` file for each site.

To disable the web browser opening behavior use `--no-view` option.

To render just one site, use `--site-name SITE_NAME` option.

Here is when the `docs build` command should be called:

- when you want to fully rebuild a Data Docs site
- after a new expectation suite is added or an existing one is edited
- after new data is profiled (only if you declined the prompt to build data docs when running the profiling command)

When a new validation result is generated after running a Validation Operator, the Data Docs sites will add this result automatically if the operator has the `UpdateDataDocsAction` action configured (read actions).

Using the raw API

The underlying python API for rendering documentation is still new and evolving. Use the following snippet as a guide for how to profile a single batch of data and build documentation from the `validation_result`.

```
import os
import great_expectations as ge

from great_expectations.profile.basic_dataset_profiler import BasicDatasetProfiler
from great_expectations.render.renderer import ProfilingResultsPageRenderer,
↳ ExpectationSuitePageRenderer
```

(continues on next page)

(continued from previous page)

```

from great_expectations.render.view import DefaultJinjaPageView

profiling_html_filepath = '/path/into/which/to/save/results.html'

# obtain the DataContext object
context = ge.data_context.DataContext()

# load a batch to profile
context.create_expectation_suite('default')
batch = context.get_batch(
    batch_kwargs=context.build_batch_kwargs("my_datasource", "my_batch_kwargs_generator", "my_asset")
    expectation_suite_name='default',
)

# run the profiler on the batch - this returns an expectation suite and validation_
↳ results for this suite
expectation_suite, validation_result = BasicDatasetProfiler().profile(batch)

# use a renderer to produce a document model from the validation results
document_model = ProfilingResultsPageRenderer().render(validation_result)

# use a view to render the document model (produced by the renderer) into a HTML_
↳ document
os.makedirs(os.path.dirname(profiling_html_filepath), exist_ok=True)
with open(profiling_html_filepath, 'w') as writer:
    writer.write(DefaultJinjaPageView().render(document_model))

```

Customizing Data Docs

Introduction

Data Docs uses the [Jinja](#) template engine to generate HTML pages. The built-in Jinja templates used to compile Data Docs pages are implemented in the `great_expectations.render.view` module and are tied to *View* classes. Views determine how page content is displayed. The content data that Views specify and consume is generated by *Renderer* classes and are implemented in the `great_expectations.render.renderer` module. Renderers take Great Expectations objects as input and return typed dictionaries - Views take these dictionaries as input and output rendered HTML.

Built-In Views and Renderers

Out of the box, Data Docs supports two top-level Views (i.e. pages), `great_expectations.render.view.DefaultJinjaIndexPageView`, for a site index page, and `great_expectations.render.view.DefaultJinjaPageView` for all other pages. Pages are broken into sections - `great_expectations.render.view.DefaultJinjaSectionView` - which are composed of UI components - `great_expectations.render.view.DefaultJinjaComponentView`. Each of these Views references a single base Jinja template, which can incorporate any number of other templates through inheritance.

Data Docs comes with the following built-in site page Renderers:

- `great_expectations.render.renderer.SiteIndexPageRenderer` (index page)
- `great_expectations.render.renderer.ProfilingResultsPageRenderer` (Profiling Results pages)

- `great_expectations.render.renderer.ExpectationSuitePageRenderer` (Expectation Suite pages)
- `great_expectations.render.renderer.ValidationResultsPageRenderer` (Validation Results pages)

Analogous to the base View templates referenced above, these Renderers can be thought of as base Renderers for the primary Data Docs pages, and may call on many other ancillary Renderers.

It is possible to extend Renderers and Views and customize the particular class used to render any of the objects in your documentation.

Other Tools

In addition to Jinja, Data Docs draws on the following libraries to compile HTML documentation, which you can use to further customize HTML documentation:

- Bootstrap
- Font Awesome
- jQuery
- Altair

Making Minor Adjustments

Many of the HTML elements in the default Data Docs pages have pre-configured classes that you may use to make minor adjustments using your own custom CSS. By default, when you run `great_expectations init`, Great Expectations creates a scaffold within the `plugins` directory for customizing Data Docs. Within this scaffold is a file called `data_docs_custom_styles.css` - this CSS file contains all the pre-configured classes you may use to customize the look and feel of the default Data Docs pages. All custom CSS, applied to these pre-configured classes or any other HTML elements, must be placed in this file.

Scaffolded directory tree:

```
plugins
├── custom_data_docs
│   ├── renderers
│   ├── styles
│   │   └── data_docs_custom_styles.css
│   └── views
```

Using Custom Views and Renderers

Suppose you start a new Great Expectations project by running `great_expectations init` and compile your first Data Docs site. After looking over the local site, you decide you want to implement the following changes:

1. A completely new Expectation Suite page, requiring a new View and Renderer
2. A smaller modification to the default Validation page, swapping out a child renderer for a custom version
3. Remove Profiling Results pages from the documentation

To make these changes, you must first implement the custom View and Renderers and ensure they are available in the `plugins` directory specified in your project configuration (`plugins/` by default). Note that you can use a sub-directory and standard python submodule notation, but you must include an `__init__.py` file in your custom package.

By default, when you run `great_expectations init`, Great Expectations creates placeholder directories for your custom views, renderers, and CSS stylesheets within the `plugins` directory. If you wish, you may save your custom views and renderers in an alternate location however, any CSS stylesheets must be saved to `plugins/custom_data_docs/styles`.

Scaffolded directory tree:

```
plugins
├── custom_data_docs
│   ├── renderers
│   ├── styles
│   │   └── data_docs_custom_styles.css
│   └── views
```

When you are done with your implementations, your `plugins/` directory has the following structure:

```
plugins
├── custom_data_docs
│   ├── renderers
│   │   ├── __init__.py
│   │   ├── custom_table_renderer.py
│   │   └── custom_expectation_suite_page_renderer.py
│   ├── styles
│   │   └── data_docs_custom_styles.css
│   └── views
│       ├── __init__.py
│       └── custom_expectation_suite_view.py
```

For Data Docs to use your custom Views and Renderers when compiling your local Data Docs site, you must specify where to use them in the `data_docs_sites` section of your project configuration.

Before modifying your project configuration, the relevant section looks like this:

```
data_docs_sites:
  local_site:
    class_name: SiteBuilder
    store_backend:
      class_name: TupleFilesystemStoreBackend
      base_directory: uncommitted/data_docs/local_site/
    site_index_builder:
      class_name: DefaultSiteIndexBuilder
```

This is what it looks like after your changes are added:

```
data_docs_sites:
  local_site:
    class_name: SiteBuilder
    store_backend:
      class_name: TupleFilesystemStoreBackend
      base_directory: uncommitted/data_docs/local_site/
    site_index_builder:
      class_name: DefaultSiteIndexBuilder
    site_section_builders:
      expectations:
        renderer:
          module_name: custom_data_docs.renderers.custom_expectation_suite_page_
↪renderer
          class_name: CustomExpectationSuitePageRenderer
```

(continues on next page)

(continued from previous page)

```

view:
    module_name: custom_data_docs.views.custom_expectation_suite_view
    class_name: CustomExpectationSuiteView
validations:
    renderer:
        module_name: great_expectations.render.renderer
        class_name: ValidationResultsPageRenderer
        column_section_renderer:
            class_name: ValidationResultsColumnSectionRenderer
        table_renderer:
            module_name: custom_data_docs.renderers.custom_table_renderer
            class_name: CustomTableRenderer
profiling:

```

By providing an empty profiling key within `site_section_builders`, your third goal is achieved and Data Docs will no longer render Profiling Results pages. The same can be achieved by setting the profiling key to any of the following values: ['0', 'None', 'False', 'false', 'FALSE', 'none', 'NONE'].

Lastly, to compile your newly-customized Data Docs local site, you run `great_expectations docs build` from the command line.

`site_section_builders` defaults:

```

site_section_builders:
    expectations: # if empty, or one of ['0', 'None', 'False', 'false', 'FALSE', 'none',
↳ 'NONE'], section not rendered
        class_name: DefaultSiteSectionBuilder
        source_store_name: expectations_store
    renderer:
        module_name: great_expectations.render.renderer
        class_name: ExpectationSuitePageRenderer

    validations: # if empty, or one of ['0', 'None', 'False', 'false', 'FALSE', 'none',
↳ 'NONE'], section not rendered
        class_name: DefaultSiteSectionBuilder
        source_store_name: validations_store
        run_name_filter:
            ne: profiling # exclude validations with run_name "profiling" - reserved for_
↳ profiling results
        renderer:
            module_name: great_expectations.render.renderer
            class_name: ValidationResultsPageRenderer

    profiling: # if empty, or one of ['0', 'None', 'False', 'false', 'FALSE', 'none',
↳ 'NONE'], section not rendered
        class_name: DefaultSiteSectionBuilder
        source_store_name: validations_store
        run_name_filter:
            eq: profiling # include ONLY validations with run_name "profiling" - reserved_
↳ for profiling results
        renderer:
            module_name: great_expectations.render.renderer
            class_name: ProfilingResultsPageRenderer

```

Re-Purposing Built-In Views

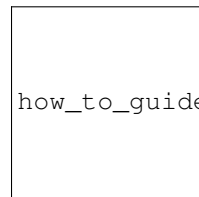
If you would like to re-purpose a built-in View, you may do so by implementing a custom renderer that outputs an appropriately typed and structured dictionary for that View.

Built-in Views and corresponding input types:

- `great_expectations.render.view.DefaultJinjaPageView:` `great_expectations.render.types.RenderedDocumentContent`
- `great_expectations.render.view.DefaultJinjaSectionView:` `great_expectations.render.types.RenderedSectionContent`
- `great_expectations.render.view.DefaultJinjaComponentView:` `great_expectations.render.types.RenderedComponentContent`

An example of a custom page Renderer, using all built-in UI elements is provided below.

Custom Page Renderer Example



how_to_guides/spare_parts/./images/customizing_data_docs.png

Dependencies

- Font Awesome 5.10.1
- Bootstrap 4.3.1
- jQuery 3.2.1
- altair 3.1.0
- Vega 5.3.5
- Vega-Lite 3.2.1
- Vega-Embed 4.0.0

Data Docs is implemented in the `great_expectations.render` module.

Warning: This doc is spare parts: leftover pieces of old documentation. It's potentially helpful, but may be incomplete, incorrect, or confusing.

3.10.4 Profiling Reference

Profiling produces a special kind of *Data Docs* that are purely descriptive.

Expectations and Profiling

In order to characterize a data asset, Profiling uses an Expectation Suite. Unlike the Expectations that are typically used for data validation, these expectations do not necessarily apply any constraints; they can simply identify statistics or other data characteristics that should be evaluated and made available in GE. For example, when the `BasicDatasetProfiler` encounters a numeric column, it will add an `expect_column_mean_to_be_between` expectation but choose the `min_value` and `max_value` to both be `None`: essentially only saying that it expects a mean to exist.

```
{
  "expectation_type": "expect_column_mean_to_be_between",
  "kwargs": {
    "column": "rating",
    "min_value": null,
    "max_value": null
  }
}
```

To “profile” a datasource, therefore, the `BasicDatasetProfiler` included in GE will generate a large number of very loosely-specified expectations. Effectively it is asserting that the given statistic is relevant for evaluating batches of that data asset, but it is not yet sure what the statistic’s value should be.

In addition to creating an expectation suite, profiling data tests the suite against data. The `validation_result` contains the output of that expectation suite when validated against the same batch of data. For a loosely specified expectation like in our example above, getting the observed value was the sole purpose of the expectation.

```
{
  "success": true,
  "result": {
    "observed_value": 4.05,
    "element_count": 10000,
    "missing_count": 0,
    "missing_percent": 0
  }
}
```

Running a profiler on a data asset can also be useful to produce a large number of expectations to review and potentially transfer to a new expectation suite used for validation in a pipeline.

How to Run Profiling

Run During Init

The `great_expectations init` command will auto-generate an example Expectation Suite using a very basic profiler that quickly glances at 1,000 rows of your data. This is not a production suite - it is only meant to show examples of Expectations, many of which may not be meaningful.

Expectation Suites generated by the profiler will be saved in the configured `expectations` directory for Expectation Suites. The Expectation Suite name by default is the name of the profiler that generated it. Validation results will be saved in the `uncommitted/validations` directory by default. When profiling is complete, Great Expectations will build and launch Data Docs based on your data.

Run From Command Line

The GE command-line interface can profile a datasource:

```
great_expectations datasource profile DATASOURCE_NAME
```

Expectation Suites generated by the profiler will be saved in the configured `expectations` directory for Expectation Suites. The Expectation Suite name by default is the name of the profiler that generated it. Validation results will be saved in the `uncommitted/validations` directory by default. When profiling is complete, Great Expectations will build and launch Data Docs based on your data.

See [Data Docs](#) for more information.

Run From Jupyter Notebook

If you want to profile just one data asset in a datasource (e.g., one table in the database), you can do it using Python in a Jupyter notebook:

```
from great_expectations.profile.basic_dataset_profiler import BasicDatasetProfiler

# obtain the DataContext object
context = ge.data_context.DataContext()

# load a batch from the data asset
batch = context.get_batch('ratings')

# run the profiler on the batch - this returns an expectation suite and validation_
↳ results for this suite
expectation_suite, validation_result = BasicDatasetProfiler.profile(batch)

# save the resulting expectation suite with a custom name
context.save_expectation_suite(expectation_suite, "ratings", "my_profiled_expectations
↳ ")
```

Custom Profilers

Like most things in Great Expectations, Profilers are designed to be extensible. You can develop your own profiler by subclassing `DatasetProfiler`, or from the parent `DataAssetProfiler` class itself. For help, advice, and ideas on developing custom profilers, please get in touch on [the Great Expectations slack channel](#).

Profiling Limitations

Inferring Data Types

When profiling CSV files, the profiler makes assumptions, such as considering the first line to be the header. Overriding these assumptions is currently possible only when running profiling in Python by passing extra arguments to `get_batch`.

Data Samples

Since profiling and expectations are so tightly linked, getting samples of *expected* data requires a slightly different approach than the normal path for profiling. Stay tuned for more in this area!

Warning: This doc is spare parts: leftover pieces of old documentation. It's potentially helpful, but may be incomplete, incorrect, or confusing.

3.10.5 How to validate tables in a database using cron

Deploying automated testing adjacent to a data pipeline

You might find yourself in a situation where you do not have the engineering resources, skills, desire, or permissions to embed Great Expectations into your pipeline. As long as your data is accessible you can still reap the benefits of automated data testing.

Note: This is a fast and convenient way to get the benefits of automated data testing without requiring engineering efforts to build Great Expectations into your pipelines.

A tap is an executable python file that runs validates a batch of data against an expectation suite. Taps are a convenient way to generate a data validation script that can be run manually or via a scheduler.

Let's make a new tap using the `tap new` command.

To do this we'll specify the name of the suite and the name of the new python file we want to create. For this example, let's say we want to validate a batch of data against the `movieratings.ratings` expectation suite, and we want to make a new file called `movieratings.ratings_tap.py`

```
$ great_expectations tap new movieratings.ratings movieratings.ratings_tap.py
This is a BETA feature which may change.

Which table would you like to use? (Choose one)
  1. ratings (table)
  Don't see the table in the list above? Just type the SQL query
: 1
A new tap has been generated!
To run this tap, run: python movieratings.ratings_tap.py
You can edit this script or place this code snippet in your pipeline.
```

If you open the generated tap file you'll see it's only a few lines of code to get validations running! It will look like this:

```
"""
A basic generated Great Expectations tap that validates a single batch of data.

Data that is validated is controlled by BatchKwargs, which can be adjusted in
this script.

Data are validated by use of the `ActionListValidationOperator` which is
configured by default. The default configuration of this Validation Operator
saves validation results to your results store and then updates Data Docs.
```

(continues on next page)

(continued from previous page)

This makes viewing validation results easy for you and your team.

Usage:

- Run this file: ``python movieratings.ratings_tap.py``.
- This can be run manually or via a scheduler such as cron.
- If your pipeline runner supports python snippets you can paste this into your pipeline.

```
"""
import sys
import great_expectations as ge

# tap configuration
context = ge.DataContext()
suite = context.get_expectation_suite("movieratings.ratings_tap")
# You can modify your BatchKwargs to select different data
batch_kwargs = {
    "table": "ratings",
    "schema": "movieratings",
    "datasource": "movieratings",
}

# tap validation process
batch = context.get_batch(batch_kwargs, suite)
results = context.run_validation_operator("action_list_operator", [batch])

if not results["success"]:
    print("Validation Failed!")
    sys.exit(1)

print("Validation Succeeded!")
sys.exit(0)
```

To run this and validate a batch of data, run:

```
$ python movieratings.ratings_tap.py
Validation Succeeded!
```

This can easily be run manually anytime you want to check your data. It can also easily be run on a schedule basis with a scheduler such as cron.

You'll want to view the detailed data quality reports in Data Docs by running `great_expectations docs build`.

For example, if you wanted to run this script nightly at 04:00, you'd add something like this to your crontab.

```
$ crontab -e
0 4 * * * /full/path/to/python /full/path/to/movieratings.ratings_tap.py
```

If you don't have access to a scheduler, you can always make checking your data part of your daily routine. Once you experience how much time and pain this saves you, we recommend getting engineering resources to embed Great Expectations validations into your pipeline.

Warning: This doc is spare parts: leftover pieces of old documentation. It's potentially helpful, but may be incomplete, incorrect, or confusing.

3.10.6 How to validate data in a pipeline with a Checkpoint

Note: This is an ideal way to deploy automated data testing if you want to take automated interventions based on the results of data validation. For example, you may want your pipeline to quarantine data that does not meet your expectations.

A data engineer can add a *Validation Operator* to your pipeline and configure it. These *Validation Operators* evaluate the new batches of data that flow through your pipeline against the expectations your team defined in the previous sections.

While data pipelines can be implemented with various technologies, at their core they are all DAGs (directed acyclic graphs) of computations and transformations over data.

This drawing shows an example of a node in a pipeline that loads data from a CSV file into a database table.

- Two expectation suites are deployed to monitor data quality in this pipeline.
- The first suite validates the pipeline's input - the CSV file - before the pipeline executes.
- The second suite validates the pipeline's output - the data loaded into the table.



To implement this validation logic, a data engineer inserts a Python code snippet into the pipeline - before and after the node. The code snippet prepares the data for the GE Validation Operator and calls the operator to perform the validation.

The exact mechanism of deploying this code snippet depends on the technology used for the pipeline.

If Airflow drives the pipeline, the engineer adds a new node in the Airflow DAG. This node will run a `PythonOperator` that executes this snippet. If the data is invalid, the Airflow `PythonOperator` will raise an error which will stop the rest of the execution.

If the pipeline uses something other than Airflow for orchestration, as long as it is possible to add a Python code snippet before and/or after a node, this will work.

Below is an example of this code snippet, with comments that explain what each line does.

```
# Data Context is a GE object that represents your project.
# Your project's great_expectations.yml contains all the config
# options for the project's GE Data Context.
context = ge.data_context.DataContext()

datasource_name = "my_production_postgres" # a datasource configured in your great_
↳ expectations.yml

# Tell GE how to fetch the batch of data that should be validated...

# ... from the result set of a SQL query:
batch_kwargs = {"query": "your SQL query", "datasource": datasource_name}

# ... or from a database table:
# batch_kwargs = {"table": "name of your db table", "datasource": datasource_name}
```

(continues on next page)

(continued from previous page)

```
# ... or from a file:
# batch_kwargs = {"path": "path to your data file", "datasource": datasource_name}

# ... or from a Pandas or PySpark DataFrame
# batch_kwargs = {"dataset": "your Pandas or PySpark DataFrame", "datasource": ↵
↵datasource_name}

# Get the batch of data you want to validate.
# Specify the name of the expectation suite that holds the expectations.
expectation_suite_name = "movieratings.ratings" # this is an example of
                                                    # a suite that you created
batch = context.get_batch(batch_kwargs, expectation_suite_name)

# Call a validation operator to validate the batch.
# The operator will evaluate the data against the expectations
# and perform a list of actions, such as saving the validation
# result, updating Data Docs, and firing a notification (e.g., Slack).
results = context.run_validation_operator(
    "action_list_operator",
    assets_to_validate=[batch],
    run_id=run_id) # e.g., Airflow run id or some run identifier that your pipeline ↵
↵uses.

if not results["success"]:
    # Decide what your pipeline should do in case the data does not
    # meet your expectations.
```


REFERENCE

These pages provide a high-level overview of key great expectations features, with an emphasis on explaining the thinking behind the tools.

4.1 Glossary of Expectations

This is a list of all built-in Expectations. Expectations are extendable so you can create custom expectations for your data domain! To do so see this article: [custom_expectations_reference](#).

4.1.1 Dataset

Dataset objects model tabular data and include expectations with row and column semantics. Many Dataset expectations are implemented using `column_map_expectation` and `column_aggregate_expectation` decorators.

Not all expectations are currently implemented for each backend. Please see *Table of Expectation Implementations By Backend*.

Table shape

- `expect_column_to_exist`
- `expect_table_columns_to_match_ordered_list`
- `expect_table_row_count_to_be_between`
- `expect_table_row_count_to_equal`

Missing values, unique values, and types

- `expect_column_values_to_be_unique`
- `expect_column_values_to_not_be_null`
- `expect_column_values_to_be_null`
- `expect_column_values_to_be_of_type`
- `expect_column_values_to_be_in_type_list`

Sets and ranges

- `expect_column_values_to_be_in_set`
- `expect_column_values_to_not_be_in_set`
- `expect_column_values_to_be_between`
- `expect_column_values_to_be_increasing`
- `expect_column_values_to_be_decreasing`

String matching

- `expect_column_value_lengths_to_be_between`
- `expect_column_value_lengths_to_equal`
- `expect_column_values_to_match_regex`
- `expect_column_values_to_not_match_regex`
- `expect_column_values_to_match_regex_list`
- `expect_column_values_to_not_match_regex_list`

Datetime and JSON parsing

- `expect_column_values_to_match_strftime_format`
- `expect_column_values_to_be_dateutil_parseable`
- `expect_column_values_to_be_json_parseable`
- `expect_column_values_to_match_json_schema`

Aggregate functions

- `expect_column_distinct_values_to_be_in_set`
- `expect_column_distinct_values_to_contain_set`
- `expect_column_distinct_values_to_equal_set`
- `expect_column_mean_to_be_between`
- `expect_column_median_to_be_between`
- `expect_column_quantile_values_to_be_between`
- `expect_column_stddev_to_be_between`
- `expect_column_unique_value_count_to_be_between`
- `expect_column_proportion_of_unique_values_to_be_between`
- `expect_column_most_common_value_to_be_in_set`
- `expect_column_max_to_be_between`
- `expect_column_min_to_be_between`
- `expect_column_sum_to_be_between`

Multi-column

- `expect_column_pair_values_A_to_be_greater_than_B`
- `expect_column_pair_values_to_be_equal`
- `expect_column_pair_values_to_be_in_set`
- `expect_multicolumn_values_to_be_unique`

Distributional functions

- `expect_column_kl_divergence_to_be_less_than`
- `expect_column_bootstrapped_ks_test_p_value_to_be_greater_than`
- `expect_column_chisquare_test_p_value_to_be_greater_than`
- `expect_column_parameterized_distribution_ks_test_p_value_to_be_greater_than`

4.1.2 FileDataAsset

File data assets reason at the file level, and the line level (for text data).

- `expect_file_line_regex_match_count_to_be_between`
- `expect_file_line_regex_match_count_to_equal`
- `expect_file_hash_to_equal`
- `expect_file_size_to_be_between`
- `expect_file_to_exist`
- `expect_file_to_have_valid_table_header`
- `expect_file_to_be_valid_json`

4.2 Core concepts

4.2.1 Expectations

Expectations are assertions for data. They help accelerate data engineering and increase analytic integrity, by making it possible to answer a critical question: *what can I expect of my data?*

Expectations are declarative statements that a computer can evaluate, and that are semantically meaningful to humans, like `expect_column_values_to_be_unique` or `expect_column_mean_to_be_between`.

Expectation Configurations describe specific Expectations for data. They combine an Expectation and specific parameters to make it possible to evaluate whether the expectation is true for some specific data. For example, they might provide expected values for a column or the name of a column whose values should be unique.

Expectation Implementations provide the critical translation layer between what we expect and how to verify the expectation in data or express it in *Data Docs*. Expectation Implementations are tailored for specific validation engines where the actual expectation is executed.

Expectation Suites combine multiple Expectation Configurations into an overall description of a dataset. Expectation Suites should have names corresponding to the kind of data they define, like “NPI” for National Provider Identifier data or “company.users” for a users table.

Expectations

Expectations are the workhorse abstraction in Great Expectations. Like assertions in traditional python unit tests, Expectations provide a flexible, declarative language for describing expected behavior. Unlike traditional unit tests, Great Expectations applies Expectations to data instead of code.

Great Expectations' built-in library includes more than 50 common Expectations, such as:

- `expect_column_values_to_not_be_null`
- `expect_column_values_to_match_regex`
- `expect_column_values_to_be_unique`
- `expect_column_values_to_match_strftime_format`
- `expect_table_row_count_to_be_between`
- `expect_column_median_to_be_between`

For a full list of available Expectations, please check out the [Glossary of Expectations](#). Please note that not all Expectations are implemented on all Execution engines yet. You can see the grid of supported Expectations here. We welcome [contributions](#) to fill in the gaps.

You can also extend Great Expectations by creating your own custom Expectations.

Expectations *enhance communication* about your data and *amplify quality* in data applications. Using expectations helps reduce trips to domain experts and avoids leaving insights about data on the “cutting room floor.”

Attention: Not all Expectations are implemented on all execution engines yet. You can see the grid of supported Expectations here. We welcome [contributions](#) to fill in the gaps.

Expectation Suites

Expectation Suites combine multiple expectations into an overall description of a dataset. For example, a team can group all the expectations about its `rating` table in the movie ratings database from our previous example into an Expectation Suite and call it `movieratings.ratings`. Note these names are completely flexible and the only constraint on the name of a suite is that it must be unique to a given project.

Each Expectation Suite is saved as a JSON file in the `great_expectations/expectations` subdirectory of the Data Context. Users check these files into the version control each time they are updated, same way they treat their source files. This discipline allows data quality to be an integral part of versioned pipeline releases.

The lifecycle of an Expectation Suite starts with creating it. Then it goes through an iterative loop of Review and Edit as the team's understanding of the data described by the suite evolves.

Methods for creating and editing Expectations

Generating expectations is one of the most important parts of using Great Expectations effectively, and there are a variety of methods for generating and encoding expectations. When expectations are encoded in the GE format, they become shareable and persistent sources of truth about how data was expected to behave-and how it actually did.

There are several paths to generating expectations:

1. Automated inspection of datasets. Currently, the profiler mechanism in GE produces expectation suites that can be used for validation. In some cases, the goal is [Profiling](#) your data, and in other cases automated inspection can produce expectations that will be used in validating future batches of data.

2. Expertise. Rich experience from Subject Matter Experts, Analysts, and data owners is often a critical source of expectations. Interviewing experts and encoding their tacit knowledge of common distributions, values, or failure conditions can be an excellent way to generate expectations.
3. Exploratory Analysis. Using GE in an exploratory analysis workflow (e.g. within Jupyter notebooks) is an important way to develop experience with both raw and derived datasets and generate useful and testable expectations about characteristics that may be important for the data's eventual purpose, whether reporting or feeding another downstream model or data system.

Custom expectations

Expectations are especially useful when they capture critical aspects of data understanding that analysts and practitioners know based on its *semantic* meaning. It's common to want to extend Great Expectations with application- or domain-specific Expectations. For example:

```
expect_column_text_to_be_in_english
expect_column_value_to_be_valid_icd_code
```

These Expectations aren't included in the default set, but could be very useful for specific applications.

Fear not! Great Expectations is designed for customization and extensibility.

Building custom expectations is easy and allows your custom logic to become part of the validation, documentation, and even profiling workflows that make Great Expectations stand out. See the guide on [custom_expectations_reference](#) for more information on building expectations and updating DataContext configurations to automatically load batches of data with custom Data Assets.

Distributional Expectations

Distributional expectations help identify when new datasets or samples may be different than expected, and can help ensure that assumptions developed during exploratory analysis still hold as new data becomes available. You should use distributional expectations in the same way as other expectations: to help accelerate identification of risks as diverse as changes in a system being modeled or disruptions to a complex upstream data feed.

Great Expectations provides a direct, expressive framework for describing distributional expectations. Most distributional expectations apply nonparametric tests, although it is possible to build expectations from parameterized distributions.

Partition Objects

The core constructs of a great expectations distributional expectation are the partition (histogram bins) and associated weights.

For continuous data:

- A partition is defined by an ordered list of points that define intervals on the real number line. Note that partition intervals do not need to be uniform.
- Each bin in a partition is partially open: a data element x is in bin i if $\text{lower_bound}_i \leq x < \text{upper_bound}_i$.
- However, following the behavior of `numpy.histogram`, a data element x is in the largest bin k if $x == \text{upper_bound}_k$.
- A partition object can also include `tail_weights` which extend from `-Infinity` to the lowest bound, and from the highest bound to `+Infinity`.

- Partition weights define the probability of the associated interval. Note that this applies a “piecewise uniform” distribution to the data for the purpose of statistical tests. The weights must define a valid probability distribution, i.e. they must be non-negative numbers that sum to 1.

Example continuous partition object:

```
partition = {  
    "bins": [0, 1, 2, 10],  
    "weights": [0.2, 0.3, 0.3],  
    "tail_weights": [0.1, 0.1]  
}
```

```
>>> json.dumps(partition, indent=2)  
{  
  "bins": [  
    0,  
    1,  
    2,  
    10  
  ],  
  "weights": [  
    0.3,  
    0.3,  
    0.4  
  ],  
  "tail_weights": [  
    0.1,  
    0.1  
  ]  
}
```

For discrete/categorical data:

- A partition defines the categorical values present in the data.
- Partition weights define the probability of the associated categorical value.

Example discrete partition object:

```
{  
    "values": [ "cat", "dog", "fish"],  
    "weights": [0.3, 0.3, 0.4]  
}
```

Constructing Partition Objects

Convenience functions are available to easily construct partition objects from existing data:

- *build_continuous_partition_object*
- *build_categorical_partition_object*

Convenience functions are also provided to validate that an object is valid:

- *is_valid_continuous_partition_object*
- *is_valid_categorical_partition_object*

Tests interpret partition objects literally, so care should be taken when a partition includes a segment with zero weight. The convenience methods consequently allow you to include small amounts of residual weight on the “tails” of a dataset used to construct a partition.

Available Distributional Expectations

Kullback-Leibler (KL) divergence is available as an expectation for both categorical and continuous data (continuous data will be discretized according to the provided partition prior to computing divergence). Unlike KS and Chi-Squared tests which can use a p-value, you must provide a threshold for the relative entropy to use KL divergence. Further, KL divergence is not symmetric.

- `expect_column_kl_divergence_to_be_less_than`

For continuous data, the `expect_column_bootstrapped_ks_test_p_value_to_be_greater_than` expectation uses the Kolmogorov-Smirnov (KS) test, which compares the actual and expected cumulative densities of the data. Because of the `partition_object`'s piecewise uniform approximation of the expected distribution, the test would be overly sensitive to differences when used with a sample of data of much larger than the size of the partition. The expectation consequently uses a bootstrapping method to sample the provided data with tunable specificity.

- `expect_column_bootstrapped_ks_test_p_value_to_be_greater_than`

For categorical data, the `expect_column_chisquare_test_p_value_to_be_greater_than` expectation uses the Chi-Squared test. The Chi-Squared test works with expected and observed counts, but that is handled internally in this function – both the input and output to this function are valid partition objects (ie with weights that are probabilities and sum to 1).

- `expect_column_chisquare_test_p_value_to_be_greater_than`

Standard arguments for expectations

All Expectations return a json-serializable dictionary when evaluated, and share four standard (optional) arguments:

- `result_format`: controls what information is returned from the evaluation of the expectation expectation.
- `include_config`: If true, then the expectation suite itself is returned as part of the result object.
- `catch_exceptions`: If true, execution will not fail if the Expectation encounters an error. Instead, it will return `success = False` and provide an informative error message.
- `meta`: allows user-supplied meta-data to be stored with an expectation.

result_format

See `result_format` for more information.

include_config

All Expectations accept a boolean `include_config` parameter. If true, then the expectation suite itself is returned as part of the result object

```
>> expect_column_values_to_be_in_set (
    "my_var",
    ['B', 'C', 'D', 'F', 'G', 'H'],
    result_format="COMPLETE",
    include_config=True,
```

(continues on next page)

(continued from previous page)

```

)

{
  'exception_index_list': [0, 10, 11, 12, 13, 14],
  'exception_list': ['A', 'E', 'E', 'E', 'E', 'E'],
  'expectation_type': 'expect_column_values_to_be_in_set',
  'expectation_kwargs': {
    'column': 'my_var',
    'result_format': 'COMPLETE',
    'value_set': ['B', 'C', 'D', 'F', 'G', 'H']
  },
  'success': False
}

```

catch_exceptions

All Expectations accept a boolean *catch_exceptions* parameter. If true, execution will not fail if the Expectation encounters an error. Instead, it will return False and (in *BASIC* and *SUMMARY* modes) an informative error message

```

{
  "result": False,
  "raised_exception": True,
  "exception_traceback": "..."
}

```

catch_exceptions is on by default in command-line validation mode, and off by default in exploration mode.

meta

All Expectations accept an optional *meta* parameter. If *meta* is a valid JSON-serializable dictionary, it will be passed through to the *expectation_result* object without modification. The *meta* parameter can be used to add helpful mark-down annotations to Expectations (shown below). These Expectation “notes” are rendered within Expectation Suite pages in Data Docs.

```

>> my_df.expect_column_values_to_be_in_set(
    "my_column",
    ["a", "b", "c"],
    meta={
        "notes": {
            "format": "markdown",
            "content": [
                "#### These are expectation notes \n - you can use markdown \n - or just_
→strings"
            ]
        }
    }
)
{
  "success": False,
  "meta": {
    "notes": {
      "format": "markdown",
      "content": [

```

(continues on next page)

(continued from previous page)

```

        """### These are expectation notes \n - you can use markdown \n - or just_
    ↪strings"
    ]
    }
}

```

mostly

mostly is a special argument that is automatically available in all *column_map_expectations*. *mostly* must be a float between 0 and 1. Great Expectations evaluates it as a percentage, allowing some wiggle room when evaluating expectations: as long as *mostly* percent of rows evaluate to *True*, the expectation returns “*success*”: *True*.

```

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>> my_df.expect_column_values_to_be_between(
    "my_column",
    min_value=0,
    max_value=7
)
{
    "success": False,
    ...
}

>> my_df.expect_column_values_to_be_between(
    "my_column",
    min_value=0,
    max_value=7,
    mostly=0.7
)
{
    "success": True,
    ...
}

```

Expectations with *mostly* return exception lists even if they succeed:

```

>> my_df.expect_column_values_to_be_between(
    "my_column",
    min_value=0,
    max_value=7,
    mostly=0.7
)
{
    "success": true
    "result": {
        "unexpected_percent": 0.2,
        "partial_unexpected_index_list": [
            8,
            9
        ],
        "partial_unexpected_list": [
            8,
            9
        ]
    }
}

```

(continues on next page)

(continued from previous page)

```
    ],
    "unexpected_percent_nonmissing": 0.2,
    "unexpected_count": 2
  }
}
```

Dataset defaults

This default behavior for *result_format*, *include_config*, *catch_exceptions* can be overridden at the Dataset level:

```
my_dataset.set_default_expectation_argument("result_format", "SUMMARY")
```

In validation mode, they can be overridden using flags:

```
great_expectations validation csv my_dataset.csv my_expectations.json --result_
↪format=BOOLEAN_ONLY --catch_exceptions=False --include_config=True
```

result_format

The *result_format* parameter may be either a string or a dictionary which specifies the fields to return in *result*.

- For string usage, see *result_format*.
- For dictionary usage, *result_format* which may include the following keys:
 - *result_format*: Sets the fields to return in result.
 - *partial_unexpected_count*: Sets the number of results to include in *partial_unexpected_count*, if applicable. If set to 0, this will suppress the unexpected counts.

result_format

Great Expectations supports four values for *result_format*: *BOOLEAN_ONLY*, *BASIC*, *SUMMARY*, and *COMPLETE*. Each successive value includes more detail and so can support different use cases for working with Great Expectations, including interactive exploratory work and automatic validation.

Fields within <i>result</i>	BOOLEAN_ONLY	BASIC	SUMMARY	COMPLETE
element_count	no	yes	yes	yes
missing_count	no	yes	yes	yes
missing_percent	no	yes	yes	yes
details (dictionary)	Defined on a per-expectation basis			
Fields defined for <i>column_map_expectation</i> type expectations:				
unexpected_count	no	yes	yes	yes
unexpected_percent	no	yes	yes	yes
unexpected_percent_nonmissing	no	yes	yes	yes
partial_unexpected_list	no	yes	yes	yes
partial_unexpected_index_list	no	no	yes	yes
partial_unexpected_counts	no	no	yes	yes
unexpected_index_list	no	no	no	yes
unexpected_list	no	no	no	yes
Fields defined for <i>column_aggregate_expectation</i> type expectations:				
observed_value	no	yes	yes	yes
details (e.g. statistical details)	no	no	yes	yes

<i>result_format</i> Setting	Example use case
BOOLEAN_ONLY	Automatic validation. No result is returned.
BASIC	Exploratory analysis in a notebook.
SUMMARY	Detailed exploratory work with follow-on investigation.
COMPLETE	Debugging pipelines or developing detailed regression tests.

result_format examples

```
>> print(list(my_df.my_var))
['A', 'B', 'B', 'C', 'C', 'C', 'D', 'D', 'D', 'D', 'E', 'E', 'E', 'E', 'E', 'F', 'F',
↪ 'F', 'F', 'F', 'F', 'G', 'G', 'G', 'G', 'G', 'G', 'H', 'H', 'H', 'H', 'H', 'H',
↪ 'H', 'H']

>> my_df.expect_column_values_to_be_in_set(
    "my_var",
    ["B", "C", "D", "F", "G", "H"],
    result_format={'result_format': 'BOOLEAN_ONLY'}
)
{
    'success': False
}
```

```
>> my_df.expect_column_values_to_be_in_set(
    "my_var",
    ["B", "C", "D", "F", "G", "H"],
    result_format={'result_format': 'BASIC'}
)
```

(continues on next page)

(continued from previous page)

```

{
  'success': False,
  'result': {
    'unexpected_count': 6,
    'unexpected_percent': 0.16666666666666666,
    'unexpected_percent_nonmissing': 0.16666666666666666,
    'partial_unexpected_list': ['A', 'E', 'E', 'E', 'E', 'E']
  }
}

>> expect_column_values_to_match_regex(
  "my_column",
  "[A-Z][a-z]+",
  result_format={'result_format': 'SUMMARY'}
)
{
  'success': False,
  'result': {
    'element_count': 36,
    'unexpected_count': 6,
    'unexpected_percent': 0.16666666666666666,
    'unexpected_percent_nonmissing': 0.16666666666666666,
    'missing_count': 0,
    'missing_percent': 0.0,
    'partial_unexpected_counts': [{'value': 'A', 'count': 1}, {'value': 'E',
↪ 'count': 5}],
    'partial_unexpected_index_list': [0, 10, 11, 12, 13, 14],
    'partial_unexpected_list': ['A', 'E', 'E', 'E', 'E', 'E']
  }
}

>> my_df.expect_column_values_to_be_in_set(
  "my_var",
  ["B", "C", "D", "F", "G", "H"],
  result_format={'result_format': 'COMPLETE'}
)
{
  'success': False,
  'result': {
    'unexpected_index_list': [0, 10, 11, 12, 13, 14],
    'unexpected_list': ['A', 'E', 'E', 'E', 'E', 'E']
  }
}

```

The out-of-the-box default is `{'result_format': 'BASIC'}`.

Behavior for *BOOLEAN_ONLY*

When the *result_format* is *BOOLEAN_ONLY*, no *result* is returned. The result of evaluating the expectation is exclusively returned via the value of the *success* parameter.

For example:

```
>> my_df.expect_column_values_to_be_in_set(
    "possible_benefactors",
    ["Joe Gargery", "Mrs. Gargery", "Mr. Pumblechook", "Ms. Havisham", "Mr. Jaggers"]
    result_format={'result_format': 'BOOLEAN_ONLY'}
)
{
    'success': False
}

>> my_df.expect_column_values_to_be_in_set(
    "possible_benefactors",
    ["Joe Gargery", "Mrs. Gargery", "Mr. Pumblechook", "Ms. Havisham", "Mr. Jaggers",
    ↪ "Mr. Magwitch"]
    result_format={'result_format': 'BOOLEAN_ONLY'}
)
{
    'success': False
}
```

Behavior for *BASIC*

A *result* is generated with a basic justification for why an expectation was met or not. The format is intended for quick, at-a-glance feedback. For example, it tends to work well in jupyter notebooks.

Great Expectations has standard behavior for support for describing the results of *column_map_expectation* and *column_aggregate_expectation* expectations.

column_map_expectation applies a boolean test function to each element within a column, and so returns a list of unexpected values to justify the expectation result.

The basic *result* includes:

```
{
    "success" : Boolean,
    "result" : {
        "partial_unexpected_list" : [A list of up to 20 values that violate the_
    ↪ expectation]
        "unexpected_count" : The total count of unexpected values in the column
        "unexpected_percent" : The overall percent of unexpected values
        "unexpected_percent_nonmissing" : The percent of unexpected values, excluding_
    ↪ missing values from the denominator
    }
}
```

Note: when unexpected values are duplicated, *unexpected_list* will contain multiple copies of the value.

```
[1, 2, 2, 3, 3, 3, None, None, None, None]
```

```
expect_column_values_to_be_unique
```

(continues on next page)

(continued from previous page)

```
{
  "success" : Boolean,
  "result" : {
    "partial_unexpected_list" : [2,2,3,3,3]
    "unexpected_count" : 5,
    "unexpected_percent" : 0.5,
    "unexpected_percent_nonmissing" : 0.8333333
  }
}
```

column_aggregate_expectation computes a single aggregate value for the column, and so returns a single *observed_value* to justify the expectation result.

The basic *result* includes:

```
{
  "success" : Boolean,
  "result" : {
    "observed_value" : The aggregate statistic computed for the column
  }
}
```

For example:

```
[1, 1, 2, 2]

expect_column_mean_to_be_between

{
  "success" : Boolean,
  "result" : {
    "observed_value" : 1.5
  }
}
```

Behavior for **SUMMARY**

A *result* is generated with a summary justification for why an expectation was met or not. The format is intended for more detailed exploratory work and includes additional information beyond what is included by *BASIC*. For example, it can support generating dashboard results of whether a set of expectations are being met.

Great Expectations has standard behavior for support for describing the results of *column_map_expectation* and *column_aggregate_expectation* expectations.

column_map_expectation applies a boolean test function to each element within a column, and so returns a list of unexpected values to justify the expectation result.

The summary *result* includes:

```
{
  'success': False,
  'result': {
    'element_count': The total number of values in the column
    'unexpected_count': The total count of unexpected values in the column (also
↪ in `BASIC`)
    'unexpected_percent': The overall percent of unexpected values (also in
↪ `BASIC`)
```

(continues on next page)

(continued from previous page)

```

    'unexpected_percent_nonmissing': The percent of unexpected values, excluding
    ↪missing values from the denominator (also in `BASIC`)
    "partial_unexpected_list" : [A list of up to 20 values that violate the
    ↪expectation] (also in `BASIC`)
    'missing_count': The number of missing values in the column
    'missing_percent': The total percent of missing values in the column
    'partial_unexpected_counts': [{A list of objects with value and counts,
    ↪showing the number of times each of the unexpected values occurs}]
    'partial_unexpected_index_list': [A list of up to 20 of the indices of the
    ↪unexpected values in the column]
    }
}

```

For example:

```

{
  'success': False,
  'result': {
    'element_count': 36,
    'unexpected_count': 6,
    'unexpected_percent': 0.16666666666666666,
    'unexpected_percent_nonmissing': 0.16666666666666666,
    'missing_count': 0,
    'missing_percent': 0.0,
    'partial_unexpected_counts': [{'value': 'A', 'count': 1}, {'value': 'E',
    ↪'count': 5}],
    'partial_unexpected_index_list': [0, 10, 11, 12, 13, 14],
    'partial_unexpected_list': ['A', 'E', 'E', 'E', 'E', 'E']
  }
}

```

column_aggregate_expectation computes a single aggregate value for the column, and so returns a *observed_value* to justify the expectation result. It also includes additional information regarding observed values and counts, depending on the specific expectation.

The summary *result* includes:

```

{
  'success': False,
  'result': {
    'observed_value': The aggregate statistic computed for the column (also in
    ↪`BASIC`)
    'element_count': The total number of values in the column
    'missing_count': The number of missing values in the column
    'missing_percent': The total percent of missing values in the column
    'details': {<expectation-specific result justification fields>}
  }
}

```

For example:

```

[1, 1, 2, 2, NaN]

expect_column_mean_to_be_between

{
  "success" : Boolean,

```

(continues on next page)

(continued from previous page)

```

    "result" : {
      "observed_value" : 1.5,
      'element_count': 5,
      'missing_count': 1,
      'missing_percent': 0.2
    }
  }
}

```

Behavior for *COMPLETE*

A *result* is generated with all available justification for why an expectation was met or not. The format is intended for debugging pipelines or developing detailed regression tests.

Great Expectations has standard behavior for support for describing the results of *column_map_expectation* and *column_aggregate_expectation* expectations.

column_map_expectation applies a boolean test function to each element within a column, and so returns a list of unexpected values to justify the expectation result.

The complete *result* includes:

```

{
  'success': False,
  'result': {
    "unexpected_list" : [A list of all values that violate the expectation]
    'unexpected_index_list': [A list of the indices of the unexpected values in_
↪the column]
    'element_count': The total number of values in the column (also in `SUMMARY`)
    'unexpected_count': The total count of unexpected values in the column (also_
↪in `SUMMARY`)
    'unexpected_percent': The overall percent of unexpected values (also in_
↪`SUMMARY`)
    'unexpected_percent_nonmissing': The percent of unexpected values, excluding_
↪missing values from the denominator (also in `SUMMARY`)
    'missing_count': The number of missing values in the column (also in_
↪`SUMMARY`)
    'missing_percent': The total percent of missing values in the column (also_
↪in `SUMMARY`)
  }
}

```

For example:

```

{
  'success': False,
  'result': {
    'element_count': 36,
    'unexpected_count': 6,
    'unexpected_percent': 0.16666666666666666,
    'unexpected_percent_nonmissing': 0.16666666666666666,
    'missing_count': 0,
    'missing_percent': 0.0,
    'unexpected_index_list': [0, 10, 11, 12, 13, 14],
    'unexpected_list': ['A', 'E', 'E', 'E', 'E', 'E']
  }
}

```

column_aggregate_expectation computes a single aggregate value for the column, and so returns a *observed_value* to justify the expectation result. It also includes additional information regarding observed values and counts, depending on the specific expectation.

The complete *result* includes:

```
{
  'success': False,
  'result': {
    'observed_value': The aggregate statistic computed for the column (also in `SUMMARY`)
    'element_count': The total number of values in the column (also in `SUMMARY`)
    'missing_count': The number of missing values in the column (also in `SUMMARY`)
    'missing_percent': The total percent of missing values in the column (also in `SUMMARY`)
    'details': {<expectation-specific result justification fields, which may be more detailed than in `SUMMARY`>}
  }
}
```

For example:

```
[1, 1, 2, 2, NaN]

expect_column_mean_to_be_between

{
  "success" : Boolean,
  "result" : {
    "observed_value" : 1.5,
    "element_count": 5,
    "missing_count": 1,
    "missing_percent": 0.2
  }
}
```

Table of Expectation Implementations By Backend

Because Great Expectations can run against different platforms, not all expectations have been implemented for all platforms. This table details which are implemented. Note we love pull-requests to help us fill out the missing implementations!

Expectations	Pandas	SQL	Spark
<i>expect_column_to_exist</i>	Y	Y	Y
<i>expect_table_columns_to_match_ordered_list</i>	Y	Y	Y
<i>expect_table_row_count_to_be_between</i>	Y	Y	Y
<i>expect_table_row_count_to_equal</i>	Y	Y	Y
<i>expect_column_values_to_be_unique</i>	Y	Y	Y
<i>expect_column_values_to_not_be_null</i>	Y	Y	Y
<i>expect_column_values_to_be_null</i>	Y	Y	Y
<i>expect_column_values_to_be_of_type</i>	Y	Y	Y
<i>expect_column_values_to_be_in_type_list</i>	Y	Y	Y
<i>expect_column_values_to_be_in_set</i>	Y	Y	Y

continues on next page

Table 1 – continued from previous page

<i>expect_column_values_to_not_be_in_set</i>	Y	Y	Y
<i>expect_column_values_to_be_between</i>	Y	Y	Y
<i>expect_column_values_to_be_increasing</i>	Y	N	N
<i>expect_column_values_to_be_decreasing</i>	Y	N	N
<i>expect_column_value_lengths_to_be_between</i>	Y	Y	Y
<i>expect_column_value_lengths_to_equal</i>	Y	Y	Y
<i>expect_column_values_to_match_regex</i>	Y	Y	Y
<i>expect_column_values_to_not_match_regex</i>	Y	Y	Y
<i>expect_column_values_to_match_regex_list</i>	Y	Y	N
<i>expect_column_values_to_not_match_regex_list</i>	Y	Y	N
<i>expect_column_values_to_match_strftime_format</i>	Y	N	N
<i>expect_column_values_to_be_dateutil_parseable</i>	Y	N	N
<i>expect_column_values_to_be_json_parseable</i>	Y	N	N
<i>expect_column_values_to_match_json_schema</i>	Y	N	N
<i>expect_column_parameterized_distribution_ks_test_p_value_to_be_greater_than</i>	Y	N	N
<i>expect_column_distinct_values_to_equal_set</i>	Y	Y	Y
<i>expect_column_distinct_values_to_contain_set</i>	Y	Y	Y
<i>expect_column_mean_to_be_between</i>	Y	Y	Y
<i>expect_column_median_to_be_between</i>	Y	Y	Y
<i>expect_column_stdev_to_be_between</i>	Y	N	Y
<i>expect_column_unique_value_count_to_be_between</i>	Y	Y	Y
<i>expect_column_proportion_of_unique_values_to_be_between</i>	Y	Y	Y
<i>expect_column_most_common_value_to_be_in_set</i>	Y	N	Y
<i>expect_column_sum_to_be_between</i>	Y	Y	Y
<i>expect_column_min_to_be_between</i>	Y	Y	Y
<i>expect_column_max_to_be_between</i>	Y	Y	Y
<i>expect_column_chisquare_test_p_value_to_be_greater_than</i>	Y	Y	Y
<i>expect_column_bootstrapped_ks_test_p_value_to_be_greater_than</i>	Y	N	N
<i>expect_column_kl_divergence_to_be_less_than</i>	Y	Y	Y
<i>expect_column_pair_values_to_be_equal</i>	Y	N	Y
<i>expect_column_pair_values_A_to_be_greater_than_B</i>	Y	N	Y
<i>expect_column_pair_values_to_be_in_set</i>	Y	N	N
<i>expect_multicolumn_values_to_be_unique</i>	Y	N	Y

4.2.2 Validation

Great Expectations makes it possible to validate your data against an Expectation Suite. Validation produces a detailed report of how the data meets your expectations – and where it doesn't.

Attention: The `DataAsset` class will be refactored and renamed in an upcoming release of Great Expectations to make it easier to create custom expectations and ensure Expectation Implementations are consistent across different validation engines.

A **DataAsset** is a Great Expectations object that can create and validate Expectations against specific data. DataAssets are connected to data. A DataAsset can evaluate Expectations wherever you access your data, using different **ValidationEngines** such as Pandas, Spark, or SQLAlchemy.

An **Expectation Validation Result** captures the output of checking an expectation against data. It describes whether the data met the expectation, and additional metrics from the data such as the percentage of unique values or observed mean.

An **Expectation Suite Validation Result** combines multiple Expectation Validation Results and metadata about the validation into a single report.

A **Metric** is a value produced by Great Expectations when evaluating one or more batches of data, such as an observed mean or distribution of data.

A **Validation Operator** stitches together resources provided by the Data Context to provide an easy way to deploy Great Expectations in your environment. It executes configurable ****Action****s such as updating Data Docs, sending a notification to your team about validation results, or storing a result in a shared S3 bucket.

A **Checkpoint** is a configuration for a Validation Operator that specifies which Batches of data and Expectation Suites should be validated.

Validation

Once you've constructed and stored Expectations, you can use them to validate new data. Validation generates a report that details any specific deviations from expected values.

We recommend using *DataContexts* to manage expectation suites and coordinate validation across runs.

Validation Results

The report contains information about:

- the overall success (the *success* field),
- summary statistics of the expectations (the *statistics* field), and
- the detailed results of each expectation (the *results* field).

An example report looks like the following:

```
>> import json
>> import great_expectations as ge
>> my_expectation_suite = json.load(file("my_titanic_expectations.json"))
>> my_df = ge.read_csv(
    "./tests/examples/titanic.csv",
    expectation_suite=my_expectation_suite
)
>> my_df.validate()

{
  "results" : [
    {
      "expectation_type": "expect_column_to_exist",
      "success": True,
      "kwargs": {
        "column": "Unnamed: 0"
      }
    },
    ...
    {
      "unexpected_list": 30.397989417989415,
      "expectation_type": "expect_column_mean_to_be_between",
      "success": True,
      "kwargs": {
        "column": "Age",
        "max_value": 40,
```

(continues on next page)

```
        "min_value": 20
    }
},
{
    "unexpected_list": [],
    "expectation_type": "expect_column_values_to_be_between",
    "success": True,
    "kwargs": {
        "column": "Age",
        "max_value": 80,
        "min_value": 0
    }
},
{
    "unexpected_list": [
        "Downton (?Douton), Mr William James",
        "Jacobsohn Mr Samuel",
        "Seman Master Betros"
    ],
    "expectation_type": "expect_column_values_to_match_regex",
    "success": True,
    "kwargs": {
        "regex": "[A-Z][a-z]+(?: \\([A-Z][a-z]+\\))?",
        "column": "Name",
        "mostly": 0.95
    }
},
{
    "unexpected_list": [
        "*"
    ],
    "expectation_type": "expect_column_values_to_be_in_set",
    "success": False,
    "kwargs": {
        "column": "PClass",
        "value_set": [
            "1st",
            "2nd",
            "3rd"
        ]
    }
},
],
"success", False,
"statistics": {
    "evaluated_expectations": 10,
    "successful_expectations": 9,
    "unsuccessful_expectations": 1,
    "success_percent": 90.0,
}
```

Reviewing Validation Results

The easiest way to review Validation Results is to view them from your local Data Docs site, where you can also conveniently view Expectation Suites and with additional configuration, Profiling Results (see [Data Docs Site Configuration](#)). Out of the box, Great Expectations Data Docs is configured to compile a local data documentation site when you start a new project by running `great_expectations init`. By default, this local site is saved to the `uncommitted/data_docs/local_site/` directory of your project and will contain pages for Expectation Suites and Validation Results.

If you would like to review the raw validation results in JSON format, the default Validation Results directory is `uncommitted/validations/`. Note that by default, Data Docs will only compile Validation Results located in this directory.

To learn more about setting up Great Expectations for your team read [Using Great Expectations on Teams](#).

Validation Operators

The example above demonstrates how to validate one batch of data against one expectation suite. The `validate` method returns a dictionary of validation results. This is sufficient when exploring your data and getting to know Great Expectations. When deploying Great Expectations in a real data pipeline, you will typically discover additional needs:

- validating a group of batches that are logically related
- validating a batch against several expectation suites
- doing something with the validation results (e.g., saving them for a later review, sending notifications in case of failures, etc.).

Validation Operators are mini-applications that can be configured to implement these scenarios.

Read [Validation Operators And Actions Introduction](#) to learn more.

Deployment patterns

Useful deployment patterns include:

- Include validation at the end of a complex data transformation, to verify that no cases were lost, duplicated, or improperly merged.
- Include validation at the *beginning* of a script applying a machine learning model to a new batch of data, to verify that its distributed similarly to the training and testing set.
- Automatically trigger table-level validation when new data is dropped to an FTP site or S3 bucket, and send the validation report to the uploader and bucket owner by email.
- Schedule database validation jobs using cron, then capture errors and warnings (if any) and post them to Slack.
- Validate as part of an Airflow task: if Expectations are violated, raise an error and stop DAG propagation until the problem is resolved. Alternatively, you can implement expectations that raise warnings without halting the DAG.

For certain deployment patterns, it may be useful to parameterize expectations, and supply evaluation parameters at validation time. See [Evaluation Parameters](#) for more information.

Validation Operators And Actions Introduction

The *validate* method evaluates one batch of data against one expectation suite and returns a dictionary of validation results. This is sufficient when you explore your data and get to know Great Expectations. When deploying Great Expectations in a real data pipeline, you will typically discover additional needs:

- validating a group of batches that are logically related
- validating a batch against several expectation suites
- doing something with the validation results (e.g., saving them for a later review, sending notifications in case of failures, etc.).

Validation Operators provide a convenient abstraction for both bundling the validation of multiple expectation suites and the actions that should be taken after the validation.

Finding a Validation Operator that is right for you

Each Validation Operator encodes a particular set of business rules around validation. Currently, two Validation Operator implementations are included in the Great Expectations distribution.

The classes that implement them are in `great_expectations.validation_operators` module.

ActionListValidationOperator

ActionListValidationOperator validates each batch in the list that is passed as *assets_to_validate* argument to its *run* method against the expectation suite included within that batch and then invokes a list of configured actions on every validation result.

Actions are Python classes with a *run* method that takes a result of validating a batch against an expectation suite and does something with it (e.g., save validation results to the disk or send a Slack notification). Classes that implement this API can be configured to be added to the list of actions for this operator (and other operators that use actions). Read more about actions here: [actions](#).

An instance of this operator is included in the default configuration file *great_expectations.yml* that *great_expectations init* command creates.

A user can choose the actions to perform in their instance of the operator.

Read more about ActionListValidationOperator here: [action_list_validation_operator](#).

WarningAndFailureExpectationSuitesValidationOperator

WarningAndFailureExpectationSuitesValidationOperator implements a business logic pattern that many data practitioners consider useful.

It validates each batch of data against two different expectation suites: “failure” and “warning”. Group the expectations that you create about a data asset into two expectation suites. The critical expectations that you are confident that the data should meet go into the expectation suite that this operator calls “failure” (meaning, not meeting these expectations should be considered a failure in the pipeline). The rest of expectations go into the “warning” expectation suite.

The failure Expectation Suite contains expectations that are considered important enough to justify stopping the pipeline when they are violated.

WarningAndFailureExpectationSuitesValidationOperator retrieves two expectation suites for every data asset in the *assets_to_validate* argument of its *run* method.

After completing all the validations, it sends a Slack notification with the success status. Note that it doesn't use an Action to send its Slack notification, because the notification has also been customized to summarize information from both suites.

Read more about `WarningAndFailureExpectationSuitesValidationOperator` here: `warning_and_failure_expectation_suites_validation_operator`

ValidationActions

The Validation Operator implementations above invoke actions.

An action is a way to take an arbitrary method and make it configurable and runnable within a data context.

The only requirement from an action is for it to have a `take_action` method.

GE comes with a list of actions that we consider useful and you can reuse in your pipelines. Most of them take in validation results and do something with them.

Validation Results

Validation results include information about each expectation that was validated and a `statistics` section that describes the overall number of validated expectations and results.

For example:

```
{
  "meta": {
    "data_asset_name": "data_dir/default/titanic",
    "expectation_suite_name": "default",
    "run_id": {
      "run_name": "my_run_name",
      "run_time": "2019-07-01T123434.12345Z"
    }
  },
  "results": [
    {
      "expectation_config": {
        "expectation_type": "expect_column_values_to_have_odd_lengths",
        "kwargs": {
          "column": "Name",
          "result_format": "SUMMARY"
        }
      },
      "exception_info": {
        "exception_message": null,
        "exception_traceback": null,
        "raised_exception": false
      },
      "success": false,
      "result": {
        "partial_unexpected_index_list": [
          0,
          5,
          6,
          7,
          8,
          9,
        ]
      }
    }
  ]
}
```

(continues on next page)

(continued from previous page)

```
11,  
12,  
13,  
14,  
15,  
18,  
20,  
21,  
22,  
23,  
27,  
31,  
32,  
33  
],  
"unexpected_count": 660,  
"unexpected_percent": 0.5026656511805027,  
"partial_unexpected_list": [  
    "Allen, Miss Elisabeth Walton",  
    "Anderson, Mr Harry",  
    "Andrews, Miss Kornelia Theodosia",  
    "Andrews, Mr Thomas, jr",  
    "Appleton, Mrs Edward Dale (Charlotte Lamson)",  
    "Artagaveytia, Mr Ramon",  
    "Astor, Mrs John Jacob (Madeleine Talmadge Force)",  
    "Aubert, Mrs Leontine Pauline",  
    "Barkworth, Mr Algernon H",  
    "Baumann, Mr John D",  
    "Baxter, Mrs James (Helene DeLaudeniére Chaput)",  
    "Beckwith, Mr Richard Leonard",  
    "Behr, Mr Karl Howell",  
    "Birnbaum, Mr Jakob",  
    "Bishop, Mr Dickinson H",  
    "Bishop, Mrs Dickinson H (Helen Walton)",  
    "Bonnell, Miss Caroline",  
    "Bowerman, Miss Elsie Edith",  
    "Bradley, Mr George",  
    "Brady, Mr John Bertram"  
],  
"missing_percent": 0.0,  
"element_count": 1313,  
"unexpected_percent_nonmissing": 0.5026656511805027,  
"missing_count": 0  
}  
}  
],  
"success": false,  
"statistics": {  
    "evaluated_expectations": 1,  
    "successful_expectations": 0,  
    "unsuccessful_expectations": 1,  
    "success_percent": 0  
}  
}
```

Reviewing Validation Results

TODO: Description of process for reviewing validation results

Metrics Reference

Metrics are still a **beta feature** in Great Expectations. Expect changes to the API.

A Metric is a value that Great Expectations can use to evaluate expectations or to store externally. A metric could be a statistic, such as the minimum value of the column, or a more complex object, such as a histogram.

Expectation Validation Results and Expectation Suite Validation Results can expose metrics that are defined by specific expectations that have been validated, called “Expectation Defined Metrics.” In the future, we plan to allow Expectations to have more control over the metrics that they generate, expose, and use in testing.

The following examples demonstrate how metrics are defined:

```
res = df.expect_column_values_to_be_in_set("Sex", ["male", "female"])
res.get_metric("expect_column_values_to_be_in_set.result.missing_count", column="Sex")
```

Metrics

Metrics are values derived from one or more Data Assets that can be used to evaluate expectations or to summarize the result of Validation. A Metric is obtained from an ExpectationValidationResult or ExpectationSuiteValidationResult by providing the *metric_name* and *metric_kwargs* or *metric_kwargs_id*.

A metric name is a dot-delimited string that identifies the value, such as *expect_column_values_to_be_unique.success* or *expect_column_values_to_be_between.result.unexpected_percent*.

Metric Kwargs are key-value pairs that identify the metric within the context of the validation, such as “column”: “Age”. Different metrics may require different Kwargs.

A *metric_kwargs_id* is a string representation of the Metric Kwargs that can be used as a database key. For simple cases, it could be easily readable, such as *column=Age*, but when there are multiple keys and values or complex values, it will most likely be an md5 hash of key/value pairs. It can also be None in the case that there are no kwargs required to identify the metric.

See the [Metrics Reference](#) or [Saving Metrics Tutorial](#) for more information.

4.2.3 Data Context

A **Data Context** stitches together all the features available with Great Expectations, making it possible to easily manage configurations for resources such as Datasources, Validation Operators, Data Docs Sites, and Stores.

A **Data Context Configuration** is a yaml file that can be committed to source control to ensure that all the settings related to your validation are appropriately versioned and visible to your team. It can flexibly describe plugins and other customizations for accessing datasources or building data docs sites.

A **Store** provides a consistent API to manage access to Expectations, Expectation Suite Validation Results and other Great Expectations assets, making it easy to share resources across a team that uses AWS, Azure, GCP, local storage, or something else entirely.

An **Evaluation Parameter Store** makes it possible to build expectation suites that depend on values from other batches of data, such as ensuring that the number of rows in a downstream dataset equals the number of unique values from an upstream one. A Data Context can manage a store to facilitate that validation scenario.

A **Metric** Store makes facilitates saving any metric or statistic generated during validation, for example making it easy to create a dashboard showing key output from running Great Expectations.

DataContexts

A DataContext represents a Great Expectations project. It organizes storage and access for expectation suites, data-sources, notification settings, and data fixtures.

The DataContext is configured via a yml file stored in a directory called `great_expectations`; the configuration file as well as managed expectation suites should be stored in version control.

DataContexts manage connections to your data and compute resources, and support integration with execution frameworks (such as airflow, Nifi, dbt, or dagster) to describe and produce batches of data ready for analysis. Those features enable fetching, validation, profiling, and documentation of your data in a way that is meaningful within your existing infrastructure and work environment.

DataContexts also manage Expectation Suites. Expectation Suites combine multiple Expectation Configurations into an overall description of a dataset. Expectation Suites should have names corresponding to the kind of data they define, like “NPI” for National Provider Identifier data or “company.users” for a users table.

The DataContext also provides other services, such as storing and substituting evaluation parameters during validation. See [Data Context Evaluation Parameter Store](#) for more information.

See the `data_context_reference` for more information.

Evaluation Parameters

Often, the specific parameters associated with an expectation will be derived from upstream steps in a processing pipeline. For example, we may want to *expect_table_row_count_to_equal* a value stored in a previous step.

Great Expectations makes it possible to use “Evaluation Parameters” to accomplish that goal. We declare Expectations using parameters that need to be provided at validation time. During interactive development, we can even provide a temporary value that should be used during the initial evaluation of the expectation.

```
>>> my_df.expect_table_row_count_to_equal(
...     value={"$PARAMETER": "upstream_row_count", "$PARAMETER.upstream_row_count": 10}
...     result_format={'result_format': 'BOOLEAN_ONLY'})
{
  'success': True
}
```

More typically, when validating expectations, you can provide evaluation parameters that are only available at runtime:

```
my_df.validate(expectation_suite=my_dag_step_config, evaluation_parameters={"upstream_
↪row_count": upstream_row_count})
```


Evaluation Parameter Expressions

In many cases, evaluation parameters are most useful when they can allow a range of values. For example, we might want to specify that a new table's row count should be between 90 - 110 % of an upstream table's row count (or a count from a previous run). Evaluation parameters support basic arithmetic expressions to accomplish that goal:

```
>>> my_df.expect_table_row_count_to_be_between(
...     min_value={"$PARAMETER": "trunc(upstream_row_count * 0.9)"},
...     max_value={"$PARAMETER": "trunc(upstream_row_count * 1.1)"},
...     result_format={'result_format': 'BOOLEAN_ONLY'})
{
  'success': True
}
```

Evaluation parameters are not limited to simple values, for example you could include a list as a parameter value:

```
my_df.column_values_to_be_in_set("my_column", value_set={"$PARAMETER": "runtime_values
↪"})
my_df.validate(evaluation_parameters={"runtime_values": [1, 2, 3]})
```

However, it is not possible to mix complex values with arithmetic expressions.

Storing Evaluation Parameters

Data Context Evaluation Parameter Store

A DataContext can automatically identify and store evaluation parameters that are referenced in other expectation suites. The evaluation parameter store uses a URN schema for identifying dependencies between expectation suites.

The DataContext-recognized URN must begin with the string `urn:great_expectations`. Valid URNs must have one of the following structures to be recognized by the Great Expectations DataContext:

```
urn:great_expectations:validations:<expectation_suite_name>:<metric_name>
urn:great_expectations:validations:<expectation_suite_name>:<metric_name>:<metric_
↪kwargs_id>
```

Replace names in `<>` with the desired name. For example:

```
urn:great_expectations:validations:dickens_data:expect_column_proportion_of_unique_
↪values_to_be_between,result.observed_value:column=Title
```

Storing Parameters in an Expectation Suite

You can also store parameter values in a special dictionary called `evaluation_parameters` that is stored in the `expectation_suite` to be available to multiple expectations or while declaring additional expectations.

```
>>> my_df.set_evaluation_parameter("upstream_row_count", 10)
>>> my_df.get_evaluation_parameter("upstream_row_count")
```

If a parameter has been stored, then it does not need to be provided for a new expectation to be declared:

```
>>> my_df.set_evaluation_parameter("upstream_row_count", 10)
>>> my_df.expect_table_row_count_to_be_between(max_value={"$PARAMETER": "upstream_row_
↪count"})
```

4.2.4 Datasources

Attention: Datasource configuration will be changing soon to make it easier to:

- adjust configuration for where data is stored and validated independently.
- understand the roles of Batch Kwarg, Batch Kwarg Generators, Batch Parameters, and Batch Markers.

Datasources, Batch Kwarg Generators, Batch Parameters, and Batch Kwarg make it easier to connect Great Expectations to your data. Together, they address questions such as:

- How do I get data into my Great Expectations DataAsset?
- How do I tell my Datasource how to access my specific data?
- How do I use Great Expectations to store Batch Kwarg configurations or logically describe data when I need to build equivalent Batch Kwarg for different datasources?
- How do I know what data is available from my datasource?

A **Datasource** is a connection to a **Validation Engine** (a compute environment such as Pandas, Spark, or a SQL-compatible database) and one or more data storage locations. For a SQL database, the Validation Engine and data storage locations will be the same, but for Spark or Pandas, you may be reading data from a remote location such as an S3 bucket but validating it in a cluster or local machine. The Datasource produces Batches of data that Great Expectations can validate in that environment.

Batch Kwarg are specific instructions for a Datasource about what data should be prepared as a Batch for validation. The Batch could reference a specific database table, the most recent log file delivered to S3, or a subset of one of those objects, for example just the first 10,000 rows.

A **Batch Kwarg Generator** produces datasource-specific Batch Kwarg. The most basic Batch Kwarg Generator simply stores Batch Kwarg by name to make it easy to retrieve them and obtain batches of data. But Batch Kwarg Generators can be more powerful and offer stronger guarantees about reproducibility and compatibility with other tools, and help identify available Data Assets and Partitions by inspecting a storage environment.

Batch Parameters provide instructions for how to retrieve stored Batch Kwarg or build new Batch Kwarg that reflect partitions, deliveries, or slices of logical data assets.

Batch Markers provide additional metadata a batch to help ensure reproducibility, such as the timestamp at which it was created.

A **Batch Kwarg Generator** translates Batch Parameters to datasource-specific Batch Kwarg. A Batch Kwarg Generator can also identify data assets and partitions by inspecting a storage environment.

Datasources

Datasources are responsible for connecting data and compute infrastructure. Each Datasource provides Great Expectations Data Assets connected to a specific compute environment, such as a SQL database, a Spark cluster, or a local in-memory Pandas DataFrame. Datasources know how to access data from relevant sources such as an existing object from a DAG runner, a SQL database, an S3 bucket, GCS, or a local filesystem.

To bridge the gap between those worlds, Datasources can interact closely with *Batch Kwarg Generators* which are aware of a source of data and can produce identifying information, called “batch_kwarg” that datasources can use to get individual batches of data.

See [Datasource Reference](#) for more detail about configuring and using datasources in your DataContext.

See `datasource module docs` `datasource_module` for more detail about available datasources.

Datasource Reference

To have a Datasource produce Data Assets of a custom type, such as when adding custom expectations by subclassing an existing DataAsset type, use the `data_asset_type` parameter to configure the datasource to load and return DataAssets of the custom type.

For example:

```
datasources:
  pandas:
    class_name: PandasDatasource
    data_asset_type:
      class_name: MyCustomPandasAsset
      module_name: internal_pandas_assets
```

Given the above configuration, we can observe the following:

```
>>> batch_kwargs = {
...     "datasource": "pandas",
...     "dataset": {"a": [1, 2, 3]}
... }
>>> batch = context.get_batch(batch_kwargs, my_suite)
>>> isinstance(batch, MyCustomPandasAsset)
True
```

Batch Kwarg Generators

`batch_kwargs` are specific instructions for a *Datasources* about what data should be prepared as a “batch” for validation. The batch could be a specific database table, the most recent log file delivered to S3, or even a subset of one of those objects such as the first 10,000 rows.

A BatchKwargGenerator builds those instructions for Datasources by inspecting storage backends or data, or by maintaining configuration such as commonly-used paths or filepath conventions. That allows BatchKwargGenerators to add flexibility in how to obtain data such as by exposing time-based partitions or sampling data.

For example, a Batch Kwarg Generator could be **configured** to produce a SQL query that logically represents “rows in the Events table with a type code of ‘X’ that occurred within seven days of a given timestamp.” With that configuration, you could provide a timestamp as a partition name, and the Batch Kwarg Generator will produce instructions that a SQLAlchemyDatasource could use to materialize a SQLAlchemyDataset corresponding to that batch of data and ready for validation.

4.2.5 Data Docs

With Great Expectations, your tests are your docs, and your docs are your tests. Data Docs makes it possible to produce clear visual descriptions of what you expect, what you observe, and how they differ.

An **Expectation Suite Renderer** creates a page that shows what you expect from data. Its language is prescriptive, for example translating a fully-configured `expect_column_values_to_not_be_null` expectation into the English phrase, “column address values must not be null, at least 80% of the time.”

A **Validation Result Renderer** produces an overview of the result of validating a batch of data with an Expectation Suite. It shows the difference between observed and expected values.

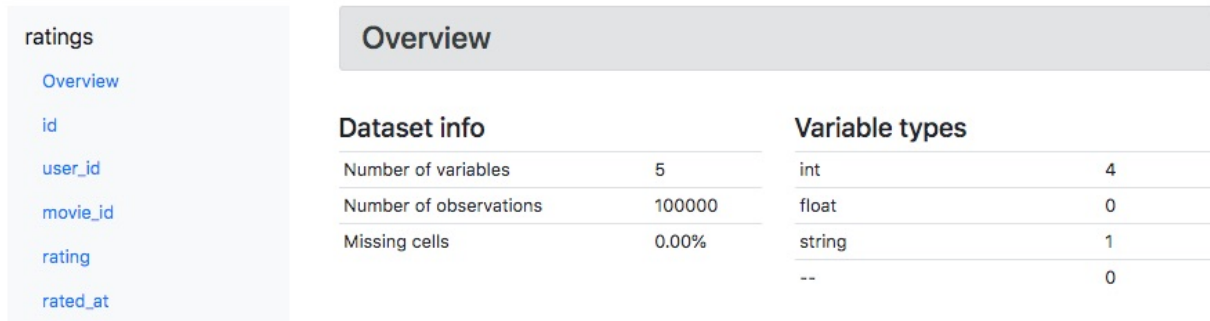
A **Profiling Renderer** details the observed metrics produced from a validation without comparing them to specific expected values. It provides a detailed look into what Great Expectations learned about your data.

Data Docs

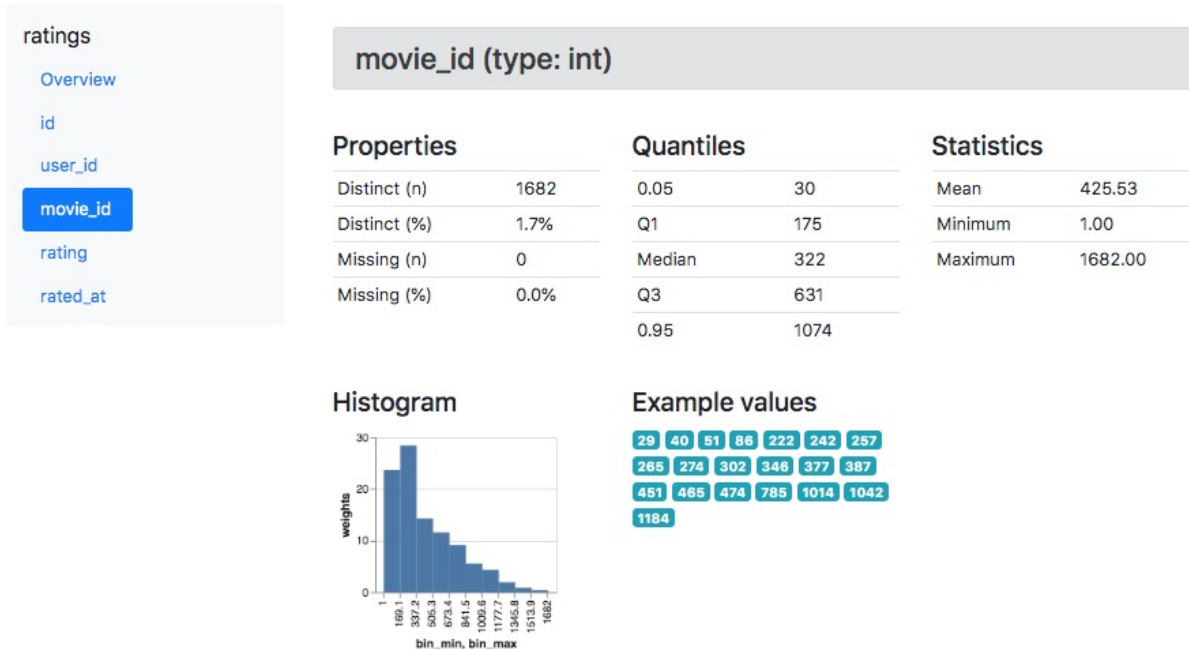
Data Docs compiles raw Great Expectations objects including Expectations and Validations into structured documents such as HTML documentation that display key characteristics of a dataset.

HTML documentation takes expectation suites and validation results and produces clear, functional, and self-healing documentation of expected and observed data characteristics. Together with profiling, it can help to rapidly create a clearer picture of your data, and keep your entire team on the same page as data evolves.

For example, the default BasicDatasetProfiler in GE will produce validation_results which compile to a page for each table or DataFrame including an overview section:



And then detailed statistics for each column:



The Great Expectations DataContext uses a configurable “data documentation site” to define which artifacts to compile and how to render them as documentation. Multiple sites can be configured inside a project, each suitable for a particular data documentation use case.

For example, we have identified three common use cases for using documentation in a data project. They are to:

1. Visualize all Great Expectations artifacts from the local repository of a project as HTML: expectation suites, validation results and profiling results.
2. Maintain a “shared source of truth” for a team working on a data project. Such documentation renders all the

artifacts committed in the source control system (expectation suites and profiling results) and a continuously updating data quality report, built from a chronological list of validations by run id.

3. Share a spec of a dataset with a client or a partner. This is similar to API documentation in software development. This documentation would include profiling results of the dataset to give the reader a quick way to grasp what the data looks like, and one or more expectation suites that encode what is expected from the data to be considered valid.


To support these (and possibly other) use cases Great Expectations has a concept of “data documentation site”. Multiple sites can be configured inside a project, each suitable for a particular data documentation use case.

Here is an example of a site:

diabetes_source

default

Data Asset	Profiling Results	Expectation Suite	Validation Results
diabetic_data	<ul style="list-style-type: none"> BasicDatasetProfiler-ProfilingResults 	BasicDatasetProfiler default	<ul style="list-style-type: none"> 2019-08-07T214759.510308Z 2019-08-07T214815.225802Z profiling 2019-08-07T214430.260260Z 2019-08-07T214413.168543Z
diabetic_data_copy	<ul style="list-style-type: none"> BasicDatasetProfiler-ProfilingResults 	BasicDatasetProfiler	<ul style="list-style-type: none"> profiling
IDs_mapping	<ul style="list-style-type: none"> BasicDatasetProfiler-ProfilingResults 	BasicDatasetProfiler	<ul style="list-style-type: none"> profiling



Documentation autogenerated using [Great Expectations](#).

This is a beta feature! Expect changes in API, behavior, and design.

The behavior of a site is controlled by configuration in the DataContext’s `great_expectations.yml` file.

Users can specify

- which datasources to document (by default, all)
- whether to include expectations, validations and profiling results sections
- where the expectations and validations should be read from (filesystem, S3, or GCS)
- where the HTML files should be written (filesystem, S3, or GCS)
- which renderer and view class should be used to render each section

Customizing HTML Documentation

The HTML documentation generated by Great Expectations Data Docs is fully customizable. If you would like to customize the look and feel of these pages or create your own, see [Customizing Data Docs](#).

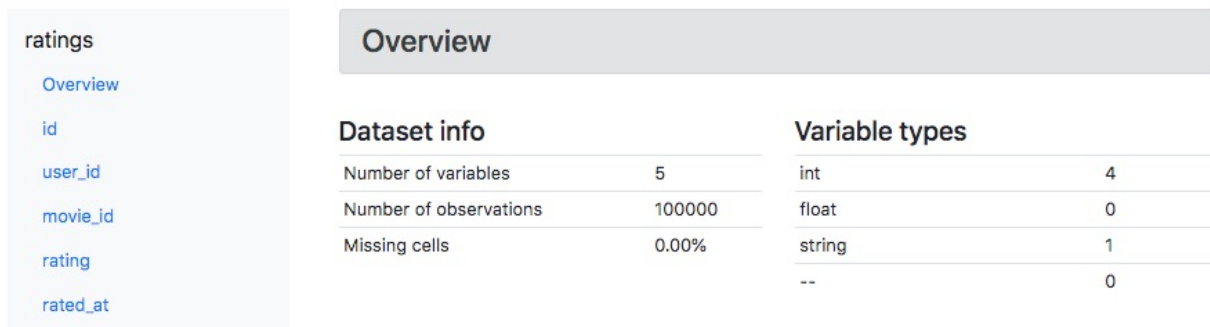
See the `data_docs_reference` for more information.

Profiling

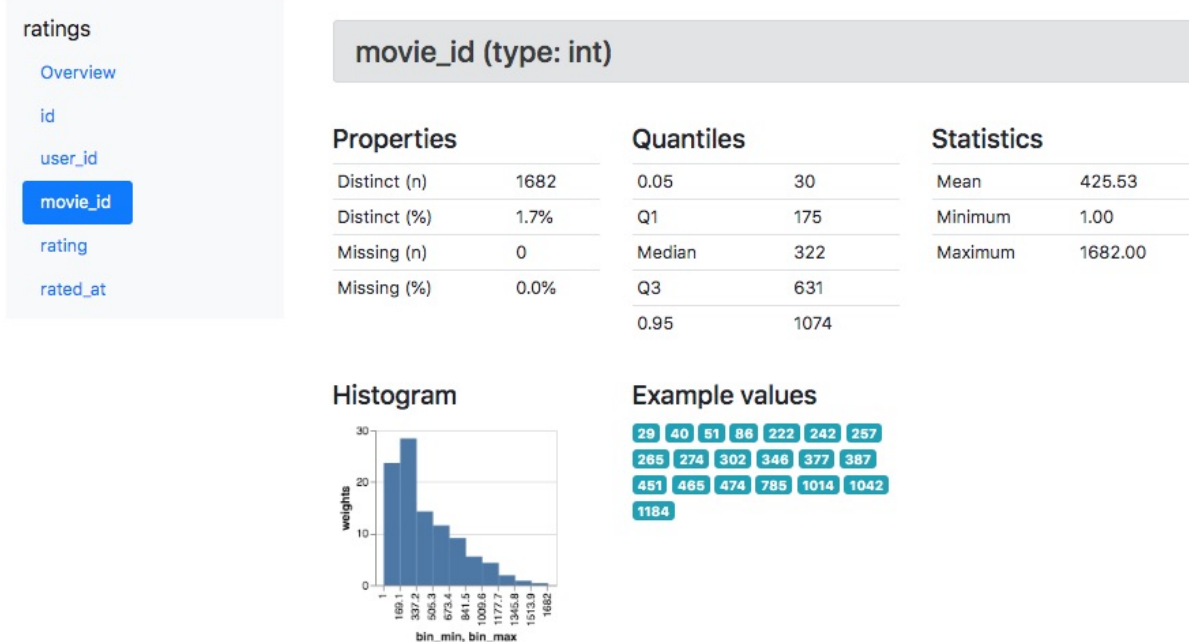
Profiling is a way of Rendering Validation Results to produce a summary of observed characteristics. When Validation Results are rendered as Profiling data, they create a new section in *Data Docs*. By computing the **observed** properties of data, Profiling helps to understand and reason about the data's **expected** properties.

To produce a useful data overview, Great Expectations uses a *profiler* to build a special Expectation Suite. Unlike the Expectations that are typically used for data validation, expectations for Profiling do not necessarily apply any constraints. They can simply identify statistics or other data characteristics that should be evaluated and made available in Great Expectations. For example, when the included `BasicDatasetProfiler` encounters a numeric column, it will add an `expect_column_mean_to_be_between` expectation but choose the `min_value` and `max_value` to both be `None`: essentially only saying that it expects a mean to exist.

The default `BasicDatasetProfiler` will thus produce a page for each table or `DataFrame` including an overview section:



And then detailed statistics for each column:



Profiling is still a beta feature in Great Expectations. Over time, we plan to extend and improve the `BasicDatasetProfiler` and also add additional profilers.

Warning: `BasicDatasetProfiler` will evaluate the entire batch without limits or sampling, which may be very time consuming. As a rule of thumb, we recommend starting with small batches of data.

See the `profiling_reference` for more information.

4.2.6 Profiling

Profiling helps you understand your data by describing it and even building expectation suites based on previous batches of data. Profiling lets you ask:

- What is this dataset like?

A **Profiler** reviews data assets and produces new Metrics, Expectation Suites, and Expectation Suite Validation Results that describe the data. A profiler can create a “stub” of high-level expectations based on what it sees in the data. Profilers can also be extended to create more specific expectations based on team conventions or statistical properties. Finally, Profilers can take advantage of metrics produced by Great Expectations when validating data to create useful overviews of data.

Profilers

Great Expectations provides a mechanism to automatically generate expectations, using a feature called a *Profiler*. A Profiler builds an Expectation Suite from one or more Data Assets. It usually also validates the data against the newly-generated Expectation Suite to return a Validation Result. There are several Profilers included with Great Expectations.

A Profiler makes it possible to quickly create a starting point for generating expectations about a Dataset. For example, during the *init* flow, Great Expectations uses the *BasicSuiteBuilderProfiler* to demonstrate important features of Expectations by creating and validating an Expectation Suite that has several different kinds of expectations built from a small sample of data. A Profiler is also critical to generating the Expectation Suites used during *Profiling*.

You can also extend Profilers to capture organizational knowledge about your data. For example, a team might have a convention that all columns **named** “id” are primary keys, whereas all columns ending with the **suffix** “_id” are foreign keys. In that case, when the team using Great Expectations first encounters a new dataset that followed the convention, a Profiler could use that knowledge to add an `expect_column_values_to_be_unique` Expectation to the “id” column (but not, for example an “address_id” column).

4.3 Supporting resources

Great Expectations requires a python compute environment and access to data, either locally or through a database or distributed cluster. In addition, developing with great expectations relies heavily on tools in the Python engineering ecosystem: pip, virtual environments, and jupyter notebooks. We also assume some level of familiarity with git and version control.

See the links below for good, practical tutorials for these tools.

4.3.1 pip

- <https://pip.pypa.io/en/stable/>
- <https://www.datacamp.com/community/tutorials/pip-python-package-manager>

4.3.2 virtual environments

- <https://virtualenv.pypa.io/en/latest/>
- <https://python-guide-cn.readthedocs.io/en/latest/dev/virtualenvs.html>
- <https://www.dabapps.com/blog/introduction-to-pip-and-virtualenv-python/>

4.3.3 jupyter notebooks and jupyter lab

- <https://jupyter.org/>
- <https://jupyterlab.readthedocs.io/en/stable/>
- <https://towardsdatascience.com/jupyter-lab-evolution-of-the-jupyter-notebook-5297cacde6b>

4.3.4 git

- <https://git-scm.com/>
- <https://reference.github.com/>
- <https://www.atlassian.com/git/tutorials>

4.4 Changelog

4.4.1 0.11.5

- [FEATURE] Add support for `expect_column_values_to_match_regex_list` exception for Spark backend
- [ENHANCEMENT] Added 3 new usage stats events: “cli.new_ds_choice”, “data_context.add_datasource”, and “datasource.sqlalchemy.connect”
- [ENHANCEMENT] Support `platform_specific_separator` flag for `TupleS3StoreBackend` prefix
- [ENHANCEMENT] Allow environment substitution in `config_variables.yml`
- [BUGFIX] fixed issue where calling `head()` on a `SqlAlchemyDataset` would fail if the underlying table is empty
- [BUGFIX] fixed bug in rounding of mostly argument to nullity expectations produced by the `BasicSuiteBuilder-Profiler`
- [DOCS] New How-to guide: How to add a Validation Operator (+ updated in Validation Operator doc strings)

4.4.2 0.11.4

- [BUGFIX] Fixed an error that crashed the CLI when called in an environment with neither `SQLAlchemy` nor `google.auth` installed

4.4.3 0.11.3

- [ENHANCEMENT] Removed the misleading scary “Site doesn’t exist or is inaccessible” message that the CLI displayed before building Data Docs for the first time.
- [ENHANCEMENT] Catch `sqlalchemy.exc.ArgumentError` and `google.auth.exceptions.GoogleAuthError` in `SqlAlchemyDatasource __init__` and re-raise them as `DatasourceInitializationError` - this allows the CLI to execute its retry logic when users provide a malformed SQLAlchemy URL or attempt to connect to a BigQuery project without having proper authentication.
- [BUGFIX] Fixed issue where the URL of the Glossary of Expectations article in the auto-generated suite edit notebook was wrong (out of date) (#1557).
- [BUGFIX] Use `renderer_type` to set paths in jinja templates instead of `utm_medium` since `utm_medium` is optional
- [ENHANCEMENT] Bring in `custom_views_directory` in `DefaultJinjaView` to enable custom jinja templates stored in `plugins` dir
- [BUGFIX] fixed glossary links in walkthrough modal, README, CTA button, scaffold notebook
- [BUGFIX] Improved `TupleGCSSStoreBackend` configurability (#1398 #1399)
- [BUGFIX] Data Docs: switch `bootstrap-table-filter-control.min.js` to CDN
- [ENHANCEMENT] `BasicSuiteBuilderProfiler` now rounds mostly values for readability
- [DOCS] Add AutoAPI as the primary source for API Reference docs.

4.4.4 0.11.2

- [FEATURE] Add support for `expect_volumn_values_to_match_json_schema` exception for Spark backend (thanks @chipmyersjr!)
- [ENHANCEMENT] Add formatted `__repr__` for `ValidationOperatorResult`
- [ENHANCEMENT] add option to suppress logging when getting expectation suite
- [BUGFIX] Fix object name construction when calling `SqlAlchemyDataset.head` (thanks @mascab!)
- [BUGFIX] Fixed bug where evaluation parameters used in arithmetic expressions would not be identified as upstream dependencies.
- [BUGFIX] Fix issue where `DatabaseStoreBackend` threw `IntegrityError` when storing same metric twice
- [FEATURE] Added new cli upgrade helper to help facilitate upgrading projects to be compatible with GE 0.11. See [Upgrading to 0.11.x](#) for more info.
- [BUGFIX] Fixed bug preventing GCS Data Docs sites to cleaned
- [BUGFIX] Correct doc link in checkpoint yaml
- [BUGFIX] Fixed issue where CLI checkpoint list truncated names (#1518)
- [BUGFIX] Fix S3 Batch Kwarg Generator incorrect migration to new `build_batch_kwarg` API
- [BUGFIX] Fix missing images in data docs walkthrough modal
- [BUGFIX] Fix bug in checkpoints that was causing incorrect `run_time` to be set
- [BUGFIX] Fix issue where data docs could remove trailing zeros from values when low precision was requested

4.4.5 0.11.1

- [BUGFIX] Fixed bug that was caused by comparison between timezone aware and non-aware datetimes
- [DOCS] Updated docs with info on typed run ids and validation operator results
- [BUGFIX] Update call-to-action buttons on index page with correct URLs

4.4.6 0.11.0

- [BREAKING] `run_id` is now typed using the new `RunIdentifier` class, which consists of a `run_time` and `run_name`. Existing projects that have Expectation Suite Validation Results must be migrated. See [Upgrading to 0.11.x](#) for instructions.
- [BREAKING] `ValidationMetric` and `ValidationMetricIdentifier` objects now have a `data_asset_name` attribute. Existing projects with evaluation parameter stores that have database backends must be migrated. See [Upgrading to 0.11.x](#) for instructions.
- [BREAKING] `ValidationOperator.run` now returns an instance of new type, `ValidationOperatorResult` (instead of a dictionary). If your code uses output from Validation Operators, it must be updated.
- Major update to the styling and organization of documentation! Watch for more content and reorganization as we continue to improve the documentation experience with Great Expectations.
- [FEATURE] Data Docs: redesigned index page with paginated/sortable/searchable/filterable tables
- [FEATURE] Data Docs: searchable tables on Expectation Suite Validation Result pages
- `data_asset_name` is now added to `batch_kwargs` by `batch_kwargs_generators` (if available) and surfaced in Data Docs
- Renamed all `generator_asset` parameters to `data_asset_name`
- Updated the `dateutil` dependency
- Added experimental `QueryStore`
- Removed deprecated `cli tap` command
- Added of 0.11 upgrade helper
- Corrected Scaffold maturity language in notebook to Experimental
- Updated the installation/configuration documentation for Snowflake users
- [ENHANCEMENT] Improved error messages for misconfigured checkpoints.
- [BUGFIX] Fixed bug that could cause some substituted variables in `DataContext` config to be saved to `great_expectations.yml`

4.4.7 0.10.12

- [DOCS] Improved help for CLI *checkpoint* command
- [BUGFIX] BasicSuiteBuilderProfiler could include extra expectations when only some expectations were selected (#1422)
- [FEATURE] add support for *expect_multicolumn_values_to_be_unique* and *expect_column_pair_values_A_to_be_greater_than_B* to Spark. Thanks @WilliamWsyHK!
- [ENHANCEMENT] Allow a dictionary of variables can be passed to the DataContext constructor to allow override config variables at runtime. Thanks @balexander!
- [FEATURE] add support for *expect_column_pair_values_A_to_be_greater_than_B* to Spark.
- [BUGFIX] Remove SQLAlchemy typehints to avoid requiring library (thanks @mzjp2)!
- [BUGFIX] Fix issue where quantile boundaries could not be set to zero. Thanks @kokes!

4.4.8 0.10.11

- Bugfix: build_data_docs list_keys for GCS returns keys and when empty a more user friendly message
- ENHANCEMENT: Enable Redshift Quantile Profiling

4.4.9 0.10.10

- Removed out-of-date Airflow integration examples. This repo provides a comprehensive example of Airflow integration: [#GE Airflow Example](#)
- Bugfix suite scaffold notebook now has correct suite name in first markdown cell.
- Bugfix: fixed an example in the custom expectations documentation article - “result” key was missing in the returned dictionary
- Data Docs Bugfix: template string substitution is now done using `.safe_substitute()`, to handle cases where string templates or substitution params have extraneous \$ signs. Also added logic to handle templates where intended output has groupings of 2 or more \$ signs
- Docs fix: fix in yml for example `action_list_operator` for metrics
- GE is now auto-linted using Black

-
- DataContext.get_docs_sites_urls now raises error if non-existent site_name is specified
 - Bugfix for the CLI command *docs build* ignoring the `--site_name` argument (#1378)
 - Bugfix and refactor for *datasource delete* CLI command (#1386) @mzjp2
 - Instantiate datasources and validate config only when datasource is used (#1374) @mzjp2
 - suite delete changed from an optional argument to a required one
 - bugfix for uploading objects to GCP #1393
 - added a new usage stats event for the case when a data context is created through CLI
 - tuplefilestore backend, expectationstore backend remove_key bugs fixed
 - no url is returned on empty data_docs site
 - return url for resource only if key exists

- Test added for the period special char case
- updated checkpoint module to not require sqlalchemy
- added BigQuery as an option in the list of databases in the CLI
- added special cases for handling BigQuery - table names are already qualified with schema name, so we must make sure that we do not prepend the schema name twice
- changed the prompt for the name of the temp table in BigQuery in the CLI to hint that a fully qualified name (project.dataset.table) should be provided
- Bugfix for: `expect_column_quantile_values_to_be_between` expectation throws an “unexpected keyword WITHIN” on BigQuery (#1391)

4.4.10 0.10.8

- added support for overriding the default jupyter command via a `GE_JUPYTER_COMMAND` environment variable (#1347) @nehiljain
- Bugfix for checkpoint missing template (#1379)

4.4.11 0.10.7

- crud delete suite bug fix

4.4.12 0.10.6

- Checkpoints: a new feature to ease deployment of suites into your pipelines - `DataContext.list_checkpoints()` returns a list of checkpoint names found in the project - `DataContext.get_checkpoint()` returns a validated dictionary loaded from yml - new cli commands
 - *checkpoint new*
 - *checkpoint list*
 - *checkpoint run*
 - *checkpoint script*
- marked cli *tap* commands as deprecating on next release
- marked cli *validation-operator run* command as deprecating
- internal improvements in the cli code
- Improve UpdateDataDocsAction docs

4.4.13 0.10.5

- improvements to `ge.read_json` tests
- tidy up the changelog
 - Fix bullet list spacing issues
 - Fix 0.10. formatting
 - Drop `roadmap_and_changelog.rst` and move `changelog.rst` to the top level of the table of contents
- `DataContext.run_validation_operator()` now raises a `DataContextError` if: - no batches are passed - batches are of the wrong type - no matching validation operator is found in the project
- Clarified scaffolding language in scaffold notebook
- `DataContext.create()` adds an additional directory: *checkpoints*
- Marked `tap` command for deprecation in next major release

4.4.14 0.10.4

- consolidated error handling in CLI `DataContext` loading
- new cli command *suite scaffold* to speed up creation of suites
- new cli command *suite demo* that creates an example suite
- Update `bigquery.rst` [#1330](#)
- Fix datetime reference in `create_expectations.rst` [#1321](#) Thanks @jschendel !
- Update issue templates
- CLI command experimental decorator
- Update `style_guide.rst`
- Add pull request template
- Use pickle to generate hash for dataframes with unhashable objects. [#1315](#) Thanks @shahinism !
- Unpin `pytest`

4.4.15 0.10.3

- Use pickle to generate hash for dataframes with unhashable objects.

4.4.16 0.10.2

- renamed `NotebookRenderer` to `SuiteEditNotebookRenderer`
- `SuiteEditNotebookRenderer` now lints using `black`
- New `SuiteScaffoldNotebookRenderer` renderer to expedite suite creation
- removed `autopep8` dependency
- bugfix: extra backslash in S3 urls if store was configured without a prefix [#1314](#)

4.4.17 0.10.1

- removing bootstrap scrollspy on table of contents [#1282](#)
- Silently tolerate connection timeout during usage stats reporting

4.4.18 0.10.0

- (BREAKING) Clarified API language: renamed all `generator` parameters and methods to the more correct `batch_kwargs_generator` language. Existing projects may require simple migration steps. See [Upgrading to 0.10.x](#) for instructions.
- Adds anonymized usage statistics to Great Expectations. See this article for details: [Usage Statistics](#).
- CLI: improve look/consistency of `docs list`, `suite list`, and `datasource list` output; add `store list` and `validation-operator list` commands.
- New SuiteBuilderProfiler that facilitates faster suite generation by allowing columns to be profiled
- Added two convenience methods to ExpectationSuite: `get_table_expectations` & `get_column_expectations`
- Added optional `profiler_configuration` to `DataContext.profile()` and `DataAsset.profile()`
- Added `list_available_expectation_types()` to DataAsset

4.4.19 0.9.11

- Add evaluation parameters support in `WarningAndFailureExpectationSuitesValidationOperator` [#1284](#) thanks [@balexander](#)
- Fix compatibility with MS SQL Server. [#1269](#) thanks [@kepiej](#)
- Bug fixes for `query_generator` [#1292](#) thanks [@ian-whitestone](#)

4.4.20 0.9.10

- Data Docs: improve configurability of `site_section_builders`
- `TupleFilesystemStoreBackend` now ignore `.ipynb_checkpoints` directories [#1203](#)
- bugfix for Data Docs links encoding on S3 [#1235](#)

4.4.21 0.9.9

- Allow evaluation parameters support in `run_validation_operator`
- Add `log_level` parameter to `jupyter-ux.setup_notebook_logging`.
- Add experimental `display_profiled_column_evrs_as_section` and `display_column_evrs_as_section` methods, with a minor (nonbreaking) refactor to create a new `_render_for_jupyter` method.
- Allow selection of site in `UpdateDataDocsAction` with new arg `target_site_names` in `great_expectations.yml`
- Fix issue with regular expression support in `BigQuery` ([#1244](#))

4.4.22 0.9.8

- Allow basic operations in evaluation parameters, with or without evaluation parameters.
- When unexpected exceptions occur (e.g., during data docs rendering), the user will see detailed error messages, providing information about the specific issue as well as the stack trace.
- Remove the “project new” option from the command line (since it is not implemented; users can only run “init” to create a new project).
- Update type detection for bigquery based on driver changes in pybigquery driver 0.4.14. Added a warning for users who are running an older pybigquery driver
- added execution tests to the NotebookRenderer to mitigate codegen risks
- Add option “persist”, true by default, for SparkDFDataset to persist the DataFrame it is passed. This addresses #1133 in a deeper way (thanks @tejsvirai for the robust debugging support and reproduction on spark).
 - Disabling this option should *only* be done if the user has *already* externally persisted the DataFrame, or if the dataset is too large to persist but *computations are guaranteed to be stable across jobs*.
- Enable passing dataset kwargs through datasource via dataset_options batch_kwarg.
- Fix AttributeError when validating expectations from a JSON file
- Data Docs: fix bug that was causing erratic scrolling behavior when table of contents contains many columns
- Data Docs: add ability to hide how-to buttons and related content in Data Docs

4.4.23 0.9.7

- Update marshmallow dependency to >3. NOTE: as of this release, you **MUST** use marshmallow >3.0, which **REQUIRES** python 3. (#1187) @jcampbell
 - Schema checking is now stricter for expectation suites, and data_asset_name must not be present as a top-level key in expectation suite json. It is safe to remove.
 - Similarly, datasource configuration must now adhere strictly to the required schema, including having any required credentials stored in the “credentials” dictionary.
- New beta CLI command: `tap new` that generates an executable python file to expedite deployments. (#1193) @Aylr
- bugfix in TableBatchKwargsGenerator docs
- Added feature maturity in README (#1203) @kyleaton
- Fix failing test that should skip if postgresql not running (#1199) @cicdw

4.4.24 0.9.6

- validate result dict when instantiating an ExpectationValidationResult (#1133)
- DataDocs: Expectation Suite name on Validation Result pages now link to Expectation Suite page
- `great_expectations init`: cli now asks user if csv has header when adding a Spark Datasource with csv file
- Improve support for using GCP Storage Bucket as a Data Docs Site backend (thanks @hammadzz)
- fix notebook renderer handling for expectations with no column kwarg and table not in their name (#1194)

4.4.25 0.9.5

- Fixed unexpected behavior with suite edit, data docs and jupyter
- pytest pinned to 5.3.5

4.4.26 0.9.4

- Update CLI *init* flow to support snowflake transient tables
- Use filename for default expectation suite name in CLI *init*
- Tables created by SQLAlchemyDataset use a shorter name with 8 hex characters of randomness instead of a full uuid
- Better error message when config substitution variable is missing
- removed an unused directory in the GE folder
- removed obsolete config error handling
- Docs typo fixes
- Jupyter notebook improvements
- *great_expectations init* improvements
- Simpler messaging in validation notebooks
- replaced hacky loop with suite list call in notebooks
- CLI suite new now supports *–empty* flag that generates an empty suite and opens a notebook
- add error handling to *init* flow for cases where user tries using a broken file

4.4.27 0.9.3

- Add support for transient table creation in snowflake (#1012)
- Improve path support in TupleStoreBackend for better cross-platform compatibility
- New features on *ExpectationSuite*
 - `add_citation()`
 - `get_citations()`
- *SampleExpectationsDatasetProfiler* now leaves a citation containing the original batch kwargs
- *great_expectations suite edit* now uses batch_kwargs from citations if they exist
- Bugfix :: suite edit notebooks no longer blow away the existing suite while loading a batch of data
- More robust and tested logic in *suite edit*
- DataDocs: bugfixes and improvements for smaller viewports
- Bugfix :: fix for bug that crashes SampleExpectationsDatasetProfiler if unexpected_percent is of type decimal.Decimal (#1109)

4.4.28 0.9.2

- Fixes #1095
- Added a `list_expectation_suites` function to `data_context`, and a corresponding CLI function - `suite list`.
- CI no longer enforces legacy python tests.

4.4.29 0.9.1

- Bugfix for dynamic “How to Edit This Expectation Suite” command in DataDocs

4.4.30 0.9.0

Version 0.9.0 is a major update to Great Expectations! The DataContext has continued to evolve into a powerful tool for ensuring that Expectation Suites can properly represent the way users think about their data, and upgrading will make it much easier to store and share expectation suites, and to build data docs that support your whole team. You’ll get awesome new features including improvements to data docs look and the ability to choose and store metrics for building flexible data quality dashboards.

The changes for version 0.9.0 fall into several broad areas:

1. Onboarding

Release 0.9.0 of Great Expectations makes it much easier to get started with the project. The `init` flow has grown to support a much wider array of use cases and to use more natural language rather than introducing GreatExpectations concepts earlier. You can more easily configure different backends and datasources, take advantage of guided walkthroughs to find and profile data, and share project configurations with colleagues.

If you have already completed the `init` flow using a previous version of Great Expectations, you do not need to rerun the command. However, **there are some small changes to your configuration that will be required**. See [Migrating between versions](#) for details.

2. CLI Command Improvements

With this release we have introduced a consistent naming pattern for accessing subcommands based on the noun (a Great Expectations object like `suite` or `docs`) and verb (an action like `edit` or `new`). The new user experience will allow us to more naturally organize access to CLI tools as new functionality is added.

3. Expectation Suite Naming and Namespace Changes

Defining shared expectation suites and validating data from different sources is much easier in this release. The DataContext, which manages storage and configuration of expectations, validations, profiling, and data docs, no longer requires that expectation suites live in a datasource-specific “namespace.” Instead, you should name suites with the logical name corresponding to your data, making it easy to share them or validate against different data sources. For example, the expectation suite “npi” for National Provider Identifier data can now be shared across teams who access the same logical data in local systems using Pandas, on a distributed Spark cluster, or via a relational database.

Batch Kwarg, or instructions for a datasource to build a batch of data, are similarly freed from a required namespace, and you can more easily integrate Great Expectations into workflows where you do not need to use a BatchKwarg-Generator (usually because you have a batch of data ready to validate, such as in a table or a known directory).

The most noticeable impact of this API change is in the complete removal of the `DataAssetIdentifier` class. For example, the `create_expectation_suite` and `get_batch` methods now no longer require a `data_asset_name` parameter, relying only on the `expectation_suite_name` and `batch_kwarg` to do their job. Similarly, there is no more asset name normalization required. See the upgrade guide for more information.

4. Metrics and Evaluation Parameter Stores

Metrics have received much more love in this release of Great Expectations! We've improved the system for declaring evaluation parameters that support dependencies between different expectation suites, so you can easily identify a particular field in the result of one expectation to use as the input into another. And the MetricsStore is now much more flexible, supporting a new `ValidationAction` that makes it possible to select metrics from a validation result to be saved in a database where they can power a dashboard.

5. Internal Type Changes and Improvements

Finally, in this release, we have done a lot of work under the hood to make things more robust, including updating all of the internal objects to be more strongly typed. That change, while largely invisible to end users, paves the way for some really exciting opportunities for extending Great Expectations as we build a bigger community around the project.

We are really excited about this release, and encourage you to upgrade right away to take advantage of the more flexible naming and simpler API for creating, accessing, and sharing your expectations. As always feel free to join us on Slack for questions you don't see addressed!

4.4.31 0.8.9__develop

4.4.32 0.8.8

- Add support for `allow_relative_error` to `expect_column_quantile_values_to_be_between`, allowing Redshift users access to this expectation
- Add support for checking backend type information for datetime columns using `expect_column_min_to_be_between` and `expect_column_max_to_be_between`

4.4.33 0.8.7

- Add support for `expect_column_values_to_be_of_type` for BigQuery backend (#940)
- Add image CDN for community usage stats
- Documentation improvements and fixes

4.4.34 0.8.6

- Raise informative error if config variables are declared but unavailable
- Update ExpectationsStore defaults to be consistent across all `FixedLengthTupleStoreBackend` objects
- Add support for setting `spark_options` via `SparkDFDatasource`
- Include `tail_weights` by default when using `build_continuous_partition_object`
- Fix Redshift quantiles computation and type detection
- Allow boto3 options to be configured (#887)

4.4.35 0.8.5

- **BREAKING CHANGE:** move all reader options from the top-level `batch_kwargs` object to a sub-dictionary called “reader_options” for `SparkDFDatasource` and `PandasDatasource`. This means it is no longer possible to specify supplemental reader-specific options at the top-level of `get_batch`, `yield_batch_kwargs` or `build_batch_kwargs` calls, and instead, you must explicitly specify that they are reader_options, e.g. by a call such as: `context.yield_batch_kwargs(data_asset_name, reader_options={'encoding': 'utf-8'})`.
- **BREAKING CHANGE:** move all query_params from the top-level `batch_kwargs` object to a sub-dictionary called “query_params” for `SqlAlchemyDatasource`. This means it is no longer possible to specify supplemental query_params at the top-level of `get_batch`, `yield_batch_kwargs` or `build_batch_kwargs` calls, and instead, you must explicitly specify that they are query_params, e.g. by a call such as: `context.yield_batch_kwargs(data_asset_name, query_params={'schema': 'foo'})`.
- Add support for filtering validation result suites and validation result pages to show only failed expectations in generated documentation
- Add support for limit parameter to `batch_kwargs` for all datasources: `Pandas`, `SqlAlchemy`, and `SparkDF`; add support to generators to support building `batch_kwargs` with limits specified.
- Include `raw_query` and `query_params` in `query_generator batch_kwargs`
- Rename generator keyword arguments from `data_asset_name` to `generator_asset` to avoid ambiguity with normalized names
- Consistently migrate timestamp from `batch_kwargs` to `batch_id`
- Include `batch_id` in validation results
- Fix issue where `batch_id` was not included in some generated datasets
- Fix rendering issue with `expect_table_columns_to_match_ordered_list` expectation
- Add support for GCP, including `BigQuery` and `GCS`
- Add support to S3 generator for retrieving directories by specifying the `directory_assets` configuration
- Fix warning regarding implicit `class_name` during init flow
- Expose `build_generator` API publicly on datasources
- Allow configuration of known extensions and return more informative message when `SubdirReaderBatchKwargsGenerator` cannot find relevant files.
- Add support for `allow_relative_error` on internal dataset quantile functions, and add support for `build_continuous_partition_object` in `Redshift`
- Fix truncated scroll bars in `value_counts` graphs

4.4.36 0.8.4.post0

- Correct a packaging issue resulting in missing notebooks in tarball release; update docs to reflect new notebook locations.

4.4.37 0.8.4

- Improved the tutorials that walk new users through the process of creating expectations and validating data
- Changed the flow of the init command - now it creates the scaffolding of the project and adds a datasource. After that users can choose their path.
- Added a component with links to useful tutorials to the index page of the Data Docs website
- Improved the UX of adding a SQL datasource in the CLI - now the CLI asks for specific credentials for Postgres, MySQL, Redshift and Snowflake, allows continuing debugging in the config file and has better error messages
- Added batch_kwargs information to DataDocs validation results
- Fix an issue affecting file stores on Windows

4.4.38 0.8.3

- Fix a bug in data-docs' rendering of mostly parameter
- Correct wording for expect_column_proportion_of_unique_values_to_be_between
- Set charset and meta tags to avoid unicode decode error in some browser/backend configurations
- Improve formatting of empirical histograms in validation result data docs
- Add support for using environment variables in *config_variables_file_path*
- Documentation improvements and corrections

4.4.39 0.8.2.post0

- Correct a packaging issue resulting in missing css files in tarball release

4.4.40 0.8.2

- Add easier support for customizing data-docs css
- Use higher precision for rendering 'mostly' parameter in data-docs; add more consistent locale-based formatting in data-docs
- Fix an issue causing visual overlap of large numbers of validation results in build-docs index
- Documentation fixes (thanks @DanielOliver!) and improvements
- Minor CLI wording fixes
- Improved handling of MySql temporary tables
- Improved detection of older config versions

4.4.41 0.8.1

- Fix an issue where version was reported as ‘0+unknown’

4.4.42 0.8.0

Version 0.8.0 is a significant update to Great Expectations, with many improvements focused on configurability and usability. See the [Migrating between versions](#) guide for more details on specific changes, which include several breaking changes to configs and APIs.

Highlights include:

1. Validation Operators and Actions. Validation operators make it easy to integrate GE into a variety of pipeline runners. They offer one-line integration that emphasizes configurability. See the [Validation Operators And Actions Introduction](#) feature guide for more information.
 - The DataContext `get_batch` method no longer treats `expectation_suite_name` or `batch_kwargs` as optional; they must be explicitly specified.
 - The top-level GE `validate` method allows more options for specifying the specific `data_asset` class to use.
2. First-class support for plugins in a DataContext, with several features that make it easier to configure and maintain DataContexts across common deployment patterns.
 - **Environments:** A DataContext can now manage [Managing Environment and Secrets](#) more easily thanks to more dynamic and flexible variable substitution.
 - **Stores:** A new internal abstraction for DataContexts, `stores_reference`, make extending GE easier by consolidating logic for reading and writing resources from a database, local, or cloud storage.
 - **Types:** Utilities configured in a DataContext are now referenced using `class_name` and `module_name` throughout the DataContext configuration, making it easier to extend or supplement pre-built resources. For now, the “type” parameter is still supported but expect it to be removed in a future release.
3. Partitioners: Batch Kwargs are clarified and enhanced to help easily reference well-known chunks of data using a `partition_id`. Batch ID and Batch Fingerprint help round out support for enhanced metadata around data assets that GE validates. See `batch_identifiers` for more information. The `GlobReaderBatchKwargsGenerator`, `QueryBatchKwargsGenerator`, `S3GlobReaderBatchKwargsGenerator`, `SubdirReaderBatchKwargsGenerator`, and `TableBatchKwargsGenerator` all support `partition_id` for easily accessing data assets.
4. Other Improvements:
 - We’re beginning a long process of some under-the-covers refactors designed to make GE more maintainable as we begin adding additional features.
 - Restructured documentation: our docs have a new structure and have been reorganized to provide space for more easily adding and accessing reference material. Stay tuned for additional detail.
 - The command `build-documentation` has been renamed `build-docs` and now by default opens the Data Docs in the users’ browser.

4.4.43 v0.7.11

- Fix an issue where `head()` lost the column name for `SqlAlchemyDataset` objects with a single column
- Fix logic for the 'auto' bin selection of *build_continuous_partition_object*
- Add missing `jinja2` dependency
- Fix an issue with inconsistent availability of `strict_min` and `strict_max` options on `expect_column_values_to_be_between`
- Fix an issue where expectation suite evaluation parameters could be overridden by values during validate operation

4.4.44 v0.7.10

- Fix an issue in generated documentation where the Home button failed to return to the index
- Add S3 Generator to module docs and improve module docs formatting
- Add support for views to `QueryBatchKwargsGenerator`
- Add success/failure icons to index page
- Return to uniform histogram creation during profiling to avoid large partitions for internal performance reasons

4.4.45 v0.7.9

- Add an S3 generator, which will introspect a configured bucket and generate `batch_kwargs` from identified objects
- Add support to `PandasDatasource` and `SparkDFDatasource` for reading directly from S3
- Enhance the Site Index page in documentation so that validation results are sorted and display the newest items first when using the default run-id scheme
- Add a new utility method, *build_continuous_partition_object* which will build partition objects using the dataset API and so supports any GE backend.
- Fix an issue where columns with spaces in their names caused failures in some `SqlAlchemyDataset` and `SparkDFDataset` expectations
- Fix an issue where generated queries including null checks failed on MSSQL (#695)
- Fix an issue where evaluation parameters passed in as a set instead of a list could cause JSON serialization problems for the result object (#699)

4.4.46 v0.7.8

- BREAKING: slack webhook URL now must be in the `profiles.yml` file (treat as a secret)
- Profiler improvements:
 - Display candidate profiling data assets in alphabetical order
 - Add columns to the `expectation_suite` meta during profiling to support human-readable description information
- Improve handling of optional dependencies during CLI init
- Improve documentation for `create_expectations` notebook

- Fix several anachronistic documentation and docstring phrases (#659, #660, #668, #681; #thanks @StevenM-Mortimer)
- Fix data docs rendering issues:
 - documentation rendering failure from unrecognized profiled column type (#679; thanks @dinedal))
 - PY2 failure on encountering unicode (#676)

4.4.47 0.7.7

- Standardize the way that plugin module loading works. DataContext will begin to use the new-style class and plugin identification moving forward; yml configs should specify class_name and module_name (with module_name optional for GE types). For now, it is possible to use the “type” parameter in configuration (as before).
- Add support for custom data_asset_type to all datasources
- Add support for strict_min and strict_max to inequality-based expectations to allow strict inequality checks (thanks @RoyalTS!)
- Add support for reader_method = “delta” to SparkDFDatasource
- Fix databricks generator (thanks @sspitz3!)
- Improve performance of DataContext loading by moving optional import
- Fix several memory and performance issues in SparkDFDataset.
 - Use only distinct value count instead of bringing values to driver
 - Migrate away from UDF for set membership, nullity, and regex expectations
- Fix several UI issues in the data_documentation
 - Move prescriptive dataset expectations to Overview section
 - Fix broken link on Home breadcrumb
 - Scroll follows navigation properly
 - Improved flow for long items in value_set
 - Improved testing for ValidationRenderer
 - Clarify dependencies introduced in documentation sites
 - Improve testing and documentation for site_builder, including run_id filter
 - Fix missing header in Index page and cut-off tooltip
 - Add run_id to path for validation files

4.4.48 0.7.6

- New Validation Renderer! Supports turning validation results into HTML and displays differences between the expected and the observed attributes of a dataset.
- Data Documentation sites are now fully configurable; a data context can be configured to generate multiple sites built with different GE objects to support a variety of data documentation use cases. See data documentation guide for more detail.
- CLI now has a new top-level command, *build-documentation* that can support rendering documentation for specified sites and even named data assets in a specific site.
- Introduced DotDict and LooselyTypedDotDict classes that allow to enforce typing of dictionaries.
- Bug fixes: improved internal logic of rendering data documentation, slack notification, and CLI profile command when datasource argument was not provided.

4.4.49 0.7.5

- Fix missing requirement for py pandoc brought in from markdown support for notes rendering.

4.4.50 0.7.4

- Fix numerous rendering bugs and formatting issues for rendering documentation.
- Add support for pandas extension dtypes in pandas backend of `expect_column_values_to_be_of_type` and `expect_column_values_to_be_in_type_list` and fix bug affecting some dtype-based checks.
- Add datetime and boolean column-type detection in BasicDatasetProfiler.
- Improve BasicDatasetProfiler performance by disabling interactive evaluation when output of expectation is not immediately used for determining next expectations in profile.
- Add support for rendering `expectation_suite` and `expectation_level` notes from meta in docs.
- Fix minor formatting issue in readthedocs documentation.

4.4.51 0.7.3

- **BREAKING:** Harmonize `expect_column_values_to_be_of_type` and `expect_column_values_to_be_in_type_list` semantics in Pandas with other backends, including support for None type and `type_list` parameters to support profiling. *These type expectations now rely exclusively on native python or numpy type names.*
- Add configurable support for Custom DataAsset modules to DataContext
- Improve support for setting and inheriting custom `data_asset_type` names
- Add tooltips with expectations backing data elements to rendered documentation
- Allow better selective disabling of tests (thanks @RoyalITS)
- Fix documentation build errors causing missing code blocks on readthedocs
- Update the parameter naming system in DataContext to reflect `data_asset_name` and `expectation_suite_name`
- Change scary warning about discarding expectations to be clearer, less scary, and only in log
- Improve profiler support for boolean types, `value_counts`, and type detection

- Allow user to specify data_assets to profile via CLI
- Support CLI rendering of expectation_suite and EVR-based documentation

4.4.52 0.7.2

- Improved error detection and handling in CLI “add datasource” feature
- Fixes in rendering of profiling results (descriptive renderer of validation results)
- Query Generator of SQLAlchemy datasource adds tables in non-default schemas to the data asset namespace
- Added convenience methods to display HTML renderers of sections in Jupyter notebooks
- Implemented prescriptive rendering of expectations for most expectation types

4.4.53 0.7.1

- Added documentation/tutorials/videos for onboarding and new profiling and documentation features
- Added prescriptive documentation built from expectation suites
- Improved index, layout, and navigation of data context HTML documentation site
- Bug fix: non-Python files were not included in the package
- Improved the rendering logic to gracefully deal with failed expectations
- Improved the basic dataset profiler to be more resilient
- Implement `expect_column_values_to_be_of_type`, `expect_column_values_to_be_in_type_list` for SparkDF-Dataset
- Updated CLI with a new documentation command and improved profile and render commands
- Expectation suites and validation results within a data context are saved in a more readable form (with indentation)
- Improved compatibility between SparkDatasource and InMemoryGenerator
- Optimization for Pandas column type checking
- Optimization for Spark duplicate value expectation (thanks @orenovadia!)
- Default `run_id` format no longer includes “:” and specifies UTC time
- Other internal improvements and bug fixes

4.4.54 0.7.0

Version 0.7 of Great Expectations is HUGE. It introduces several major new features and a large number of improvements, including breaking API changes.

The core vocabulary of expectations remains consistent. Upgrading to the new version of GE will primarily require changes to code that uses data contexts; existing expectation suites will require only changes to top-level names.

- Major update of Data Contexts. Data Contexts now offer significantly more support for building and maintaining expectation suites and interacting with existing pipeline systems, including providing a namespace for objects. They can handle integrating, registering, and storing validation results, and provide a namespace for data assets, making **batches** first-class citizens in GE. Read more: [DataContexts](#) or `great_expectations.data_context`

- Major refactor of `autoinspect`. `Autoinspect` is now built around a module called “profile” which provides a class-based structure for building expectation suites. There is no longer a default “`autoinspect_func`” – calling `autoinspect` requires explicitly passing the desired profiler. See [Profiling](#)
- New “Compile to Docs” feature produces beautiful documentation from expectations and expectation validation reports, helping keep teams on the same page.
- Name clarifications: we’ve stopped using the overloaded terms “expectations config” and “config” and instead use “expectation suite” to refer to a collection (or suite!) of expectations that can be used for validating a data asset.
 - Expectation Suites include several top level keys that are useful for organizing content in a data context: `data_asset_name`, `expectation_suite_name`, and `data_asset_type`. When a `data_asset` is validated, those keys will be placed in the *meta* key of the validation result.
- Major enhancement to the CLI tool including *init*, *render* and more flexibility with *validate*
- Added helper notebooks to make it easy to get started. Each notebook acts as a combination of tutorial and code scaffolding, to help you quickly learn best practices by applying them to your own data.
- Relaxed constraints on expectation parameter values, making it possible to declare many column aggregate expectations in a way that is always “vacuously” true, such as `expect_column_values_to_be_between None and None`. This makes it possible to progressively tighten expectations while using them as the basis for profiling results and documentation.
- Enabled caching on dataset objects by default.
- Bugfixes and improvements:
 - New expectations:
 - * `expect_column_quantile_values_to_be_between`
 - * `expect_column_distinct_values_to_be_in_set`
 - Added support for `head` method on all current backends, returning a `PandasDataset`
 - More implemented expectations for `SparkDF Dataset` with optimizations
 - * `expect_column_values_to_be_between`
 - * `expect_column_median_to_be_between`
 - * `expect_column_value_lengths_to_be_between`
 - Optimized histogram fetching for `SqlalchemyDataset` and `SparkDFDataset`
 - Added cross-platform internal partition method, paving path for improved profiling
 - Fixed bug with `outputstrftime` not being honored in `PandasDataset`
 - Fixed series naming for column value counts
 - Standardized naming for `expect_column_values_to_be_of_type`
 - Standardized and made explicit use of sample normalization in `stdev` calculation
 - Added `from_dataset` helper
 - Internal testing improvements
 - Documentation reorganization and improvements
 - Introduce custom exceptions for more detailed error logs

4.4.55 0.6.1

- Re-add testing (and support) for py2
- NOTE: Support for SQLAlchemyDataset and SparkDFDataset is enabled via optional install (e.g. `pip install great_expectations[sqlalchemy]` or `pip install great_expectations[spark]`)

4.4.56 0.6.0

- Add support for SparkDFDataset and caching (HUGE work from @cselig)
- Migrate distributional expectations to new testing framework
- Add support for two new expectations: `expect_column_distinct_values_to_contain_set` and `expect_column_distinct_values_to_equal_set` (thanks @RoyalTS)
- FUTURE BREAKING CHANGE: The new cache mechanism for Datasets, when enabled, causes GE to assume that dataset does not change between evaluation of individual expectations. We anticipate this will become the future default behavior.
- BREAKING CHANGE: Drop official support pandas < 0.22

4.4.57 0.5.1

- **Fix** issue where no `result_format` available for `expect_column_values_to_be_null` caused error
- Use vectorized computation in pandas (#443, #445; thanks @RoyalTS)

4.4.58 0.5.0

- Restructured class hierarchy to have a more generic DataAsset parent that maintains expectation logic separate from the tabular organization of Dataset expectations
- Added new FileDataAsset and associated expectations (#416 thanks @anhollis)
- Added support for date/datetime type columns in some SQLAlchemy expectations (#413)
- Added support for a multicolumn expectation, `expect_multicolumn_values_to_be_unique` (#408)
- **Optimization:** You can now disable *partial_unexpected_counts* by setting the *partial_unexpected_count* value to 0 in the `result_format` argument, and we do not compute it when it would not be returned. (#431, thanks @eugmandel)
- **Fix:** Correct error in `unexpected_percent` computations for sqlalchemy when unexpected values exceed limit (#424)
- **Fix:** Pass meta object to expectation result (#415, thanks @jseeman)
- Add support for multicolumn expectations, with *expect_multicolumn_values_to_be_unique* as an example (#406)
- Add dataset class to `from_pandas` to simplify using custom datasets (#404, thanks @jtilly)
- Add schema support for sqlalchemy data context (#410, thanks @rahulj51)
- Minor documentation, warning, and testing improvements (thanks @zdog).

4.4.59 0.4.5

- Add a new autoinspect API and remove default expectations.
- Improve details for `expect_table_columns_to_match_ordered_list` (#379, thanks @rlshuhart)
- Linting fixes (thanks @elsander)
- Add support for `dataset_class` in `from_pandas` (thanks @jtilly)
- Improve redshift compatibility by correcting faulty `isnull` operator (thanks @avanderm)
- Adjust partitions to use `tail_weight` to improve JSON compatibility and support special cases of KL Divergence (thanks @anhollis)
- Enable `custom_sql` datasets for databases with multiple schemas, by adding a fallback for column reflection (#387, thanks @elsander)
- Remove *IF NOT EXISTS* check for custom sql temporary tables, for Redshift compatibility (#372, thanks @elsander)
- Allow users to pass args/kwargs for engine creation in `SqlAlchemyDataContext` (#369, thanks @elsander)
- Add support for custom schema in `SqlAlchemyDataset` (#370, thanks @elsander)
- Use `getfullargspec` to avoid deprecation warnings.
- Add `expect_column_values_to_be_unique` to `SqlAlchemyDataset`
- **Fix** map expectations for categorical columns (thanks @eugmandel)
- Improve internal testing suite (thanks @anhollis and @ccnobbli)
- Consistently use `value_set` instead of mixing `value_set` and `values_set` (thanks @njsmith8)

4.4.60 0.4.4

- Improve CLI help and set CLI return value to the number of unmet expectations
- Add error handling for empty columns to `SqlAlchemyDataset`, and associated tests
- **Fix** broken support for older pandas versions (#346)
- **Fix** pandas deepcopy issue (#342)

4.4.61 0.4.3

- Improve type lists in `expect_column_type_to_be_in_list` (thanks @smontanaro and @ccnobbli)
- Update cli to use `entry_points` for conda compatibility, and add version option to cli
- Remove extraneous development dependency to airflow
- Address `SQLAlchemy` warnings in median computation
- Improve glossary in documentation
- Add ‘statistics’ section to validation report with overall validation results (thanks @sotte)
- Add support for parameterized expectations
- Improve support for custom expectations with better error messages (thanks @syk0saje)
- Implement `expect_column_value_lengths_to_be_between` for `SQLAlchemy` (thanks @ccnobbli)

- **Fix** PandasDataset subclasses to inherit child class

4.4.62 0.4.2

- **Fix** bugs in `expect_column_values_to_[not]_be_null`: computing unexpected value percentages and handling all-null (thanks @ccnobbli)
- Support mysql use of Decimal type (thanks @bouke-nederstigt)
- Add new expectation `expect_column_values_to_not_match_regex_list`.
 - Change behavior of `expect_column_values_to_match_regex_list` to use python `re.findall` in PandasDataset, relaxing matching of individuals expressions to allow matches anywhere in the string.
- **Fix** documentation errors and other small errors (thanks @roblim, @ccnobbli)

4.4.63 0.4.1

- Correct inclusion of new `data_context` module in source distribution

4.4.64 0.4.0

- Initial implementation of data context API and SQLAlchemyDataset including implementations of the following expectations:
 - `expect_column_to_exist`
 - `expect_table_row_count_to_be`
 - `expect_table_row_count_to_be_between`
 - `expect_column_values_to_not_be_null`
 - `expect_column_values_to_be_null`
 - `expect_column_values_to_be_in_set`
 - `expect_column_values_to_be_between`
 - `expect_column_mean_to_be`
 - `expect_column_min_to_be`
 - `expect_column_max_to_be`
 - `expect_column_sum_to_be`
 - `expect_column_unique_value_count_to_be_between`
 - `expect_column_proportion_of_unique_values_to_be_between`
- Major refactor of `output_format` to new `result_format` parameter. See docs for full details:
 - `exception_list` and related uses of the term `exception` have been renamed to `unexpected`
 - Output formats are explicitly hierarchical now, with `BOOLEAN_ONLY` < `BASIC` < `SUMMARY` < `COMPLETE`. All *column_aggregate_expectation* expectations now return element count and related information included at the `BASIC` level or higher.
- New expectation available for parameterized distributions—`expect_column_parameterized_distribution_ks_test_p_value_to_be_greater_than` (what a name! :) – (thanks @ccnobbli)

- `ge.from_pandas()` utility (thanks @schrockn)
- Pandas operations on a `PandasDataset` now return another `PandasDataset` (thanks @dlwhite5)
- `expect_column_to_exist` now takes a `column_index` parameter to specify column order (thanks @louispotok)
- Top-level validate option (`ge.validate()`)
- `ge.read_json()` helper (thanks @rjurney)
- Behind-the-scenes improvements to testing framework to ensure parity across data contexts.
- Documentation improvements, bug-fixes, and internal api improvements

4.4.65 0.3.2

- Include requirements file in source dist to support conda

4.4.66 0.3.1

- **Fix** infinite recursion error when building custom expectations
- Catch dateutil parsing overflow errors

4.4.67 0.2

- Distributional expectations and associated helpers are improved and renamed to be more clear regarding the tests they apply
- Expectation decorators have been refactored significantly to streamline implementing expectations and support custom expectations
- API and examples for custom expectations are available
- New output formats are available for all expectations
- Significant improvements to test suite and compatibility

4.5 Reference: Spare parts

These docs are leftover pieces of old documentation. They're potentially helpful, but may be incomplete, incorrect, or confusing.

Once each file is tidied up and rehabilitated, it can be moved to another section. Some of these article may turn into more than one guides elsewhere.

Warning: These docs are leftover pieces of old documentation. They're potentially helpful, but may be incomplete, incorrect, or confusing.

4.5.1 Doctest Examples

Use these examples during migration to testable docs.

The block below is not rendered in the final documentation, but *does* affect the namespace.

This block is a standard doctest block, with one statement.

```
>>> npi.expect_column_values_to_be_unique("provider_id") ==_
↳ ExpectationValidationResult(
...     **{
...         "result": {
...             "element_count": 3,
...             "missing_count": 0,
...             "missing_percent": 0.0,
...             "unexpected_count": 0,
...             "unexpected_percent": 0.0,
...             "unexpected_percent_nonmissing": 0.0,
...             "partial_unexpected_list": []
...         },
...         "success": True,
...         "exception_info": None,
...         "meta": {},
...         "expectation_config": ExpectationConfiguration(**{
...             "expectation_type": "expect_column_values_to_be_unique",
...             "meta": {},
...             "kwargs": {
...                 "column": "provider_id",
...                 "result_format": "BASIC"
...             }
...         })
...     })
True
```

This block is tested only in that it must not raise an exception. No output test happens from a code-block.

```
assert npi.expect_column_values_to_be_unique("provider_id") !=_
↳ ExpectationValidationResult(
    meta={},
    result={
        "element_count": 3,
        "missing_count": 0,
        "missing_percent": 0.0,
        "unexpected_count": 0,
        "unexpected_percent": 0.0,
        "unexpected_percent_nonmissing": 0.0,
        "partial_unexpected_list": []
    },
    success=True,
    expectation_config=ExpectationConfiguration(**{
        "expectation_type": "expect_column_values_to_be_unique",
        "meta": {},
        "kwargs": {
            "column": "provider_id",
            "result_format": "BASIC"
        }
    }),
    exception_info=None
)
```

These three lines will be evaluated as classic doctest:

```
>>> df = pd.read_csv("/opt/data/titanic/Titanic.csv")
>>> df = ge.dataset.PandasDataset(df)
>>> res = df.expect_column_values_to_be_in_set("Sex", ["male", "female"])
```

This section would often fail, but will be skipped because of the Sphinx comment. It **will** be rendered.

```
>>> print(res)
{
  "exception_info": null,
  "success": true,
  "meta": {},
  "result": {
    "element_count": 1313,
    "missing_count": 0,
    "missing_percent": 0.0,
    "unexpected_count": 0,
    "unexpected_percent": 0.0,
    "unexpected_percent_nonmissing": 0.0,
    "partial_unexpected_list": []
  },
  "expectation_config": {
    "expectation_type": "expect_column_values_to_be_in_set",
    "meta": {},
    "kwargs": {
      "column": "Sex",
      "value_set": [
        "male",
        "female"
      ],
      "result_format": "BASIC"
    }
  }
}
```

4.5.2 Using Great Expectations on Teams

Now that you've enjoyed single player mode, let's bring Great Expectations to your team.

When you move from single user to collaborative use there are a few major things to consider.

Major Considerations

Where Should Expectations Live?

If you followed our best practice recommendation of committing the `great_expectations` directory to your source control repository, then this question is already answered! Expectations live right in your repo!

It is also possible to store expectations on cloud storage providers—we recommend enabling versioning in that case. See the documentation on [customizing the data docs store backend](#) for more information, or follow the [tutorial](#).

Where Should Validations Live?

When using the default Validation Operators, Validations are stored in your `great_expectations/uncommitted/validations/` directory. Because these may include examples of data (which could be sensitive or regulated) these Validations **should not** be committed to a source control system.

You can configure a Store to write these to a cloud provider blob storage such as Amazon S3, Google Cloud Storage, or some other securely mounted file system. This will depend on your team's deployment patterns.

Where Should Data Docs Live?

Similar to Validations, Data Docs are by default stored in your `great_expectations/uncommitted/data_docs/` directory. Because these may include examples of data (which could be sensitive or regulated) these **should not** be committed to a source control system.

You can configure a store to write these to a cloud provider blob storage such as Amazon S3, Google Cloud Storage, or some other securely mounted file system.

See the [data docs reference](#) for more information on configuring data docs to use cloud storage, or follow the [tutorial](#) to configure a site now.

Where Should Notifications Go?

Some teams enjoy realtime data quality alerts in Slack. We find setting up a channel with an obvious name like `data-quality-notifications` a nice place to have Great Expectations post to.

How Do You On-board a New Teammate?

If you have a project in a repo that includes your `great_expectations` directory, onboarding is simple! Your teammates will need to:

1. Clone the repo.
2. Run `great_expectations init` to create any missing directories.
3. Add any secrets required by your datasources to the file `great_expectations/uncommitted/config_variables.yml`.

COMMUNITY RESOURCES

We're committed to supporting and growing the community around Great Expectations. It's not enough to build a great tool. We want to build a great community as well.

Open source doesn't always have the best reputation for being friendly and welcoming, and that makes us sad. Everyone belongs in open source, and Great Expectations is dedicated to making you feel welcome.

5.1 Get in touch with the Great Expectations team

Join our public slack channel here: [join slack](#). We're not always available, but we're there and responsive an awful lot of the time.

5.2 Ask a question

Slack is good for that, too: [join slack](#).

5.3 File a bug report or feature request

If you have bugfix or feature request, please [upvote an existing issue](#) or [open a new issue](#) on GitHub and we'll see what we can do.

5.4 Contribute code or documentation

We welcome contributions to Great Expectations. Please start with our [Contributing](#) guide and don't be shy with questions!

CONTRIBUTING

Table of contents for Contributing:

6.1 Setting up your dev environment

6.1.1 Prerequisites

In order to contribute to Great Expectations, you will need the following:

- A GitHub account—this is sufficient if you *only want to contribute to the documentation*.
- If you want to contribute code, you will also need a working version of Git on your computer. Please refer to the [Git setup instructions](#) for your environment.
- We also recommend going through the [SSH key setup process on GitHub](#) for easier authentication.

6.1.2 Fork and clone the repository

1. Fork the Great Expectations repo

- Go to the [Great Expectations repo on GitHub](#).
- Click the `Fork` button in the top right. This will make a copy of the repo in your own GitHub account.
- GitHub will take you to your forked version of the repository.

2. Clone your fork

- Click the green `Clone` button and choose the SSH or HTTPS URL depending on your setup.
- Copy the URL and run `git clone <url>` in your local terminal.
- This will clone the `develop` branch of the `great_expectations` repo. Please use `develop` (not `main`!) as the starting point for your work.
- Atlassian has a [nice tutorial for developing on a fork](#).

3. Add the upstream remote

- On your local machine, `cd` into the `great_expectations` repo you cloned in the previous step.
- Run:

```
git remote add upstream git@github.com:great-expectations/great_expectations.git
```
- This sets up a remote called `upstream` to track changes to the main branch.

4. Create a feature branch to start working on your changes.

- Ex: `git checkout -b feature/my-feature-name`
- We do not currently follow a strict naming convention for branches. Please pick something clear and self-explanatory, so that it will be easy for others to get the gist of your work.

6.1.3 Install python dependencies

5. Create a new virtual environment

- Make a new virtual environment (e.g. using `virtualenv` or `conda`), name it “`great_expectations_dev`” or similar.
- Ex: `virtualenv great_expectations_dev; source great_expectations_dev/bin/activate`
- This is not required, but highly recommended.

6. Install dependencies from `requirements-dev.txt`

- `pip install -r requirements-dev.txt`
- This will ensure that sure you have the right libraries installed in your python environment.

7. Install `great_expectations` from your cloned repo

- `pip install -e .`
- `-e` will install Great Expectations in “`editable`” mode. This is not required, but is often very convenient as a developer.

6.1.4 (Optional) Configure resources for testing and documentation

Depending on which features of Great Expectations you want to work on, you may want to configure different back-ends for local testing, such as `postgresql` and `Spark`. Also, there are a couple of extra steps if you want to build documentation locally.

If you want to develop against local `postgresql`:

- To simplify setup, the repository includes a `docker-compose` file that can stand up a local `postgresql` container. To use it, you’ll need to have [docker installed](#).
- Navigate to `assets/docker/postgresql` in your `great_expectations` repo and run `docker-compose up -d`
- Within the same directory, you can run `docker-compose ps` to verify that the container is running. You should see something like:

Name	Command	State	
↔Ports			

postgresql_travis_db_1	docker-entrypoint.sh postgres	Up	0.0.0.
↔0:5432->5432/tcp			

- Once you’re done testing, you can shut down your `postgres` container by running `docker-compose down` from the same directory.
- Caution: If another service is using port 5432, `docker` may start the container but silently fail to set up the port. In that case, you will probably see errors like this:

```
psycopg2.OperationalError: could not connect to server: Connection refused
Is the server running on host "localhost" (::1) and accepting
TCP/IP connections on port 5432?
could not connect to server: Connection refused
Is the server running on host "localhost" (127.0.0.1) and accepting
TCP/IP connections on port 5432?
```

- Or this...

```
sqlalchemy.exc.OperationalError: (psycopg2.OperationalError) FATAL:  _
↳ database "test_ci" does not exist
(Background on this error at: http://sqlalche.me/e/e3q8)
```

If you want to develop against local Spark:

- In most cases, `pip install requirements-dev.txt` should set up pyspark for you.
- If you don't have Java installed, you will probably need to install it and set your `PATH` or `JAVA_HOME` environment variables appropriately.
- You can find official installation instructions for spark [here](#).

If you want to build documentation locally:

- `pip install -r docs/requirements.txt`
- To build documentation, the command is `cd docs; make html`
- Documentation will be generated in `docs/build/html/` with the `index.html` as the index page.

6.1.5 Run tests to confirm that everything is working

You can run all tests by running `pytest` in the `great_expectations` directory root. Please see [Testing](#) for testing options and details.

6.1.6 Start coding!

At this point, you have everything you need to start coding!

6.2 Contribution checklist

Following these instructions helps us make sure the code review and merge process go smoothly.

6.2.1 Before submitting a pull request

Once your code is ready, please go through the following checklist before submitting a pull request.

1. Have you signed the CLA?

- A Contributor License Agreement helps guarantee that contributions to Great Expectations will always remain free and open.
- Please see [Contributor license agreement \(CLA\)](#) for more information and instructions for how to sign the CLA the first time you contribute to Great Expectations.

- If you’ve included your (physical) mailing address in the CLA, we’ll send you a personalized Great Expectations mug once your first PR is merged!

2. Have you followed the Style Guide for code and comments?

- The *Style Guide* is here.
- Thanks for helping us keep the codebase and documentation clean and consistent, so that it’s easier to maintain it as a community!

3. Is your branch up to date with upstream/develop?

- Update your local repository with the most recent code from the main Great Expectations repository.
- For changes with few or no merge conflicts, you can do this by creating a draft pull request in GitHub and clicking `Update branch`.
- You can also rebase your branch from `upstream/develop`. In general, the steps are:
 1. Run `git fetch upstream` then `git rebase upstream/develop`.
 2. Fix any merge conflicts that arise from the rebase.
 3. Make sure to add and commit all your changes in this step.
 4. Re-run tests to ensure the rebase did not introduce any new issues.
- Atlassian and Github both have good tutorials for rebasing: [Atlassian’s tutorial](#), [Github’s tutorial](#).

4. Have you written and run all the tests you need?

- See *Writing unit and integration tests* for details on how to write unit tests in Great Expectations.
- Please make certain to run `pytest` to verify that all tests pass locally. See *Running tests* for details.

5. Have you added a bullet with your changes under the “develop” heading in the Changelog?

- Please add a bullet point to `docs/changelog/changelog.rst`, in the `develop` section.
- You can see the past Changelog here: *Changelog*

If you’ve checked off all these items, you’re now ready to submit a pull request!

6.2.2 How to submit a pull request

When you’re done with your work...

1. Create a PR

- Push to the remote fork of your repo.
- Follow [these instructions](#) to create a PR from your commit.
- Provide background for reviewers so they can understand and approve your PR more quickly:
 - Choose a short title which sums up the changes that you have made.
 - Add a tag to help categorize the PR:
 - * [BUGFIX] for PRs that address minor bugs without changing behavior,
 - * [ENHANCEMENT] for PRs that enhance an existing feature,
 - * [FEATURE] for significant PRs that add a new feature likely to require being added to our feature maturity matrix,
 - * [DOCS] for PRs that focus on improving documentation, or

- * [MAINTENANCE] for PRs that focus on updating repository settings or related chores.
- Summarize your changes using a few clear sentences (sometimes screenshots are nice too!). A good guide is to aim for a collection of commit message summaries that provide more details about what your changes do, like “Fixed handling of malformed datasource configuration” or “Improved docstrings for store module”
- Finally, in the section for design review, include a description of any prior discussion or coordination on the features in the PR, such as mentioning the number of the issue where discussion has taken place, e.g. “Closes #123”, linking to a relevant discuss or slack article, citing a team meeting, or even noting that no discussion is relevant because the issue is small.

2. Confirm the contributor license agreement (CLA)

- If you’ve followed the checklist above, you will have already signed the CLA and won’t see the CLA bot.
- Otherwise, you will see a comment from the “CLA Bot” on the PR that asks you to complete the CLA form. Please do so.
- Once you’ve signed the form, add a new comment to the PR with the line `@cla-bot check`. This will trigger the CLA bot to refresh.

3. Verify continuous integration checks

- Wait for the other continuous integration (CI) checks to go green and watch out for a comment from the automated linter that checks for syntax and formatting issues.
- Fix any issues that are flagged.

4. Wait for a core team member to approve and merge your PR

- Once all checks pass, a Great Expectations team member will approve your PR and merge it.
- GitHub will notify you of comments or a successful merge according to your notification settings.

5. Resolve any issues

- There will probably be discussion about the pull request. It’s normal for a request to require some changes before merging it into the main Great Expectations project. We enjoy working with contributors to help them get their code accepted. There are many approaches to fixing a problem and it is important to find the best approach before writing too much code!

6. Do a victory dance

- Congratulations! You’ve just contributed to Great Expectations!

6.3 Making changes directly through Github

If you want to change documentation, but not code, we suggest using the GitHub markdown editor, which means you don’t have to fork the repo at all. Here’s how you do this:

6.3.1 Start editing

1A. Edit from docs.greatexpectations.io

- Go to the [Great Expectations docs](#).
- On each page, you'll see an `Edit on GitHub` button in the top right. Click this to go to the source file in the Great Expectations GitHub repo.

1B. Edit from Github

- If you're already on GitHub, the docs are located in `great_expectations > docs`. You can directly navigate to the respective page you want to edit (but getting there from [docs.great_expectations.io](#) is a little easier).
- In the top right of the grey header bar of the actual file, click the pencil icon to get into edit mode on GitHub.

2. Make edits

- Make your edits and use the Preview tab to preview changes.
- Please pay close attention to the *Style Guide*.

6.3.2 Submit a pull request

3. Submit your edits as a PR

- When you're done, add a meaningful commit message at the bottom. Use a short title and a meaningful explanation of what you changed and why.
- Click the `Propose File Change` button at the bottom of the page.
- Click the `Create Pull Request` button.
- Optionally: Add comment to explain your change, if it's not already in the commit message.
- Click the next `Create Pull Request` button to create the actual PR.

4. Sign the CLA

- If this is your first contribution to Great Expectations, You will see a comment from the "CLA Bot" that asks you to complete the Contributor Licence Agreement form.
- Please complete the form and comment on the PR to say that you've signed the form.

5. Verify continuous integration checks

- Wait for the other Continuous Integration (CI) checks to go green and watch out for a comment from the automated linter that checks for syntax and formatting issues.
- Fix any issues that are flagged. (For documentation changes, it's unlikely that you'll have any issues.)

6. Wait for a core team member to approve and merge your PR

- Once all checks pass, a Great Expectations team member will approve your PR and merge it.
- GitHub will notify you of comments or a successful merge according to your notification settings.
- If there are any issues, please address them promptly.

7. Do a victory dance

- Congratulations! You've just contributed to Great Expectations!

6.4 Testing

6.4.1 Running tests

You can run all unit tests by running `pytest` in the `great_expectations` directory root.

If you did not configure optional backends for testing, tests against these backends will fail.

You can suppress these tests by adding the following flags:

- `--no-postgresql` will skip postgres tests
- `--no-spark` will skip spark tests
- `--no-sqlalchemy` will skip all tests using sqlalchemy (i.e. all database backends)

For example, you can run `pytest --no-spark --no-sqlalchemy` to skip all local backend tests (with the exception of the pandas backend). Please note that these tests will still be run by the CI as soon as you open a PR, so some tests might fail if your code changes affected them.

Note: as of early 2020, the tests generate many warnings. Most of these are generated by dependencies (pandas, sqlalchemy, etc.) You can suppress them with pytest's `--disable-pytest-warnings` flag: `pytest --no-spark --no-sqlalchemy --disable-pytest-warnings`

6.4.2 Writing unit and integration tests

Production code in Great Expectations must be thoroughly tested. In general, we insist on unit tests for all branches of every method, including likely error states. Most new feature contributions should include several unit tests. Contributions that modify or extend existing features should include a test of the new behavior.

Experimental code in Great Expectations need only be tested lightly. We are moving to a convention where experimental features are clearly labeled in documentation and the code itself. However, this convention is not uniformly applied today.

Most of Great Expectations' integration testing is in the CLI, which naturally exercises most of the core code paths. Because integration tests require a lot of developer time to maintain, most contributions should *not* include new integration tests, unless they change the CLI itself.

Note: we do not currently test Great Expectations against all types of SQL database. CI test coverage for SQL is limited to postgresql and sqlite. We have observed some bugs because of unsupported features or differences in SQL dialects, and we are actively working to improve dialect-specific support and testing.

6.4.3 Unit tests for Expectations

One of Great Expectations' important promises is that the same Expectation will produce the same result across all supported execution environments: pandas, sqlalchemy, and Spark.

To accomplish this, Great Expectations encapsulates unit tests for Expectations as JSON files. These files are used as fixtures and executed using a specialized test runner that executes tests against all execution environments.

Test fixture files are structured as follows:

```
{
  "expectation_type" : "expect_column_max_to_be_between",
  "datasets" : [{
    "data" : {...},
    "schemas" : {...},
```

(continues on next page)

```

    "tests" : [...]
  }]
}

```

Each item under `datasets` includes three entries: `data`, `schemas`, and `tests`.

data

...defines a dataframe of sample data to apply Expectations against. The dataframe is defined as a dictionary of lists, with keys containing column names and values containing lists of data entries. All lists within a dataset must have the same length.

```

"data" : {
  "w" : [1, 2, 3, 4, 5, 5, 4, 3, 2, 1],
  "x" : [2, 3, 4, 5, 6, 7, 8, 9, null, null],
  "y" : [1, 1, 1, 2, 2, 2, 3, 3, 3, 4],
  "z" : ["a", "b", "c", "d", "e", null, null, null, null, null],
  "zz" : ["1/1/2016", "1/2/2016", "2/2/2016", "2/2/2016", "3/1/2016", "2/1/
↪2017", null, null, null, null],
  "a" : [null, 0, null, null, 1, null, null, 2, null, null],
},

```

schemas

...define the types to be used when instantiating tests against different execution environments, including different SQL dialects. Each schema is defined as dictionary with column names and types as key-value pairs. If the schema isn't specified for a given execution environment, Great Expectations will introspect values and attempt to guess the schema.

```

"schemas": {
  "sqlite": {
    "w" : "INTEGER",
    "x" : "INTEGER",
    "y" : "INTEGER",
    "z" : "VARCHAR",
    "zz" : "DATETIME",
    "a" : "INTEGER",
  },
  "postgresql": {
    "w" : "INTEGER",
    "x" : "INTEGER",
    "y" : "INTEGER",
    "z" : "TEXT",
    "zz" : "TIMESTAMP",
    "a" : "INTEGER",
  }
},

```

tests

...define the tests to be executed against the dataframe. Each item in `tests` must have `title`, `exact_match_out`, `in`, and `out`. The test runner will execute the named Expectation once for each item, with the values in `in` supplied as `kwargs`.

The test passes if the values in the expectation validation result correspond with the values in `out`. If `exact_match_out` is `true`, then every field in the Expectation output must have a corresponding, matching field in `out`. If it's `false`, then only the fields specified in `out` need to match. For most use cases, `false` is a better fit, because it allows narrower targeting of the relevant output.

`suppress_test_for` is an optional parameter to disable an Expectation for a specific list of backends.

See an example below. For other examples

```
"tests" : [{
  "title": "Basic negative test case",
  "exact_match_out" : false,
  "in": {
    "column": "w",
    "result_format": "BASIC",
    "min_value": null,
    "max_value": 4
  },
  "out": {
    "success": false,
    "observed_value": 5
  },
  "suppress_test_for": ["sqlite"]
},
...
]
```

The test fixture files are stored in subdirectories of `tests/test_definitions/` corresponding to the class of Expectation:

- `column_map_expectations`
- `column_aggregate_expectations`
- `column_pair_map_expectations`
- `column_distributional_expectations`
- `multicolumn_map_expectations`
- `other_expectations`

By convention, the name of the the file is the name of the Expectation, with a `.json` suffix. Creating a new json file will automatically add the new Expectation tests to the test suite.

Note: If you are implementing a new Expectation, but don't plan to immediately implement it for all execution environments, you should add the new test to the appropriate list(s) in the `candidate_test_is_on_temporary_notimplemented_list` method within `tests/test_utils.py`. Often, we see Expectations developed first for pandas, then later extended to SQLAlchemy and Spark.

You can run just the Expectation tests with `pytest tests/test_definitions/test_expectations.py`.

6.4.4 Manual testing

We do manual testing (e.g. against various databases and backends) before major releases and in response to specific bugs and issues.

6.5 Levels of maturity

Features and code within Great Expectations are separated into three levels of maturity: Experimental, Beta, and Production.

Being explicit about these levels allows us to enable experimentation, without creating unnecessary thrash when features and APIs evolve. It also helps streamline development, by giving contributors a clear, incremental path to create and improve the Great Expectations code base.

For users of Great Expectations, our goal is to enable informed decisions about when to adopt which features.

For contributors to Great Expectations, our goal is to channel creativity by always making the next step as clear as possible.

This grid provides guidelines for how the maintainers of Great Expectations evaluate levels of maturity. Maintainers will always exercise some discretion in determining when any given feature is ready to graduate to the next level. If you have ideas or suggestions for leveling up a specific feature, please raise them in Github issues, and we'll work with you to get there.

* Experimental classes log warning-level messages when initialized: “Warning: great_expectations.some_module.SomeClass is experimental. Methods, APIs, and core behavior may change in the future.”

** In the special case of Expectations, some gaps in implementation are allowed in beta (e.g. works in pandas and Spark, but not yet in SQLAlchemy; validation and rendering work, but not profiling yet)

6.6 Style Guide

Note: This style guide will be enforced for all incoming PRs. However, certain legacy areas within the repo do not yet fully adhere to the style guide. We welcome PRs to bring these areas up to code.

6.6.1 General conventions

- **The project name “Great Expectations” is always spaced and capitalized.** Good: “Great Expectations”. Bad: “great_expectations”, “great expectations”, “GE.”
- **We refer to ourselves in the first person plural.** Good: “we”, “our”. Bad: “I”. This helps us avoid awkward passive sentences. Occasionally, we refer to ourselves as “the Great Expectations team” (or community) for clarity.
- **We refer to developers and users as “you”.** Good: “you can,” “you might want to.”
- **We reserve the word “should” for strong directives, not just helpful guidance.**
- **Dickens allusions, puns, and references are strongly encouraged.** When referencing the works of Dickens, strict accuracy is required. “A Muppet Christmas Carol” is considered canon.

6.6.2 code

- **Methods are almost always named using snake_case.**
- **Methods that behave as operators (e.g. comparison or equality) are named using camelCase.** These methods are rare and should be changed with great caution. Please reach out to us if you see the need for a change of this kind.
- **Experimental methods should log an experimental warning when called:** “Warning: some_method is experimental. Methods, APIs, and core behavior may change in the future.”
- **Experimental classes should log an experimental warning when initialized:** “Warning: great_expectations.some_module.SomeClass is experimental. Methods, APIs, and core behavior may change in the future.”
- **Docstrings are highly recommended.** We use the Sphinx’s [Napoleon extension](#) to build documentation from Google-style docstrings.
- **Lint your code.** Our CI system will check using `black` and `isort`. We have a git pre-commit configuration in the repo, so you can just run `pre-commit install` to automatically run your changes through the linting process before submitting.

Expectations

- **Use unambiguous Expectation names,** even if they’re a bit longer, e.g. `expect_columns_to_match_ordered_list` instead of `expect_columns_to_be`.
- **Avoid abbreviations,** e.g. `column_index` instead of `column_idx`.
- **Expectation names should reflect their decorators:**
 - `expect_table...` for methods decorated directly with `@expectation`
 - `expect_column_values...` for `@column_map_expectation`
 - `expect_column...` for `@column_aggregate_expectation`
 - `expect_column_pair_values...` for `@column_pair_map_expectation`

The CLI

The [CLI](#) has some conventions of its own.

- The CLI never writes to disk without asking first.
- Questions are always phrased as conversational sentences.
- Sections are divided by headers: “===== Profiling =====”
- We use punctuation: Please finish sentences with periods, questions marks, or an occasional exclamation point.
- Keep indentation and line spacing consistent! (We’re pythonistas, natch.)
- Include exactly one blank line after every question.
- Within those constraints, shorter is better. When in doubt, shorten.
- Clickable links (usually to documentation) are blue.
- Copyable bash commands are green.
- All top-level bash commands must be nouns: “docs build”, not “build docs”

6.6.3 .rst files

Organization

Within the table of contents, each section has specific role to play. Broadly speaking, we follow Divio’s excellent [Documentation System](#), with the caveat that our “Reference” section is their “Explanation” section, and our “Module docs” section is their “Reference section”.

- **Introduction** explains the Why of Great Expectations, so that potential users can quickly decide whether or not the library can help them.
- **Tutorials** help users and contributors get started quickly. Along the way they orient new users to concepts that will be important to know later.
- **How-to guides** help users accomplish specific goals that go beyond the generic tutorials. Article titles within this section always start with “How to”: “How to create custom Expectations”. They often reference specific tools or infrastructure: “How to validate Expectations from within a notebook”, “How to build data docs in S3.”
- **Reference** articles explain the architecture of Great Expectations. These articles explain core concepts, discuss alternatives and options, and provide context, history, and direction for the project. Reference articles avoid giving specific technical advice. They also avoid implementation details that can be captured in docstrings instead.
- **Community** helps expand the Great Expectations community by explaining how to get in touch to ask questions, make contributions, etc.
- **Module docs** link through to module docstrings themselves.

Titles

- **Headers are capitalized like sentences.** Yep: “Installing within a project.” Nope: “Installing Within a Project.”
- **For sections within “how to”-type guides, titles should be short, imperative sentences.** Avoid extra words. Good: “Configure data documentation”. Nope: “Configuring data documentation”. Avoid: “Configure documentation for your data”
- **Please follow the Sphinx guide for sections to determine which of the many, confusing .rst underlining conventions to use:** [Sphinx guide for sections](#)

Core concepts and classes

- **Core concepts are always capitalized, and always are linked on first reference within each page.** Pretend the docs are a fantasy novel, and core concepts are magic.
 - Wrong: “You can create expectation suites as follows...”
 - Better: “You can create *Expectation Suites* as follows...”
 - Avoid: “You can create suites of Expectations as follows...”
- **Class names are written in upper camel case, and always linked on first reference.** Good: “ValidationOperator.” Bad: “validationOperator”, “validation operator”. If a word is both a core concept and a class name, prefer the core concept unless the text refers specifically to the class.

File names, RST refs, and links

- **File names should parallel titles, so that URLs and titles are similar.** For example: the page titled *Initialize a project* has this filename: `initialize_a_project.rst`, which produces this URL: `initialize_a_project.html`
- **Use snake case for file names.**
- **Refs are ``_{filename}`` or ``_{folder_name}_{filename}``.** Ex: `_getting_started__initialize_a_project`
- **Links to docs in the API Reference section**

- Link to a module: `great_expectations.data_context.data_context`
- Abbreviated link to a module: `data_context`
- Link to a class: `great_expectations.data_context.data_context.BaseDataContext`
- Abbreviated link to a class: `BaseDataContext`
- Link to a method in a class: `great_expectations.data_context.data_context.BaseDataContext.validate_config()`
- Abbreviated link to a method in a class: `validate_config()`
- Link to an attribute in a class: `great_expectations.data_context.data_context.BaseDataContext.GE_DIR`
- Abbreviated link to an attribute in a class: `GE_DIR`
- Link to a function in a module: `great_expectations.jupyter_ux.display_column_evrs_as_section`
- Abbreviated to a function in a module: `display_column_evrs_as_section`

Code formatting

- **For inline code in RST, make sure to use double backticks.** This isn't markdown, folks:
 - Yep: The `init` command will walk you through setting up a new project and connecting to your data.
 - Nope: The `init` command will walk you through setting up a new project and connecting to your data.
- **For inline bash blocks, do not include a leading \$.** It makes it hard for users to copy-paste code blocks.

6.7 Miscellaneous

6.7.1 Core team

- James Campbell
- Abe Gong
- Eugene Mandel
- Rob Lim
- Taylor Miller
- Alex Shertinsky
- Tal Gluck
- Kyle Eaton
- Sam Bail
- William Shin
- Ben Castleton

6.7.2 Contributor license agreement (CLA)

When you contribute code, you affirm that the contribution is your original work and that you license the work to the project under the project's open source license. Whether or not you state this explicitly, by submitting any copyrighted material via pull request, email, or other means you agree to license the material under the project's open source license and warrant that you have the legal authority to do so.

Please make sure you have signed our Contributor License Agreement (either [Individual Contributor License Agreement v1.0](#) or [Software Grant and Corporate Contributor License Agreement \("Agreement"\) v1.0](#)).

We are not asking you to assign copyright to us, but to give us the right to distribute your code without restriction. We ask this of all contributors in order to assure our users of the origin and continuing existence of the code. You only need to sign the CLA once.

6.7.3 Release checklist

GE core team members use this checklist to ship releases.

1. If this is a major release (incrementing either the first or second version number) the manual acceptance testing must be completed.
 - This [private google doc](#) outlines the procedure. (Note this will be made public eventually)
2. Merge all approved PRs into develop.
3. Make a new branch from develop called something like `release-prep-2020-06-01`.
4. In this branch, update the version number in the `.travis.yml` file (look in the deploy section). (This sed snippet is handy if you change the numbers `sed -i '' 's/0\.9\.6/0\.9\.7/g' .travis.yml`)
5. Update the `changelog.rst`: move all things under develop under a new heading with the new release number.
 - Verify that any changes to requirements are specifically identified in the changelog
6. Submit this as a PR against develop
7. After successful checks, get it approved and merged.
8. Update your local branches and switch to main: `git fetch --all; git checkout main; git pull`.
9. Merge the now-updated develop branch into main and trigger the release: `git merge origin/develop; git push`
10. Wait for all the build to complete. It should include 4 test jobs and a deploy job, which handles the actual publishing of code to pypi. You can watch the progress of these builds on Travis.
11. Check [PyPI](#) for the new release
12. Create an annotated git tag:
 - Run `git tag -a <<VERSION>> -m "<<VERSION>>"` with the correct new version.
 - Push the tag up by running `git push origin <<VERSION>>` with the correct new version.
 - Merge main into develop so that the tagged commit becomes part of the history for develop: `git checkout develop; git pull; git merge main`
 - On develop, add a new “develop” section header to `changelog.rst`, and push the updated file with message “Update changelog for develop”

13. [Create the release on GitHub](#) with the version number. Copy the changelog notes into the release notes, and update any rst-specific links to use github issue numbers.
 - The deploy step will automatically create a draft for the release.
 - Generally, we use the name of the tag (Ex: “0.11.2”) as the release title.
14. Notify kyle@superconductive.com about any community-contributed PRs that should be celebrated.
15. Socialize the release on GE slack by copying the changelog with an optional nice personal message (thank people if you can)
16. Review the automatically-generated PR for conda-forge (<https://github.com/conda-forge/great-expectations-feedstock/pulls>), updating requirements as necessary and verifying the build status.

Beta Release Notes

- To ship a beta release, follow the above checklist, but use the branch name `v0.11.x` as the equivalent of `main` and `v0.11.x-develop` as the equivalent of `develop`
- Ship the release using beta version numbers when updating the `.travis.yml` and when creating the annotated tag (e.g. *0.11.0b0*)

- [genindex](#)
- [modindex](#)

7.1 API Reference

This page contains auto-generated API reference documentation¹.

7.1.1 `great_expectations`

Subpackages

`great_expectations.cli`

Subpackages

`great_expectations.cli.upgrade_helpers`

Submodules

`great_expectations.cli.upgrade_helpers.base_upgrade_helper`

Module Contents

Classes

`BaseUpgradeHelper()`

Base UpgradeHelper abstract class

class `great_expectations.cli.upgrade_helpers.base_upgrade_helper.BaseUpgradeHelper`
Bases: `abc.ABC`
Base UpgradeHelper abstract class
abstract `get_upgrade_overview` (*self*)

¹ Created with `sphinx-autoapi`

```
abstract upgrade_project (self)
```

```
great_expectations.cli.upgrade_helpers.upgrade_helper_v11
```

Module Contents

Classes

<code>UpgradeHelperV11(data_context=None, text_root_dir=None)</code>	con-	Base UpgradeHelper abstract class
--	------	-----------------------------------

```
class great_expectations.cli.upgrade_helpers.upgrade_helper_v11.UpgradeHelperV11 (data_context=
con-
text_root_dir=
```

```
Bases:          great_expectations.cli.upgrade_helpers.base_upgrade_helper.
BaseUpgradeHelper
```

```
Base UpgradeHelper abstract class
```

```
_generate_upgrade_checklist (self)
```

```
_process_docs_site_for_checklist (self, site_name, site_config)
```

```
_process_validations_store_for_checklist (self, store_name, store)
```

```
_process_metrics_store_for_checklist (self, store_name, store)
```

```
upgrade_store_backend (self, store_backend, store_name=None, site_name=None)
```

```
_update_upgrade_log (self, store_backend, source_key=None, dest_key=None, store_name=None,
site_name=None, exception_message=None)
```

```
_update_validation_result_json (self, source_key, dest_key, run_name, store_backend)
```

```
_get_tuple_filesystem_store_backend_run_time (self, source_key, store_backend)
```

```
_get_tuple_s3_store_backend_run_time (self, source_key, store_backend)
```

```
_get_tuple_gcs_store_backend_run_time (self, source_key, store_backend)
```

```
_get_skipped_store_and_site_names (self)
```

```
get_upgrade_overview (self)
```

```
_save_upgrade_log (self)
```

```
_generate_upgrade_report (self)
```

```
upgrade_project (self)
```

Package Contents

Classes

<code>UpgradeHelperV11</code> (data_context=None, text_root_dir=None)	con-	Base UpgradeHelper abstract class
--	------	-----------------------------------

```
class great_expectations.cli.upgrade_helpers.UpgradeHelperV11 (data_context=None,
                                                                con-
                                                                text_root_dir=None)
    Bases:
        great_expectations.cli.upgrade_helpers.base_upgrade_helper.
        BaseUpgradeHelper
    Base UpgradeHelper abstract class
    _generate_upgrade_checklist (self)
    _process_docs_site_for_checklist (self, site_name, site_config)
    _process_validations_store_for_checklist (self, store_name, store)
    _process_metrics_store_for_checklist (self, store_name, store)
    upgrade_store_backend (self, store_backend, store_name=None, site_name=None)
    _update_upgrade_log (self, store_backend, source_key=None, dest_key=None, store_name=None,
                        site_name=None, exception_message=None)
    _update_validation_result_json (self, source_key, dest_key, run_name, store_backend)
    _get_tuple_filesystem_store_backend_run_time (self, source_key, store_backend)
    _get_tuple_s3_store_backend_run_time (self, source_key, store_backend)
    _get_tuple_gcs_store_backend_run_time (self, source_key, store_backend)
    _get_skipped_store_and_site_names (self)
    get_upgrade_overview (self)
    _save_upgrade_log (self)
    _generate_upgrade_report (self)
    upgrade_project (self)
great_expectations.cli.upgrade_helpers.GE_UPGRADE_HELPER_VERSION_MAP
```

Submodules

`great_expectations.cli.checkpoint`

Module Contents

Functions

<code>checkpoint()</code>	Checkpoint operations
---------------------------	-----------------------

continues on next page

Table 4 – continued from previous page

<code>checkpoint_new</code> (checkpoint, suite, directory, data-source)	Create a new checkpoint for easy deployments. (Experimental)
<code>_verify_checkpoint_does_not_exist</code> (context: DataContext, checkpoint: str, usage_event: str)	
<code>_write_checkpoint_to_disk</code> (context: DataContext, checkpoint: dict, checkpoint_name: str)	
<code>_load_checkpoint_yaml_template</code> ()	
<code>checkpoint_list</code> (directory)	List configured checkpoints. (Experimental)
<code>checkpoint_run</code> (checkpoint, directory)	Run a checkpoint. (Experimental)
<code>checkpoint_script</code> (checkpoint, directory)	Create a python script to run a checkpoint. (Experimental)
<code>_validate_at_least_one_suite_is_listed</code> (context: DataContext, batch: dict, checkpoint_file: str)	
<code>_load_script_template</code> ()	
<code>_write_checkpoint_script_to_disk</code> (context_directory: str, checkpoint_name: str, script_path: str)	

great_expectations.cli.checkpoint.**SQLAlchemyError**

great_expectations.cli.checkpoint.**SQLAlchemyError**

great_expectations.cli.checkpoint.**yaml**

great_expectations.cli.checkpoint.**checkpoint** ()

Checkpoint operations

A checkpoint is a bundle of one or more batches of data with one or more Expectation Suites.

A checkpoint can be as simple as one batch of data paired with one Expectation Suite.

A checkpoint can be as complex as many batches of data across different datasources paired with one or more Expectation Suites each.

great_expectations.cli.checkpoint.**checkpoint_new** (checkpoint, suite, directory, data-source)

Create a new checkpoint for easy deployments. (Experimental)

great_expectations.cli.checkpoint.**_verify_checkpoint_does_not_exist** (context: DataContext, checkpoint: str, usage_event: str) → None

great_expectations.cli.checkpoint.**_write_checkpoint_to_disk** (context: DataContext, checkpoint: dict, checkpoint_name: str) → str

great_expectations.cli.checkpoint.**_load_checkpoint_yaml_template** () → dict

great_expectations.cli.checkpoint.**checkpoint_list** (directory)

List configured checkpoints. (Experimental)

great_expectations.cli.checkpoint.**checkpoint_run** (checkpoint, directory)

Run a checkpoint. (Experimental)


```
great_expectations.cli.checkpoint.checkpoint_script (checkpoint, directory)
```

Create a python script to run a checkpoint. (Experimental)

Checkpoints can be run directly without this script using the *great_expectations checkpoint run* command.

This script is provided for those who wish to run checkpoints via python.

```
great_expectations.cli.checkpoint._validate_at_least_one_suite_is_listed (context:
                                                                    Dat-
                                                                    a-
                                                                    Con-
                                                                    text,
                                                                    batch:
                                                                    dict,
                                                                    check-
                                                                    point_file:
                                                                    str)
                                                                    →
                                                                    None
```

```
great_expectations.cli.checkpoint._load_script_template () → str
```

```
great_expectations.cli.checkpoint._write_checkpoint_script_to_disk (context_directory:
                                                                    str, check-
                                                                    point_name:
                                                                    str,
                                                                    script_path:
                                                                    str) →
                                                                    None
```

`great_expectations.cli.checkpoint_script_template`

This is a basic generated Great Expectations script that runs a checkpoint.

A checkpoint is a list of one or more batches paired with one or more Expectation Suites and a configurable Validation Operator.

Checkpoints can be run directly without this script using the *great_expectations checkpoint run* command. This script is provided for those who wish to run checkpoints via python.

Data that is validated is controlled by BatchKwargs, which can be adjusted in the checkpoint file: `great_expectations/checkpoints/{0}.yaml`.

Data are validated by use of the *ActionListValidationOperator* which is configured by default. The default configuration of this Validation Operator saves validation results to your results store and then updates Data Docs.

This makes viewing validation results easy for you and your team.

Usage: - Run this file: `python great_expectations/uncommitted/run_{0}.py`. - This can be run manually or via a scheduler such as cron. - If your pipeline runner supports python snippets you can paste this into your pipeline.

Module Contents

```
great_expectations.cli.checkpoint_script_template.context
great_expectations.cli.checkpoint_script_template.checkpoint
great_expectations.cli.checkpoint_script_template.batches_to_validate = []
great_expectations.cli.checkpoint_script_template.batch_kwargs
great_expectations.cli.checkpoint_script_template.results
```

great_expectations.cli.cli

Module Contents

Functions

<code>cli(verbose)</code>	Welcome to the great_expectations CLI!
<code>main()</code>	

```
great_expectations.cli.cli.cli(verbose)
    Welcome to the great_expectations CLI!

    Most commands follow this format: great_expectations <NOUN> <VERB>

    The nouns are: datasource, docs, project, suite, validation-operator

    Most nouns accept the following verbs: new, list, edit

    In particular, the CLI supports the following special commands:

    • great_expectations init : create a new great_expectations project
    • great_expectations datasource profile : profile a datasource
    • great_expectations docs build : compile documentation from expectations
```

```
great_expectations.cli.cli.main()
```

great_expectations.cli.cli_logging

Module Contents

Functions

<code>_set_up_logger()</code>

```
great_expectations.cli.cli_logging.logger
great_expectations.cli.cli_logging._set_up_logger()
```



```
great_expectations.cli.cli_messages.NO_DATASOURCES_FOUND
great_expectations.cli.cli_messages.SETUP_SUCCESS =
<cyan>Congratulations! Great Expectations is now set up.</cyan>
great_expectations.cli.cli_messages.SECTION_SEPARATOR =
```

```
great_expectations.cli.cli_messages.DONE = Done
```

great_expectations.cli.datasource

Module Contents

Classes

<i>DatasourceTypes()</i>	Generic enumeration.
<i>SupportedDatabases()</i>	Generic enumeration.

Functions

<i>datasource()</i>	Datasource operations
<i>datasource_new(directory)</i>	Add a new datasource to the data context.
<i>delete_datasource(directory, datasource)</i>	Delete the datasource specified as an argument
<i>datasource_list(directory)</i>	List known datasources.
<i>_build_datasource_intro_string(datasource_count)</i>	
<i>datasource_profile(datasource, batch_kwargs_generator_name, data_assets, profile_all_data_assets, directory, view, additional_batch_kwargs)</i>	Profile a datasource (Experimental)
<i>add_datasource(context, choose_one_data_asset=False)</i>	Interactive flow for adding a datasource to an existing context.
<i>_add_pandas_datasource(context, passthrough_generator_only=True, prompt_for_datasource_name=True)</i>	
<i>load_library(library_name, stall_instructions_string=None)</i>	in- Dynamically load a module from strings or raise a helpful error.
<i>_add_sqlalchemy_datasource(context, prompt_for_datasource_name=True)</i>	
<i>_should_hide_input()</i>	This is a workaround to help identify Windows and adjust the prompts accordingly
<i>_collect_postgres_credentials(default_credentials=None)</i>	
<i>_collect_snowflake_credentials(default_credentials=None)</i>	
<i>_collect_bigquery_credentials(default_credentials=None)</i>	
<i>_collect_mysql_credentials(default_credentials=None)</i>	
<i>_collect_redshift_credentials(default_credentials=None)</i>	

continues on next page

Table 8 – continued from previous page

<code>_add_spark_datasource(context, passthrough_generator_only=True, prompt_for_datasource_name=True)</code>	
<code>select_batch_kwargs_generator(context, datasource_name, available_data_assets_dict=None)</code>	
<code>get_batch_kwargs(context, datasource_name=None, batch_kwargs_generator_name=None, data_asset_name=None, additional_batch_kwargs=None)</code>	This method manages the interaction with user necessary to obtain batch_kwargs for a batch of a data asset.
<code>_get_batch_kwargs_from_generator_or_from_file_path(context, datasource_name, batch_kwargs_generator_name=None, additional_batch_kwargs=None)</code>	
<code>_get_batch_kwargs_for_sqlalchemy_datasource(context, datasource_name, additional_batch_kwargs=None)</code>	
<code>profile_datasource(context, datasource_name, batch_kwargs_generator_name=None, data_assets=None, profile_all_data_assets=False, max_data_assets=20, additional_batch_kwargs=None, open_docs=False)</code>	“Profile a named datasource using the specified context

`great_expectations.cli.datasource.logger`

class `great_expectations.cli.datasource.DatasourceTypes`

Bases: `enum.Enum`

Generic enumeration.

Derive from this class to define new enumerations.

PANDAS = pandas

SQL = sql

SPARK = spark

`great_expectations.cli.datasource.DATASOURCE_TYPE_BY_DATASOURCE_CLASS`

`great_expectations.cli.datasource.MANUAL_GENERATOR_CLASSES`

class `great_expectations.cli.datasource.SupportedDatabases`

Bases: `enum.Enum`

Generic enumeration.

Derive from this class to define new enumerations.

MYSQL = MySQL

POSTGRES = Postgres

REDSHIFT = Redshift

SNOWFLAKE = Snowflake

BIGQUERY = BigQuery

OTHER = other - Do you have a working SQLAlchemy connection string?

`great_expectations.cli.datasource.datasource()`

Datasource operations

`great_expectations.cli.datasource.datasource_new(directory)`

Add a new datasource to the data context.

`great_expectations.cli.datasource.delete_datasource(directory, datasource)`

Delete the datasource specified as an argument

`great_expectations.cli.datasource.datasource_list(directory)`

List known datasources.

`great_expectations.cli.datasource._build_datasource_intro_string(datasource_count)`

`great_expectations.cli.datasource.datasource_profile(datasource,
batch_kwarg_generator_name,
data_assets, pro-
file_all_data_assets, di-
rectory, view, addi-
tional_batch_kwarg)`

Profile a datasource (Experimental)

If the optional `data_assets` and `profile_all_data_assets` arguments are not specified, the profiler will check if the number of data assets in the datasource exceeds the internally defined limit. If it does, it will prompt the user to either specify the list of data assets to profile or to profile all. If the limit is not exceeded, the profiler will profile all data assets in the datasource.

`great_expectations.cli.datasource.add_datasource(context,
choose_one_data_asset=False)`

Interactive flow for adding a datasource to an existing context.

Parameters

- **context** –
- **choose_one_data_asset** – optional - if True, this signals the method that the intent is to let user choose just one data asset (e.g., a file) and there is no need to configure a batch kwarg generator that comprehensively scans the datasource for data assets

Returns a tuple: `datasource_name`, `data_source_type`

`great_expectations.cli.datasource._add_pandas_datasource(context,
passthrough_generator_only=True,
prompt_for_datasource_name=True)`

`great_expectations.cli.datasource.load_library(library_name, in-
stall_instructions_string=None)`

Dynamically load a module from strings or raise a helpful error.

Parameters

- **library_name** – name of the library to load
- **install_instructions_string** – optional - used when the install instructions are different from ‘pip install library_name’

Returns True if the library was loaded successfully, False otherwise

`great_expectations.cli.datasource._add_sqlalchemy_datasource(context,
prompt_for_datasource_name=True)`

`great_expectations.cli.datasource._should_hide_input()`

This is a workaround to help identify Windows and adjust the prompts accordingly since hidden prompts may freeze in certain Windows terminals

`great_expectations.cli.datasource._collect_postgres_credentials(default_credentials=None)`

`great_expectations.cli.datasource._collect_snowflake_credentials(default_credentials=None)`

`great_expectations.cli.datasource._collect_bigquery_credentials(default_credentials=None)`

```

great_expectations.cli.datasource._collect_mysql_credentials (default_credentials=None)
great_expectations.cli.datasource._collect_redshift_credentials (default_credentials=None)
great_expectations.cli.datasource._add_spark_datasource (context,
                                                         passthrough_generator_only=True,
                                                         prompt_for_datasource_name=True)
great_expectations.cli.datasource.select_batch_kwargs_generator (context, data-
                                                                source_name,
                                                                avail-
                                                                able_data_assets_dict=None)
great_expectations.cli.datasource.get_batch_kwargs (context, datasource_name=None,
                                                    batch_kwargs_generator_name=None,
                                                    data_asset_name=None, addi-
                                                    tional_batch_kwargs=None)

```

This method manages the interaction with user necessary to obtain batch_kwargs for a batch of a data asset.

In order to get batch_kwargs this method needs datasource_name, batch_kwargs_generator_name and data_asset_name to combine them into a fully qualified data asset identifier(datasource_name/batch_kwargs_generator_name/data_asset_name). All three arguments are optional. If they are present, the method uses their values. Otherwise, the method prompts user to enter them interactively. Since it is possible for any of these three components to be passed to this method as empty values and to get their values after interacting with user, this method returns these components' values in case they changed.

If the datasource has batch_kwargs_generators that can list available data asset names, the method lets user choose a name from that list (note: if there are multiple batch_kwargs_generators, user has to choose one first). If a name known to the chosen batch_kwargs_generator is selected, the batch_kwargs_generators will be able to yield batch_kwargs. The method also gives user an alternative to selecting the data asset name from the batch_kwargs_generators's list - user can type in a name for their data asset. In this case a passthrough batch kwargs batch_kwargs_generators will be used to construct a fully qualified data asset identifier (note: if the datasource has no passthrough batch_kwargs_generators configured, the method will exist with a failure). Since no batch_kwargs_generators can yield batch_kwargs for this data asset name, the method prompts user to specify batch_kwargs by choosing a file (if the datasource is pandas or spark) or by writing a SQL query (if the datasource points to a database).

Parameters

- **context** –
- **datasource_name** –
- **batch_kwargs_generator_name** –
- **data_asset_name** –
- **additional_batch_kwargs** –

Returns a tuple: (datasource_name, batch_kwargs_generator_name, data_asset_name, batch_kwargs). The components of the tuple were passed into the methods as optional arguments, but their values might have changed after this method's execution. If the returned batch_kwargs is None, it means that the batch_kwargs_generator will know to yield batch_kwargs when called.

```

great_expectations.cli.datasource._get_batch_kwargs_from_generator_or_from_file_path (context,
                                                                                       data-
                                                                                       source_
                                                                                       batch_k
                                                                                       ad-
                                                                                       di-
                                                                                       tional_b

```

```
great_expectations.cli.datasource._get_batch_kwargs_for_sqlalchemy_datasource(context,
                                                                                data-
                                                                                source_name,
                                                                                ad-
                                                                                di-
                                                                                tional_batch_kwa

great_expectations.cli.datasource.profile_datasource(context,      datasource_name,
                                                       batch_kwargs_generator_name=None,
                                                       data_assets=None,      pro-
                                                       file_all_data_assets=False,
                                                       max_data_assets=20,      addi-
                                                       tional_batch_kwargs=None,
                                                       open_docs=False)
```

“Profile a named datasource using the specified context

```
great_expectations.cli.datasource.msg_prompt_choose_datasource = Configure a datasource: 1
great_expectations.cli.datasource.msg_prompt_choose_database
great_expectations.cli.datasource.msg_prompt_filesys_enter_base_path =
Enter the path (relative or absolute) of the root directory where the data files are stored.
great_expectations.cli.datasource.msg_prompt_datasource_name =
Give your new Datasource a short name.
great_expectations.cli.datasource.msg_db_config =
```

Next, we will configure database credentials and store them in the {0:s} section of this config file:
great_expectations/uncommitted/config_variables.yml:

```
great_expectations.cli.datasource.msg_unknown_data_source =
```

Do we not have the type of data source you want?

- Please create a GitHub issue here so we can discuss it!
- <blue>https://github.com/great-expectations/great_expectations/issues/new</blue>

great_expectations.cli.docs

Module Contents

Functions

<code>docs()</code>	Data Docs operations
<code>docs_build(directory, site_name, view=True)</code>	Build Data Docs for a project.
<code>docs_list(directory)</code>	List known Data Docs Sites.
<code>clean_data_docs(directory, site_name=None, all=None)</code>	Delete data docs
<code>_build_intro_string(docs_sites_strings)</code>	
<code>build_docs(context, site_name=None, view=True)</code>	Build documentation in a context

```
great_expectations.cli.docs.docs()
Data Docs operations
```



```
great_expectations.cli.docs.docs_build(directory, site_name, view=True)
    Build Data Docs for a project.
great_expectations.cli.docs.docs_list(directory)
    List known Data Docs Sites.
great_expectations.cli.docs.clean_data_docs(directory, site_name=None, all=None)
    Delete data docs
great_expectations.cli.docs._build_intro_string(docs_sites_strings)
great_expectations.cli.docs.build_docs(context, site_name=None, view=True)
    Build documentation in a context
```

```
great_expectations.cli.init
```

Module Contents

Functions

<code>init(target_directory, view, usage_stats)</code>	Initialize a new Great Expectations project.
<code>_slack_setup(context)</code>	
<code>_get_full_path_to_ge_dir(target_directory)</code>	

```
great_expectations.cli.init.SQLAlchemyError
```

```
great_expectations.cli.init.init(target_directory, view, usage_stats)
    Initialize a new Great Expectations project.
```

This guided input walks the user through setting up a new project and also onboards a new developer in an existing project.

It scaffolds directories, sets up notebooks, creates a project file, and appends to a `.gitignore` file.

```
great_expectations.cli.init._slack_setup(context)
great_expectations.cli.init._get_full_path_to_ge_dir(target_directory)
```

```
great_expectations.cli.mark
```

Module Contents

Classes

<code>Mark()</code>	Marks for feature readiness.
---------------------	------------------------------

```
class great_expectations.cli.mark.Mark
    Marks for feature readiness.

    Usage: from great_expectations.cli.mark import Mark as mark
    @mark.blah def your_function()

    static cli_as_experimental (func: Callable)
```

Apply as a decorator to CLI commands that are Experimental.

static cli_as_beta (*func: Callable*)

Apply as a decorator to CLI commands that are beta.

static cli_as_deprecation (*message: str = '<yellow>Heads up! This feature will be deprecated in the next major release</yellow>'*)

Apply as a decorator to CLI commands that will be deprecated.

great_expectations.cli.project

Module Contents

Functions

<code>project()</code>	Project operations
<code>project_check_config(directory)</code>	Check a config for validity and help with migrations.
<code>project_upgrade(directory)</code>	Upgrade a project after installing the next Great Expectations major version.
<code>do_config_check(target_directory)</code>	

`great_expectations.cli.project.project()`

Project operations

`great_expectations.cli.project.project_check_config(directory)`

Check a config for validity and help with migrations.

`great_expectations.cli.project.project_upgrade(directory)`

Upgrade a project after installing the next Great Expectations major version.

`great_expectations.cli.project.do_config_check(target_directory)`

great_expectations.cli.store

Module Contents

Functions

<code>store()</code>	Store operations
<code>store_list(directory)</code>	List known Stores.

`great_expectations.cli.store.store()`

Store operations

`great_expectations.cli.store.store_list(directory)`

List known Stores.

great_expectations.cli.suite**Module Contents****Functions**

<code>suite()</code>	Expectation Suite operations
<code>suite_edit(suite, datasource, directory, jupyter, batch_kwargs)</code>	Generate a Jupyter notebook for editing an existing Expectation Suite.
<code>_suite_edit(suite, datasource, directory, jupyter, batch_kwargs, usage_event)</code>	
<code>suite_demo(suite, directory, view)</code>	Create a new demo Expectation Suite.
<code>suite_new(suite, directory, empty, jupyter, view, batch_kwargs)</code>	Create a new Expectation Suite.
<code>_suite_new(suite: str, directory: str, empty: bool, jupyter: bool, view: bool, batch_kwargs, usage_event: str)</code>	
<code>suite_delete(suite, directory)</code>	Delete an expectation suite from the expectation store.
<code>suite_scaffold(suite, directory, jupyter)</code>	Scaffold a new Expectation Suite.
<code>_suite_scaffold(suite: str, directory: str, jupyter: bool)</code>	
<code>suite_list(directory)</code>	Lists available Expectation Suites.
<code>_get_notebook_path(context, notebook_name)</code>	

`great_expectations.cli.suite.json_parse_exception`

`great_expectations.cli.suite.SQLAlchemyError`

`great_expectations.cli.suite.suite()`
Expectation Suite operations

`great_expectations.cli.suite.suite_edit(suite, datasource, directory, jupyter, batch_kwargs)`

Generate a Jupyter notebook for editing an existing Expectation Suite.

The SUITE argument is required. This is the name you gave to the suite when you created it.

A batch of data is required to edit the suite, which is used as a sample.

The edit command will help you specify a batch interactively. Or you can specify them manually by providing `--batch-kwarg`s in valid JSON format.

Read more about specifying batches of data in the documentation: <https://docs.greatexpectations.io/>

`great_expectations.cli.suite._suite_edit(suite, datasource, directory, jupyter, batch_kwargs, usage_event)`

`great_expectations.cli.suite.suite_demo(suite, directory, view)`
Create a new demo Expectation Suite.

Great Expectations will choose a couple of columns and generate expectations about them to demonstrate some examples of assertions you can make about your data.

`great_expectations.cli.suite.suite_new(suite, directory, empty, jupyter, view, batch_kwargs)`
Create a new Expectation Suite.

Great Expectations will choose a couple of columns and generate expectations about them to demonstrate some examples of assertions you can make about your data.

If you wish to skip the examples, add the `-empty` flag.

```
great_expectations.cli.suite._suite_new(suite: str, directory: str, empty: bool, jupyter: bool,
                                         view: bool, batch_kwargs, usage_event: str) →
                                         None
```

```
great_expectations.cli.suite.suite_delete(suite, directory)
Delete an expectation suite from the expectation store.
```

```
great_expectations.cli.suite.suite_scaffold(suite, directory, jupyter)
Scaffold a new Expectation Suite.
```

```
great_expectations.cli.suite._suite_scaffold(suite: str, directory: str, jupyter: bool) →
                                         None
```

```
great_expectations.cli.suite.suite_list(directory)
Lists available Expectation Suites.
```

```
great_expectations.cli.suite._get_notebook_path(context, notebook_name)
```

great_expectations.cli.tap_template

A basic generated Great Expectations checkpoint that validates a single batch of data.

Data that is validated is controlled by BatchKwargs, which can be adjusted in this script.

Data are validated by use of the *ActionListValidationOperator* which is configured by default. The default configuration of this Validation Operator saves validation results to your results store and then updates Data Docs.

This makes viewing validation results easy for you and your team.

Usage: - Run this file: `python {0}`. - This can be run manually or via a scheduler such as cron. - If your pipeline runner supports python snippets you can paste this into your pipeline.

Module Contents

```
great_expectations.cli.tap_template.context
great_expectations.cli.tap_template.suite
great_expectations.cli.tap_template.batch_kwargs
great_expectations.cli.tap_template.batch
great_expectations.cli.tap_template.results
```

great_expectations.cli.toolkit

Module Contents

Classes

```
MyYAML(: Any, _kw: Optional[Text] = enforce, typ:
Any = None, pure: Any = False, output: Any = None,
plug_ins=None)
```

Functions

<code>create_expectation_suite</code>	<code>(context, data_source_name=None, batch_kwarg_generator_name=None, generator_asset=None, batch_kwarg=None, expectation_suite_name=None, additional_batch_kwarg=None, empty_suite=False, show_intro_message=False, flag_build_docs=True, open_docs=False, profiler_configuration='demo', data_asset_name=None)</code>	Create a new expectation suite.
<code>_profile_to_create_a_suite</code>	<code>(additional_batch_kwarg, batch_kwarg, batch_kwarg_generator_name, context, datasource_name, expectation_suite_name, data_asset_name, profiler_configuration)</code>	
<code>_raise_profiling_errors</code>	<code>(profiling_results)</code>	
<code>attempt_to_open_validation_results_in_data_docs</code>	<code>(context, profiling_results)</code>	
<code>_get_default_expectation_suite_name</code>	<code>(batch_kwarg, data_asset_name)</code>	
<code>tell_user_suite_exists</code>	<code>(suite_name: str)</code>	
<code>create_empty_suite</code>	<code>(context: DataContext, expectation_suite_name: str, batch_kwarg)</code>	
<code>launch_jupyter_notebook</code>	<code>(notebook_path: str)</code>	
<code>load_batch</code>	<code>(context: DataContext, suite: Union[str, ExpectationSuite], batch_kwarg: Union[dict, BatchKwarg])</code>	
<code>load_expectation_suite</code>	<code>(context: DataContext, suite_name: str, usage_event: str)</code>	Load an expectation suite from a given context.
<code>exit_with_failure_message_and_stats</code>	<code>(context: DataContext, usage_event: str, message: str)</code>	
<code>load_checkpoint</code>	<code>(context: DataContext, checkpoint_name: str, usage_event: str)</code>	Load a checkpoint or raise helpful errors.
<code>select_datasource</code>	<code>(context: DataContext, datasource_name: str = None)</code>	Select a datasource interactively.
<code>load_data_context_with_error_handling</code>	<code>(directly_return_data_context: bool = False)</code>	Return a DataContext with good error handling and exit codes.
<code>upgrade_project</code>	<code>(context_root_dir, ge_config_version, from_cli_upgrade_command=False)</code>	
<code>confirm_proceed_or_exit</code>	<code>(confirm_prompt='Would you like to proceed?', continuation_message='Ok, exiting now. You can always read more at https://docs.greatexpectations.io/!', exit_on_no=True)</code>	Every CLI command that starts a potentially lengthy (>1 sec) computation

```
class great_expectations.cli.toolkit.MyYAML (: Any, _kw: Optional[Text] = enforce, typ:
                                             Any = None, pure: Any = False, output: Any
                                             = None, plug_ins=None)
```

```
    Bases: ruamel.yaml.YAML
```

```
    dump (self, data, stream=None, **kw)
```

```
great_expectations.cli.toolkit.yaml
```

```
great_expectations.cli.toolkit.default_flow_style = False
```

```
great_expectations.cli.toolkit.create_expectation_suite(context,          data-
                                                         source_name=None,
                                                         batch_kwarg_generator_name=None,
                                                         generator_asset=None,
                                                         batch_kwarg=None,
                                                         expecta-
                                                         tion_suite_name=None,
                                                         addi-
                                                         tional_batch_kwarg=None,
                                                         empty_suite=False,
                                                         show_intro_message=False,
                                                         flag_build_docs=True,
                                                         open_docs=False,    pro-
                                                         filer_configuration='demo',
                                                         data_asset_name=None)

Create a new expectation suite.

WARNING: the flow and name of this method and its interaction with _profile_to_create_a_suite require a
serious revisiting. :return: a tuple: (success, suite name, profiling_results)

great_expectations.cli.toolkit._profile_to_create_a_suite(additional_batch_kwarg,
                                                         batch_kwarg,
                                                         batch_kwarg_generator_name,
                                                         context,          data-
                                                         source_name,        ex-
                                                         pectation_suite_name,
                                                         data_asset_name,
                                                         profiler_configuration)

great_expectations.cli.toolkit._raise_profiling_errors(profiling_results)

great_expectations.cli.toolkit.attempt_to_open_validation_results_in_data_docs(context,
                                                                                pro-
                                                                                fil-
                                                                                ing_results)

great_expectations.cli.toolkit._get_default_expectation_suite_name(batch_kwarg,
                                                                    data_asset_name)

great_expectations.cli.toolkit.tell_user_suite_exists(suite_name: str) → None

great_expectations.cli.toolkit.create_empty_suite(context:      DataContext,    ex-
                                                         pectation_suite_name: str,
                                                         batch_kwarg) → None

great_expectations.cli.toolkit.launch_jupyter_notebook(notebook_path: str) →
                                                         None

great_expectations.cli.toolkit.load_batch(context: DataContext, suite: Union[str, Ex-
                                                         pectationSuite], batch_kwarg: Union[dict,
                                                         BatchKwarg]) → DataAsset

great_expectations.cli.toolkit.load_expectation_suite(context:      DataContext,
                                                         suite_name: str, usage_event:
                                                         str) → ExpectationSuite
```

Load an expectation suite from a given context.

Handles a suite name with or without `.json` :param usage_event:

```
great_expectations.cli.toolkit.exit_with_failure_message_and_stats (context:
                                                                    DataCon-
                                                                    text, us-
                                                                    age_event:
                                                                    str, mes-
                                                                    sage: str)
                                                                    → None
```

```
great_expectations.cli.toolkit.load_checkpoint (context:      DataContext,   check-
point_name:  str, usage_event:  str)
                                                    → dict
```

Load a checkpoint or raise helpful errors.

```
great_expectations.cli.toolkit.select_datasource (context:      DataContext,   data-
source_name:  str = None) →
Ddatasource
```

Select a datasource interactively.

```
great_expectations.cli.toolkit.load_data_context_with_error_handling (directory:
                                                                    str,
                                                                    from_cli_upgrade_command:
                                                                    bool =
                                                                    False)
                                                                    → Dat-
                                                                    aCon-
                                                                    text
```

Return a DataContext with good error handling and exit codes.

```
great_expectations.cli.toolkit.upgrade_project (context_root_dir,   ge_config_version,
from_cli_upgrade_command=False)
```

```
great_expectations.cli.toolkit.confirm_proceed_or_exit (confirm_prompt='Would
you like to proceed?', con-
tinuation_message='Ok,
exiting now. You can
always read more at
https://docs.greatexpectations.io/
!', exit_on_no=True)
```

Every CLI command that starts a potentially lengthy (>1 sec) computation or modifies some resources (e.g., edits the config file, adds objects to the stores) must follow this pattern: 1. Explain which resources will be created/modified/deleted 2. Use this method to ask for user's confirmation

The goal of this standardization is for the users to expect consistency - if you saw one command, you know what to expect from all others.

If the user does not confirm, the program should exit. The purpose of the exit_on_no parameter is to provide the option to perform cleanup actions before exiting outside of the function.

great_expectations.cli.util

Module Contents

Functions

cli_message(string)

cli_colorize_string(string)

continues on next page

Table 17 – continued from previous page

<code>cli_message_list(string_list, list_intro_string=None)</code>	Simple util function for displaying simple lists in cli
<code>action_list_to_string(action_list)</code>	Util function for turning an action list into pretty string
<code>cli_message_dict(dict_, indent=3, bullet_char='-', message_list=None, recursion_flag=False)</code>	Util function for displaying nested dicts representing ge objects in cli
<code>is_sane_slack_webhook(url)</code>	Really basic sanity checking.

`great_expectations.cli.util.colored`

`great_expectations.cli.util.cli_message(string)`

`great_expectations.cli.util.cli_colorize_string(string)`

`great_expectations.cli.util.cli_message_list(string_list, list_intro_string=None)`
Simple util function for displaying simple lists in cli

`great_expectations.cli.util.action_list_to_string(action_list)`
Util function for turning an action list into pretty string

`great_expectations.cli.util.cli_message_dict(dict_, indent=3, bullet_char='-', message_list=None, recursion_flag=False)`
Util function for displaying nested dicts representing ge objects in cli

`great_expectations.cli.util.is_sane_slack_webhook(url)`
Really basic sanity checking.

`great_expectations.cli.validation_operator`

Module Contents

Functions

<code>validation_operator()</code>	Validation Operator operations
<code>validation_operator_list(directory)</code>	List known Validation Operators.
<code>validation_operator_run(name, run_name, validation_config_file, suite, directory)</code>	Run a validation operator against some data.
<code>_validate_valdiation_config(valdiation_config)</code>	

`great_expectations.cli.validation_operator.json_parse_exception`

`great_expectations.cli.validation_operator.SQLAlchemyError`

`great_expectations.cli.validation_operator.validation_operator()`
Validation Operator operations

`great_expectations.cli.validation_operator.validation_operator_list(directory)`
List known Validation Operators.

`great_expectations.cli.validation_operator.validation_operator_run(name, run_name, validation_config_file, suite, directory)`
Run a validation operator against some data.

There are two modes to run this command:

1. Interactive (good for development):

Specify the name of the validation operator using the `--name` argument and the name of the expectation suite using the `--suite` argument.

The cli will help you specify the batch of data that you want to validate interactively.

2. Non-interactive (good for production):

Use the `--validation_config_file` argument to specify the path of the validation configuration JSON file. This file can be used to instruct a validation operator to validate multiple batches of data and use multiple expectation suites to validate each batch.

Learn how to create a validation config file here: https://great-expectations.readthedocs.io/en/latest/command_line.html#great-expectations-validation-operator-run-validation-config-file-validation-config-file-path

This command exits with 0 if the validation operator ran and the “success” attribute in its return object is True. Otherwise, the command exits with 1.

To learn more about validation operators, go here: <https://great-expectations.readthedocs.io/en/latest/features/validation.html#validation-operators>

```
great_expectations.cli.validation_operator._validate_valdiation_config(validation_config)
```

Package Contents

Functions

`cli`(None)

`main`()

```
great_expectations.cli.cli(verbose)
```

Welcome to the great_expectations CLI!

Most commands follow this format: `great_expectations <NOUN> <VERB>`

The nouns are: `datasource`, `docs`, `project`, `suite`, `validation-operator`

Most nouns accept the following verbs: `new`, `list`, `edit`

In particular, the CLI supports the following special commands:

- `great_expectations init` : create a new great_expectations project
- `great_expectations datasource profile` : profile a datasource
- `great_expectations docs build` : compile documentation from expectations

```
great_expectations.cli.main()
```

`great_expectations.core`

Subpackages

`great_expectations.core.usage_statistics`

Subpackages

`great_expectations.core.usage_statistics.anonymizers`

Submodules

`great_expectations.core.usage_statistics.anonymizers.action_anonymizer`

Module Contents

Classes

<code><i>ActionAnonymizer</i>(salt=None)</code>	Anonymize string names in an optionally-consistent way.
---	---

class `great_expectations.core.usage_statistics.anonymizers.action_anonymizer.ActionAnonymizer`

Bases: `great_expectations.core.usage_statistics.anonymizers.anonymizer.Anonymizer`

Anonymize string names in an optionally-consistent way.

anonymize_action_info (*self*, *action_name*, *action_obj*)

`great_expectations.core.usage_statistics.anonymizers.anonymizer`

Module Contents

Classes

<code><i>Anonymizer</i>(salt=None)</code>	Anonymize string names in an optionally-consistent way.
---	---

`great_expectations.core.usage_statistics.anonymizers.anonymizer.logger`

class `great_expectations.core.usage_statistics.anonymizers.anonymizer.Anonymizer` (*salt=None*)

Bases: `object`

Anonymize string names in an optionally-consistent way.

property **salt** (*self*)

anonymize (*self*, *string_*)

anonymize_object_info (*self*, *anonymized_info_dict*, *ge_classes*, *object_=None*, *object_class=None*, *object_config=None*)

`great_expectations.core.usage_statistics.anonymizers.batch_anonymizer`

Module Contents

Classes

<code>BatchAnonymizer(salt=None)</code>	Anonymize string names in an optionally-consistent way.
---	---

class `great_expectations.core.usage_statistics.anonymizers.batch_anonymizer.BatchAnonymizer`

Bases: `great_expectations.core.usage_statistics.anonymizers.anonymizer.Anonymizer`

Anonymize string names in an optionally-consistent way.

anonymize_batch_info (*self*, *batch*)

`great_expectations.core.usage_statistics.anonymizers.batch_kwargs_anonymizer`

Module Contents

Classes

<code>BatchKwargsAnonymizer(salt=None)</code>	Anonymize string names in an optionally-consistent way.
---	---

class `great_expectations.core.usage_statistics.anonymizers.batch_kwargs_anonymizer.BatchKwargsAnonymizer`

Bases: `great_expectations.core.usage_statistics.anonymizers.anonymizer.Anonymizer`

Anonymize string names in an optionally-consistent way.

anonymize_batch_kwargs (*self*, *batch_kwargs*)

`great_expectations.core.usage_statistics.anonymizers.data_docs_site_anonymizer`

Module Contents

Classes

<code>DataDocsSiteAnonymizer(salt=None)</code>	Anonymize string names in an optionally-consistent way.
--	---

class `great_expectations.core.usage_statistics.anonymizers.data_docs_site_anonymizer.DataDocsSiteAnonymizer`

Bases: `great_expectations.core.usage_statistics.anonymizers.anonymizer.Anonymizer`

Anonymize string names in an optionally-consistent way.

anonymize_data_docs_site_info (*self*, *site_name*, *site_config*)

`great_expectations.core.usage_statistics.anonymizers.datasource_anonymizer`

Module Contents

Classes

<i><code>DatasourceAnonymizer</code></i> (salt=None)	Anonymize string names in an optionally-consistent way.
--	---

class `great_expectations.core.usage_statistics.anonymizers.datasource_anonymizer.DatasourceAnonymizer`
 Bases: `great_expectations.core.usage_statistics.anonymizers.anonymizer.Anonymizer`

Anonymize string names in an optionally-consistent way.

`anonymize_datasource_info` (*self*, *name*, *config*)

`great_expectations.core.usage_statistics.anonymizers.expectation_suite_anonymizer`

Module Contents

Classes

<i><code>ExpectationSuiteAnonymizer</code></i> (salt=None)	Anonymize string names in an optionally-consistent way.
--	---

`great_expectations.core.usage_statistics.anonymizers.expectation_suite_anonymizer.GE_EXPECTATION_SUITE_ANONYMIZER`
class `great_expectations.core.usage_statistics.anonymizers.expectation_suite_anonymizer.ExpectationSuiteAnonymizer`
 Bases: `great_expectations.core.usage_statistics.anonymizers.anonymizer.Anonymizer`

Anonymize string names in an optionally-consistent way.

`anonymize_expectation_suite_info` (*self*, *expectation_suite*)

`great_expectations.core.usage_statistics.anonymizers.site_builder_anonymizer`

Module Contents

Classes

<i><code>SiteBuilderAnonymizer</code></i> (salt=None)	Anonymize string names in an optionally-consistent way.
---	---

class `great_expectations.core.usage_statistics.anonymizers.site_builder_anonymizer.SiteBuilderAnonymizer`
 Bases: `great_expectations.core.usage_statistics.anonymizers.anonymizer.Anonymizer`

Anonymize string names in an optionally-consistent way.

`anonymize_site_builder_info` (*self*, *site_builder_config*)

`great_expectations.core.usage_statistics.anonymizers.store_anonymizer`

Module Contents

Classes

<code>StoreAnonymizer(salt=None)</code>	Anonymize string names in an optionally-consistent way.
---	---

class `great_expectations.core.usage_statistics.anonymizers.store_anonymizer.StoreAnonymizer`

Bases: `great_expectations.core.usage_statistics.anonymizers.anonymizer.Anonymizer`

Anonymize string names in an optionally-consistent way.

anonymize_store_info (*self*, *store_name*, *store_obj*)

`great_expectations.core.usage_statistics.anonymizers.store_backend_anonymizer`

Module Contents

Classes

<code>StoreBackendAnonymizer(salt=None)</code>	Anonymize string names in an optionally-consistent way.
--	---

class `great_expectations.core.usage_statistics.anonymizers.store_backend_anonymizer.StoreBackendAnonymizer`

Bases: `great_expectations.core.usage_statistics.anonymizers.anonymizer.Anonymizer`

Anonymize string names in an optionally-consistent way.

anonymize_store_backend_info (*self*, *store_backend_obj*=None, *store_backend_object_config*=None)

`great_expectations.core.usage_statistics.anonymizers.validation_operator_anonymizer`

Module Contents

Classes

<code>ValidationOperatorAnonymizer(salt=None)</code>	Anonymize string names in an optionally-consistent way.
--	---

class `great_expectations.core.usage_statistics.anonymizers.validation_operator_anonymizer.ValidationOperatorAnonymizer`

Bases: `great_expectations.core.usage_statistics.anonymizers.anonymizer.Anonymizer`

Anonymize string names in an optionally-consistent way.

anonymize_validation_operator_info (*self*, *validation_operator_name*, *validation_operator_obj*)

Submodules

`great_expectations.core.usage_statistics.schemas`

Module Contents

`great_expectations.core.usage_statistics.schemas.anonymized_string_schema`
`great_expectations.core.usage_statistics.schemas.anonymized_datasource_schema`
`great_expectations.core.usage_statistics.schemas.anonymized_class_info_schema`
`great_expectations.core.usage_statistics.schemas.anonymized_store_schema`
`great_expectations.core.usage_statistics.schemas.anonymized_action_schema`
`great_expectations.core.usage_statistics.schemas.anonymized_validation_operator_schema`
`great_expectations.core.usage_statistics.schemas.empty_payload_schema`
`great_expectations.core.usage_statistics.schemas.anonymized_data_docs_site_schema`
`great_expectations.core.usage_statistics.schemas.anonymized_expectation_suite_schema`
`great_expectations.core.usage_statistics.schemas.init_payload_schema`
`great_expectations.core.usage_statistics.schemas.anonymized_batch_schema`
`great_expectations.core.usage_statistics.schemas.run_validation_operator_payload_schema`
`great_expectations.core.usage_statistics.schemas.save_or_edit_expectation_suite_payload_schema`
`great_expectations.core.usage_statistics.schemas.cli_new_ds_choice_payload`
`great_expectations.core.usage_statistics.schemas.datasource_sqlalchemy_connect_payload`
`great_expectations.core.usage_statistics.schemas.usage_statistics_record_schema`

`great_expectations.core.usage_statistics.usage_statistics`

Module Contents

Classes

`UsageStatisticsHandler(data_context,`
`data_context_id, usage_statistics_url)`

Functions

`get_usage_statistics_handler(args_array)`
`usage_statistics_enabled_method(func=None,` A decorator for usage statistics which defaults to the less
`event_name=None, args_payload_fn=None, re-` detailed payload schema.
`sult_payload_fn=None)`

continues on next page

Table 32 – continued from previous page

<code>run_validation_operator_usage_statistics</code>	<code>(data_context,</code>
<code>validation_operator_name,</code>	<code>assets_to_validate,</code>
<code>run_id=None, **kwargs)</code>	
<code>save_expectation_suite_usage_statistics</code>	<code>(data_context,</code>
<code>expectation_suite, expectation_suite_name=None)</code>	
<code>edit_expectation_suite_usage_statistics</code>	<code>(data_context,</code>
<code>expectation_suite_name)</code>	
<code>add_datasource_usage_statistics</code>	<code>(data_context,</code>
<code>name, **kwargs)</code>	
<code>send_usage_message</code>	<code>(data_context, event, send a usage statistics message.</code>
<code>event_payload=None, success=None)</code>	

`great_expectations.core.usage_statistics.usage_statistics.STOP_SIGNAL`

`great_expectations.core.usage_statistics.usage_statistics.logger`

`great_expectations.core.usage_statistics.usage_statistics._anonymizers`

class `great_expectations.core.usage_statistics.usage_statistics.UsageStatisticsHandler` (`data_`
`data_`
`us-`
`age_`

Bases: `object`

`_teardown` (`self, signum: int, frame`)

`_close_worker` (`self`)

`_requests_worker` (`self`)

`send_usage_message` (`self, event, event_payload=None, success=None`)
 send a usage statistics message.

`build_init_payload` (`self`)
 Adds information that may be available only after full data context construction, but is useful to calculate
 only one time (for example, anonymization).

`build_envelope` (`self, message`)

`validate_message` (`self, message, schema`)

`emit` (`self, message`)
 Emit a message.

`great_expectations.core.usage_statistics.usage_statistics.get_usage_statistics_handler` (`args,`

`great_expectations.core.usage_statistics.usage_statistics.usage_statistics_enabled_method` (`f`

A decorator for usage statistics which defaults to the less detailed payload schema.

`great_expectations.core.usage_statistics.usage_statistics.run_validation_operator_usage_stat`

`great_expectations.core.usage_statistics.usage_statistics.save_expectation_suite_usage_stat`

`great_expectations.core.usage_statistics.usage_statistics.edit_expectation_suite_usage_stat`

`great_expectations.core.usage_statistics.usage_statistics.add_datasource_usage_statistics (data`

`great_expectations.core.usage_statistics.usage_statistics.send_usage_message (data_context, event, event_payload=None, success=None)`

send a usage statistics message.

Submodules

`great_expectations.core.batch`

Module Contents

Classes

<code>Batch</code>	<code>(datasource_name, batch_kwargs, data, batch_parameters, batch_markers, data_context)</code>
<code>class great_expectations.core.batch.Batch</code>	<code>(datasource_name, batch_kwargs, data, batch_parameters, batch_markers, data_context)</code>
<code>Bases:</code>	<code>great_expectations.types.DictDot</code>
<code>property datasource_name</code>	<code>(self)</code>


```

property batch_kwargs (self)
property data (self)
property batch_parameters (self)
property batch_markers (self)
property data_context (self)

```

`great_expectations.core.data_context_key`

Module Contents

Classes

<code>DataContextKey()</code>	DataContextKey objects are used to uniquely identify resources used by the DataContext.
<code>StringKey(key)</code>	A simple DataContextKey with just a single string value

class `great_expectations.core.data_context_key.DataContextKey`

Bases: `object`

DataContextKey objects are used to uniquely identify resources used by the DataContext.

A DataContextKey is designed to support clear naming with multiple representations including a hashable version making it suitable for use as the key in a dictionary.

```

abstract to_tuple (self)
classmethod from_tuple (cls, tuple_)
abstract to_fixed_length_tuple (self)
abstract classmethod from_fixed_length_tuple (cls, tuple_)
__eq__ (self, other)
    Return self==value.
__ne__ (self, other)
    Return self!=value.
__hash__ (self)
    Return hash(self).
__repr__ (self)
    Return repr(self).

```

class `great_expectations.core.data_context_key.StringKey(key)`

Bases: `great_expectations.core.data_context_key.DataContextKey`

A simple DataContextKey with just a single string value

```

to_tuple (self)
to_fixed_length_tuple (self)
classmethod from_fixed_length_tuple (cls, tuple_)

```

great_expectations.core.evaluation_parameters

Module Contents

Classes

<code>EvaluationParameterParser()</code>	This Evaluation Parameter Parser uses pyparsing to provide a basic expression language capable of evaluating
--	--

Functions

<code>build_evaluation_parameters(expectation_args, evaluation_parameters=None, interactive_evaluation=True, data_context=None)</code>	Build a dictionary of parameters to evaluate, using the provided evaluation_parameters,
<code>find_evaluation_parameter_dependencies(parameter_expression)</code>	Parse a parameter expression to identify dependencies including GE URNs.
<code>parse_evaluation_parameter(parameter_expression, evaluation_parameters=None, data_context=None)</code>	Use the provided evaluation_parameters dict to parse a given parameter expression.

great_expectations.core.evaluation_parameters.logger

great_expectations.core.evaluation_parameters._epsilon = 1e-12

class great_expectations.core.evaluation_parameters.EvaluationParameterParser

This Evaluation Parameter Parser uses pyparsing to provide a basic expression language capable of evaluating parameters using values available only at run time.

expop :: '^' multop :: '*' | '/' addop :: '+' | '-' integer :: ['+' | '-'] '0'..'9'+ atom :: PI | E | real | fn '(' expr ')' | '(' expr ')' factor :: atom [expop factor]* term :: factor [multop factor]* expr :: term [addop term]*

The parser is modified from: <https://github.com/pyparsing/pyparsing/blob/master/examples/fourFn.py>

opn

fn

push_first (self, toks)

push_unary_minus (self, toks)

clear_stack (self)

get_parser (self)

evaluate_stack (self, s)

great_expectations.core.evaluation_parameters.build_evaluation_parameters (expectation_args, evaluation_parameters=None, interactive_evaluation=True, data_context=None)

Build a dictionary of parameters to evaluate, using the provided `evaluation_parameters`, AND mutate `expectation_args` by removing any parameter values passed in as temporary values during exploratory work.

`great_expectations.core.evaluation_parameters.expr`

`great_expectations.core.evaluation_parameters.find_evaluation_parameter_dependencies` (*parameter_expression*)

Parse a parameter expression to identify dependencies including GE URNs.

Parameters `parameter_expression` – the parameter to parse

Returns

- “urns”: set of strings that are valid GE URN objects
- “other”: set of non-GE URN strings that are required to evaluate the parameter expression

Return type a dictionary including

`great_expectations.core.evaluation_parameters.parse_evaluation_parameter` (*parameter_expression*, *evaluation_parameters=None*, *data_context=None*)

Use the provided `evaluation_parameters` dict to parse a given parameter expression.

Parameters

- **parameter_expression** (*str*) – A string, potentially containing basic arithmetic operations and functions, and variables to be substituted
- **evaluation_parameters** (*dict*) – A dictionary of name-value pairs consisting of values to substitute
- **data_context** (*DataContext*) – A data context to use to obtain metrics, if necessary

The parser will allow arithmetic operations `+`, `-`, `/`, `*`, as well as basic functions, including `trunc()` and `round()` to obtain integer values when needed for certain expectations (e.g. `expect_column_value_length_to_be_between`).

Valid variables must begin with an alphabetic character and may contain alphanumeric characters plus ‘`_`’ and ‘`$`’, EXCEPT if they begin with the string “`urn:great_expectations`” in which case they may also include additional characters to support inclusion of GE URLs (see [Evaluation Parameters](#) for more information).

`great_expectations.core.id_dict`

Module Contents

Classes

<code>IDDict()</code>	<code>dict()</code> -> new empty dictionary
<code>BatchKwargs()</code>	<code>dict()</code> -> new empty dictionary
<code>MetricKwargs()</code>	<code>dict()</code> -> new empty dictionary

class `great_expectations.core.id_dict.IDDict`

Bases: `dict`

`dict()` -> new empty dictionary `dict(mapping)` -> new dictionary initialized from a mapping object’s

(key, value) pairs

dict(iterable) -> new dictionary initialized as if via: `d = { }` for `k, v` in `iterable`:

`d[k] = v`

dict(kwargs) -> new dictionary initialized with the name=value pairs** in the keyword argument list. For example: `dict(one=1, two=2)`

`_id_ignore_keys`

`to_id` (*self*, *id_keys=None*, *id_ignore_keys=None*)

class `great_expectations.core.id_dict.BatchKwargs`

Bases: `great_expectations.core.id_dict.IDDict`

`dict()` -> new empty dictionary `dict(mapping)` -> new dictionary initialized from a mapping object's

(key, value) pairs

dict(iterable) -> new dictionary initialized as if via: `d = { }` for `k, v` in `iterable`:

`d[k] = v`

dict(kwargs) -> new dictionary initialized with the name=value pairs** in the keyword argument list. For example: `dict(one=1, two=2)`

class `great_expectations.core.id_dict.MetricKwargs`

Bases: `great_expectations.core.id_dict.IDDict`

`dict()` -> new empty dictionary `dict(mapping)` -> new dictionary initialized from a mapping object's

(key, value) pairs

dict(iterable) -> new dictionary initialized as if via: `d = { }` for `k, v` in `iterable`:

`d[k] = v`

dict(kwargs) -> new dictionary initialized with the name=value pairs** in the keyword argument list. For example: `dict(one=1, two=2)`

`great_expectations.core.metric`

Module Contents

Classes

<code>Metric</code> (<i>metric_name</i> , <i>metric_kwargs</i> , <i>metric_value</i>)	A Metric associates a value with some name and configuration. The specific configuration parameters that are
<code>MetricIdentifier</code> (<i>metric_name</i> , <i>metric_kwargs_id</i>)	A MetricIdentifier serves as a key to store and retrieve Metrics.
<code>BatchMetric</code> (<i>metric_name</i> , <i>metric_kwargs</i> , <i>batch_identifier</i> , <i>metric_value</i>)	A BatchMetric is a metric associated with a particular Batch of data.
<code>ValidationMetric</code> (<i>run_id</i> , <i>data_asset_name</i> , <i>expectation_suite_identifier</i> , <i>metric_name</i> , <i>metric_kwargs</i> , <i>metric_value</i>)	A Metric associates a value with some name and configuration. The specific configuration parameters that are

continues on next page

Table 38 – continued from previous page

<code>ValidationMetricIdentifier</code>	<code>(run_id, data_asset_name, expectation_suite_identifier, metric_name, metric_kwargs_id)</code>	A <code>MetricIdentifier</code> serves as a key to store and retrieve Metrics.
<hr/>		
class	<code>great_expectations.core.metric.Metric</code>	<code>(metric_name, metric_kwargs, metric_value)</code>
	Bases: <code>object</code>	
	A Metric associates a value with some name and configuration. The specific configuration parameters that are relevant for a given metric's identity depend on the metric. For example, the metric <code>column_mean</code> depends on a column name.	
	property <code>metric_name</code> (<i>self</i>)	
	property <code>metric_kwargs</code> (<i>self</i>)	
	property <code>metric_kwargs_id</code> (<i>self</i>)	
class	<code>great_expectations.core.metric.MetricIdentifier</code>	<code>(metric_name, metric_kwargs_id)</code>
	Bases: <code>great_expectations.core.data_context_key.DataContextKey</code>	
	A <code>MetricIdentifier</code> serves as a key to store and retrieve Metrics.	
	property <code>metric_name</code> (<i>self</i>)	
	property <code>metric_kwargs_id</code> (<i>self</i>)	
	classmethod <code>from_object</code> (<i>cls</i> , <i>metric</i>)	
	to_fixed_length_tuple (<i>self</i>)	
	to_tuple (<i>self</i>)	
	classmethod <code>from_fixed_length_tuple</code> (<i>cls</i> , <i>tuple_</i>)	
	classmethod <code>from_tuple</code> (<i>cls</i> , <i>tuple_</i>)	
class	<code>great_expectations.core.metric.BatchMetric</code>	<code>(metric_name, metric_kwargs, batch_identifier, metric_value)</code>
	Bases: <code>great_expectations.core.metric.Metric</code>	
	A <code>BatchMetric</code> is a metric associated with a particular Batch of data.	
	property <code>batch_identifier</code> (<i>self</i>)	
class	<code>great_expectations.core.metric.ValidationMetric</code>	<code>(run_id, data_asset_name, expectation_suite_identifier, metric_name, metric_kwargs, metric_value)</code>
	Bases: <code>great_expectations.core.metric.Metric</code>	
	A Metric associates a value with some name and configuration. The specific configuration parameters that are relevant for a given metric's identity depend on the metric. For example, the metric <code>column_mean</code> depends on a column name.	
	property <code>run_id</code> (<i>self</i>)	
	property <code>data_asset_name</code> (<i>self</i>)	
	property <code>expectation_suite_identifier</code> (<i>self</i>)	

```
class great_expectations.core.metric.ValidationMetricIdentifier(run_id,  
                                                             data_asset_name,  
                                                             expecta-  
                                                             tion_suite_identifier,  
                                                             metric_name,  
                                                             met-  
                                                             ric_kwargs_id)
```

Bases: *great_expectations.core.metric.MetricIdentifier*

A MetricIdentifier serves as a key to store and retrieve Metrics.

```
property run_id(self)  
property data_asset_name(self)  
property expectation_suite_identifier(self)  
classmethod from_object(cls, validation_metric)  
to_tuple(self)  
to_fixed_length_tuple(self)  
to_evaluation_parameter_urn(self)  
classmethod from_tuple(cls, tuple_)  
classmethod from_fixed_length_tuple(cls, tuple_)
```

great_expectations.core.urn

Module Contents

```
great_expectations.core.urn.urn_word  
great_expectations.core.urn.ge_metrics_urn  
great_expectations.core.urn.ge_validations_urn  
great_expectations.core.urn.ge_stores_urn  
great_expectations.core.urn.ge_urn
```

great_expectations.core.util

Module Contents

Functions

```
nested_update(d, u)
```

```
great_expectations.core.util.nested_update(d, u)
```

Package Contents

Classes

<i>DataContextKey()</i>	DataContextKey objects are used to uniquely identify resources used by the DataContext.
<i>IDDict()</i>	dict() -> new empty dictionary
<i>DictDot()</i>	
<i>RunIdentifier</i> (run_name=None, run_time=None)	A RunIdentifier identifies a run (collection of validations) by run_name and run_time.
<i>RunIdentifierSchema</i> (*, only: types.StrSequenceOrSet = None, exclude: types.StrSequenceOrSet = (), many: bool = False, context: typing.Dict = None, load_only: types.StrSequenceOrSet = (), dump_only: types.StrSequenceOrSet = (), partial: typing.Union[bool, types.StrSequenceOrSet] = False, unknown: str = None)	Base schema class with which to define custom schemas.
<i>ExpectationKwargs</i> (*args, **kwargs)	dict() -> new empty dictionary
<i>ExpectationConfiguration</i> (expectation_type, kwargs, meta=None, success_on_last_run=None)	ExpectationConfiguration defines the parameters and name of a specific expectation.
<i>ExpectationConfigurationSchema</i> (*, only: types.StrSequenceOrSet = None, exclude: types.StrSequenceOrSet = (), many: bool = False, context: typing.Dict = None, load_only: types.StrSequenceOrSet = (), dump_only: types.StrSequenceOrSet = (), partial: typing.Union[bool, types.StrSequenceOrSet] = False, unknown: str = None)	Base schema class with which to define custom schemas.
<i>ExpectationSuite</i> (expectation_suite_name, expectations=None, evaluation_parameters=None, data_asset_type=None, meta=None)	
<i>ExpectationSuiteSchema</i> (*, only: types.StrSequenceOrSet = None, exclude: types.StrSequenceOrSet = (), many: bool = False, context: typing.Dict = None, load_only: types.StrSequenceOrSet = (), dump_only: types.StrSequenceOrSet = (), partial: typing.Union[bool, types.StrSequenceOrSet] = False, unknown: str = None)	Base schema class with which to define custom schemas.
<i>ExpectationValidationResult</i> (success=None, expectation_config=None, result=None, meta=None, exception_info=None)	
<i>ExpectationValidationResultSchema</i> (*, only: types.StrSequenceOrSet = None, exclude: types.StrSequenceOrSet = (), many: bool = False, context: typing.Dict = None, load_only: types.StrSequenceOrSet = (), dump_only: types.StrSequenceOrSet = (), partial: typing.Union[bool, types.StrSequenceOrSet] = False, unknown: str = None)	Base schema class with which to define custom schemas.
<i>ExpectationSuiteValidationResult</i> (success=None, results=None, evaluation_parameters=None, statistics=None, meta=None)	

continues on next page

Table 40 – continued from previous page

<i>ExpectationSuiteValidationResultSchema</i> (*,	Base schema class with which to define custom schemas.
only: types.StrSequenceOrSet = None, exclude:	
types.StrSequenceOrSet = (), many: bool = False, context:	
typing.Dict = None, load_only: types.StrSequenceOrSet	
= (), dump_only: types.StrSequenceOrSet = (), partial:	
typing.Union[bool, types.StrSequenceOrSet] = False,	
unknown: str = None)	

Functions

<i>find_evaluation_parameter_dependencies</i> (parameter_expression)	Parse a parameter expression to identify dependencies including GE URNs.
<i>nested_update</i> (d, u)	
<i>in_jupyter_notebook</i> ()	
<i>get_metric_kwargs_id</i> (metric_name, metric_kwargs)	
<i>convert_to_json_serializable</i> (data)	Helper function to convert an object to one that is json serializable
<i>ensure_json_serializable</i> (data)	Helper function to convert an object to one that is json serializable
<i>_deduplicate_evaluation_parameter_dependencies</i> (dependencies)	

great_expectations.core.ge_version

class great_expectations.core.DataContextKey

Bases: object

DataContextKey objects are used to uniquely identify resources used by the DataContext.

A DataContextKey is designed to support clear naming with multiple representations including a hashable version making it suitable for use as the key in a dictionary.

abstract to_tuple (self)

classmethod from_tuple (cls, tuple_)

abstract to_fixed_length_tuple (self)

abstract classmethod from_fixed_length_tuple (cls, tuple_)

__eq__ (self, other)
Return self==value.

__ne__ (self, other)
Return self!=value.

__hash__ (self)
Return hash(self).

__repr__ (self)
Return repr(self).

great_expectations.core.**find_evaluation_parameter_dependencies** (parameter_expression)
Parse a parameter expression to identify dependencies including GE URNs.

Parameters parameter_expression – the parameter to parse

Returns

- “urns”: set of strings that are valid GE URN objects
- “other”: set of non-GE URN strings that are required to evaluate the parameter expression

Return type a dictionary including

class `great_expectations.core.IDDict`

Bases: `dict`

`dict()` -> new empty dictionary `dict(mapping)` -> new dictionary initialized from a mapping object's

(key, value) pairs

dict(iterable) -> new dictionary initialized as if via: `d = { } for k, v in iterable:`

`d[k] = v`

dict(kwargs)** -> new dictionary initialized with the name=value pairs in the keyword argument list. For example: `dict(one=1, two=2)`

`_id_ignore_keys`

`to_id` (*self*, *id_keys=None*, *id_ignore_keys=None*)

`great_expectations.core.ge_urn`

`great_expectations.core.nested_update` (*d*, *u*)

exception `great_expectations.core.InvalidCacheValueError` (*result_dict*)

Bases: `great_expectations.exceptions.GreatExpectationsError`

Common base class for all non-exit exceptions.

exception `great_expectations.core.InvalidExpectationConfigurationError` (*message*)

Bases: `great_expectations.exceptions.GreatExpectationsError`

Common base class for all non-exit exceptions.

exception `great_expectations.core.InvalidExpectationKwargsError` (*message*)

Bases: `great_expectations.exceptions.GreatExpectationsError`

Common base class for all non-exit exceptions.

exception `great_expectations.core.ParserError` (*message*)

Bases: `great_expectations.exceptions.GreatExpectationsError`

Common base class for all non-exit exceptions.

exception `great_expectations.core.UnavailableMetricError` (*message*)

Bases: `great_expectations.exceptions.GreatExpectationsError`

Common base class for all non-exit exceptions.

class `great_expectations.core.DictDot`

Bases: `object`

`__getitem__` (*self*, *item*)

`__setitem__` (*self*, *key*, *value*)

`__delitem__` (*self*, *key*)

`great_expectations.core.logger`

`great_expectations.core.RESULT_FORMATS` = ['BOOLEAN_ONLY', 'BASIC', 'COMPLETE', 'SUMMARY']

`great_expectations.core.EvaluationParameterIdentifier`

`great_expectations.core.in_jupyter_notebook()`

`great_expectations.core.get_metric_kwargs_id(metric_name, metric_kwargs)`

`great_expectations.core.convert_to_json_serializable(data)`

Helper function to convert an object to one that is json serializable

Parameters `data` – an object to attempt to convert a corresponding json-serializable object

Returns (dict) A converted test_object

Warning: test_obj may also be converted in place.

`great_expectations.core.ensure_json_serializable(data)`

Helper function to convert an object to one that is json serializable

Parameters `data` – an object to attempt to convert a corresponding json-serializable object

Returns (dict) A converted test_object

Warning: test_obj may also be converted in place.

class `great_expectations.core.RunIdentifier` (*run_name=None, run_time=None*)

Bases: `great_expectations.core.data_context_key.DataContextKey`

A RunIdentifier identifies a run (collection of validations) by run_name and run_time.

property `run_name` (*self*)

property `run_time` (*self*)

to_tuple (*self*)

to_fixed_length_tuple (*self*)

__repr__ (*self*)
Return repr(self).

__str__ (*self*)
Return str(self).

to_json_dict (*self*)

classmethod `from_tuple` (*cls, tuple_*)

classmethod `from_fixed_length_tuple` (*cls, tuple_*)

class `great_expectations.core.RunIdentifierSchema` (*, *only: types.StrSequenceOrSet*
= *None*, *exclude:*
types.StrSequenceOrSet = *()*,
many: bool = *False*, *con-*
text: typing.Dict = *None*,
load_only: types.StrSequenceOrSet
= *()*, *dump_only:*
types.StrSequenceOrSet = *()*,
partial: typing.Union[bool,
types.StrSequenceOrSet] = *False*,
unknown: str = *None*)

Bases: `marshmallow.Schema`

Base schema class with which to define custom schemas.

Example usage:

```
import datetime as dt
from dataclasses import dataclass

from marshmallow import Schema, fields

@dataclass
class Album:
    title: str
    release_date: dt.date

class AlbumSchema(Schema):
    title = fields.Str()
    release_date = fields.Date()

album = Album("Beggars Banquet", dt.date(1968, 12, 6))
schema = AlbumSchema()
data = schema.dump(album)
data # {'release_date': '1968-12-06', 'title': 'Beggars Banquet'}
```

Parameters

- **only** – Whitelist of the declared fields to select when instantiating the Schema. If None, all fields are used. Nested fields can be represented with dot delimiters.
- **exclude** – Blacklist of the declared fields to exclude when instantiating the Schema. If a field appears in both *only* and *exclude*, it is not used. Nested fields can be represented with dot delimiters.
- **many** – Should be set to *True* if *obj* is a collection so that the object will be serialized to a list.
- **context** – Optional context passed to `fields.Method` and `fields.Function` fields.
- **load_only** – Fields to skip during serialization (write-only fields)
- **dump_only** – Fields to skip during deserialization (read-only fields)
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to *Nested* fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.
- **unknown** – Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*.

Changed in version 3.0.0: *prefix* parameter removed.

Changed in version 2.0.0: `__validators__`, `__preprocessors__`, and `__data_handlers__` are removed in favor of `marshmallow.decorators.validates_schema`, `marshmallow.decorators.pre_load` and `marshmallow.decorators.post_dump`. `__accessor__` and `__error_handler__` are deprecated. Implement the `handle_error` and `get_attribute` methods instead.

run_name

run_time

make_run_identifier (*self*, *data*, ***kwargs*)

class great_expectations.core.**ExpectationKwargs** (**args*, ***kwargs*)

Bases: dict

dict() -> new empty dictionary dict(mapping) -> new dictionary initialized from a mapping object's

(key, value) pairs

dict(iterable) -> new dictionary initialized as if via: d = { } for k, v in iterable:

d[k] = v

dict(kwargs)** -> new dictionary initialized with the name=value pairs in the keyword argument list. For example: dict(one=1, two=2)

ignored_keys = ['result_format', 'include_config', 'catch_exceptions']

ExpectationKwargs store information necessary to evaluate an expectation.

isEquivalentTo (*self*, *other*)

__repr__ (*self*)

Return repr(self).

__str__ (*self*)

Return str(self).

to_json_dict (*self*)

great_expectations.core.**_deduplicate_evaluation_parameter_dependencies** (*dependencies*)

class great_expectations.core.**ExpectationConfiguration** (*expectation_type*, *kwargs*,
meta=None, *success_on_last_run=None*)

Bases: *great_expectations.types.DictDot*

ExpectationConfiguration defines the parameters and name of a specific expectation.

property expectation_type (*self*)

property kwargs (*self*)

isEquivalentTo (*self*, *other*)

ExpectationConfiguration equivalence does not include meta, and relies on *equivalence* of kwargs.

__eq__ (*self*, *other*)

ExpectationConfiguration equality does include meta, but ignores instance identity.

__ne__ (*self*, *other*)

Return self!=value.

__repr__ (*self*)

Return repr(self).

__str__ (*self*)

Return str(self).

to_json_dict (*self*)

get_evaluation_parameter_dependencies (*self*)

```
class great_expectations.core.ExpectationConfigurationSchema(*,
                                                             only:
                                                             types.StrSequenceOrSet
                                                             = None, exclude:
                                                             types.StrSequenceOrSet
                                                             = (), many: bool
                                                             = False, context:
                                                             typing.Dict =
                                                             None, load_only:
                                                             types.StrSequenceOrSet
                                                             = (), dump_only:
                                                             types.StrSequenceOrSet
                                                             = (), partial: typing.Union[bool,
                                                             types.StrSequenceOrSet]
                                                             = False, unknown:
                                                             str = None)
```

Bases: `marshmallow.Schema`

Base schema class with which to define custom schemas.

Example usage:

```
import datetime as dt
from dataclasses import dataclass

from marshmallow import Schema, fields

@dataclass
class Album:
    title: str
    release_date: dt.date

class AlbumSchema(Schema):
    title = fields.Str()
    release_date = fields.Date()

album = Album("Beggars Banquet", dt.date(1968, 12, 6))
schema = AlbumSchema()
data = schema.dump(album)
data # {'release_date': '1968-12-06', 'title': 'Beggars Banquet'}
```

Parameters

- **only** – Whitelist of the declared fields to select when instantiating the Schema. If None, all fields are used. Nested fields can be represented with dot delimiters.
- **exclude** – Blacklist of the declared fields to exclude when instantiating the Schema. If a field appears in both *only* and *exclude*, it is not used. Nested fields can be represented with dot delimiters.
- **many** – Should be set to *True* if *obj* is a collection so that the object will be serialized to a list.
- **context** – Optional context passed to `fields.Method` and `fields.Function` fields.

- **load_only** – Fields to skip during serialization (write-only fields)
- **dump_only** – Fields to skip during deserialization (read-only fields)
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to *Nested* fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.
- **unknown** – Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*.

Changed in version 3.0.0: *prefix* parameter removed.

Changed in version 2.0.0: *__validators__*, *__preprocessors__*, and *__data_handlers__* are removed in favor of *marshmallow.decorators.validates_schema*, *marshmallow.decorators.pre_load* and *marshmallow.decorators.post_dump*. *__accessor__* and *__error_handler__* are deprecated. Implement the *handle_error* and *get_attribute* methods instead.

expectation_type

kwargs

meta

make_expectation_configuration (*self*, *data*, ***kwargs*)

```
class great_expectations.core.ExpectationSuite (expectation_suite_name,          ex-
                                                  pections=None,          evalu-
                                                  ation_parameters=None,
                                                  data_asset_type=None, meta=None)
```

Bases: object

add_citation (*self*, *comment*, *batch_kwargs=None*, *batch_markers=None*, *batch_parameters=None*, *citation_date=None*)

isEquivalentTo (*self*, *other*)

ExpectationSuite equivalence relies only on expectations and evaluation parameters. It does not include: -
data_asset_name - expectation_suite_name - meta - data_asset_type

__eq__ (*self*, *other*)

ExpectationSuite equality ignores instance identity, relying only on properties.

__ne__ (*self*, *other*)

Return self!=value.

__repr__ (*self*)

Return repr(self).

__str__ (*self*)

Return str(self).

to_json_dict (*self*)

get_evaluation_parameter_dependencies (*self*)

get_citations (*self*, *sort=True*, *require_batch_kwargs=False*)

get_table_expectations (*self*)

Return a list of table expectations.

get_column_expectations (*self*)

Return a list of column map expectations.

static _filter_citations (*citations*, *filter_key*)

static _sort_citations (*citations*)

_copy_and_clean_up_expectation (*self*, *expectation*, *discard_result_format_kwargs=True*,
discard_include_config_kwargs=True, *discard_catch_exceptions_kwargs=True*)

Returns copy of *expectation* without *success_on_last_run* and other specified key-value pairs removed

Returns a copy of specified expectation will not have *success_on_last_run* key-value. The other key-value pairs will be removed by default but will remain in the copy if specified.

Parameters

- **expectation** (*json*) – The expectation to copy and clean.
- **discard_result_format_kwargs** (*boolean*) – if True, will remove the kwarg *output_format* key-value pair from the copied expectation.
- **discard_include_config_kwargs** (*boolean*) – if True, will remove the kwarg *include_config* key-value pair from the copied expectation.
- **discard_catch_exceptions_kwargs** (*boolean*) – if True, will remove the kwarg *catch_exceptions* key-value pair from the copied expectation.

Returns A copy of the provided expectation with *success_on_last_run* and other specified key-value pairs removed

Note: This method may move to `ExpectationConfiguration`, minus the “copy” part.

_copy_and_clean_up_expectations_from_indexes (*self*, *match_indexes*, *discard_result_format_kwargs=True*,
discard_include_config_kwargs=True, *discard_catch_exceptions_kwargs=True*)

Copies and cleans all expectations provided by their index in `DataAsset._expectation_suite.expectations`.

Applies the `_copy_and_clean_up_expectation` method to multiple expectations, provided by their index in `DataAsset._expectation_suite.expectations`. Returns a list of the copied and cleaned expectations.

Parameters

- **match_indexes** (*List*) – Index numbers of the expectations from *expectation_config.expectations* to be copied and cleaned.
- **discard_result_format_kwargs** (*boolean*) – if True, will remove the kwarg *output_format* key-value pair from the copied expectation.
- **discard_include_config_kwargs** (*boolean*) – if True, will remove the kwarg *include_config* key-value pair from the copied expectation.
- **discard_catch_exceptions_kwargs** (*boolean*) – if True, will remove the kwarg *catch_exceptions* key-value pair from the copied expectation.

Returns A list of the copied expectations with *success_on_last_run* and other specified key-value pairs removed.

See also:

`_copy_and_clean_expectation`

append_expectation (*self*, *expectation_config*)

Appends an expectation.

Parameters **expectation_config** (`ExpectationConfiguration`) – The expectation to be added to the list.

Notes

May want to add type-checking in the future.

find_expectation_indexes (*self*, *expectation_type=None*, *column=None*, *expectation_kwargs=None*)

Find matching expectations and return their indexes. :param expectation_type=None: The name of the expectation type to be matched. :param column=None: The name of the column to be matched. :param expectation_kwargs=None: A dictionary of kwargs to match against.

Returns A list of indexes for matching expectation objects. If there are no matches, the list will be empty.

find_expectations (*self*, *expectation_type=None*, *column=None*, *expectation_kwargs=None*, *discard_result_format_kwargs=True*, *discard_include_config_kwargs=True*, *discard_catch_exceptions_kwargs=True*)

Find matching expectations and return them. :param expectation_type=None: The name of the expectation type to be matched. :param column=None: The name of the column to be matched. :param expectation_kwargs=None: A dictionary of kwargs to match against. :param discard_result_format_kwargs=True: In returned expectation object(s), suppress the *result_format* parameter. :param discard_include_config_kwargs=True: In returned expectation object(s), suppress the *include_config* parameter. :param discard_catch_exceptions_kwargs=True: In returned expectation object(s), suppress the *catch_exceptions* parameter.

Returns A list of matching expectation objects. If there are no matches, the list will be empty.

remove_expectation (*self*, *expectation_type=None*, *column=None*, *expectation_kwargs=None*, *remove_multiple_matches=False*, *dry_run=False*)

Remove matching expectation(s). :param expectation_type=None: The name of the expectation type to be matched. :param column=None: The name of the column to be matched. :param expectation_kwargs=None: A dictionary of kwargs to match against. :param remove_multiple_matches=False: Match multiple expectations :param dry_run=False: Return a list of matching expectations without removing

Returns None, unless dry_run=True. If dry_run=True and remove_multiple_matches=False then return the expectation that *would be* removed. If dry_run=True and remove_multiple_matches=True then return a list of expectations that *would be* removed.

Note: If remove_expectation doesn't find any matches, it raises a ValueError. If remove_expectation finds more than one matches and remove_multiple_matches!=True, it raises a ValueError. If dry_run=True, then *remove_expectation* acts as a thin layer to find_expectations, with the default values for discard_result_format_kwargs, discard_include_config_kwargs, and discard_catch_exceptions_kwargs

```
class great_expectations.core.ExpectationSuiteSchema(*, only:
    types.StrSequenceOrSet
    = None, exclude:
    types.StrSequenceOrSet =
    (), many: bool = False, context:
    typing.Dict = None, load_only:
    types.StrSequenceOrSet
    = (), dump_only:
    types.StrSequenceOrSet =
    (), partial: typing.Union[bool,
    types.StrSequenceOrSet] =
    False, unknown: str = None)
```

Bases: `marshmallow.Schema`

Base schema class with which to define custom schemas.

Example usage:

```
import datetime as dt
from dataclasses import dataclass

from marshmallow import Schema, fields

@dataclass
class Album:
    title: str
    release_date: dt.date

class AlbumSchema(Schema):
    title = fields.Str()
    release_date = fields.Date()

album = Album("Beggars Banquet", dt.date(1968, 12, 6))
schema = AlbumSchema()
data = schema.dump(album)
data # {'release_date': '1968-12-06', 'title': 'Beggars Banquet'}
```

Parameters

- **only** – Whitelist of the declared fields to select when instantiating the Schema. If None, all fields are used. Nested fields can be represented with dot delimiters.
- **exclude** – Blacklist of the declared fields to exclude when instantiating the Schema. If a field appears in both *only* and *exclude*, it is not used. Nested fields can be represented with dot delimiters.
- **many** – Should be set to *True* if *obj* is a collection so that the object will be serialized to a list.
- **context** – Optional context passed to *fields.Method* and *fields.Function* fields.
- **load_only** – Fields to skip during serialization (write-only fields)
- **dump_only** – Fields to skip during deserialization (read-only fields)
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to *Nested* fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.
- **unknown** – Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*.

Changed in version 3.0.0: *prefix* parameter removed.

Changed in version 2.0.0: *__validators__*, *__preprocessors__*, and *__data_handlers__* are removed in favor of *marshmallow.decorators.validates_schema*, *marshmallow.decorators.pre_load* and *marshmallow.decorators.post_dump*. *__accessor__* and *__error_handler__* are deprecated. Implement the *handle_error* and *get_attribute* methods instead.

expectation_suite_name

```
expectations
evaluation_parameters
data_asset_type
meta
clean_empty (self, data)
prepare_dump (self, data, **kwargs)
make_expectation_suite (self, data, **kwargs)
```

```
class great_expectations.core.ExpectationValidationResult (success=None, expectation_config=None, result=None, meta=None, exception_info=None)
```

Bases: object

```
__eq__ (self, other)
    ExpectationValidationResult equality ignores instance identity, relying only on properties.
```

```
__repr__ (self)
    Return repr(self).
```

```
__str__ (self)
    Return str(self).
```

```
validate_result_dict (self, result)
```

```
to_json_dict (self)
```

```
get_metric (self, metric_name, **kwargs)
```

```
class great_expectations.core.ExpectationValidationResultSchema (*, only:
    types.StrSequenceOrSet
    = None,
    exclude:
    types.StrSequenceOrSet
    = (), many:
    bool = False,
    context:
    typing.Dict
    = None,
    load_only:
    types.StrSequenceOrSet
    = (),
    dump_only:
    types.StrSequenceOrSet
    = (), partial:
    typing.Union[bool,
    types.StrSequenceOrSet]
    = False, unknown: str =
    None)
```

Bases: marshmallow.Schema

Base schema class with which to define custom schemas.

Example usage:

```

import datetime as dt
from dataclasses import dataclass

from marshmallow import Schema, fields

@dataclass
class Album:
    title: str
    release_date: dt.date

class AlbumSchema(Schema):
    title = fields.Str()
    release_date = fields.Date()

album = Album("Beggars Banquet", dt.date(1968, 12, 6))
schema = AlbumSchema()
data = schema.dump(album)
data # {'release_date': '1968-12-06', 'title': 'Beggars Banquet'}

```

Parameters

- **only** – Whitelist of the declared fields to select when instantiating the Schema. If None, all fields are used. Nested fields can be represented with dot delimiters.
- **exclude** – Blacklist of the declared fields to exclude when instantiating the Schema. If a field appears in both *only* and *exclude*, it is not used. Nested fields can be represented with dot delimiters.
- **many** – Should be set to *True* if *obj* is a collection so that the object will be serialized to a list.
- **context** – Optional context passed to *fields.Method* and *fields.Function* fields.
- **load_only** – Fields to skip during serialization (write-only fields)
- **dump_only** – Fields to skip during deserialization (read-only fields)
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to Nested fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.
- **unknown** – Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*.

Changed in version 3.0.0: *prefix* parameter removed.

Changed in version 2.0.0: *__validators__*, *__preprocessors__*, and *__data_handlers__* are removed in favor of *marshmallow.decorators.validates_schema*, *marshmallow.decorators.pre_load* and *marshmallow.decorators.post_dump*. *__accessor__* and *__error_handler__* are deprecated. Implement the *handle_error* and *get_attribute* methods instead.

success

expectation_config

result

```
meta
exception_info
convert_result_to_serializable (self, data, **kwargs)
make_expectation_validation_result (self, data, **kwargs)
class great_expectations.core.ExpectationSuiteValidationResult (success=None,
                                                                results=None,
                                                                evalua-
                                                                tion_parameters=None,
                                                                statistics=None,
                                                                meta=None)

Bases: great_expectations.types.DictDot
__eq__ (self, other)
    ExpectationSuiteValidationResult equality ignores instance identity, relying only on properties.
__repr__ (self)
    Return repr(self).
__str__ (self)
    Return str(self).
to_json_dict (self)
get_metric (self, metric_name, **kwargs)
```

```

class great_expectations.core.ExpectationSuiteValidationResultSchema(*, only:
    types.StrSequenceOrSet
    =
    None,
    exclude:
    types.StrSequenceOrSet
    = (),
    many:
    bool =
    False,
    context:
    typing.Dict
    =
    None,
    load_only:
    types.StrSequenceOrSet
    = (),
    dump_only:
    types.StrSequenceOrSet
    = (),
    partial:
    typing.Union[bool,
    types.StrSequenceOrSet]
    =
    False,
    unknown:
    str =
    None)

```

Bases: `marshmallow.Schema`

Base schema class with which to define custom schemas.

Example usage:

```

import datetime as dt
from dataclasses import dataclass

from marshmallow import Schema, fields

@dataclass
class Album:
    title: str
    release_date: dt.date

class AlbumSchema(Schema):
    title = fields.Str()
    release_date = fields.Date()

```

(continues on next page)

(continued from previous page)

```
album = Album("Beggars Banquet", dt.date(1968, 12, 6))
schema = AlbumSchema()
data = schema.dump(album)
data # {'release_date': '1968-12-06', 'title': 'Beggars Banquet'}
```

Parameters

- **only** – Whitelist of the declared fields to select when instantiating the Schema. If None, all fields are used. Nested fields can be represented with dot delimiters.
- **exclude** – Blacklist of the declared fields to exclude when instantiating the Schema. If a field appears in both *only* and *exclude*, it is not used. Nested fields can be represented with dot delimiters.
- **many** – Should be set to *True* if *obj* is a collection so that the object will be serialized to a list.
- **context** – Optional context passed to `fields.Method` and `fields.Function` fields.
- **load_only** – Fields to skip during serialization (write-only fields)
- **dump_only** – Fields to skip during deserialization (read-only fields)
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to *Nested* fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.
- **unknown** – Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*.

Changed in version 3.0.0: *prefix* parameter removed.

Changed in version 2.0.0: `__validators__`, `__preprocessors__`, and `__data_handlers__` are removed in favor of `marshmallow.decorators.validates_schema`, `marshmallow.decorators.pre_load` and `marshmallow.decorators.post_dump`. `__accessor__` and `__error_handler__` are deprecated. Implement the `handle_error` and `get_attribute` methods instead.

success

results

evaluation_parameters

statistics

meta

prepare_dump (*self*, *data*, ***kwargs*)

make_expectation_suite_validation_result (*self*, *data*, ***kwargs*)

`great_expectations.core.expectationConfigurationSchema`

`great_expectations.core.expectationSuiteSchema`

`great_expectations.core.expectationValidationResultSchema`

`great_expectations.core.expectationSuiteValidationResultSchema`

`great_expectations.core.runIdentifierSchema`

`great_expectations.data_asset`

Submodules

`great_expectations.data_asset.data_asset`

Module Contents

Classes

`DataAsset(*args, **kwargs)`

Functions

`_calc_validation_statistics(validation_results)` Calculate summary statistics for the validation results and

`great_expectations.data_asset.data_asset.logger`**class** `great_expectations.data_asset.data_asset.DataAsset(*args, **kwargs)`Bases: `object``_data_asset_type = DataAsset``list_available_expectation_types(self)``autoinspect(self, profiler)`

Deprecated: use profile instead.

Use the provided profiler to evaluate this data_asset and assign the resulting expectation suite as its own.

Parameters `profiler` – The profiler to use**Returns** `tuple(expectation_suite, validation_results)``profile(self, profiler, profiler_configuration=None)`

Use the provided profiler to evaluate this data_asset and assign the resulting expectation suite as its own.

Parameters

- **profiler** – The profiler to use
- **profiler_configuration** – Optional profiler configuration dict

Returns `tuple(expectation_suite, validation_results)``edit_expectation_suite(self)`**classmethod** `expectation(cls, method_arg_names)`

Manages configuration and running of expectation objects.

Expectation builds and saves a new expectation configuration to the DataAsset object. It is the core decorator used by great expectations to manage expectation configurations.

Parameters `method_arg_names` (`List`) – An ordered list of the arguments used by the method implementing the expectation (typically the result of inspection). Positional arguments are explicitly mapped to keyword arguments when the expectation is run.

Notes

Intermediate decorators that call the core `@expectation` decorator will most likely need to pass their decorated methods' signature up to the expectation decorator. For example, the `MetaPandasDataset column_map_expectation` decorator relies on the `DataAsset` expectation decorator, but will pass through the signature from the implementing method.

@expectation intercepts and takes action based on the following parameters:

- `include_config` (boolean or None) : If True, then include the generated expectation config as part of the result object. For more detail, see [include_config](#).
- `catch_exceptions` (boolean or None) : If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **result_format** (str or None) [Which output mode to use: `BOOLEAN_ONLY`, `BASIC`, `COMPLETE`, or `SUMMARY`.] For more detail, see [result_format](#).
- `meta` (dict or None): A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

`_initialize_expectations` (*self*, *expectation_suite=None*, *expectation_suite_name=None*)

Instantiates `_expectation_suite` as empty by default or with a specified expectation *config*. In addition, this always sets the *default_expectation_args* to:

include_config: False, *catch_exceptions*: False, *output_format*: 'BASIC'

By default, initializes `data_asset_type` to the name of the implementing class, but subclasses that have interoperable semantics (e.g. `Dataset`) may override that parameter to clarify their interoperability.

Parameters

- **`expectation_suite`** (*json*) – A json-serializable expectation config. If None, creates default `_expectation_suite` with an empty list of expectations and key value `data_asset_name` as `data_asset_name`.
- **`expectation_suite_name`** (*string*) – The name to assign to the `expectation_suite.expectation_suite_name`

Returns None

`append_expectation` (*self*, *expectation_config*)

This method is a thin wrapper for `ExpectationSuite.append_expectation`

`_append_expectation` (*self*, *expectation_config*)

This method

This method should become a thin wrapper for `ExpectationSuite.append_expectation`

Included for backwards compatibility.

`_copy_and_clean_up_expectation` (*self*, *expectation*, *discard_result_format_kwargs=True*,
discard_include_config_kwargs=True, *discard_catch_exceptions_kwargs=True*)

This method is a thin wrapper for `ExpectationSuite._copy_and_clean_up_expectation`

`_copy_and_clean_up_expectations_from_indexes` (*self*, *match_indexes*, *discard_result_format_kwargs=True*,
discard_include_config_kwargs=True, *discard_catch_exceptions_kwargs=True*)

This method is a thin wrapper for `ExpectationSuite._copy_and_clean_up_expectations_from_indexes`

find_expectation_indexes (*self*, *expectation_type=None*, *column=None*, *expectation_kwargs=None*)

This method is a thin wrapper for `ExpectationSuite.find_expectation_indexes`

find_expectations (*self*, *expectation_type=None*, *column=None*, *expectation_kwargs=None*, *discard_result_format_kwargs=True*, *discard_include_config_kwargs=True*, *discard_catch_exceptions_kwargs=True*)

This method is a thin wrapper for `ExpectationSuite.find_expectations()`

remove_expectation (*self*, *expectation_type=None*, *column=None*, *expectation_kwargs=None*, *remove_multiple_matches=False*, *dry_run=False*)

This method is a thin wrapper for `ExpectationSuite.remove()`

set_config_value (*self*, *key*, *value*)

get_config_value (*self*, *key*)

property batch_kwargs (*self*)

property batch_id (*self*)

property batch_markers (*self*)

property batch_parameters (*self*)

discard_failing_expectations (*self*)

get_default_expectation_arguments (*self*)

Fetch default expectation arguments for this `data_asset`

Returns

A dictionary containing all the current default expectation arguments for a `data_asset`

Ex:

```
{
    "include_config" : True,
    "catch_exceptions" : False,
    "result_format" : 'BASIC'
}
```

See also:

`set_default_expectation_arguments`

set_default_expectation_argument (*self*, *argument*, *value*)

Set a default expectation argument for this `data_asset`

Parameters

- **argument** (*string*) – The argument to be replaced
- **value** – The New argument to use for replacement

Returns None

See also:

`get_default_expectation_arguments`

get_expectations_config (*self*, *discard_failed_expectations=True*, *discard_result_format_kwargs=True*, *discard_include_config_kwargs=True*, *discard_catch_exceptions_kwargs=True*, *suppress_warnings=False*)

```
get_expectation_suite(self, discard_failed_expectations=True, discard_result_format_kwargs=True, discard_include_config_kwargs=True, discard_card_catch_exceptions_kwargs=True, suppress_warnings=False, suppress_logging=False)
```

Returns `_expectation_config` as a JSON object, and perform some cleaning along the way.

Parameters

- **discard_failed_expectations** (*boolean*) – Only include expectations with `success_on_last_run=True` in the exported config. Defaults to *True*.
- **discard_result_format_kwargs** (*boolean*) – In returned expectation objects, suppress the `result_format` parameter. Defaults to *True*.
- **discard_include_config_kwargs** (*boolean*) – In returned expectation objects, suppress the `include_config` parameter. Defaults to *True*.
- **discard_catch_exceptions_kwargs** (*boolean*) – In returned expectation objects, suppress the `catch_exceptions` parameter. Defaults to *True*.
- **suppress_warnings** (*boolean*) – If true, do not include warnings in logging information about the operation.
- **suppress_logging** (*boolean*) – If true, do not create a log entry (useful when using `get_expectation_suite` programmatically)

Returns An expectation suite.

Note: `get_expectation_suite` does not affect the underlying expectation suite at all. The returned suite is a copy of `_expectation_suite`, not the original object.

```
save_expectation_suite(self, filepath=None, discard_failed_expectations=True, discard_result_format_kwargs=True, discard_include_config_kwargs=True, discard_card_catch_exceptions_kwargs=True, suppress_warnings=False)
```

Writes `_expectation_config` to a JSON file.

Writes the DataAsset's expectation config to the specified JSON `filepath`. Failing expectations can be excluded from the JSON expectations config with `discard_failed_expectations`. The kwarg key-value pairs `result_format`, `include_config`, and `catch_exceptions` are optionally excluded from the JSON expectations config.

Parameters

- **filepath** (*string*) – The location and name to write the JSON config file to.
- **discard_failed_expectations** (*boolean*) – If True, excludes expectations that do not return `success = True`. If False, all expectations are written to the JSON config file.
- **discard_result_format_kwargs** (*boolean*) – If True, the `result_format` attribute for each expectation is not written to the JSON config file.
- **discard_include_config_kwargs** (*boolean*) – If True, the `include_config` attribute for each expectation is not written to the JSON config file.
- **discard_catch_exceptions_kwargs** (*boolean*) – If True, the `catch_exceptions` attribute for each expectation is not written to the JSON config file.

- **suppress_warnings** (*boolean*) – If True, all warnings raised by Great Expectations, as a result of dropped expectations, are suppressed.

validate (*self*, *expectation_suite=None*, *run_id=None*, *data_context=None*, *evaluation_parameters=None*, *catch_exceptions=True*, *result_format=None*, *only_return_failures=False*, *run_name=None*, *run_time=None*)

Generates a JSON-formatted report describing the outcome of all expectations.

Use the default *expectation_suite=None* to validate the expectations config associated with the DataAsset.

Parameters

- **expectation_suite** (*json or None*) – If None, uses the expectations config generated with the DataAsset during the current session. If a JSON file, validates those expectations.
- **run_name** (*str*) – Used to identify this validation result as part of a collection of validations. See DataContext for more information.
- **data_context** (*DataContext*) – A datacontext object to use as part of validation for binding evaluation parameters and registering validation results.
- **evaluation_parameters** (*dict or None*) – If None, uses the evaluation parameters from the expectation_suite provided or as part of the data_asset. If a dict, uses the evaluation parameters in the dictionary.
- **catch_exceptions** (*boolean*) – If True, exceptions raised by tests will not end validation and will be described in the returned report.
- **result_format** (*string or None*) – If None, uses the default value ('BASIC' or as specified). If string, the returned expectation output follows the specified format ('BOOLEAN_ONLY', 'BASIC', etc.).
- **only_return_failures** (*boolean*) – If True, expectation results are only returned when `success = False`

Returns

A JSON-formatted dictionary containing a list of the validation results. An example of the returned format:

```
{
  "results": [
    {
      "unexpected_list": [unexpected_value_1, unexpected_value_2],
      "expectation_type": "expect_*",
      "kwargs": {
        "column": "Column_Name",
        "output_format": "SUMMARY"
      },
      "success": true,
      "raised_exception": false,
      "exception_traceback": null
    },
    {
      ... (Second expectation results)
    },
    ... (More expectations results)
  ],
  "success": true,
  "statistics": {
```

(continues on next page)

(continued from previous page)

```

    "evaluated_expectations": n,
    "successful_expectations": m,
    "unsuccessful_expectations": n - m,
    "success_percent": m / n
  }
}

```

Notes

If the configuration object was built with a different version of great expectations then the current environment. If no version was found in the configuration file.

Raises `AttributeError` – if `'catch_exceptions'`=None and an expectation throws an `AttributeError` –

`get_evaluation_parameter` (*self*, *parameter_name*, *default_value=None*)

Get an evaluation parameter value that has been stored in meta.

Parameters

- **`parameter_name`** (*string*) – The name of the parameter to store.
- **`default_value`** (*any*) – The default value to be returned if the parameter is not found.

Returns The current value of the evaluation parameter.

`set_evaluation_parameter` (*self*, *parameter_name*, *parameter_value*)

Provide a value to be stored in the `data_asset` `evaluation_parameters` object and used to evaluate parameterized expectations.

Parameters

- **`parameter_name`** (*string*) – The name of the kwarg to be replaced at evaluation time
- **`parameter_value`** (*any*) – The value to be used

`add_citation` (*self*, *comment*, *batch_kwargs=None*, *batch_markers=None*, *batch_parameters=None*, *citation_date=None*)

`property expectation_suite_name` (*self*)

Gets the current `expectation_suite` name of this `data_asset` as stored in the expectations configuration.

`_format_map_output` (*self*, *result_format*, *success*, *element_count*, *nonnull_count*, *unexpected_count*, *unexpected_list*, *unexpected_index_list*)

Helper function to construct expectation result objects for `map_expectations` (such as `column_map_expectation` and `file_lines_map_expectation`).

Expectations support four `result_formats`: `BOOLEAN_ONLY`, `BASIC`, `SUMMARY`, and `COMPLETE`. In each case, the object returned has a different set of populated fields. See [result_format](#) for more information.

This function handles the logic for mapping those fields for `column_map_expectations`.

`_calc_map_expectation_success` (*self*, *success_count*, *nonnull_count*, *mostly*)

Calculate success and percent_success for `column_map_expectations`

Parameters

- **`success_count`** (*int*) – The number of successful values in the column
- **`nonnull_count`** (*int*) – The number of nonnull values in the column

- **mostly** (*float or None*) – A value between 0 and 1 (or None), indicating the fraction of successes required to pass the expectation as a whole. If mostly=None, then all values must succeed in order for the expectation as a whole to succeed.

Returns success (boolean), percent_success (float)

test_expectation_function (*self, function, *args, **kwargs*)

Test a generic expectation function

Parameters

- **function** (*func*) – The function to be tested. (Must be a valid expectation function.)
- ***args** – Positional arguments to be passed the the function
- ****kwargs** – Keyword arguments to be passed the the function

Returns A JSON-serializable expectation result object.

Notes

This function is a thin layer to allow quick testing of new expectation functions, without having to define custom classes, etc. To use developed expectations from the command-line tool, you will still need to define custom classes, etc.

Check out `custom_expectations_reference` for more information.

`great_expectations.data_asset.data_asset.ValidationStatistics`

`great_expectations.data_asset.data_asset._calc_validation_statistics` (*validation_results*)

Calculate summary statistics for the validation results and return `ExpectationStatistics`.

`great_expectations.data_asset.file_data_asset`

Module Contents

Classes

<code>MetaFileDataAsset(*args, **kwargs)</code>	MetaFileDataset is a thin layer above FileDataset.
<code>FileDataAsset(file_path=None, *args, **kwargs)</code>	FileDataset instantiates the great_expectations Expectations API as a

class `great_expectations.data_asset.file_data_asset.MetaFileDataAsset` (**args, **kwargs*)

Bases: `great_expectations.data_asset.data_asset.DataAsset`

MetaFileDataset is a thin layer above FileDataset. This two-layer inheritance is required to make `@classmethod` decorators work. Practically speaking, that means that MetaFileDataset implements expectation decorators, like `file_lines_map_expectation` and FileDataset implements the expectation methods themselves.

classmethod `file_lines_map_expectation` (*cls, func*)

Constructs an expectation using file lines map semantics. The `file_lines_map_expectations` decorator handles boilerplate issues surrounding the common pattern of evaluating truthiness of some condition on an line by line basis in a file.

Parameters **func** (*function*) – The function implementing an expectation that will be applied line by line across a file. The function should take a file and return information about

how many lines met expectations.

Notes

Users can specify skip value *k* that will cause the expectation function to disregard the first *k* lines of the file.

`file_lines_map_expectation` will add a `kwargs_lines` to the called function with the nonnull lines to process.

`null_lines_regex` defines a regex used to skip lines, but can be overridden

See also:

`expect_file_line_regex_match_count_to_be_between` for an example of a `file_lines_map_expectation`

```
class great_expectations.data_asset.file_data_asset.FileDataAsset (file_path=None,
                                                                    *args,
                                                                    **kwargs)
```

Bases: `great_expectations.data_asset.file_data_asset.MetaFileDataAsset`

FileDataset instantiates the great_expectations Expectations API as a subclass of a python file object. For the full API reference, please see `DataAsset`

`_data_asset_type = FileDataAsset`

```
expect_file_line_regex_match_count_to_be_between (self, regex, ex-
                                                    pected_min_count=0, ex-
                                                    pected_max_count=None,
                                                    skip=None, mostly=None,
                                                    null_lines_regex='\\s*$',
                                                    result_format=None,
                                                    include_config=True,
                                                    catch_exceptions=None,
                                                    meta=None, _lines=None)
```

Expect the number of times a regular expression appears on each line of a file to be between a maximum and minimum value.

Parameters

- **regex** – A string that can be compiled as valid regular expression to match
- **expected_min_count** (*None or nonnegative integer*) – Specifies the minimum number of times regex is expected to appear on each line of the file
- **expected_max_count** (*None or nonnegative integer*) – Specifies the maximum number of times regex is expected to appear on each line of the file

Keyword Arguments

- **skip** (*None or nonnegative integer*) – Integer specifying the first lines in the file the method should skip before assessing expectations
- **mostly** (*None or number between 0 and 1*) – Specifies an acceptable error for expectations. If the percentage of unexpected lines is less than mostly, the method still returns true even if all lines don't match the expectation criteria.
- **null_lines_regex** (*valid regular expression or None*) – If not none, a regex to skip lines as null. Defaults to empty or whitespace-only lines.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).
- **_lines** (*list*) – The lines over which to operate (provided by the `file_lines_map_expectation` decorator)

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

```
expect_file_line_regex_match_count_to_equal(self, regex, expected_count=0,
                                             skip=None, mostly=None,
                                             nonnull_lines_regex="\s*$",
                                             result_format=None, include_config=True,
                                             catch_exceptions=None, meta=None,
                                             _lines=None)
```

Expect the number of times a regular expression appears on each line of a file to be between a maximum and minimum value.

Parameters

- **regex** – A string that can be compiled as valid regular expression to match
- **expected_count** (*None or nonnegative integer*) – Specifies the number of times regex is expected to appear on each line of the file

Keyword Arguments

- **skip** (*None or nonnegative integer*) – Integer specifying the first lines in the file the method should skip before assessing expectations
- **mostly** (*None or number between 0 and 1*) – Specifies an acceptable error for expectations. If the percentage of unexpected lines is less than mostly, the method still returns true even if all lines don't match the expectation criteria.
- **nonnull_lines_regex** (*valid regular expression or None*) – If not none, a regex to skip lines as null. Defaults to empty or whitespace-only lines.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

- **_lines** (*list*) – The lines over which to operate (provided by the `file_lines_map_expectation` decorator)

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to `result_format` and `include_config`, `catch_exceptions`, and `meta`.

expect_file_hash_to_equal (*self*, *value*, *hash_alg='md5'*, *result_format=None*, *include_config=True*, *catch_exceptions=None*, *meta=None*)

Expect computed file hash to equal some given value.

Parameters *value* – A string to compare with the computed hash value

Keyword Arguments

- **hash_alg** (*string*) – Indicates the hash algorithm to use
- **result_format** (*str or None*) – Which output mode to use: `BOOLEAN_ONLY`, `BASIC`, `COMPLETE`, or `SUMMARY`. For more detail, see `result_format`.
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see `include_config`.
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see `catch_exceptions`.
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see `meta`.

Returns A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to `result_format` and `include_config`, `catch_exceptions`, and `meta`.

expect_file_size_to_be_between (*self*, *minsize=0*, *maxsize=None*, *result_format=None*, *include_config=True*, *catch_exceptions=None*, *meta=None*)

Expect file size to be between a user specified maxsize and minsize.

Parameters

- **minsize** (*integer*) – minimum expected file size
- **maxsize** (*integer*) – maximum expected file size

Keyword Arguments

- **result_format** (*str or None*) – Which output mode to use: `BOOLEAN_ONLY`, `BASIC`, `COMPLETE`, or `SUMMARY`. For more detail, see `result_format`.
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see `include_config`.
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see `catch_exceptions`.
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see `meta`.

Returns A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to `result_format` and `include_config`, `catch_exceptions`, and `meta`.

expect_file_to_exist (*self*, *filepath=None*, *result_format=None*, *include_config=True*, *catch_exceptions=None*, *meta=None*)

Checks to see if a file specified by the user actually exists

Parameters **filepath** (*str* or *None*) – The filepath to evaluate. If none, will check the currently-configured path object of this FileDataAsset.

Keyword Arguments

- **result_format** (*str* or *None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean* or *None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict* or *None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

expect_file_to_have_valid_table_header (*self*, *regex*, *skip=None*, *result_format=None*, *include_config=True*, *catch_exceptions=None*, *meta=None*)

Checks to see if a file has a line with unique delimited values, such a line may be used as a table header.

Keyword Arguments

- **skip** (*nonnegative integer*) – Integer specifying the first lines in the file the method should skip before assessing expectations
- **regex** (*string*) – A string that can be compiled as valid regular expression. Used to specify the elements of the table header (the column headers)
- **result_format** (*str* or *None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean* or *None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict* or *None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

expect_file_to_be_valid_json (*self*, *schema=None*, *result_format=None*, *include_config=True*, *catch_exceptions=None*, *meta=None*)

Parameters

- **schema** – string optional JSON schema file on which JSON data file is validated against
- **result_format** (*str* or *None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).

- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification.

For more detail, see [meta](#).

Returns A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

`great_expectations.data_asset.util`

Module Contents

Classes

[`DocInherit`](#)(*mthd*)

Functions

<code>parse_result_format</code> (<i>result_format</i>)	This is a simple helper utility that can be used to parse a string <i>result_format</i> into the dict format used
<code>recursively_convert_to_json_serializable</code> (<i>test_obj</i>)	Helper function to convert a dict object to one that is serializable

`great_expectations.data_asset.util.parse_result_format` (*result_format*)

This is a simple helper utility that can be used to parse a string *result_format* into the dict format used internally by `great_expectations`. It is not necessary but allows shorthand for *result_format* in cases where there is no need to specify a custom *partial_unexpected_count*.

class `great_expectations.data_asset.util.DocInherit` (*mthd*)

Bases: `object`

`__get__` (*self, obj, cls*)

`great_expectations.data_asset.util.recursively_convert_to_json_serializable` (*test_obj*)

Helper function to convert a dict object to one that is serializable

Parameters *test_obj* – an object to attempt to convert a corresponding json-serializable object

Returns (dict) A converted *test_obj*

Warning: *test_obj* may also be converted in place.

Package Contents

Classes

DataAsset(*args, **kwargs)

FileDataAsset(file_path=None, *args, **kwargs) FileDataset instantiates the great_expectations Expectations API as a

class great_expectations.data_asset.**DataAsset** (*args, **kwargs)

Bases: object

_data_asset_type = DataAsset

list_available_expectation_types (self)

autoinspect (self, profiler)

Deprecated: use profile instead.

Use the provided profiler to evaluate this data_asset and assign the resulting expectation suite as its own.

Parameters **profiler** – The profiler to use

Returns tuple(expectation_suite, validation_results)

profile (self, profiler, profiler_configuration=None)

Use the provided profiler to evaluate this data_asset and assign the resulting expectation suite as its own.

Parameters

- **profiler** – The profiler to use
- **profiler_configuration** – Optional profiler configuration dict

Returns tuple(expectation_suite, validation_results)

edit_expectation_suite (self)

classmethod expectation (cls, method_arg_names)

Manages configuration and running of expectation objects.

Expectation builds and saves a new expectation configuration to the DataAsset object. It is the core decorator used by great expectations to manage expectation configurations.

Parameters **method_arg_names** (*List*) – An ordered list of the arguments used by the method implementing the expectation (typically the result of inspection). Positional arguments are explicitly mapped to keyword arguments when the expectation is run.

Notes

Intermediate decorators that call the core @expectation decorator will most likely need to pass their decorated methods' signature up to the expectation decorator. For example, the MetaPandasDataset column_map_expectation decorator relies on the DataAsset expectation decorator, but will pass through the signature from the implementing method.

@expectation intercepts and takes action based on the following parameters:

- **include_config** (boolean or None) : If True, then include the generated expectation config as part of the result object. For more detail, see [include_config](#).

- `catch_exceptions` (boolean or None) : If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **`result_format` (str or None)** [Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*.] For more detail, see [result_format](#).
- `meta` (dict or None): A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

`_initialize_expectations` (*self*, *expectation_suite=None*, *expectation_suite_name=None*)

Instantiates `_expectation_suite` as empty by default or with a specified expectation *config*. In addition, this always sets the *default_expectation_args* to:

include_config: False, *catch_exceptions*: False, *output_format*: 'BASIC'

By default, initializes *data_asset_type* to the name of the implementing class, but subclasses that have interoperable semantics (e.g. Dataset) may override that parameter to clarify their interoperability.

Parameters

- **`expectation_suite`** (*json*) – A json-serializable expectation config. If None, creates default `_expectation_suite` with an empty list of expectations and key value *data_asset_name* as *data_asset_name*.
- **`expectation_suite_name`** (*string*) – The name to assign to the *expectation_suite.expectation_suite_name*

Returns None

`append_expectation` (*self*, *expectation_config*)

This method is a thin wrapper for `ExpectationSuite.append_expectation`

`_append_expectation` (*self*, *expectation_config*)

This method

This method should become a thin wrapper for `ExpectationSuite.append_expectation`

Included for backwards compatibility.

`_copy_and_clean_up_expectation` (*self*, *expectation*, *discard_result_format_kwargs=True*,
discard_include_config_kwargs=True, *discard_catch_exceptions_kwargs=True*)

This method is a thin wrapper for `ExpectationSuite._copy_and_clean_up_expectation`

`_copy_and_clean_up_expectations_from_indexes` (*self*, *match_indexes*, *discard_result_format_kwargs=True*,
discard_include_config_kwargs=True, *discard_catch_exceptions_kwargs=True*)

This method is a thin wrapper for `ExpectationSuite._copy_and_clean_up_expectations_from_indexes`

`find_expectation_indexes` (*self*, *expectation_type=None*, *column=None*, *expectation_kwargs=None*)

This method is a thin wrapper for `ExpectationSuite.find_expectation_indexes`

`find_expectations` (*self*, *expectation_type=None*, *column=None*, *expectation_kwargs=None*, *discard_result_format_kwargs=True*, *discard_include_config_kwargs=True*, *discard_catch_exceptions_kwargs=True*)

This method is a thin wrapper for `ExpectationSuite.find_expectations()`

`remove_expectation` (*self*, *expectation_type=None*, *column=None*, *expectation_kwargs=None*, *remove_multiple_matches=False*, *dry_run=False*)

This method is a thin wrapper for `ExpectationSuite.remove()`

`set_config_value` (*self*, *key*, *value*)

get_config_value (*self*, *key*)
property batch_kwargs (*self*)
property batch_id (*self*)
property batch_markers (*self*)
property batch_parameters (*self*)
discard_failing_expectations (*self*)
get_default_expectation_arguments (*self*)
 Fetch default expectation arguments for this data_asset

Returns

A dictionary containing all the current default expectation arguments for a data_asset

Ex:

```
{
    "include_config" : True,
    "catch_exceptions" : False,
    "result_format" : 'BASIC'
}
```

See also:

set_default_expectation_arguments

set_default_expectation_argument (*self*, *argument*, *value*)
 Set a default expectation argument for this data_asset

Parameters

- **argument** (*string*) – The argument to be replaced
- **value** – The New argument to use for replacement

Returns

None

See also:

get_default_expectation_arguments

get_expectations_config (*self*, *discard_failed_expectations=True*, *dis-*
card_result_format_kwargs=True, *dis-*
card_include_config_kwargs=True, *dis-*
card_catch_exceptions_kwargs=True, *suppress_warnings=False*)
get_expectation_suite (*self*, *discard_failed_expectations=True*, *dis-*
card_result_format_kwargs=True, *dis-*
card_include_config_kwargs=True, *dis-*
card_catch_exceptions_kwargs=True, *suppress_warnings=False*,
suppress_logging=False)

Returns _expectation_config as a JSON object, and perform some cleaning along the way.

Parameters

- **discard_failed_expectations** (*boolean*) – Only include expectations with success_on_last_run=True in the exported config. Defaults to *True*.
- **discard_result_format_kwargs** (*boolean*) – In returned expectation objects, suppress the *result_format* parameter. Defaults to *True*.

- **discard_include_config_kwargs** (*boolean*) – In returned expectation objects, suppress the *include_config* parameter. Defaults to *True*.
- **discard_catch_exceptions_kwargs** (*boolean*) – In returned expectation objects, suppress the *catch_exceptions* parameter. Defaults to *True*.
- **suppress_warnings** (*boolean*) – If true, do not include warnings in logging information about the operation.
- **suppress_logging** (*boolean*) – If true, do not create a log entry (useful when using *get_expectation_suite* programmatically)

Returns An expectation suite.

Note: *get_expectation_suite* does not affect the underlying expectation suite at all. The returned suite is a copy of *_expectation_suite*, not the original object.

```
save_expectation_suite (self,          filepath=None,          discard_failed_expectations=True,  
                        discard_result_format_kwargs=True,          dis-  
                        card_include_config_kwargs=True,          dis-  
                        card_catch_exceptions_kwargs=True, suppress_warnings=False)
```

Writes *_expectation_config* to a JSON file.

Writes the DataAsset's expectation config to the specified JSON filepath. Failing expectations can be excluded from the JSON expectations config with *discard_failed_expectations*. The kwarg key-value pairs *result_format*, *include_config*, and *catch_exceptions* are optionally excluded from the JSON expectations config.

Parameters

- **filepath** (*string*) – The location and name to write the JSON config file to.
- **discard_failed_expectations** (*boolean*) – If True, excludes expectations that do not return *success = True*. If False, all expectations are written to the JSON config file.
- **discard_result_format_kwargs** (*boolean*) – If True, the *result_format* attribute for each expectation is not written to the JSON config file.
- **discard_include_config_kwargs** (*boolean*) – If True, the *include_config* attribute for each expectation is not written to the JSON config file.
- **discard_catch_exceptions_kwargs** (*boolean*) – If True, the *catch_exceptions* attribute for each expectation is not written to the JSON config file.
- **suppress_warnings** (*boolean*) – If True, all warnings raised by Great Expectations, as a result of dropped expectations, are suppressed.

```
validate (self,          expectation_suite=None,          run_id=None,          data_context=None,          eval-  
            uation_parameters=None,          catch_exceptions=True,          result_format=None,  
            only_return_failures=False, run_name=None, run_time=None)
```

Generates a JSON-formatted report describing the outcome of all expectations.

Use the default *expectation_suite=None* to validate the expectations config associated with the DataAsset.

Parameters

- **expectation_suite** (*json or None*) – If None, uses the expectations config generated with the DataAsset during the current session. If a JSON file, validates those expectations.

- **run_name** (*str*) – Used to identify this validation result as part of a collection of validations. See `DataContext` for more information.
- **data_context** (`DataContext`) – A `datacontext` object to use as part of validation for binding evaluation parameters and registering validation results.
- **evaluation_parameters** (*dict or None*) – If `None`, uses the `evaluation_parameters` from the `expectation_suite` provided or as part of the `data_asset`. If a `dict`, uses the evaluation parameters in the dictionary.
- **catch_exceptions** (*boolean*) – If `True`, exceptions raised by tests will not end validation and will be described in the returned report.
- **result_format** (*string or None*) – If `None`, uses the default value (`'BASIC'` or as specified). If `string`, the returned expectation output follows the specified format (`'BOOLEAN_ONLY'`, `'BASIC'`, etc.).
- **only_return_failures** (*boolean*) – If `True`, expectation results are only returned when `success = False`

Returns

A JSON-formatted dictionary containing a list of the validation results. An example of the returned format:

```
{
  "results": [
    {
      "unexpected_list": [unexpected_value_1, unexpected_value_2],
      "expectation_type": "expect_*",
      "kwargs": {
        "column": "Column_Name",
        "output_format": "SUMMARY"
      },
      "success": true,
      "raised_exception": false,
      "exception_traceback": null
    },
    {
      ... (Second expectation results)
    },
    ... (More expectations results)
  ],
  "success": true,
  "statistics": {
    "evaluated_expectations": n,
    "successful_expectations": m,
    "unsuccessful_expectations": n - m,
    "success_percent": m / n
  }
}
```

Notes

If the configuration object was built with a different version of great expectations then the current environment. If no version was found in the configuration file.

Raises `AttributeError` – if `'catch_exceptions'=None` and an expectation throws an `AttributeError` –

`get_evaluation_parameter` (*self*, *parameter_name*, *default_value=None*)

Get an evaluation parameter value that has been stored in meta.

Parameters

- **`parameter_name`** (*string*) – The name of the parameter to store.
- **`default_value`** (*any*) – The default value to be returned if the parameter is not found.

Returns The current value of the evaluation parameter.

`set_evaluation_parameter` (*self*, *parameter_name*, *parameter_value*)

Provide a value to be stored in the `data_asset` `evaluation_parameters` object and used to evaluate parameterized expectations.

Parameters

- **`parameter_name`** (*string*) – The name of the kwarg to be replaced at evaluation time
- **`parameter_value`** (*any*) – The value to be used

`add_citation` (*self*, *comment*, *batch_kwarg=None*, *batch_markers=None*, *batch_parameters=None*, *citation_date=None*)

`property expectation_suite_name` (*self*)

Gets the current `expectation_suite` name of this `data_asset` as stored in the expectations configuration.

`_format_map_output` (*self*, *result_format*, *success*, *element_count*, *nonnull_count*, *unexpected_count*, *unexpected_list*, *unexpected_index_list*)

Helper function to construct expectation result objects for `map_expectations` (such as `column_map_expectation` and `file_lines_map_expectation`).

Expectations support four `result_formats`: `BOOLEAN_ONLY`, `BASIC`, `SUMMARY`, and `COMPLETE`. In each case, the object returned has a different set of populated fields. See [result_format](#) for more information.

This function handles the logic for mapping those fields for `column_map_expectations`.

`_calc_map_expectation_success` (*self*, *success_count*, *nonnull_count*, *mostly*)

Calculate `success` and `percent_success` for `column_map_expectations`

Parameters

- **`success_count`** (*int*) – The number of successful values in the column
- **`nonnull_count`** (*int*) – The number of nonnull values in the column
- **`mostly`** (*float or None*) – A value between 0 and 1 (or None), indicating the fraction of successes required to pass the expectation as a whole. If `mostly=None`, then all values must succeed in order for the expectation as a whole to succeed.

Returns `success` (boolean), `percent_success` (float)

`test_expectation_function` (*self*, *function*, **args*, ***kwargs*)

Test a generic expectation function

Parameters

- **function** (*func*) – The function to be tested. (Must be a valid expectation function.)
- ***args** – Positional arguments to be passed the the function
- ****kwargs** – Keyword arguments to be passed the the function

Returns A JSON-serializable expectation result object.

Notes

This function is a thin layer to allow quick testing of new expectation functions, without having to define custom classes, etc. To use developed expectations from the command-line tool, you will still need to define custom classes, etc.

Check out `custom_expectations_reference` for more information.

class `great_expectations.data_asset.FileDataAsset` (*file_path=None, *args, **kwargs*)

Bases: `great_expectations.data_asset.file_data_asset.MetaFileDataAsset`

FileDataset instantiates the great_expectations Expectations API as a subclass of a python file object. For the full API reference, please see `DataAsset`

_data_asset_type = FileDataAsset

expect_file_line_regex_match_count_to_be_between (*self, regex, expected_min_count=0, expected_max_count=None, skip=None, mostly=None, null_lines_regex='\\s*\$', result_format=None, include_config=True, catch_exceptions=None, meta=None, _lines=None*)

Expect the number of times a regular expression appears on each line of a file to be between a maximum and minimum value.

Parameters

- **regex** – A string that can be compiled as valid regular expression to match
- **expected_min_count** (*None or nonnegative integer*) – Specifies the minimum number of times regex is expected to appear on each line of the file
- **expected_max_count** (*None or nonnegative integer*) – Specifies the maximum number of times regex is expected to appear on each line of the file

Keyword Arguments

- **skip** (*None or nonnegative integer*) – Integer specifying the first lines in the file the method should skip before assessing expectations
- **mostly** (*None or number between 0 and 1*) – Specifies an acceptable error for expectations. If the percentage of unexpected lines is less than mostly, the method still returns true even if all lines don't match the expectation criteria.
- **null_lines_regex** (*valid regular expression or None*) – If not none, a regex to skip lines as null. Defaults to empty or whitespace-only lines.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: `BOOLEAN_ONLY`, `BASIC`, `COMPLETE`, or `SUMMARY`. For more detail, see [result_format](#).

- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).
- **_lines** (*list*) – The lines over which to operate (provided by the `file_lines_map_expectation` decorator)

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

```
expect_file_line_regex_match_count_to_equal (self, regex, expected_count=0,
                                              skip=None, mostly=None,
                                              nonnull_lines_regex="\s*$",
                                              result_format=None, include_config=True,
                                              catch_exceptions=None, meta=None,
                                              _lines=None)
```

Expect the number of times a regular expression appears on each line of a file to be between a maximum and minimum value.

Parameters

- **regex** – A string that can be compiled as valid regular expression to match
- **expected_count** (*None or nonnegative integer*) – Specifies the number of times regex is expected to appear on each line of the file

Keyword Arguments

- **skip** (*None or nonnegative integer*) – Integer specifying the first lines in the file the method should skip before assessing expectations
- **mostly** (*None or number between 0 and 1*) – Specifies an acceptable error for expectations. If the percentage of unexpected lines is less than mostly, the method still returns true even if all lines don't match the expectation criteria.
- **nonnull_lines_regex** (*valid regular expression or None*) – If not none, a regex to skip lines as null. Defaults to empty or whitespace-only lines.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: `BOOLEAN_ONLY`, `BASIC`, `COMPLETE`, or `SUMMARY`. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).
- **_lines** (*list*) – The lines over which to operate (provided by the `file_lines_map_expectation` decorator)

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to *result_format* and *include_config*, *catch_exceptions*, and *meta*.

expect_file_hash_to_equal (*self*, *value*, *hash_alg*='md5', *result_format*=None, *include_config*=True, *catch_exceptions*=None, *meta*=None)
Expect computed file hash to equal some given value.

Parameters *value* – A string to compare with the computed hash value

Keyword Arguments

- **hash_alg** (*string*) – Indicates the hash algorithm to use
- **result_format** (*str* or *None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see *result_format*.
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see *include_config*.
- **catch_exceptions** (*boolean* or *None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see *catch_exceptions*.
- **meta** (*dict* or *None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see *meta*.

Returns A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to *result_format* and *include_config*, *catch_exceptions*, and *meta*.

expect_file_size_to_be_between (*self*, *minsize*=0, *maxsize*=None, *result_format*=None, *include_config*=True, *catch_exceptions*=None, *meta*=None)
Expect file size to be between a user specified maxsize and minsize.

Parameters

- **minsize** (*integer*) – minimum expected file size
- **maxsize** (*integer*) – maximum expected file size

Keyword Arguments

- **result_format** (*str* or *None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see *result_format*.
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see *include_config*.
- **catch_exceptions** (*boolean* or *None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see *catch_exceptions*.
- **meta** (*dict* or *None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see *meta*.

Returns A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to *result_format* and *include_config*, *catch_exceptions*, and *meta*.

expect_file_to_exist (*self*, *filepath*=None, *result_format*=None, *include_config*=True, *catch_exceptions*=None, *meta*=None)
Checks to see if a file specified by the user actually exists

Parameters `filepath` (*str or None*) – The filepath to evaluate. If none, will check the currently-configured path object of this FileDataAsset.

Keyword Arguments

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

expect_file_to_have_valid_table_header (*self, regex, skip=None, result_format=None, include_config=True, catch_exceptions=None, meta=None*)

Checks to see if a file has a line with unique delimited values, such a line may be used as a table header.

Keyword Arguments

- **skip** (*nonnegative integer*) – Integer specifying the first lines in the file the method should skip before assessing expectations
- **regex** (*string*) – A string that can be compiled as valid regular expression. Used to specify the elements of the table header (the column headers)
- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

expect_file_to_be_valid_json (*self, schema=None, result_format=None, include_config=True, catch_exceptions=None, meta=None*)

Parameters

- **schema** – string optional JSON schema file on which JSON data file is validated against
- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).

- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification.

For more detail, see [meta](#).

Returns A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

`great_expectations.data_context`

Subpackages

`great_expectations.data_context.store`

Submodules

`great_expectations.data_context.store.database_store_backend`

Module Contents

Classes

<code>DatabaseStoreBackend(credentials, table_name, key_columns, fixed_length_key=True)</code>	A store backend acts as a key-value store that can accept tuples as keys, to abstract away
--	--

`great_expectations.data_context.store.database_store_backend.sqlalchemy`

`great_expectations.data_context.store.database_store_backend.logger`

class `great_expectations.data_context.store.database_store_backend.DatabaseStoreBackend` (*credentials, table_name, key_columns, fixed_length_key=True*)

Bases: `great_expectations.data_context.store.store_backend.StoreBackend`

A store backend acts as a key-value store that can accept tuples as keys, to abstract away reading and writing to a persistence layer.

In general a StoreBackend implementation must provide implementations of:

- `_get`
- `_set`
- `list_keys`
- `_has_key`

`_get` (*self, key*)

`_set` (*self, key, value, **kwargs*)

```
abstract _move (self)
_has_key (self, key)
list_keys (self, prefix=())
remove_key (self, key)
```

`great_expectations.data_context.store.expectations_store`

Module Contents

Classes

<code>ExpectationsStore</code> (store_backend=None, time_environment=None)	run-	A store is responsible for reading and writing Great Expectations objects
---	------	---

class `great_expectations.data_context.store.expectations_store.ExpectationsStore` (*store_backend*, *run-time_environment*)

Bases: `great_expectations.data_context.store.store.Store`

A store is responsible for reading and writing Great Expectations objects to appropriate backends. It provides a generic API that the DataContext can use independently of any particular ORM and backend.

An implementation of a store will generally need to define the following:

- `serialize`
- `deserialize`
- `_key_class` (class of expected key type)

All keys must have a `to_tuple()` method.

`_key_class`

`remove_key` (*self*, *key*)

`serialize` (*self*, *key*, *value*)

`deserialize` (*self*, *key*, *value*)

`great_expectations.data_context.store.html_site_store`

Module Contents

Classes

<code>HtmlSiteStore</code> (store_backend=None, time_environment=None)	run-
---	------

`great_expectations.data_context.store.html_site_store.logger`

```
class great_expectations.data_context.store.html_site_store.HtmlSiteStore (store_backend=None,
                                                                           run-
                                                                           time_environment=None)

Bases: object

_key_class

get (self, key)

set (self, key, serialized_value)

get_url_for_resource (self, resource_identifier=None, only_if_exists=True)
    Return the URL of the HTML document that renders a resource (e.g., an expectation suite or a validation
    result).

    Parameters resource_identifier – ExpectationSuiteIdentifier, ValidationResultIdenti-
        fier or any other type's identifier. The argument is optional - when not supplied, the method
        returns the URL of the index page.

    Returns URL (string)

_validate_key (self, key)

list_keys (self)

write_index_page (self, page)
    This third param_store has a special method, which uses a zero-length tuple as a key.

clean_site (self)

copy_static_assets (self, static_assets_source_dir=None)
    Copies static assets, using a special "static_assets" backend store that accepts variable-length tuples as
    keys, with no filepath_template.
```

`great_expectations.data_context.store.metric_store`

Module Contents

Classes

<code>MetricStore(store_backend=None)</code>	A MetricStore stores ValidationMetric information to be used between runs.
<code>EvaluationParameterStore(store_backend=None)</code>	A MetricStore stores ValidationMetric information to be used between runs.

```
class great_expectations.data_context.store.metric_store.MetricStore (store_backend=None)
    Bases: great_expectations.data_context.store.store.Store
```

A MetricStore stores ValidationMetric information to be used between runs.

```
_key_class

_validate_value (self, value)

serialize (self, key, value)

deserialize (self, key, value)
```

```
class great_expectations.data_context.store.metric_store.EvaluationParameterStore (store_backen
    Bases: great_expectations.data_context.store.metric_store.MetricStore
```

A MetricStore stores ValidationMetric information to be used between runs.

get_bind_params (*self*, *run_id*)

`great_expectations.data_context.store.query_store`

Module Contents

Classes

<code>SqlAlchemyQueryStore</code> (<i>credentials</i> , <i>queries=None</i> , <i>store_backend=None</i> , <i>runtime_environment=None</i>)	<code>SqlAlchemyQueryStore</code> stores queries by name, and makes it possible to retrieve the resulting value by query
---	--

`great_expectations.data_context.store.query_store.sqlalchemy`

`great_expectations.data_context.store.query_store.logger`

class `great_expectations.data_context.store.query_store.SqlAlchemyQueryStore` (*credentials*,
queries=None,
store_backend=None,
run-
time_environment=None)

Bases: `great_expectations.data_context.store.Store`

`SqlAlchemyQueryStore` stores queries by name, and makes it possible to retrieve the resulting value by query name.

_key_class

_convert_key (*self*, *key*)

get (*self*, *key*)

set (*self*, *key*, *value*)

get_query_result (*self*, *key*, *query_parameters=None*)

`great_expectations.data_context.store.store`

Module Contents

Classes

<code>Store</code> (<i>store_backend=None</i> , <i>time_environment=None</i>)	<i>run-</i>	A store is responsible for reading and writing Great Expectations objects
--	-------------	---

`great_expectations.data_context.store.store.logger`

class `great_expectations.data_context.store.store.Store` (*store_backend=None*, *run-*
time_environment=None)

Bases: `object`

A store is responsible for reading and writing Great Expectations objects to appropriate backends. It provides a generic API that the DataContext can use independently of any particular ORM and backend.

An implementation of a store will generally need to define the following:

- `serialize`
- `deserialize`
- `_key_class` (class of expected key type)

All keys must have a `to_tuple()` method.

`_key_class`

`_validate_key` (*self*, *key*)

`property store_backend` (*self*)

`serialize` (*self*, *key*, *value*)

`key_to_tuple` (*self*, *key*)

`tuple_to_key` (*self*, *tuple_*)

`deserialize` (*self*, *key*, *value*)

`get` (*self*, *key*)

`set` (*self*, *key*, *value*)

`list_keys` (*self*)

`has_key` (*self*, *key*)

`great_expectations.data_context.store.store_backend`

Module Contents

Classes

<code>StoreBackend</code> (<i>fixed_length_key=False</i>)	A store backend acts as a key-value store that can accept tuples as keys, to abstract away
<code>InMemoryStoreBackend</code> (<i>runtime_environment=None</i> , <i>fixed_length_key=False</i>)	Uses an in-memory dictionary as a store backend.

class `great_expectations.data_context.store.store_backend.StoreBackend` (*fixed_length_key=False*)
 Bases: `object`

A store backend acts as a key-value store that can accept tuples as keys, to abstract away reading and writing to a persistence layer.

In general a `StoreBackend` implementation must provide implementations of:

- `_get`
- `_set`
- `list_keys`
- `_has_key`

`IGNORED_FILES` = `['.ipynb_checkpoints']`

`property fixed_length_key` (*self*)

```
get (self, key)
set (self, key, value, **kwargs)
move (self, source_key, dest_key, **kwargs)
has_key (self, key)
get_url_for_key (self, key, protocol=None)
_validate_key (self, key)
_validate_value (self, value)
abstract _get (self, key)
abstract _set (self, key, value, **kwargs)
abstract _move (self, source_key, dest_key, **kwargs)
abstract list_keys (self, prefix=())
abstract remove_key (self, key)
abstract _has_key (self, key)
is_ignored_key (self, key)
```

```
class great_expectations.data_context.store.store_backend.InMemoryStoreBackend (runtime_environment, fixed_length_key:
```

Bases: [great_expectations.data_context.store.store_backend.StoreBackend](#)

Uses an in-memory dictionary as a store backend.

```
_get (self, key)
_set (self, key, value, **kwargs)
_move (self, source_key, dest_key, **kwargs)
list_keys (self, prefix=())
_has_key (self, key)
remove_key (self, key)
```

[great_expectations.data_context.store.tuple_store_backend](#)

Module Contents

Classes

TupleStoreBackend (filepath_template=None, filepath_prefix=None, filepath_suffix=None, forbidden_substrings=None, platform_specific_separator=True, fixed_length_key=False)	If filepath_template is provided, the key to this StoreBackend abstract class must be a tuple with
TupleFilesystemStoreBackend (base_directory, filepath_template=None, filepath_prefix=None, filepath_suffix=None, forbidden_substrings=None, platform_specific_separator=True, root_directory=None, fixed_length_key=False)	Uses a local filepath as a store.

continues on next page

Table 55 – continued from previous page

<code>TupleS3StoreBackend(bucket, prefix='', filepath_template=None, filepath_prefix=None, filepath_suffix=None, forbidden_substrings=None, platform_specific_separator=False, fixed_length_key=False)</code>	Uses an S3 bucket as a store.
<code>TupleGCSStoreBackend(bucket, project, prefix='', filepath_template=None, filepath_prefix=None, filepath_suffix=None, forbidden_substrings=None, platform_specific_separator=False, fixed_length_key=False, public_urls=True)</code>	Uses a GCS bucket as a store.

`great_expectations.data_context.store.tuple_store_backend.logger`

class `great_expectations.data_context.store.tuple_store_backend.TupleStoreBackend` (*filepath_template, filepath_prefix, filepath_suffix, forbidden_substrings, platform_specific_separator, fixed_length_key*)

Bases: `great_expectations.data_context.store.store_backend.StoreBackend`

If `filepath_template` is provided, the key to this `StoreBackend` abstract class must be a tuple with fixed length equal to the number of unique components matching the regex `r"{d+}"`

For example, in the following template path: `expectations/{0}/{1}/{2}/prefix-{2}.json`, keys must have three components.

`_validate_key(self, key)`

`_validate_value(self, value)`

`_convert_key_to_filepath(self, key)`

`_convert_filepath_to_key(self, filepath)`

`verify_that_key_to_filepath_operation_is_reversible(self)`

class `great_expectations.data_context.store.tuple_store_backend.TupleFilesystemStoreBackend`

Bases: `great_expectations.data_context.store.tuple_store_backend.TupleStoreBackend`

Uses a local filepath as a store.

The key to this `StoreBackend` must be a tuple with fixed length based on the `filepath_template`, or a variable-length tuple may be used and returned with an optional `filepath_suffix` (to be) added. The `filepath_template` is a string template used to convert the key to a filepath.

```
_get (self, key)  
_set (self, key, value, **kwargs)  
_move (self, source_key, dest_key, **kwargs)  
list_keys (self, prefix=())  
rrmdir (self, mroot, curpath)  
    recursively removes empty dirs between curpath and mroot inclusive  
remove_key (self, key)  
get_url_for_key (self, key, protocol=None)  
_has_key (self, key)
```

```
class great_expectations.data_context.store.tuple_store_backend.TupleS3StoreBackend (bucket,
```

```
    pre-  
    fix="",  
    filepath_t  
    filepath_p  
    filepath_s  
    for-  
    bid-  
    den_subs  
    plat-  
    form_spe  
    fixed_leng
```

```
Bases:          great_expectations.data_context.store.tuple_store_backend.  
              TupleStoreBackend
```

Uses an S3 bucket as a store.

The key to this StoreBackend must be a tuple with fixed length based on the `filepath_template`, or a variable-length tuple may be used and returned with an optional `filepath_suffix` (to be) added. The `filepath_template` is a string template used to convert the key to a filepath.

```
_get (self, key)  
_set (self, key, value, content_encoding='utf-8', content_type='application/json')  
_move (self, source_key, dest_key, **kwargs)  
list_keys (self)  
get_url_for_key (self, key, protocol=None)  
remove_key (self, key)  
_has_key (self, key)
```

Uses a GCS bucket as a store.

The `filepath_template` is a string template used to convert the key to a filepath.

```
_get (self, key)
```

```
list_keys (self)
```

```
remove_key (self, key)
```

```
_has_key (self, key)
```

Module Contents

Classes

<code>ValidationsStore</code> (store_backend=None, time_environment=None)	run-	A store is responsible for reading and writing Great Expectations objects
---	------	---

```
class great_expectations.data_context.store.validations_store.ValidationsStore (store_backend=None,
run-
time environment=None)
```

A store is responsible for reading and writing *Great Expectations* objects to appropriate backends. It provides a generic API that the *DataContext* can use independently of any particular ORM and backend.

An implementation of a store will generally need to define the following:

- `serialize`
- `deserialize`
- `_key_class` (class of expected key type)

All keys must have a `to_tuple()` method.

`_key_class`

`serialize` (*self*, *key*, *value*)

`deserialize` (*self*, *key*, *value*)

Package Contents

Classes

<code>DatabaseStoreBackend</code> (<i>credentials</i> , <i>table_name</i> , <i>key_columns</i> , <i>fixed_length_key=True</i>)	A store backend acts as a key-value store that can accept tuples as keys, to abstract away
<code>ExpectationsStore</code> (<i>store_backend=None</i> , <i>runtime_environment=None</i>)	A store is responsible for reading and writing Great Expectations objects
<code>HtmlSiteStore</code> (<i>store_backend=None</i> , <i>runtime_environment=None</i>)	
<code>EvaluationParameterStore</code> (<i>store_backend=None</i>)	A MetricStore stores ValidationMetric information to be used between runs.
<code>MetricStore</code> (<i>store_backend=None</i>)	A MetricStore stores ValidationMetric information to be used between runs.
<code>Store</code> (<i>store_backend=None</i> , <i>runtime_environment=None</i>)	A store is responsible for reading and writing Great Expectations objects
<code>InMemoryStoreBackend</code> (<i>runtime_environment=None</i> , <i>fixed_length_key=False</i>)	Uses an in-memory dictionary as a store backend.
<code>StoreBackend</code> (<i>fixed_length_key=False</i>)	A store backend acts as a key-value store that can accept tuples as keys, to abstract away
<code>TupleFilesystemStoreBackend</code> (<i>base_directory</i> , <i>filepath_template=None</i> , <i>filepath_prefix=None</i> , <i>filepath_suffix=None</i> , <i>forbidden_substrings=None</i> , <i>platform_specific_separator=True</i> , <i>root_directory=None</i> , <i>fixed_length_key=False</i>)	Uses a local filepath as a store.
<code>TupleGCSStoreBackend</code> (<i>bucket</i> , <i>project</i> , <i>prefix=''</i> , <i>filepath_template=None</i> , <i>filepath_prefix=None</i> , <i>filepath_suffix=None</i> , <i>forbidden_substrings=None</i> , <i>platform_specific_separator=False</i> , <i>fixed_length_key=False</i> , <i>public_urls=True</i>)	Uses a GCS bucket as a store.
<code>TupleS3StoreBackend</code> (<i>bucket</i> , <i>prefix=''</i> , <i>filepath_template=None</i> , <i>filepath_prefix=None</i> , <i>filepath_suffix=None</i> , <i>forbidden_substrings=None</i> , <i>platform_specific_separator=False</i> , <i>fixed_length_key=False</i>)	Uses an S3 bucket as a store.
<code>TupleStoreBackend</code> (<i>filepath_template=None</i> , <i>filepath_prefix=None</i> , <i>filepath_suffix=None</i> , <i>forbidden_substrings=None</i> , <i>platform_specific_separator=True</i> , <i>fixed_length_key=False</i>)	If <i>filepath_template</i> is provided, the key to this StoreBackend abstract class must be a tuple with

continues on next page

Table 57 – continued from previous page

<code>ValidationsStore(store_backend=None, run-time_environment=None)</code>	A store is responsible for reading and writing Great Expectations objects
--	---

```
class great_expectations.data_context.store.DatabaseStoreBackend(
    credentials,
    table_name,
    key_columns,
    fixed_length_key=True)

Bases: great_expectations.data_context.store.store_backend.StoreBackend
```

A store backend acts as a key-value store that can accept tuples as keys, to abstract away reading and writing to a persistence layer.

In general a StoreBackend implementation must provide implementations of:

- `_get`
- `_set`
- `list_keys`
- `_has_key`

```
_get(self, key)
_set(self, key, value, **kwargs)
abstract _move(self)
_has_key(self, key)
list_keys(self, prefix=())
remove_key(self, key)
```

```
class great_expectations.data_context.store.ExpectationsStore(
    store_backend=None,
    run-time_environment=None)

Bases: great_expectations.data_context.store.store.Store
```

A store is responsible for reading and writing Great Expectations objects to appropriate backends. It provides a generic API that the DataContext can use independently of any particular ORM and backend.

An implementation of a store will generally need to define the following:

- `serialize`
- `deserialize`
- `_key_class` (class of expected key type)

All keys must have a `to_tuple()` method.

```
_key_class
remove_key(self, key)
serialize(self, key, value)
deserialize(self, key, value)
```

```
class great_expectations.data_context.store.HtmlSiteStore(
    store_backend=None,
    run-time_environment=None)

Bases: object
```

```
_key_class
```

get (*self*, *key*)

set (*self*, *key*, *serialized_value*)

get_url_for_resource (*self*, *resource_identifier=None*, *only_if_exists=True*)

Return the URL of the HTML document that renders a resource (e.g., an expectation suite or a validation result).

Parameters **resource_identifier** – ExpectationSuiteIdentifier, ValidationResultIdentifier or any other type's identifier. The argument is optional - when not supplied, the method returns the URL of the index page.

Returns URL (string)

_validate_key (*self*, *key*)

list_keys (*self*)

write_index_page (*self*, *page*)

This third param_store has a special method, which uses a zero-length tuple as a key.

clean_site (*self*)

copy_static_assets (*self*, *static_assets_source_dir=None*)

Copies static assets, using a special “static_assets” backend store that accepts variable-length tuples as keys, with no filepath_template.

class great_expectations.data_context.store.**EvaluationParameterStore** (*store_backend=None*)

Bases: *great_expectations.data_context.store.metric_store.MetricStore*

A MetricStore stores ValidationMetric information to be used between runs.

get_bind_params (*self*, *run_id*)

class great_expectations.data_context.store.**MetricStore** (*store_backend=None*)

Bases: *great_expectations.data_context.store.store.Store*

A MetricStore stores ValidationMetric information to be used between runs.

_key_class

_validate_value (*self*, *value*)

serialize (*self*, *key*, *value*)

deserialize (*self*, *key*, *value*)

class great_expectations.data_context.store.**Store** (*store_backend=None*, *run-time_environment=None*)

Bases: object

A store is responsible for reading and writing Great Expectations objects to appropriate backends. It provides a generic API that the DataContext can use independently of any particular ORM and backend.

An implementation of a store will generally need to define the following:

- **serialize**
- **deserialize**
- **_key_class** (class of expected key type)

All keys must have a `to_tuple()` method.

_key_class

_validate_key (*self*, *key*)


```
property store_backend (self)
```

```
serialize (self, key, value)
```

```
key_to_tuple (self, key)
```

```
tuple_to_key (self, tuple_)
```

```
deserialize (self, key, value)
```

```
get (self, key)
```

```
set (self, key, value)
```

```
list_keys (self)
```

```
has_key (self, key)
```

```
class great_expectations.data_context.store.InMemoryStoreBackend (runtime_environment=None,
                                                                    fixed_length_key=False)
    Bases: great_expectations.data_context.store.store_backend.StoreBackend
```

Uses an in-memory dictionary as a store backend.

```
_get (self, key)
```

```
_set (self, key, value, **kwargs)
```

```
_move (self, source_key, dest_key, **kwargs)
```

```
list_keys (self, prefix=())
```

```
_has_key (self, key)
```

```
remove_key (self, key)
```

```
class great_expectations.data_context.store.StoreBackend (fixed_length_key=False)
    Bases: object
```

A store backend acts as a key-value store that can accept tuples as keys, to abstract away reading and writing to a persistence layer.

In general a StoreBackend implementation must provide implementations of:

- `_get`
- `_set`
- `list_keys`
- `_has_key`

```
IGNORED_FILES = ['.ipynb_checkpoints']
```

```
property fixed_length_key (self)
```

```
get (self, key)
```

```
set (self, key, value, **kwargs)
```

```
move (self, source_key, dest_key, **kwargs)
```

```
has_key (self, key)
```

```
get_url_for_key (self, key, protocol=None)
```

```
_validate_key (self, key)
```

```
_validate_value (self, value)
```

```
abstract _get (self, key)
```

```
abstract _set (self, key, value, **kwargs)
abstract _move (self, source_key, dest_key, **kwargs)
abstract list_keys (self, prefix=())
abstract remove_key (self, key)
abstract _has_key (self, key)
is_ignored_key (self, key)
```

```
class great_expectations.data_context.store.TupleFilesystemStoreBackend (base_directory,
                                                                           filepath_template=None,
                                                                           filepath_prefix=None,
                                                                           filepath_suffix=None,
                                                                           for-
                                                                           bid-
                                                                           den_substrings=None,
                                                                           plat-
                                                                           form_specific_separator=T
                                                                           root_directory=None,
                                                                           fixed_length_key=False)
```

Bases: [great_expectations.data_context.store.tuple_store_backend.TupleStoreBackend](#)

Uses a local filepath as a store.

The key to this StoreBackend must be a tuple with fixed length based on the filepath_template, or a variable-length tuple may be used and returned with an optional filepath_suffix (to be) added. The filepath_template is a string template used to convert the key to a filepath.

```
_get (self, key)
_set (self, key, value, **kwargs)
_move (self, source_key, dest_key, **kwargs)
list_keys (self, prefix=())
rmdir (self, mroot, curpath)
    recursively removes empty dirs between curpath and mroot inclusive
remove_key (self, key)
get_url_for_key (self, key, protocol=None)
_has_key (self, key)
```

```
class great_expectations.data_context.store.TupleGCSSStoreBackend (bucket,
                                                                      project,
                                                                      prefix="",
                                                                      filepath_template=None,
                                                                      filepath_prefix=None,
                                                                      filepath_suffix=None,
                                                                      forbid-
                                                                      den_substrings=None,
                                                                      plat-
                                                                      form_specific_separator=False,
                                                                      fixed_length_key=False,
                                                                      pub-
                                                                      lic_urls=True)
```

Bases: `great_expectations.data_context.store.tuple_store_backend.TupleStoreBackend`

Uses a GCS bucket as a store.

The key to this StoreBackend must be a tuple with fixed length based on the `filepath_template`, or a variable-length tuple may be used and returned with an optional `filepath_suffix` (to be) added.

The `filepath_template` is a string template used to convert the key to a filepath.

```
_move (self, source_key, dest_key, **kwargs)
_get (self, key)
_set (self, key, value, content_encoding='utf-8', content_type='application/json')
list_keys (self)
get_url_for_key (self, key, protocol=None)
remove_key (self, key)
_has_key (self, key)
```

```
class great_expectations.data_context.store.TupleS3StoreBackend (bucket,
                                                                    prefix="",
                                                                    filepath_template=None,
                                                                    filepath_prefix=None,
                                                                    filepath_suffix=None,
                                                                    forbid-
                                                                    den_substrings=None,
                                                                    plat-
                                                                    form_specific_separator=False,
                                                                    fixed_length_key=False)
```

Bases: `great_expectations.data_context.store.tuple_store_backend.TupleStoreBackend`

Uses an S3 bucket as a store.

The key to this StoreBackend must be a tuple with fixed length based on the `filepath_template`, or a variable-length tuple may be used and returned with an optional `filepath_suffix` (to be) added. The `filepath_template` is a string template used to convert the key to a filepath.

```
_get (self, key)
_set (self, key, value, content_encoding='utf-8', content_type='application/json')
_move (self, source_key, dest_key, **kwargs)
list_keys (self)
get_url_for_key (self, key, protocol=None)
remove_key (self, key)
_has_key (self, key)
```

```
class great_expectations.data_context.store.TupleStoreBackend (filepath_template=None,
                                                                    filepath_prefix=None,
                                                                    filepath_suffix=None,
                                                                    forbid-
                                                                    den_substrings=None,
                                                                    plat-
                                                                    form_specific_separator=True,
                                                                    fixed_length_key=False)
```

Bases: *great_expectations.data_context.store.store_backend.StoreBackend*

If filepath_template is provided, the key to this StoreBackend abstract class must be a tuple with fixed length equal to the number of unique components matching the regex `r''{d+}''`

For example, in the following template path: `expectations/{0}/{1}/{2}/prefix-{2}.json`, keys must have three components.

`_validate_key (self, key)`

`_validate_value (self, value)`

`_convert_key_to_filepath (self, key)`

`_convert_filepath_to_key (self, filepath)`

`verify_that_key_to_filepath_operation_is_reversible (self)`

class `great_expectations.data_context.store.ValidationsStore` (*store_backend=None, run-time_environment=None*)

Bases: *great_expectations.data_context.store.store.Store*

A store is responsible for reading and writing Great Expectations objects to appropriate backends. It provides a generic API that the DataContext can use independently of any particular ORM and backend.

An implementation of a store will generally need to define the following:

- `serialize`
- `deserialize`
- `_key_class` (class of expected key type)

All keys must have a `to_tuple()` method.

`_key_class`

`serialize (self, key, value)`

`deserialize (self, key, value)`

`great_expectations.data_context.types`

Submodules

`great_expectations.data_context.types.base`

Module Contents

Classes

<code>DataContextConfig</code>	<code>(config_version,</code>	<code>data-</code>
<code>sources,</code>	<code>expectations_store_name,</code>	<code>valida-</code>
<code>tions_store_name,</code>	<code>evaluation_parameter_store_name,</code>	
<code>plugins_directory,</code>	<code>validation_operators,</code>	<code>stores,</code>
<code>data_docs_sites,</code>	<code>config_variables_file_path=None,</code>	<code>anony-</code>
<code>mous_usage_statistics=None,</code>	<code>commented_map=None)</code>	

continues on next page

Table 58 – continued from previous page

<i>DatasourceConfig</i> (class_name, module_name=None, data_asset_type=None, batch_kwarg_generators=None, credentials=None, reader_method=None, limit=None, **kwargs)	
<i>AnonymizedUsageStatisticsConfig</i> (enabled=True, data_context_id=None, usage_statistics_url=None)	
<i>AnonymizedUsageStatisticsConfigSchema</i> (*, only: types.StrSequenceOrSet = None, exclude: types.StrSequenceOrSet = (), many: bool = False, context: typing.Dict = None, load_only: types.StrSequenceOrSet = (), dump_only: types.StrSequenceOrSet = (), partial: typing.Union[bool, types.StrSequenceOrSet] = False, unknown: str = None)	Base schema class with which to define custom schemas.
<i>DatasourceConfigSchema</i> (*, only: types.StrSequenceOrSet = None, exclude: types.StrSequenceOrSet = (), many: bool = False, context: typing.Dict = None, load_only: types.StrSequenceOrSet = (), dump_only: types.StrSequenceOrSet = (), partial: typing.Union[bool, types.StrSequenceOrSet] = False, unknown: str = None)	Base schema class with which to define custom schemas.
<i>DataContextConfigSchema</i> (*, only: types.StrSequenceOrSet = None, exclude: types.StrSequenceOrSet = (), many: bool = False, context: typing.Dict = None, load_only: types.StrSequenceOrSet = (), dump_only: types.StrSequenceOrSet = (), partial: typing.Union[bool, types.StrSequenceOrSet] = False, unknown: str = None)	Base schema class with which to define custom schemas.

```
great_expectations.data_context.types.base.logger
```

```
great_expectations.data_context.types.base.yaml
```

```
great_expectations.data_context.types.base.CURRENT_CONFIG_VERSION = 2
```

```
great_expectations.data_context.types.base.MINIMUM_SUPPORTED_CONFIG_VERSION = 2
```

```
great_expectations.data_context.types.base.DEFAULT_USAGE_STATISTICS_URL = https://stats.gre
```

```
class great_expectations.data_context.types.base.DataContextConfig(config_version,  
                                                                    data-  
                                                                    sources,  
                                                                    expecta-  
                                                                    tions_store_name,  
                                                                    valida-  
                                                                    tions_store_name,  
                                                                    evalua-  
                                                                    tion_parameter_store_name,  
                                                                    plug-  
                                                                    ins_directory,  
                                                                    valida-  
                                                                    tion_operators,  
                                                                    stores,  
                                                                    data_docs_sites,  
                                                                    con-  
                                                                    fig_variables_file_path=None,  
                                                                    anony-  
                                                                    mous_usage_statistics=None,  
                                                                    com-  
                                                                    mented_map=None)
```

Bases: *great_expectations.types.DictDot*

property **commented_map** (*self*)

property **config_version** (*self*)

classmethod **from_commented_map** (*cls, commented_map*)

to_yaml (*self, outfile*)

```
class great_expectations.data_context.types.base.DatasourceConfig(class_name,  
                                                                    mod-  
                                                                    ule_name=None,  
                                                                    data_asset_type=None,  
                                                                    batch_kwargs_generators=None,  
                                                                    creden-  
                                                                    tials=None,  
                                                                    reader_method=None,  
                                                                    limit=None,  
                                                                    **kwargs)
```

Bases: *great_expectations.types.DictDot*

property **class_name** (*self*)

property **module_name** (*self*)

```
class great_expectations.data_context.types.base.AnonymizedUsageStatisticsConfig(enabled=True,  
                                                                    data_context_  
                                                                    us-  
                                                                    age_statistics,
```

Bases: *great_expectations.types.DictDot*

property **enabled** (*self*)

property **data_context_id** (*self*)

property **explicit_id** (*self*)

property **usage_statistics_url** (*self*)

```

class great_expectations.data_context.types.base.AnonymizedUsageStatisticsConfigSchema (*,
only:
types
=
None
ex-
clude
types
=
(),
many
bool
=
False
con-
text:
typ-
ing.D
=
None
load_
types
=
(),
dump
types
=
(),
par-
tial:
typ-
ing.U
types
=
False
un-
know
str
=
None

```

Bases: `marshmallow.Schema`

Base schema class with which to define custom schemas.

Example usage:

```

import datetime as dt
from dataclasses import dataclass

from marshmallow import Schema, fields

@dataclass
class Album:
    title: str

```

(continues on next page)

(continued from previous page)

```

        release_date: dt.date

class AlbumSchema(Schema):
    title = fields.Str()
    release_date = fields.Date()

album = Album("Beggars Banquet", dt.date(1968, 12, 6))
schema = AlbumSchema()
data = schema.dump(album)
data # {'release_date': '1968-12-06', 'title': 'Beggars Banquet'}
```

Parameters

- **only** – Whitelist of the declared fields to select when instantiating the Schema. If None, all fields are used. Nested fields can be represented with dot delimiters.
- **exclude** – Blacklist of the declared fields to exclude when instantiating the Schema. If a field appears in both *only* and *exclude*, it is not used. Nested fields can be represented with dot delimiters.
- **many** – Should be set to *True* if *obj* is a collection so that the object will be serialized to a list.
- **context** – Optional context passed to *fields.Method* and *fields.Function* fields.
- **load_only** – Fields to skip during serialization (write-only fields)
- **dump_only** – Fields to skip during deserialization (read-only fields)
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to *Nested* fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.
- **unknown** – Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*.

Changed in version 3.0.0: *prefix* parameter removed.

Changed in version 2.0.0: *__validators__*, *__preprocessors__*, and *__data_handlers__* are removed in favor of *marshmallow.decorators.validates_schema*, *marshmallow.decorators.pre_load* and *marshmallow.decorators.post_dump*. *__accessor__* and *__error_handler__* are deprecated. Implement the *handle_error* and *get_attribute* methods instead.

data_context_id

enabled

usage_statistics_url

_explicit_url

make_usage_statistics_config(*self*, *data*, ***kwargs*)

filter_implicit(*self*, *data*, ***kwargs*)


```

class great_expectations.data_context.types.base.DatasourceConfigSchema(*,
                                                                    only:
                                                                    types.StrSequenceOrSet
                                                                    =
                                                                    None,
                                                                    ex-
                                                                    clude:
                                                                    types.StrSequenceOrSet
                                                                    =
                                                                    (),
                                                                    many:
                                                                    bool
                                                                    =
                                                                    False,
                                                                    con-
                                                                    text:
                                                                    typ-
                                                                    ing.Dict
                                                                    =
                                                                    None,
                                                                    load_only:
                                                                    types.StrSequenceOrSet
                                                                    =
                                                                    (),
                                                                    dump_only:
                                                                    types.StrSequenceOrSet
                                                                    =
                                                                    (),
                                                                    par-
                                                                    tial:
                                                                    typ-
                                                                    ing.Union[bool,
                                                                    types.StrSequenceOrSet]
                                                                    =
                                                                    False,
                                                                    un-
                                                                    known:
                                                                    str
                                                                    =
                                                                    None)

```

Bases: `marshmallow.Schema`

Base schema class with which to define custom schemas.

Example usage:

```

import datetime as dt
from dataclasses import dataclass

from marshmallow import Schema, fields


@dataclass
class Album:
    title: str

```

(continues on next page)

(continued from previous page)

```
release_date: dt.date

class AlbumSchema(Schema):
    title = fields.Str()
    release_date = fields.Date()

album = Album("Beggars Banquet", dt.date(1968, 12, 6))
schema = AlbumSchema()
data = schema.dump(album)
data # {'release_date': '1968-12-06', 'title': 'Beggars Banquet'}
```

Parameters

- **only** – Whitelist of the declared fields to select when instantiating the Schema. If None, all fields are used. Nested fields can be represented with dot delimiters.
- **exclude** – Blacklist of the declared fields to exclude when instantiating the Schema. If a field appears in both *only* and *exclude*, it is not used. Nested fields can be represented with dot delimiters.
- **many** – Should be set to *True* if *obj* is a collection so that the object will be serialized to a list.
- **context** – Optional context passed to `fields.Method` and `fields.Function` fields.
- **load_only** – Fields to skip during serialization (write-only fields)
- **dump_only** – Fields to skip during deserialization (read-only fields)
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to *Nested* fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.
- **unknown** – Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*.

Changed in version 3.0.0: *prefix* parameter removed.

Changed in version 2.0.0: `__validators__`, `__preprocessors__`, and `__data_handlers__` are removed in favor of `marshmallow.decorators.validates_schema`, `marshmallow.decorators.pre_load` and `marshmallow.decorators.post_dump`. `__accessor__` and `__error_handler__` are deprecated. Implement the `handle_error` and `get_attribute` methods instead.

class Meta

unknown

class_name

module_name

data_asset_type

batch_kwargs_generators

credentials

validate_schema (*self*, *data*, ***kwargs*)

```
make_datasource_config(self, data, **kwargs)
```

```
class great_expectations.data_context.types.base.DataContextConfigSchema(*,
    only:
        types.StrSequenceOrSet
    =
    None,
    ex-
    clude:
        types.StrSequenceOrSet
    =
    (),
    many:
        bool
    =
    False,
    con-
    text:
        typ-
        ing.Dict
    =
    None,
    load_only:
        types.StrSequenceOrSet
    =
    (),
    dump_only:
        types.StrSequenceOrSet
    =
    (),
    par-
    tial:
        typ-
        ing.Union[bool,
        types.StrSequenceOrSet]
    =
    False,
    un-
    known:
        str
    =
    None)
```

Bases: `marshmallow.Schema`

Base schema class with which to define custom schemas.

Example usage:

```
import datetime as dt
from dataclasses import dataclass

from marshmallow import Schema, fields

@dataclass
class Album:
```

(continues on next page)

(continued from previous page)

```
title: str
release_date: dt.date

class AlbumSchema(Schema):
    title = fields.Str()
    release_date = fields.Date()

album = Album("Beggars Banquet", dt.date(1968, 12, 6))
schema = AlbumSchema()
data = schema.dump(album)
data # {'release_date': '1968-12-06', 'title': 'Beggars Banquet'}
```

Parameters

- **only** – Whitelist of the declared fields to select when instantiating the Schema. If None, all fields are used. Nested fields can be represented with dot delimiters.
- **exclude** – Blacklist of the declared fields to exclude when instantiating the Schema. If a field appears in both *only* and *exclude*, it is not used. Nested fields can be represented with dot delimiters.
- **many** – Should be set to *True* if *obj* is a collection so that the object will be serialized to a list.
- **context** – Optional context passed to `fields.Method` and `fields.Function` fields.
- **load_only** – Fields to skip during serialization (write-only fields)
- **dump_only** – Fields to skip during deserialization (read-only fields)
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to Nested fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.
- **unknown** – Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*.

Changed in version 3.0.0: *prefix* parameter removed.

Changed in version 2.0.0: `__validators__`, `__preprocessors__`, and `__data_handlers__` are removed in favor of `marshmallow.decorators.validates_schema`, `marshmallow.decorators.pre_load` and `marshmallow.decorators.post_dump`. `__accessor__` and `__error_handler__` are deprecated. Implement the `handle_error` and `get_attribute` methods instead.

config_version

datasources

expectations_store_name

validations_store_name

evaluation_parameter_store_name

plugins_directory

validation_operators

stores

`data_docs_sites``config_variables_file_path``anonymous_usage_statistics``handle_error(self, exc, data, **kwargs)`

Log and raise our custom exception when (de)serialization fails.

`validate_schema(self, data, **kwargs)``great_expectations.data_context.types.base.DataContextConfigSchema``great_expectations.data_context.types.base.datasourceConfigSchema``great_expectations.data_context.types.base.anonymizedUsageStatisticsSchema``great_expectations.data_context.types.base_resource_identifiers``great_expectations.data_context.types.resource_identifiers`

Module Contents

Classes

<code>ExpectationSuiteIdentifier(expectation_suite_name: str)</code>	DataContextKey objects are used to uniquely identify resources used by the DataContext.
<code>ExpectationSuiteIdentifierSchema(*, only: types.StrSequenceOrSet = None, exclude: types.StrSequenceOrSet = (), many: bool = False, context: typing.Dict = None, load_only: types.StrSequenceOrSet = (), dump_only: types.StrSequenceOrSet = (), partial: typing.Union[bool, types.StrSequenceOrSet] = False, unknown: str = None)</code>	Base schema class with which to define custom schemas.
<code>BatchIdentifier(batch_identifier: Union[BatchKwargs, dict, str], data_asset_name: str = None)</code>	A BatchIdentifier tracks
<code>BatchIdentifierSchema(*, only: types.StrSequenceOrSet = None, exclude: types.StrSequenceOrSet = (), many: bool = False, context: typing.Dict = None, load_only: types.StrSequenceOrSet = (), dump_only: types.StrSequenceOrSet = (), partial: typing.Union[bool, types.StrSequenceOrSet] = False, unknown: str = None)</code>	Base schema class with which to define custom schemas.
<code>ValidationResultIdentifier(expectation_suite_identifier: str, run_id, batch_identifier)</code>	A ValidationResultIdentifier identifies a validation result by the fully qualified expectation_suite_identifier
<code>ValidationResultIdentifierSchema(*, only: types.StrSequenceOrSet = None, exclude: types.StrSequenceOrSet = (), many: bool = False, context: typing.Dict = None, load_only: types.StrSequenceOrSet = (), dump_only: types.StrSequenceOrSet = (), partial: typing.Union[bool, types.StrSequenceOrSet] = False, unknown: str = None)</code>	Base schema class with which to define custom schemas.

continues on next page

Table 59 – continued from previous page

<i>SiteSectionIdentifier</i> (site_section_name, source_identifier)	re-	DataContextKey objects are used to uniquely identify resources used by the DataContext.
---	-----	---

`great_expectations.data_context.types.resource_identifiers.logger`

class `great_expectations.data_context.types.resource_identifiers.ExpectationSuiteIdentifier`

Bases: `great_expectations.core.data_context_key.DataContextKey`

DataContextKey objects are used to uniquely identify resources used by the DataContext.

A DataContextKey is designed to support clear naming with multiple representations including a hashable version making it suitable for use as the key in a dictionary.

property `expectation_suite_name` (*self*)

to_tuple (*self*)

to_fixed_length_tuple (*self*)

classmethod `from_tuple` (*cls*, *tuple_*)

classmethod `from_fixed_length_tuple` (*cls*, *tuple_*)

__repr__ (*self*)

Return repr(self).

class great_expectations.data_context.types.resource_identifiers.**ExpectationSuiteIdentifier**

Bases: `marshmallow.Schema`

Base schema class with which to define custom schemas.

Example usage:

```
import datetime as dt
from dataclasses import dataclass

from marshmallow import Schema, fields


@dataclass
class Album:
    title: str
```

(continues on next page)

(continued from previous page)

```

        release_date: dt.date

class AlbumSchema(Schema):
    title = fields.Str()
    release_date = fields.Date()

album = Album("Beggars Banquet", dt.date(1968, 12, 6))
schema = AlbumSchema()
data = schema.dump(album)
data # {'release_date': '1968-12-06', 'title': 'Beggars Banquet'}
```

Parameters

- **only** – Whitelist of the declared fields to select when instantiating the Schema. If None, all fields are used. Nested fields can be represented with dot delimiters.
- **exclude** – Blacklist of the declared fields to exclude when instantiating the Schema. If a field appears in both *only* and *exclude*, it is not used. Nested fields can be represented with dot delimiters.
- **many** – Should be set to *True* if *obj* is a collection so that the object will be serialized to a list.
- **context** – Optional context passed to `fields.Method` and `fields.Function` fields.
- **load_only** – Fields to skip during serialization (write-only fields)
- **dump_only** – Fields to skip during deserialization (read-only fields)
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to *Nested* fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.
- **unknown** – Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*.

Changed in version 3.0.0: *prefix* parameter removed.

Changed in version 2.0.0: `__validators__`, `__preprocessors__`, and `__data_handlers__` are removed in favor of `marshmallow.decorators.validates_schema`, `marshmallow.decorators.pre_load` and `marshmallow.decorators.post_dump`. `__accessor__` and `__error_handler__` are deprecated. Implement the `handle_error` and `get_attribute` methods instead.

expectation_suite_name

make_expectation_suite_identifier(*self*, *data*, ***kwargs*)

```

class great_expectations.data_context.types.resource_identifiers.BatchIdentifier(batch_identifier
    Union[BatchKey,
    dict,
    str],
    data_asset_name: str
    =
    None)
```

Bases: `great_expectations.core.data_context_key.DataContextKey`

A BatchIdentifier tracks

property `batch_identifier` (*self*)

property `data_asset_name` (*self*)

to_tuple (*self*)

classmethod `from_tuple` (*cls*, *tuple_*)

```
class great_expectations.data_context.types.resource_identifiers.BatchIdentifierSchema(*,
only:
types
=
None
ex-
clude
types
=
(),
many
bool
=
False
con-
text:
typ-
ing.D
=
None
load_
types
=
(),
dump
types
=
(),
par-
tial:
typ-
ing.U
types
=
False
un-
know
str
=
None
```

Bases: `marshmallow.Schema`

Base schema class with which to define custom schemas.

Example usage:

```
import datetime as dt
from dataclasses import dataclass

from marshmallow import Schema, fields

@dataclass
class Album:
    title: str
    release_date: dt.date

class AlbumSchema(Schema):
    title = fields.Str()
    release_date = fields.Date()

album = Album("Beggars Banquet", dt.date(1968, 12, 6))
schema = AlbumSchema()
data = schema.dump(album)
data  # {'release_date': '1968-12-06', 'title': 'Beggars Banquet'}
```

Parameters

- **only** – Whitelist of the declared fields to select when instantiating the Schema. If None, all fields are used. Nested fields can be represented with dot delimiters.
- **exclude** – Blacklist of the declared fields to exclude when instantiating the Schema. If a field appears in both *only* and *exclude*, it is not used. Nested fields can be represented with dot delimiters.
- **many** – Should be set to *True* if *obj* is a collection so that the object will be serialized to a list.
- **context** – Optional context passed to *fields.Method* and *fields.Function* fields.
- **load_only** – Fields to skip during serialization (write-only fields)
- **dump_only** – Fields to skip during deserialization (read-only fields)
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to Nested fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.
- **unknown** – Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*.

Changed in version 3.0.0: *prefix* parameter removed.

Changed in version 2.0.0: *__validators__*, *__preprocessors__*, and *__data_handlers__* are removed in favor of *marshmallow.decorators.validates_schema*, *marshmallow.decorators.pre_load* and *marshmallow.decorators.post_dump*. *__accessor__* and *__error_handler__* are deprecated. Implement the *handle_error* and *get_attribute* methods instead.

batch_identifier

data_asset_name

make_batch_identifier(*self*, *data*, ***kwargs*)

```
class great_expectations.data_context.types.resource_identifiers.ValidationResultIdentifier
```

Bases: *great_expectations.core.data_context_key.DataContextKey*

A ValidationResultIdentifier identifies a validation result by the fully qualified expectation_suite_idenfier and run_id.

```
property expectation_suite_identifier(self)
```

```
property run_id(self)
```

```
property batch_identifier(self)
```

```
to_tuple(self)
```

```
to_fixed_length_tuple(self)
```

```
classmethod from_tuple(cls, tuple_)
```

```
classmethod from_fixed_length_tuple(cls, tuple_)
```

```
classmethod from_object(cls, validation_result)
```

class great_expectations.data_context.types.resource_identifiers.ValidationResultIdentifier

Bases: `marshmallow.Schema`

Base schema class with which to define custom schemas.

Example usage:

```
import datetime as dt
from dataclasses import dataclass

from marshmallow import Schema, fields


@dataclass
class Album:
    title: str
```

(continues on next page)

(continued from previous page)

```

        release_date: dt.date

class AlbumSchema(Schema):
    title = fields.Str()
    release_date = fields.Date()

album = Album("Beggars Banquet", dt.date(1968, 12, 6))
schema = AlbumSchema()
data = schema.dump(album)
data # {'release_date': '1968-12-06', 'title': 'Beggars Banquet'}
```

Parameters

- **only** – Whitelist of the declared fields to select when instantiating the Schema. If None, all fields are used. Nested fields can be represented with dot delimiters.
- **exclude** – Blacklist of the declared fields to exclude when instantiating the Schema. If a field appears in both *only* and *exclude*, it is not used. Nested fields can be represented with dot delimiters.
- **many** – Should be set to *True* if *obj* is a collection so that the object will be serialized to a list.
- **context** – Optional context passed to *fields.Method* and *fields.Function* fields.
- **load_only** – Fields to skip during serialization (write-only fields)
- **dump_only** – Fields to skip during deserialization (read-only fields)
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to *Nested* fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.
- **unknown** – Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*.

Changed in version 3.0.0: *prefix* parameter removed.

Changed in version 2.0.0: *__validators__*, *__preprocessors__*, and *__data_handlers__* are removed in favor of *marshmallow.decorators.validates_schema*, *marshmallow.decorators.pre_load* and *marshmallow.decorators.post_dump*. *__accessor__* and *__error_handler__* are deprecated. Implement the *handle_error* and *get_attribute* methods instead.

expectation_suite_identifier

run_id

batch_identifier

make_validation_result_identifier(*self*, *data*, ***kwargs*)

class great_expectations.data_context.types.resource_identifiers.**SiteSectionIdentifier**(*site_id*, *re-source*)

Bases: *great_expectations.core.data_context_key.DataContextKey*

DataContextKey objects are used to uniquely identify resources used by the *DataContext*.

A `DataContextKey` is designed to support clear naming with multiple representations including a hashable version making it suitable for use as the key in a dictionary.

property `site_section_name` (*self*)

property `resource_identifier` (*self*)

to_tuple (*self*)

classmethod `from_tuple` (*cls*, *tuple_*)

`great_expectations.data_context.types.resource_identifiers.expectationSuiteIdentifierSchema`

`great_expectations.data_context.types.resource_identifiers.validationResultIdentifierSchema`

`great_expectations.data_context.types.resource_identifiers.runIdentifierSchema`

`great_expectations.data_context.types.resource_identifiers.batchIdentifierSchema`

Submodules

`great_expectations.data_context.data_context`

Module Contents

Classes

<code>BaseDataContext</code> (<i>project_config</i> , <i>context_root_dir</i> =None, <i>runtime_environment</i> =None)	con-	This class implements most of the functionality of <code>DataContext</code> , with a few exceptions.
<code>DataContext</code> (<i>context_root_dir</i> =None, <i>runtime_environment</i> =None)	run-	A <code>DataContext</code> represents a Great Expectations project. It organizes storage and access for
<code>ExplorerDataContext</code> (<i>context_root_dir</i> =None, <i>expectation_explorer</i> =True)	ex-	A <code>DataContext</code> represents a Great Expectations project. It organizes storage and access for

Functions

`_get_metric_configuration_tuples`(*metric_configuration*,
base_kwargs=None)

`great_expectations.data_context.data_context.SQLAlchemyError`

`great_expectations.data_context.data_context.logger`

`great_expectations.data_context.data_context.yaml`

`great_expectations.data_context.data_context.default_flow_style = False`

class `great_expectations.data_context.data_context.BaseDataContext` (*project_config*,
con-
text_root_dir=None,
run-
time_environment=None)

Bases: `object`

This class implements most of the functionality of `DataContext`, with a few exceptions.

1. BaseDataContext does not attempt to keep its project_config in sync with a file on disc.
2. **BaseDataContext doesn't attempt to "guess" paths or objects types. Instead, that logic is pushed into DataContext class.**

Together, these changes make BaseDataContext class more testable.

```

PROFILING_ERROR_CODE_TOO_MANY_DATA_ASSETS = 2
PROFILING_ERROR_CODE_SPECIFIED_DATA_ASSETS_NOT_FOUND = 3
PROFILING_ERROR_CODE_NO_BATCH_KWARGS_GENERATORS_FOUND = 4
PROFILING_ERROR_CODE_MULTIPLE_BATCH_KWARGS_GENERATORS_FOUND = 5
UNCOMMITTED_DIRECTORIES = ['data_docs', 'validations']
GE_UNCOMMITTED_DIR = uncommitted
CHECKPOINTS_DIR = checkpoints
BASE_DIRECTORIES
NOTEBOOK_SUBDIRECTORIES = ['pandas', 'spark', 'sql']
GE_DIR = great_expectations
GE_YML = great_expectations.yml
GE_EDIT_NOTEBOOK_DIR
FALSEY_STRINGS = ['FALSE', 'false', 'False', 'f', 'F', '0']
GLOBAL_CONFIG_PATHS
classmethod validate_config(cls, project_config)
    _build_store(self, store_name, store_config)
    _init_stores(self, store_configs)
        Initialize all Stores for this DataContext.

```

Stores are a good fit for reading/writing objects that:

1. follow a clear key-value pattern, and
2. are usually edited programmatically, using the Context

In general, Stores should take over most of the reading and writing to disk that DataContext had previously done. As of 9/21/2019, the following Stores had not yet been implemented

- great_expectations.yml
- expectations
- data documentation
- config_variables
- anything accessed via write_resource

Note that stores do NOT manage plugins.

```

_apply_global_config_overrides(self)
_get_global_config_value(self, environment_variable=None, conf_file_section=None,
                        conf_file_option=None)
_check_global_usage_statistics_opt_out(self)

```

`_initialize_usage_statistics` (*self*, *usage_statistics_config*: *AnonymizedUsageStatisticsConfig*)

Initialize the usage statistics system.

`add_store` (*self*, *store_name*, *store_config*)

Add a new Store to the DataContext and (for convenience) return the instantiated Store object.

Parameters

- **`store_name`** (*str*) – a key for the new Store in `self._stores`
- **`store_config`** (*dict*) – a config for the Store to add

Returns `store` (Store)

`add_validation_operator` (*self*, *validation_operator_name*, *validation_operator_config*)

Add a new ValidationOperator to the DataContext and (for convenience) return the instantiated object.

Parameters

- **`validation_operator_name`** (*str*) – a key for the new ValidationOperator in `self._validation_operators`
- **`validation_operator_config`** (*dict*) – a config for the ValidationOperator to add

Returns `validation_operator` (ValidationOperator)

`_normalize_absolute_or_relative_path` (*self*, *path*)

`_normalize_store_path` (*self*, *resource_store*)

`get_docs_sites_urls` (*self*, *resource_identifier*=None, *site_name*: *Optional[str]* = None, *only_if_exists*=True)

Get URLs for a resource for all data docs sites.

This function will return URLs for any configured site even if the sites have not been built yet.

Parameters

- **`resource_identifier`** (*object*) – optional. It can be an identifier of ExpectationSuite's, ValidationResults and other resources that have typed identifiers. If not provided, the method will return the URLs of the index page.
- **`site_name`** – Optionally specify which site to open. If not specified, return all urls in the project.

Returns

a list of URLs. Each item is the URL for the resource for a data docs site

Return type `list`

`_load_site_builder_from_site_config` (*self*, *site_config*)

`open_data_docs` (*self*, *resource_identifier*: *Optional[str]* = None, *site_name*: *Optional[str]* = None, *only_if_exists*=True)

A stdlib cross-platform way to open a file in a browser.

Parameters

- **`resource_identifier`** – ExpectationSuiteIdentifier, ValidationResultIdentifier or any other type's identifier. The argument is optional - when not supplied, the method returns the URL of the index page.
- **`site_name`** – Optionally specify which site to open. If not specified, open all docs found in the project.

property root_directory (*self*)

The root directory for configuration objects in the data context; the location in which `great_expectations.yml` is located.

property plugins_directory (*self*)

The directory in which custom plugin modules should be placed.

property _project_config_with_variables_substituted (*self*)

property anonymous_usage_statistics (*self*)

property stores (*self*)

A single holder for all Stores in this context

property datasources (*self*)

A single holder for all Datasources in this context

property expectations_store_name (*self*)

property data_context_id (*self*)

property instance_id (*self*)

_load_config_variables_file (*self*)

Get all config variables from the default location.

get_config_with_variables_substituted (*self*, *config=None*)

save_config_variable (*self*, *config_variable_name*, *value*)

Save config variable value

Parameters

- **config_variable_name** – name of the property
- **value** – the value to save for the property

Returns

None

delete_datasource (*self*, *datasource_name=None*)

Delete a data source :param *datasource_name*: The name of the datasource to delete.

Raises ValueError – If the datasource name isn't provided or cannot be found.

get_available_data_asset_names (*self*, *datasource_names=None*, *batch_kwargs_generator_names=None*)

Inspect datasource and batch kwargs generators to provide available *data_asset* objects.

Parameters

- **datasource_names** – list of datasources for which to provide available *data_asset_name* objects. If None, return available data assets for all datasources.
- **batch_kwargs_generator_names** – list of batch kwargs generators for which to provide available
- **objects.** (*data_asset_name*) –

Returns

Dictionary describing available data assets

```
{
  datasource_name: {
    batch_kwargs_generator_name: [ data_asset_1, data_asset_2, ... ]
    ...
  }
}
```

(continues on next page)

(continued from previous page)

```

    )
    ...
)

```

Return type `data_asset_names` (dict)

build_batch_kwarg (*self*, *datasource*, *batch_kwarg_generator*, *data_asset_name=None*, *partition_id=None*, ***kwargs*)

Builds batch kwarg using the provided *datasource*, *batch kwarg generator*, and *batch_parameters*.

Parameters

- **datasource** (*str*) – the name of the *datasource* for which to build *batch_kwarg*
- **batch_kwarg_generator** (*str*) – the name of the *batch kwarg generator* to use to build *batch_kwarg*
- **data_asset_name** (*str*) – an optional name *batch_parameter*
- ****kwargs** – additional *batch_parameters*

Returns `BatchKwarg`

get_batch (*self*, *batch_kwarg*: *Union[dict, BatchKwarg]*, *expectation_suite_name*: *Union[str, ExpectationSuite]*, *data_asset_type=None*, *batch_parameters=None*)

Build a batch of data using *batch_kwarg*, and return a `DataAsset` with *expectation_suite_name* attached. If *batch_parameters* are included, they will be available as attributes of the batch.

Parameters

- **batch_kwarg** – the *batch_kwarg* to use; must include a *datasource* key
- **expectation_suite_name** – The `ExpectationSuite` or the name of the *expectation_suite* to get
- **data_asset_type** – the type of *data_asset* to build, with associated *expectation* implementations. This can generally be inferred from the *datasource*.
- **batch_parameters** – optional parameters to store as the reference description of the batch. They should reflect parameters that would provide the passed *BatchKwarg*.

Returns `DataAsset`

run_validation_operator (*self*, *validation_operator_name*, *assets_to_validate*, *run_id=None*, *evaluation_parameters=None*, *run_name=None*, *run_time=None*, *result_format={'result_format': 'SUMMARY'}*, ***kwargs*)

Run a validation operator to validate data assets and to perform the business logic around validation that the operator implements.

Parameters

- **validation_operator_name** – name of the operator, as appears in the context's config file
- **assets_to_validate** – a list that specifies the data assets that the operator will validate. The members of the list can be either batches, or a tuple that will allow the operator to fetch the batch: (*batch_kwarg*, *expectation_suite_name*)
- **run_name** – The *run_name* for the validation; if *None*, a default value will be used
- ****kwargs** – Additional *kwargs* to pass to the validation operator

Returns `ValidationOperatorResult`

list_validation_operator_names (*self*)

add_datasource (*self*, *name*, *initialize=True*, ***kwargs*)

Add a new datasource to the data context, with configuration provided as kwargs. :param name: the name for the new datasource to add :param initialize: if False, add the datasource to the config, but do not initialize it, for example if a user needs to debug database connectivity.

Parameters **kwargs** (*keyword arguments*) – the configuration for the new datasource

Returns datasource (Datasource)

add_batch_kwargs_generator (*self*, *datasource_name*, *batch_kwargs_generator_name*, *class_name*, ***kwargs*)

Add a batch kwargs generator to the named datasource, using the provided configuration.

Parameters

- **datasource_name** – name of datasource to which to add the new batch kwargs generator
- **batch_kwargs_generator_name** – name of the generator to add
- **class_name** – class of the batch kwargs generator to add
- ****kwargs** – batch kwargs generator configuration, provided as kwargs

Returns:

get_config (*self*)

_build_datasource_from_config (*self*, *name*, *config*)

get_datasource (*self*, *datasource_name: str = 'default'*)

Get the named datasource

Parameters **datasource_name** (*str*) – the name of the datasource from the configuration

Returns datasource (Datasource)

list_expectation_suites (*self*)

Return a list of available expectation suite names.

list_datasources (*self*)

List currently-configured datasources on this context.

Returns each dictionary includes “name”, “class_name”, and “module_name” keys

Return type List(dict)

list_stores (*self*)

List currently-configured Stores on this context

list_validation_operators (*self*)

List currently-configured Validation Operators on this context

create_expectation_suite (*self*, *expectation_suite_name*, *overwrite_existing=False*)

Build a new expectation suite and save it into the data_context expectation store.

Parameters

- **expectation_suite_name** – The name of the expectation_suite to create
- **overwrite_existing** (*boolean*) – Whether to overwrite expectation suite if expectation suite with given name already exists.

Returns A new (empty) expectation suite.

delete_expectation_suite (*self*, *expectation_suite_name*)

Delete specified expectation suite from data_context expectation store.

Parameters **expectation_suite_name** – The name of the expectation_suite to create

Returns True for Success and False for Failure.

get_expectation_suite (*self*, *expectation_suite_name*)

Get a named expectation suite for the provided data_asset_name.

Parameters **expectation_suite_name** (*str*) – the name for the expectation suite

Returns expectation_suite

list_expectation_suite_names (*self*)

Lists the available expectation suite names

save_expectation_suite (*self*, *expectation_suite*, *expectation_suite_name=None*)

Save the provided expectation suite into the DataContext.

Parameters

- **expectation_suite** – the suite to save
- **expectation_suite_name** – the name of this expectation suite. If no name is provided the name will be read from the suite

Returns None

_store_metrics (*self*, *requested_metrics*, *validation_results*, *target_store_name*)

requested_metrics is a dictionary like this:

requested_metrics:

*: # The asterisk here matches *any expectation suite name # use the 'kwargs' key to request metrics that are defined by kwargs, # for example because they are defined only for a particular column # - column: # Age: # - expect_column_min_to_be_between.result.observed_value*

- statistics.evaluated_expectations
- statistics.successful_expectations

Parameters

- **requested_metrics** –
- **validation_results** –
- **target_store_name** –

Returns:

store_validation_result_metrics (*self*, *requested_metrics*, *validation_results*, *target_store_name*)

store_evaluation_parameters (*self*, *validation_results*, *target_store_name=None*)

property_evaluation_parameter_store (*self*)

property_evaluation_parameter_store_name (*self*)

property_validations_store_name (*self*)

property_validations_store (*self*)

_compile_evaluation_parameter_dependencies (*self*)

get_validation_result (*self, expectation_suite_name, run_id=None, batch_identifier=None, validations_store_name=None, failed_only=False*)

Get validation results from a configured store.

Parameters

- **data_asset_name** – name of data asset for which to get validation result
- **expectation_suite_name** – expectation_suite name for which to get validation result (default: “default”)
- **run_id** – run_id for which to get validation result (if None, fetch the latest result by alphanumeric sort)
- **validations_store_name** – the name of the store from which to get validation results
- **failed_only** – if True, filter the result to return only failed expectations

Returns validation_result

update_return_obj (*self, data_asset, return_obj*)

Helper called by data_asset.

Parameters

- **data_asset** – The data_asset whose validation produced the current return object
- **return_obj** – the return object to update

Returns the return object, potentially changed into a widget by the configured expectation explorer

Return type return_obj

build_data_docs (*self, site_names=None, resource_identifiers=None, dry_run=False*)

Build Data Docs for your project.

These make it simple to visualize data quality in your project. These include Expectations, Validations & Profiles. The are built for all Datasources from JSON artifacts in the local repo including validations & profiles from the uncommitted directory.

Parameters

- **site_names** – if specified, build data docs only for these sites, otherwise, build all the sites specified in the context’s config
- **resource_identifiers** – a list of resource identifiers (ExpectationSuiteIdentifier, ValidationResultIdentifier). If specified, rebuild HTML (or other views the data docs sites are rendering) only for the resources in this list. This supports incremental build of data docs sites (e.g., when a new validation result is created) and avoids full rebuild.
- **dry_run** – a flag, if True, the method returns a structure containing the URLs of the sites that *would* be built, but it does not build these sites. The motivation for adding this flag was to allow the CLI to display the the URLs before building and to let users confirm.

Returns A dictionary with the names of the updated data documentation sites as keys and the the location info of their index.html files as values

clean_data_docs (*self, site_name=None*)

```
profile_datasource (self,          datasource_name,          batch_kwargs_generator_name=None,
                    data_assets=None,  max_data_assets=20,  profile_all_data_assets=True,
                    profiler=BasicDatasetProfiler, profiler_configuration=None, dry_run=False,
                    run_id=None,      additional_batch_kwargs=None,      run_name=None,
                    run_time=None)
```

Profile the named datasource using the named profiler.

Parameters

- **datasource_name** – the name of the datasource for which to profile data_assets
- **batch_kwargs_generator_name** – the name of the batch kwargs generator to use to get batches
- **data_assets** – list of data asset names to profile
- **max_data_assets** – if the number of data assets the batch kwargs generator yields is greater than this max_data_assets, profile_all_data_assets=True is required to profile all
- **profile_all_data_assets** – when True, all data assets are profiled, regardless of their number
- **profiler** – the profiler class to use
- **profiler_configuration** – Optional profiler configuration dict
- **dry_run** – when true, the method checks arguments and reports if can profile or specifies the arguments that are missing
- **additional_batch_kwargs** – Additional keyword arguments to be provided to get_batch when loading the data asset.

Returns

A dictionary:

```
{
    "success": True/False,
    "results": List of (expectation_suite, EVR) tuples for each of
    ↳ the data_assets found in the datasource
}
```

When success = False, the error details are under “error” key

```
profile_data_asset (self,          datasource_name,          batch_kwargs_generator_name=None,
                    data_asset_name=None,          batch_kwargs=None,          expecta-
                    tion_suite_name=None,          profiler=BasicDatasetProfiler,          pro-
                    filer_configuration=None, run_id=None, additional_batch_kwargs=None,
                    run_name=None, run_time=None)
```

Profile a data asset

Parameters

- **datasource_name** – the name of the datasource to which the profiled data asset belongs
- **batch_kwargs_generator_name** – the name of the batch kwargs generator to use to get batches (only if batch_kwargs are not provided)
- **data_asset_name** – the name of the profiled data asset
- **batch_kwargs** – optional - if set, the method will use the value to fetch the batch to be profiled. If not passed, the batch kwargs generator (generator_name arg) will choose a batch

- **profiler** – the profiler class to use
- **profiler_configuration** – Optional profiler configuration dict
- **run_name** – optional - if set, the validation result created by the profiler will be under the provided run_name
- **additional_batch_kwargs** –

:returns A dictionary:

```
{
    "success": True/False,
    "results": List of (expectation_suite, EVR) tuples for each of the
    ↳ data_assets found in the datasource
}
```

When success = False, the error details are under “error” key

```
class great_expectations.data_context.data_context.DataContext (context_root_dir=None,
                                                                run-
                                                                time_environment=None)
```

Bases: *great_expectations.data_context.data_context.BaseDataContext*

A DataContext represents a Great Expectations project. It organizes storage and access for expectation suites, datasources, notification settings, and data fixtures.

The DataContext is configured via a yml file stored in a directory called great_expectations; the configuration file as well as managed expectation suites should be stored in version control.

Use the *create* classmethod to create a new empty config, or instantiate the DataContext by passing the path to an existing data context root directory.

DataContexts use data sources you’re already familiar with. BatchKwargGenerators help introspect data stores and data execution frameworks (such as airflow, Nifi, dbt, or dagster) to describe and produce batches of data ready for analysis. This enables fetching, validation, profiling, and documentation of your data in a way that is meaningful within your existing infrastructure and work environment.

DataContexts use a datasource-based namespace, where each accessible type of data has a three-part normalized *data_asset_name*, consisting of *datasource/generator/data_asset_name*.

- The datasource actually connects to a source of materialized data and returns Great Expectations DataAssets connected to a compute environment and ready for validation.
- The BatchKwargGenerator knows how to introspect datasources and produce identifying “batch_kwargs” that define particular slices of data.
- The data_asset_name is a specific name – often a table name or other name familiar to users – that batch kwargs generators can slice into batches.

An expectation suite is a collection of expectations ready to be applied to a batch of data. Since in many projects it is useful to have different expectations evaluate in different contexts—profiling vs. testing; warning vs. error; high vs. low compute; ML model or dashboard—suites provide a namespace option for selecting which expectations a DataContext returns.

In many simple projects, the datasource or batch kwargs generator name may be omitted and the DataContext will infer the correct name when there is no ambiguity.

Similarly, if no expectation suite name is provided, the DataContext will assume the name “default”.

```
classmethod create(cls, project_root_dir=None, usage_statistics_enabled=True, run-
                  time_environment=None)
    Build a new great_expectations directory and DataContext object in the provided project_root_dir.
```

create will not create a new “great_expectations” directory in the provided folder, provided one does not already exist. Then, it will initialize a new DataContext in that folder and write the resulting config.

Parameters

- **project_root_dir** – path to the root directory in which to create a new great_expectations directory
- **runtime_environment** – a dictionary of config variables that
- **both those set in config_variables.yml and the environment (override) –**

Returns DataContext

classmethod all_uncommitted_directories_exist (*cls, ge_dir*)

Check if all uncommitted directories exist.

classmethod config_variables_yaml_exist (*cls, ge_dir*)

Check if all config_variables.yml exists.

classmethod write_config_variables_template_to_disk (*cls, uncommitted_dir*)

classmethod write_project_template_to_disk (*cls, ge_dir, us-*
age_statistics_enabled=True)

classmethod scaffold_directories (*cls, base_dir*)

Safely create GE directories for a new project.

classmethod scaffold_custom_data_docs (*cls, plugins_dir*)

Copy custom data docs templates

classmethod scaffold_notebooks (*cls, base_dir*)

Copy template notebooks into the notebooks directory for a project.

_load_project_config (*self*)

Reads the project configuration from the project configuration file. The file may contain \${SOME_VARIABLE} variables - see self._project_config_with_variables_substituted for how these are substituted.

Returns the configuration object read from the file

list_checkpoints (*self*)

List checkpoints. (Experimental)

get_checkpoint (*self, checkpoint_name: str*)

Load a checkpoint. (Experimental)

_list_ymls_in_checkpoints_directory (*self*)

_save_project_config (*self*)

Save the current project to disk.

add_store (*self, store_name, store_config*)

Add a new Store to the DataContext and (for convenience) return the instantiated Store object.

Parameters

- **store_name** (*str*) – a key for the new Store in in self._stores
- **store_config** (*dict*) – a config for the Store to add

Returns store (Store)

add_datasource (*self*, *name*, ***kwargs*)

Add a new datasource to the data context, with configuration provided as kwargs. :param name: the name for the new datasource to add :param initialize: if False, add the datasource to the config, but do not

initialize it, for example if a user needs to debug database connectivity.

Parameters *kwargs* (*keyword arguments*) – the configuration for the new datasource

Returns datasource (Datasource)

classmethod **find_context_root_dir** (*cls*)

classmethod **get_ge_config_version** (*cls*, *context_root_dir=None*)

classmethod **set_ge_config_version** (*cls*, *config_version*, *context_root_dir=None*, *validate_config_version=True*)

classmethod **find_context_yaml_file** (*cls*, *search_start_dir=None*)

Search for the yaml file starting here and moving upward.

classmethod **does_config_exist_on_disk** (*cls*, *context_root_dir*)

Return True if the great_expectations.yml exists on disk.

classmethod **is_project_initialized** (*cls*, *ge_dir*)

Return True if the project is initialized.

To be considered initialized, all of the following must be true: - all project directories exist (including uncommitted directories) - a valid great_expectations.yml is on disk - a config_variables.yml is on disk - the project has at least one datasource - the project has at least one suite

classmethod **does_project_have_a_datasource_in_config_file** (*cls*, *ge_dir*)

classmethod **_does_context_have_at_least_one_datasource** (*cls*, *ge_dir*)

classmethod **_does_context_have_at_least_one_suite** (*cls*, *ge_dir*)

classmethod **_attempt_context_instantiation** (*cls*, *ge_dir*)

static **_validate_checkpoint** (*checkpoint: dict*, *checkpoint_name: str*)

class great_expectations.data_context.data_context.**ExplorerDataContext** (*context_root_dir=None*,
*ex-
pec-
ta-
tion_explorer=True*)

Bases: `great_expectations.data_context.data_context.DataContext`

A DataContext represents a Great Expectations project. It organizes storage and access for expectation suites, datasources, notification settings, and data fixtures.

The DataContext is configured via a yaml file stored in a directory called great_expectations; the configuration file as well as managed expectation suites should be stored in version control.

Use the *create* classmethod to create a new empty config, or instantiate the DataContext by passing the path to an existing data context root directory.

DataContexts use data sources you're already familiar with. BatchKwargGenerators help introspect data stores and data execution frameworks (such as airflow, Nifi, dbt, or dagster) to describe and produce batches of data ready for analysis. This enables fetching, validation, profiling, and documentation of your data in a way that is meaningful within your existing infrastructure and work environment.

DataContexts use a datasource-based namespace, where each accessible type of data has a three-part normalized *data_asset_name*, consisting of *datasource/generator/data_asset_name*.

- The datasource actually connects to a source of materialized data and returns Great Expectations DataAssets connected to a compute environment and ready for validation.
- The BatchKwargGenerator knows how to introspect datasources and produce identifying “batch_kwarg” that define particular slices of data.
- The `data_asset_name` is a specific name – often a table name or other name familiar to users – that batch kwarg generators can slice into batches.

An expectation suite is a collection of expectations ready to be applied to a batch of data. Since in many projects it is useful to have different expectations evaluate in different contexts—profiling vs. testing; warning vs. error; high vs. low compute; ML model or dashboard—suites provide a namespace option for selecting which expectations a `DataContext` returns.

In many simple projects, the datasource or batch kwarg generator name may be omitted and the `DataContext` will infer the correct name when there is no ambiguity.

Similarly, if no expectation suite name is provided, the `DataContext` will assume the name “default”.

update_return_obj (*self*, *data_asset*, *return_obj*)

Helper called by `data_asset`.

Parameters

- **data_asset** – The `data_asset` whose validation produced the current return object
- **return_obj** – the return object to update

Returns the return object, potentially changed into a widget by the configured expectation explorer

Return type `return_obj`

`great_expectations.data_context.data_context._get_metric_configuration_tuples` (*metric_configuration_tuples*, *base_kwarg=Non*

`great_expectations.data_context.templates`

Module Contents

```
great_expectations.data_context.templates.PROJECT_HELP_COMMENT =
```

```
# Welcome to Great Expectations! Always know what to expect from your data. ## Here you can define datasources,
batch kwarg generators, integrations and # more. This file is intended to be committed to your repo. For help with #
configuration please: # - Read our docs: https://docs.greatexpectations.io/en/latest/reference/data\_context\_reference.html#configuration # - Join our slack channel: http://greatexpectations.io/slack
```

```
config_version: 2
```

```
# Datasources tell Great Expectations where your data lives and how to get it. # You can use the CLI command
great_expectations datasource new to help you # add a new datasource. Read more at https://docs.greatexpectations.io/en/latest/features/datasource.html datasources: { }
```

```
great_expectations.data_context.templates.CONFIG_VARIABLES_INTRO =
```

```
# This config file supports variable substitution which enables: 1) keeping # secrets out of source control & 2)
environment-based configuration changes # such as staging vs prod. ## When GE encounters substitution syntax
(like my_key: ${my_value} or # my_key: $my_value) in the great_expectations.yml file, it will attempt # to replace the
value of my_key with the value from an environment # variable my_value or a corresponding key read from this config
file, # which is defined through the config_variables_file_path. # Environment variables take precedence over variables
```

defined here. ## Substitution values defined here can be a simple (non-nested) value, # nested value such as a dictionary, or an environment variable (i.e. `{ENV_VAR}`) ### https://docs.greatexpectations.io/en/latest/how_to_guides/configuring_data_contexts/how_to_use_a_yaml_file_or_environment_variables_to_populate_credentials.html

```
great_expectations.data_context.templates.CONFIG_VARIABLES_TEMPLATE
great_expectations.data_context.templates.PROJECT_OPTIONAL_CONFIG_COMMENT
great_expectations.data_context.templates.ANONYMIZED_USAGE_STATISTICS_ENABLED =
anonymous_usage_statistics: enabled: True
great_expectations.data_context.templates.ANONYMIZED_USAGE_STATISTICS_DISABLED =
anonymous_usage_statistics: enabled: False
great_expectations.data_context.templates.PROJECT_TEMPLATE_USAGE_STATISTICS_ENABLED
great_expectations.data_context.templates.PROJECT_TEMPLATE_USAGE_STATISTICS_DISABLED
```

great_expectations.data_context.util

Module Contents

Functions

<code>load_class(class_name, module_name)</code>	Dynamically load a class from strings or raise a helpful error.
<code>instantiate_class_from_config(config, run-time_environment, config_defaults=None)</code>	Build a GE class from configuration dictionaries.
<code>format_dict_for_error_message(dict_)</code>	
<code>substitute_config_variable(template_str, config_variables_dict)</code>	This method takes a string, and if it contains a pattern <code>{SOME_VARIABLE}</code> or <code>\$SOME_VARIABLE</code> ,
<code>substitute_all_config_variables(data, replace_variables_dict)</code>	Substitute all config variables of the form <code>{SOME_VARIABLE}</code> in a dictionary-like
<code>file_relative_path(dunderfile, relative_path)</code>	This function is useful when one needs to load a file that is

great_expectations.data_context.util.logger

great_expectations.data_context.util.load_class (*class_name, module_name*)
Dynamically load a class from strings or raise a helpful error.

great_expectations.data_context.util.instantiate_class_from_config (*config, run-time_environment, config_defaults=None*)

Build a GE class from configuration dictionaries.

great_expectations.data_context.util.format_dict_for_error_message (*dict_*)

great_expectations.data_context.util.substitute_config_variable (*template_str, config_variables_dict*)

This method takes a string, and if it contains a pattern `{SOME_VARIABLE}` or `$SOME_VARIABLE`, returns a string where the pattern is replaced with the value of `SOME_VARIABLE`, otherwise returns the string unchanged.

If the environment variable `SOME_VARIABLE` is set, the method uses its value for substitution. If it is not set, the value of `SOME_VARIABLE` is looked up in the config variables store (file). If it is not found there, the input string is returned as is.

Parameters

- **template_str** – a string that might or might not be of the form `${SOME_VARIABLE}` or `$SOME_VARIABLE`
- **config_variables_dict** – a dictionary of config variables. It is loaded from the config variables store (by default, “uncommitted/config_variables.yml” file)

Returns

`great_expectations.data_context.util.substitute_all_config_variables` (*data*,
re-place_variables_dict)

Substitute all config variables of the form `${SOME_VARIABLE}` in a dictionary-like config object for their values.

The method traverses the dictionary recursively.

Parameters

- **data** –
- **replace_variables_dict** –

Returns a dictionary with all the variables replaced with their values

`great_expectations.data_context.util.file_relative_path` (*dunderfile*, *relative_path*)

This function is useful when one needs to load a file that is relative to the position of the current file. (Such as when you encode a configuration file path in source file and want in runnable in any current working directory)

It is meant to be used like the following: `file_relative_path(__file__, 'path/relative/to/file')`

H/T https://github.com/dagster-io/dagster/blob/8a250e9619a49e8bff8e9aa7435df89c2d2ea039/python_modules/dagster/dagster/utis/__init__.py#L34

Package Contents

Classes

<code>BaseDataContext</code> (<i>project_config</i> , <i>context_root_dir=None</i> , <i>runtime_environment=None</i>)	con-	This class implements most of the functionality of DataContext, with a few exceptions.
<code>DataContext</code> (<i>context_root_dir=None</i> , <i>time_environment=None</i>)	run-	A DataContext represents a Great Expectations project. It organizes storage and access for
<code>ExplorerDataContext</code> (<i>context_root_dir=None</i> , <i>expectation_explorer=True</i>)	ex-	A DataContext represents a Great Expectations project. It organizes storage and access for

class `great_expectations.data_context.BaseDataContext` (*project_config*,
context_root_dir=None, *runtime_environment=None*)

Bases: `object`

This class implements most of the functionality of DataContext, with a few exceptions.

1. BaseDataContext does not attempt to keep its `project_config` in sync with a file on disc.
2. BaseDataContext doesn't attempt to “guess” paths or objects types. Instead, that logic is pushed

into DataContext class.

Together, these changes make BaseDataContext class more testable.

```

PROFILING_ERROR_CODE_TOO_MANY_DATA_ASSETS = 2
PROFILING_ERROR_CODE_SPECIFIED_DATA_ASSETS_NOT_FOUND = 3
PROFILING_ERROR_CODE_NO_BATCH_KWARGS_GENERATORS_FOUND = 4
PROFILING_ERROR_CODE_MULTIPLE_BATCH_KWARGS_GENERATORS_FOUND = 5
UNCOMMITTED_DIRECTORIES = ['data_docs', 'validations']
GE_UNCOMMITTED_DIR = uncommitted
CHECKPOINTS_DIR = checkpoints
BASE_DIRECTORIES
NOTEBOOK_SUBDIRECTORIES = ['pandas', 'spark', 'sql']
GE_DIR = great_expectations
GE_YML = great_expectations.yml
GE_EDIT_NOTEBOOK_DIR
FALSEY_STRINGS = ['FALSE', 'false', 'False', 'f', 'F', '0']
GLOBAL_CONFIG_PATHS
classmethod validate_config(cls, project_config)

```

```
_build_store(self, store_name, store_config)
```

```
_init_stores(self, store_configs)
```

Initialize all Stores for this DataContext.

Stores are a good fit for reading/writing objects that:

1. follow a clear key-value pattern, and
2. are usually edited programmatically, using the Context

In general, Stores should take over most of the reading and writing to disk that DataContext had previously done. As of 9/21/2019, the following Stores had not yet been implemented

- great_expectations.yml
- expectations
- data documentation
- config_variables
- anything accessed via write_resource

Note that stores do NOT manage plugins.

```
_apply_global_config_overrides(self)
```

```
_get_global_config_value(self, environment_variable=None, conf_file_section=None,
                           conf_file_option=None)
```

```
_check_global_usage_statistics_opt_out(self)
```

```
_initialize_usage_statistics(self, usage_statistics_config: AnonymizedUsageStatisticsCon-
                               fig)
```

Initialize the usage statistics system.

add_store (*self*, *store_name*, *store_config*)

Add a new Store to the DataContext and (for convenience) return the instantiated Store object.

Parameters

- **store_name** (*str*) – a key for the new Store in in *self._stores*
- **store_config** (*dict*) – a config for the Store to add

Returns store (Store)

add_validation_operator (*self*, *validation_operator_name*, *validation_operator_config*)

Add a new ValidationOperator to the DataContext and (for convenience) return the instantiated object.

Parameters

- **validation_operator_name** (*str*) – a key for the new ValidationOperator in in *self._validation_operators*
- **validation_operator_config** (*dict*) – a config for the ValidationOperator to add

Returns validation_operator (ValidationOperator)

_normalize_absolute_or_relative_path (*self*, *path*)

_normalize_store_path (*self*, *resource_store*)

get_docs_sites_urls (*self*, *resource_identifier=None*, *site_name: Optional[str] = None*,
only_if_exists=True)

Get URLs for a resource for all data docs sites.

This function will return URLs for any configured site even if the sites have not been built yet.

Parameters

- **resource_identifier** (*object*) – optional. It can be an identifier of ExpectationSuite's, ValidationResults and other resources that have typed identifiers. If not provided, the method will return the URLs of the index page.
- **site_name** – Optionally specify which site to open. If not specified, return all urls in the project.

Returns

a list of URLs. Each item is the URL for the resource for a data docs site

Return type list

_load_site_builder_from_site_config (*self*, *site_config*)

open_data_docs (*self*, *resource_identifier: Optional[str] = None*, *site_name: Optional[str] = None*,
only_if_exists=True)

A stdlib cross-platform way to open a file in a browser.

Parameters

- **resource_identifier** – ExpectationSuiteIdentifier, ValidationResultIdentifier or any other type's identifier. The argument is optional - when not supplied, the method returns the URL of the index page.
- **site_name** – Optionally specify which site to open. If not specified, open all docs found in the project.

property root_directory (*self*)

The root directory for configuration objects in the data context; the location in which `great_expectations.yml` is located.

property plugins_directory (*self*)

The directory in which custom plugin modules should be placed.

property _project_config_with_variables_substituted (*self*)

property anonymous_usage_statistics (*self*)

property stores (*self*)

A single holder for all Stores in this context

property datasources (*self*)

A single holder for all Datasources in this context

property expectations_store_name (*self*)

property data_context_id (*self*)

property instance_id (*self*)

_load_config_variables_file (*self*)

Get all config variables from the default location.

get_config_with_variables_substituted (*self*, *config=None*)

save_config_variable (*self*, *config_variable_name*, *value*)

Save config variable value

Parameters

- **config_variable_name** – name of the property
- **value** – the value to save for the property

Returns None

delete_datasource (*self*, *datasource_name=None*)

Delete a data source :param datasource_name: The name of the datasource to delete.

Raises ValueError – If the datasource name isn't provided or cannot be found.

get_available_data_asset_names (*self*, *datasource_names=None*, *batch_kwargs_generator_names=None*)

Inspect datasource and batch kwargs generators to provide available data_asset objects.

Parameters

- **datasource_names** – list of datasources for which to provide available data_asset_name objects. If None, return available data assets for all datasources.
- **batch_kwargs_generator_names** – list of batch kwargs generators for which to provide available
- **objects.** (*data_asset_name*) –

Returns

Dictionary describing available data assets

```
{
  datasource_name: {
    batch_kwargs_generator_name: [ data_asset_1, data_asset_2, ... ]
    ...
  }
  ...
}
```

Return type `data_asset_names` (dict)

build_batch_kwarg (*self*, *datasource*, *batch_kwarg_generator*, *data_asset_name=None*, *partition_id=None*, ***kwargs*)

Builds batch kwarg using the provided datasource, batch kwarg generator, and batch_parameters.

Parameters

- **datasource** (*str*) – the name of the datasource for which to build batch_kwarg
- **batch_kwarg_generator** (*str*) – the name of the batch kwarg generator to use to build batch_kwarg
- **data_asset_name** (*str*) – an optional name batch_parameter
- ****kwargs** – additional batch_parameters

Returns `BatchKwarg`

get_batch (*self*, *batch_kwarg*: *Union[dict, BatchKwarg]*, *expectation_suite_name*: *Union[str, ExpectationSuite]*, *data_asset_type=None*, *batch_parameters=None*)

Build a batch of data using batch_kwarg, and return a DataAsset with expectation_suite_name attached. If batch_parameters are included, they will be available as attributes of the batch.

Parameters

- **batch_kwarg** – the batch_kwarg to use; must include a datasource key
- **expectation_suite_name** – The ExpectationSuite or the name of the expectation_suite to get
- **data_asset_type** – the type of data_asset to build, with associated expectation implementations. This can generally be inferred from the datasource.
- **batch_parameters** – optional parameters to store as the reference description of the batch. They should reflect parameters that would provide the passed BatchKwarg.

Returns `DataAsset`

run_validation_operator (*self*, *validation_operator_name*, *assets_to_validate*, *run_id=None*, *evaluation_parameters=None*, *run_name=None*, *run_time=None*, *result_format={'result_format': 'SUMMARY'}*, ***kwargs*)

Run a validation operator to validate data assets and to perform the business logic around validation that the operator implements.

Parameters

- **validation_operator_name** – name of the operator, as appears in the context's config file
- **assets_to_validate** – a list that specifies the data assets that the operator will validate. The members of the list can be either batches, or a tuple that will allow the operator to fetch the batch: (batch_kwarg, expectation_suite_name)
- **run_name** – The run_name for the validation; if None, a default value will be used
- ****kwargs** – Additional kwargs to pass to the validation operator

Returns `ValidationOperatorResult`

list_validation_operator_names (*self*)

add_datasource (*self*, *name*, *initialize=True*, ***kwargs*)

Add a new datasource to the data context, with configuration provided as kwargs. :param name: the name for the new datasource to add :param initialize: if False, add the datasource to the config, but do not

initialize it, for example if a user needs to debug database connectivity.

Parameters **kwargs** (*keyword arguments*) – the configuration for the new datasource

Returns datasource (Datasource)

add_batch_kwargs_generator (*self*, *datasource_name*, *batch_kwargs_generator_name*, *class_name*, ***kwargs*)

Add a batch kwargs generator to the named datasource, using the provided configuration.

Parameters

- **datasource_name** – name of datasource to which to add the new batch kwargs generator
- **batch_kwargs_generator_name** – name of the generator to add
- **class_name** – class of the batch kwargs generator to add
- ****kwargs** – batch kwargs generator configuration, provided as kwargs

Returns:

get_config (*self*)

_build_datasource_from_config (*self*, *name*, *config*)

get_datasource (*self*, *datasource_name*: *str* = 'default')

Get the named datasource

Parameters **datasource_name** (*str*) – the name of the datasource from the configuration

Returns datasource (Datasource)

list_expectation_suites (*self*)

Return a list of available expectation suite names.

list_datasources (*self*)

List currently-configured datasources on this context.

Returns each dictionary includes “name”, “class_name”, and “module_name” keys

Return type List(dict)

list_stores (*self*)

List currently-configured Stores on this context

list_validation_operators (*self*)

List currently-configured Validation Operators on this context

create_expectation_suite (*self*, *expectation_suite_name*, *overwrite_existing=False*)

Build a new expectation suite and save it into the data_context expectation store.

Parameters

- **expectation_suite_name** – The name of the expectation_suite to create
- **overwrite_existing** (*boolean*) – Whether to overwrite expectation suite if expectation suite with given name already exists.

Returns A new (empty) expectation suite.

delete_expectation_suite (*self*, *expectation_suite_name*)

Delete specified expectation suite from data_context expectation store.

Parameters **expectation_suite_name** – The name of the expectation_suite to create

Returns True for Success and False for Failure.

get_expectation_suite (*self*, *expectation_suite_name*)

Get a named expectation suite for the provided data_asset_name.

Parameters **expectation_suite_name** (*str*) – the name for the expectation suite

Returns expectation_suite

list_expectation_suite_names (*self*)

Lists the available expectation suite names

save_expectation_suite (*self*, *expectation_suite*, *expectation_suite_name=None*)

Save the provided expectation suite into the DataContext.

Parameters

- **expectation_suite** – the suite to save
- **expectation_suite_name** – the name of this expectation suite. If no name is provided the name will be read from the suite

Returns None

_store_metrics (*self*, *requested_metrics*, *validation_results*, *target_store_name*)

requested_metrics is a dictionary like this:

requested_metrics:

*: # The asterisk here matches *any expectation suite name # use the 'kwargs' key to request metrics that are defined by kwargs, # for example because they are defined only for a particular column # - column: # Age: # - expect_column_min_to_be_between.result.observed_value*

- statistics.evaluated_expectations
- statistics.successful_expectations

Parameters

- **requested_metrics** –
- **validation_results** –
- **target_store_name** –

Returns:

store_validation_result_metrics (*self*, *requested_metrics*, *validation_results*, *target_store_name*)

store_evaluation_parameters (*self*, *validation_results*, *target_store_name=None*)

property_evaluation_parameter_store (*self*)

property_evaluation_parameter_store_name (*self*)

property_validations_store_name (*self*)

property_validations_store (*self*)

_compile_evaluation_parameter_dependencies (*self*)

get_validation_result (*self*, *expectation_suite_name*, *run_id=None*, *batch_identifier=None*, *validations_store_name=None*, *failed_only=False*)

Get validation results from a configured store.

Parameters

- **data_asset_name** – name of data asset for which to get validation result
- **expectation_suite_name** – expectation_suite name for which to get validation result (default: “default”)
- **run_id** – run_id for which to get validation result (if None, fetch the latest result by alphanumeric sort)
- **validations_store_name** – the name of the store from which to get validation results
- **failed_only** – if True, filter the result to return only failed expectations

Returns validation_result

update_return_obj (*self*, *data_asset*, *return_obj*)

Helper called by data_asset.

Parameters

- **data_asset** – The data_asset whose validation produced the current return object
- **return_obj** – the return object to update

Returns the return object, potentially changed into a widget by the configured expectation explorer

Return type return_obj

build_data_docs (*self*, *site_names=None*, *resource_identifiers=None*, *dry_run=False*)

Build Data Docs for your project.

These make it simple to visualize data quality in your project. These include Expectations, Validations & Profiles. The are built for all Datasources from JSON artifacts in the local repo including validations & profiles from the uncommitted directory.

Parameters

- **site_names** – if specified, build data docs only for these sites, otherwise, build all the sites specified in the context’s config
- **resource_identifiers** – a list of resource identifiers (ExpectationSuiteIdentifier, ValidationResultIdentifier). If specified, rebuild HTML (or other views the data docs sites are rendering) only for the resources in this list. This supports incremental build of data docs sites (e.g., when a new validation result is created) and avoids full rebuild.
- **dry_run** – a flag, if True, the method returns a structure containing the URLs of the sites that *would* be built, but it does not build these sites. The motivation for adding this flag was to allow the CLI to display the the URLs before building and to let users confirm.

Returns A dictionary with the names of the updated data documentation sites as keys and the the location info of their index.html files as values

clean_data_docs (*self*, *site_name=None*)

profile_datasource (*self*, *datasource_name*, *batch_kwargs_generator_name=None*, *data_assets=None*, *max_data_assets=20*, *profile_all_data_assets=True*, *profiler=BasicDatasetProfiler*, *profiler_configuration=None*, *dry_run=False*, *run_id=None*, *additional_batch_kwargs=None*, *run_name=None*, *run_time=None*)

Profile the named datasource using the named profiler.

Parameters

- **datasource_name** – the name of the datasource for which to profile data_assets
- **batch_kwargs_generator_name** – the name of the batch kwargs generator to use to get batches
- **data_assets** – list of data asset names to profile
- **max_data_assets** – if the number of data assets the batch kwargs generator yields is greater than this max_data_assets, profile_all_data_assets=True is required to profile all
- **profile_all_data_assets** – when True, all data assets are profiled, regardless of their number
- **profiler** – the profiler class to use
- **profiler_configuration** – Optional profiler configuration dict
- **dry_run** – when true, the method checks arguments and reports if can profile or specifies the arguments that are missing
- **additional_batch_kwargs** – Additional keyword arguments to be provided to get_batch when loading the data asset.

Returns

A dictionary:

```
{
    "success": True/False,
    "results": List of (expectation_suite, EVR) tuples for each of
    ↳ the data_assets found in the datasource
}
```

When success = False, the error details are under “error” key

```
profile_data_asset (self,          datasource_name,          batch_kwargs_generator_name=None,
                    data_asset_name=None,          batch_kwargs=None,          expecta-
                    tion_suite_name=None,          profiler=BasicDatasetProfiler,          pro-
                    filer_configuration=None, run_id=None, additional_batch_kwargs=None,
                    run_name=None, run_time=None)
```

Profile a data asset

Parameters

- **datasource_name** – the name of the datasource to which the profiled data asset belongs
- **batch_kwargs_generator_name** – the name of the batch kwargs generator to use to get batches (only if batch_kwargs are not provided)
- **data_asset_name** – the name of the profiled data asset
- **batch_kwargs** – optional - if set, the method will use the value to fetch the batch to be profiled. If not passed, the batch kwargs generator (generator_name arg) will choose a batch
- **profiler** – the profiler class to use
- **profiler_configuration** – Optional profiler configuration dict
- **run_name** – optional - if set, the validation result created by the profiler will be under the provided run_name
- **additional_batch_kwargs** –

:returns A dictionary:

```
{
    "success": True/False,
    "results": List of (expectation_suite, EVR) tuples for each of the
    ↳ data_assets found in the datasource
}
```

When success = False, the error details are under “error” key

class great_expectations.data_context.**DataContext** (*context_root_dir=None, run-
time_environment=None*)
Bases: *great_expectations.data_context.data_context.BaseDataContext*

A DataContext represents a Great Expectations project. It organizes storage and access for expectation suites, datasources, notification settings, and data fixtures.

The DataContext is configured via a yml file stored in a directory called great_expectations; the configuration file as well as managed expectation suites should be stored in version control.

Use the *create* classmethod to create a new empty config, or instantiate the DataContext by passing the path to an existing data context root directory.

DataContexts use data sources you’re already familiar with. BatchKwargGenerators help introspect data stores and data execution frameworks (such as airflow, Nifi, dbt, or dagster) to describe and produce batches of data ready for analysis. This enables fetching, validation, profiling, and documentation of your data in a way that is meaningful within your existing infrastructure and work environment.

DataContexts use a datasource-based namespace, where each accessible type of data has a three-part normalized *data_asset_name*, consisting of *datasource/generator/data_asset_name*.

- The datasource actually connects to a source of materialized data and returns Great Expectations DataAssets connected to a compute environment and ready for validation.
- The BatchKwargGenerator knows how to introspect datasources and produce identifying “batch_kwarg” that define particular slices of data.
- The *data_asset_name* is a specific name – often a table name or other name familiar to users – that batch kwarg generators can slice into batches.

An expectation suite is a collection of expectations ready to be applied to a batch of data. Since in many projects it is useful to have different expectations evaluate in different contexts—profiling vs. testing; warning vs. error; high vs. low compute; ML model or dashboard—suites provide a namespace option for selecting which expectations a DataContext returns.

In many simple projects, the datasource or batch kwarg generator name may be omitted and the DataContext will infer the correct name when there is no ambiguity.

Similarly, if no expectation suite name is provided, the DataContext will assume the name “default”.

classmethod create (*cls, project_root_dir=None, usage_statistics_enabled=True, run-
time_environment=None*)

Build a new great_expectations directory and DataContext object in the provided project_root_dir.

create will not create a new “great_expectations” directory in the provided folder, provided one does not already exist. Then, it will initialize a new DataContext in that folder and write the resulting config.

Parameters

- **project_root_dir** – path to the root directory in which to create a new great_expectations directory
- **runtime_environment** – a dictionary of config variables that

- both those set in `config_variables.yml` and the environment (*override*) –

Returns DataContext

classmethod `all_uncommitted_directories_exist` (*cls*, *ge_dir*)

Check if all uncommitted directories exist.

classmethod `config_variables_yml_exist` (*cls*, *ge_dir*)

Check if all config_variables.yml exists.

classmethod `write_config_variables_template_to_disk` (*cls*, *uncommitted_dir*)

classmethod `write_project_template_to_disk` (*cls*, *ge_dir*, *usage_statistics_enabled=True*)

classmethod `scaffold_directories` (*cls*, *base_dir*)

Safely create GE directories for a new project.

classmethod `scaffold_custom_data_docs` (*cls*, *plugins_dir*)

Copy custom data docs templates

classmethod `scaffold_notebooks` (*cls*, *base_dir*)

Copy template notebooks into the notebooks directory for a project.

load_project_config (*self*)

Reads the project configuration from the project configuration file. The file may contain \${SOME_VARIABLE} variables - see `self._project_config_with_variables_substituted` for how these are substituted.

Returns the configuration object read from the file

list_checkpoints (*self*)

List checkpoints. (Experimental)

get_checkpoint (*self*, *checkpoint_name: str*)

Load a checkpoint. (Experimental)

_list_ymls_in_checkpoints_directory (*self*)

_save_project_config (*self*)

Save the current project to disk.

add_store (*self*, *store_name*, *store_config*)

Add a new Store to the DataContext and (for convenience) return the instantiated Store object.

Parameters

- **store_name** (*str*) – a key for the new Store in `self._stores`
- **store_config** (*dict*) – a config for the Store to add

Returns store (Store)

add_datasource (*self*, *name*, ***kwargs*)

Add a new datasource to the data context, with configuration provided as kwargs. :param name: the name for the new datasource to add :param initialize: if False, add the datasource to the config, but do not initialize it, for example if a user needs to debug database connectivity.

Parameters **kwargs** (*keyword arguments*) – the configuration for the new datasource

Returns datasource (Datasource)

classmethod `find_context_root_dir` (*cls*)

```

classmethod get_ge_config_version (cls, context_root_dir=None)

classmethod set_ge_config_version (cls, config_version, context_root_dir=None, validate_config_version=True)

classmethod find_context_yaml_file (cls, search_start_dir=None)
    Search for the yaml file starting here and moving upward.

classmethod does_config_exist_on_disk (cls, context_root_dir)
    Return True if the great_expectations.yml exists on disk.

classmethod is_project_initialized (cls, ge_dir)
    Return True if the project is initialized.

    To be considered initialized, all of the following must be true: - all project directories exist (including uncommitted directories) - a valid great_expectations.yml is on disk - a config_variables.yml is on disk - the project has at least one datasource - the project has at least one suite

classmethod does_project_have_a_datasource_in_config_file (cls, ge_dir)

classmethod _does_context_have_at_least_one_datasource (cls, ge_dir)

classmethod _does_context_have_at_least_one_suite (cls, ge_dir)

classmethod _attempt_context_instantiation (cls, ge_dir)

static _validate_checkpoint (checkpoint: dict, checkpoint_name: str)

class great_expectations.data_context.ExplorerDataContext (context_root_dir=None, expectation_explorer=True)

```

Bases: `great_expectations.data_context.data_context.DataContext`

A DataContext represents a Great Expectations project. It organizes storage and access for expectation suites, datasources, notification settings, and data fixtures.

The DataContext is configured via a yaml file stored in a directory called `great_expectations`; the configuration file as well as managed expectation suites should be stored in version control.

Use the `create` classmethod to create a new empty config, or instantiate the DataContext by passing the path to an existing data context root directory.

DataContexts use data sources you're already familiar with. BatchKwargGenerators help introspect data stores and data execution frameworks (such as airflow, Nifi, dbt, or dagster) to describe and produce batches of data ready for analysis. This enables fetching, validation, profiling, and documentation of your data in a way that is meaningful within your existing infrastructure and work environment.

DataContexts use a datasource-based namespace, where each accessible type of data has a three-part normalized `data_asset_name`, consisting of `datasource/generator/data_asset_name`.

- The datasource actually connects to a source of materialized data and returns Great Expectations DataAssets connected to a compute environment and ready for validation.
- The BatchKwargGenerator knows how to introspect datasources and produce identifying "batch_kwargs" that define particular slices of data.
- The `data_asset_name` is a specific name – often a table name or other name familiar to users – that batch kwarg generators can slice into batches.

An expectation suite is a collection of expectations ready to be applied to a batch of data. Since in many projects it is useful to have different expectations evaluate in different contexts—profiling vs. testing; warning vs. error; high vs. low compute; ML model or dashboard—suites provide a namespace option for selecting which expectations a DataContext returns.

In many simple projects, the datasource or batch kwargs generator name may be omitted and the DataContext will infer the correct name when there is no ambiguity.

Similarly, if no expectation suite name is provided, the DataContext will assume the name “default”.

update_return_obj (*self*, *data_asset*, *return_obj*)

Helper called by *data_asset*.

Parameters

- **data_asset** – The *data_asset* whose validation produced the current return object
- **return_obj** – the return object to update

Returns the return object, potentially changed into a widget by the configured expectation explorer

Return type *return_obj*

`great_expectations.dataset`

Submodules

`great_expectations.dataset.dataset`

Module Contents

Classes

<code>MetaDataset(*args, **kwargs)</code>	Holds expectation decorators.
<code>Dataset(*args, **kwargs)</code>	Holds expectation decorators.

class `great_expectations.dataset.dataset.MetaDataset` (**args*, ***kwargs*)

Bases: `great_expectations.data_asset.data_asset.DataAsset`

Holds expectation decorators.

abstract classmethod `column_map_expectation` (*cls*, *func*)

Constructs an expectation using column-map semantics.

The `column_map_expectation` decorator handles boilerplate issues surrounding the common pattern of evaluating truthiness of some condition on a per-row basis.

Parameters **func** (*function*) – The function implementing a row-wise expectation. The function should take a column of data and return an equally-long column of boolean values corresponding to the truthiness of the underlying expectation.

Notes

`column_map_expectation` intercepts and takes action based on the following parameters: mostly (None or a float between 0 and 1): Return “*success*”: *True* if at least mostly fraction of values match the expectation. For more detail, see *mostly*.

`column_map_expectation` *excludes null values* from being passed to the function

Depending on the *result_format* selected, `column_map_expectation` can additional data to a return object, including *element_count*, *nonnull_values*, *nonnull_count*, *success_count*, *unexpected_list*, and *unexpected_index_list*. See `_format_map_output`

See also:

expect_column_values_to_be_in_set for an example of a `column_map_expectation`

classmethod `column_aggregate_expectation` (*cls, func*)

Constructs an expectation using column-aggregate semantics.

The `column_aggregate_expectation` decorator handles boilerplate issues surrounding the common pattern of evaluating truthiness of some condition on an aggregated-column basis.

Parameters `func` (*function*) – The function implementing an expectation using an aggregate property of a column. The function should take a column of data and return the aggregate value it computes.

Notes

`column_aggregate_expectation` *excludes null values* from being passed to the function

See also:

expect_column_mean_to_be_between for an example of a `column_aggregate_expectation`

class `great_expectations.dataset.dataset.Dataset` (**args, **kwargs*)

Bases: `great_expectations.dataset.dataset.MetaDataset`

Holds expectation decorators.

`_data_asset_type = Dataset`

`hashable_getters = ['get_column_min', 'get_column_max', 'get_column_mean', 'get_column...`

classmethod `from_dataset` (*cls, dataset=None*)

This base implementation naively passes arguments on to the real constructor, which is suitable really when a constructor knows to take its own type. In general, this should be overridden

abstract `get_row_count` (*self*)

Returns: int, table row count

abstract `get_column_count` (*self*)

Returns: int, table column count

abstract `get_table_columns` (*self*)

Returns: List[str], list of column names

abstract `get_column_nonnull_count` (*self, column*)

Returns: int

abstract `get_column_mean` (*self, column*)

Returns: float

abstract get_column_value_counts (*self*, *column*, *sort*='value', *collate*=None)

Get a series containing the frequency counts of unique values from the named column.

Parameters

- **column** – the column for which to obtain value_counts
- **sort** (*string*) – must be one of “value”, “count”, or “none”. - if “value” then values in the resulting partition object will be sorted lexicographically - if “count” then values will be sorted according to descending count (frequency) - if “none” then values will not be sorted
- **collate** (*string*) – the collate (sort) method to be used on supported backends (SqlAlchemy only)

Returns pd.Series of value counts for a column, sorted according to the value requested in sort

abstract get_column_sum (*self*, *column*)

Returns: float

abstract get_column_max (*self*, *column*, *parse_strings_as_datetimes*=False)

Returns: any

abstract get_column_min (*self*, *column*, *parse_strings_as_datetimes*=False)

Returns: any

abstract get_column_unique_count (*self*, *column*)

Returns: int

abstract get_column_modes (*self*, *column*)

Returns: List[any], list of modes (ties OK)

abstract get_column_median (*self*, *column*)

Returns: any

abstract get_column_quantiles (*self*, *column*, *quantiles*, *allow_relative_error*=False)

Get the values in column closest to the requested quantiles :param column: name of column :type column: string :param quantiles: the quantiles to return. quantiles *must* be a tuple to ensure caching is possible :type quantiles: tuple of float

Returns the nearest values in the dataset to those quantiles

Return type List[any]

abstract get_column_stdev (*self*, *column*)

Returns: float

get_column_partition (*self*, *column*, *bins*='uniform', *n_bins*=10, *allow_relative_error*=False)

Get a partition of the range of values in the specified column.

Parameters

- **column** – the name of the column
- **bins** – ‘uniform’ for evenly spaced bins or ‘quantile’ for bins spaced according to quantiles
- **n_bins** – the number of bins to produce
- **allow_relative_error** – passed to get_column_quantiles, set to False for only precise values, True to allow approximate values on systems with only binary choice (e.g. Redshift), and to a value between zero and one for systems that allow specification of relative error (e.g. SparkDFDataset).

Returns A list of bins

abstract get_column_hist (*self*, *column*, *bins*)

Get a histogram of column values :param column: the column for which to generate the histogram :param bins: the bins to slice the histogram. bins *must* be a tuple to ensure caching is possible :type bins: tuple

Returns: List[int], a list of counts corresponding to bins

abstract get_column_count_in_range (*self*, *column*, *min_val=None*, *max_val=None*, *strict_min=False*, *strict_max=True*)

Returns: int

test_column_map_expectation_function (*self*, *function*, **args*, ***kwargs*)

Test a column map expectation function

Parameters

- **function** (*func*) – The function to be tested. (Must be a valid column_map_expectation function.)
- ***args** – Positional arguments to be passed the the function
- ****kwargs** – Keyword arguments to be passed the the function

Returns A JSON-serializable expectation result object.

Notes

This function is a thin layer to allow quick testing of new expectation functions, without having to define custom classes, etc. To use developed expectations from the command-line tool, you'll still need to define custom classes, etc.

Check out `custom_expectations_reference` for more information.

test_column_aggregate_expectation_function (*self*, *function*, **args*, ***kwargs*)

Test a column aggregate expectation function

Parameters

- **function** (*func*) – The function to be tested. (Must be a valid column_aggregate_expectation function.)
- ***args** – Positional arguments to be passed the the function
- ****kwargs** – Keyword arguments to be passed the the function

Returns A JSON-serializable expectation result object.

Notes

This function is a thin layer to allow quick testing of new expectation functions, without having to define custom classes, etc. To use developed expectations from the command-line tool, you'll still need to define custom classes, etc.

Check out `custom_expectations_reference` for more information.

expect_column_to_exist (*self*, *column*, *column_index=None*, *result_format=None*, *include_config=True*, *catch_exceptions=None*, *meta=None*)

Expect the specified column to exist.

`expect_column_to_exist` is a *expectation*, not a `column_map_expectation` or `column_aggregate_expectation`.

Parameters **column** (*str*) – The column name.

Other Parameters

- **column_index** (*int or None*) – If not None, checks the order of the columns. The expectation will fail if the column is not in location `column_index` (zero-indexed).
- **result_format** (*str or None*) – Which output mode to use: `BOOLEAN_ONLY`, `BASIC`, `COMPLETE`, or `SUMMARY`. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

expect_table_columns_to_match_ordered_list (*self, column_list, result_format=None, include_config=True, catch_exceptions=None, meta=None*)

Expect the columns to exactly match a specified list.

`expect_table_columns_to_match_ordered_list` is a [expectation](#), not a `column_map_expectation` or `column_aggregate_expectation`.

Parameters `column_list` (*list of str*) – The column names, in the correct order.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: `BOOLEAN_ONLY`, `BASIC`, `COMPLETE`, or `SUMMARY`. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

expect_table_column_count_to_be_between (*self, min_value=None, max_value=None, result_format=None, include_config=True, catch_exceptions=None, meta=None*)

Expect the number of columns to be between two values.

`expect_table_column_count_to_be_between` is a [expectation](#), not a `column_map_expectation` or `column_aggregate_expectation`.

Keyword Arguments

- **min_value** (*int or None*) – The minimum number of columns, inclusive.

- **max_value** (*int or None*) – The maximum number of columns, inclusive.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

Notes

- min_value and max_value are both inclusive.
- If min_value is None, then max_value is treated as an upper bound, and the number of acceptable columns has no minimum.
- If max_value is None, then min_value is treated as a lower bound, and the number of acceptable columns has no maximum.

See also:

`expect_table_column_count_to_equal`

```
expect_table_column_count_to_equal (self, value, result_format=None, include_config=True, catch_exceptions=None, meta=None)
```

Expect the number of columns to equal a value.

`expect_table_column_count_to_equal` is a [expectation](#), not a `column_map_expectation` or `column_aggregate_expectation`.

Parameters **value** (*int*) – The expected number of columns.

Other Parameters

- **result_format** (*string or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[expect_table_column_count_to_be_between](#)

expect_table_row_count_to_be_between (*self*, *min_value=None*, *max_value=None*,
result_format=None, *include_config=True*,
catch_exceptions=None, *meta=None*)

Expect the number of rows to be between two values.

[expect_table_row_count_to_be_between](#) is a [expectation](#), not a [column_map_expectation](#) or [column_aggregate_expectation](#).

Keyword Arguments

- **min_value** (*int or None*) – The minimum number of rows, inclusive.
- **max_value** (*int or None*) – The maximum number of rows, inclusive.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

Notes

- *min_value* and *max_value* are both inclusive.
- If *min_value* is None, then *max_value* is treated as an upper bound, and the number of acceptable rows has no minimum.
- If *max_value* is None, then *min_value* is treated as a lower bound, and the number of acceptable rows has no maximum.

See also:

[expect_table_row_count_to_equal](#)

expect_table_row_count_to_equal (*self*, *value*, *result_format=None*, *include_config=True*,
catch_exceptions=None, *meta=None*)

Expect the number of rows to equal a value.

`expect_table_row_count_to_equal` is a [expectation](#), not a `column_map_expectation` or `column_aggregate_expectation`.

Parameters `value` (*int*) – The expected number of rows.

Other Parameters

- **result_format** (*string or None*) – Which output mode to use: `BOOLEAN_ONLY`, `BASIC`, `COMPLETE`, or `SUMMARY`. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

`expect_table_row_count_to_be_between`

```
abstract expect_column_values_to_be_unique (self,          column,          mostly=None,
                                             result_format=None,          in-
                                             clude_config=True,
                                             catch_exceptions=None, meta=None)
```

Expect each column value to be unique.

This expectation detects duplicates. All duplicated values are counted as exceptions.

For example, `[1, 2, 3, 3, 3]` will return `[3, 3, 3]` in `result.exceptions_list`, with `unexpected_percent = 60.0`.

`expect_column_values_to_be_unique` is a [column_map_expectation](#).

Parameters `column` (*str*) – The column name.

Keyword Arguments `mostly` (*None or a float between 0 and 1*) – Return “success”: `True` if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: `BOOLEAN_ONLY`, `BASIC`, `COMPLETE`, or `SUMMARY`. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to *result_format* and *include_config*, *catch_exceptions*, and *meta*.

```
abstract expect_column_values_to_not_be_null(self, column, mostly=None,
                                              result_format=None, include_config=True,
                                              catch_exceptions=None, meta=None)
```

Expect column values to not be null.

To be counted as an exception, values must be explicitly null or missing, such as a NULL in PostgreSQL or an np.NaN in pandas. Empty strings don't count as null unless they have been coerced to a null type.

`expect_column_values_to_not_be_null` is a *column_map_expectation*.

Parameters `column` (*str*) – The column name.

Keyword Arguments `mostly` (*None or a float between 0 and 1*) – Return “success”: *True* if at least mostly fraction of values match the expectation. For more detail, see *mostly*.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see *result_format*.
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see *include_config*.
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see *catch_exceptions*.
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see *meta*.

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to *result_format* and *include_config*, *catch_exceptions*, and *meta*.

See also:

expect_column_values_to_be_null

```
abstract expect_column_values_to_be_null(self, column, mostly=None, result_format=None,
                                          include_config=True, catch_exceptions=None, meta=None)
```

Expect column values to be null.

`expect_column_values_to_be_null` is a *column_map_expectation*.

Parameters `column` (*str*) – The column name.

Keyword Arguments `mostly` (*None or a float between 0 and 1*) – Return “success”: *True* if at least mostly fraction of values match the expectation. For more detail, see *mostly*.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see *result_format*.

- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[expect_column_values_to_not_be_null](#)

```
abstract expect_column_values_to_be_of_type (self, column, type_,
                                             mostly=None, result_format=None,
                                             include_config=True,
                                             catch_exceptions=None, meta=None)
```

Expect a column to contain values of a specified data type.

`expect_column_values_to_be_of_type` is a [column_aggregate_expectation](#) for typed-column backends, and also for PandasDataset where the column dtype and provided **type_** are unambiguous constraints (any dtype except 'object' or dtype of 'object' with **type_** specified as 'object').

For PandasDataset columns with dtype of 'object' `expect_column_values_to_be_of_type` is a [column_map_expectation](#) and will independently check each row's type.

Parameters

- **column** (*str*) – The column name.
- **type_** (*str*) – A string representing the data type that each column should have as entries. Valid types are defined by the current backend implementation and are dynamically loaded. For example, valid types for PandasDataset include any numpy dtype values (such as 'int64') or native python types (such as 'int'), whereas valid types for SQLAlchemy-Dataset include types named by the current driver such as 'INTEGER' in most SQL dialects and 'TEXT' in dialects such as postgresql. Valid types for SparkDFDataset include 'StringType', 'BooleanType' and other pyspark-defined type names.

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return “success”: True if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to *result_format* and *include_config*, *catch_exceptions*, and *meta*.

See also:

expect_column_values_to_be_in_type_list

```
abstract expect_column_values_to_be_in_type_list(self, column, type_list,
                                                mostly=None, re-
                                                sult_format=None, in-
                                                clude_config=True,
                                                catch_exceptions=None,
                                                meta=None)
```

Expect a column to contain values from a specified type list.

expect_column_values_to_be_in_type_list is a *column_aggregate_expectation* for typed-column backends, and also for *PandasDataset* where the column dtype provides an unambiguous constraints (any dtype except 'object'). For *PandasDataset* columns with dtype of 'object' *expect_column_values_to_be_of_type* is a *column_map_expectation* and will independently check each row's type.

Parameters

- **column** (*str*) – The column name.
- **type_list** (*str*) – A list of strings representing the data type that each column should have as entries. Valid types are defined by the current backend implementation and are dynamically loaded. For example, valid types for *PandasDataset* include any numpy dtype values (such as 'int64') or native python types (such as 'int'), whereas valid types for a *SqlAlchemyDataset* include types named by the current driver such as 'INTEGER' in most SQL dialects and 'TEXT' in dialects such as postgresql. Valid types for *SparkDFDataset* include 'StringType', 'BooleanType' and other pyspark-defined type names.

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return “success”: *True* if at least mostly fraction of values match the expectation. For more detail, see *mostly*.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see *result_format*.
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see *include_config*.
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see *catch_exceptions*.
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see *meta*.

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to *result_format* and *include_config*, *catch_exceptions*, and *meta*.

See also:

expect_column_values_to_be_of_type

```
abstract expect_column_values_to_be_in_set (self, column, value_set, mostly=None,
                                             parse_strings_as_datetimes=None,
                                             result_format=None,           in-
                                             include_config=True,
                                             catch_exceptions=None, meta=None)
```

Expect each column value to be in a given set.

For example:

```
# my_df.my_col = [1,2,2,3,3,3]
>>> my_df.expect_column_values_to_be_in_set(
    "my_col",
    [2,3]
)
{
  "success": false
  "result": {
    "unexpected_count": 1
    "unexpected_percent": 16.666666666666666,
    "unexpected_percent_nonmissing": 16.666666666666666,
    "partial_unexpected_list": [
      1
    ],
  },
}
```

`expect_column_values_to_be_in_set` is a *column_map_expectation*.

Parameters

- **column** (*str*) – The column name.
- **value_set** (*set-like*) – A set of objects used for comparison.

Keyword Arguments

- **mostly** (*None or a float between 0 and 1*) – Return “*success*”: *True* if at least mostly fraction of values match the expectation. For more detail, see *mostly*.
- **parse_strings_as_datetimes** (*boolean or None*) – If *True* values provided in *value_set* will be parsed as datetimes before making comparisons.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see *result_format*.
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see *include_config*.
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see *catch_exceptions*.
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see *meta*.

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to *result_format* and *include_config*, *catch_exceptions*, and *meta*.

See also:

`expect_column_values_to_not_be_in_set`

```
abstract expect_column_values_to_not_be_in_set (self, column, value_set,
                                                mostly=None,
                                                parse_strings_as_datetimes=None,
                                                result_format=None,
                                                include_config=True,
                                                catch_exceptions=None,
                                                meta=None)
```

Expect column entries to not be in the set.

For example:

```
# my_df.my_col = [1,2,2,3,3,3]
>>> my_df.expect_column_values_to_not_be_in_set(
    "my_col",
    [1,2]
)
{
  "success": false
  "result": {
    "unexpected_count": 3
    "unexpected_percent": 50.0,
    "unexpected_percent_nonmissing": 50.0,
    "partial_unexpected_list": [
      1, 2, 2
    ],
  },
}
```

`expect_column_values_to_not_be_in_set` is a `column_map_expectation`.

Parameters

- **column** (*str*) – The column name.
- **value_set** (*set-like*) – A set of objects used for comparison.

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return “*success*”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to *result_format* and *include_config*, *catch_exceptions*, and *meta*.

See also:

expect_column_values_to_be_in_set

```
abstract expect_column_values_to_be_between (self, column, min_value=None,
                                              max_value=None, strict_min=False,
                                              strict_max=False, allow_cross_type_comparisons=None,
                                              parse_strings_as_datetimes=False,
                                              output_strftime_format=None,
                                              mostly=None, result_format=None,
                                              include_config=True,
                                              catch_exceptions=None, meta=None)
```

Expect column entries to be between a minimum value and a maximum value (inclusive).

expect_column_values_to_be_between is a *column_map_expectation*.

Parameters

- **column** (*str*) – The column name.
- **min_value** (*comparable type or None*) – The minimum value for a column entry.
- **max_value** (*comparable type or None*) – The maximum value for a column entry.

Keyword Arguments

- **strict_min** (*boolean*) – If True, values must be strictly larger than *min_value*, default=False
- **strict_max** (*boolean*) – If True, values must be strictly smaller than *max_value*, default=False
- **allow_cross_type_comparisons** (*boolean or None*) : If True, allow comparisons between types (e.g. integer and string). Otherwise, attempting such comparisons will raise an exception.
- **parse_strings_as_datetimes** (*boolean or None*) – If True, parse *min_value*, *max_value*, and all non-null column values to datetimes before making comparisons.
- **output_strftime_format** (*str or None*) – A valid strftime format for datetime output. Only used if *parse_strings_as_datetimes*=True.
- **mostly** (*None or a float between 0 and 1*) – Return “success”: True if at least mostly fraction of values match the expectation. For more detail, see *mostly*.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see *result_format*.
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see *include_config*.
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see *catch_exceptions*.
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see *meta*.

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to *result_format* and *include_config*, *catch_exceptions*, and *meta*.

Notes

- *min_value* and *max_value* are both inclusive unless *strict_min* or *strict_max* are set to *True*.
- If *min_value* is *None*, then *max_value* is treated as an upper bound, and there is no minimum value checked.
- If *max_value* is *None*, then *min_value* is treated as a lower bound, and there is no maximum value checked.

See also:

expect_column_value_lengths_to_be_between

```
abstract expect_column_values_to_be_increasing (self, column, strictly=None,  
                                                parse_strings_as_datetimes=False,  
                                                mostly=None, re-  
                                                sult_format=None, in-  
                                                clude_config=True,  
                                                catch_exceptions=None,  
                                                meta=None)
```

Expect column values to be increasing.

By default, this expectation only works for numeric or datetime data. When *parse_strings_as_datetimes*=*True*, it can also parse strings to datetimes.

If *strictly*=*True*, then this expectation is only satisfied if each consecutive value is strictly increasing—equal values are treated as failures.

expect_column_values_to_be_increasing is a *column_map_expectation*.

Parameters *column* (*str*) – The column name.

Keyword Arguments

- **strictly** (*Boolean* or *None*) – If *True*, values must be strictly greater than previous values
- **parse_strings_as_datetimes** (*boolean* or *None*) – If *True*, all non-null column values to datetimes before making comparisons
- **mostly** (*None* or a *float* between 0 and 1) – Return “success”: *True* if at least mostly fraction of values match the expectation. For more detail, see *mostly*.

Other Parameters

- **result_format** (*str* or *None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see *result_format*.
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see *include_config*.
- **catch_exceptions** (*boolean* or *None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see *catch_exceptions*.

- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[expect_column_values_to_be_decreasing](#)

```
abstract expect_column_values_to_be_decreasing(self, column, strictly=None,
                                                parse_strings_as_datetimes=False,
                                                mostly=None,                re-
                                                sult_format=None,              in-
                                                include_config=True,
                                                catch_exceptions=None,
                                                meta=None)
```

Expect column values to be decreasing.

By default, this expectation only works for numeric or datetime data. When [parse_strings_as_datetimes=True](#), it can also parse strings to datetimes.

If [strictly=True](#), then this expectation is only satisfied if each consecutive value is strictly decreasing—equal values are treated as failures.

[expect_column_values_to_be_decreasing](#) is a [column_map_expectation](#).

Parameters [column](#) (*str*) – The column name.

Keyword Arguments

- **strictly** (*Boolean or None*) – If True, values must be strictly greater than previous values
- **parse_strings_as_datetimes** (*boolean or None*) – If True, all non-null column values to datetimes before making comparisons
- **mostly** (*None or a float between 0 and 1*) – Return “success”: True if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: [BOOLEAN_ONLY](#), [BASIC](#), [COMPLETE](#), or [SUMMARY](#). For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[`expect_column_values_to_be_increasing`](#)

```
abstract expect_column_value_lengths_to_be_between (self, column,
                                                    min_value=None,
                                                    max_value=None,
                                                    mostly=None,      re-
                                                    sult_format=None,    in-
                                                    clude_config=True,
                                                    catch_exceptions=None,
                                                    meta=None)
```

Expect column entries to be strings with length between a minimum value and a maximum value (inclusive).

This expectation only works for string-type values. Invoking it on ints or floats will raise a `TypeError`.

`expect_column_value_lengths_to_be_between` is a [`column_map_expectation`](#).

Parameters `column` (*str*) – The column name.

Keyword Arguments

- **min_value** (*int or None*) – The minimum value for a column entry length.
- **max_value** (*int or None*) – The maximum value for a column entry length.
- **mostly** (*None or a float between 0 and 1*) – Return “success”: *True* if at least mostly fraction of values match the expectation. For more detail, see [`mostly`](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [`result_format`](#).
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see [`include_config`](#).
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see [`catch_exceptions`](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [`meta`](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [`result_format`](#) and [`include_config`](#), [`catch_exceptions`](#), and [`meta`](#).

Notes

- `min_value` and `max_value` are both inclusive.
- If `min_value` is *None*, then `max_value` is treated as an upper bound, and the number of acceptable rows has no minimum.
- If `max_value` is *None*, then `min_value` is treated as a lower bound, and the number of acceptable rows has no maximum.

See also:

expect_column_value_lengths_to_equal

```
abstract expect_column_value_lengths_to_equal (self, column, value,
                                                mostly=None, result_format=None,
                                                include_config=True,
                                                catch_exceptions=None,
                                                meta=None)
```

Expect column entries to be strings with length equal to the provided value.

This expectation only works for string-type values. Invoking it on ints or floats will raise a `TypeError`.

`expect_column_values_to_be_between` is a *column_map_expectation*.

Parameters

- **column** (*str*) – The column name.
- **value** (*int or None*) – The expected value for a column entry length.

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return “*success*”: *True* if at least mostly fraction of values match the expectation. For more detail, see *mostly*.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see *result_format*.
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see *include_config*.
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see *catch_exceptions*.
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see *meta*.

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to *result_format* and *include_config*, *catch_exceptions*, and *meta*.

See also:

expect_column_value_lengths_to_be_between

```
abstract expect_column_values_to_match_regex (self, column, regex,
                                                mostly=None, result_format=None,
                                                include_config=True,
                                                catch_exceptions=None,
                                                meta=None)
```

Expect column entries to be strings that match a given regular expression. Valid matches can be found anywhere in the string, for example “[at]+” will identify the following strings as expected: “cat”, “hat”, “aa”, “a”, and “t”, and the following strings as unexpected: “fish”, “dog”.

`expect_column_values_to_match_regex` is a *column_map_expectation*.

Parameters

- **column** (*str*) – The column name.
- **regex** (*str*) – The regular expression the column entries should match.

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return “*success*”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[expect_column_values_to_not_match_regex](#)

[expect_column_values_to_match_regex_list](#)

```
abstract expect_column_values_to_not_match_regex(self, column, regex,
                                                  mostly=None, re-
                                                  sult_format=None, in-
                                                  clude_config=True,
                                                  catch_exceptions=None,
                                                  meta=None)
```

Expect column entries to be strings that do NOT match a given regular expression. The regex must not match any portion of the provided string. For example, “[at]+” would identify the following strings as expected: “fish”, “dog”, and the following as unexpected: “cat”, “hat”.

`expect_column_values_to_not_match_regex` is a [column_map_expectation](#).

Parameters

- **column** (*str*) – The column name.
- **regex** (*str*) – The regular expression the column entries should NOT match.

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return “*success*”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).

- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[expect_column_values_to_match_regex](#)

[expect_column_values_to_match_regex_list](#)

```
abstract expect_column_values_to_match_regex_list(self, column, regex_list,
                                                  match_on='any',
                                                  mostly=None,           re-
                                                  result_format=None,       in-
                                                  include_config=True,
                                                  catch_exceptions=None,
                                                  meta=None)
```

Expect the column entries to be strings that can be matched to either any of or all of a list of regular expressions. Matches can be anywhere in the string.

`expect_column_values_to_match_regex_list` is a [column_map_expectation](#).

Parameters

- **column** (*str*) – The column name.
- **regex_list** (*list*) – The list of regular expressions which the column entries should match

Keyword Arguments

- **match_on=** (*string*) – “any” or “all”. Use “any” if the value should match at least one regular expression in the list. Use “all” if it should match each regular expression in the list.
- **mostly** (*None or a float between 0 and 1*) – Return “success”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[`expect_column_values_to_match_regex`](#)

[`expect_column_values_to_not_match_regex`](#)

```
abstract expect_column_values_to_not_match_regex_list(self, column, regex_list,
                                                    mostly=None, re-
                                                    sult_format=None,
                                                    include_config=True,
                                                    catch_exceptions=None,
                                                    meta=None)
```

Expect the column entries to be strings that do not match any of a list of regular expressions. Matches can be anywhere in the string.

`expect_column_values_to_not_match_regex_list` is a [`column_map_expectation`](#).

Parameters

- **column** (*str*) – The column name.
- **regex_list** (*list*) – The list of regular expressions which the column entries should not match

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return “*success*”: *True* if at least mostly fraction of values match the expectation. For more detail, see [`mostly`](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: `BOOLEAN_ONLY`, `BASIC`, `COMPLETE`, or `SUMMARY`. For more detail, see [`result_format`](#).
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see [`include_config`](#).
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see [`catch_exceptions`](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [`meta`](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [`result_format`](#) and [`include_config`](#), [`catch_exceptions`](#), and [`meta`](#).

See also:

[`expect_column_values_to_match_regex_list`](#)

```
abstract expect_column_values_to_match_strftime_format(self, column,
                                                    strftime_format,
                                                    mostly=None, re-
                                                    sult_format=None,
                                                    include_config=True,
                                                    catch_exceptions=None,
                                                    meta=None)
```

Expect column entries to be strings representing a date or time with a given format.

`expect_column_values_to_match_strftime_format` is a [`column_map_expectation`](#).

Parameters

- **column** (*str*) – The column name.
- **strftime_format** (*str*) – A strftime format string to use for matching

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return “*success*”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

```
abstract expect_column_values_to_be_dateutil_parseable (self, column,
                                                         mostly=None, re-
                                                         sult_format=None,
                                                         include_config=True,
                                                         catch_exceptions=None,
                                                         meta=None)
```

Expect column entries to be parsable using dateutil.

`expect_column_values_to_be_dateutil_parseable` is a [column_map_expectation](#).

Parameters **column** (*str*) – The column name.

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return “*success*”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

```
abstract expect_column_values_to_be_json_parseable(self, column, mostly=None,
                                                    result_format=None,
                                                    include_config=True,
                                                    catch_exceptions=None,
                                                    meta=None)
```

Expect column entries to be data written in JavaScript Object Notation.

`expect_column_values_to_be_json_parseable` is a [column_map_expectation](#).

Parameters `column` (*str*) – The column name.

Keyword Arguments `mostly` (*None or a float between 0 and 1*) – Return “*success*”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[expect_column_values_to_match_json_schema](#)

```
abstract expect_column_values_to_match_json_schema(self, column, json_schema,
                                                    mostly=None,          re-
                                                    sult_format=None,      in-
                                                    clude_config=True,
                                                    catch_exceptions=None,
                                                    meta=None)
```

Expect column entries to be JSON objects matching a given JSON schema.

`expect_column_values_to_match_json_schema` is a [column_map_expectation](#).

Parameters `column` (*str*) – The column name.

Keyword Arguments `mostly` (*None or a float between 0 and 1*) – Return “*success*”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see [include_config](#).

- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[expect_column_values_to_be_json_parseable](#)

The [JSON-schema docs](#).

abstract expect_column_parameterized_distribution_ks_test_p_value_to_be_greater_than (se

Expect the column values to be distributed similarly to a scipy distribution. This expectation compares the provided column to the specified continuous distribution with a parametric Kolmogorov-Smirnov test. The K-S test compares the provided column to the cumulative density function (CDF) of the specified scipy distribution. If you don't know the desired distribution shape parameters, use the `ge.dataset.util.infer_distribution_parameters()` utility function to estimate them.

It returns 'success'=True if the p-value from the K-S test is greater than or equal to the provided p-value.

`expect_column_parameterized_distribution_ks_test_p_value_to_be_greater_than` is a [column_aggregate_expectation](#).

Parameters

- **column** (*str*) – The column name.
- **distribution** (*str*) – The scipy distribution name. See: <https://docs.scipy.org/doc/scipy/reference/stats.html> Currently supported distributions are listed in the Notes section below.
- **p_value** (*float*) – The threshold p-value for a passing test. Default is 0.05.
- **params** (*dict or list*) – A dictionary or positional list of shape parameters that describe the distribution you want to test the data against. Include key values specific to the distribution from the appropriate scipy distribution CDF function. 'loc' and 'scale' are used as translational parameters. See <https://docs.scipy.org/doc/scipy/reference/stats.html#continuous-distributions>

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

Notes

These fields in the result object are customized for this expectation:

```
{
  "details": {
    "expected_params" (dict): The specified or inferred parameters of the
    ↪ distribution to test against
    "ks_results" (dict): The raw result of stats.kstest()
  }
}
```

- The Kolmogorov-Smirnov test's null hypothesis is that the column is similar to the provided distribution.
- Supported scipy distributions:
 - norm
 - beta
 - gamma
 - uniform
 - chi2
 - expon

```
expect_column_distinct_values_to_be_in_set (self, column, value_set,
                                              parse_strings_as_datetimes=None,
                                              result_format=None, include_config=True,
                                              catch_exceptions=None, meta=None)
```

Expect the set of distinct column values to be contained by a given set.

The success value for this expectation will match that of `expect_column_values_to_be_in_set`. However, `expect_column_distinct_values_to_be_in_set` is a *column_aggregate_expectation*.

For example:


```
# my_df.my_col = [1,2,2,3,3,3]
>>> my_df.expect_column_distinct_values_to_be_in_set(
    "my_col",
    [2, 3, 4]
)
{
  "success": false
  "result": {
    "observed_value": [1,2,3],
    "details": {
      "value_counts": [
        {
          "value": 1,
          "count": 1
        },
        {
          "value": 2,
          "count": 1
        },
        {
          "value": 3,
          "count": 1
        }
      ]
    }
  }
}
```

Parameters

- **column** (*str*) – The column name.
- **value_set** (*set-like*) – A set of objects used for comparison.

Keyword Arguments **parse_strings_as_datetimes** (*boolean or None*) – If True values provided in value_set will be parsed as datetimes before making comparisons.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[expect_column_distinct_values_to_contain_set](#)

```
expect_column_distinct_values_to_equal_set (self, column, value_set,
                                             parse_strings_as_datetimes=None,
                                             result_format=None, include_config=True,
                                             catch_exceptions=None, meta=None)
```

Expect the set of distinct column values to equal a given set.

In contrast to `expect_column_distinct_values_to_contain_set()` this ensures not only that a certain set of values are present in the column but that these *and only these* values are present.

`expect_column_distinct_values_to_equal_set` is a *column_aggregate_expectation*.

For example:

```
# my_df.my_col = [1,2,2,3,3,3]
>>> my_df.expect_column_distinct_values_to_equal_set(
    "my_col",
    [2,3]
)
{
  "success": false
  "result": {
    "observed_value": [1,2,3]
  },
}
```

Parameters

- **column** (*str*) – The column name.
- **value_set** (*set-like*) – A set of objects used for comparison.

Keyword Arguments `parse_strings_as_datetimes` (*boolean or None*) – If True values provided in `value_set` will be parsed as datetimes before making comparisons.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see *result_format*.
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see *include_config*.
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see *catch_exceptions*.
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see *meta*.

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to *result_format* and *include_config*, *catch_exceptions*, and *meta*.

See also:

expect_column_distinct_values_to_contain_set

```
expect_column_distinct_values_to_contain_set (self, column, value_set,
                                              parse_strings_as_datetimes=None,
                                              result_format=None, include_config=True,
                                              catch_exceptions=None, meta=None)
```

Expect the set of distinct column values to contain a given set.

In contrast to `expect_column_values_to_be_in_set()` this ensures not that all column values are members of the given set but that values from the set *must* be present in the column.

`expect_column_distinct_values_to_contain_set` is a *column_aggregate_expectation*.

For example:

```
# my_df.my_col = [1,2,2,3,3,3]
>>> my_df.expect_column_distinct_values_to_contain_set(
    "my_col",
    [2,3]
)
{
  "success": true
  "result": {
    "observed_value": [1,2,3]
  },
}
```

Parameters

- **column** (*str*) – The column name.
- **value_set** (*set-like*) – A set of objects used for comparison.

Keyword Arguments `parse_strings_as_datetimes` (*boolean or None*) – If True values provided in `value_set` will be parsed as datetimes before making comparisons.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[expect_column_distinct_values_to_equal_set](#)

```
expect_column_mean_to_be_between(self, column, min_value=None, max_value=None,  
                                  strict_min=False, strict_max=False, re-  
                                  sult_format=None, include_config=True,  
                                  catch_exceptions=None, meta=None)
```

Expect the column mean to be between a minimum value and a maximum value (inclusive).

`expect_column_mean_to_be_between` is a [column_aggregate_expectation](#).

Parameters

- **column** (*str*) – The column name.
- **min_value** (*float or None*) – The minimum value for the column mean.
- **max_value** (*float or None*) – The maximum value for the column mean.
- **strict_min** (*boolean*) – If True, the column mean must be strictly larger than `min_value`, default=False
- **strict_max** (*boolean*) – If True, the column mean must be strictly smaller than `max_value`, default=False

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: `BOOLEAN_ONLY`, `BASIC`, `COMPLETE`, or `SUMMARY`. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

Notes

These fields in the result object are customized for this expectation:

```
{  
    "observed_value": (float) The true mean for the column  
}
```

- `min_value` and `max_value` are both inclusive unless `strict_min` or `strict_max` are set to True.
- If `min_value` is None, then `max_value` is treated as an upper bound.
- If `max_value` is None, then `min_value` is treated as a lower bound.

See also:

[expect_column_median_to_be_between](#)

[expect_column_stddev_to_be_between](#)

```
expect_column_median_to_be_between(self, column, min_value=None, max_value=None,
                                     strict_min=False, strict_max=False, re-
                                     sult_format=None, include_config=True,
                                     catch_exceptions=None, meta=None)
```

Expect the column median to be between a minimum value and a maximum value.

`expect_column_median_to_be_between` is a *column_aggregate_expectation*.

Parameters

- **column** (*str*) – The column name.
- **min_value** (*int or None*) – The minimum value for the column median.
- **max_value** (*int or None*) – The maximum value for the column median.
- **strict_min** (*boolean*) – If True, the column median must be strictly larger than `min_value`, default=False
- **strict_max** (*boolean*) – If True, the column median must be strictly smaller than `max_value`, default=False

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see *result_format*.
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see *include_config*.
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see *catch_exceptions*.
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see *meta*.

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to *result_format* and *include_config*, *catch_exceptions*, and *meta*.

Notes

These fields in the result object are customized for this expectation:

```
{
  "observed_value": (float) The true median for the column
}
```

- `min_value` and `max_value` are both inclusive unless `strict_min` or `strict_max` are set to True.
- If `min_value` is None, then `max_value` is treated as an upper bound
- If `max_value` is None, then `min_value` is treated as a lower bound

See also:

expect_column_mean_to_be_between

expect_column_stddev_to_be_between

```
expect_column_quantile_values_to_be_between(self, column, quantile_ranges,  
                                              allow_relative_error=False,  
                                              result_format=None, include_config=True,  
                                              catch_exceptions=None, meta=None)
```

Expect specific provided column quantiles to be between provided minimum and maximum values.

`quantile_ranges` must be a dictionary with two keys:

- `quantiles`: (list of float) increasing ordered list of desired quantile values
- `value_ranges`: (list of lists): Each element in this list consists of a list with two values, a lower and upper bound (inclusive) for the corresponding quantile.

For each provided range:

- `min_value` and `max_value` are both inclusive.
- If `min_value` is `None`, then `max_value` is treated as an upper bound only
- If `max_value` is `None`, then `min_value` is treated as a lower bound only

The length of the quantiles list and `quantile_values` list must be equal.

For example:

```
# my_df.my_col = [1,2,2,3,3,3,4]
>>> my_df.expect_column_quantile_values_to_be_between(
    "my_col",
    {
        "quantiles": [0., 0.333, 0.6667, 1.],
        "value_ranges": [[0,1], [2,3], [3,4], [4,5]]
    }
)
{
  "success": True,
  "result": {
    "observed_value": {
      "quantiles": [0., 0.333, 0.6667, 1.],
      "values": [1, 2, 3, 4],
    }
  },
  "element_count": 7,
  "missing_count": 0,
  "missing_percent": 0.0,
  "details": {
    "success_details": [true, true, true, true]
  }
}
```

`expect_column_quantile_values_to_be_between` can be computationally intensive for large datasets.

`expect_column_quantile_values_to_be_between` is a `column_aggregate_expectation`.

Parameters

- **`column`** (*str*) – The column name.
- **`quantile_ranges`** (*dictionary*) – Quantiles and associated value ranges for the column. See above for details.

- **allow_relative_error** (*boolean*) – Whether to allow relative error in quantile communications on backends that support or require it.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

Notes

These fields in the result object are customized for this expectation: `:: details.success_details`

See also:

[expect_column_min_to_be_between](#)

[expect_column_max_to_be_between](#)

[expect_column_median_to_be_between](#)

expect_column_stdev_to_be_between(*self*, *column*, *min_value=None*, *max_value=None*, *strict_min=False*, *strict_max=False*, *result_format=None*, *include_config=True*, *catch_exceptions=None*, *meta=None*)

Expect the column standard deviation to be between a minimum value and a maximum value. Uses sample standard deviation (normalized by N-1).

`expect_column_stdev_to_be_between` is a `column_aggregate_expectation`.

Parameters

- **column** (*str*) – The column name.
- **min_value** (*float or None*) – The minimum value for the column standard deviation.
- **max_value** (*float or None*) – The maximum value for the column standard deviation.
- **strict_min** (*boolean*) – If True, the column standard deviation must be strictly larger than `min_value`, default=False
- **strict_max** (*boolean*) – If True, the column standard deviation must be strictly smaller than `max_value`, default=False

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

Notes

These fields in the result object are customized for this expectation:

```
{
    "observed_value": (float) The true standard deviation for the column
}
```

- `min_value` and `max_value` are both inclusive unless `strict_min` or `strict_max` are set to True.
- If `min_value` is None, then `max_value` is treated as an upper bound
- If `max_value` is None, then `min_value` is treated as a lower bound

See also:

[expect_column_mean_to_be_between](#)

[expect_column_median_to_be_between](#)

```
expect_column_unique_value_count_to_be_between(self, column, min_value=None,
                                                  max_value=None,           re-
                                                  result_format=None,       in-
                                                  include_config=True,
                                                  catch_exceptions=None,
                                                  meta=None)
```

Expect the number of unique values to be between a minimum value and a maximum value.

`expect_column_unique_value_count_to_be_between` is a `column_aggregate_expectation`.

Parameters

- **column** (*str*) – The column name.
- **min_value** (*int or None*) – The minimum number of unique values allowed.
- **max_value** (*int or None*) – The maximum number of unique values allowed.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).

- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

Notes

These fields in the result object are customized for this expectation:

```
{
    "observed_value": (int) The number of unique values in the column
}
```

- min_value and max_value are both inclusive.
- If min_value is None, then max_value is treated as an upper bound
- If max_value is None, then min_value is treated as a lower bound

See also:

[expect_column_proportion_of_unique_values_to_be_between](#)

```
expect_column_proportion_of_unique_values_to_be_between (self, column,
                                                           min_value=0,
                                                           max_value=1,
                                                           strict_min=False,
                                                           strict_max=False,
                                                           result_format=None,
                                                           include_config=True,
                                                           catch_exceptions=None,
                                                           meta=None)
```

Expect the proportion of unique values to be between a minimum value and a maximum value.

For example, in a column containing [1, 2, 2, 3, 3, 3, 4, 4, 4, 4], there are 4 unique values and 10 total values for a proportion of 0.4.

`expect_column_proportion_of_unique_values_to_be_between` is a `column_aggregate_expectation`.

Parameters

- **column** (*str*) – The column name.
- **min_value** (*float or None*) – The minimum proportion of unique values. (Proportions are on the range 0 to 1)
- **max_value** (*float or None*) – The maximum proportion of unique values. (Proportions are on the range 0 to 1)

- **strict_min** (*boolean*) – If True, the minimum proportion of unique values must be strictly larger than min_value, default=False
- **strict_max** (*boolean*) – If True, the maximum proportion of unique values must be strictly smaller than max_value, default=False

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

Notes

These fields in the result object are customized for this expectation:

```
{
    "observed_value": (float) The proportion of unique values in the column
}
```

- min_value and max_value are both inclusive unless strict_min or strict_max are set to True.
- If min_value is None, then max_value is treated as an upper bound
- If max_value is None, then min_value is treated as a lower bound

See also:

[expect_column_unique_value_count_to_be_between](#)

```
expect_column_most_common_value_to_be_in_set (self, column, value_set,
                                                ties_okay=None, re-
                                                sult_format=None, in-
                                                clude_config=True,
                                                catch_exceptions=None,
                                                meta=None)
```

Expect the most common value to be within the designated value set

expect_column_most_common_value_to_be_in_set is a `column_aggregate_expectation`.

Parameters

- **column** (*str*) – The column name
- **value_set** (*set-like*) – A list of potential values to match

Keyword Arguments `ties_okay` (*boolean or None*) – If True, then the expectation will still succeed if values outside the designated set are as common (but not more common) than designated values

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: `BOOLEAN_ONLY`, `BASIC`, `COMPLETE`, or `SUMMARY`. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

Notes

These fields in the result object are customized for this expectation:

```
{
  "observed_value": (list) The most common values in the column
}
```

`observed_value` contains a list of the most common values. Often, this will just be a single element. But if there's a tie for most common among multiple values, `observed_value` will contain a single copy of each most common value.

expect_column_sum_to_be_between (*self, column, min_value=None, max_value=None, strict_min=False, strict_max=False, result_format=None, include_config=True, catch_exceptions=None, meta=None*)

Expect the column to sum to be between an min and max value

`expect_column_sum_to_be_between` is a `column_aggregate_expectation`.

Parameters

- **column** (*str*) – The column name
- **min_value** (*comparable type or None*) – The minimal sum allowed.
- **max_value** (*comparable type or None*) – The maximal sum allowed.
- **strict_min** (*boolean*) – If True, the minimal sum must be strictly larger than `min_value`, default=False
- **strict_max** (*boolean*) – If True, the maximal sum must be strictly smaller than `max_value`, default=False

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: `BOOLEAN_ONLY`, `BASIC`, `COMPLETE`, or `SUMMARY`. For more detail, see [result_format](#).

- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

Notes

These fields in the result object are customized for this expectation:

```
{  
    "observed_value": (list) The actual column sum  
}
```

- `min_value` and `max_value` are both inclusive unless `strict_min` or `strict_max` are set to True.
- If `min_value` is None, then `max_value` is treated as an upper bound
- If `max_value` is None, then `min_value` is treated as a lower bound

```
expect_column_min_to_be_between (self, column, min_value=None, max_value=None,  
                                  strict_min=False, strict_max=False,  
                                  parse_strings_as_datetimes=False, out-  
                                  put_strftime_format=None, result_format=None,  
                                  include_config=True, catch_exceptions=None,  
                                  meta=None)
```

Expect the column minimum to be between an min and max value

`expect_column_min_to_be_between` is a `column_aggregate_expectation`.

Parameters

- **column** (*str*) – The column name
- **min_value** (*comparable type or None*) – The minimal column minimum allowed.
- **max_value** (*comparable type or None*) – The maximal column minimum allowed.
- **strict_min** (*boolean*) – If True, the minimal column minimum must be strictly larger than `min_value`, default=False
- **strict_max** (*boolean*) – If True, the maximal column minimum must be strictly smaller than `max_value`, default=False

Keyword Arguments

- **parse_strings_as_datetimes** (*Boolean or None*) – If True, parse `min_value`, `max_value`s, and all non-null column values to datetimes before making comparisons.

- **output_strftime_format** (*str or None*) – A valid strftime format for datetime output. Only used if `parse_strings_as_datetimes=True`.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: `BOOLEAN_ONLY`, `BASIC`, `COMPLETE`, or `SUMMARY`. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

Notes

These fields in the result object are customized for this expectation:

```
{
    "observed_value": (list) The actual column min
}
```

- `min_value` and `max_value` are both inclusive unless `strict_min` or `strict_max` are set to True.
- If `min_value` is None, then `max_value` is treated as an upper bound
- If `max_value` is None, then `min_value` is treated as a lower bound

```
expect_column_max_to_be_between(self, column, min_value=None, max_value=None,
                                strict_min=False, strict_max=False,
                                parse_strings_as_datetimes=False, output_strftime_format=None,
                                result_format=None, include_config=True, catch_exceptions=None,
                                meta=None)
```

Expect the column max to be between an min and max value

`expect_column_max_to_be_between` is a `column_aggregate_expectation`.

Parameters

- **column** (*str*) – The column name
- **min_value** (*comparable type or None*) – The minimum number of unique values allowed.
- **max_value** (*comparable type or None*) – The maximum number of unique values allowed.

Keyword Arguments

- **parse_strings_as_datetimes** (*Boolean or None*) – If True, parse `min_value`, `max_value`s, and all non-null column values to datetimes before making comparisons.
- **output_strftime_format** (*str or None*) – A valid strftime format for datetime output. Only used if `parse_strings_as_datetimes=True`.
- **strict_min** (*boolean*) – If True, the minimal column minimum must be strictly larger than `min_value`, default=False
- **strict_max** (*boolean*) – If True, the maximal column minimum must be strictly smaller than `max_value`, default=False

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: `BOOLEAN_ONLY`, `BASIC`, `COMPLETE`, or `SUMMARY`. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

Notes

These fields in the result object are customized for this expectation:

```
{
  "observed_value": (list) The actual column max
}
```

- `min_value` and `max_value` are both inclusive unless `strict_min` or `strict_max` are set to True.
- If `min_value` is None, then `max_value` is treated as an upper bound
- If `max_value` is None, then `min_value` is treated as a lower bound

expect_column_chisquare_test_p_value_to_be_greater_than(*self*, *column*, *partition_object=None*, *p=0.05*, *tail_weight_holdout=0*, *result_format=None*, *include_config=True*, *catch_exceptions=None*, *meta=None*)

Expect column values to be distributed similarly to the provided categorical partition. This expectation compares categorical distributions using a Chi-squared test. It returns `success=True` if values in the column match the distribution of the provided partition.

`expect_column_chisquare_test_p_value_to_be_greater_than` is a `column_aggregate_expectation`.

Parameters

- **column** (*str*) – The column name.
- **partition_object** (*dict*) – The expected partition object (see [Partition Objects](#)).
- **p** (*float*) – The p-value threshold for rejecting the null hypothesis of the Chi-Squared test. For values below the specified threshold, the expectation will return `success=False`, rejecting the null hypothesis that the distributions are the same. Defaults to 0.05.

Keyword Arguments **tail_weight_holdout** (*float between 0 and 1 or None*) – The amount of weight to split uniformly between values observed in the data but not present in the provided partition. `tail_weight_holdout` provides a mechanism to make the test less strict by assigning positive weights to unknown values observed in the data that are not present in the partition.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: `BOOLEAN_ONLY`, `BASIC`, `COMPLETE`, or `SUMMARY`. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

Notes

These fields in the result object are customized for this expectation:

```
{
  "observed_value": (float) The true p-value of the Chi-squared test
  "details": {
    "observed_partition" (dict):
      The partition observed in the data.
    "expected_partition" (dict):
      The partition expected from the data, after including tail_weight_
↪holdout
  }
}
```

```
abstract expect_column_bootstrapped_ks_test_p_value_to_be_greater_than(self,
                                                                    column,
                                                                    partition_object=None,
                                                                    p=0.05,
                                                                    bootstrap_samples=None,
                                                                    bootstrap_sample_size=None,
                                                                    result_format=None,
                                                                    include_config=True,
                                                                    catch_exceptions=None,
                                                                    meta=None)
```

Expect column values to be distributed similarly to the provided continuous partition. This expectation compares continuous distributions using a bootstrapped Kolmogorov-Smirnov test. It returns *success=True* if values in the column match the distribution of the provided partition.

The expected cumulative density function (CDF) is constructed as a linear interpolation between the bins, using the provided weights. Consequently the test expects a piecewise uniform distribution using the bins from the provided partition object.

`expect_column_bootstrapped_ks_test_p_value_to_be_greater_than` is a `column_aggregate_expectation`.

Parameters

- **column** (*str*) – The column name.
- **partition_object** (*dict*) – The expected partition object (see [Partition Objects](#)).
- **p** (*float*) – The p-value threshold for the Kolmogorov-Smirnov test. For values below the specified threshold the expectation will return *success=False*, rejecting the null hypothesis that the distributions are the same. Defaults to 0.05.

Keyword Arguments

- **bootstrap_samples** (*int*) – The number bootstrap rounds. Defaults to 1000.
- **bootstrap_sample_size** (*int*) – The number of samples to take from the column for each bootstrap. A larger sample will increase the specificity of the test. Defaults to `2 * len(partition_object['weights'])`

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to *result_format* and *include_config*, *catch_exceptions*, and *meta*.

Notes

These fields in the result object are customized for this expectation:

```
{
  "observed_value": (float) The true p-value of the KS test
  "details": {
    "bootstrap_samples": The number of bootstrap rounds used
    "bootstrap_sample_size": The number of samples taken from
      the column in each bootstrap round
    "observed_cdf": The cumulative density function observed
      in the data, a dict containing 'x' values and cdf_values
      (suitable for plotting)
    "expected_cdf" (dict):
      The cumulative density function expected based on the
      partition object, a dict containing 'x' values and
      cdf_values (suitable for plotting)
    "observed_partition" (dict):
      The partition observed on the data, using the provided
      bins but also expanding from min(column) to max(column)
    "expected_partition" (dict):
      The partition expected from the data. For KS test,
      this will always be the partition_object parameter
  }
}
```

```
expect_column_kl_divergence_to_be_less_than(self, column, partition_object=None, threshold=None,
                                             tail_weight_holdout=0, internal_weight_holdout=0, bucketize_data=True,
                                             result_format=None, include_config=True, catch_exceptions=None, meta=None)
```

Expect the Kulback-Leibler (KL) divergence (relative entropy) of the specified column with respect to the partition object to be lower than the provided threshold.

KL divergence compares two distributions. The higher the divergence value (relative entropy), the larger the difference between the two distributions. A relative entropy of zero indicates that the data are distributed identically, *when binned according to the provided partition*.

In many practical contexts, choosing a value between 0.5 and 1 will provide a useful test.

This expectation works on both categorical and continuous partitions. See notes below for details.

`expect_column_kl_divergence_to_be_less_than` is a `column_aggregate_expectation`.

Parameters

- **column** (*str*) – The column name.
- **partition_object** (*dict*) – The expected partition object (see *Partition Objects*).
- **threshold** (*float*) – The maximum KL divergence to for which to return *success=True*. If KL divergence is larger than the provided threshold, the test will return *success=False*.

Keyword Arguments

- **internal_weight_holdout** (*float between 0 and 1 or None*) – The amount of weight to split uniformly among zero-weighted partition bins. `internal_weight_holdout` provides a mechanisms to make the test less strict by assigning positive weights to values observed in the data for which the partition explicitly expected zero weight. With no `internal_weight_holdout`, any value observed in such a region will cause KL divergence to rise to +Infinity. Defaults to 0.
- **tail_weight_holdout** (*float between 0 and 1 or None*) – The amount of weight to add to the tails of the histogram. Tail weight holdout is split evenly between $(-\text{Infinity}, \min(\text{partition_object}[\text{'bins'}]))$ and $(\max(\text{partition_object}[\text{'bins'}]), +\text{Infinity})$. `tail_weight_holdout` provides a mechanism to make the test less strict by assigning positive weights to values observed in the data that are not present in the partition. With no `tail_weight_holdout`, any value observed outside the provided `partition_object` will cause KL divergence to rise to +Infinity. Defaults to 0.
- **bucketize_data** (*boolean*) – If True, then continuous data will be bucketized before evaluation. Setting this parameter to false allows evaluation of KL divergence with a None partition object for profiling against discrete data.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: `BOOLEAN_ONLY`, `BASIC`, `COMPLETE`, or `SUMMARY`. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

Notes

These fields in the result object are customized for this expectation:

```
{
  "observed_value": (float) The true KL divergence (relative entropy) or None,
  ↳ if the value is calculated as infinity, -infinity, or NaN
  "details": {
    "observed_partition": (dict) The partition observed in the data
    "expected_partition": (dict) The partition against which the data were,
    ↳ compared,
                                after applying specified weight holdouts.
  }
}
```

If the `partition_object` is categorical, this expectation will expect the values in column to also be categorical.

- If the column includes values that are not present in the partition, the `tail_weight_holdout` will be equally split among those values, providing a mechanism to weaken the strictness of the expectation (otherwise, relative entropy would immediately go to infinity).
- If the partition includes values that are not present in the column, the test will simply include zero weight for that value.

If the `partition_object` is continuous, this expectation will discretize the values in the column according to the bins specified in the `partition_object`, and apply the test to the resulting distribution.

- The `internal_weight_holdout` and `tail_weight_holdout` parameters provide a mechanism to weaken the expectation, since an expected weight of zero would drive relative entropy to be infinite if any data are observed in that interval.
- If `internal_weight_holdout` is specified, that value will be distributed equally among any intervals with weight zero in the `partition_object`.
- If `tail_weight_holdout` is specified, that value will be appended to the tails of the bins ((-Infinity, min(bins)) and (max(bins), Infinity)).

If relative entropy/kl divergence goes to infinity for any of the reasons mentioned above, the observed value will be set to `None`. This is because `inf`, `-inf`, `Nan`, are not json serializable and cause some json parsers to crash when encountered. The python `None` token will be serialized to null in json.

See also:

`expect_column_chisquare_test_p_value_to_be_greater_than`

`expect_column_bootstrapped_ks_test_p_value_to_be_greater_than`

```
abstract expect_column_pair_values_to_be_equal(self, column_A, column_B, ignore_row_if='both_values_are_missing',
                                              result_format=None,
                                              include_config=True,
                                              catch_exceptions=None,
                                              meta=None)
```

Expect the values in column A to be the same as column B.

Parameters

- **column_A** (*str*) – The first column name
- **column_B** (*str*) – The second column name

Keyword Arguments **ignore_row_if** (*str*) – “both_values_are_missing”, “either_value_is_missing”, “neither”

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: `BOOLEAN_ONLY`, `BASIC`, `COMPLETE`, or `SUMMARY`. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to *result_format* and *include_config*, *catch_exceptions*, and *meta*.

```
abstract expect_column_pair_values_A_to_be_greater_than_B(self, column_A,
                                                           column_B,
                                                           or_equal=None,
                                                           parse_strings_as_datetimes=False,
                                                           allow_cross_type_comparisons=None,
                                                           ignore_row_if='both_values_are_missing',
                                                           result_format=None,
                                                           include_config=True,
                                                           catch_exceptions=None,
                                                           meta=None)
```

Expect values in column A to be greater than column B.

Parameters

- **column_A** (*str*) – The first column name
- **column_B** (*str*) – The second column name
- **or_equal** (*boolean or None*) – If True, then values can be equal, not strictly greater

Keyword Arguments

- **allow_cross_type_comparisons** (*boolean or None*) – If True, allow comparisons between types (e.g. integer and string). Otherwise, attempting such comparisons will raise an exception.
- **ignore_row_if** (*str*) – “both_values_are_missing”, “either_value_is_missing”, “neither

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see *result_format*.
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see *include_config*.
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see *catch_exceptions*.
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see *meta*.

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to *result_format* and *include_config*, *catch_exceptions*, and *meta*.

```

abstract expect_column_pair_values_to_be_in_set (self, column_A, column_B, value_pairs_set, ignore_row_if='both_values_are_missing', result_format=None, include_config=True, catch_exceptions=None, meta=None)

```

Expect paired values from columns A and B to belong to a set of valid pairs.

Parameters

- **column_A** (*str*) – The first column name
- **column_B** (*str*) – The second column name
- **value_pairs_set** (*list of tuples*) – All the valid pairs to be matched

Keyword Arguments **ignore_row_if** (*str*) – “both_values_are_missing”, “either_value_is_missing”, “never”

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

```

abstract expect_multicolumn_values_to_be_unique (self, column_list, ignore_row_if='all_values_are_missing', result_format=None, include_config=True, catch_exceptions=None, meta=None)

```

Expect the values for each row to be unique across the columns listed.

Parameters **column_list** (*tuple or list*) – The first column name

Keyword Arguments **ignore_row_if** (*str*) – “all_values_are_missing”, “any_value_is_missing”, “never”

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).

- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

static `_parse_value_set` (*value_set*)

attempt_allowing_relative_error (*self*)

Subclasses can override this method if the respective data source (e.g., Redshift) supports “approximate” mode. In certain cases (e.g., for SparkDFDataset), a fraction between 0 and 1 (i.e., not only a boolean) is allowed.

`great_expectations.dataset.pandas_dataset`

Module Contents

Classes

<code>MetaPandasDataset(*args, **kwargs)</code>	MetaPandasDataset is a thin layer between Dataset and PandasDataset.
<code>PandasDataset(*args, **kwargs)</code>	PandasDataset instantiates the great_expectations Expectations API as a subclass of a pandas.DataFrame.

`great_expectations.dataset.pandas_dataset.logger`

class `great_expectations.dataset.pandas_dataset.MetaPandasDataset` (**args, **kwargs*)

Bases: `great_expectations.dataset.dataset.Dataset`

MetaPandasDataset is a thin layer between Dataset and PandasDataset.

This two-layer inheritance is required to make @classmethod decorators work.

Practically speaking, that means that MetaPandasDataset implements expectation decorators, like `column_map_expectation` and `column_aggregate_expectation`, and PandasDataset implements the expectation methods themselves.

classmethod `column_map_expectation` (*cls, func*)

Constructs an expectation using column-map semantics.

The MetaPandasDataset implementation replaces the “column” parameter supplied by the user with a pandas Series object containing the actual column from the relevant pandas dataframe. This simplifies the implementing expectation logic while preserving the standard Dataset signature and expected behavior.

See `column_map_expectation` for full documentation of this function.

classmethod `column_pair_map_expectation` (*cls, func*)

The `column_pair_map_expectation` decorator handles boilerplate issues surrounding the common pattern of evaluating truthiness of some condition on a per row basis across a pair of columns.

classmethod `multicolumn_map_expectation` (*cls, func*)

The `multicolumn_map_expectation` decorator handles boilerplate issues surrounding the common pattern of evaluating truthiness of some condition on a per row basis across a set of columns.

```
class great_expectations.dataset.pandas_dataset.PandasDataset (*args, **kwargs)
    Bases: great_expectations.dataset.pandas_dataset.MetaPandasDataset, pandas.
    DataFrame
```

PandasDataset instantiates the great_expectations Expectations API as a subclass of a pandas.DataFrame.

For the full API reference, please see Dataset

Notes

1. Samples and Subsets of PandaDataSet have ALL the expectations of the original data frame unless the user specifies the `discard_subset_failing_expectations = True` property on the original data frame.
2. Concatenations, joins, and merges of PandaDataSets contain NO expectations (since no autoinspection is performed by default).

`_internal_names`

`_internal_names_set`

property `_constructor` (*self*)

Used when a manipulation result has the same dimensions as the original.

`__finalize__` (*self*, *other*, *method=None*, **kwargs)

Propagate metadata from other to self.

Parameters

- **other** (the object from which to get the attributes that we are going) – to propagate
- **method** (optional, a passed method name ; possibly to take different) – types of propagation actions based on this

`get_row_count` (*self*)

Returns: int, table row count

`get_column_count` (*self*)

Returns: int, table column count

`get_table_columns` (*self*)

Returns: List[str], list of column names

`get_column_sum` (*self*, *column*)

Returns: float

`get_column_max` (*self*, *column*, *parse_strings_as_datetimes=False*)

Returns: any

`get_column_min` (*self*, *column*, *parse_strings_as_datetimes=False*)

Returns: any

`get_column_mean` (*self*, *column*)

Returns: float

`get_column_nonnull_count` (*self*, *column*)

Returns: int

`get_column_value_counts` (*self*, *column*, *sort='value'*, *collate=None*)

Get a series containing the frequency counts of unique values from the named column.

Parameters

- **column** – the column for which to obtain value_counts
- **sort** (*string*) – must be one of “value”, “count”, or “none”. - if “value” then values in the resulting partition object will be sorted lexicographically - if “count” then values will be sorted according to descending count (frequency) - if “none” then values will not be sorted
- **collate** (*string*) – the collate (sort) method to be used on supported backends (SqlAlchemy only)

Returns `pd.Series` of value counts for a column, sorted according to the value requested in sort

get_column_unique_count (*self, column*)

Returns: int

get_column_modes (*self, column*)

Returns: List[any], list of modes (ties OK)

get_column_median (*self, column*)

Returns: any

get_column_quantiles (*self, column, quantiles, allow_relative_error=False*)

Get the values in column closest to the requested quantiles :param column: name of column :type column: string :param quantiles: the quantiles to return. quantiles *must* be a tuple to ensure caching is possible :type quantiles: tuple of float

Returns the nearest values in the dataset to those quantiles

Return type List[any]

get_column_stddev (*self, column*)

Returns: float

get_column_hist (*self, column, bins*)

Get a histogram of column values :param column: the column for which to generate the histogram :param bins: the bins to slice the histogram. bins *must* be a tuple to ensure caching is possible :type bins: tuple

Returns: List[int], a list of counts corresponding to bins

get_column_count_in_range (*self, column, min_val=None, max_val=None, strict_min=False, strict_max=True*)

Returns: int

expect_column_values_to_be_unique (*self, column, mostly=None, result_format=None, include_config=True, catch_exceptions=None, meta=None*)

Expect each column value to be unique.

This expectation detects duplicates. All duplicated values are counted as exceptions.

For example, `[1, 2, 3, 3, 3]` will return `[3, 3, 3]` in `result.exceptions_list`, with `unexpected_percent = 60.0`.

`expect_column_values_to_be_unique` is a [column_map_expectation](#).

Parameters **column** (*str*) – The column name.

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return “success”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: `BOOLEAN_ONLY`, `BASIC`, `COMPLETE`, or `SUMMARY`. For more detail, see [result_format](#).

- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

expect_column_values_to_not_be_null (*self, column, mostly=None, result_format=None, include_config=True, catch_exceptions=None, meta=None, include_nulls=True*)

Expect column values to not be null.

To be counted as an exception, values must be explicitly null or missing, such as a NULL in PostgreSQL or an np.NaN in pandas. Empty strings don't count as null unless they have been coerced to a null type.

`expect_column_values_to_not_be_null` is a [column_map_expectation](#).

Parameters **column** (*str*) – The column name.

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return “success”: True if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[expect_column_values_to_be_null](#)

expect_column_values_to_be_null (*self, column, mostly=None, result_format=None, include_config=True, catch_exceptions=None, meta=None*)

Expect column values to be null.

`expect_column_values_to_be_null` is a [column_map_expectation](#).

Parameters **column** (*str*) – The column name.

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return “*success*”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[expect_column_values_to_not_be_null](#)

expect_column_values_to_be_of_type (*self, column, type_, **kwargs*)

The pandas implementation of this expectation takes *kwargs* mostly, *result_format*, *include_config*, *catch_exceptions*, and *meta* as other expectations, however it declares *****kwargs*** because it needs to be able to fork into either aggregate or map semantics depending on the column type (see below).

In Pandas, columns *may* be typed, or they may be of the generic “object” type which can include rows with different storage types in the same column.

To respect that implementation, the *expect_column_values_to_be_of_type* expectations will first attempt to use the column dtype information to determine whether the column is restricted to the provided type. If that is possible, then *expect_column_values_to_be_of_type* will return aggregate information including an *observed_value*, similarly to other backends.

If it is not possible (because the column dtype is “object” but a more specific type was specified), then *PandasDataset* will use column map semantics: it will return map expectation results and check each value individually, which can be substantially slower.

Unfortunately, the “object” type is also used to contain any string-type columns (including ‘*str*’ and numpy ‘*string_*’ (bytes)); consequently, it is not possible to test for string columns using aggregate semantics.

```
_expect_column_values_to_be_of_type_aggregate (self, column, type_, mostly=None,  
                                              result_format=None,  
                                              include_config=True,  
                                              catch_exceptions=None,  
                                              meta=None)
```

```
static _native_type_type_map (type_)
```

```
_expect_column_values_to_be_of_type_map (self, column, type_, mostly=None, re-  
                                         sult_format=None, include_config=True,  
                                         catch_exceptions=None, meta=None)
```

```
expect_column_values_to_be_in_type_list (self, column, type_list, **kwargs)
```

The pandas implementation of this expectation takes *kwargs* mostly, *result_format*, *include_config*,

catch_exceptions, and meta as other expectations, however it declares ****kwargs** because it needs to be able to fork into either aggregate or map semantics depending on the column type (see below).

In Pandas, columns *may* be typed, or they may be of the generic “object” type which can include rows with different storage types in the same column.

To respect that implementation, the `expect_column_values_to_be_of_type` expectations will first attempt to use the column dtype information to determine whether the column is restricted to the provided type. If that is possible, then `expect_column_values_to_be_of_type` will return aggregate information including an `observed_value`, similarly to other backends.

If it is not possible (because the column dtype is “object” but a more specific type was specified), then `PandasDataset` will use column map semantics: it will return map expectation results and check each value individually, which can be substantially slower.

Unfortunately, the “object” type is also used to contain any string-type columns (including ‘str’ and numpy **‘string_’** (bytes)); consequently, it is not possible to test for string columns using aggregate semantics.

```
_expect_column_values_to_be_in_type_list__aggregate (self, column, type_list,
                                                    mostly=None, re-
                                                    sult_format=None,
                                                    include_config=True,
                                                    catch_exceptions=None,
                                                    meta=None)

_expect_column_values_to_be_in_type_list__map (self, column, type_list,
                                                  mostly=None, result_format=None,
                                                  include_config=True,
                                                  catch_exceptions=None,
                                                  meta=None)

expect_column_values_to_be_in_set (self, column, value_set, mostly=None,
                                     parse_strings_as_datetimes=None, re-
                                     sult_format=None, include_config=True,
                                     catch_exceptions=None, meta=None)
```

Expect each column value to be in a given set.

For example:

```
# my_df.my_col = [1,2,2,3,3,3]
>>> my_df.expect_column_values_to_be_in_set(
    "my_col",
    [2,3]
)
{
  "success": false
  "result": {
    "unexpected_count": 1
    "unexpected_percent": 16.666666666666666,
    "unexpected_percent_nonmissing": 16.666666666666666,
    "partial_unexpected_list": [
      1
    ],
  },
}
```

`expect_column_values_to_be_in_set` is a *column_map_expectation*.

Parameters

- **column** (*str*) – The column name.

- **value_set** (*set-like*) – A set of objects used for comparison.

Keyword Arguments

- **mostly** (*None or a float between 0 and 1*) – Return “success”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).
- **parse_strings_as_datetimes** (*boolean or None*) – If *True* values provided in *value_set* will be parsed as datetimes before making comparisons.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[expect_column_values_to_not_be_in_set](#)

```
expect_column_values_to_not_be_in_set(self, column, value_set, mostly=None,  
                                       parse_strings_as_datetimes=None, re-  
                                       sult_format=None, include_config=True,  
                                       catch_exceptions=None, meta=None)
```

Expect column entries to not be in the set.

For example:

```
# my_df.my_col = [1, 2, 2, 3, 3, 3]
>>> my_df.expect_column_values_to_not_be_in_set(
    "my_col",
    [1, 2]
)
{
  "success": false
  "result": {
    "unexpected_count": 3
    "unexpected_percent": 50.0,
    "unexpected_percent_nonmissing": 50.0,
    "partial_unexpected_list": [
      1, 2, 2
    ],
  },
}
```

`expect_column_values_to_not_be_in_set` is a [column_map_expectation](#).

Parameters

- **column** (*str*) – The column name.
- **value_set** (*set-like*) – A set of objects used for comparison.

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return “success”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[expect_column_values_to_be_in_set](#)

```
expect_column_values_to_be_between(self, column, min_value=None, max_value=None,
                                     strict_min=False, strict_max=False,
                                     parse_strings_as_datetimes=None,
                                     output_strftime_format=None, allow_cross_type_comparisons=None,
                                     mostly=None, result_format=None, include_config=True,
                                     catch_exceptions=None, meta=None)
```

Expect column entries to be between a minimum value and a maximum value (inclusive).

`expect_column_values_to_be_between` is a [column_map_expectation](#).

Parameters

- **column** (*str*) – The column name.
- **min_value** (*comparable type or None*) – The minimum value for a column entry.
- **max_value** (*comparable type or None*) – The maximum value for a column entry.

Keyword Arguments

- **strict_min** (*boolean*) – If *True*, values must be strictly larger than `min_value`, default=*False*
- **strict_max** (*boolean*) – If *True*, values must be strictly smaller than `max_value`, default=*False*
- **allow_cross_type_comparisons** (*boolean or None*) : If *True*, allow comparisons between types (e.g. integer and string). Otherwise, attempting such comparisons will raise an exception.

- **parse_strings_as_datetimes** (*boolean or None*) – If True, parse `min_value`, `max_value`, and all non-null column values to datetimes before making comparisons.
- **output_strftime_format** (*str or None*) – A valid strftime format for datetime output. Only used if `parse_strings_as_datetimes=True`.
- **mostly** (*None or a float between 0 and 1*) – Return “success”: True if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: `BOOLEAN_ONLY`, `BASIC`, `COMPLETE`, or `SUMMARY`. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

Notes

- `min_value` and `max_value` are both inclusive unless `strict_min` or `strict_max` are set to True.
- If `min_value` is None, then `max_value` is treated as an upper bound, and there is no minimum value checked.
- If `max_value` is None, then `min_value` is treated as a lower bound, and there is no maximum value checked.

See also:

[expect_column_value_lengths_to_be_between](#)

```
expect_column_values_to_be_increasing(self,          column,          strictly=None,
                                       parse_strings_as_datetimes=None,
                                       mostly=None,      result_format=None,  in-
                                       clude_config=True,  catch_exceptions=None,
                                       meta=None)
```

Expect column values to be increasing.

By default, this expectation only works for numeric or datetime data. When `parse_strings_as_datetimes=True`, it can also parse strings to datetimes.

If `strictly=True`, then this expectation is only satisfied if each consecutive value is strictly increasing—equal values are treated as failures.

`expect_column_values_to_be_increasing` is a [column_map_expectation](#).

Parameters `column` (*str*) – The column name.

Keyword Arguments

- **strictly** (*Boolean or None*) – If True, values must be strictly greater than previous values
- **parse_strings_as_datetimes** (*boolean or None*) – If True, all non-null column values to datetimes before making comparisons
- **mostly** (*None or a float between 0 and 1*) – Return “success”: True if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[expect_column_values_to_be_decreasing](#)

```
expect_column_values_to_be_decreasing(self,          column,          strictly=None,
                                       parse_strings_as_datetimes=None,
                                       mostly=None,      result_format=None,  in-
                                       clude_config=True,  catch_exceptions=None,
                                       meta=None)
```

Expect column values to be decreasing.

By default, this expectation only works for numeric or datetime data. When *parse_strings_as_datetimes=True*, it can also parse strings to datetimes.

If *strictly=True*, then this expectation is only satisfied if each consecutive value is strictly decreasing—equal values are treated as failures.

`expect_column_values_to_be_decreasing` is a [column_map_expectation](#).

Parameters `column` (*str*) – The column name.

Keyword Arguments

- **strictly** (*Boolean or None*) – If True, values must be strictly greater than previous values
- **parse_strings_as_datetimes** (*boolean or None*) – If True, all non-null column values to datetimes before making comparisons
- **mostly** (*None or a float between 0 and 1*) – Return “success”: True if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).

- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[expect_column_values_to_be_increasing](#)

expect_column_value_lengths_to_be_between (*self, column, min_value=None, max_value=None, mostly=None, result_format=None, include_config=True, catch_exceptions=None, meta=None*)

Expect column entries to be strings with length between a minimum value and a maximum value (inclusive).

This expectation only works for string-type values. Invoking it on ints or floats will raise a `TypeError`.

`expect_column_value_lengths_to_be_between` is a [column_map_expectation](#).

Parameters `column` (*str*) – The column name.

Keyword Arguments

- **min_value** (*int or None*) – The minimum value for a column entry length.
- **max_value** (*int or None*) – The maximum value for a column entry length.
- **mostly** (*None or a float between 0 and 1*) – Return “success”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

Notes

- `min_value` and `max_value` are both inclusive.
- If `min_value` is `None`, then `max_value` is treated as an upper bound, and the number of acceptable rows has no minimum.
- If `max_value` is `None`, then `min_value` is treated as a lower bound, and the number of acceptable rows has no maximum.

See also:

[`expect_column_value_lengths_to_equal`](#)

`expect_column_value_lengths_to_equal`(*self*, *column*, *value*, *mostly=None*, *result_format=None*, *include_config=True*, *catch_exceptions=None*, *meta=None*)

Expect column entries to be strings with length equal to the provided value.

This expectation only works for string-type values. Invoking it on ints or floats will raise a `TypeError`.

`expect_column_values_to_be_between` is a [`column_map_expectation`](#).

Parameters

- **`column`** (*str*) – The column name.
- **`value`** (*int or None*) – The expected value for a column entry length.

Keyword Arguments **`mostly`** (*None or a float between 0 and 1*) – Return “*success*”: *True* if at least mostly fraction of values match the expectation. For more detail, see [`mostly`](#).

Other Parameters

- **`result_format`** (*str or None*) – Which output mode to use: `BOOLEAN_ONLY`, `BASIC`, `COMPLETE`, or `SUMMARY`. For more detail, see [`result_format`](#).
- **`include_config`** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see [`include_config`](#).
- **`catch_exceptions`** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see [`catch_exceptions`](#).
- **`meta`** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [`meta`](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [`result_format`](#) and [`include_config`](#), [`catch_exceptions`](#), and [`meta`](#).

See also:

[`expect_column_value_lengths_to_be_between`](#)

`expect_column_values_to_match_regex`(*self*, *column*, *regex*, *mostly=None*, *result_format=None*, *include_config=True*, *catch_exceptions=None*, *meta=None*)

Expect column entries to be strings that match a given regular expression. Valid matches can be found anywhere in the string, for example “[at]+” will identify the following strings as expected: “cat”, “hat”, “aa”, “a”, and “t”, and the following strings as unexpected: “fish”, “dog”.

`expect_column_values_to_match_regex` is a *column_map_expectation*.

Parameters

- **column** (*str*) – The column name.
- **regex** (*str*) – The regular expression the column entries should match.

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return “*success*”: *True* if at least mostly fraction of values match the expectation. For more detail, see *mostly*.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see *result_format*.
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see *include_config*.
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see *catch_exceptions*.
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see *meta*.

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to *result_format* and *include_config*, *catch_exceptions*, and *meta*.

See also:

expect_column_values_to_not_match_regex

expect_column_values_to_match_regex_list

expect_column_values_to_not_match_regex (*self, column, regex, mostly=None, result_format=None, include_config=True, catch_exceptions=None, meta=None*)

Expect column entries to be strings that do NOT match a given regular expression. The regex must not match any portion of the provided string. For example, “[at]+” would identify the following strings as expected: “fish”, “dog”, and the following as unexpected: “cat”, “hat”.

`expect_column_values_to_not_match_regex` is a *column_map_expectation*.

Parameters

- **column** (*str*) – The column name.
- **regex** (*str*) – The regular expression the column entries should NOT match.

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return “*success*”: *True* if at least mostly fraction of values match the expectation. For more detail, see *mostly*.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see *result_format*.
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see *include_config*.

- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[expect_column_values_to_match_regex](#)

[expect_column_values_to_match_regex_list](#)

```
expect_column_values_to_match_regex_list(self, column, regex_list,
                                          match_on='any', mostly=None, re-
                                          sult_format=None, include_config=True,
                                          catch_exceptions=None, meta=None)
```

Expect the column entries to be strings that can be matched to either any of or all of a list of regular expressions. Matches can be anywhere in the string.

`expect_column_values_to_match_regex_list` is a [column_map_expectation](#).

Parameters

- **column** (*str*) – The column name.
- **regex_list** (*list*) – The list of regular expressions which the column entries should match

Keyword Arguments

- **match_on=** (*string*) – “any” or “all”. Use “any” if the value should match at least one regular expression in the list. Use “all” if it should match each regular expression in the list.
- **mostly** (*None or a float between 0 and 1*) – Return “success”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[`expect_column_values_to_match_regex`](#)

[`expect_column_values_to_not_match_regex`](#)

`expect_column_values_to_not_match_regex_list` (*self*, *column*, *regex_list*,
mostly=None, *result_format=None*,
include_config=True,
catch_exceptions=None,
meta=None)

Expect the column entries to be strings that do not match any of a list of regular expressions. Matches can be anywhere in the string.

`expect_column_values_to_not_match_regex_list` is a [`column_map_expectation`](#).

Parameters

- **`column`** (*str*) – The column name.
- **`regex_list`** (*list*) – The list of regular expressions which the column entries should not match

Keyword Arguments **`mostly`** (*None* or a float between 0 and 1) – Return “*success*”: *True* if at least mostly fraction of values match the expectation. For more detail, see [`mostly`](#).

Other Parameters

- **`result_format`** (*str* or *None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [`result_format`](#).
- **`include_config`** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see [`include_config`](#).
- **`catch_exceptions`** (*boolean* or *None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see [`catch_exceptions`](#).
- **`meta`** (*dict* or *None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [`meta`](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [`result_format`](#) and [`include_config`](#), [`catch_exceptions`](#), and [`meta`](#).

See also:

[`expect_column_values_to_match_regex_list`](#)

`expect_column_values_to_match_strftime_format` (*self*, *column*, *strftime_format*,
mostly=None, *result_format=None*,
include_config=True,
catch_exceptions=None,
meta=None)

Expect column entries to be strings representing a date or time with a given format.

`expect_column_values_to_match_strftime_format` is a [`column_map_expectation`](#).

Parameters

- **`column`** (*str*) – The column name.
- **`strftime_format`** (*str*) – A strftime format string to use for matching

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return “*success*”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

```
expect_column_values_to_be_dateutil_parseable(self, column, mostly=None,
                                              result_format=None,
                                              include_config=True,
                                              catch_exceptions=None,
                                              meta=None)
```

Expect column entries to be parsable using dateutil.

`expect_column_values_to_be_dateutil_parseable` is a [column_map_expectation](#).

Parameters **column** (*str*) – The column name.

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return “*success*”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

```
expect_column_values_to_be_json_parseable(self, column, mostly=None, re-
                                          sult_format=None, include_config=True,
                                          catch_exceptions=None, meta=None)
```

Expect column entries to be data written in JavaScript Object Notation.

`expect_column_values_to_be_json_parseable` is a *column_map_expectation*.

Parameters `column` (*str*) – The column name.

Keyword Arguments `mostly` (*None or a float between 0 and 1*) – Return “*success*”: *True* if at least mostly fraction of values match the expectation. For more detail, see *mostly*.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see *result_format*.
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see *include_config*.
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see *catch_exceptions*.
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see *meta*.

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to *result_format* and *include_config*, *catch_exceptions*, and *meta*.

See also:

expect_column_values_to_match_json_schema

```
expect_column_values_to_match_json_schema(self, column, json_schema,
                                           mostly=None, result_format=None,
                                           include_config=True,
                                           catch_exceptions=None, meta=None)
```

Expect column entries to be JSON objects matching a given JSON schema.

`expect_column_values_to_match_json_schema` is a *column_map_expectation*.

Parameters `column` (*str*) – The column name.

Keyword Arguments `mostly` (*None or a float between 0 and 1*) – Return “*success*”: *True* if at least mostly fraction of values match the expectation. For more detail, see *mostly*.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see *result_format*.
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see *include_config*.
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see *catch_exceptions*.
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see *meta*.

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to *result_format* and *include_config*, *catch_exceptions*, and *meta*.

See also:

expect_column_values_to_be_json_parseable

The JSON-schema docs.

```
expect_column_parameterized_distribution_ks_test_p_value_to_be_greater_than(self,
                                                                              column,
                                                                              distribution,
                                                                              p_value=0.05,
                                                                              params=None,
                                                                              result_format=None,
                                                                              include_config=True,
                                                                              catch_exceptions=True,
                                                                              meta=None)
```

Expect the column values to be distributed similarly to a scipy distribution. This expectation compares the provided column to the specified continuous distribution with a parametric Kolmogorov-Smirnov test. The K-S test compares the provided column to the cumulative density function (CDF) of the specified scipy distribution. If you don't know the desired distribution shape parameters, use the *ge.dataset.util.infer_distribution_parameters()* utility function to estimate them.

It returns 'success'=True if the p-value from the K-S test is greater than or equal to the provided p-value.

expect_column_parameterized_distribution_ks_test_p_value_to_be_greater_than is a *column_aggregate_expectation*.

Parameters

- **column** (*str*) – The column name.
- **distribution** (*str*) – The scipy distribution name. See: <https://docs.scipy.org/doc/scipy/reference/stats.html> Currently supported distributions are listed in the Notes section below.
- **p_value** (*float*) – The threshold p-value for a passing test. Default is 0.05.
- **params** (*dict or list*) – A dictionary or positional list of shape parameters that describe the distribution you want to test the data against. Include key values specific to the distribution from the appropriate scipy distribution CDF function. 'loc' and 'scale' are used as translational parameters. See <https://docs.scipy.org/doc/scipy/reference/stats.html#continuous-distributions>

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see *result_format*.
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see *include_config*.
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see *catch_exceptions*.

- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

Notes

These fields in the result object are customized for this expectation:

```
{
    "details": {
        "expected_params" (dict): The specified or inferred parameters of the
        ↪ distribution to test against
        "ks_results" (dict): The raw result of stats.kstest()
    }
}
```

- The Kolmogorov-Smirnov test's null hypothesis is that the column is similar to the provided distribution.
- Supported scipy distributions:
 - norm
 - beta
 - gamma
 - uniform
 - chi2
 - expon

```
expect_column_bootstrapped_ks_test_p_value_to_be_greater_than(self, column, partition_object=None, p=0.05, bootstrap_samples=None, bootstrap_sample_size=None, result_format=None, include_config=True, catch_exceptions=None, meta=None)
```

Expect column values to be distributed similarly to the provided continuous partition. This expectation compares continuous distributions using a bootstrapped Kolmogorov-Smirnov test. It returns *success=True* if values in the column match the distribution of the provided partition.

The expected cumulative density function (CDF) is constructed as a linear interpolation between the bins, using the provided weights. Consequently the test expects a piecewise uniform distribution using the bins from the provided partition object.

`expect_column_bootstrapped_ks_test_p_value_to_be_greater_than` is a `column_aggregate_expectation`.

Parameters

- **column** (*str*) – The column name.
- **partition_object** (*dict*) – The expected partition object (see [Partition Objects](#)).
- **p** (*float*) – The p-value threshold for the Kolmogorov-Smirnov test. For values below the specified threshold the expectation will return `success=False`, rejecting the null hypothesis that the distributions are the same. Defaults to 0.05.

Keyword Arguments

- **bootstrap_samples** (*int*) – The number bootstrap rounds. Defaults to 1000.
- **bootstrap_sample_size** (*int*) – The number of samples to take from the column for each bootstrap. A larger sample will increase the specificity of the test. Defaults to `2 * len(partition_object['weights'])`

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: `BOOLEAN_ONLY`, `BASIC`, `COMPLETE`, or `SUMMARY`. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

Notes

These fields in the result object are customized for this expectation:

```
{
  "observed_value": (float) The true p-value of the KS test
  "details": {
    "bootstrap_samples": The number of bootstrap rounds used
    "bootstrap_sample_size": The number of samples taken from
                           the column in each bootstrap round
    "observed_cdf": The cumulative density function observed
                   in the data, a dict containing 'x' values and cdf_values
                   (suitable for plotting)
    "expected_cdf" (dict):
      The cumulative density function expected based on the
      partition object, a dict containing 'x' values and
      cdf_values (suitable for plotting)
    "observed_partition" (dict):
      The partition observed on the data, using the provided
```

(continues on next page)

(continued from previous page)

```

        bins but also expanding from min(column) to max(column)
    "expected_partition" (dict):
        The partition expected from the data. For KS test,
        this will always be the partition_object parameter
    }
}

```

```

expect_column_pair_values_to_be_equal(self, column_A, column_B, ignore_row_if='both_values_are_missing',
                                       result_format=None, include_config=True,
                                       catch_exceptions=None, meta=None)

```

Expect the values in column A to be the same as column B.

Parameters

- **column_A** (*str*) – The first column name
- **column_B** (*str*) – The second column name

Keyword Arguments **ignore_row_if** (*str*) – “both_values_are_missing”, “either_value_is_missing”, “neither”

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

```

expect_column_pair_values_A_to_be_greater_than_B(self, column_A, column_B, or_equal=None,
                                                  parse_strings_as_datetimes=None,
                                                  allow_cross_type_comparisons=None,
                                                  ignore_row_if='both_values_are_missing',
                                                  result_format=None,
                                                  include_config=True,
                                                  catch_exceptions=None,
                                                  meta=None)

```

Expect values in column A to be greater than column B.

Parameters

- **column_A** (*str*) – The first column name
- **column_B** (*str*) – The second column name

- **or_equal** (*boolean or None*) – If True, then values can be equal, not strictly greater

Keyword Arguments

- **allow_cross_type_comparisons** (*boolean or None*) – If True, allow comparisons between types (e.g. integer and string). Otherwise, attempting such comparisons will raise an exception.
- **ignore_row_if** (*str*) – “both_values_are_missing”, “either_value_is_missing”, “neither”

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

```
expect_column_pair_values_to_be_in_set (self, column_A, column_B, value_pairs_set,
                                         ignore_row_if='both_values_are_missing',
                                         result_format=None, include_config=True,
                                         catch_exceptions=None, meta=None)
```

Expect paired values from columns A and B to belong to a set of valid pairs.

Parameters

- **column_A** (*str*) – The first column name
- **column_B** (*str*) – The second column name
- **value_pairs_set** (*list of tuples*) – All the valid pairs to be matched

Keyword Arguments **ignore_row_if** (*str*) – “both_values_are_missing”, “either_value_is_missing”, “never”

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to *result_format* and *include_config*, *catch_exceptions*, and *meta*.

```
expect_multicolumn_values_to_be_unique (self, column_list, ignore_row_if='all_values_are_missing', result_format=None, include_config=True, catch_exceptions=None, meta=None)
```

Expect the values for each row to be unique across the columns listed.

Parameters *column_list* (*tuple or list*) – The first column name

Keyword Arguments *ignore_row_if* (*str*) – “all_values_are_missing”, “any_value_is_missing”, “never”

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see *result_format*.
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see *include_config*.
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see *catch_exceptions*.
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see *meta*.

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to *result_format* and *include_config*, *catch_exceptions*, and *meta*.

`great_expectations.dataset.sparkdf_dataset`

Module Contents**Classes**

<code>MetaSparkDFDataset(*args, **kwargs)</code>	MetaSparkDFDataset is a thin layer between Dataset and SparkDFDataset.
<code>SparkDFDataset(spark_df, *args, **kwargs)</code>	This class holds an attribute <i>spark_df</i> which is a <code>spark.sql.DataFrame</code> .

`great_expectations.dataset.sparkdf_dataset.logger`

```
class great_expectations.dataset.sparkdf_dataset.MetaSparkDFDataset (*args,  
                                                                    **kwargs)
```

Bases: `great_expectations.dataset.dataset.Dataset`

MetaSparkDFDataset is a thin layer between Dataset and SparkDFDataset. This two-layer inheritance is required to make @classmethod decorators work. Practically speaking, that means that MetaSparkDFDataset implements expectation decorators, like *column_map_expectation* and *column_aggregate_expectation*, and

SparkDFDataset implements the expectation methods themselves.

classmethod column_map_expectation (*cls, func*)

Constructs an expectation using column-map semantics.

The MetaSparkDFDataset implementation replaces the “column” parameter supplied by the user with a Spark Dataframe with the actual column data. The current approach for functions implementing expectation logic is to append a column named “__success” to this dataframe and return to this decorator.

See `column_map_expectation` for full documentation of this function.

classmethod column_pair_map_expectation (*cls, func*)

The `column_pair_map_expectation` decorator handles boilerplate issues surrounding the common pattern of evaluating truthiness of some condition on a per row basis across a pair of columns.

classmethod multicolumn_map_expectation (*cls, func*)

The `multicolumn_map_expectation` decorator handles boilerplate issues surrounding the common pattern of evaluating truthiness of some condition on a per row basis across a set of columns.

class `great_expectations.dataset.sparkdf_dataset.SparkDFDataset` (*spark_df,*
**args,*
***kwargs*)

Bases: `great_expectations.dataset.sparkdf_dataset.MetaSparkDFDataset`

This class holds an attribute `spark_df` which is a `spark.sql.DataFrame`.

classmethod from_dataset (*cls, dataset=None*)

This base implementation naively passes arguments on to the real constructor, which is suitable really when a constructor knows to take its own type. In general, this should be overridden

head (*self, n=5*)

Returns a *PandasDataset* with the first *n* rows of the given Dataset

get_row_count (*self*)

Returns: int, table row count

get_column_count (*self*)

Returns: int, table column count

get_table_columns (*self*)

Returns: List[str], list of column names

get_column_nonnull_count (*self, column*)

Returns: int

get_column_mean (*self, column*)

Returns: float

get_column_sum (*self, column*)

Returns: float

_describe_column (*self, column*)

get_column_max (*self, column, parse_strings_as_datetimes=False*)

Returns: any

get_column_min (*self, column, parse_strings_as_datetimes=False*)

Returns: any

get_column_value_counts (*self, column, sort='value', collate=None*)

Get a series containing the frequency counts of unique values from the named column.

Parameters

- **column** – the column for which to obtain `value_counts`

- **sort** (*string*) – must be one of “value”, “count”, or “none”. - if “value” then values in the resulting partition object will be sorted lexicographically - if “count” then values will be sorted according to descending count (frequency) - if “none” then values will not be sorted
- **collate** (*string*) – the collate (sort) method to be used on supported backends (SqlAlchemy only)

Returns `pd.Series` of value counts for a column, sorted according to the value requested in sort

get_column_unique_count (*self, column*)

Returns: int

get_column_modes (*self, column*)

leverages computation done in `_get_column_value_counts`

get_column_median (*self, column*)

Returns: any

get_column_quantiles (*self, column, quantiles, allow_relative_error=False*)

Get the values in column closest to the requested quantiles :param column: name of column :type column: string :param quantiles: the quantiles to return. quantiles *must* be a tuple to ensure caching is possible :type quantiles: tuple of float

Returns the nearest values in the dataset to those quantiles

Return type List[any]

get_column_stdev (*self, column*)

Returns: float

get_column_hist (*self, column, bins*)

return a list of counts corresponding to bins

get_column_count_in_range (*self, column, min_val=None, max_val=None, strict_min=False, strict_max=True*)

Returns: int

static _apply_dateutil_parse (*column*)

expect_column_values_to_be_in_set (*self, column, value_set, mostly=None, parse_strings_as_datetimes=None, result_format=None, include_config=True, catch_exceptions=None, meta=None*)

Expect each column value to be in a given set.

For example:

```
# my_df.my_col = [1,2,2,3,3,3]
>>> my_df.expect_column_values_to_be_in_set(
    "my_col",
    [2,3]
)
{
  "success": false
  "result": {
    "unexpected_count": 1
    "unexpected_percent": 16.666666666666666,
    "unexpected_percent_nonmissing": 16.666666666666666,
    "partial_unexpected_list": [
      1
    ],
  },
}
```

(continues on next page)

(continued from previous page)

```

    },
}

```

`expect_column_values_to_be_in_set` is a *column_map_expectation*.

Parameters

- **column** (*str*) – The column name.
- **value_set** (*set-like*) – A set of objects used for comparison.

Keyword Arguments

- **mostly** (*None or a float between 0 and 1*) – Return “success”: *True* if at least mostly fraction of values match the expectation. For more detail, see *mostly*.
- **parse_strings_as_datetimes** (*boolean or None*) – If *True* values provided in *value_set* will be parsed as datetimes before making comparisons.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see *result_format*.
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see *include_config*.
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see *catch_exceptions*.
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see *meta*.

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to *result_format* and *include_config*, *catch_exceptions*, and *meta*.

See also:

expect_column_values_to_not_be_in_set

```
expect_column_values_to_not_be_in_set(self, column, value_set, mostly=None, re-
                                         sult_format=None, include_config=True,
                                         catch_exceptions=None, meta=None)
```

Expect column entries to not be in the set.

For example:

```

# my_df.my_col = [1,2,2,3,3,3]
>>> my_df.expect_column_values_to_not_be_in_set(
    "my_col",
    [1,2]
)
{
  "success": false
  "result": {
    "unexpected_count": 3
    "unexpected_percent": 50.0,
    "unexpected_percent_nonmissing": 50.0,

```

(continues on next page)

(continued from previous page)

```

    "partial_unexpected_list": [
        1, 2, 2
    ],
},
}

```

`expect_column_values_to_not_be_in_set` is a *column_map_expectation*.

Parameters

- **column** (*str*) – The column name.
- **value_set** (*set-like*) – A set of objects used for comparison.

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return “*success*”: *True* if at least mostly fraction of values match the expectation. For more detail, see *mostly*.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see *result_format*.
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see *include_config*.
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see *catch_exceptions*.
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see *meta*.

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to *result_format* and *include_config*, *catch_exceptions*, and *meta*.

See also:

expect_column_values_to_be_in_set

expect_column_values_to_be_between (*self, column, min_value=None, max_value=None, strict_min=False, strict_max=False, parse_strings_as_datetimes=None, output_strftime_format=None, allow_cross_type_comparisons=None, mostly=None, result_format=None, include_config=True, catch_exceptions=None, meta=None*)

Expect column entries to be between a minimum value and a maximum value (inclusive).

`expect_column_values_to_be_between` is a *column_map_expectation*.

Parameters

- **column** (*str*) – The column name.
- **min_value** (*comparable type or None*) – The minimum value for a column entry.
- **max_value** (*comparable type or None*) – The maximum value for a column entry.

Keyword Arguments

- **strict_min** (*boolean*) – If True, values must be strictly larger than min_value, default=False
- **strict_max** (*boolean*) – If True, values must be strictly smaller than max_value, default=False
- **allow_cross_type_comparisons** (*boolean or None*) : If True, allow comparisons between types (e.g. integer and string). Otherwise, attempting such comparisons will raise an exception.
- **parse_strings_as_datetimes** (*boolean or None*) – If True, parse min_value, max_value, and all non-null column values to datetimes before making comparisons.
- **output_strftime_format** (*str or None*) – A valid strftime format for datetime output. Only used if parse_strings_as_datetimes=True.
- **mostly** (*None or a float between 0 and 1*) – Return “success”: True if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

Notes

- min_value and max_value are both inclusive unless strict_min or strict_max are set to True.
- If min_value is None, then max_value is treated as an upper bound, and there is no minimum value checked.
- If max_value is None, then min_value is treated as a lower bound, and there is no maximum value checked.

See also:

[expect_column_value_lengths_to_be_between](#)

```
expect_column_value_lengths_to_be_between(self, column, min_value=None,
                                             max_value=None, mostly=None, re-
                                             sult_format=None, include_config=True,
                                             catch_exceptions=None, meta=None)
```

Expect column entries to be strings with length between a minimum value and a maximum value (inclusive).

This expectation only works for string-type values. Invoking it on ints or floats will raise a `TypeError`.

`expect_column_value_lengths_to_be_between` is a [column_map_expectation](#).

Parameters `column` (*str*) – The column name.

Keyword Arguments

- **min_value** (*int or None*) – The minimum value for a column entry length.
- **max_value** (*int or None*) – The maximum value for a column entry length.
- **mostly** (*None or a float between 0 and 1*) – Return “success”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

Notes

- `min_value` and `max_value` are both inclusive.
- If `min_value` is *None*, then `max_value` is treated as an upper bound, and the number of acceptable rows has no minimum.
- If `max_value` is *None*, then `min_value` is treated as a lower bound, and the number of acceptable rows has no maximum.

See also:

[expect_column_value_lengths_to_equal](#)

expect_column_values_to_be_unique (*self, column, mostly=None, result_format=None, include_config=True, catch_exceptions=None, meta=None*)

Expect each column value to be unique.

This expectation detects duplicates. All duplicated values are counted as exceptions.

For example, `[1, 2, 3, 3, 3]` will return `[3, 3, 3]` in `result.exceptions_list`, with `unexpected_percent = 60.0`.

`expect_column_values_to_be_unique` is a [column_map_expectation](#).

Parameters `column` (*str*) – The column name.

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return “*success*”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

expect_column_value_lengths_to_equal (*self, column, value, mostly=None, result_format=None, include_config=True, catch_exceptions=None, meta=None*)

Expect column entries to be strings with length equal to the provided value.

This expectation only works for string-type values. Invoking it on ints or floats will raise a *TypeError*.

`expect_column_values_to_be_between` is a [column_map_expectation](#).

Parameters

- **column** (*str*) – The column name.
- **value** (*int or None*) – The expected value for a column entry length.

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return “*success*”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[`expect_column_value_lengths_to_be_between`](#)

`expect_column_values_to_match_strftime_format` (*self*, *column*, *strftime_format*, *mostly=None*, *result_format=None*, *include_config=True*, *catch_exceptions=None*, *meta=None*)

Expect column entries to be strings representing a date or time with a given format.

`expect_column_values_to_match_strftime_format` is a [`column_map_expectation`](#).

Parameters

- **`column`** (*str*) – The column name.
- **`strftime_format`** (*str*) – A strftime format string to use for matching

Keyword Arguments **`mostly`** (*None* or a float between 0 and 1) – Return “success”: *True* if at least mostly fraction of values match the expectation. For more detail, see [`mostly`](#).

Other Parameters

- **`result_format`** (*str* or *None*) – Which output mode to use: `BOOLEAN_ONLY`, `BASIC`, `COMPLETE`, or `SUMMARY`. For more detail, see [`result_format`](#).
- **`include_config`** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see [`include_config`](#).
- **`catch_exceptions`** (*boolean* or *None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see [`catch_exceptions`](#).
- **`meta`** (*dict* or *None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [`meta`](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [`result_format`](#) and [`include_config`](#), [`catch_exceptions`](#), and [`meta`](#).

`expect_column_values_to_not_be_null` (*self*, *column*, *mostly=None*, *result_format=None*, *include_config=True*, *catch_exceptions=None*, *meta=None*)

Expect column values to not be null.

To be counted as an exception, values must be explicitly null or missing, such as a `NULL` in PostgreSQL or an `np.NaN` in pandas. Empty strings don’t count as null unless they have been coerced to a null type.

`expect_column_values_to_not_be_null` is a [`column_map_expectation`](#).

Parameters **`column`** (*str*) – The column name.

Keyword Arguments **`mostly`** (*None* or a float between 0 and 1) – Return “success”: *True* if at least mostly fraction of values match the expectation. For more detail, see [`mostly`](#).

Other Parameters

- **`result_format`** (*str* or *None*) – Which output mode to use: `BOOLEAN_ONLY`, `BASIC`, `COMPLETE`, or `SUMMARY`. For more detail, see [`result_format`](#).

- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[expect_column_values_to_be_null](#)

```
expect_column_values_to_be_null(self, column, mostly=None, result_format=None,
                                include_config=True, catch_exceptions=None,
                                meta=None)
```

Expect column values to be null.

`expect_column_values_to_be_null` is a [column_map_expectation](#).

Parameters `column` (*str*) – The column name.

Keyword Arguments `mostly` (*None or a float between 0 and 1*) – Return “success”: True if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: `BOOLEAN_ONLY`, `BASIC`, `COMPLETE`, or `SUMMARY`. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[expect_column_values_to_not_be_null](#)

```
expect_column_values_to_match_json_schema(self, column, json_schema,
                                            mostly=None, result_format=None,
                                            include_config=True,
                                            catch_exceptions=None, meta=None)
```

Expect column entries to be JSON objects matching a given JSON schema.

`expect_column_values_to_match_json_schema` is a [column_map_expectation](#).

Parameters `column` (*str*) – The column name.

Keyword Arguments `mostly` (*None or a float between 0 and 1*) – Return “*success*”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[expect_column_values_to_be_json_parseable](#)

The [JSON-schema docs](#).

expect_column_values_to_be_of_type (*self*, *column*, *type_*, *mostly=None*, *result_format=None*, *include_config=True*, *catch_exceptions=None*, *meta=None*)

Expect a column to contain values of a specified data type.

`expect_column_values_to_be_of_type` is a [column_aggregate_expectation](#) for typed-column backends, and also for `PandasDataset` where the column dtype and provided **type_** are unambiguous constraints (any dtype except ‘object’ or dtype of ‘object’ with **type_** specified as ‘object’).

For `PandasDataset` columns with dtype of ‘object’ `expect_column_values_to_be_of_type` is a [column_map_expectation](#) and will independently check each row’s type.

Parameters

- **column** (*str*) – The column name.
- **type_** (*str*) – A string representing the data type that each column should have as entries. Valid types are defined by the current backend implementation and are dynamically loaded. For example, valid types for `PandasDataset` include any numpy dtype values (such as ‘int64’) or native python types (such as ‘int’), whereas valid types for a `SqlAlchemyDataset` include types named by the current driver such as ‘INTEGER’ in most SQL dialects and ‘TEXT’ in dialects such as postgresql. Valid types for `SparkDFDataset` include ‘StringType’, ‘BooleanType’ and other pyspark-defined type names.

Keyword Arguments `mostly` (*None or a float between 0 and 1*) – Return “*success*”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).

- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[expect_column_values_to_be_in_type_list](#)

expect_column_values_to_be_in_type_list(*self, column, type_list, mostly=None, result_format=None, include_config=True, catch_exceptions=None, meta=None*)

Expect a column to contain values from a specified type list.

`expect_column_values_to_be_in_type_list` is a [column_aggregate_expectation](#) for typed-column backends, and also for PandasDataset where the column dtype provides an unambiguous constraints (any dtype except 'object'). For PandasDataset columns with dtype of 'object' `expect_column_values_to_be_of_type` is a [column_map_expectation](#) and will independently check each row's type.

Parameters

- **column** (*str*) – The column name.
- **type_list** (*str*) – A list of strings representing the data type that each column should have as entries. Valid types are defined by the current backend implementation and are dynamically loaded. For example, valid types for PandasDataset include any numpy dtype values (such as 'int64') or native python types (such as 'int'), whereas valid types for a SQLAlchemyDataset include types named by the current driver such as 'INTEGER' in most SQL dialects and 'TEXT' in dialects such as postgresql. Valid types for SparkDFDataset include 'StringType', 'BooleanType' and other pyspark-defined type names.

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return “success”: True if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to *result_format* and *include_config*, *catch_exceptions*, and *meta*.

See also:

expect_column_values_to_be_of_type

```
expect_column_values_to_match_regex(self, column, regex, mostly=None, result_format=None, include_config=True, catch_exceptions=None, meta=None)
```

Expect column entries to be strings that match a given regular expression. Valid matches can be found anywhere in the string, for example “[at]+” will identify the following strings as expected: “cat”, “hat”, “aa”, “a”, and “t”, and the following strings as unexpected: “fish”, “dog”.

expect_column_values_to_match_regex is a *column_map_expectation*.

Parameters

- **column** (*str*) – The column name.
- **regex** (*str*) – The regular expression the column entries should match.

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return “success”: *True* if at least mostly fraction of values match the expectation. For more detail, see *mostly*.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see *result_format*.
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see *include_config*.
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see *catch_exceptions*.
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see *meta*.

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to *result_format* and *include_config*, *catch_exceptions*, and *meta*.

See also:

expect_column_values_to_not_match_regex

expect_column_values_to_match_regex_list

```
expect_column_values_to_not_match_regex(self, column, regex, mostly=None, result_format=None, include_config=True, catch_exceptions=None, meta=None)
```

Expect column entries to be strings that do NOT match a given regular expression. The regex must not match any portion of the provided string. For example, “[at]+” would identify the following strings as expected: “fish”, “dog”, and the following as unexpected: “cat”, “hat”.

expect_column_values_to_not_match_regex is a *column_map_expectation*.

Parameters

- **column** (*str*) – The column name.

- **regex** (*str*) – The regular expression the column entries should NOT match.

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return “success”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[expect_column_values_to_match_regex](#)

[expect_column_values_to_match_regex_list](#)

```
expect_column_values_to_match_regex_list (self, column, regex_list,
                                             match_on='any', mostly=None, re-
                                             sult_format=None, include_config=True,
                                             catch_exceptions=None, meta=None)
```

Expect the column entries to be strings that can be matched to either any of or all of a list of regular expressions. Matches can be anywhere in the string.

`expect_column_values_to_match_regex_list` is a [column_map_expectation](#).

Parameters

- **column** (*str*) – The column name.
- **regex_list** (*list*) – The list of regular expressions which the column entries should match

Keyword Arguments

- **match_on=** (*string*) – “any” or “all”. Use “any” if the value should match at least one regular expression in the list. Use “all” if it should match each regular expression in the list.
- **mostly** (*None or a float between 0 and 1*) – Return “success”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see [include_config](#).

- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[expect_column_values_to_match_regex](#)

[expect_column_values_to_not_match_regex](#)

```
expect_column_pair_values_to_be_equal(self, column_A, column_B, ignore_row_if='both_values_are_missing', result_format=None, include_config=True, catch_exceptions=None, meta=None)
```

Expect the values in column A to be the same as column B.

Parameters

- **column_A** (*str*) – The first column name
- **column_B** (*str*) – The second column name

Keyword Arguments **ignore_row_if** (*str*) – “both_values_are_missing”, “either_value_is_missing”, “neither”

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

```
expect_column_pair_values_A_to_be_greater_than_B(self, column_A, column_B, or_equal=None, parse_strings_as_datetimes=None, allow_cross_type_comparisons=None, ignore_row_if='both_values_are_missing', result_format=None, include_config=True, catch_exceptions=None, meta=None)
```

Expect values in column A to be greater than column B.

Parameters

- **column_A** (*str*) – The first column name
- **column_B** (*str*) – The second column name
- **or_equal** (*boolean or None*) – If True, then values can be equal, not strictly greater

Keyword Arguments

- **allow_cross_type_comparisons** (*boolean or None*) – If True, allow comparisons between types (e.g. integer and string). Otherwise, attempting such comparisons will raise an exception.
- **ignore_row_if** (*str*) – “both_values_are_missing”, “either_value_is_missing”, “neither

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

```
expect_multicolumn_values_to_be_unique(self, column_list, ignore_row_if='all_values_are_missing', result_format=None, include_config=True, catch_exceptions=None, meta=None)
```

Expect the values for each row to be unique across the columns listed.

Parameters **column_list** (*tuple or list*) – The first column name

Keyword Arguments **ignore_row_if** (*str*) – “all_values_are_missing”, “any_value_is_missing”, “never”

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

`great_expectations.dataset.sqlalchemy_dataset`

Module Contents

Classes

<code>SqlAlchemyBatchReference</code> (engine, table_name=None, schema=None, query=None)	
<code>MetaSqlAlchemyDataset</code> (*args, **kwargs)	Holds expectation decorators.
<code>SqlAlchemyDataset</code> (table_name=None, engine=None, connection_string=None, custom_sql=None, schema=None, *args, **kwargs)	Holds expectation decorators.

`great_expectations.dataset.sqlalchemy_dataset.logger`

`great_expectations.dataset.sqlalchemy_dataset.sqlalchemy_psycopg2`

`great_expectations.dataset.sqlalchemy_dataset.sqlalchemy_redshift`

`great_expectations.dataset.sqlalchemy_dataset.snowflake`

`great_expectations.dataset.sqlalchemy_dataset.bigquery_types_tuple`

class `great_expectations.dataset.sqlalchemy_dataset.SqlAlchemyBatchReference` (*engine, table_name=None, schema=None, query=None*)

Bases: `object`

get_init_kwargs (*self*)

class `great_expectations.dataset.sqlalchemy_dataset.MetaSqlAlchemyDataset` (*args, **kwargs)

Bases: `great_expectations.dataset.dataset.Dataset`

Holds expectation decorators.

classmethod `column_map_expectation` (*cls, func*)

For SQLAlchemy, this decorator allows individual column_map_expectations to simply return the filter that

describes the expected condition on their data.

The decorator will then use that filter to obtain unexpected elements, relevant counts, and return the formatted object.

```
class great_expectations.dataset.sqlalchemy_dataset.SqlAlchemyDataset (table_name=None,  
                                                                    en-  
                                                                    gine=None,  
                                                                    con-  
                                                                    nec-  
                                                                    tion_string=None,  
                                                                    cus-  
                                                                    tom_sql=None,  
                                                                    schema=None,  
                                                                    *args,  
                                                                    **kwargs)
```

Bases: `great_expectations.dataset.sqlalchemy_dataset.MetaSqlAlchemyDataset`

Holds expectation decorators.

```
classmethod from_dataset (cls, dataset=None)
```

This base implementation naively passes arguments on to the real constructor, which is suitable really when a constructor knows to take its own type. In general, this should be overridden

```
property sql_engine_dialect (self)
```

```
attempt_allowing_relative_error (self)
```

Subclasses can override this method if the respective data source (e.g., Redshift) supports “approximate” mode. In certain cases (e.g., for SparkDFDataset), a fraction between 0 and 1 (i.e., not only a boolean) is allowed.

```
head (self, n=5)
```

Returns a *PandasDataset* with the first *n* rows of the given Dataset

```
get_row_count (self)
```

Returns: int, table row count

```
get_column_count (self)
```

Returns: int, table column count

```
get_table_columns (self)
```

Returns: List[str], list of column names

```
get_column_nonnull_count (self, column)
```

Returns: int

```
get_column_sum (self, column)
```

Returns: float

```
get_column_max (self, column, parse_strings_as_datetimes=False)
```

Returns: any

```
get_column_min (self, column, parse_strings_as_datetimes=False)
```

Returns: any

```
get_column_value_counts (self, column, sort='value', collate=None)
```

Get a series containing the frequency counts of unique values from the named column.

Parameters

- **column** – the column for which to obtain value_counts

- **sort** (*string*) – must be one of “value”, “count”, or “none”. - if “value” then values in the resulting partition object will be sorted lexicographically - if “count” then values will be sorted according to descending count (frequency) - if “none” then values will not be sorted
- **collate** (*string*) – the collate (sort) method to be used on supported backends (SqlAlchemy only)

Returns `pd.Series` of value counts for a column, sorted according to the value requested in `sort`

get_column_mean (*self, column*)

Returns: float

get_column_unique_count (*self, column*)

Returns: int

get_column_median (*self, column*)

Returns: any

get_column_quantiles (*self, column: str, quantiles, allow_relative_error: bool = False*)

Get the values in column closest to the requested quantiles :param column: name of column :type column: string :param quantiles: the quantiles to return. quantiles *must* be a tuple to ensure caching is possible :type quantiles: tuple of float

Returns the nearest values in the dataset to those quantiles

Return type `List[any]`

get_column_stdev (*self, column*)

Returns: float

get_column_hist (*self, column, bins*)

return a list of counts corresponding to bins

Parameters

- **column** – the name of the column for which to get the histogram
- **bins** – tuple of bin edges for which to get histogram values; *must* be tuple to support caching

get_column_count_in_range (*self, column, min_val=None, max_val=None, strict_min=False, strict_max=True*)

Returns: int

create_temporary_table (*self, table_name, custom_sql, schema_name=None*)

Create Temporary table based on sql query. This will be used as a basis for executing expectations. WARNING: this feature is new in v0.4. It hasn't been tested in all SQL dialects, and may change based on community feedback. :param custom_sql:

column_reflection_fallback (*self*)

If we can't reflect the table, use a query to at least get column names.

expect_column_values_to_be_null (*self, column, mostly=None, result_format=None, include_config=True, catch_exceptions=None, meta=None*)

Expect column values to be null.

`expect_column_values_to_be_null` is a [column_map_expectation](#).

Parameters **column** (*str*) – The column name.

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return “success”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[expect_column_values_to_not_be_null](#)

```
expect_column_values_to_not_be_null(self, column, mostly=None, result_format=None,
                                     include_config=True, catch_exceptions=None,
                                     meta=None)
```

Expect column values to not be null.

To be counted as an exception, values must be explicitly null or missing, such as a NULL in PostgreSQL or an np.NaN in pandas. Empty strings don't count as null unless they have been coerced to a null type.

`expect_column_values_to_not_be_null` is a [column_map_expectation](#).

Parameters **column** (*str*) – The column name.

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return “success”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[expect_column_values_to_be_null](#)

```
_get_dialect_type_module(self)
```

```
expect_column_values_to_be_of_type(self, column, type_, mostly=None, re-  
sult_format=None, include_config=True,  
catch_exceptions=None, meta=None)
```

Expect a column to contain values of a specified data type.

`expect_column_values_to_be_of_type` is a [column_aggregate_expectation](#) for typed-column backends, and also for `PandasDataset` where the column dtype and provided `type_` are unambiguous constraints (any dtype except 'object' or dtype of 'object' with `type_` specified as 'object').

For `PandasDataset` columns with dtype of 'object' `expect_column_values_to_be_of_type` is a [column_map_expectation](#) and will independently check each row's type.

Parameters

- **column** (*str*) – The column name.
- **type_** (*str*) – A string representing the data type that each column should have as entries. Valid types are defined by the current backend implementation and are dynamically loaded. For example, valid types for `PandasDataset` include any numpy dtype values (such as 'int64') or native python types (such as 'int'), whereas valid types for a `SqlAlchemyDataset` include types named by the current driver such as 'INTEGER' in most SQL dialects and 'TEXT' in dialects such as postgresql. Valid types for `SparkDFDataset` include 'StringType', 'BooleanType' and other pyspark-defined type names.

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return “success”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[expect_column_values_to_be_in_type_list](#)

```
expect_column_values_to_be_in_type_list(self, column, type_list, mostly=None, re-  
sult_format=None, include_config=True,  
catch_exceptions=None, meta=None)
```

Expect a column to contain values from a specified type list.

`expect_column_values_to_be_in_type_list` is a [column_aggregate_expectation](#) for typed-column backends, and also for `PandasDataset` where the column dtype provides an unambiguous constraints (any dtype except 'object'). For `PandasDataset` columns with dtype of 'object' `expect_column_values_to_be_of_type` is a [column_map_expectation](#) and will independently check each row's type.

Parameters

- **column** (*str*) – The column name.
- **type_list** (*str*) – A list of strings representing the data type that each column should have as entries. Valid types are defined by the current backend implementation and are dynamically loaded. For example, valid types for `PandasDataset` include any numpy dtype values (such as `'int64'`) or native python types (such as `'int'`), whereas valid types for a `SqlAlchemyDataset` include types named by the current driver such as `'INTEGER'` in most SQL dialects and `'TEXT'` in dialects such as postgresql. Valid types for `SparkDFDataset` include `'StringType'`, `'BooleanType'` and other pyspark-defined type names.

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return “*success*”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: `BOOLEAN_ONLY`, `BASIC`, `COMPLETE`, or `SUMMARY`. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[expect_column_values_to_be_of_type](#)

```
expect_column_values_to_be_in_set (self, column, value_set, mostly=None,
                                     parse_strings_as_datetimes=None, re-
                                     sult_format=None, include_config=True,
                                     catch_exceptions=None, meta=None)
```

Expect each column value to be in a given set.

For example:

```
# my_df.my_col = [1, 2, 2, 3, 3, 3]
>>> my_df.expect_column_values_to_be_in_set(
    "my_col",
    [2, 3]
)
{
  "success": false
  "result": {
    "unexpected_count": 1
    "unexpected_percent": 16.666666666666666,
    "unexpected_percent_nonmissing": 16.666666666666666,
    "partial_unexpected_list": [
      1
    ]
  }
}
```

(continues on next page)

(continued from previous page)

```

    ],
  },
}

```

`expect_column_values_to_be_in_set` is a *column_map_expectation*.

Parameters

- **column** (*str*) – The column name.
- **value_set** (*set-like*) – A set of objects used for comparison.

Keyword Arguments

- **mostly** (*None or a float between 0 and 1*) – Return “success”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).
- **parse_strings_as_datetimes** (*boolean or None*) – If *True* values provided in `value_set` will be parsed as datetimes before making comparisons.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[expect_column_values_to_not_be_in_set](#)

```

expect_column_values_to_not_be_in_set(self, column, value_set, mostly=None,
                                         parse_strings_as_datetimes=None, re-
                                         sult_format=None, include_config=True,
                                         catch_exceptions=None, meta=None)

```

Expect column entries to not be in the set.

For example:

```

# my_df.my_col = [1, 2, 2, 3, 3, 3]
>>> my_df.expect_column_values_to_not_be_in_set(
    "my_col",
    [1, 2]
)
{
  "success": false
  "result": {
    "unexpected_count": 3
  }
}

```

(continues on next page)

(continued from previous page)

```

    "unexpected_percent": 50.0,
    "unexpected_percent_nonmissing": 50.0,
    "partial_unexpected_list": [
        1, 2, 2
    ],
    },
    },
)

```

`expect_column_values_to_not_be_in_set` is a *column_map_expectation*.

Parameters

- **column** (*str*) – The column name.
- **value_set** (*set-like*) – A set of objects used for comparison.

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return “*success*”: *True* if at least mostly fraction of values match the expectation. For more detail, see *mostly*.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see *result_format*.
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see *include_config*.
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see *catch_exceptions*.
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see *meta*.

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to *result_format* and *include_config*, *catch_exceptions*, and *meta*.

See also:

expect_column_values_to_be_in_set

expect_column_values_to_be_between (*self, column, min_value=None, max_value=None, strict_min=False, strict_max=False, allow_cross_type_comparisons=None, parse_strings_as_datetimes=None, output_strftime_format=None, mostly=None, result_format=None, include_config=True, catch_exceptions=None, meta=None*)

Expect column entries to be between a minimum value and a maximum value (inclusive).

`expect_column_values_to_be_between` is a *column_map_expectation*.

Parameters

- **column** (*str*) – The column name.
- **min_value** (*comparable type or None*) – The minimum value for a column entry.

- **max_value** (*comparable type or None*) – The maximum value for a column entry.

Keyword Arguments

- **strict_min** (*boolean*) – If True, values must be strictly larger than min_value, default=False
- **strict_max** (*boolean*) – If True, values must be strictly smaller than max_value, default=False
- **allow_cross_type_comparisons** (*boolean or None*) : If True, allow comparisons between types (e.g. integer and string). Otherwise, attempting such comparisons will raise an exception.
- **parse_strings_as_datetimes** (*boolean or None*) – If True, parse min_value, max_value, and all non-null column values to datetimes before making comparisons.
- **output_strftime_format** (*str or None*) – A valid strftime format for datetime output. Only used if parse_strings_as_datetimes=True.
- **mostly** (*None or a float between 0 and 1*) – Return “success”: True if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

Notes

- min_value and max_value are both inclusive unless strict_min or strict_max are set to True.
- If min_value is None, then max_value is treated as an upper bound, and there is no minimum value checked.
- If max_value is None, then min_value is treated as a lower bound, and there is no maximum value checked.

See also:

[expect_column_value_lengths_to_be_between](#)

```
expect_column_value_lengths_to_equal(self, column, value, mostly=None, result_format=None, include_config=True, catch_exceptions=None, meta=None)
```

Expect column entries to be strings with length equal to the provided value.

This expectation only works for string-type values. Invoking it on ints or floats will raise a `TypeError`.

`expect_column_values_to_be_between` is a *column_map_expectation*.

Parameters

- **column** (*str*) – The column name.
- **value** (*int or None*) – The expected value for a column entry length.

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return “*success*”: *True* if at least mostly fraction of values match the expectation. For more detail, see *mostly*.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see *result_format*.
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see *include_config*.
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see *catch_exceptions*.
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see *meta*.

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to *result_format* and *include_config*, *catch_exceptions*, and *meta*.

See also:

expect_column_value_lengths_to_be_between

```
expect_column_value_lengths_to_be_between(self, column, min_value=None,
                                           max_value=None, mostly=None, re-
                                           sult_format=None, include_config=True,
                                           catch_exceptions=None, meta=None)
```

Expect column entries to be strings with length between a minimum value and a maximum value (inclusive).

This expectation only works for string-type values. Invoking it on ints or floats will raise a `TypeError`.

`expect_column_value_lengths_to_be_between` is a *column_map_expectation*.

Parameters **column** (*str*) – The column name.

Keyword Arguments

- **min_value** (*int or None*) – The minimum value for a column entry length.
- **max_value** (*int or None*) – The maximum value for a column entry length.
- **mostly** (*None or a float between 0 and 1*) – Return “*success*”: *True* if at least mostly fraction of values match the expectation. For more detail, see *mostly*.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see *result_format*.

- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

Notes

- min_value and max_value are both inclusive.
- If min_value is None, then max_value is treated as an upper bound, and the number of acceptable rows has no minimum.
- If max_value is None, then min_value is treated as a lower bound, and the number of acceptable rows has no maximum.

See also:

[expect_column_value_lengths_to_equal](#)

expect_column_values_to_be_unique(*self*, *column*, *mostly=None*, *result_format=None*, *include_config=True*, *catch_exceptions=None*, *meta=None*)

Expect each column value to be unique.

This expectation detects duplicates. All duplicated values are counted as exceptions.

For example, `[1, 2, 3, 3, 3]` will return `[3, 3, 3]` in `result.exceptions_list`, with `unexpected_percent = 60.0`.

`expect_column_values_to_be_unique` is a [column_map_expectation](#).

Parameters `column` (*str*) – The column name.

Keyword Arguments `mostly` (*None or a float between 0 and 1*) – Return “success”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: `BOOLEAN_ONLY`, `BASIC`, `COMPLETE`, or `SUMMARY`. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to *result_format* and *include_config*, *catch_exceptions*, and *meta*.

`_get_dialect_regex_expression(self, column, regex, positive=True)`

`expect_column_values_to_match_regex(self, column, regex, mostly=None, result_format=None, include_config=True, catch_exceptions=None, meta=None)`

Expect column entries to be strings that match a given regular expression. Valid matches can be found anywhere in the string, for example “[at]+” will identify the following strings as expected: “cat”, “hat”, “aa”, “a”, and “t”, and the following strings as unexpected: “fish”, “dog”.

`expect_column_values_to_match_regex` is a *column_map_expectation*.

Parameters

- **column** (*str*) – The column name.
- **regex** (*str*) – The regular expression the column entries should match.

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return “success”: *True* if at least mostly fraction of values match the expectation. For more detail, see *mostly*.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see *result_format*.
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see *include_config*.
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see *catch_exceptions*.
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see *meta*.

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to *result_format* and *include_config*, *catch_exceptions*, and *meta*.

See also:

expect_column_values_to_not_match_regex

expect_column_values_to_match_regex_list

`expect_column_values_to_not_match_regex(self, column, regex, mostly=None, result_format=None, include_config=True, catch_exceptions=None, meta=None)`

Expect column entries to be strings that do NOT match a given regular expression. The regex must not match any portion of the provided string. For example, “[at]+” would identify the following strings as expected: “fish”, “dog”, and the following as unexpected: “cat”, “hat”.

`expect_column_values_to_not_match_regex` is a *column_map_expectation*.

Parameters

- **column** (*str*) – The column name.
- **regex** (*str*) – The regular expression the column entries should NOT match.

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return “*success*”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[expect_column_values_to_match_regex](#)

[expect_column_values_to_match_regex_list](#)

expect_column_values_to_match_regex_list (*self*, *column*, *regex_list*, *match_on='any'*, *mostly=None*, *result_format=None*, *include_config=True*, *catch_exceptions=None*, *meta=None*)

Expect the column entries to be strings that can be matched to either any of or all of a list of regular expressions. Matches can be anywhere in the string.

`expect_column_values_to_match_regex_list` is a [column_map_expectation](#).

Parameters

- **column** (*str*) – The column name.
- **regex_list** (*list*) – The list of regular expressions which the column entries should match

Keyword Arguments

- **match_on=** (*string*) – “any” or “all”. Use “any” if the value should match at least one regular expression in the list. Use “all” if it should match each regular expression in the list.
- **mostly** (*None or a float between 0 and 1*) – Return “*success*”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).

- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[expect_column_values_to_match_regex](#)

[expect_column_values_to_not_match_regex](#)

```
expect_column_values_to_not_match_regex_list(self, column, regex_list,
                                              mostly=None, result_format=None,
                                              include_config=True,
                                              catch_exceptions=None,
                                              meta=None)
```

Expect the column entries to be strings that do not match any of a list of regular expressions. Matches can be anywhere in the string.

`expect_column_values_to_not_match_regex_list` is a [column_map_expectation](#).

Parameters

- **column** (*str*) – The column name.
- **regex_list** (*list*) – The list of regular expressions which the column entries should not match

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return “*success*”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

`expect_column_values_to_match_regex_list`

`great_expectations.dataset.util`

Module Contents

Functions

<code>is_valid_partition_object(partition_object)</code>	Tests whether a given object is a valid continuous or categorical partition object.
<code>is_valid_categorical_partition_object(partition_object)</code>	Tests whether a given object is a valid categorical partition object.
<code>is_valid_continuous_partition_object(partition_object)</code>	Tests whether a given object is a valid continuous partition object. See Partition Objects .
<code>categorical_partition_data(data)</code>	Convenience method for creating weights from categorical data.
<code>kde_partition_data(data, estimate_tails=True)</code>	Convenience method for building a partition and weights using a gaussian Kernel Density Estimate and default bandwidth.
<code>partition_data(data, bins='auto', n_bins=10)</code>	
<code>continuous_partition_data(data, bins='auto', n_bins=10, **kwargs)</code>	Convenience method for building a partition object on continuous data
<code>build_continuous_partition_object(dataset, column, bins='auto', n_bins=10, allow_relative_error=False)</code>	Convenience method for building a partition object on continuous data from a dataset and column
<code>build_categorical_partition_object(dataset, column, sort='value')</code>	Convenience method for building a partition object on categorical data from a dataset and column
<code>infer_distribution_parameters(data, distribution, params=None)</code>	Convenience method for determining the shape parameters of a given distribution
<code>_scipy_distribution_positional_args_from_params(params)</code>	Helper function that returns positional arguments for a scipy distribution using a dict of parameters.
<code>validate_distribution_parameters(distribution, params)</code>	Ensures that necessary parameters for a distribution are present and that all parameters are sensical.
<code>create_multiple_expectations(df, columns, expectation_type, *args, **kwargs)</code>	Creates an identical expectation for each of the given columns with the specified arguments, if any.
<code>get_approximate_percentile_disc_sql(selects: List, sql_engine_dialect: Any)</code>	
<code>check_sql_engine_dialect(actual_sql_engine_dialect: Any, candidate_sql_engine_dialect: Any)</code>	

`great_expectations.dataset.util.logger`

`great_expectations.dataset.util.is_valid_partition_object(partition_object)`

Tests whether a given object is a valid continuous or categorical partition object. :param partition_object: The partition_object to evaluate :return: Boolean

`great_expectations.dataset.util.is_valid_categorical_partition_object(partition_object)`

Tests whether a given object is a valid categorical partition object. :param partition_object: The partition_object to evaluate :return: Boolean

`great_expectations.dataset.util.is_valid_continuous_partition_object(partition_object)`

Tests whether a given object is a valid continuous partition object. See [Partition Objects](#).

Parameters `partition_object` – The partition_object to evaluate

Returns Boolean

`great_expectations.dataset.util.categorical_partition_data(data)`

Convenience method for creating weights from categorical data.

Parameters `data` (*list-like*) – The data from which to construct the estimate.

Returns

A new partition object:

```
{
  "values": (list) The categorical values present in the data
  "weights": (list) The weights of the values in the partition.
}
```

See [Partition Objects](#).

`great_expectations.dataset.util.kde_partition_data(data, estimate_tails=True)`

Convenience method for building a partition and weights using a gaussian Kernel Density Estimate and default bandwidth.

Parameters

- **data** (*list-like*) – The data from which to construct the estimate
- **estimate_tails** (*bool*) – Whether to estimate the tails of the distribution to keep the partition object finite

Returns

A new partition_object:

```
{
  "bins": (list) The endpoints of the partial partition of reals,
  "weights": (list) The densities of the bins implied by the_
  ↪partition.
}

See :ref:`partition_object`.
```

`great_expectations.dataset.util.partition_data(data, bins='auto', n_bins=10)`

`great_expectations.dataset.util.continuous_partition_data(data, bins='auto', n_bins=10, **kwargs)`

Convenience method for building a partition object on continuous data

Parameters

- **data** (*list-like*) – The data from which to construct the estimate.
- **bins** (*string*) – One of ‘uniform’ (for uniformly spaced bins), ‘ntile’ (for percentile-spaced bins), or ‘auto’ (for automatically spaced bins)
- **n_bins** (*int*) – Ignored if bins is auto.
- **kwargs** (*mapping*) – Additional keyword arguments to be passed to numpy histogram

Returns

A new partition_object:

```
{
  "bins": (list) The endpoints of the partial partition of reals,
  "weights": (list) The densities of the bins implied by the_
  ↪partition.
}
See :ref:`partition_object`.
```

```
great_expectations.dataset.util.build_continuous_partition_object (dataset,
                                                                    column,
                                                                    bins='auto',
                                                                    n_bins=10,
                                                                    al-
                                                                    low_relative_error=False)
```

Convenience method for building a partition object on continuous data from a dataset and column

Parameters

- **dataset** (*GE Dataset*) – the dataset for which to compute the partition
- **column** (*string*) – The name of the column for which to construct the estimate.
- **bins** (*string*) – One of ‘uniform’ (for uniformly spaced bins), ‘ntile’ (for percentile-spaced bins), or ‘auto’ (for automatically spaced bins)
- **n_bins** (*int*) – Ignored if bins is auto.
- **allow_relative_error** – passed to `get_column_quantiles`, set to `False` for only precise values, `True` to allow approximate values on systems with only binary choice (e.g. Redshift), and to a value between zero and one for systems that allow specification of relative error (e.g. SparkDFDataset).

Returns

A new `partition_object`:

```
{
  "bins": (list) The endpoints of the partial partition of reals,
  "weights": (list) The densities of the bins implied by the_
  ↪partition.
}
See :ref:`partition_object`.
```

```
great_expectations.dataset.util.build_categorical_partition_object (dataset,
                                                                    column,
                                                                    sort='value')
```

Convenience method for building a partition object on categorical data from a dataset and column

Parameters

- **dataset** (*GE Dataset*) – the dataset for which to compute the partition
- **column** (*string*) – The name of the column for which to construct the estimate.
- **sort** (*string*) – must be one of “value”, “count”, or “none”. - if “value” then values in the resulting partition object will be sorted lexicographically - if “count” then values will be sorted according to descending count (frequency) - if “none” then values will not be sorted

Returns

A new `partition_object`:

```
{
    "values": (list) the categorical values for which each weight
    ↪applies,
    "weights": (list) The densities of the values implied by the
    ↪partition.
}
See :ref:`partition_object`.
```

`great_expectations.dataset.util.infer_distribution_parameters` (*data*, *distribution*,
params=None)

Convenience method for determining the shape parameters of a given distribution

Parameters

- **data** (*list-like*) – The data to build shape parameters from.
- **distribution** (*string*) – Scipy distribution, determines which parameters to build.
- **params** (*dict or None*) – The known parameters. Parameters given here will not be altered. Keep as None to infer all necessary parameters from the data data.

Returns

A dictionary of named parameters:

```
{
    "mean": (float),
    "std_dev": (float),
    "loc": (float),
    "scale": (float),
    "alpha": (float),
    "beta": (float),
    "min": (float),
    "max": (float),
    "df": (float)
}

See: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.kstest.html#scipy.stats.kstest
```

`great_expectations.dataset.util._scipy_distribution_positional_args_from_dict` (*distribution*,
params)

Helper function that returns positional arguments for a scipy distribution using a dict of parameters.

See the `cdf()` function here <https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.beta.html#Methods> to see an example of scipy's positional arguments. This function returns the arguments specified by the `scipy.stat.distribution.cdf()` for the distribution.

Parameters

- **distribution** (*string*) – The scipy distribution name.
- **params** (*dict*) – A dict of named parameters.

Raises `AttributeError` – If an unsupported distribution is provided.

`great_expectations.dataset.util.validate_distribution_parameters` (*distribution*,
params)

Ensures that necessary parameters for a distribution are present and that all parameters are sensical.

If parameters necessary to construct a distribution are missing or invalid, this function raises `ValueError` with an informative description. Note that 'loc' and 'scale' are optional arguments, and that 'scale' must be positive.

Parameters

- **distribution** (*string*) – The scipy distribution name, e.g. normal distribution is ‘norm’.
- **params** (*dict or list*) – The distribution shape parameters in a named dictionary or positional list form following the scipy cdf argument scheme.

params={‘mean’: 40, ‘std_dev’: 5} or params=[40, 5]

Exceptions: `ValueError`: With an informative description, usually when necessary parameters are omitted or are invalid.

`great_expectations.dataset.util.create_multiple_expectations` (*df, columns, expectation_type, *args, **kwargs*)

Creates an identical expectation for each of the given columns with the specified arguments, if any.

Parameters

- **df** (*great_expectations.dataset*) – A great expectations dataset object.
- **columns** (*list*) – A list of column names represented as strings.
- **expectation_type** (*string*) – The expectation type.

Raises

- **KeyError** if the provided column does not exist. –
- **AttributeError** if the provided expectation type does not exist or df is not a valid great expectations dataset. –

Returns A list of expectation results.

`great_expectations.dataset.util.get_approximate_percentile_disc_sql` (*selects: List, sql_engine_dialect: Any*) → *str*

`great_expectations.dataset.util.check_sql_engine_dialect` (*actual_sql_engine_dialect: Any, candidate_sql_engine_dialect: Any*) → *bool*

Package Contents**Classes**

<code>Dataset(*args, **kwargs)</code>	Holds expectation decorators.
<code>MetaPandasDataset(*args, **kwargs)</code>	MetaPandasDataset is a thin layer between Dataset and PandasDataset.
<code>PandasDataset(*args, **kwargs)</code>	PandasDataset instantiates the great_expectations Expectations API as a subclass of a pandas.DataFrame.

class `great_expectations.dataset.Dataset` (**args, **kwargs*)
Bases: `great_expectations.dataset.dataset.MetaDataset`

Holds expectation decorators.

_data_asset_type = Dataset

hashable_getters = ['get_column_min', 'get_column_max', 'get_column_mean', 'get_column

classmethod from_dataset (cls, dataset=None)

This base implementation naively passes arguments on to the real constructor, which is suitable really when a constructor knows to take its own type. In general, this should be overridden

abstract get_row_count (self)

Returns: int, table row count

abstract get_column_count (self)

Returns: int, table column count

abstract get_table_columns (self)

Returns: List[str], list of column names

abstract get_column_nonnull_count (self, column)

Returns: int

abstract get_column_mean (self, column)

Returns: float

abstract get_column_value_counts (self, column, sort='value', collate=None)

Get a series containing the frequency counts of unique values from the named column.

Parameters

- **column** – the column for which to obtain value_counts
- **sort** (*string*) – must be one of “value”, “count”, or “none”. - if “value” then values in the resulting partition object will be sorted lexicographically - if “count” then values will be sorted according to descending count (frequency) - if “none” then values will not be sorted
- **collate** (*string*) – the collate (sort) method to be used on supported backends (SqlAlchemy only)

Returns pd.Series of value counts for a column, sorted according to the value requested in sort

abstract get_column_sum (self, column)

Returns: float

abstract get_column_max (self, column, parse_strings_as_datetimes=False)

Returns: any

abstract get_column_min (self, column, parse_strings_as_datetimes=False)

Returns: any

abstract get_column_unique_count (self, column)

Returns: int

abstract get_column_modes (self, column)

Returns: List[any], list of modes (ties OK)

abstract get_column_median (self, column)

Returns: any

abstract get_column_quantiles (self, column, quantiles, allow_relative_error=False)

Get the values in column closest to the requested quantiles :param column: name of column :type column: string :param quantiles: the quantiles to return. quantiles *must* be a tuple to ensure caching is possible :type quantiles: tuple of float

Returns the nearest values in the dataset to those quantiles

Return type List[any]

abstract get_column_stdev (*self*, *column*)

Returns: float

get_column_partition (*self*, *column*, *bins*='uniform', *n_bins*=10, *allow_relative_error*=False)

Get a partition of the range of values in the specified column.

Parameters

- **column** – the name of the column
- **bins** – ‘uniform’ for evenly spaced bins or ‘quantile’ for bins spaced according to quantiles
- **n_bins** – the number of bins to produce
- **allow_relative_error** – passed to get_column_quantiles, set to False for only precise values, True to allow approximate values on systems with only binary choice (e.g. Redshift), and to a value between zero and one for systems that allow specification of relative error (e.g. SparkDFDataset).

Returns A list of bins

abstract get_column_hist (*self*, *column*, *bins*)

Get a histogram of column values :param column: the column for which to generate the histogram :param bins: the bins to slice the histogram. bins *must* be a tuple to ensure caching is possible :type bins: tuple

Returns: List[int], a list of counts corresponding to bins

abstract get_column_count_in_range (*self*, *column*, *min_val*=None, *max_val*=None, *strict_min*=False, *strict_max*=True)

Returns: int

test_column_map_expectation_function (*self*, *function*, **args*, ***kwargs*)

Test a column map expectation function

Parameters

- **function** (*func*) – The function to be tested. (Must be a valid column_map_expectation function.)
- ***args** – Positional arguments to be passed the the function
- ****kwargs** – Keyword arguments to be passed the the function

Returns A JSON-serializable expectation result object.

Notes

This function is a thin layer to allow quick testing of new expectation functions, without having to define custom classes, etc. To use developed expectations from the command-line tool, you’ll still need to define custom classes, etc.

Check out custom_expectations_reference for more information.

test_column_aggregate_expectation_function (*self*, *function*, **args*, ***kwargs*)

Test a column aggregate expectation function

Parameters

- **function** (*func*) – The function to be tested. (Must be a valid column_aggregate_expectation function.)

- ***args** – Positional arguments to be passed the the function
- ****kwargs** – Keyword arguments to be passed the the function

Returns A JSON-serializable expectation result object.

Notes

This function is a thin layer to allow quick testing of new expectation functions, without having to define custom classes, etc. To use developed expectations from the command-line tool, you'll still need to define custom classes, etc.

Check out `custom_expectations_reference` for more information.

expect_column_to_exist (*self*, *column*, *column_index=None*, *result_format=None*, *include_config=True*, *catch_exceptions=None*, *meta=None*)

Expect the specified column to exist.

`expect_column_to_exist` is a *expectation*, not a *column_map_expectation* or *column_aggregate_expectation*.

Parameters *column* (*str*) – The column name.

Other Parameters

- **column_index** (*int or None*) – If not None, checks the order of the columns. The expectation will fail if the column is not in location `column_index` (zero-indexed).
- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see *result_format*.
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see *include_config*.
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see *catch_exceptions*.
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see *meta*.

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to *result_format* and *include_config*, *catch_exceptions*, and *meta*.

expect_table_columns_to_match_ordered_list (*self*, *column_list*, *result_format=None*, *include_config=True*, *catch_exceptions=None*, *meta=None*)

Expect the columns to exactly match a specified list.

`expect_table_columns_to_match_ordered_list` is a *expectation*, not a *column_map_expectation* or *column_aggregate_expectation*.

Parameters *column_list* (*list of str*) – The column names, in the correct order.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see *result_format*.
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see *include_config*.

- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

expect_table_column_count_to_be_between (*self, min_value=None, max_value=None, result_format=None, include_config=True, catch_exceptions=None, meta=None*)

Expect the number of columns to be between two values.

`expect_table_column_count_to_be_between` is a [expectation](#), not a `column_map_expectation` or `column_aggregate_expectation`.

Keyword Arguments

- **min_value** (*int or None*) – The minimum number of columns, inclusive.
- **max_value** (*int or None*) – The maximum number of columns, inclusive.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: `BOOLEAN_ONLY`, `BASIC`, `COMPLETE`, or `SUMMARY`. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

Notes

- `min_value` and `max_value` are both inclusive.
- If `min_value` is `None`, then `max_value` is treated as an upper bound, and the number of acceptable columns has no minimum.
- If `max_value` is `None`, then `min_value` is treated as a lower bound, and the number of acceptable columns has no maximum.

See also:

`expect_table_column_count_to_equal`

```
expect_table_column_count_to_equal (self, value, result_format=None, include_config=True, catch_exceptions=None, meta=None)
```

Expect the number of columns to equal a value.

`expect_table_column_count_to_equal` is a [expectation](#), not a `column_map_expectation` or `column_aggregate_expectation`.

Parameters `value` (*int*) – The expected number of columns.

Other Parameters

- **result_format** (*string or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

`expect_table_column_count_to_be_between`

```
expect_table_row_count_to_be_between (self, min_value=None, max_value=None, result_format=None, include_config=True, catch_exceptions=None, meta=None)
```

Expect the number of rows to be between two values.

`expect_table_row_count_to_be_between` is a [expectation](#), not a `column_map_expectation` or `column_aggregate_expectation`.

Keyword Arguments

- **min_value** (*int or None*) – The minimum number of rows, inclusive.
- **max_value** (*int or None*) – The maximum number of rows, inclusive.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to *result_format* and *include_config*, *catch_exceptions*, and *meta*.

Notes

- *min_value* and *max_value* are both inclusive.
- If *min_value* is *None*, then *max_value* is treated as an upper bound, and the number of acceptable rows has no minimum.
- If *max_value* is *None*, then *min_value* is treated as a lower bound, and the number of acceptable rows has no maximum.

See also:

`expect_table_row_count_to_equal`

`expect_table_row_count_to_equal` (*self*, *value*, *result_format=None*, *include_config=True*,
catch_exceptions=None, *meta=None*)

Expect the number of rows to equal a value.

`expect_table_row_count_to_equal` is a *expectation*, not a *column_map_expectation* or *column_aggregate_expectation*.

Parameters *value* (*int*) – The expected number of rows.

Other Parameters

- ***result_format*** (*string or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see *result_format*.
- ***include_config*** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see *include_config*.
- ***catch_exceptions*** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see *catch_exceptions*.
- ***meta*** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see *meta*.

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to *result_format* and *include_config*, *catch_exceptions*, and *meta*.

See also:

`expect_table_row_count_to_be_between`

`abstract expect_column_values_to_be_unique` (*self*, *column*, *mostly=None*,
result_format=None, *in-*
clude_config=True,
catch_exceptions=None, *meta=None*)

Expect each column value to be unique.

This expectation detects duplicates. All duplicated values are counted as exceptions.

For example, `[1, 2, 3, 3, 3]` will return `[3, 3, 3]` in `result.exceptions_list`, with `unexpected_percent = 60.0`.

`expect_column_values_to_be_unique` is a *column_map_expectation*.

Parameters *column* (*str*) – The column name.

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return “*success*”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

```
abstract expect_column_values_to_not_be_null (self, column, mostly=None,
                                              result_format=None, include_config=True,
                                              catch_exceptions=None, meta=None)
```

Expect column values to not be null.

To be counted as an exception, values must be explicitly null or missing, such as a *NULL* in PostgreSQL or an *np.NaN* in pandas. Empty strings don’t count as null unless they have been coerced to a null type.

`expect_column_values_to_not_be_null` is a [column_map_expectation](#).

Parameters **column** (*str*) – The column name.

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return “*success*”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[`expect_column_values_to_be_null`](#)

```
abstract expect_column_values_to_be_null(self, column, mostly=None, re-
                                         sult_format=None, include_config=True,
                                         catch_exceptions=None, meta=None)
```

Expect column values to be null.

`expect_column_values_to_be_null` is a [`column_map_expectation`](#).

Parameters `column` (*str*) – The column name.

Keyword Arguments `mostly` (*None or a float between 0 and 1*) – Return “success”: *True* if at least mostly fraction of values match the expectation. For more detail, see [`mostly`](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [`result_format`](#).
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see [`include_config`](#).
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see [`catch_exceptions`](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [`meta`](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [`result_format`](#) and [`include_config`](#), [`catch_exceptions`](#), and [`meta`](#).

See also:

[`expect_column_values_to_not_be_null`](#)

```
abstract expect_column_values_to_be_of_type(self, column, type_,
                                             mostly=None, result_format=None,
                                             include_config=True,
                                             catch_exceptions=None, meta=None)
```

Expect a column to contain values of a specified data type.

`expect_column_values_to_be_of_type` is a [`column_aggregate_expectation`](#) for typed-column backends, and also for `PandasDataset` where the column dtype and provided **type_** are unambiguous constraints (any dtype except ‘object’ or dtype of ‘object’ with **type_** specified as ‘object’).

For `PandasDataset` columns with dtype of ‘object’ `expect_column_values_to_be_of_type` is a [`column_map_expectation`](#) and will independently check each row’s type.

Parameters

- **column** (*str*) – The column name.
- **type_** (*str*) – A string representing the data type that each column should have as entries. Valid types are defined by the current backend implementation and are dynamically loaded. For example, valid types for `PandasDataset` include any numpy dtype values (such

as ‘int64’) or native python types (such as ‘int’), whereas valid types for a SQLAlchemy-Dataset include types named by the current driver such as ‘INTEGER’ in most SQL dialects and ‘TEXT’ in dialects such as postgresql. Valid types for SparkDFDataset include ‘StringType’, ‘BooleanType’ and other pyspark-defined type names.

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return “*success*”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[expect_column_values_to_be_in_type_list](#)

```
abstract expect_column_values_to_be_in_type_list (self, column, type_list,
                                                    mostly=None, re-
                                                    sult_format=None, in-
                                                    clude_config=True,
                                                    catch_exceptions=None,
                                                    meta=None)
```

Expect a column to contain values from a specified type list.

`expect_column_values_to_be_in_type_list` is a [column_aggregate_expectation](#) for typed-column backends, and also for `PandasDataset` where the column dtype provides an unambiguous constraints (any dtype except ‘object’). For `PandasDataset` columns with dtype of ‘object’ `expect_column_values_to_be_of_type` is a [column_map_expectation](#) and will independently check each row’s type.

Parameters

- **column** (*str*) – The column name.
- **type_list** (*str*) – A list of strings representing the data type that each column should have as entries. Valid types are defined by the current backend implementation and are dynamically loaded. For example, valid types for `PandasDataset` include any numpy dtype values (such as ‘int64’) or native python types (such as ‘int’), whereas valid types for a `SQLAlchemyDataset` include types named by the current driver such as ‘INTEGER’ in most SQL dialects and ‘TEXT’ in dialects such as postgresql. Valid types for `SparkDFDataset` include ‘StringType’, ‘BooleanType’ and other pyspark-defined type names.

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return “*success*”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[expect_column_values_to_be_of_type](#)

abstract expect_column_values_to_be_in_set (*self, column, value_set, mostly=None, parse_strings_as_datetimes=None, result_format=None, include_config=True, catch_exceptions=None, meta=None*)

Expect each column value to be in a given set.

For example:

```
# my_df.my_col = [1,2,2,3,3,3]
>>> my_df.expect_column_values_to_be_in_set(
    "my_col",
    [2,3]
)
{
  "success": false
  "result": {
    "unexpected_count": 1
    "unexpected_percent": 16.666666666666666,
    "unexpected_percent_nonmissing": 16.666666666666666,
    "partial_unexpected_list": [
      1
    ],
  },
}
```

`expect_column_values_to_be_in_set` is a [column_map_expectation](#).

Parameters

- **column** (*str*) – The column name.
- **value_set** (*set-like*) – A set of objects used for comparison.

Keyword Arguments

- **mostly** (*None or a float between 0 and 1*) – Return “*success*”: True if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

- **parse_strings_as_datetimes** (*boolean or None*) – If True values provided in `value_set` will be parsed as datetimes before making comparisons.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: `BOOLEAN_ONLY`, `BASIC`, `COMPLETE`, or `SUMMARY`. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[expect_column_values_to_not_be_in_set](#)

```
abstract expect_column_values_to_not_be_in_set (self, column, value_set,
                                                mostly=None,
                                                parse_strings_as_datetimes=None,
                                                result_format=None,
                                                include_config=True,
                                                catch_exceptions=None,
                                                meta=None)
```

Expect column entries to not be in the set.

For example:

```
# my_df.my_col = [1,2,2,3,3,3]
>>> my_df.expect_column_values_to_not_be_in_set(
    "my_col",
    [1,2]
)
{
  "success": false
  "result": {
    "unexpected_count": 3
    "unexpected_percent": 50.0,
    "unexpected_percent_nonmissing": 50.0,
    "partial_unexpected_list": [
      1, 2, 2
    ],
  },
}
```

`expect_column_values_to_not_be_in_set` is a [column_map_expectation](#).

Parameters

- **column** (*str*) – The column name.
- **value_set** (*set-like*) – A set of objects used for comparison.

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return “success”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[expect_column_values_to_be_in_set](#)

```
abstract expect_column_values_to_be_between(self, column, min_value=None,
                                             max_value=None, strict_min=False,
                                             strict_max=False, allow_cross_type_comparisons=None,
                                             parse_strings_as_datetimes=False,
                                             output_strftime_format=None,
                                             mostly=None, result_format=None,
                                             include_config=True,
                                             catch_exceptions=None, meta=None)
```

Expect column entries to be between a minimum value and a maximum value (inclusive).

`expect_column_values_to_be_between` is a [column_map_expectation](#).

Parameters

- **column** (*str*) – The column name.
- **min_value** (*comparable type or None*) – The minimum value for a column entry.
- **max_value** (*comparable type or None*) – The maximum value for a column entry.

Keyword Arguments

- **strict_min** (*boolean*) – If *True*, values must be strictly larger than `min_value`, default=*False*
- **strict_max** (*boolean*) – If *True*, values must be strictly smaller than `max_value`, default=*False*
- **allow_cross_type_comparisons** (*boolean or None*) : If *True*, allow comparisons between types (e.g. integer and string). Otherwise, attempting such comparisons will raise an exception.

- **parse_strings_as_datetimes** (*boolean or None*) – If True, parse min_value, max_value, and all non-null column values to datetimes before making comparisons.
- **output_strftime_format** (*str or None*) – A valid strftime format for datetime output. Only used if parse_strings_as_datetimes=True.
- **mostly** (*None or a float between 0 and 1*) – Return “success”: True if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

Notes

- min_value and max_value are both inclusive unless strict_min or strict_max are set to True.
- If min_value is None, then max_value is treated as an upper bound, and there is no minimum value checked.
- If max_value is None, then min_value is treated as a lower bound, and there is no maximum value checked.

See also:

[expect_column_value_lengths_to_be_between](#)

```
abstract expect_column_values_to_be_increasing(self, column, strictly=None,
                                                parse_strings_as_datetimes=False,
                                                mostly=None,                      re-
                                                sult_format=None,                  in-
                                                clude_config=True,
                                                catch_exceptions=None,
                                                meta=None)
```

Expect column values to be increasing.

By default, this expectation only works for numeric or datetime data. When *parse_strings_as_datetimes=True*, it can also parse strings to datetimes.

If *strictly=True*, then this expectation is only satisfied if each consecutive value is strictly increasing—equal values are treated as failures.

`expect_column_values_to_be_increasing` is a [column_map_expectation](#).

Parameters `column` (*str*) – The column name.

Keyword Arguments

- **strictly** (*Boolean or None*) – If True, values must be strictly greater than previous values
- **parse_strings_as_datetimes** (*boolean or None*) – If True, all non-null column values to datetimes before making comparisons
- **mostly** (*None or a float between 0 and 1*) – Return “success”: True if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[expect_column_values_to_be_decreasing](#)

```
abstract expect_column_values_to_be_decreasing(self, column, strictly=None,
                                                parse_strings_as_datetimes=False,
                                                mostly=None, result_format=None,
                                                include_config=True,
                                                catch_exceptions=None,
                                                meta=None)
```

Expect column values to be decreasing.

By default, this expectation only works for numeric or datetime data. When `parse_strings_as_datetimes=True`, it can also parse strings to datetimes.

If `strictly=True`, then this expectation is only satisfied if each consecutive value is strictly decreasing—equal values are treated as failures.

`expect_column_values_to_be_decreasing` is a [column_map_expectation](#).

Parameters `column` (*str*) – The column name.

Keyword Arguments

- **strictly** (*Boolean or None*) – If True, values must be strictly greater than previous values
- **parse_strings_as_datetimes** (*boolean or None*) – If True, all non-null column values to datetimes before making comparisons

- **mostly** (*None or a float between 0 and 1*) – Return “success”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[expect_column_values_to_be_increasing](#)

```
abstract expect_column_value_lengths_to_be_between(self, column,
                                                    min_value=None,
                                                    max_value=None,
                                                    mostly=None,           re-
                                                    result_format=None,       in-
                                                    include_config=True,
                                                    catch_exceptions=None,
                                                    meta=None)
```

Expect column entries to be strings with length between a minimum value and a maximum value (inclusive).

This expectation only works for string-type values. Invoking it on ints or floats will raise a `TypeError`.

`expect_column_value_lengths_to_be_between` is a [column_map_expectation](#).

Parameters `column` (*str*) – The column name.

Keyword Arguments

- **min_value** (*int or None*) – The minimum value for a column entry length.
- **max_value** (*int or None*) – The maximum value for a column entry length.
- **mostly** (*None or a float between 0 and 1*) – Return “success”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).

- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

Notes

- min_value and max_value are both inclusive.
- If min_value is None, then max_value is treated as an upper bound, and the number of acceptable rows has no minimum.
- If max_value is None, then min_value is treated as a lower bound, and the number of acceptable rows has no maximum.

See also:

[expect_column_value_lengths_to_equal](#)

```
abstract expect_column_value_lengths_to_equal (self, column, value,
                                              mostly=None, result_format=None,
                                              include_config=True,
                                              catch_exceptions=None,
                                              meta=None)
```

Expect column entries to be strings with length equal to the provided value.

This expectation only works for string-type values. Invoking it on ints or floats will raise a `TypeError`.

`expect_column_values_to_be_between` is a [column_map_expectation](#).

Parameters

- **column** (*str*) – The column name.
- **value** (*int or None*) – The expected value for a column entry length.

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return “success”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to *result_format* and *include_config*, *catch_exceptions*, and *meta*.

See also:

expect_column_value_lengths_to_be_between

```
abstract expect_column_values_to_match_regex(self,          column,          regex,
                                              mostly=None,    result_format=None,
                                              include_config=True,
                                              catch_exceptions=None,
                                              meta=None)
```

Expect column entries to be strings that match a given regular expression. Valid matches can be found anywhere in the string, for example “[at]+” will identify the following strings as expected: “cat”, “hat”, “aa”, “a”, and “t”, and the following strings as unexpected: “fish”, “dog”.

expect_column_values_to_match_regex is a *column_map_expectation*.

Parameters

- **column** (*str*) – The column name.
- **regex** (*str*) – The regular expression the column entries should match.

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return “*success*”: *True* if at least mostly fraction of values match the expectation. For more detail, see *mostly*.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see *result_format*.
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see *include_config*.
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see *catch_exceptions*.
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see *meta*.

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to *result_format* and *include_config*, *catch_exceptions*, and *meta*.

See also:

expect_column_values_to_not_match_regex

expect_column_values_to_match_regex_list

```
abstract expect_column_values_to_not_match_regex(self,          column,          regex,
                                                  mostly=None,          re-
                                                  sult_format=None,        in-
                                                  clude_config=True,
                                                  catch_exceptions=None,
                                                  meta=None)
```

Expect column entries to be strings that do NOT match a given regular expression. The regex must not match any portion of the provided string. For example, “[at]+” would identify the following strings as expected: “fish”, “dog”, and the following as unexpected: “cat”, “hat”.

`expect_column_values_to_not_match_regex` is a *column_map_expectation*.

Parameters

- **column** (*str*) – The column name.
- **regex** (*str*) – The regular expression the column entries should NOT match.

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return “success”: *True* if at least mostly fraction of values match the expectation. For more detail, see *mostly*.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see *result_format*.
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see *include_config*.
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see *catch_exceptions*.
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see *meta*.

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to *result_format* and *include_config*, *catch_exceptions*, and *meta*.

See also:

expect_column_values_to_match_regex

expect_column_values_to_match_regex_list

```
abstract expect_column_values_to_match_regex_list (self, column, regex_list,
                                                    match_on='any',
                                                    mostly=None,           re-
                                                    result_format=None,       in-
                                                    include_config=True,
                                                    catch_exceptions=None,
                                                    meta=None)
```

Expect the column entries to be strings that can be matched to either any of or all of a list of regular expressions. Matches can be anywhere in the string.

`expect_column_values_to_match_regex_list` is a *column_map_expectation*.

Parameters

- **column** (*str*) – The column name.
- **regex_list** (*list*) – The list of regular expressions which the column entries should match

Keyword Arguments

- **match_on=** (*string*) – “any” or “all”. Use “any” if the value should match at least one regular expression in the list. Use “all” if it should match each regular expression in the list.
- **mostly** (*None or a float between 0 and 1*) – Return “success”: *True* if at least mostly fraction of values match the expectation. For more detail, see *mostly*.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[expect_column_values_to_match_regex](#)

[expect_column_values_to_not_match_regex](#)

```
abstract expect_column_values_to_not_match_regex_list(self, column, regex_list,
                                                    mostly=None, re-
                                                    sult_format=None,
                                                    include_config=True,
                                                    catch_exceptions=None,
                                                    meta=None)
```

Expect the column entries to be strings that do not match any of a list of regular expressions. Matches can be anywhere in the string.

`expect_column_values_to_not_match_regex_list` is a [column_map_expectation](#).

Parameters

- **column** (*str*) – The column name.
- **regex_list** (*list*) – The list of regular expressions which the column entries should not match

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return “success”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to *result_format* and *include_config*, *catch_exceptions*, and *meta*.

See also:

expect_column_values_to_match_regex_list

```
abstract expect_column_values_to_match_strftime_format (self,          column,
                                                         strftime_format,
                                                         mostly=None,      re-
                                                         sult_format=None,
                                                         include_config=True,
                                                         catch_exceptions=None,
                                                         meta=None)
```

Expect column entries to be strings representing a date or time with a given format.

expect_column_values_to_match_strftime_format is a *column_map_expectation*.

Parameters

- **column** (*str*) – The column name.
- **strftime_format** (*str*) – A strftime format string to use for matching

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return “*success*”: *True* if at least mostly fraction of values match the expectation. For more detail, see *mostly*.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see *result_format*.
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see *include_config*.
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see *catch_exceptions*.
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see *meta*.

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to *result_format* and *include_config*, *catch_exceptions*, and *meta*.

```
abstract expect_column_values_to_be_dateutil_parseable (self,          column,
                                                         mostly=None,      re-
                                                         sult_format=None,
                                                         include_config=True,
                                                         catch_exceptions=None,
                                                         meta=None)
```

Expect column entries to be parsable using dateutil.

expect_column_values_to_be_dateutil_parseable is a *column_map_expectation*.

Parameters **column** (*str*) – The column name.

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return “*success*”: *True* if at least mostly fraction of values match the expectation. For more detail, see *mostly*.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

```
abstract expect_column_values_to_be_json_parseable(self, column, mostly=None,
                                                    result_format=None,
                                                    include_config=True,
                                                    catch_exceptions=None,
                                                    meta=None)
```

Expect column entries to be data written in JavaScript Object Notation.

`expect_column_values_to_be_json_parseable` is a [column_map_expectation](#).

Parameters `column` (*str*) – The column name.

Keyword Arguments `mostly` (*None or a float between 0 and 1*) – Return “success”: True if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[expect_column_values_to_match_json_schema](#)

```
abstract expect_column_values_to_match_json_schema(self, column, json_schema,
                                                    mostly=None,          re-
                                                    sult_format=None,      in-
                                                    clude_config=True,
                                                    catch_exceptions=None,
                                                    meta=None)
```

Expect column entries to be JSON objects matching a given JSON schema.

`expect_column_values_to_match_json_schema` is a [column_map_expectation](#).

Parameters `column` (*str*) – The column name.

Keyword Arguments `mostly` (*None or a float between 0 and 1*) – Return “*success*”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[expect_column_values_to_be_json_parseable](#)

The JSON-schema docs.

```
abstract expect_column_parameterized_distribution_ks_test_p_value_to_be_greater_than(self, column, distribution, p_value)
```

Expect the column values to be distributed similarly to a scipy distribution. This expectation compares the provided column to the specified continuous distribution with a parametric Kolmogorov-Smirnov test. The K-S test compares the provided column to the cumulative density function (CDF)

of the specified scipy distribution. If you don't know the desired distribution shape parameters, use the `ge.dataset.util.infer_distribution_parameters()` utility function to estimate them.

It returns 'success'=True if the p-value from the K-S test is greater than or equal to the provided p-value.

`expect_column_parameterized_distribution_ks_test_p_value_to_be_greater_than` is a *column_aggregate_expectation*.

Parameters

- **column** (*str*) – The column name.
- **distribution** (*str*) – The scipy distribution name. See: <https://docs.scipy.org/doc/scipy/reference/stats.html> Currently supported distributions are listed in the Notes section below.
- **p_value** (*float*) – The threshold p-value for a passing test. Default is 0.05.
- **params** (*dict or list*) – A dictionary or positional list of shape parameters that describe the distribution you want to test the data against. Include key values specific to the distribution from the appropriate scipy distribution CDF function. 'loc' and 'scale' are used as translational parameters. See <https://docs.scipy.org/doc/scipy/reference/stats.html#continuous-distributions>

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see *result_format*.
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see *include_config*.
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see *catch_exceptions*.
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see *meta*.

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to *result_format* and *include_config*, *catch_exceptions*, and *meta*.

Notes

These fields in the result object are customized for this expectation:

```
{
  "details": {
    "expected_params" (dict): The specified or inferred parameters of the
    ↪ distribution to test against
    "ks_results" (dict): The raw result of stats.kstest()
  }
}
```

- The Kolmogorov-Smirnov test's null hypothesis is that the column is similar to the provided distribution.
- Supported scipy distributions:
 - norm

- beta
- gamma
- uniform
- chi2
- expon

```
expect_column_distinct_values_to_be_in_set (self, column, value_set,
                                              parse_strings_as_datetimes=None,
                                              result_format=None, include_config=True,
                                              catch_exceptions=None, meta=None)
```

Expect the set of distinct column values to be contained by a given set.

The success value for this expectation will match that of `expect_column_values_to_be_in_set`. However, `expect_column_distinct_values_to_be_in_set` is a *column_aggregate_expectation*.

For example:

```
# my_df.my_col = [1,2,2,3,3,3]
>>> my_df.expect_column_distinct_values_to_be_in_set(
    "my_col",
    [2, 3, 4]
)
{
  "success": false
  "result": {
    "observed_value": [1,2,3],
    "details": {
      "value_counts": [
        {
          "value": 1,
          "count": 1
        },
        {
          "value": 2,
          "count": 1
        },
        {
          "value": 3,
          "count": 1
        }
      ]
    }
  }
}
```

Parameters

- **column** (*str*) – The column name.
- **value_set** (*set-like*) – A set of objects used for comparison.

Keyword Arguments **parse_strings_as_datetimes** (*boolean or None*) – If True values provided in `value_set` will be parsed as datetimes before making comparisons.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[expect_column_distinct_values_to_contain_set](#)

```
expect_column_distinct_values_to_equal_set (self, column, value_set,
                                             parse_strings_as_datetimes=None,
                                             result_format=None, include_config=True,
                                             catch_exceptions=None, meta=None)
```

Expect the set of distinct column values to equal a given set.

In contrast to `expect_column_distinct_values_to_contain_set()` this ensures not only that a certain set of values are present in the column but that these *and only these* values are present.

`expect_column_distinct_values_to_equal_set` is a [column_aggregate_expectation](#).

For example:

```
# my_df.my_col = [1,2,2,3,3,3]
>>> my_df.expect_column_distinct_values_to_equal_set(
    "my_col",
    [2,3]
)
{
  "success": false
  "result": {
    "observed_value": [1,2,3]
  },
}
```

Parameters

- **column** (*str*) – The column name.
- **value_set** (*set-like*) – A set of objects used for comparison.

Keyword Arguments **parse_strings_as_datetimes** (*boolean or None*) – If True values provided in `value_set` will be parsed as datetimes before making comparisons.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).

- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[expect_column_distinct_values_to_contain_set](#)

```
expect_column_distinct_values_to_contain_set (self, column, value_set,  
                                              parse_strings_as_datetimes=None,  
                                              result_format=None, include_config=True,  
                                              catch_exceptions=None,  
                                              meta=None)
```

Expect the set of distinct column values to contain a given set.

In contrast to `expect_column_values_to_be_in_set()` this ensures not that all column values are members of the given set but that values from the set *must* be present in the column.

`expect_column_distinct_values_to_contain_set` is a [column_aggregate_expectation](#).

For example:

```
# my_df.my_col = [1,2,2,3,3,3]
>>> my_df.expect_column_distinct_values_to_contain_set (
    "my_col",
    [2,3]
)
{
  "success": true
  "result": {
    "observed_value": [1,2,3]
  },
}
```

Parameters

- **column** (*str*) – The column name.
- **value_set** (*set-like*) – A set of objects used for comparison.

Keyword Arguments **parse_strings_as_datetimes** (*boolean or None*) – If True values provided in `value_set` will be parsed as datetimes before making comparisons.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: `BOOLEAN_ONLY`, `BASIC`, `COMPLETE`, or `SUMMARY`. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).

- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[expect_column_distinct_values_to_equal_set](#)

```
expect_column_mean_to_be_between(self, column, min_value=None, max_value=None,
                                   strict_min=False, strict_max=False, re-
                                   sult_format=None, include_config=True,
                                   catch_exceptions=None, meta=None)
```

Expect the column mean to be between a minimum value and a maximum value (inclusive).

`expect_column_mean_to_be_between` is a [column_aggregate_expectation](#).

Parameters

- **column** (*str*) – The column name.
- **min_value** (*float or None*) – The minimum value for the column mean.
- **max_value** (*float or None*) – The maximum value for the column mean.
- **strict_min** (*boolean*) – If True, the column mean must be strictly larger than `min_value`, default=False
- **strict_max** (*boolean*) – If True, the column mean must be strictly smaller than `max_value`, default=False

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: `BOOLEAN_ONLY`, `BASIC`, `COMPLETE`, or `SUMMARY`. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

Notes

These fields in the result object are customized for this expectation:

```
{
    "observed_value": (float) The true mean for the column
}
```

- `min_value` and `max_value` are both inclusive unless `strict_min` or `strict_max` are set to `True`.
- If `min_value` is `None`, then `max_value` is treated as an upper bound.
- If `max_value` is `None`, then `min_value` is treated as a lower bound.

See also:

[`expect_column_median_to_be_between`](#)

[`expect_column_stdev_to_be_between`](#)

```
expect_column_median_to_be_between(self, column, min_value=None, max_value=None,
                                     strict_min=False,      strict_max=False,      re-
                                     sult_format=None,        include_config=True,
                                     catch_exceptions=None, meta=None)
```

Expect the column median to be between a minimum value and a maximum value.

`expect_column_median_to_be_between` is a [`column_aggregate_expectation`](#).

Parameters

- **column** (*str*) – The column name.
- **min_value** (*int or None*) – The minimum value for the column median.
- **max_value** (*int or None*) – The maximum value for the column median.
- **strict_min** (*boolean*) – If `True`, the column median must be strictly larger than `min_value`, default=`False`
- **strict_max** (*boolean*) – If `True`, the column median must be strictly smaller than `max_value`, default=`False`

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: `BOOLEAN_ONLY`, `BASIC`, `COMPLETE`, or `SUMMARY`. For more detail, see [`result_format`](#).
- **include_config** (*boolean*) – If `True`, then include the expectation config as part of the result object. For more detail, see [`include_config`](#).
- **catch_exceptions** (*boolean or None*) – If `True`, then catch exceptions and include them as part of the result object. For more detail, see [`catch_exceptions`](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [`meta`](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [`result_format`](#) and [`include_config`](#), [`catch_exceptions`](#), and [`meta`](#).

Notes

These fields in the result object are customized for this expectation:

```
{
    "observed_value": (float) The true median for the column
}
```

- min_value and max_value are both inclusive unless strict_min or strict_max are set to True.
- If min_value is None, then max_value is treated as an upper bound
- If max_value is None, then min_value is treated as a lower bound

See also:

`expect_column_mean_to_be_between`

`expect_column_stddev_to_be_between`

```
expect_column_quantile_values_to_be_between(self, column, quantile_ranges,
                                              allow_relative_error=False,
                                              result_format=None, include_config=True,
                                              catch_exceptions=None, meta=None)
```

Expect specific provided column quantiles to be between provided minimum and maximum values.

quantile_ranges must be a dictionary with two keys:

- quantiles: (list of float) increasing ordered list of desired quantile values
- value_ranges: (list of lists): Each element in this list consists of a list with two values, a lower and upper bound (inclusive) for the corresponding quantile.

For each provided range:

- min_value and max_value are both inclusive.
- If min_value is None, then max_value is treated as an upper bound only
- If max_value is None, then min_value is treated as a lower bound only

The length of the quantiles list and quantile_values list must be equal.

For example:

```
# my_df.my_col = [1,2,2,3,3,3,4]
>>> my_df.expect_column_quantile_values_to_be_between(
    "my_col",
    {
        "quantiles": [0., 0.333, 0.6667, 1.],
        "value_ranges": [[0,1], [2,3], [3,4], [4,5]]
    }
)
{
    "success": True,
    "result": {
        "observed_value": {
            "quantiles": [0., 0.333, 0.6667, 1.],
            "values": [1, 2, 3, 4],
        }
    },
    "element_count": 7,
```

(continues on next page)

(continued from previous page)

```

    "missing_count": 0,
    "missing_percent": 0.0,
    "details": {
        "success_details": [true, true, true, true]
    }
}
}
}

```

`expect_column_quantile_values_to_be_between` can be computationally intensive for large datasets.

`expect_column_quantile_values_to_be_between` is a `column_aggregate_expectation`.

Parameters

- **column** (*str*) – The column name.
- **quantile_ranges** (*dictionary*) – Quantiles and associated value ranges for the column. See above for details.
- **allow_relative_error** (*boolean*) – Whether to allow relative error in quantile communications on backends that support or require it.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: `BOOLEAN_ONLY`, `BASIC`, `COMPLETE`, or `SUMMARY`. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

Notes

These fields in the result object are customized for this expectation: `:: details.success_details`

See also:

[expect_column_min_to_be_between](#)

[expect_column_max_to_be_between](#)

[expect_column_median_to_be_between](#)

expect_column_stdev_to_be_between (*self, column, min_value=None, max_value=None, strict_min=False, strict_max=False, result_format=None, include_config=True, catch_exceptions=None, meta=None*)

Expect the column standard deviation to be between a minimum value and a maximum value. Uses sample standard deviation (normalized by N-1).

`expect_column_stdev_to_be_between` is a `column_aggregate_expectation`.

Parameters

- **column** (*str*) – The column name.
- **min_value** (*float or None*) – The minimum value for the column standard deviation.
- **max_value** (*float or None*) – The maximum value for the column standard deviation.
- **strict_min** (*boolean*) – If True, the column standard deviation must be strictly larger than `min_value`, default=False
- **strict_max** (*boolean*) – If True, the column standard deviation must be strictly smaller than `max_value`, default=False

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

Notes

These fields in the result object are customized for this expectation:

```
{
    "observed_value": (float) The true standard deviation for the column
}
```

- `min_value` and `max_value` are both inclusive unless `strict_min` or `strict_max` are set to True.
- If `min_value` is None, then `max_value` is treated as an upper bound
- If `max_value` is None, then `min_value` is treated as a lower bound

See also:

[expect_column_mean_to_be_between](#)

[expect_column_median_to_be_between](#)

```
expect_column_unique_value_count_to_be_between(self, column, min_value=None,
                                                  max_value=None,          re-
                                                  sult_format=None,        in-
                                                  clude_config=True,
                                                  catch_exceptions=None,
                                                  meta=None)
```

Expect the number of unique values to be between a minimum value and a maximum value.

`expect_column_unique_value_count_to_be_between` is a `column_aggregate_expectation`.

Parameters

- **column** (*str*) – The column name.
- **min_value** (*int or None*) – The minimum number of unique values allowed.
- **max_value** (*int or None*) – The maximum number of unique values allowed.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

Notes

These fields in the result object are customized for this expectation:

```
{
  "observed_value": (int) The number of unique values in the column
}
```

- `min_value` and `max_value` are both inclusive.
- If `min_value` is `None`, then `max_value` is treated as an upper bound
- If `max_value` is `None`, then `min_value` is treated as a lower bound

See also:

[`expect_column_proportion_of_unique_values_to_be_between`](#)

```

expect_column_proportion_of_unique_values_to_be_between(self, column,
                                                         min_value=0,
                                                         max_value=1,
                                                         strict_min=False,
                                                         strict_max=False,
                                                         result_format=None,
                                                         include_config=True,
                                                         catch_exceptions=None,
                                                         meta=None)

```

Expect the proportion of unique values to be between a minimum value and a maximum value.

For example, in a column containing [1, 2, 2, 3, 3, 3, 4, 4, 4, 4], there are 4 unique values and 10 total values for a proportion of 0.4.

`expect_column_proportion_of_unique_values_to_be_between` is a `column_aggregate_expectation`.

Parameters

- **column** (*str*) – The column name.
- **min_value** (*float or None*) – The minimum proportion of unique values. (Proportions are on the range 0 to 1)
- **max_value** (*float or None*) – The maximum proportion of unique values. (Proportions are on the range 0 to 1)
- **strict_min** (*boolean*) – If True, the minimum proportion of unique values must be strictly larger than `min_value`, default=False
- **strict_max** (*boolean*) – If True, the maximum proportion of unique values must be strictly smaller than `max_value`, default=False

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: `BOOLEAN_ONLY`, `BASIC`, `COMPLETE`, or `SUMMARY`. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

Notes

These fields in the result object are customized for this expectation:

```
{
    "observed_value": (float) The proportion of unique values in the column
}
```

- `min_value` and `max_value` are both inclusive unless `strict_min` or `strict_max` are set to `True`.
- If `min_value` is `None`, then `max_value` is treated as an upper bound
- If `max_value` is `None`, then `min_value` is treated as a lower bound

See also:

[`expect_column_unique_value_count_to_be_between`](#)

```
expect_column_most_common_value_to_be_in_set(self, column, value_set,
                                                ties_okay=None, re-
                                                sult_format=None, in-
                                                clude_config=True,
                                                catch_exceptions=None,
                                                meta=None)
```

Expect the most common value to be within the designated value set

`expect_column_most_common_value_to_be_in_set` is a `column_aggregate_expectation`.

Parameters

- **column** (*str*) – The column name
- **value_set** (*set-like*) – A list of potential values to match

Keyword Arguments **ties_okay** (*boolean or None*) – If `True`, then the expectation will still succeed if values outside the designated set are as common (but not more common) than designated values

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: `BOOLEAN_ONLY`, `BASIC`, `COMPLETE`, or `SUMMARY`. For more detail, see [`result_format`](#).
- **include_config** (*boolean*) – If `True`, then include the expectation config as part of the result object. For more detail, see [`include_config`](#).
- **catch_exceptions** (*boolean or None*) – If `True`, then catch exceptions and include them as part of the result object. For more detail, see [`catch_exceptions`](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [`meta`](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [`result_format`](#) and [`include_config`](#), [`catch_exceptions`](#), and [`meta`](#).

Notes

These fields in the result object are customized for this expectation:

```
{
  "observed_value": (list) The most common values in the column
}
```

observed_value contains a list of the most common values. Often, this will just be a single element. But if there's a tie for most common among multiple values, *observed_value* will contain a single copy of each most common value.

```
expect_column_sum_to_be_between(self, column, min_value=None, max_value=None,
                                strict_min=False, strict_max=False, re-
                                sult_format=None, include_config=True,
                                catch_exceptions=None, meta=None)
```

Expect the column to sum to be between an min and max value

`expect_column_sum_to_be_between` is a `column_aggregate_expectation`.

Parameters

- **column** (*str*) – The column name
- **min_value** (*comparable type or None*) – The minimal sum allowed.
- **max_value** (*comparable type or None*) – The maximal sum allowed.
- **strict_min** (*boolean*) – If True, the minimal sum must be strictly larger than `min_value`, default=False
- **strict_max** (*boolean*) – If True, the maximal sum must be strictly smaller than `max_value`, default=False

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: `BOOLEAN_ONLY`, `BASIC`, `COMPLETE`, or `SUMMARY`. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

Notes

These fields in the result object are customized for this expectation:

```
{
  "observed_value": (list) The actual column sum
}
```

- `min_value` and `max_value` are both inclusive unless `strict_min` or `strict_max` are set to `True`.
- If `min_value` is `None`, then `max_value` is treated as an upper bound
- If `max_value` is `None`, then `min_value` is treated as a lower bound

```
expect_column_min_to_be_between(self, column, min_value=None, max_value=None,
                                strict_min=False, strict_max=False,
                                parse_strings_as_datetimes=False,
                                output_strftime_format=None, result_format=None,
                                include_config=True, catch_exceptions=None,
                                meta=None)
```

Expect the column minimum to be between an min and max value

`expect_column_min_to_be_between` is a `column_aggregate_expectation`.

Parameters

- **column** (*str*) – The column name
- **min_value** (*comparable type or None*) – The minimal column minimum allowed.
- **max_value** (*comparable type or None*) – The maximal column minimum allowed.
- **strict_min** (*boolean*) – If `True`, the minimal column minimum must be strictly larger than `min_value`, default=`False`
- **strict_max** (*boolean*) – If `True`, the maximal column minimum must be strictly smaller than `max_value`, default=`False`

Keyword Arguments

- **parse_strings_as_datetimes** (*Boolean or None*) – If `True`, parse `min_value`, `max_value`s, and all non-null column values to datetimes before making comparisons.
- **output_strftime_format** (*str or None*) – A valid strftime format for datetime output. Only used if `parse_strings_as_datetimes=True`.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: `BOOLEAN_ONLY`, `BASIC`, `COMPLETE`, or `SUMMARY`. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If `True`, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If `True`, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to *result_format* and *include_config*, *catch_exceptions*, and *meta*.

Notes

These fields in the result object are customized for this expectation:

```
{
  "observed_value": (list) The actual column min
}
```

- `min_value` and `max_value` are both inclusive unless `strict_min` or `strict_max` are set to `True`.
- If `min_value` is `None`, then `max_value` is treated as an upper bound
- If `max_value` is `None`, then `min_value` is treated as a lower bound

```
expect_column_max_to_be_between(self, column, min_value=None, max_value=None,
                                  strict_min=False, strict_max=False,
                                  parse_strings_as_datetimes=False,
                                  output_strftime_format=None, result_format=None,
                                  include_config=True, catch_exceptions=None,
                                  meta=None)
```

Expect the column max to be between an min and max value

`expect_column_max_to_be_between` is a `column_aggregate_expectation`.

Parameters

- **column** (*str*) – The column name
- **min_value** (*comparable type or None*) – The minimum number of unique values allowed.
- **max_value** (*comparable type or None*) – The maximum number of unique values allowed.

Keyword Arguments

- **parse_strings_as_datetimes** (*Boolean or None*) – If `True`, parse `min_value`, `max_value`s, and all non-null column values to datetimes before making comparisons.
- **output_strftime_format** (*str or None*) – A valid strftime format for datetime output. Only used if `parse_strings_as_datetimes=True`.
- **strict_min** (*boolean*) – If `True`, the minimal column minimum must be strictly larger than `min_value`, default=`False`
- **strict_max** (*boolean*) – If `True`, the maximal column minimum must be strictly smaller than `max_value`, default=`False`

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: `BOOLEAN_ONLY`, `BASIC`, `COMPLETE`, or `SUMMARY`. For more detail, see *result_format*.

- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

Notes

These fields in the result object are customized for this expectation:

```
{
  "observed_value": (list) The actual column max
}
```

- `min_value` and `max_value` are both inclusive unless `strict_min` or `strict_max` are set to True.
- If `min_value` is None, then `max_value` is treated as an upper bound
- If `max_value` is None, then `min_value` is treated as a lower bound

expect_column_chisquare_test_p_value_to_be_greater_than(*self, column, partition_object=None, p=0.05, tail_weight_holdout=0, result_format=None, include_config=True, catch_exceptions=None, meta=None*)

Expect column values to be distributed similarly to the provided categorical partition. This expectation compares categorical distributions using a Chi-squared test. It returns `success=True` if values in the column match the distribution of the provided partition.

`expect_column_chisquare_test_p_value_to_be_greater_than` is a `column_aggregate_expectation`.

Parameters

- **column** (*str*) – The column name.
- **partition_object** (*dict*) – The expected partition object (see [Partition Objects](#)).
- **p** (*float*) – The p-value threshold for rejecting the null hypothesis of the Chi-Squared test. For values below the specified threshold, the expectation will return `success=False`, rejecting the null hypothesis that the distributions are the same. Defaults to 0.05.

Keyword Arguments **tail_weight_holdout** (*float between 0 and 1 or None*) – The amount of weight to split uniformly between values observed in the data but not present in the provided partition. `tail_weight_holdout` provides a mechanism to make the test less strict by assigning positive weights to unknown values observed in the data that are not present in the partition.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

Notes

These fields in the result object are customized for this expectation:

```
{
  "observed_value": (float) The true p-value of the Chi-squared test
  "details": {
    "observed_partition" (dict):
      The partition observed in the data.
    "expected_partition" (dict):
      The partition expected from the data, after including tail_weight_
↪holdout
  }
}
```

```
abstract expect_column_bootstrapped_ks_test_p_value_to_be_greater_than (self,
                                                                           col-
                                                                           umn,
                                                                           par-
                                                                           ti-
                                                                           tion_object=None,
                                                                           p=0.05,
                                                                           boot-
                                                                           strap_samples=None,
                                                                           boot-
                                                                           strap_sample_size=None,
                                                                           re-
                                                                           sult_format=None,
                                                                           in-
                                                                           clude_config=True,
                                                                           catch_exceptions=None,
                                                                           meta=None)
```

Expect column values to be distributed similarly to the provided continuous partition. This expectation compares continuous distributions using a bootstrapped Kolmogorov-Smirnov test. It returns *success=True* if values in the column match the distribution of the provided partition.

The expected cumulative density function (CDF) is constructed as a linear interpolation between the bins, using the provided weights. Consequently the test expects a piecewise uniform distribution using the bins from the provided partition object.

`expect_column_bootstrapped_ks_test_p_value_to_be_greater_than` is a `column_aggregate_expectation`.

Parameters

- **column** (*str*) – The column name.
- **partition_object** (*dict*) – The expected partition object (see [Partition Objects](#)).
- **p** (*float*) – The p-value threshold for the Kolmogorov-Smirnov test. For values below the specified threshold the expectation will return *success=False*, rejecting the null hypothesis that the distributions are the same. Defaults to 0.05.

Keyword Arguments

- **bootstrap_samples** (*int*) – The number bootstrap rounds. Defaults to 1000.
- **bootstrap_sample_size** (*int*) – The number of samples to take from the column for each bootstrap. A larger sample will increase the specificity of the test. Defaults to `2 * len(partition_object['weights'])`

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

Notes

These fields in the result object are customized for this expectation:

```
{
  "observed_value": (float) The true p-value of the KS test
  "details": {
    "bootstrap_samples": The number of bootstrap rounds used
    "bootstrap_sample_size": The number of samples taken from
      the column in each bootstrap round
    "observed_cdf": The cumulative density function observed
      in the data, a dict containing 'x' values and cdf_values
      (suitable for plotting)
    "expected_cdf" (dict):
      The cumulative density function expected based on the
      partition object, a dict containing 'x' values and
```

(continues on next page)

(continued from previous page)

```

        cdf_values (suitable for plotting)
    "observed_partition" (dict):
        The partition observed on the data, using the provided
        bins but also expanding from min(column) to max(column)
    "expected_partition" (dict):
        The partition expected from the data. For KS test,
        this will always be the partition_object parameter
    }
}

```

```

expect_column_kl_divergence_to_be_less_than(self, column, partition_object=None, threshold=None, tail_weight_holdout=0, internal_weight_holdout=0, bucketize_data=True, result_format=None, include_config=True, catch_exceptions=None, meta=None)

```

Expect the Kulback-Leibler (KL) divergence (relative entropy) of the specified column with respect to the partition object to be lower than the provided threshold.

KL divergence compares two distributions. The higher the divergence value (relative entropy), the larger the difference between the two distributions. A relative entropy of zero indicates that the data are distributed identically, *when binned according to the provided partition*.

In many practical contexts, choosing a value between 0.5 and 1 will provide a useful test.

This expectation works on both categorical and continuous partitions. See notes below for details.

`expect_column_kl_divergence_to_be_less_than` is a `column_aggregate_expectation`.

Parameters

- **column** (*str*) – The column name.
- **partition_object** (*dict*) – The expected partition object (see [Partition Objects](#)).
- **threshold** (*float*) – The maximum KL divergence to for which to return *success=True*. If KL divergence is larger than the provided threshold, the test will return *success=False*.

Keyword Arguments

- **internal_weight_holdout** (*float between 0 and 1 or None*) – The amount of weight to split uniformly among zero-weighted partition bins. `internal_weight_holdout` provides a mechanisms to make the test less strict by assigning positive weights to values observed in the data for which the partition explicitly expected zero weight. With no `internal_weight_holdout`, any value observed in such a region will cause KL divergence to rise to +Infinity. Defaults to 0.
- **tail_weight_holdout** (*float between 0 and 1 or None*) – The amount of weight to add to the tails of the histogram. Tail weight holdout is split evenly between `(-Infinity, min(partition_object['bins']))` and `(max(partition_object['bins']), +Infinity)`. `tail_weight_holdout` provides a mechanism to make the test less strict by assigning positive weights to values observed in the data that are not present in the partition. With no `tail_weight_holdout`, any value observed outside the provided `partition_object` will cause KL divergence to rise to +Infinity. Defaults to 0.
- **bucketize_data** (*boolean*) – If `True`, then continuous data will be bucketized before evaluation. Setting this parameter to `false` allows evaluation of KL divergence with a `None` partition object for profiling against discrete data.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

Notes

These fields in the result object are customized for this expectation:

```
{
  "observed_value": (float) The true KL divergence (relative entropy) or None,
  ↳if the value is calculated as infinity, -infinity, or NaN
  "details": {
    "observed_partition": (dict) The partition observed in the data
    "expected_partition": (dict) The partition against which the data were,
  ↳compared,
                                after applying specified weight holdouts.
  }
}
```

If the `partition_object` is categorical, this expectation will expect the values in column to also be categorical.

- If the column includes values that are not present in the partition, the `tail_weight_holdout` will be equally split among those values, providing a mechanism to weaken the strictness of the expectation (otherwise, relative entropy would immediately go to infinity).
- If the partition includes values that are not present in the column, the test will simply include zero weight for that value.

If the `partition_object` is continuous, this expectation will discretize the values in the column according to the bins specified in the `partition_object`, and apply the test to the resulting distribution.

- The `internal_weight_holdout` and `tail_weight_holdout` parameters provide a mechanism to weaken the expectation, since an expected weight of zero would drive relative entropy to be infinite if any data are observed in that interval.
- If `internal_weight_holdout` is specified, that value will be distributed equally among any intervals with weight zero in the `partition_object`.
- If `tail_weight_holdout` is specified, that value will be appended to the tails of the bins ((-Infinity, min(bins)) and (max(bins), Infinity)).

If relative entropy/kl divergence goes to infinity for any of the reasons mentioned above, the observed value will be set to `None`. This is because `inf`, `-inf`, `Nan`, are not json serializable and

cause some json parsers to crash when encountered. The python None token will be serialized to null in json.

See also:

[`expect_column_chisquare_test_p_value_to_be_greater_than`](#)

[`expect_column_bootstrapped_ks_test_p_value_to_be_greater_than`](#)

```
abstract expect_column_pair_values_to_be_equal(self, column_A, column_B, ignore_row_if='both_values_are_missing',
result_format=None,
include_config=True,
catch_exceptions=None,
meta=None)
```

Expect the values in column A to be the same as column B.

Parameters

- **column_A** (*str*) – The first column name
- **column_B** (*str*) – The second column name

Keyword Arguments **ignore_row_if** (*str*) – “both_values_are_missing”, “either_value_is_missing”, “neither”

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [*result_format*](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [*include_config*](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [*catch_exceptions*](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [*meta*](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [*result_format*](#) and [*include_config*](#), [*catch_exceptions*](#), and [*meta*](#).

```
abstract expect_column_pair_values_A_to_be_greater_than_B(self, column_A,
column_B,
or_equal=None,
parse_strings_as_datetimes=False,
allow_cross_type_comparisons=None,
ignore_row_if='both_values_are_missing',
result_format=None,
include_config=True,
catch_exceptions=None,
meta=None)
```

Expect values in column A to be greater than column B.

Parameters

- **column_A** (*str*) – The first column name
- **column_B** (*str*) – The second column name
- **or_equal** (*boolean or None*) – If True, then values can be equal, not strictly greater

Keyword Arguments

- **allow_cross_type_comparisons** (*boolean or None*) – If True, allow comparisons between types (e.g. integer and string). Otherwise, attempting such comparisons will raise an exception.
- **ignore_row_if** (*str*) – “both_values_are_missing”, “either_value_is_missing”, “neither”

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

```
abstract expect_column_pair_values_to_be_in_set (self, column_A, column_B, value_pairs_set, ignore_row_if='both_values_are_missing', result_format=None, include_config=True, catch_exceptions=None, meta=None)
```

Expect paired values from columns A and B to belong to a set of valid pairs.

Parameters

- **column_A** (*str*) – The first column name
- **column_B** (*str*) – The second column name
- **value_pairs_set** (*list of tuples*) – All the valid pairs to be matched

Keyword Arguments **ignore_row_if** (*str*) – “both_values_are_missing”, “either_value_is_missing”, “never”

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).

- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

```
abstract expect_multicolumn_values_to_be_unique(self, column_list, ignore_row_if='all_values_are_missing',
result_format=None,
include_config=True,
catch_exceptions=None,
meta=None)
```

Expect the values for each row to be unique across the columns listed.

Parameters **column_list** (*tuple or list*) – The first column name

Keyword Arguments **ignore_row_if** (*str*) – “all_values_are_missing”, “any_value_is_missing”, “never”

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

```
static _parse_value_set(value_set)
```

```
attempt_allowing_relative_error(self)
```

Subclasses can override this method if the respective data source (e.g., Redshift) supports “approximate” mode. In certain cases (e.g., for SparkDFDataset), a fraction between 0 and 1 (i.e., not only a boolean) is allowed.

```
class great_expectations.dataset.MetaPandasDataset(*args, **kwargs)
```

Bases: [great_expectations.dataset.dataset.Dataset](#)

MetaPandasDataset is a thin layer between Dataset and PandasDataset.

This two-layer inheritance is required to make @classmethod decorators work.

Practically speaking, that means that MetaPandasDataset implements expectation decorators, like *column_map_expectation* and *column_aggregate_expectation*, and PandasDataset implements the expectation methods themselves.

classmethod `column_map_expectation` (*cls, func*)

Constructs an expectation using column-map semantics.

The MetaPandasDataset implementation replaces the “column” parameter supplied by the user with a pandas Series object containing the actual column from the relevant pandas dataframe. This simplifies the implementing expectation logic while preserving the standard Dataset signature and expected behavior.

See `column_map_expectation` for full documentation of this function.

classmethod `column_pair_map_expectation` (*cls, func*)

The `column_pair_map_expectation` decorator handles boilerplate issues surrounding the common pattern of evaluating truthiness of some condition on a per row basis across a pair of columns.

classmethod `multicolumn_map_expectation` (*cls, func*)

The `multicolumn_map_expectation` decorator handles boilerplate issues surrounding the common pattern of evaluating truthiness of some condition on a per row basis across a set of columns.

class `great_expectations.dataset.PandasDataset` (**args, **kwargs*)

Bases: `great_expectations.dataset.pandas_dataset.MetaPandasDataset`, `pandas.DataFrame`

`PandasDataset` instantiates the `great_expectations` Expectations API as a subclass of a `pandas.DataFrame`.

For the full API reference, please see `Dataset`

Notes

1. Samples and Subsets of `PandaDataSet` have ALL the expectations of the original data frame unless the user specifies the `discard_subset_failing_expectations = True` property on the original data frame.
2. Concatenations, joins, and merges of `PandaDataSets` contain NO expectations (since no autoinspection is performed by default).

`_internal_names`

`_internal_names_set`

property `_constructor` (*self*)

Used when a manipulation result has the same dimensions as the original.

`__finalize__` (*self, other, method=None, **kwargs*)

Propagate metadata from other to self.

Parameters

- **other** (the object from which to get the attributes that we are going) – to propagate
- **method** (*optional*, a passed method name ; possibly to take different) – types of propagation actions based on this

`get_row_count` (*self*)

Returns: int, table row count

`get_column_count` (*self*)

Returns: int, table column count

`get_table_columns` (*self*)

Returns: List[str], list of column names

get_column_sum (*self*, *column*)

Returns: float

get_column_max (*self*, *column*, *parse_strings_as_datetimes=False*)

Returns: any

get_column_min (*self*, *column*, *parse_strings_as_datetimes=False*)

Returns: any

get_column_mean (*self*, *column*)

Returns: float

get_column_nonnull_count (*self*, *column*)

Returns: int

get_column_value_counts (*self*, *column*, *sort='value'*, *collate=None*)

Get a series containing the frequency counts of unique values from the named column.

Parameters

- **column** – the column for which to obtain value_counts
- **sort** (*string*) – must be one of “value”, “count”, or “none”. - if “value” then values in the resulting partition object will be sorted lexicographically - if “count” then values will be sorted according to descending count (frequency) - if “none” then values will not be sorted
- **collate** (*string*) – the collate (sort) method to be used on supported backends (SqlAlchemy only)

Returns pd.Series of value counts for a column, sorted according to the value requested in sort

get_column_unique_count (*self*, *column*)

Returns: int

get_column_modes (*self*, *column*)

Returns: List[any], list of modes (ties OK)

get_column_median (*self*, *column*)

Returns: any

get_column_quantiles (*self*, *column*, *quantiles*, *allow_relative_error=False*)

Get the values in column closest to the requested quantiles :param column: name of column :type column: string :param quantiles: the quantiles to return. quantiles *must* be a tuple to ensure caching is possible :type quantiles: tuple of float

Returns the nearest values in the dataset to those quantiles

Return type List[any]

get_column_stdev (*self*, *column*)

Returns: float

get_column_hist (*self*, *column*, *bins*)

Get a histogram of column values :param column: the column for which to generate the histogram :param bins: the bins to slice the histogram. bins *must* be a tuple to ensure caching is possible :type bins: tuple

Returns: List[int], a list of counts corresponding to bins

get_column_count_in_range (*self*, *column*, *min_val=None*, *max_val=None*, *strict_min=False*, *strict_max=True*)

Returns: int

```
expect_column_values_to_be_unique(self, column, mostly=None, result_format=None,  
                                   include_config=True, catch_exceptions=None,  
                                   meta=None)
```

Expect each column value to be unique.

This expectation detects duplicates. All duplicated values are counted as exceptions.

For example, `[1, 2, 3, 3, 3]` will return `[3, 3, 3]` in `result.exceptions_list`, with `unexpected_percent = 60.0`.

`expect_column_values_to_be_unique` is a [column_map_expectation](#).

Parameters `column` (*str*) – The column name.

Keyword Arguments `mostly` (*None or a float between 0 and 1*) – Return “*success*”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

```
expect_column_values_to_not_be_null(self, column, mostly=None, result_format=None,  
                                   include_config=True, catch_exceptions=None,  
                                   meta=None, include_nulls=True)
```

Expect column values to not be null.

To be counted as an exception, values must be explicitly null or missing, such as a `NULL` in PostgreSQL or an `np.NaN` in pandas. Empty strings don’t count as null unless they have been coerced to a null type.

`expect_column_values_to_not_be_null` is a [column_map_expectation](#).

Parameters `column` (*str*) – The column name.

Keyword Arguments `mostly` (*None or a float between 0 and 1*) – Return “*success*”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).

- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[expect_column_values_to_be_null](#)

```
expect_column_values_to_be_null(self, column, mostly=None, result_format=None,
                                include_config=True, catch_exceptions=None,
                                meta=None)
```

Expect column values to be null.

`expect_column_values_to_be_null` is a [column_map_expectation](#).

Parameters `column` (*str*) – The column name.

Keyword Arguments `mostly` (*None or a float between 0 and 1*) – Return “success”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: `BOOLEAN_ONLY`, `BASIC`, `COMPLETE`, or `SUMMARY`. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[expect_column_values_to_not_be_null](#)

```
expect_column_values_to_be_of_type(self, column, type_, **kwargs)
```

The pandas implementation of this expectation takes kwargs `mostly`, `result_format`, `include_config`, `catch_exceptions`, and `meta` as other expectations, however it declares `**kwargs` because it needs to be able to fork into either aggregate or map semantics depending on the column type (see below).

In Pandas, columns *may* be typed, or they may be of the generic “object” type which can include rows with different storage types in the same column.

To respect that implementation, the `expect_column_values_to_be_of_type` expectations will first attempt to use the column dtype information to determine whether the column is restricted to the provided type. If that is possible, then `expect_column_values_to_be_of_type` will return aggregate information including an `observed_value`, similarly to other backends.

If it is not possible (because the column dtype is “object” but a more specific type was specified), then PandasDataset will use column map semantics: it will return map expectation results and check each value individually, which can be substantially slower.

Unfortunately, the “object” type is also used to contain any string-type columns (including ‘str’ and numpy `‘string_’` (bytes)); consequently, it is not possible to test for string columns using aggregate semantics.

```
_expect_column_values_to_be_of_type__aggregate (self, column, type_, mostly=None,
                                                result_format=None,
                                                include_config=True,
                                                catch_exceptions=None,
                                                meta=None)
```

```
static _native_type_type_map (type_)
```

```
_expect_column_values_to_be_of_type__map (self, column, type_, mostly=None, re-
                                            sult_format=None, include_config=True,
                                            catch_exceptions=None, meta=None)
```

```
expect_column_values_to_be_in_type_list (self, column, type_list, **kwargs)
```

The pandas implementation of this expectation takes kwargs mostly, result_format, include_config, catch_exceptions, and meta as other expectations, however it declares ****kwargs** because it needs to be able to fork into either aggregate or map semantics depending on the column type (see below).

In Pandas, columns *may* be typed, or they may be of the generic “object” type which can include rows with different storage types in the same column.

To respect that implementation, the expect_column_values_to_be_of_type expectations will first attempt to use the column dtype information to determine whether the column is restricted to the provided type. If that is possible, then expect_column_values_to_be_of_type will return aggregate information including an observed_value, similarly to other backends.

If it is not possible (because the column dtype is “object” but a more specific type was specified), then PandasDataset will use column map semantics: it will return map expectation results and check each value individually, which can be substantially slower.

Unfortunately, the “object” type is also used to contain any string-type columns (including ‘str’ and numpy `‘string_’` (bytes)); consequently, it is not possible to test for string columns using aggregate semantics.

```
_expect_column_values_to_be_in_type_list__aggregate (self, column, type_list,
                                                        mostly=None, re-
                                                        sult_format=None,
                                                        include_config=True,
                                                        catch_exceptions=None,
                                                        meta=None)
```

```
_expect_column_values_to_be_in_type_list__map (self, column, type_list,
                                                  mostly=None, result_format=None,
                                                  include_config=True,
                                                  catch_exceptions=None,
                                                  meta=None)
```

```
expect_column_values_to_be_in_set (self, column, value_set, mostly=None,
                                     parse_strings_as_datetimes=None, re-
                                     sult_format=None, include_config=True,
                                     catch_exceptions=None, meta=None)
```

Expect each column value to be in a given set.

For example:


```
# my_df.my_col = [1,2,2,3,3,3]
>>> my_df.expect_column_values_to_be_in_set(
    "my_col",
    [2,3]
)
{
  "success": false
  "result": {
    "unexpected_count": 1
    "unexpected_percent": 16.666666666666666,
    "unexpected_percent_nonmissing": 16.666666666666666,
    "partial_unexpected_list": [
      1
    ],
  },
}
```

`expect_column_values_to_be_in_set` is a *column_map_expectation*.

Parameters

- **column** (*str*) – The column name.
- **value_set** (*set-like*) – A set of objects used for comparison.

Keyword Arguments

- **mostly** (*None or a float between 0 and 1*) – Return “success”: *True* if at least mostly fraction of values match the expectation. For more detail, see *mostly*.
- **parse_strings_as_datetimes** (*boolean or None*) – If *True* values provided in *value_set* will be parsed as datetimes before making comparisons.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see *result_format*.
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see *include_config*.
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see *catch_exceptions*.
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see *meta*.

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to *result_format* and *include_config*, *catch_exceptions*, and *meta*.

See also:

expect_column_values_to_not_be_in_set

```
expect_column_values_to_not_be_in_set(self, column, value_set, mostly=None,
                                         parse_strings_as_datetimes=None, re-
                                         sult_format=None, include_config=True,
                                         catch_exceptions=None, meta=None)
```

Expect column entries to not be in the set.

For example:

```
# my_df.my_col = [1,2,2,3,3,3]
>>> my_df.expect_column_values_to_not_be_in_set(
    "my_col",
    [1,2]
)
{
  "success": false
  "result": {
    "unexpected_count": 3
    "unexpected_percent": 50.0,
    "unexpected_percent_nonmissing": 50.0,
    "partial_unexpected_list": [
      1, 2, 2
    ],
  },
}
```

`expect_column_values_to_not_be_in_set` is a [column_map_expectation](#).

Parameters

- **column** (*str*) – The column name.
- **value_set** (*set-like*) – A set of objects used for comparison.

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return “*success*”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[expect_column_values_to_be_in_set](#)

expect_column_values_to_be_between (*self, column, min_value=None, max_value=None, strict_min=False, strict_max=False, parse_strings_as_datetimes=None, output_strftime_format=None, allow_cross_type_comparisons=None, mostly=None, result_format=None, include_config=True, catch_exceptions=None, meta=None*)

Expect column entries to be between a minimum value and a maximum value (inclusive).

`expect_column_values_to_be_between` is a [*column_map_expectation*](#).

Parameters

- **column** (*str*) – The column name.
- **min_value** (*comparable type or None*) – The minimum value for a column entry.
- **max_value** (*comparable type or None*) – The maximum value for a column entry.

Keyword Arguments

- **strict_min** (*boolean*) – If True, values must be strictly larger than `min_value`, default=False
- **strict_max** (*boolean*) – If True, values must be strictly smaller than `max_value`, default=False
- **allow_cross_type_comparisons** (*boolean or None*) : If True, allow comparisons between types (e.g. integer and string). Otherwise, attempting such comparisons will raise an exception.
- **parse_strings_as_datetimes** (*boolean or None*) – If True, parse `min_value`, `max_value`, and all non-null column values to datetimes before making comparisons.
- **output_strftime_format** (*str or None*) – A valid strftime format for datetime output. Only used if `parse_strings_as_datetimes=True`.
- **mostly** (*None or a float between 0 and 1*) – Return “success”: True if at least mostly fraction of values match the expectation. For more detail, see [*mostly*](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: `BOOLEAN_ONLY`, `BASIC`, `COMPLETE`, or `SUMMARY`. For more detail, see [*result_format*](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [*include_config*](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [*catch_exceptions*](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [*meta*](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [*result_format*](#) and [*include_config*](#), [*catch_exceptions*](#), and [*meta*](#).

Notes

- `min_value` and `max_value` are both inclusive unless `strict_min` or `strict_max` are set to `True`.
- If `min_value` is `None`, then `max_value` is treated as an upper bound, and there is no minimum value checked.
- If `max_value` is `None`, then `min_value` is treated as a lower bound, and there is no maximum value checked.

See also:

[`expect_column_value_lengths_to_be_between`](#)

```
expect_column_values_to_be_increasing(self, column, strictly=None,
                                         parse_strings_as_datetimes=None,
                                         mostly=None, result_format=None, include_config=True,
                                         catch_exceptions=None, meta=None)
```

Expect column values to be increasing.

By default, this expectation only works for numeric or datetime data. When `parse_strings_as_datetimes=True`, it can also parse strings to datetimes.

If `strictly=True`, then this expectation is only satisfied if each consecutive value is strictly increasing—equal values are treated as failures.

`expect_column_values_to_be_increasing` is a [`column_map_expectation`](#).

Parameters `column` (*str*) – The column name.

Keyword Arguments

- **`strictly`** (*Boolean or None*) – If `True`, values must be strictly greater than previous values
- **`parse_strings_as_datetimes`** (*boolean or None*) – If `True`, all non-null column values to datetimes before making comparisons
- **`mostly`** (*None or a float between 0 and 1*) – Return “success”: `True` if at least mostly fraction of values match the expectation. For more detail, see [`mostly`](#).

Other Parameters

- **`result_format`** (*str or None*) – Which output mode to use: `BOOLEAN_ONLY`, `BASIC`, `COMPLETE`, or `SUMMARY`. For more detail, see [`result_format`](#).
- **`include_config`** (*boolean*) – If `True`, then include the expectation config as part of the result object. For more detail, see [`include_config`](#).
- **`catch_exceptions`** (*boolean or None*) – If `True`, then catch exceptions and include them as part of the result object. For more detail, see [`catch_exceptions`](#).
- **`meta`** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [`meta`](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [`result_format`](#) and [`include_config`](#), [`catch_exceptions`](#), and [`meta`](#).

See also:

[`expect_column_values_to_be_decreasing`](#)

```
expect_column_values_to_be_decreasing(self, column, strictly=None,
                                       parse_strings_as_datetimes=None,
                                       mostly=None, result_format=None, include_config=True,
                                       catch_exceptions=None, meta=None)
```

Expect column values to be decreasing.

By default, this expectation only works for numeric or datetime data. When `parse_strings_as_datetimes=True`, it can also parse strings to datetimes.

If `strictly=True`, then this expectation is only satisfied if each consecutive value is strictly decreasing—equal values are treated as failures.

`expect_column_values_to_be_decreasing` is a [`column_map_expectation`](#).

Parameters `column` (*str*) – The column name.

Keyword Arguments

- **strictly** (*Boolean or None*) – If True, values must be strictly greater than previous values
- **parse_strings_as_datetimes** (*boolean or None*) – If True, all non-null column values to datetimes before making comparisons
- **mostly** (*None or a float between 0 and 1*) – Return “success”: True if at least mostly fraction of values match the expectation. For more detail, see [`mostly`](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: `BOOLEAN_ONLY`, `BASIC`, `COMPLETE`, or `SUMMARY`. For more detail, see [`result_format`](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [`include_config`](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [`catch_exceptions`](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [`meta`](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [`result_format`](#) and [`include_config`](#), [`catch_exceptions`](#), and [`meta`](#).

See also:

[`expect_column_values_to_be_increasing`](#)

```
expect_column_value_lengths_to_be_between(self, column, min_value=None,
                                             max_value=None, mostly=None, result_format=None,
                                             include_config=True, catch_exceptions=None, meta=None)
```

Expect column entries to be strings with length between a minimum value and a maximum value (inclusive).

This expectation only works for string-type values. Invoking it on ints or floats will raise a `TypeError`.

`expect_column_value_lengths_to_be_between` is a *column_map_expectation*.

Parameters `column` (*str*) – The column name.

Keyword Arguments

- **min_value** (*int or None*) – The minimum value for a column entry length.
- **max_value** (*int or None*) – The maximum value for a column entry length.
- **mostly** (*None or a float between 0 and 1*) – Return “success”: *True* if at least mostly fraction of values match the expectation. For more detail, see *mostly*.

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see *result_format*.
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see *include_config*.
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see *catch_exceptions*.
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see *meta*.

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to *result_format* and *include_config*, *catch_exceptions*, and *meta*.

Notes

- `min_value` and `max_value` are both inclusive.
- If `min_value` is *None*, then `max_value` is treated as an upper bound, and the number of acceptable rows has no minimum.
- If `max_value` is *None*, then `min_value` is treated as a lower bound, and the number of acceptable rows has no maximum.

See also:

expect_column_value_lengths_to_equal

```
expect_column_value_lengths_to_equal(self, column, value, mostly=None, re-  
sult_format=None, include_config=True,  
catch_exceptions=None, meta=None)
```

Expect column entries to be strings with length equal to the provided value.

This expectation only works for string-type values. Invoking it on ints or floats will raise a *TypeError*.

`expect_column_values_to_be_between` is a *column_map_expectation*.

Parameters

- **column** (*str*) – The column name.
- **value** (*int or None*) – The expected value for a column entry length.

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return “*success*”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[expect_column_value_lengths_to_be_between](#)

expect_column_values_to_match_regex (*self, column, regex, mostly=None, result_format=None, include_config=True, catch_exceptions=None, meta=None*)

Expect column entries to be strings that match a given regular expression. Valid matches can be found anywhere in the string, for example “[at]+” will identify the following strings as expected: “cat”, “hat”, “aa”, “a”, and “t”, and the following strings as unexpected: “fish”, “dog”.

`expect_column_values_to_match_regex` is a [column_map_expectation](#).

Parameters

- **column** (*str*) – The column name.
- **regex** (*str*) – The regular expression the column entries should match.

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return “*success*”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to *result_format* and *include_config*, *catch_exceptions*, and *meta*.

See also:

expect_column_values_to_not_match_regex

expect_column_values_to_match_regex_list

expect_column_values_to_not_match_regex(*self*, *column*, *regex*, *mostly=None*, *result_format=None*, *include_config=True*, *catch_exceptions=None*, *meta=None*)

Expect column entries to be strings that do NOT match a given regular expression. The regex must not match any portion of the provided string. For example, “[at]+” would identify the following strings as expected: “fish”, “dog”, and the following as unexpected: “cat”, “hat”.

expect_column_values_to_not_match_regex is a *column_map_expectation*.

Parameters

- **column** (*str*) – The column name.
- **regex** (*str*) – The regular expression the column entries should NOT match.

Keyword Arguments **mostly** (*None* or a float between 0 and 1) – Return “success”: *True* if at least mostly fraction of values match the expectation. For more detail, see *mostly*.

Other Parameters

- **result_format** (*str* or *None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see *result_format*.
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see *include_config*.
- **catch_exceptions** (*boolean* or *None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see *catch_exceptions*.
- **meta** (*dict* or *None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see *meta*.

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to *result_format* and *include_config*, *catch_exceptions*, and *meta*.

See also:

expect_column_values_to_match_regex

expect_column_values_to_match_regex_list

expect_column_values_to_match_regex_list(*self*, *column*, *regex_list*, *match_on='any'*, *mostly=None*, *result_format=None*, *include_config=True*, *catch_exceptions=None*, *meta=None*)

Expect the column entries to be strings that can be matched to either any of or all of a list of regular expressions. Matches can be anywhere in the string.

expect_column_values_to_match_regex_list is a *column_map_expectation*.

Parameters

- **column** (*str*) – The column name.
- **regex_list** (*list*) – The list of regular expressions which the column entries should match

Keyword Arguments

- **match_on=** (*string*) – “any” or “all”. Use “any” if the value should match at least one regular expression in the list. Use “all” if it should match each regular expression in the list.
- **mostly** (*None or a float between 0 and 1*) – Return “success”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[expect_column_values_to_match_regex](#)

[expect_column_values_to_not_match_regex](#)

```
expect_column_values_to_not_match_regex_list (self, column, regex_list,
                                              mostly=None, result_format=None,
                                              include_config=True,
                                              catch_exceptions=None,
                                              meta=None)
```

Expect the column entries to be strings that do not match any of a list of regular expressions. Matches can be anywhere in the string.

`expect_column_values_to_not_match_regex_list` is a [column_map_expectation](#).

Parameters

- **column** (*str*) – The column name.
- **regex_list** (*list*) – The list of regular expressions which the column entries should not match

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return “success”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).

- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[expect_column_values_to_match_regex_list](#)

```
expect_column_values_to_match_strftime_format(self, column, strftime_format,
                                                mostly=None, result_format=None,
                                                include_config=True,
                                                catch_exceptions=None,
                                                meta=None)
```

Expect column entries to be strings representing a date or time with a given format.

`expect_column_values_to_match_strftime_format` is a [column_map_expectation](#).

Parameters

- **column** (*str*) – The column name.
- **strftime_format** (*str*) – A strftime format string to use for matching

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return “success”: True if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: `BOOLEAN_ONLY`, `BASIC`, `COMPLETE`, or `SUMMARY`. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

```
expect_column_values_to_be_dateutil_parseable(self, column, mostly=None,
                                                result_format=None,
                                                include_config=True,
                                                catch_exceptions=None,
                                                meta=None)
```

Expect column entries to be parsable using dateutil.

`expect_column_values_to_be_dateutil_parsable` is a [column_map_expectation](#).

Parameters `column` (*str*) – The column name.

Keyword Arguments `mostly` (*None or a float between 0 and 1*) – Return “*success*”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

`expect_column_values_to_be_json_parsable` (*self, column, mostly=None, result_format=None, include_config=True, catch_exceptions=None, meta=None*)

Expect column entries to be data written in JavaScript Object Notation.

`expect_column_values_to_be_json_parsable` is a [column_map_expectation](#).

Parameters `column` (*str*) – The column name.

Keyword Arguments `mostly` (*None or a float between 0 and 1*) – Return “*success*”: *True* if at least mostly fraction of values match the expectation. For more detail, see [mostly](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

See also:

[`expect_column_values_to_match_json_schema`](#)

```
expect_column_values_to_match_json_schema(self, column, json_schema,
                                           mostly=None, result_format=None,
                                           include_config=True,
                                           catch_exceptions=None, meta=None)
```

Expect column entries to be JSON objects matching a given JSON schema.

`expect_column_values_to_match_json_schema` is a [`column_map_expectation`](#).

Parameters `column` (*str*) – The column name.

Keyword Arguments `mostly` (*None or a float between 0 and 1*) – Return “success”: *True* if at least mostly fraction of values match the expectation. For more detail, see [`mostly`](#).

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [`result_format`](#).
- **include_config** (*boolean*) – If *True*, then include the expectation config as part of the result object. For more detail, see [`include_config`](#).
- **catch_exceptions** (*boolean or None*) – If *True*, then catch exceptions and include them as part of the result object. For more detail, see [`catch_exceptions`](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [`meta`](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [`result_format`](#) and [`include_config`](#), [`catch_exceptions`](#), and [`meta`](#).

See also:

[`expect_column_values_to_be_json_parseable`](#)

The JSON-schema docs.

```
expect_column_parameterized_distribution_ks_test_p_value_to_be_greater_than(self,
col-
umn,
dis-
tri-
bu-
tion,
p_value=0.05,
params=None,
re-
sult_format=None,
in-
clude_config=True,
catch_exceptions=None,
meta=None)
```

Expect the column values to be distributed similarly to a scipy distribution. This expectation compares the provided column to the specified continuous distribution with a parametric Kolmogorov-Smirnov test. The K-S test compares the provided column to the cumulative density function (CDF)

of the specified scipy distribution. If you don't know the desired distribution shape parameters, use the `ge.dataset.util.infer_distribution_parameters()` utility function to estimate them.

It returns 'success'=True if the p-value from the K-S test is greater than or equal to the provided p-value.

`expect_column_parameterized_distribution_ks_test_p_value_to_be_greater_than` is a `column_aggregate_expectation`.

Parameters

- **column** (*str*) – The column name.
- **distribution** (*str*) – The scipy distribution name. See: <https://docs.scipy.org/doc/scipy/reference/stats.html> Currently supported distributions are listed in the Notes section below.
- **p_value** (*float*) – The threshold p-value for a passing test. Default is 0.05.
- **params** (*dict or list*) – A dictionary or positional list of shape parameters that describe the distribution you want to test the data against. Include key values specific to the distribution from the appropriate scipy distribution CDF function. 'loc' and 'scale' are used as translational parameters. See <https://docs.scipy.org/doc/scipy/reference/stats.html#continuous-distributions>

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: `BOOLEAN_ONLY`, `BASIC`, `COMPLETE`, or `SUMMARY`. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

Notes

These fields in the result object are customized for this expectation:

```
{
  "details": {
    "expected_params" (dict): The specified or inferred parameters of the
    ↪ distribution to test against
    "ks_results" (dict): The raw result of stats.kstest()
  }
}
```

- The Kolmogorov-Smirnov test's null hypothesis is that the column is similar to the provided distribution.
- Supported scipy distributions:
 - norm

- beta
- gamma
- uniform
- chi2
- expon

```
expect_column_bootstrapped_ks_test_p_value_to_be_greater_than(self, column, partition_object=None, p=0.05, bootstrap_samples=None, bootstrap_sample_size=None, result_format=None, include_config=True, catch_exceptions=None, meta=None)
```

Expect column values to be distributed similarly to the provided continuous partition. This expectation compares continuous distributions using a bootstrapped Kolmogorov-Smirnov test. It returns *success=True* if values in the column match the distribution of the provided partition.

The expected cumulative density function (CDF) is constructed as a linear interpolation between the bins, using the provided weights. Consequently the test expects a piecewise uniform distribution using the bins from the provided partition object.

`expect_column_bootstrapped_ks_test_p_value_to_be_greater_than` is a `column_aggregate_expectation`.

Parameters

- **column** (*str*) – The column name.
- **partition_object** (*dict*) – The expected partition object (see [Partition Objects](#)).
- **p** (*float*) – The p-value threshold for the Kolmogorov-Smirnov test. For values below the specified threshold the expectation will return *success=False*, rejecting the null hypothesis that the distributions are the same. Defaults to 0.05.

Keyword Arguments

- **bootstrap_samples** (*int*) – The number bootstrap rounds. Defaults to 1000.
- **bootstrap_sample_size** (*int*) – The number of samples to take from the column for each bootstrap. A larger sample will increase the specificity of the test. Defaults to `2 * len(partition_object['weights'])`

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).

- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

Notes

These fields in the result object are customized for this expectation:

```
{
  "observed_value": (float) The true p-value of the KS test
  "details": {
    "bootstrap_samples": The number of bootstrap rounds used
    "bootstrap_sample_size": The number of samples taken from
                             the column in each bootstrap round
    "observed_cdf": The cumulative density function observed
                    in the data, a dict containing 'x' values and cdf_values
                    (suitable for plotting)
    "expected_cdf" (dict):
      The cumulative density function expected based on the
      partition object, a dict containing 'x' values and
      cdf_values (suitable for plotting)
    "observed_partition" (dict):
      The partition observed on the data, using the provided
      bins but also expanding from min(column) to max(column)
    "expected_partition" (dict):
      The partition expected from the data. For KS test,
      this will always be the partition_object parameter
  }
}
```

```
expect_column_pair_values_to_be_equal(self, column_A, column_B, ignore_row_if='both_values_are_missing',
                                     result_format=None, include_config=True,
                                     catch_exceptions=None, meta=None)
```

Expect the values in column A to be the same as column B.

Parameters

- **column_A** (*str*) – The first column name
- **column_B** (*str*) – The second column name

Keyword Arguments **ignore_row_if** (*str*) – “both_values_are_missing”, “either_value_is_missing”, “neither”

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).

- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

```
expect_column_pair_values_A_to_be_greater_than_B(self,          column_A,          col-  
                                                    umn_B,          or_equal=None,  
                                                    parse_strings_as_datetimes=None,  
al-  
                                                    low_cross_type_comparisons=None,  
ig-  
                                                    nore_row_if='both_values_are_missing',  
                                                    result_format=None,  
                                                    include_config=True,  
                                                    catch_exceptions=None,  
                                                    meta=None)
```

Expect values in column A to be greater than column B.

Parameters

- **column_A** (*str*) – The first column name
- **column_B** (*str*) – The second column name
- **or_equal** (*boolean or None*) – If True, then values can be equal, not strictly greater

Keyword Arguments

- **allow_cross_type_comparisons** (*boolean or None*) – If True, allow comparisons between types (e.g. integer and string). Otherwise, attempting such comparisons will raise an exception.
- **ignore_row_if** (*str*) – “both_values_are_missing”, “either_value_is_missing”, “neither

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).


```
expect_column_pair_values_to_be_in_set (self, column_A, column_B, value_pairs_set,
                                         ignore_row_if='both_values_are_missing',
                                         result_format=None, include_config=True,
                                         catch_exceptions=None, meta=None)
```

Expect paired values from columns A and B to belong to a set of valid pairs.

Parameters

- **column_A** (*str*) – The first column name
- **column_B** (*str*) – The second column name
- **value_pairs_set** (*list of tuples*) – All the valid pairs to be matched

Keyword Arguments **ignore_row_if** (*str*) – “both_values_are_missing”, “either_value_is_missing”, “never”

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to [result_format](#) and [include_config](#), [catch_exceptions](#), and [meta](#).

```
expect_multicolumn_values_to_be_unique (self, column_list, ignore_row_if='all_values_are_missing',
                                         result_format=None, include_config=True,
                                         catch_exceptions=None, meta=None)
```

Expect the values for each row to be unique across the columns listed.

Parameters **column_list** (*tuple or list*) – The first column name

Keyword Arguments **ignore_row_if** (*str*) – “all_values_are_missing”, “any_value_is_missing”, “never”

Other Parameters

- **result_format** (*str or None*) – Which output mode to use: *BOOLEAN_ONLY*, *BASIC*, *COMPLETE*, or *SUMMARY*. For more detail, see [result_format](#).
- **include_config** (*boolean*) – If True, then include the expectation config as part of the result object. For more detail, see [include_config](#).
- **catch_exceptions** (*boolean or None*) – If True, then catch exceptions and include them as part of the result object. For more detail, see [catch_exceptions](#).
- **meta** (*dict or None*) – A JSON-serializable dictionary (nesting allowed) that will be included in the output without modification. For more detail, see [meta](#).

Returns

A JSON-serializable expectation result object.

Exact fields vary depending on the values passed to *result_format* and *include_config*, *catch_exceptions*, and *meta*.

`great_expectations.dataset.logger`

`great_expectations.datasource`

Subpackages

`great_expectations.datasource.batch_kwarg_generator`

Submodules

`great_expectations.datasource.batch_kwarg_generator.batch_kwarg_generator`

Module Contents

Classes

<code>BatchKwargGenerator(name, datasource)</code>	BatchKwargGenerators produce identifying information, called “batch_kwarg” that datasources
--	---

`great_expectations.datasource.batch_kwarg_generator.batch_kwarg_generator.logger`

class `great_expectations.datasource.batch_kwarg_generator.batch_kwarg_generator.BatchKwargGenerator`

Bases: object

BatchKwargGenerators produce identifying information, called “batch_kwarg” that datasources can use to get individual batches of data. They add flexibility in how to obtain data such as with time-based partitioning, downsampling, or other techniques appropriate for the datasource.

For example, a batch kwarg generator could produce a SQL query that logically represents “rows in the Events table with a timestamp on February 7, 2012,” which a SQLAlchemyDatasource could use to materialize a SQLAlchemyDataset corresponding to that batch of data and ready for validation.

A batch is a sample from a data asset, sliced according to a particular rule. For example, an hourly slide of the Events table or “most recent *users* records.”

A Batch is the primary unit of validation in the Great Expectations DataContext. Batches include metadata that identifies how they were constructed—the same “batch_kwarg” assembled by the batch kwarg generator. While not every datasource will enable re-fetching a specific batch of data, GE can store snapshots of batches or store metadata from an external data version control system.

Example Generator Configurations follow:

```
my_datasource_1:
  class_name: PandasDatasource
  batch_kwarg_generators:
    # This generator will provide two data assets, corresponding to the globs_
    →defined under the "file_logs"
    # and "data_asset_2" keys. The file_logs asset will be partitioned according_
    →to the match group
```

(continues on next page)

(continued from previous page)

```

# defined in partition_regex
default:
    class_name: GlobReaderBatchKwargsGenerator
    base_directory: /var/logs
    reader_options:
        sep: "
    globs:
        file_logs:
            glob: logs/*.gz
            partition_regex: logs/file_(\d{0,4})_\.log\.gz
        data_asset_2:
            glob: data/*.csv

my_datasource_2:
    class_name: PandasDatasource
    batch_kwargs_generators:
        # This generator will create one data asset per subdirectory in /data
        # Each asset will have partitions corresponding to the filenames in that
        ↪subdirectory
        default:
            class_name: SubdirReaderBatchKwargsGenerator
            reader_options:
                sep: "
            base_directory: /data

my_datasource_3:
    class_name: SQLAlchemyDatasource
    batch_kwargs_generators:
        # This generator will search for a file named with the name of the requested
        ↪data asset and the
        # .sql suffix to open with a query to use to generate data
        default:
            class_name: QueryBatchKwargsGenerator

```

`_batch_kwargs_type`**`recognized_batch_parameters`****`property name`** (*self*)**`abstract _get_iterator`** (*self*, *data_asset_name*, ***kwargs*)**`abstract get_available_data_asset_names`** (*self*)

Return the list of asset names known by this batch kwargs generator.

Returns A list of available names**`abstract get_available_partition_ids`** (*self*, *generator_asset=None*, *data_asset_name=None*)Applies the current `_partitioner` to the batches available on *data_asset_name* and returns a list of valid `partition_id` strings that can be used to identify batches of data.**Parameters** *data_asset_name* – the data asset whose partitions should be returned.**Returns** A list of `partition_id` strings**`get_config`** (*self*)**`reset_iterator`** (*self*, *generator_asset=None*, *data_asset_name=None*, ***kwargs*)**`get_iterator`** (*self*, *generator_asset=None*, *data_asset_name=None*, ***kwargs*)

```
build_batch_kwargs (self, data_asset_name=None, partition_id=None, **kwargs)
abstract _build_batch_kwargs (self, batch_parameters)
yield_batch_kwargs (self, generator_asset=None, data_asset_name=None, **kwargs)
```

`great_expectations.datasource.batch_kwargs_generator.databricks_batch_kwargs_generator`

Module Contents

Classes

`DatabricksTableBatchKwargsGenerator`(name='default', **Meant to be used in a Databricks notebook**
datasource=None, database='default')

`great_expectations.datasource.batch_kwargs_generator.databricks_batch_kwargs_generator.log`
class `great_expectations.datasource.batch_kwargs_generator.databricks_batch_kwargs_generator`

Bases: `great_expectations.datasource.batch_kwargs_generator.
batch_kwargs_generator.BatchKwargsGenerator`

Meant to be used in a Databricks notebook

get_available_data_asset_names (self)
Return the list of asset names known by this batch kwargs generator.

Returns A list of available names

_get_iterator (self, data_asset_name, **kwargs)

`great_expectations.datasource.batch_kwargs_generator.glob_reader_batch_kwargs_generator`

Module Contents

Classes

`GlobReaderBatchKwargsGenerator`(name='default', `GlobReaderBatchKwargsGenerator` processes files in a di-
datasource=None, base_directory='/data', rectory according to glob patterns to produce batches of
reader_options=None, asset_globs=None, data.
reader_method=None)

`great_expectations.datasource.batch_kwargs_generator.glob_reader_batch_kwargs_generator.log`

class great_expectations.datasource.batch_kwarg_generator.glob_reader_batch_kwarg_generator

Bases: `great_expectations.datasource.batch_kwarg_generator.batch_kwarg_generator.BatchKwargGenerator`

GlobReaderBatchKwargGenerator processes files in a directory according to glob patterns to produce batches of data.

A more interesting asset_glob might look like the following:

```
daily_logs:
  glob: daily_logs/*.csv
  partition_regex: daily_logs/((19|20)\d\d[- /.](0[1-9]|1[012])[- /.](0[1-
↪9]|1[12][0-9]|3[01]))_(.*)\.csv
```

The “glob” key ensures that every csv file in the daily_logs directory is considered a batch for this data asset. The “partition_regex” key ensures that files whose basename begins with a date (with components hyphen, space, forward slash, period, or null separated) will be identified by a partition_id equal to just the date portion of their name.

A fully configured GlobReaderBatchKwargGenerator in yml might look like the following:

```
my_datasource:
  class_name: PandasDatasource
  batch_kwarg_generators:
    my_generator:
      class_name: GlobReaderBatchKwargGenerator
      base_directory: /var/log
      reader_options:
        sep: %
        header: 0
      reader_method: csv
      asset_globs:
        wifi_logs:
          glob: wifi*.log
          partition_regex: wifi-((0[1-9]|1[012])-(0[1-9]|1[12][0-9]|3[01])-20\d\d)
↪*.log
      reader_method: csv
```

recognized_batch_parameters

property reader_options(*self*)

property asset_globs(*self*)

property reader_method(*self*)

property base_directory(*self*)

get_available_data_asset_names(*self*)

Return the list of asset names known by this batch kwarg generator.

Returns A list of available names

get_available_partition_ids (*self*, *generator_asset=None*, *data_asset_name=None*)

Applies the current `_partitioner` to the batches available on `data_asset_name` and returns a list of valid `partition_id` strings that can be used to identify batches of data.

Parameters `data_asset_name` – the data asset whose partitions should be returned.

Returns A list of `partition_id` strings

_build_batch_kwargs (*self*, *batch_parameters*)

_get_data_asset_paths (*self*, *data_asset_name*)

Returns a list of filepaths associated with the given `data_asset_name`

Parameters `data_asset_name` –

Returns paths (list)

_get_data_asset_config (*self*, *data_asset_name*)

_get_iterator (*self*, *data_asset_name*, *reader_method=None*, *reader_options=None*, *limit=None*)

_build_batch_kwargs_path_iter (*self*, *path_list*, *glob_config*, *reader_method=None*,
reader_options=None, *limit=None*)

_build_batch_kwargs_from_path (*self*, *path*, *glob_config*, *reader_method=None*,
reader_options=None, *limit=None*)

_partitioner (*self*, *path*, *glob_config*)

`great_expectations.datasource.batch_kwargs_generator.manual_batch_kwargs_generator`

Module Contents

Classes

<code>ManualBatchKwargsGenerator</code> (<code>name='default'</code> , <code>datasource=None</code> , <code>assets=None</code>)	<code>ManualBatchKwargsGenerator</code> returns manually-configured <code>batch_kwargs</code> for named data assets. It provides a
--	--

`great_expectations.datasource.batch_kwargs_generator.manual_batch_kwargs_generator.logger`

class `great_expectations.datasource.batch_kwargs_generator.manual_batch_kwargs_generator.M`

Bases: `great_expectations.datasource.batch_kwargs_generator.batch_kwargs_generator.BatchKwargsGenerator`

`ManualBatchKwargsGenerator` returns manually-configured `batch_kwargs` for named data assets. It provides a convenient way to capture complete batch definitions without requiring the configuration of a more fully-featured batch kwargs generator.

A fully configured `ManualBatchKwargsGenerator` in yml might look like the following:

```
my_datasource:
  class_name: PandasDatasource
  batch_kwargs_generators:
    my_generator:
```

(continues on next page)

(continued from previous page)

```

class_name: ManualBatchKwargsGenerator
assets:
  asset1:
    - partition_id: 1
      path: /data/file_1.csv
      reader_options:
        sep: ;
    - partition_id: 2
      path: /data/file_2.csv
      reader_options:
        header: 0
logs:
  path: data/log.csv

```

recognized_batch_parameters**property assets** (*self*)**get_available_data_asset_names** (*self*)

Return the list of asset names known by this batch kwargs generator.

Returns A list of available names**_get_data_asset_config** (*self*, *data_asset_name*)**_get_iterator** (*self*, *data_asset_name*, ***kwargs*)**get_available_partition_ids** (*self*, *generator_asset=None*, *data_asset_name=None*)Applies the current `_partitioner` to the batches available on *data_asset_name* and returns a list of valid `partition_id` strings that can be used to identify batches of data.**Parameters** *data_asset_name* – the data asset whose partitions should be returned.**Returns** A list of `partition_id` strings**_build_batch_kwargs** (*self*, *batch_parameters*)

Build batch kwargs from a partition id.

great_expectations.datasource.batch_kwargs_generator.query_batch_kwargs_generator**Module Contents****Classes**

<code>QueryBatchKwargsGenerator</code> (<i>name='default'</i> ,	Produce query-style batch_kwargs from sql files stored on
<i>datasource=None</i> ,	<i>query_store_backend=None</i> , disk
<i>queries=None</i>)	

great_expectations.datasource.batch_kwargs_generator.query_batch_kwargs_generator.**logger**great_expectations.datasource.batch_kwargs_generator.query_batch_kwargs_generator.**sqlalchemy**

class great_expectations.datasource.batch_kwarg_generator.query_batch_kwarg_generator.Qu

Bases: `great_expectations.datasource.batch_kwarg_generator.batch_kwarg_generator.BatchKwargGenerator`

Produce query-style batch_kwarg from sql files stored on disk

recognized_batch_parameters

_get_raw_query (*self*, *data_asset_name*)

_get_iterator (*self*, *data_asset_name*, *query_parameters=None*)

add_query (*self*, *generator_asset=None*, *query=None*, *data_asset_name=None*)

get_available_data_asset_names (*self*)

Return the list of asset names known by this batch kwarg generator.

Returns A list of available names

_build_batch_kwarg (*self*, *batch_parameters*)

Build batch kwarg from a partition id.

get_available_partition_ids (*self*, *generator_asset=None*, *data_asset_name=None*)

Applies the current _partitioner to the batches available on data_asset_name and returns a list of valid partition_id strings that can be used to identify batches of data.

Parameters *data_asset_name* – the data asset whose partitions should be returned.

Returns A list of partition_id strings

great_expectations.datasource.batch_kwarg_generator.s3_batch_kwarg_generator

Module Contents

Classes

`S3GlobReaderBatchKwargGenerator`(name='default', datasource=None, bucket=None, reader_options=None, assets=None, delimiter='/', reader_method=None, boto3_options=None, max_keys=1000)

S3BatchKwargGenerator provides support for generating batches of data from an S3 bucket. For the S3 batch kwarg generator, assets must

great_expectations.datasource.batch_kwarg_generator.s3_batch_kwarg_generator.**boto3**

great_expectations.datasource.batch_kwarg_generator.s3_batch_kwarg_generator.**logger**

class great_expectations.datasource.batch_kwarg_generator.s3_batch_kwarg_generator.S3Glob

Bases: `great_expectations.datasource.batch_kwarg_generator.batch_kwarg_generator.BatchKwargGenerator`

S3 BatchKwargGenerator provides support for generating batches of data from an S3 bucket. For the S3 batch kwarg generator, assets must be individually defined using a prefix and glob, although several additional configuration parameters are available for assets (see below).

Example configuration:

```
datasources:
  my_datasource:
    ...
    batch_kwarg_generator:
      my_s3_generator:
        class_name: S3GlobReaderBatchKwargGenerator
        bucket: my_bucket.my_organization.priv
        reader_method: parquet # This will be automatically inferred from suffix,
        ↳ where possible, but can be explicitly specified as well
        reader_options: # Note that reader options can be specified globally or,
        ↳ per-asset
          sep: ","
          delimiter: "/" # Note that this is the delimiter for the BUCKET KEYS. By,
        ↳ default it is "/"
          boto3_options:
            endpoint_url: $S3_ENDPOINT # Use the S3_ENDPOINT environment variable,
        ↳ to determine which endpoint to use
            max_keys: 100 # The maximum number of keys to fetch in a single list,
        ↳ objects request to s3. When accessing batch_kwarg through an iterator, the,
        ↳ iterator will silently refetch if more keys were available
          assets:
            my_first_asset:
              prefix: my_first_asset/
              regex_filter: .* # The regex filter will filter the results returned,
        ↳ by S3 for the key and prefix to only those matching the regex
              directory_assets: True
            access_logs:
              prefix: access_logs
              regex_filter: access_logs/2019.*\.csv.gz
              sep: "~"
              max_keys: 100
```

recognized_batch_parameters

property reader_options (*self*)

property assets (*self*)

property `bucket` (*self*)

get_available_data_asset_names (*self*)

Return the list of asset names known by this batch kwargs generator.

Returns A list of available names

_get_iterator (*self*, *data_asset_name*, *reader_method=None*, *reader_options=None*, *limit=None*)

_build_batch_kwargs_path_iter (*self*, *path_list*, *reader_options=None*, *limit=None*)

_build_batch_kwargs (*self*, *batch_parameters*)

_build_batch_kwargs_from_key (*self*, *key*, *asset_config=None*, *reader_method=None*,
reader_options=None, *limit=None*)

_get_asset_options (*self*, *asset_config*, *iterator_dict*)

_build_asset_iterator (*self*, *asset_config*, *iterator_dict*, *reader_method=None*,
reader_options=None, *limit=None*)

get_available_partition_ids (*self*, *generator_asset=None*, *data_asset_name=None*)

Applies the current `_partitioner` to the batches available on *data_asset_name* and returns a list of valid `partition_id` strings that can be used to identify batches of data.

Parameters `data_asset_name` – the data asset whose partitions should be returned.

Returns A list of `partition_id` strings

_partitioner (*self*, *key*, *asset_config*)

`great_expectations.datasource.batch_kwargs_generator.subdir_reader_batch_kwargs_generator`

Module Contents

Classes

<code>SubdirReaderBatchKwargsGenerator</code> (<i>name='default'</i> , <i>datasource=None</i> , <i>reader_options=None</i> , <i>reader_method=None</i>)	The <code>SubdirReaderBatchKwargsGenerator</code> inspects a filesystem and produces path-based <code>batch_kwargs</code> . <i>base_directory='/data'</i> , <i>known_extensions=None</i> ,
--	--

`great_expectations.datasource.batch_kwargs_generator.subdir_reader_batch_kwargs_generator`

`great_expectations.datasource.batch_kwargs_generator.subdir_reader_batch_kwargs_generator`

class `great_expectations.datasource.batch_kwargs_generator.subdir_reader_batch_kwargs_gene`

Bases: `great_expectations.datasource.batch_kwargs_generator.batch_kwargs_generator.BatchKwargsGenerator`

The `SubdirReaderBatchKwargsGenerator` inspects a filesystem and produces path-based `batch_kwargs`.

SubdirReaderBatchKwargsGenerator recognizes data assets using two criteria:

- for files directly in ‘base_directory’ with recognized extensions (.csv, .tsv, .parquet, .xls, .xlsx, .json), it uses the name of the file without the extension
- for other files or directories in ‘base_directory’, it uses the file or directory name

SubdirReaderBatchKwargsGenerator sees all files inside a directory of base_directory as batches of one data-source.

SubdirReaderBatchKwargsGenerator can also include configured reader_options which will be added to batch_kwargs generated by this generator.

`_default_reader_options`

`recognized_batch_parameters`

`property reader_options` (*self*)

`property known_extensions` (*self*)

`property reader_method` (*self*)

`property base_directory` (*self*)

`get_available_data_asset_names` (*self*)

Return the list of asset names known by this batch kwargs generator.

Returns A list of available names

`get_available_partition_ids` (*self*, *generator_asset=None*, *data_asset_name=None*)

Applies the current _partitioner to the batches available on data_asset_name and returns a list of valid partition_id strings that can be used to identify batches of data.

Parameters **`data_asset_name`** – the data asset whose partitions should be returned.

Returns A list of partition_id strings

`_build_batch_kwargs` (*self*, *batch_parameters*)

Parameters **`batch_parameters`** –

Returns batch_kwargs

`_get_valid_file_options` (*self*, *base_directory=None*)

`_get_iterator` (*self*, *data_asset_name*, *reader_options=None*, *limit=None*)

`_build_batch_kwargs_path_iter` (*self*, *path_list*, *reader_options=None*, *limit=None*)

`_build_batch_kwargs_from_path` (*self*, *path*, *reader_method=None*, *reader_options=None*, *limit=None*)

`great_expectations.datasource.batch_kwargs_generator.table_batch_kwargs_generator`

Module Contents

Classes

<code>AssetConfigurationSchema</code> (*, types.StrSequenceOrSet = None, exclude: types.StrSequenceOrSet = (), many: bool = False, context: typing.Dict = None, load_only: types.StrSequenceOrSet = (), dump_only: types.StrSequenceOrSet = (), partial: typing.Union[bool, types.StrSequenceOrSet] = False, unknown: str = None)	only: Base schema class with which to define custom schemas.
<code>AssetConfiguration</code> (table, schema=None)	
<code>TableBatchKwargsGenerator</code> (name='default', datasource=None, assets=None)	Provide access to already materialized tables or views in a database.

`great_expectations.datasource.batch_kwargs_generator.table_batch_kwargs_generator.logger`
`great_expectations.datasource.batch_kwargs_generator.table_batch_kwargs_generator.sqlalchemy`

class great_expectations.datasource.batch_kwarg_generator.table_batch_kwarg_generator.As

Bases: `marshmallow.Schema`

Base schema class with which to define custom schemas.

Example usage:

```
import datetime as dt
from dataclasses import dataclass

from marshmallow import Schema, fields


@dataclass
class Album:
    title: str
```

(continues on next page)

(continued from previous page)

```

        release_date: dt.date

class AlbumSchema(Schema):
    title = fields.Str()
    release_date = fields.Date()

album = Album("Beggars Banquet", dt.date(1968, 12, 6))
schema = AlbumSchema()
data = schema.dump(album)
data # {'release_date': '1968-12-06', 'title': 'Beggars Banquet'}
```

Parameters

- **only** – Whitelist of the declared fields to select when instantiating the Schema. If None, all fields are used. Nested fields can be represented with dot delimiters.
- **exclude** – Blacklist of the declared fields to exclude when instantiating the Schema. If a field appears in both *only* and *exclude*, it is not used. Nested fields can be represented with dot delimiters.
- **many** – Should be set to *True* if *obj* is a collection so that the object will be serialized to a list.
- **context** – Optional context passed to `fields.Method` and `fields.Function` fields.
- **load_only** – Fields to skip during serialization (write-only fields)
- **dump_only** – Fields to skip during deserialization (read-only fields)
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to Nested fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.
- **unknown** – Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*.

Changed in version 3.0.0: *prefix* parameter removed.

Changed in version 2.0.0: `__validators__`, `__preprocessors__`, and `__data_handlers__` are removed in favor of `marshmallow.decorators.validates_schema`, `marshmallow.decorators.pre_load` and `marshmallow.decorators.post_dump`. `__accessor__` and `__error_handler__` are deprecated. Implement the `handle_error` and `get_attribute` methods instead.

table

schema

make_asset_configuration(*self*, *data*, ***kwargs*)

class great_expectations.datasource.batch_kwargs_generator.table_batch_kwargs_generator.**AssetConfiguration**

Bases: object

property **table**(*self*)

property **schema**(*self*)

great_expectations.datasource.batch_kwargs_generator.table_batch_kwargs_generator.**asset_configuration**

class great_expectations.datasource.batch_kwarg_generator.table_batch_kwarg_generator.TableBatchKwargGenerator

Bases: `great_expectations.datasource.batch_kwarg_generator.batch_kwarg_generator.BatchKwargGenerator`

Provide access to already materialized tables or views in a database.

TableBatchKwargGenerator can be used to define specific data asset names that take and substitute parameters, for example to support referring to the same data asset but with different schemas depending on provided batch_kwarg.

The python template language is used to substitute table name portions. For example, consider the following configurations:

```
my_generator:
  class_name: TableBatchKwargGenerator
  assets:
    my_table:
      schema: $schema
      table: my_table
```

In that case, the asset my_datasource/my_generator/my_asset will refer to a table called my_table in a schema defined in batch_kwarg.

recognized_batch_parameters

_get_iterator (*self*, *data_asset_name*, *query_parameters=None*, *limit=None*, *offset=None*, *partition_id=None*)

get_available_data_asset_names (*self*)

Return the list of asset names known by this batch kwarg generator.

Returns A list of available names

_build_batch_kwarg (*self*, *batch_parameters*)

get_available_partition_ids (*self*, *generator_asset=None*, *data_asset_name=None*)

Applies the current _partitioner to the batches available on data_asset_name and returns a list of valid partition_id strings that can be used to identify batches of data.

Parameters *data_asset_name* – the data asset whose partitions should be returned.

Returns A list of partition_id strings

Package Contents

Classes

`DatabricksTableBatchKwargGenerator`(*name='default'*, *datasource=None*, *database='default'*) to be used in a Databricks notebook

`GlobReaderBatchKwargGenerator`(*name='default'*, *datasource=None*, *base_directory='/data'*, *reader_options=None*, *asset_globs=None*, *reader_method=None*) GlobReaderBatchKwargGenerator processes files in a directory according to glob patterns to produce batches of data.

continues on next page

Table 78 – continued from previous page

<i>ManualBatchKwargsGenerator</i> (name='default', datasource=None, assets=None)	ManualBatchKwargsGenerator returns manually-configured batch_kwargs for named data assets. It provides a
<i>QueryBatchKwargsGenerator</i> (name='default', datasource=None, query_store_backend=None, queries=None)	Produce query-style batch_kwargs from sql files stored on disk
<i>S3GlobReaderBatchKwargsGenerator</i> (name='default', datasource=None, bucket=None, reader_options=None, assets=None, delimiter='/', reader_method=None, boto3_options=None, max_keys=1000)	S3BatchKwargsGenerator provides support for generating batches of data from an S3 bucket. For the S3 batch kwargs generator, assets must
<i>SubdirReaderBatchKwargsGenerator</i> (name='default', datasource=None, base_directory='/data', reader_options=None, known_extensions=None, reader_method=None)	The SubdirReaderBatchKwargsGenerator inspects a filesystem and produces path-based batch_kwargs.
<i>TableBatchKwargsGenerator</i> (name='default', datasource=None, assets=None)	Provide access to already materialized tables or views in a database.

class great_expectations.datasource.batch_kwargs_generator.DatabricksTableBatchKwargsGenerator

Bases: `great_expectations.datasource.batch_kwargs_generator.batch_kwargs_generator.BatchKwargsGenerator`

Meant to be used in a Databricks notebook

get_available_data_asset_names (*self*)

Return the list of asset names known by this batch kwargs generator.

Returns A list of available names

_get_iterator (*self*, *data_asset_name*, ***kwargs*)

class great_expectations.datasource.batch_kwargs_generator.GlobReaderBatchKwargsGenerator

Bases: `great_expectations.datasource.batch_kwargs_generator.batch_kwargs_generator.BatchKwargsGenerator`

GlobReaderBatchKwargsGenerator processes files in a directory according to glob patterns to produce batches of data.

A more interesting asset_glob might look like the following:

```
daily_logs:
  glob: daily_logs/*.csv
  partition_regex: daily_logs/((19|20)\d\d[- /.](0[1-9]|1[012])[- /.](0[1-9]|1[0-9]|12)[0-9]|3[01]))_(.*)\.csv
```

The “glob” key ensures that every csv file in the daily_logs directory is considered a batch for this data asset. The “partition_regex” key ensures that files whose basename begins with a date (with components hyphen, space,

forward slash, period, or null separated) will be identified by a `partition_id` equal to just the date portion of their name.

A fully configured `GlobReaderBatchKwargsGenerator` in `yaml` might look like the following:

```
my_datasource:
  class_name: PandasDatasource
  batch_kwargs_generators:
    my_generator:
      class_name: GlobReaderBatchKwargsGenerator
      base_directory: /var/log
      reader_options:
        sep: %
        header: 0
      reader_method: csv
      asset_globs:
        wifi_logs:
          glob: wifi*.log
          partition_regex: wifi-((0[1-9]|1[012])-(0[1-9]|12)[0-9]|3[01])-20\d\d)
          ↪*.log
      reader_method: csv
```

recognized_batch_parameters

property `reader_options` (*self*)

property `asset_globs` (*self*)

property `reader_method` (*self*)

property `base_directory` (*self*)

get_available_data_asset_names (*self*)

Return the list of asset names known by this batch kwargs generator.

Returns A list of available names

get_available_partition_ids (*self*, *generator_asset=None*, *data_asset_name=None*)

Applies the current `_partitioner` to the batches available on `data_asset_name` and returns a list of valid `partition_id` strings that can be used to identify batches of data.

Parameters `data_asset_name` – the data asset whose partitions should be returned.

Returns A list of `partition_id` strings

_build_batch_kwargs (*self*, *batch_parameters*)

_get_data_asset_paths (*self*, *data_asset_name*)

Returns a list of filepaths associated with the given `data_asset_name`

Parameters `data_asset_name` –

Returns paths (list)

_get_data_asset_config (*self*, *data_asset_name*)

_get_iterator (*self*, *data_asset_name*, *reader_method=None*, *reader_options=None*, *limit=None*)

_build_batch_kwargs_path_iter (*self*, *path_list*, *glob_config*, *reader_method=None*, *reader_options=None*, *limit=None*)

_build_batch_kwargs_from_path (*self*, *path*, *glob_config*, *reader_method=None*, *reader_options=None*, *limit=None*)

_partitioner (*self*, *path*, *glob_config*)

class great_expectations.datasource.batch_kwargs_generator.**ManualBatchKwargsGenerator** (name=
data-
source=
as-
sets=N

Bases: `great_expectations.datasource.batch_kwargs_generator.
batch_kwargs_generator.BatchKwargsGenerator`

ManualBatchKwargsGenerator returns manually-configured batch_kwargs for named data assets. It provides a convenient way to capture complete batch definitions without requiring the configuration of a more fully-featured batch kwargs generator.

A fully configured ManualBatchKwargsGenerator in yml might look like the following:

```
my_datasource:
  class_name: PandasDatasource
  batch_kwargs_generators:
    my_generator:
      class_name: ManualBatchKwargsGenerator
      assets:
        asset1:
          - partition_id: 1
            path: /data/file_1.csv
            reader_options:
              sep: ;
          - partition_id: 2
            path: /data/file_2.csv
            reader_options:
              header: 0
      logs:
        path: data/log.csv
```

recognized_batch_parameters

property assets (self)

get_available_data_asset_names (self)

Return the list of asset names known by this batch kwargs generator.

Returns A list of available names

_get_data_asset_config (self, data_asset_name)

_get_iterator (self, data_asset_name, **kwargs)

get_available_partition_ids (self, generator_asset=None, data_asset_name=None)

Applies the current _partitioner to the batches available on data_asset_name and returns a list of valid partition_id strings that can be used to identify batches of data.

Parameters **data_asset_name** – the data asset whose partitions should be returned.

Returns A list of partition_id strings

_build_batch_kwargs (self, batch_parameters)

Build batch kwargs from a partition id.

class great_expectations.datasource.batch_kwargs_generator.**QueryBatchKwargsGenerator** (name=
data-
source=
query_s
queries=

Bases: `great_expectations.datasource.batch_kwargs_generator.batch_kwargs_generator.BatchKwargsGenerator`

Produce query-style batch_kwargs from sql files stored on disk

recognized_batch_parameters

_get_raw_query (*self*, *data_asset_name*)

_get_iterator (*self*, *data_asset_name*, *query_parameters=None*)

add_query (*self*, *generator_asset=None*, *query=None*, *data_asset_name=None*)

get_available_data_asset_names (*self*)

Return the list of asset names known by this batch kwargs generator.

Returns A list of available names

_build_batch_kwargs (*self*, *batch_parameters*)

Build batch kwargs from a partition id.

get_available_partition_ids (*self*, *generator_asset=None*, *data_asset_name=None*)

Applies the current _partitioner to the batches available on data_asset_name and returns a list of valid partition_id strings that can be used to identify batches of data.

Parameters *data_asset_name* – the data asset whose partitions should be returned.

Returns A list of partition_id strings

class `great_expectations.datasource.batch_kwargs_generator.S3GlobReaderBatchKwargsGenerator`

Bases: `great_expectations.datasource.batch_kwargs_generator.batch_kwargs_generator.BatchKwargsGenerator`

S3 BatchKwargGenerator provides support for generating batches of data from an S3 bucket. For the S3 batch kwargs generator, assets must be individually defined using a prefix and glob, although several additional configuration parameters are available for assets (see below).

Example configuration:

```
datasources:
  my_datasource:
    ...
    batch_kwargs_generator:
      my_s3_generator:
        class_name: S3GlobReaderBatchKwargsGenerator
        bucket: my_bucket.my_organization.priv
        reader_method: parquet # This will be automatically inferred from suffix,
        ↪where possible, but can be explicitly specified as well
        reader_options: # Note that reader options can be specified globally or
        ↪per-asset
```

(continues on next page)

(continued from previous page)

```

        sep: ","
        delimiter: "/" # Note that this is the delimiter for the BUCKET KEYS. By_
        ↪ default it is "/"
        boto3_options:
            endpoint_url: $S3_ENDPOINT # Use the S3_ENDPOINT environment variable_
        ↪ to determine which endpoint to use
            max_keys: 100 # The maximum number of keys to fetch in a single list_
        ↪ objects request to s3. When accessing batch_kwarg through an iterator, the_
        ↪ iterator will silently refetch if more keys were available
        assets:
            my_first_asset:
                prefix: my_first_asset/
                regex_filter: .* # The regex filter will filter the results returned_
        ↪ by S3 for the key and prefix to only those matching the regex
                directory_assets: True
            access_logs:
                prefix: access_logs
                regex_filter: access_logs/2019.*\.csv.gz
                sep: "~"
                max_keys: 100

```

recognized_batch_parameters**property_reader_options** (*self*)**property_assets** (*self*)**property_bucket** (*self*)**get_available_data_asset_names** (*self*)

Return the list of asset names known by this batch kwarg generator.

Returns A list of available names**_get_iterator** (*self*, *data_asset_name*, *reader_method=None*, *reader_options=None*, *limit=None*)**_build_batch_kwarg_path_iter** (*self*, *path_list*, *reader_options=None*, *limit=None*)**_build_batch_kwarg** (*self*, *batch_parameters*)**_build_batch_kwarg_from_key** (*self*, *key*, *asset_config=None*, *reader_method=None*, *reader_options=None*, *limit=None*)**_get_asset_options** (*self*, *asset_config*, *iterator_dict*)**_build_asset_iterator** (*self*, *asset_config*, *iterator_dict*, *reader_method=None*, *reader_options=None*, *limit=None*)**get_available_partition_ids** (*self*, *generator_asset=None*, *data_asset_name=None*)Applies the current `_partitioner` to the batches available on `data_asset_name` and returns a list of valid `partition_id` strings that can be used to identify batches of data.**Parameters** `data_asset_name` – the data asset whose partitions should be returned.**Returns** A list of `partition_id` strings**_partitioner** (*self*, *key*, *asset_config*)

```
class great_expectations.datasource.batch_kwargs_generator.SubdirReaderBatchKwargsGenerator
```

Bases: `great_expectations.datasource.batch_kwargs_generator.batch_kwargs_generator.BatchKwargsGenerator`

The SubdirReaderBatchKwargsGenerator inspects a filesystem and produces path-based batch_kwargs.

SubdirReaderBatchKwargsGenerator recognizes data assets using two criteria:

- for files directly in 'base_directory' with recognized extensions (.csv, .tsv, .parquet, .xls, .xlsx, .json), it uses the name of the file without the extension
- for other files or directories in 'base_directory', it uses the file or directory name

SubdirReaderBatchKwargsGenerator sees all files inside a directory of base_directory as batches of one data-source.

SubdirReaderBatchKwargsGenerator can also include configured reader_options which will be added to batch_kwargs generated by this generator.

_default_reader_options

recognized_batch_parameters

property reader_options (self)

property known_extensions (self)

property reader_method (self)

property base_directory (self)

get_available_data_asset_names (self)

Return the list of asset names known by this batch kwargs generator.

Returns A list of available names

get_available_partition_ids (self, generator_asset=None, data_asset_name=None)

Applies the current _partitioner to the batches available on data_asset_name and returns a list of valid partition_id strings that can be used to identify batches of data.

Parameters data_asset_name – the data asset whose partitions should be returned.

Returns A list of partition_id strings

_build_batch_kwargs (self, batch_parameters)

Parameters batch_parameters –

Returns batch_kwargs

_get_valid_file_options (self, base_directory=None)

_get_iterator (self, data_asset_name, reader_options=None, limit=None)

_build_batch_kwargs_path_iter (self, path_list, reader_options=None, limit=None)

_build_batch_kwargs_from_path (self, path, reader_method=None, reader_options=None, limit=None)

class great_expectations.datasource.batch_kwarg_generator.**TableBatchKwargGenerator** (*name='data-source-as-sets=None'*)

Bases: `great_expectations.datasource.batch_kwarg_generator.batch_kwarg_generator.BatchKwargGenerator`

Provide access to already materialized tables or views in a database.

TableBatchKwargGenerator can be used to define specific data asset names that take and substitute parameters, for example to support referring to the same data asset but with different schemas depending on provided batch_kwarg.

The python template language is used to substitute table name portions. For example, consider the following configurations:

```
my_generator:
  class_name: TableBatchKwargGenerator
  assets:
    my_table:
      schema: $schema
      table: my_table
```

In that case, the asset my_datasource/my_generator/my_asset will refer to a table called my_table in a schema defined in batch_kwarg.

recognized_batch_parameters

_get_iterator (*self, data_asset_name, query_parameters=None, limit=None, offset=None, partition_id=None*)

get_available_data_asset_names (*self*)

Return the list of asset names known by this batch kwarg generator.

Returns A list of available names

_build_batch_kwarg (*self, batch_parameters*)

get_available_partition_ids (*self, generator_asset=None, data_asset_name=None*)

Applies the current _partitioner to the batches available on data_asset_name and returns a list of valid partition_id strings that can be used to identify batches of data.

Parameters **data_asset_name** – the data asset whose partitions should be returned.

Returns A list of partition_id strings

great_expectations.datasource.types

Submodules

great_expectations.datasource.types.batch_kwarg

Module Contents

Classes

<code>BatchMarkers(*args, **kwargs)</code>	A BatchMarkers is a special type of BatchKwargs (so that it has a batch_fingerprint) but it generally does
<code>PandasDatasourceBatchKwargs()</code>	This is an abstract class and should not be instantiated. It's relevant for testing whether
<code>SparkDFDatasourceBatchKwargs()</code>	This is an abstract class and should not be instantiated. It's relevant for testing whether
<code>SqlAlchemyDatasourceBatchKwargs()</code>	This is an abstract class and should not be instantiated. It's relevant for testing whether
<code>PathBatchKwargs(*args, **kwargs)</code>	This is an abstract class and should not be instantiated. It's relevant for testing whether
<code>S3BatchKwargs(*args, **kwargs)</code>	This is an abstract class and should not be instantiated. It's relevant for testing whether
<code>InMemoryBatchKwargs(*args, **kwargs)</code>	This is an abstract class and should not be instantiated. It's relevant for testing whether
<code>PandasDatasourceInMemoryBatchKwargs(*args, **kwargs)</code>	This is an abstract class and should not be instantiated. It's relevant for testing whether
<code>SparkDFDatasourceInMemoryBatchKwargs(*args, **kwargs)</code>	This is an abstract class and should not be instantiated. It's relevant for testing whether
<code>SqlAlchemyDatasourceTableBatchKwargs(*args, **kwargs)</code>	This is an abstract class and should not be instantiated. It's relevant for testing whether
<code>SqlAlchemyDatasourceQueryBatchKwargs(*args, **kwargs)</code>	This is an abstract class and should not be instantiated. It's relevant for testing whether
<code>SparkDFDatasourceQueryBatchKwargs(*args, **kwargs)</code>	This is an abstract class and should not be instantiated. It's relevant for testing whether

`great_expectations.datasource.types.batch_kwargs.logger`

class `great_expectations.datasource.types.batch_kwargs.BatchMarkers` (**args*,
***kwargs*)

Bases: `great_expectations.core.id_dict.BatchKwargs`

A BatchMarkers is a special type of BatchKwargs (so that it has a batch_fingerprint) but it generally does NOT require specific keys and instead captures information about the OUTPUT of a datasource's fetch process, such as the timestamp at which a query was executed.

property `ge_load_time` (*self*)

class `great_expectations.datasource.types.batch_kwargs.PandasDatasourceBatchKwargs`

Bases: `great_expectations.core.id_dict.BatchKwargs`

This is an abstract class and should not be instantiated. It's relevant for testing whether a subclass is allowed

class `great_expectations.datasource.types.batch_kwargs.SparkDFDatasourceBatchKwargs`

Bases: `great_expectations.core.id_dict.BatchKwargs`

This is an abstract class and should not be instantiated. It's relevant for testing whether a subclass is allowed

class `great_expectations.datasource.types.batch_kwargs.SqlAlchemyDatasourceBatchKwargs`

Bases: `great_expectations.core.id_dict.BatchKwargs`

This is an abstract class and should not be instantiated. It's relevant for testing whether a subclass is allowed

property `limit` (*self*)

property `schema` (*self*)

class `great_expectations.datasource.types.batch_kwargs.PathBatchKwargs` (**args*,
***kwargs*)

Bases: `great_expectations.datasource.types.batch_kwargs.`

```
PandasDatasourceBatchKwargs, great_expectations.datasource.types.  
batch_kwargs.SparkDFDatasourceBatchKwargs
```

This is an abstract class and should not be instantiated. It's relevant for testing whether a subclass is allowed

property `path` (*self*)

property `reader_method` (*self*)

```
class great_expectations.datasource.types.batch_kwargs.S3BatchKwargs (*args,  
                                                                    **kwargs)  
  
Bases:  
    great_expectations.datasource.types.batch_kwargs.  
    PandasDatasourceBatchKwargs, great_expectations.datasource.types.  
    batch_kwargs.SparkDFDatasourceBatchKwargs
```

This is an abstract class and should not be instantiated. It's relevant for testing whether a subclass is allowed

property `s3` (*self*)

property `reader_method` (*self*)

```
class great_expectations.datasource.types.batch_kwargs.InMemoryBatchKwargs (*args,  
                                                                              **kwargs)  
  
Bases:  
    great_expectations.datasource.types.batch_kwargs.  
    PandasDatasourceBatchKwargs, great_expectations.datasource.types.  
    batch_kwargs.SparkDFDatasourceBatchKwargs
```

This is an abstract class and should not be instantiated. It's relevant for testing whether a subclass is allowed

property `dataset` (*self*)

```
class great_expectations.datasource.types.batch_kwargs.PandasDatasourceInMemoryBatchKwargs  
  
Bases: great_expectations.datasource.types.batch_kwargs.InMemoryBatchKwargs
```

This is an abstract class and should not be instantiated. It's relevant for testing whether a subclass is allowed

```
class great_expectations.datasource.types.batch_kwargs.SparkDFDatasourceInMemoryBatchKwargs  
  
Bases: great_expectations.datasource.types.batch_kwargs.InMemoryBatchKwargs
```

This is an abstract class and should not be instantiated. It's relevant for testing whether a subclass is allowed

```
class great_expectations.datasource.types.batch_kwargs.SqlAlchemyDatasourceTableBatchKwargs  
  
Bases:  
    great_expectations.datasource.types.batch_kwargs.  
    SqlAlchemyDatasourceBatchKwargs
```

This is an abstract class and should not be instantiated. It's relevant for testing whether a subclass is allowed

property `table` (*self*)

```
class great_expectations.datasource.types.batch_kwargs.SqlAlchemyDatasourceQueryBatchKwargs  
  
Bases:  
    great_expectations.datasource.types.batch_kwargs.  
    SqlAlchemyDatasourceBatchKwargs
```

This is an abstract class and should not be instantiated. It's relevant for testing whether a subclass is allowed

property `query` (*self*)

property `query_parameters` (*self*)


```
class great_expectations.datasource.types.batch_kwargs.SparkDFDatasourceQueryBatchKwargs (*args, **kwargs)
```

Bases: `great_expectations.datasource.types.batch_kwargs.SparkDFDatasourceBatchKwargs`

This is an abstract class and should not be instantiated. It's relevant for testing whether a subclass is allowed

property `query` (*self*)

`great_expectations.datasource.types.reader_methods`

Package Contents

Classes

<code>BatchKwargs()</code>	<code>dict()</code> -> new empty dictionary
<code>BatchMarkers(*args, **kwargs)</code>	A BatchMarkers is a special type of BatchKwargs (so that it has a <code>batch_fingerprint</code>) but it generally does
<code>PandasDatasourceBatchKwargs()</code>	This is an abstract class and should not be instantiated. It's relevant for testing whether
<code>SparkDFDatasourceBatchKwargs()</code>	This is an abstract class and should not be instantiated. It's relevant for testing whether
<code>SqlAlchemyDatasourceBatchKwargs()</code>	This is an abstract class and should not be instantiated. It's relevant for testing whether
<code>PathBatchKwargs(*args, **kwargs)</code>	This is an abstract class and should not be instantiated. It's relevant for testing whether
<code>S3BatchKwargs(*args, **kwargs)</code>	This is an abstract class and should not be instantiated. It's relevant for testing whether
<code>InMemoryBatchKwargs(*args, **kwargs)</code>	This is an abstract class and should not be instantiated. It's relevant for testing whether
<code>PandasDatasourceInMemoryBatchKwargs(*args, **kwargs)</code>	This is an abstract class and should not be instantiated. It's relevant for testing whether
<code>SparkDFDatasourceInMemoryBatchKwargs(*args, **kwargs)</code>	This is an abstract class and should not be instantiated. It's relevant for testing whether
<code>SqlAlchemyDatasourceTableBatchKwargs(*args, **kwargs)</code>	This is an abstract class and should not be instantiated. It's relevant for testing whether
<code>SqlAlchemyDatasourceQueryBatchKwargs(*args, **kwargs)</code>	This is an abstract class and should not be instantiated. It's relevant for testing whether
<code>SparkDFDatasourceQueryBatchKwargs(*args, **kwargs)</code>	This is an abstract class and should not be instantiated. It's relevant for testing whether

```
class great_expectations.datasource.types.BatchKwargs
```

Bases: `great_expectations.core.id_dict.IDDict`

`dict()` -> new empty dictionary
`dict(mapping)` -> new dictionary initialized from a mapping object's (key, value) pairs

`dict(iterable)` -> new dictionary initialized as if via: `d = {}` for `k, v` in iterable:

`d[k] = v`

`dict(kwargs)` -> new dictionary initialized with the name=value pairs** in the keyword argument list. For example: `dict(one=1, two=2)`

exception `great_expectations.datasource.types.InvalidBatchIdError` (*message*)
Bases: `great_expectations.exceptions.GreatExpectationsError`

Common base class for all non-exit exceptions.

exception `great_expectations.datasource.types.InvalidBatchKwargsError` (*message*)
Bases: `great_expectations.exceptions.GreatExpectationsError`

Common base class for all non-exit exceptions.

`great_expectations.datasource.types.logger`

class `great_expectations.datasource.types.BatchMarkers` (**args, **kwargs*)
Bases: `great_expectations.core.id_dict.BatchKwargs`

A BatchMarkers is a special type of BatchKwargs (so that it has a batch_fingerprint) but it generally does NOT require specific keys and instead captures information about the OUTPUT of a datasource's fetch process, such as the timestamp at which a query was executed.

property `ge_load_time` (*self*)

class `great_expectations.datasource.types.PandasDatasourceBatchKwargs`
Bases: `great_expectations.core.id_dict.BatchKwargs`

This is an abstract class and should not be instantiated. It's relevant for testing whether a subclass is allowed

class `great_expectations.datasource.types.SparkDFDatasourceBatchKwargs`
Bases: `great_expectations.core.id_dict.BatchKwargs`

This is an abstract class and should not be instantiated. It's relevant for testing whether a subclass is allowed

class `great_expectations.datasource.types.SqlAlchemyDatasourceBatchKwargs`
Bases: `great_expectations.core.id_dict.BatchKwargs`

This is an abstract class and should not be instantiated. It's relevant for testing whether a subclass is allowed

property `limit` (*self*)

property `schema` (*self*)

class `great_expectations.datasource.types.PathBatchKwargs` (**args, **kwargs*)
Bases: `great_expectations.datasource.types.batch_kwargs.PandasDatasourceBatchKwargs`, `great_expectations.datasource.types.batch_kwargs.SparkDFDatasourceBatchKwargs`

This is an abstract class and should not be instantiated. It's relevant for testing whether a subclass is allowed

property `path` (*self*)

property `reader_method` (*self*)

class `great_expectations.datasource.types.S3BatchKwargs` (**args, **kwargs*)
Bases: `great_expectations.datasource.types.batch_kwargs.PandasDatasourceBatchKwargs`, `great_expectations.datasource.types.batch_kwargs.SparkDFDatasourceBatchKwargs`

This is an abstract class and should not be instantiated. It's relevant for testing whether a subclass is allowed

property `s3` (*self*)

property `reader_method` (*self*)

class `great_expectations.datasource.types.InMemoryBatchKwargs` (**args, **kwargs*)
Bases: `great_expectations.datasource.types.batch_kwargs.PandasDatasourceBatchKwargs`, `great_expectations.datasource.types.batch_kwargs.SparkDFDatasourceBatchKwargs`

This is an abstract class and should not be instantiated. It's relevant for testing whether a subclass is allowed

property dataset (*self*)

```
class great_expectations.datasource.types.PandasDatasourceInMemoryBatchKwargs (*args,  
                                                                           **kwargs)
```

Bases: *great_expectations.datasource.types.batch_kwargs.InMemoryBatchKwargs*

This is an abstract class and should not be instantiated. It's relevant for testing whether a subclass is allowed

```
class great_expectations.datasource.types.SparkDFDatasourceInMemoryBatchKwargs (*args,  
                                                                           **kwargs)
```

Bases: *great_expectations.datasource.types.batch_kwargs.InMemoryBatchKwargs*

This is an abstract class and should not be instantiated. It's relevant for testing whether a subclass is allowed

```
class great_expectations.datasource.types.SqlAlchemyDatasourceTableBatchKwargs (*args,  
                                                                           **kwargs)
```

Bases: *great_expectations.datasource.types.batch_kwargs.SqlAlchemyDatasourceBatchKwargs*

This is an abstract class and should not be instantiated. It's relevant for testing whether a subclass is allowed

property table (*self*)

```
class great_expectations.datasource.types.SqlAlchemyDatasourceQueryBatchKwargs (*args,  
                                                                           **kwargs)
```

Bases: *great_expectations.datasource.types.batch_kwargs.SqlAlchemyDatasourceBatchKwargs*

This is an abstract class and should not be instantiated. It's relevant for testing whether a subclass is allowed

property query (*self*)

property query_parameters (*self*)

```
class great_expectations.datasource.types.SparkDFDatasourceQueryBatchKwargs (*args,  
                                                                           **kwargs)
```

Bases: *great_expectations.datasource.types.batch_kwargs.SparkDFDatasourceBatchKwargs*

This is an abstract class and should not be instantiated. It's relevant for testing whether a subclass is allowed

property query (*self*)

Submodules

`great_expectations.datasource.datasource`

Module Contents

Classes

<code>Datasource</code> (name, data_asset_type=None, batch_kwargs_generators=None, **kwargs)	A Datasource connects to a compute environment and one or more storage environments and produces batches of data
---	--

`great_expectations.datasource.datasource.logger`

`great_expectations.datasource.datasource.yaml`

```
great_expectations.datasource.datasource.default_flow_style = False
```

```
class great_expectations.datasource.datasource.Datasource(name,
                                                           data_context=None,
                                                           data_asset_type=None,
                                                           batch_kwargs_generators=None,
                                                           **kwargs)
```

Bases: `object`

A Datasource connects to a compute environment and one or more storage environments and produces batches of data that Great Expectations can validate in that compute environment.

Each Datasource provides Batches connected to a specific compute environment, such as a SQL database, a Spark cluster, or a local in-memory Pandas DataFrame.

Datasources use Batch Kwargs to specify instructions for how to access data from relevant sources such as an existing object from a DAG runner, a SQL database, S3 bucket, or local filesystem.

To bridge the gap between those worlds, Datasources interact closely with *generators* which are aware of a source of data and can produce identifying information, called “batch_kwargs” that datasources can use to get individual batches of data. They add flexibility in how to obtain data such as with time-based partitioning, downsampling, or other techniques appropriate for the datasource.

For example, a batch kwargs generator could produce a SQL query that logically represents “rows in the Events table with a timestamp on February 7, 2012,” which a SQLAlchemyDatasource could use to materialize a SQLAlchemyDataset corresponding to that batch of data and ready for validation.

Since opinionated DAG managers such as airflow, dbt, prefect.io, dagster can also act as datasources and/or batch kwargs generators for a more generic datasource.

When adding custom expectations by subclassing an existing DataAsset type, use the `data_asset_type` parameter to configure the datasource to load and return DataAssets of the custom type.

recognized_batch_parameters

```
classmethod from_configuration(cls, **kwargs)
```

Build a new datasource from a configuration dictionary.

Parameters ****kwargs** – configuration key-value pairs

Returns the newly-created datasource

Return type `datasource` (*Datasource*)

```
classmethod build_configuration(cls, class_name, module_name='great_expectations.datasource',
                               data_asset_type=None, batch_kwargs_generators=None,
                               **kwargs)
```

Build a full configuration object for a datasource, potentially including batch kwargs generators with defaults.

Parameters

- **class_name** – The name of the class for which to build the config
- **module_name** – The name of the module in which the datasource class is located
- **data_asset_type** – A ClassConfig dictionary
- **batch_kwargs_generators** – BatchKwargsGenerators configuration dictionary
- ****kwargs** – Additional kwargs to be part of the datasource constructor’s initialization

Returns A complete datasource configuration.

property name (*self*)

Property for datasource name

property config (*self*)

property data_context (*self*)

Property for attached DataContext

_build_generators (*self*)

Build batch kwargs generator objects from the datasource configuration.

Returns None

add_batch_kwargs_generator (*self*, *name*, *class_name*, ***kwargs*)

Add a BatchKwargsGenerator to the datasource.

Parameters

- **name** (*str*) – the name of the new BatchKwargsGenerator to add
- **class_name** – class of the BatchKwargsGenerator to add
- **kwargs** – additional keyword arguments will be passed directly to the new BatchKwargsGenerator’s constructor

Returns BatchKwargsGenerator (BatchKwargsGenerator)

_build_batch_kwargs_generator (*self*, ***kwargs*)

Build a BatchKwargsGenerator using the provided configuration and return the newly-built generator.

get_batch_kwargs_generator (*self*, *name*)

Get the (named) BatchKwargsGenerator from a datasource)

Parameters **name** (*str*) – name of BatchKwargsGenerator (default value is ‘default’)

Returns BatchKwargsGenerator (BatchKwargsGenerator)

list_batch_kwargs_generators (*self*)

List currently-configured BatchKwargsGenerator for this datasource.

Returns each dictionary includes “name” and “type” keys

Return type List(dict)

process_batch_parameters (*self*, *limit=None*, *dataset_options=None*)

Use datasource-specific configuration to translate any batch parameters into batch kwargs at the datasource level.

Parameters

- **limit** (*int*) – a parameter all datasources must accept to allow limiting a batch to a smaller number of rows.
- **dataset_options** (*dict*) – a set of kwargs that will be passed to the constructor of a dataset built using these batch_kwargs

Returns Result will include both parameters passed via argument and configured parameters.

Return type batch_kwargs

abstract get_batch (*self*, *batch_kwargs*, *batch_parameters=None*)

Get a batch of data from the datasource.

Parameters

- **batch_kwargs** – the BatchKwargs to use to construct the batch

- **batch_parameters** – optional parameters to store as the reference description of the batch. They should reflect parameters that would provide the passed BatchKwargs.

Returns Batch

get_available_data_asset_names (*self*, *batch_kwargs_generator_names=None*)

Returns a dictionary of data_asset_names that the specified batch kwarg generator can provide. Note that some batch kwargs generators may not be capable of describing specific named data assets, and some (such as filesystem glob batch kwargs generators) require the user to configure data asset names.

Parameters **batch_kwargs_generator_names** – the BatchKwargGenerator for which to get available data asset names.

Returns

```
{
    generator_name: {
        names: [ (data_asset_1, data_asset_1_type), (data_asset_2, data_
↪asset_2_type) ... ]
    }
    ...
}
```

Return type dictionary consisting of sets of generator assets available for the specified generators

build_batch_kwargs (*self*, *batch_kwargs_generator*, *data_asset_name=None*, *partition_id=None*, ***kwargs*)

`great_expectations.datasource.pandas_datasource`

Module Contents

Classes

<code>PandasDatasource</code> (<i>name='pandas'</i> , <i>data_context=None</i> , <i>data_asset_type=None</i> , <i>batch_kwargs_generators=None</i> , <i>boto3_options=None</i> , <i>reader_method=None</i> , <i>reader_options=None</i> , <i>limit=None</i> , <i>**kwargs</i>)	The PandasDatasource produces PandasDataset objects and supports generators capable of
--	--

`great_expectations.datasource.pandas_datasource.logger`

`great_expectations.datasource.pandas_datasource.HASH_THRESHOLD = 1000000000.0`

class `great_expectations.datasource.pandas_datasource.PandasDatasource` (*name='pandas'*,
data_context=None,
data_asset_type=None,
batch_kwargs_generators=None,
boto3_options=None,
reader_method=None,
reader_options=None,
limit=None,
***kwargs*)

Bases: `great_expectations.datasource.datasource.Datasource`

The PandasDatasource produces PandasDataset objects and supports generators capable of interacting with the local filesystem (the default subdir_reader generator), and from existing in-memory dataframes.

recognized_batch_parameters

classmethod build_configuration (*cls*, *data_asset_type=None*,
batch_kwargs_generators=None, *boto3_options=None*,
reader_method=None, *reader_options=None*,
limit=None, ***kwargs*)

Build a full configuration object for a datasource, potentially including generators with defaults.

Parameters

- **data_asset_type** – A ClassConfig dictionary
- **batch_kwargs_generators** – Generator configuration dictionary
- **boto3_options** – Optional dictionary with key-value pairs to pass to boto3 during instantiation.
- **reader_method** – Optional default reader_method for generated batches
- **reader_options** – Optional default reader_options for generated batches
- **limit** – Optional default limit for generated batches
- ****kwargs** – Additional kwargs to be part of the datasource constructor's initialization

Returns A complete datasource configuration.

process_batch_parameters (*self*, *reader_method=None*, *reader_options=None*, *limit=None*,
dataset_options=None)

Use datasource-specific configuration to translate any batch parameters into batch kwargs at the datasource level.

Parameters

- **limit** (*int*) – a parameter all datasources must accept to allow limiting a batch to a smaller number of rows.
- **dataset_options** (*dict*) – a set of kwargs that will be passed to the constructor of a dataset built using these batch_kwargs

Returns Result will include both parameters passed via argument and configured parameters.

Return type batch_kwargs

get_batch (*self*, *batch_kwargs*, *batch_parameters=None*)

Get a batch of data from the datasource.

Parameters

- **batch_kwargs** – the BatchKwargs to use to construct the batch
- **batch_parameters** – optional parameters to store as the reference description of the batch. They should reflect parameters that would provide the passed BatchKwargs.

Returns Batch

static guess_reader_method_from_path (*path*)

_get_reader_fn (*self*, *reader_method=None*, *path=None*)

Static helper for parsing reader types. If reader_method is not provided, path will be used to guess the correct reader_method.

Parameters

- **reader_method** (*str*) – the name of the reader method to use, if available.
- **path** (*str*) – the to use to guess

Returns ReaderMethod to use for the filepath

`great_expectations.datasource.sparkdf_datasource`

Module Contents

Classes

<code>SparkDFDatasource</code> (<i>name</i> ='default', <i>data_context</i> =None, <i>data_asset_type</i> =None, <i>batch_kwargs_generators</i> =None, <i>spark_config</i> =None, ** <i>kwargs</i>)	The SparkDFDatasource produces SparkDFDatasets and supports generators capable of interacting with local
--	--

`great_expectations.datasource.sparkdf_datasource.logger`

`great_expectations.datasource.sparkdf_datasource.SparkSession`

class `great_expectations.datasource.sparkdf_datasource.SparkDFDatasource` (*name*='default',
data_context=None,
data_asset_type=None,
batch_kwargs_generators
spark_config=None,
***kwargs*)

Bases: `great_expectations.datasource.datasource.Datasource`

The SparkDFDatasource produces SparkDFDatasets and supports generators capable of interacting with local filesystem (the default `subdir_reader` batch kwargs generator) and databricks notebooks.

Accepted Batch Kwargs:

- PathBatchKwargs (“path” or “s3” keys)
- InMemoryBatchKwargs (“dataset” key)
- QueryBatchKwargs (“query” key)

recognized_batch_parameters

classmethod `build_configuration` (*cls*,
data_asset_type=None,
batch_kwargs_generators=None, *spark_config*=None,
***kwargs*)

Build a full configuration object for a datasource, potentially including generators with defaults.

Parameters

- **data_asset_type** – A ClassConfig dictionary
- **batch_kwargs_generators** – Generator configuration dictionary
- **spark_config** – dictionary of key-value pairs to pass to the spark builder
- ****kwargs** – Additional kwargs to be part of the datasource constructor’s initialization

Returns A complete datasource configuration.

process_batch_parameters (*self*, *reader_method=None*, *reader_options=None*, *limit=None*, *dataset_options=None*)

Use datasource-specific configuration to translate any batch parameters into batch kwargs at the datasource level.

Parameters

- **limit** (*int*) – a parameter all datasources must accept to allow limiting a batch to a smaller number of rows.
- **dataset_options** (*dict*) – a set of kwargs that will be passed to the constructor of a dataset built using these batch_kwargs

Returns Result will include both parameters passed via argument and configured parameters.

Return type batch_kwargs

get_batch (*self*, *batch_kwargs*, *batch_parameters=None*)

class-private implementation of get_data_asset

static guess_reader_method_from_path (*path*)

_get_reader_fn (*self*, *reader*, *reader_method=None*, *path=None*)

Static helper for providing reader_fn

Parameters

- **reader** – the base spark reader to use; this should have had reader_options applied already
- **reader_method** – the name of the reader_method to use, if specified
- **path** (*str*) – the path to use to guess reader_method if it was not specified

Returns ReaderMethod to use for the filepath

`great_expectations.datasource.sqlalchemy_datasource`

Module Contents

Classes

<code>SqlAlchemyDatasource</code> (<i>name='default'</i> , <i>data_context=None</i> , <i>data_asset_type=None</i> , <i>credentials=None</i> , <i>batch_kwargs_generators=None</i> , <i>**kwargs</i>)	A SqlAlchemyDatasource will provide data_assets converting batch_kwargs using the following rules:
---	--

`great_expectations.datasource.sqlalchemy_datasource.logger`

`great_expectations.datasource.sqlalchemy_datasource.sqlalchemy`

`great_expectations.datasource.sqlalchemy_datasource.datasource_initialization_exceptions`

```
class great_expectations.datasource.sqlalchemy_datasource.SqlAlchemyDatasource (name='default',
                                         data_context=None,
                                         data_asset_type=None,
                                         credentials=None,
                                         batch_kwargs_generator=None,
                                         **kwargs)
```

Bases: `great_expectations.datasource.Datasource`

A SqlAlchemyDatasource will provide data_assets converting batch_kwargs using the following rules: - if the batch_kwargs include a table key, the datasource will provide a dataset object connected

to that table

- if the batch_kwargs include a query key, the datasource will create a temporary table using that that query. The query can be parameterized according to the standard python Template engine, which uses \$parameter, with additional kwargs passed to the get_batch method.

recognized_batch_parameters

```
classmethod build_configuration (cls,
                                data_asset_type=None,
                                batch_kwargs_generators=None,
                                **kwargs)
```

Build a full configuration object for a datasource, potentially including generators with defaults.

Parameters

- **data_asset_type** – A ClassConfig dictionary
- **batch_kwargs_generators** – Generator configuration dictionary
- ****kwargs** – Additional kwargs to be part of the datasource constructor’s initialization

Returns A complete datasource configuration.

```
_get_sqlalchemy_connection_options (self, **kwargs)
```

```
get_batch (self, batch_kwargs, batch_parameters=None)
```

Get a batch of data from the datasource.

Parameters

- **batch_kwargs** – the BatchKwargs to use to construct the batch
- **batch_parameters** – optional parameters to store as the reference description of the batch. They should reflect parameters that would provide the passed BatchKwargs.

Returns Batch

```
process_batch_parameters (self, query_parameters=None, limit=None, dataset_options=None)
```

Use datasource-specific configuration to translate any batch parameters into batch kwargs at the datasource level.

Parameters

- **limit** (*int*) – a parameter all datasources must accept to allow limiting a batch to a smaller number of rows.
- **dataset_options** (*dict*) – a set of kwargs that will be passed to the constructor of a dataset built using these batch_kwargs

Returns Result will include both parameters passed via argument and configured parameters.

Return type batch_kwargs

great_expectations.datasource.util

Module Contents

Classes

S3Url(url)

```
>>> s = S3Url("s3://bucket/hello/world")
```

Functions

hash_pandas_dataframe(df)

class great_expectations.datasource.util.**S3Url**(url)

Bases: object

```
>>> s = S3Url("s3://bucket/hello/world")
>>> s.bucket
'bucket'
>>> s.key
'hello/world'
>>> s.url
's3://bucket/hello/world'
```

```
>>> s = S3Url("s3://bucket/hello/world?qwe1=3#ddd")
>>> s.bucket
'bucket'
>>> s.key
'hello/world?qwe1=3#ddd'
>>> s.url
's3://bucket/hello/world?qwe1=3#ddd'
```

```
>>> s = S3Url("s3://bucket/hello/world#foo?bar=2")
>>> s.key
'hello/world#foo?bar=2'
>>> s.url
's3://bucket/hello/world#foo?bar=2'
```

property bucket(*self*)

property key(*self*)

property url(*self*)

great_expectations.datasource.util.**hash_pandas_dataframe**(df)

Package Contents

Classes

<code>Datasource(name, data_context=None, data_asset_type=None, batch_kwargs_generators=None, **kwargs)</code>	A Datasource connects to a compute environment and one or more storage environments and produces batches of data
<code>PandasDatasource(name='pandas', data_context=None, data_asset_type=None, batch_kwargs_generators=None, boto3_options=None, reader_method=None, reader_options=None, limit=None, **kwargs)</code>	The PandasDatasource produces PandasDataset objects and supports generators capable of
<code>SparkDFDatasource(name='default', data_context=None, data_asset_type=None, batch_kwargs_generators=None, spark_config=None, **kwargs)</code>	The SparkDFDatasource produces SparkDFDatasets and supports generators capable of interacting with local
<code>SqlAlchemyDatasource(name='default', data_context=None, data_asset_type=None, credentials=None, batch_kwargs_generators=None, **kwargs)</code>	A SqlAlchemyDatasource will provide data_assets converting batch_kwargs using the following rules:

```
class great_expectations.datasource.Datasource(name, data_context=None,
                                                data_asset_type=None,
                                                batch_kwargs_generators=None,
                                                **kwargs)
```

Bases: object

A Datasource connects to a compute environment and one or more storage environments and produces batches of data that Great Expectations can validate in that compute environment.

Each Datasource provides Batches connected to a specific compute environment, such as a SQL database, a Spark cluster, or a local in-memory Pandas DataFrame.

Datasources use Batch Kwarg to specify instructions for how to access data from relevant sources such as an existing object from a DAG runner, a SQL database, S3 bucket, or local filesystem.

To bridge the gap between those worlds, Datasources interact closely with *generators* which are aware of a source of data and can produce identifying information, called “batch_kwarg” that datasources can use to get individual batches of data. They add flexibility in how to obtain data such as with time-based partitioning, downsampling, or other techniques appropriate for the datasource.

For example, a batch kwarg generator could produce a SQL query that logically represents “rows in the Events table with a timestamp on February 7, 2012,” which a SqlAlchemyDatasource could use to materialize a SqlAlchemyDataset corresponding to that batch of data and ready for validation.

Since opinionated DAG managers such as airflow, dbt, prefect.io, dagster can also act as datasources and/or batch kwarg generators for a more generic datasource.

When adding custom expectations by subclassing an existing DataAsset type, use the data_asset_type parameter to configure the datasource to load and return DataAssets of the custom type.

recognized_batch_parameters

```
classmethod from_configuration(cls, **kwargs)
```

Build a new datasource from a configuration dictionary.

Parameters ****kwargs** – configuration key-value pairs

Returns the newly-created datasource

Return type datasource (*Datasource*)

```
classmethod build_configuration (cls, class_name, module_name='great_expectations.datasource',
                                  data_asset_type=None, batch_kwargs_generators=None,
                                  **kwargs)
```

Build a full configuration object for a datasource, potentially including batch kwargs generators with defaults.

Parameters

- **class_name** – The name of the class for which to build the config
- **module_name** – The name of the module in which the datasource class is located
- **data_asset_type** – A ClassConfig dictionary
- **batch_kwargs_generators** – BatchKwargGenerators configuration dictionary
- ****kwargs** – Additional kwargs to be part of the datasource constructor’s initialization

Returns A complete datasource configuration.

property name (*self*)

Property for datasource name

property config (*self*)

property data_context (*self*)

Property for attached DataContext

_build_generators (*self*)

Build batch kwargs generator objects from the datasource configuration.

Returns None

add_batch_kwargs_generator (*self*, name, class_name, **kwargs)

Add a BatchKwargGenerator to the datasource.

Parameters

- **name** (*str*) – the name of the new BatchKwargGenerator to add
- **class_name** – class of the BatchKwargGenerator to add
- **kwargs** – additional keyword arguments will be passed directly to the new BatchKwargGenerator’s constructor

Returns BatchKwargGenerator (BatchKwargGenerator)

_build_batch_kwargs_generator (*self*, **kwargs)

Build a BatchKwargGenerator using the provided configuration and return the newly-built generator.

get_batch_kwargs_generator (*self*, name)

Get the (named) BatchKwargGenerator from a datasource)

Parameters **name** (*str*) – name of BatchKwargGenerator (default value is ‘default’)

Returns BatchKwargGenerator (BatchKwargGenerator)

list_batch_kwargs_generators (*self*)

List currently-configured BatchKwargGenerator for this datasource.

Returns each dictionary includes “name” and “type” keys

Return type List(dict)

process_batch_parameters (*self*, *limit=None*, *dataset_options=None*)

Use datasource-specific configuration to translate any batch parameters into batch kwargs at the datasource level.

Parameters

- **limit** (*int*) – a parameter all datasources must accept to allow limiting a batch to a smaller number of rows.
- **dataset_options** (*dict*) – a set of kwargs that will be passed to the constructor of a dataset built using these batch_kwargs

Returns Result will include both parameters passed via argument and configured parameters.

Return type batch_kwargs

abstract get_batch (*self*, *batch_kwargs*, *batch_parameters=None*)

Get a batch of data from the datasource.

Parameters

- **batch_kwargs** – the BatchKwargs to use to construct the batch
- **batch_parameters** – optional parameters to store as the reference description of the batch. They should reflect parameters that would provide the passed BatchKwargs.

Returns Batch

get_available_data_asset_names (*self*, *batch_kwargs_generator_names=None*)

Returns a dictionary of data_asset_names that the specified batch kwarg generator can provide. Note that some batch kwargs generators may not be capable of describing specific named data assets, and some (such as filesystem glob batch kwargs generators) require the user to configure data asset names.

Parameters **batch_kwargs_generator_names** – the BatchKwargsGenerator for which to get available data asset names.

Returns

```
{
    generator_name: {
        names: [ (data_asset_1, data_asset_1_type), (data_asset_2, data_
↪asset_2_type) ... ]
    }
    ...
}
```

Return type dictionary consisting of sets of generator assets available for the specified generators

build_batch_kwargs (*self*, *batch_kwargs_generator*, *data_asset_name=None*, *partition_id=None*, ***kwargs*)

```
class great_expectations.datasource.PandasDatasource (name='pandas',
                                                         data_context=None,
                                                         data_asset_type=None,
                                                         batch_kwargs_generators=None,
                                                         boto3_options=None,
                                                         reader_method=None,
                                                         reader_options=None,
                                                         limit=None, **kwargs)
```

Bases: `great_expectations.datasource.datasource.Datasource`

The PandasDatasource produces PandasDataset objects and supports generators capable of interacting with the local filesystem (the default subdir_reader generator), and from existing in-memory dataframes.

recognized_batch_parameters

classmethod build_configuration (*cls*, *data_asset_type=None*,
batch_kwargs_generators=None, *boto3_options=None*,
reader_method=None, *reader_options=None*,
limit=None, ***kwargs*)

Build a full configuration object for a datasource, potentially including generators with defaults.

Parameters

- **data_asset_type** – A ClassConfig dictionary
- **batch_kwargs_generators** – Generator configuration dictionary
- **boto3_options** – Optional dictionary with key-value pairs to pass to boto3 during instantiation.
- **reader_method** – Optional default reader_method for generated batches
- **reader_options** – Optional default reader_options for generated batches
- **limit** – Optional default limit for generated batches
- ****kwargs** – Additional kwargs to be part of the datasource constructor’s initialization

Returns A complete datasource configuration.

process_batch_parameters (*self*, *reader_method=None*, *reader_options=None*, *limit=None*,
dataset_options=None)

Use datasource-specific configuration to translate any batch parameters into batch kwargs at the datasource level.

Parameters

- **limit** (*int*) – a parameter all datasources must accept to allow limiting a batch to a smaller number of rows.
- **dataset_options** (*dict*) – a set of kwargs that will be passed to the constructor of a dataset built using these batch_kwargs

Returns Result will include both parameters passed via argument and configured parameters.

Return type batch_kwargs

get_batch (*self*, *batch_kwargs*, *batch_parameters=None*)

Get a batch of data from the datasource.

Parameters

- **batch_kwargs** – the BatchKwargs to use to construct the batch
- **batch_parameters** – optional parameters to store as the reference description of the batch. They should reflect parameters that would provide the passed BatchKwargs.

Returns Batch

static guess_reader_method_from_path (*path*)

_get_reader_fn (*self*, *reader_method=None*, *path=None*)

Static helper for parsing reader types. If reader_method is not provided, path will be used to guess the correct reader_method.

Parameters

- **reader_method** (*str*) – the name of the reader method to use, if available.
- **path** (*str*) – the to use to guess

Returns ReaderMethod to use for the filepath

```
class great_expectations.datasource.SparkDFDatasource (name='default',
                                                         data_context=None,
                                                         data_asset_type=None,
                                                         batch_kwargs_generators=None,
                                                         spark_config=None,
                                                         **kwargs)
```

Bases: `great_expectations.datasource.datasource.Datasource`

The SparkDFDatasource produces SparkDFDatasets and supports generators capable of interacting with local filesystem (the default subdir_reader batch kwargs generator) and databricks notebooks.

Accepted Batch Kwargs:

- PathBatchKwargs (“path” or “s3” keys)
- InMemoryBatchKwargs (“dataset” key)
- QueryBatchKwargs (“query” key)

recognized_batch_parameters

```
classmethod build_configuration (cls,
                                 data_asset_type=None,
                                 batch_kwargs_generators=None, spark_config=None,
                                 **kwargs)
```

Build a full configuration object for a datasource, potentially including generators with defaults.

Parameters

- **data_asset_type** – A ClassConfig dictionary
- **batch_kwargs_generators** – Generator configuration dictionary
- **spark_config** – dictionary of key-value pairs to pass to the spark builder
- ****kwargs** – Additional kwargs to be part of the datasource constructor’s initialization

Returns A complete datasource configuration.

```
process_batch_parameters (self, reader_method=None, reader_options=None, limit=None,
                           dataset_options=None)
```

Use datasource-specific configuration to translate any batch parameters into batch kwargs at the datasource level.

Parameters

- **limit** (*int*) – a parameter all datasources must accept to allow limiting a batch to a smaller number of rows.
- **dataset_options** (*dict*) – a set of kwargs that will be passed to the constructor of a dataset built using these batch_kwargs

Returns Result will include both parameters passed via argument and configured parameters.

Return type batch_kwargs

```
get_batch (self, batch_kwargs, batch_parameters=None)
class-private implementation of get_data_asset
```

```
static guess_reader_method_from_path (path)
```


`_get_reader_fn` (*self*, *reader*, *reader_method=None*, *path=None*)

Static helper for providing reader_fn

Parameters

- **reader** – the base spark reader to use; this should have had reader_options applied already
- **reader_method** – the name of the reader_method to use, if specified
- **path** (*str*) – the path to use to guess reader_method if it was not specified

Returns ReaderMethod to use for the filepath

```
class great_expectations.datasource.SqlAlchemyDatasource (name='default',
                                                            data_context=None,
                                                            data_asset_type=None,
                                                            credentials=None,
                                                            batch_kwargs_generators=None,
                                                            **kwargs)
```

Bases: `great_expectations.datasource.Datasource`

A SqlAlchemyDatasource will provide data_assets converting batch_kwargs using the following rules: - if the batch_kwargs include a table key, the datasource will provide a dataset object connected to that table

- if the batch_kwargs include a query key, the datasource will create a temporary table using that that query. The query can be parameterized according to the standard python Template engine, which uses \$parameter, with additional kwargs passed to the get_batch method.

recognized_batch_parameters

```
classmethod build_configuration (cls,
                                 data_asset_type=None,
                                 batch_kwargs_generators=None, **kwargs)
```

Build a full configuration object for a datasource, potentially including generators with defaults.

Parameters

- **data_asset_type** – A ClassConfig dictionary
- **batch_kwargs_generators** – Generator configuration dictionary
- ****kwargs** – Additional kwargs to be part of the datasource constructor's initialization

Returns A complete datasource configuration.

`_get_sqlalchemy_connection_options` (*self*, ***kwargs*)

`get_batch` (*self*, *batch_kwargs*, *batch_parameters=None*)

Get a batch of data from the datasource.

Parameters

- **batch_kwargs** – the BatchKwargs to use to construct the batch
- **batch_parameters** – optional parameters to store as the reference description of the batch. They should reflect parameters that would provide the passed BatchKwargs.

Returns Batch

`process_batch_parameters` (*self*, *query_parameters=None*, *limit=None*, *dataset_options=None*)

Use datasource-specific configuration to translate any batch parameters into batch kwargs at the datasource level.

Parameters

- **limit** (*int*) – a parameter all datasources must accept to allow limiting a batch to a smaller number of rows.
- **dataset_options** (*dict*) – a set of kwargs that will be passed to the constructor of a dataset built using these batch_kwargs

Returns Result will include both parameters passed via argument and configured parameters.

Return type batch_kwargs

great_expectations.jupyter UX

Utility functions for working with great_expectations within jupyter notebooks or jupyter lab.

Submodules**great_expectations.jupyter UX.expectation_explorer****Module Contents****Classes**

ExpectationExplorer()

great_expectations.jupyter UX.expectation_explorer.**logger**

class great_expectations.jupyter UX.expectation_explorer.**ExpectationExplorer**

Bases: object

update_result (*self*, *data_asset_name*, *new_result*, *column=None*)

get_expectation_state (*self*, *data_asset_name*, *expectation_type*, *column=None*)

initialize_data_asset_state (*self*, *data_asset*)

set_expectation_state (*self*, *data_asset*, *expectation_state*, *column=None*)

expectation_kwarg_dict_to_ge_kwargs (*self*, *kwarg_dict*)

update_kwarg_widget_dict (*self*, *expectation_state*, *current_widget_dict*, *ge_kwarg_name*, *new_ge_kwarg_value*)

update_expectation_state (*self*, *existing_expectation_state*, *expectation_validation_result*, *validation_time*)

generate_boolean_checkbox_widget (*self*, *value*, *description=""*, *description_tooltip=""*, *disabled=False*)

generate_text_area_widget (*self*, *value*, *description=""*, *description_tooltip=""*, *continuous_update=False*, *placeholder=""*)

generate_text_widget (*self*, *value*, *description=""*, *description_tooltip=""*, *continuous_update=False*, *placeholder=""*, *disabled=False*)

generate_radio_buttons_widget (*self*, *value*, *options=None*, *description=""*, *description_tooltip=""*)

```

generate_remove_expectation_button (self, expectation_state)
generate_tag_button (self, expectation_state, tag, tag_list, widget_display)
generate_tag_button_list (self, expectation_state, tag_list, widget_display)
generate_zero_or_positive_integer_widget (self, value, max=int(9e+300), descrip-
                                         tion="", continuous_update=False)
generate_output_strftime_format_widget_dict (self, expectation_state, out-
                                         put_strftime_format="", column=None,
                                         **expectation_kwargs)
generate_strftime_format_widget_dict (self, expectation_state, strftime_format="", col-
                                         umn=None, **expectation_kwargs)
generate_value_widget_dict (self, expectation_state, value=None, column=None, **expecta-
                                         tion_kwargs)
generate_json_schema_widget_dict (self, expectation_state, json_schema="", column=None,
                                         **expectation_kwargs)
generate_ties_okay_widget_dict (self, expectation_state, ties_okay=None, column=None,
                                         **expectation_kwargs)
generate_match_on_widget_dict (self, expectation_state, match_on='any', column=None,
                                         **expectation_kwargs)
generate_regex_list_widget_dict (self, expectation_state, regex_list=None, column=None,
                                         **expectation_kwargs)
generate_column_list_widget_dict (self, expectation_state, column_list=None, col-
                                         umn=None, **expectation_kwargs)
generate_column_index_widget_dict (self, expectation_state, column, column_index="",
                                         **expectation_kwargs)
generate_regex_widget_dict (self, expectation_state, regex="", column=None, **expecta-
                                         tion_kwargs)
generate_parse_strings_as_datetimes_widget_dict (self, expectation_state,
                                         parse_strings_as_datetimes=None,
                                         column=None, **expecta-
                                         tion_kwargs)
generate_strictly_widget_dict (self, expectation_state, strictly=None, column=None, **ex-
                                         pectation_kwargs)
generate_mostly_widget_dict (self, expectation_state, mostly=1, column=None, **expecta-
                                         tion_kwargs)
generate_min_max_type_widget_dict (self, expectation_state, type_option=None, col-
                                         umn=None, **expectation_kwargs)
generate_min_value_widget_dict (self, expectation_state, min_value=None, column=None,
                                         **expectation_kwargs)
generate_max_value_widget_dict (self, expectation_state, max_value=None, column=None,
                                         **expectation_kwargs)
generate_value_set_widget_dict (self, expectation_state, value_set=None, column=None,
                                         **expectation_kwargs)
generate_expectation_kwarg_fallback_widget_dict (self, expectation_kwarg_name,
                                         **expectation_kwargs)
generate_column_widget (self, column=None)
generate_expectation_type_widget (self, expectation_type)

```

```

generate_basic_expectation_info_box(self, data_asset_name, expectation_type, suc-
                                     cess_widget, validation_time, column=None)
generate_expectation_result_detail_widgets(self, result=None)
create_expectation_widget(self, data_asset, expectation_validation_result, collapsed=False)
get_column_names(self, data_asset_name)
get_expectation_types(self, data_asset_name)
generate_expectation_suite_editor_widgets(self, data_asset, expectation_suite)
edit_expectation_suite(self, data_asset)

```

Package Contents

Classes

<code>ExpectationSuiteColumnSectionRenderer</code>	(bullet_list_renderer=None)
<code>ProfilingResultsColumnSectionRenderer</code>	(overview_table_renderer=None, expectation_string_renderer=None, run-time_environment=None)
<code>ValidationResultsColumnSectionRenderer</code>	(table_renderer=None)
<code>DefaultJinjaSectionView</code>	(custom_styles_directory=None, custom_views_directory=None)

Functions

<code>set_data_source</code>	(context, data_source_type=None)	TODO: Needs a docstring and tests.
<code>setup_notebook_logging</code>	(logger=None, log_level=logging.INFO)	Set up the provided logger for the GE default logging configuration.
<code>show_available_data_asset_names</code>	(context, data_source_name=None)	List asset names found in the current context.
<code>_render_for_jupyter</code>	(view, include_styling, return_without_displaying)	
<code>display_column_expectations_as_section</code>	(expectations, column, include_styling=True, return_without_displaying=False)	This is a utility function to render all of the Expectations in an ExpectationSuite with the same column name as an HTML block.
<code>display_profiled_column_evs_as_section</code>	(evs, column, include_styling=True, return_without_displaying=False)	This is a utility function to render all of the EVRs in an ExpectationSuite with the same column name as an HTML block.
<code>display_column_evs_as_section</code>	(evs, column, include_styling=True, return_without_displaying=False)	Display validation results for a single column as a section.

```

class great_expectations.jupyter_ux.ExpectationSuiteColumnSectionRenderer (bullet_list_renderer=None)
    Bases: great_expectations.render.renderer.column_section_renderer.ColumnSectionRenderer
    classmethod _render_header(cls, expectations)
    _render_bullet_list(self, expectations)

```

render (*self*, *expectations*)

class great_expectations.jupyter_ux.**ProfilingResultsColumnSectionRenderer** (*overview_table_renderer=None*, *expectation_string_renderer=None*, *run_time_environment=None*)

Bases: [great_expectations.render.renderer.column_section_renderer.ColumnSectionRenderer](#)

render (*self*, *evrs*, *section_name=None*, *column_type=None*)

classmethod **_render_header** (*cls*, *evrs*, *column_type=None*)

classmethod **_render_expectation_types** (*cls*, *evrs*, *content_blocks*)

_render_overview_table (*self*, *evrs*)

classmethod **_render_quantile_table** (*cls*, *evrs*)

classmethod **_render_stats_table** (*cls*, *evrs*)

classmethod **_render_values_set** (*cls*, *evrs*)

_render_histogram (*self*, *evrs*)

classmethod **_render_bar_chart_table** (*cls*, *evrs*)

classmethod **_render_failed** (*cls*, *evrs*)

classmethod **_render_unrecognized** (*cls*, *evrs*, *content_blocks*)

class great_expectations.jupyter_ux.**ValidationResultsColumnSectionRenderer** (*table_renderer=None*)

Bases: [great_expectations.render.renderer.column_section_renderer.ColumnSectionRenderer](#)

classmethod **_render_header** (*cls*, *validation_results*)

_render_table (*self*, *validation_results*)

render (*self*, *validation_results*)

class great_expectations.jupyter_ux.**DefaultJinjaSectionView** (*custom_styles_directory=None*, *custom_views_directory=None*)

Bases: [great_expectations.render.view.view.DefaultJinjaView](#)

Defines a method for converting a document to human-consumable form

- Font Awesome 5.10.1
- Bootstrap 4.3.1
- jQuery 3.2.1
- Vega 5
- Vega-Lite 4
- Vega-Embed 6

_template = **section.j2**

_validate_document (*self*, *document*)

```
great_expectations.jupyter_ux.set_data_source(context, data_source_type=None)
    TODO: Needs a docstring and tests.
```

```
great_expectations.jupyter_ux.setup_notebook_logging(logger=None,
    log_level=logging.INFO)
```

Set up the provided logger for the GE default logging configuration.

Parameters – the logger to configure (*logger*) –

```
great_expectations.jupyter_ux.show_available_data_asset_names(context,
    data_source_name=None)
```

List asset names found in the current context.

```
great_expectations.jupyter_ux.bootstrap_link_element = <link rel="stylesheet" href="https://
```

```
great_expectations.jupyter_ux.cooltip_style_element = <style type="text/css">
```

```
.cooltip { display:inline-block; position:relative; text-align:left;
```

```
}
```

```
.cooltip.top { min-width:200px; top:-6px; left:50%; transform:translate(-50%, -100%); padding:10px 20px;
    color:#FFFFFF; background-color:#222222; font-weight:normal; font-size:13px; border-radius:8px; posi-
    tion:absolute; z-index:99999999; box-sizing:border-box; box-shadow:0 1px 8px rgba(0,0,0,0.5); display:none;
```

```
}
```

```
.cooltip:hover .top { display:block;
```

```
}
```

```
.cooltip.top i { position:absolute; top:100%; left:50%; margin-left:-12px; width:24px; height:12px; over-
    flow:hidden;
```

```
}
```

```
.cooltip.top i::after { content:''; position:absolute; width:12px; height:12px; left:50%; transform:translate(-50%,-
    50%) rotate(45deg); background-color:#222222; box-shadow:0 1px 8px rgba(0,0,0,0.5);
```

```
} </style>
```

```
great_expectations.jupyter_ux._render_for_jupyter(view, include_styling, re-
    turn_without_displaying)
```

```
great_expectations.jupyter_ux.display_column_expectations_as_section(expectation_suite,
    col-
    umn,
    in-
    clude_styling=True,
    re-
    turn_without_displaying=False)
```

This is a utility function to render all of the Expectations in an ExpectationSuite with the same column name as an HTML block.

By default, the HTML block is rendered using ExpectationSuiteColumnSectionRenderer and the view is rendered using DefaultJinjaSectionView. Therefore, it should look exactly the same as the default renderer for build_docs.

Example usage: `exp = context.get_expectation_suite("notable_works_by_charles_dickens", "BasicDatasetProfiler")` `display_column_expectations_as_section(exp, "Type")`

```
great_expectations.jupyter_ux.display_profiled_column_evrs_as_section(evrs,
                                                                    col-
                                                                    umn,
                                                                    in-
                                                                    clude_styling=True,
                                                                    re-
                                                                    turn_without_displaying=False)
```

This is a utility function to render all of the EVRs in an ExpectationSuite with the same column name as an HTML block.

By default, the HTML block is rendered using ExpectationSuiteColumnSectionRenderer and the view is rendered using DefaultJinjaSectionView. Therefore, it should look exactly the same as the default renderer for build_docs.

Example usage: `display_column_evrs_as_section(exp, "my_column")`

WARNING: This method is experimental.

```
great_expectations.jupyter_ux.display_column_evrs_as_section(evrs, column, in-
                                                                clude_styling=True,
                                                                re-
                                                                turn_without_displaying=False)
```

Display validation results for a single column as a section.

WARNING: This method is experimental.

```
great_expectations.jupyter_ux.logger
```

great_expectations.profile

Submodules

great_expectations.profile.base

Module Contents

Classes

<code>ProfilerDataType()</code>	Generic enumeration.
<code>ProfilerCardinality()</code>	Generic enumeration.
<code>DataAssetProfiler()</code>	
<code>DatasetProfiler()</code>	

```
great_expectations.profile.base.logger
```

class great_expectations.profile.base.ProfilerDataType

Bases: `enum.Enum`

Generic enumeration.

Derive from this class to define new enumerations.

INT = `int`

FLOAT = `float`

STRING = `string`

BOOLEAN = `boolean`

DATETIME = `datetime`

UNKNOWN = `unknown`

class `great_expectations.profile.base.ProfilerCardinality`

Bases: `enum.Enum`

Generic enumeration.

Derive from this class to define new enumerations.

NONE = `none`

ONE = `one`

TWO = `two`

FEW = `few`

VERY_FEW = `very few`

MANY = `many`

VERY_MANY = `very many`

UNIQUE = `unique`

class `great_expectations.profile.base.DataAssetProfiler`

Bases: `object`

classmethod `validate(cls, data_asset)`

class `great_expectations.profile.base.DatasetProfiler`

Bases: `great_expectations.profile.base.DataAssetProfiler`

classmethod `validate(cls, dataset)`

classmethod `add_expectation_meta(cls, expectation)`

classmethod `add_meta(cls, expectation_suite, batch_kwargs=None)`

classmethod `profile(cls, data_asset, run_id=None, profiler_configuration=None, run_name=None, run_time=None)`

abstract classmethod `_profile(cls, dataset, configuration=None)`

`great_expectations.profile.basic_dataset_profiler`

Module Contents

Classes

<code>BasicDatasetProfilerBase()</code>	BasicDatasetProfilerBase provides basic logic of inferring the type and the cardinality of columns
<code>BasicDatasetProfiler()</code>	BasicDatasetProfiler is inspired by the beloved pandas profiling project.

`great_expectations.profile.basic_dataset_profiler.OperationalError`

`great_expectations.profile.basic_dataset_profiler.logger`


```
class great_expectations.profile.basic_dataset_profiler.BasicDatasetProfilerBase
    Bases: great_expectations.profile.base.DatasetProfiler
```

BasicDatasetProfilerBase provides basic logic of inferring the type and the cardinality of columns that is used by the dataset profiler classes that extend this class.

```
INT_TYPE_NAMES
```

```
FLOAT_TYPE_NAMES
```

```
STRING_TYPE_NAMES
```

```
BOOLEAN_TYPE_NAMES
```

```
DATETIME_TYPE_NAMES
```

```
classmethod _get_column_type (cls, df, column)
```

```
classmethod _get_column_cardinality (cls, df, column)
```

```
class great_expectations.profile.basic_dataset_profiler.BasicDatasetProfiler
    Bases: great_expectations.profile.basic_dataset_profiler.  
BasicDatasetProfilerBase
```

BasicDatasetProfiler is inspired by the beloved pandas_profiling project.

The profiler examines a batch of data and creates a report that answers the basic questions most data practitioners would ask about a dataset during exploratory data analysis. The profiler reports how unique the values in the column are, as well as the percentage of empty values in it. Based on the column's type it provides a description of the column by computing a number of statistics, such as min, max, mean and median, for numeric columns, and distribution of values, when appropriate.

```
classmethod _profile (cls, dataset, configuration=None)
```

```
great_expectations.profile.basic_suite_builder_profiler
```

Module Contents

Classes

<i>BasicSuiteBuilderProfiler()</i>	This profiler helps build coarse expectations for columns you care about.
------------------------------------	---

Functions

<i>_check_that_expectations_are_available</i> (dataset, expectations)
<i>_check_that_columns_exist</i> (dataset, columns)
<i>_is_nan</i> (value)
<i>_remove_table_expectations</i> (dataset, all_types_to_remove)
<i>_remove_column_expectations</i> (dataset, all_types_to_remove)

```
class great_expectations.profile.basic_suite_builder_profiler.BasicSuiteBuilderProfiler
```

Bases: `great_expectations.profile.basic_dataset_profiler.
BasicDatasetProfilerBase`

This profiler helps build coarse expectations for columns you care about.

The goal of this profiler is to expedite the process of authoring an expectation suite by building possibly relevant expectations for columns that you care about. You can then easily edit the suite and adjust or delete these expectations to hone your new suite.

Ranges of acceptable values in the expectations created by this profiler (for example, the min/max of the value in `expect_column_values_to_be_between`) are created only to demonstrate the functionality and should not be taken as the actual ranges. You should definitely edit this coarse suite.

Configuration is optional, and if not provided, this profiler will create expectations for all columns.

Configuration is a dictionary with a *columns* key containing a list of the column names you want coarse expectations created for. This dictionary can also contain a *excluded_expectations* key with a list of expectation names you do not want created or a *included_expectations* key with a list of expectation names you want created (if applicable).

For example, if you had a wide patients table and you want expectations on three columns, you'd do this:

```
suite, validation_result = BasicSuiteBuilderProfiler().profile( dataset, {"columns": ["id", "username", "ad-  
dress"]}  
)
```

For example, if you had a wide patients table and you want expectations on all columns, excluding three statistical expectations, you'd do this:

```
suite, validation_result = BasicSuiteBuilderProfiler().profile( dataset, {  
    "excluded_expectations": [  
        "expect_column_mean_to_be_between", "expect_column_median_to_be_between",  
        "expect_column_quantile_values_to_be_between",  
    ],  
}  
)
```

For example, if you had a wide patients table and you want only two types of expectations on all applicable columns you'd do this:

```
suite, validation_result = BasicSuiteBuilderProfiler().profile( dataset, {  
    "included_expectations": [  
        "expect_column_to_not_be_null", "expect_column_values_to_be_in_set",  
    ],  
}  
)
```

It can also be used to generate an expectation suite that contains one instance of every interesting expectation type.

When used in this “demo” mode, the suite is intended to demonstrate of the expressive power of expectations and provide a service similar to the one expectations glossary documentation page, but on a users’ own data.

```
suite, validation_result = BasicSuiteBuilderProfiler().profile(dataset, configuration="demo")
```

```
classmethod _get_column_type_with_caching(cls, dataset, column_name, cache)
```

```

classmethod _get_column_cardinality_with_caching(cls, dataset, column_name,
                                                cache)
classmethod _create_expectations_for_low_card_column(cls, dataset, column, col-
                                                    umn_cache)
classmethod _create_non_nullity_expectations(cls, dataset, column)
classmethod _create_expectations_for_numeric_column(cls, dataset, column)
classmethod _create_expectations_for_string_column(cls, dataset, column)
classmethod _find_next_low_card_column(cls, dataset, columns, profiled_columns, col-
                                      umn_cache)
classmethod _find_next_numeric_column(cls, dataset, columns, profiled_columns, col-
                                      umn_cache)
classmethod _find_next_string_column(cls, dataset, columns, profiled_columns, col-
                                      umn_cache)
classmethod _find_next_datetime_column(cls, dataset, columns, profiled_columns, col-
                                      umn_cache)
classmethod _create_expectations_for_datetime_column(cls, dataset, column)
classmethod _profile(cls, dataset, configuration=None)
classmethod _demo_profile(cls, dataset)
classmethod _build_table_row_count_expectation(cls, dataset, tolerance=0.1)
classmethod _build_table_column_expectations(cls, dataset)
classmethod _build_column_description_metadata(cls, dataset)
great_expectations.profile.basic_suite_builder_profiler._check_that_expectations_are_available(
    dataset, columns)

great_expectations.profile.basic_suite_builder_profiler._check_that_columns_exist(dataset,
                                          columns)
great_expectations.profile.basic_suite_builder_profiler._is_nan(value)
great_expectations.profile.basic_suite_builder_profiler._remove_table_expectations(dataset,
                                          all_types_table)
great_expectations.profile.basic_suite_builder_profiler._remove_column_expectations(dataset,
                                          all_types_table)

great_expectations.profile.columns_exist

```

Module Contents

Classes

ColumnsExistProfiler()

```

class great_expectations.profile.columns_exist.ColumnsExistProfiler
    Bases: great_expectations.profile.base.DatasetProfiler

```

classmethod `_profile` (*cls*, *dataset*, *configuration=None*)

This function will take a dataset and add expectations that each column present exists.

Parameters

- **dataset** (*great_expectations.dataset*) – The dataset to profile and to which to add expectations.
- **configuration** – Configuration for select profilers.

`great_expectations.profile.metrics_utils`

Module Contents

Functions

<code>tuple_to_hash(tuple_)</code>	
<code>kwargs_to_tuple(d)</code>	Convert expectation configuration kwargs to a canonical tuple.

`great_expectations.profile.metrics_utils.tuple_to_hash(tuple_)`

`great_expectations.profile.metrics_utils.kwargs_to_tuple(d)`

Convert expectation configuration kwargs to a canonical tuple.

`great_expectations.profile.multi_batch_validation_meta_analysis`

Package Contents

Classes

<code>BasicDatasetProfiler()</code>	BasicDatasetProfiler is inspired by the beloved pandas_profiling project.
<code>BasicSuiteBuilderProfiler()</code>	This profiler helps build coarse expectations for columns you care about.
<code>ColumnsExistProfiler()</code>	

class `great_expectations.profile.BasicDatasetProfiler`

Bases: `great_expectations.profile.basic_dataset_profiler.BasicDatasetProfilerBase`

BasicDatasetProfiler is inspired by the beloved pandas_profiling project.

The profiler examines a batch of data and creates a report that answers the basic questions most data practitioners would ask about a dataset during exploratory data analysis. The profiler reports how unique the values in the column are, as well as the percentage of empty values in it. Based on the column's type it provides a description of the column by computing a number of statistics, such as min, max, mean and median, for numeric columns, and distribution of values, when appropriate.

classmethod `_profile` (*cls*, *dataset*, *configuration=None*)

class `great_expectations.profile.BasicSuiteBuilderProfiler`

Bases: `great_expectations.profile.basic_dataset_profiler.BasicDatasetProfilerBase`

This profiler helps build coarse expectations for columns you care about.

The goal of this profiler is to expedite the process of authoring an expectation suite by building possibly relevant expectations for columns that you care about. You can then easily edit the suite and adjust or delete these expectations to hone your new suite.

Ranges of acceptable values in the expectations created by this profiler (for example, the min/max of the value in `expect_column_values_to_be_between`) are created only to demonstrate the functionality and should not be taken as the actual ranges. You should definitely edit this coarse suite.

Configuration is optional, and if not provided, this profiler will create expectations for all columns.

Configuration is a dictionary with a `columns` key containing a list of the column names you want coarse expectations created for. This dictionary can also contain a `excluded_expectations` key with a list of expectation names you do not want created or a `included_expectations` key with a list of expectation names you want created (if applicable).

For example, if you had a wide patients table and you want expectations on three columns, you'd do this:

```
suite, validation_result = BasicSuiteBuilderProfiler().profile( dataset, {"columns": ["id", "username", "address"]})
```

For example, if you had a wide patients table and you want expectations on all columns, excluding three statistical expectations, you'd do this:

```
suite, validation_result = BasicSuiteBuilderProfiler().profile( dataset, {
    "excluded_expectations": [
        "expect_column_mean_to_be_between", "expect_column_median_to_be_between",
        "expect_column_quantile_values_to_be_between",
    ],
})
```

For example, if you had a wide patients table and you want only two types of expectations on all applicable columns you'd do this:

```
suite, validation_result = BasicSuiteBuilderProfiler().profile( dataset, {
    "included_expectations": [
        "expect_column_to_not_be_null", "expect_column_values_to_be_in_set",
    ],
})
```

It can also be used to generate an expectation suite that contains one instance of every interesting expectation type.

When used in this “demo” mode, the suite is intended to demonstrate of the expressive power of expectations and provide a service similar to the one expectations glossary documentation page, but on a users’ own data.

```
suite, validation_result = BasicSuiteBuilderProfiler().profile(dataset, configuration="demo")
```

```
classmethod _get_column_type_with_caching(cls, dataset, column_name, cache)
```

```
classmethod _get_column_cardinality_with_caching(cls, dataset, column_name,
                                                  cache)
classmethod _create_expectations_for_low_card_column(cls, dataset, column, col-
                                                    umn_cache)
classmethod _create_non_nullity_expectations(cls, dataset, column)
classmethod _create_expectations_for_numeric_column(cls, dataset, column)
classmethod _create_expectations_for_string_column(cls, dataset, column)
classmethod _find_next_low_card_column(cls, dataset, columns, profiled_columns, col-
                                       umn_cache)
classmethod _find_next_numeric_column(cls, dataset, columns, profiled_columns, col-
                                       umn_cache)
classmethod _find_next_string_column(cls, dataset, columns, profiled_columns, col-
                                     umn_cache)
classmethod _find_next_datetime_column(cls, dataset, columns, profiled_columns, col-
                                       umn_cache)
classmethod _create_expectations_for_datetime_column(cls, dataset, column)
classmethod _profile(cls, dataset, configuration=None)
classmethod _demo_profile(cls, dataset)
classmethod _build_table_row_count_expectation(cls, dataset, tolerance=0.1)
classmethod _build_table_column_expectations(cls, dataset)
classmethod _build_column_description_metadata(cls, dataset)
class great_expectations.profile.ColumnsExistProfiler
  Bases: great_expectations.profile.base.DatasetProfiler
  classmethod _profile(cls, dataset, configuration=None)
    This function will take a dataset and add expectations that each column present exists.
```

Parameters

- **dataset** (*great_expectations.dataset*) – The dataset to profile and to which to add expectations.
- **configuration** – Configuration for select profilers.

`great_expectations.render`

Subpackages

`great_expectations.render.renderers`

Subpackages

`great_expectations.render.renderers.content_block`

Submodules

`great_expectations.render.renderer.content_block.bullet_list_content_block`

Module Contents

Classes

ExpectationSuiteBulletListContentBlockRenderer()

```
class great_expectations.render.renderer.content_block.bullet_list_content_block.ExpectationSuiteBulletListContentBlockRenderer
    Bases: great_expectations.render.renderer.content_block.expectation_string.ExpectationStringRenderer
    _rendered_component_type
    _content_block_type = bullet_list
    _default_element_styling
```

`great_expectations.render.renderer.content_block.content_block`

Module Contents

Classes

ContentBlockRenderer()

`great_expectations.render.renderer.content_block.content_block.logger`

```
class great_expectations.render.renderer.content_block.content_block.ContentBlockRenderer
    Bases: great_expectations.render.renderer.renderer.Renderer
    _rendered_component_type
    _default_header =
    _default_content_block_styling
    _default_element_styling
    classmethod validate_input (cls, render_object)
    classmethod render (cls, render_object, **kwargs)
    classmethod _render_expectation_meta_notes (cls, expectation)
    classmethod _process_content_block (cls, content_block, has_failed_evr)
    classmethod _get_content_block_fn (cls, expectation_type)
    classmethod list_available_expectations (cls)
    classmethod _missing_content_block_fn (cls, obj, styling, **kwargs)
    classmethod _get_content_block_styling (cls)
    classmethod _get_element_styling (cls)
    classmethod _get_header (cls)
```

`great_expectations.render.renderer.content_block.exception_list_content_block`

Module Contents

Classes

<i>ExceptionListContentBlockRenderer()</i>	Render a bullet list of exception messages raised for provided EVRs
--	---

class `great_expectations.render.renderer.content_block.exception_list_content_block.ExceptionListContentBlock`

Bases: `great_expectations.render.renderer.content_block.content_block.ContentBlockRenderer`

Render a bullet list of exception messages raised for provided EVRs

`_rendered_component_type`

`_content_block_type = bullet_list`

`_default_header = Failed expectations `

`_default_content_block_styling`

`_default_element_styling`

`classmethod _missing_content_block_fn(cls, evr, styling=None, include_column_name=True)`

`great_expectations.render.renderer.content_block.expectation_string`

Module Contents

Classes

<i>ExpectationStringRenderer()</i>	
------------------------------------	--

Functions

<i>substitute_none_for_missing(kwargs, kwarg_list)</i>	Utility function to plug Nones in when optional parameters are not specified in expectation kwargs.
--	---

`great_expectations.render.renderer.content_block.expectation_string.substitute_none_for_missing`

Utility function to plug Nones in when optional parameters are not specified in expectation kwargs.

Example

Input: `kwargs={"a":1, "b":2}, kwarg_list=["c", "d"]`

Output: `{"a":1, "b":2, "c": None, "d": None}`

This is helpful for standardizing the input objects for rendering functions. The alternative is lots of awkward *if "some_param" not in kwargs or kwargs["some_param"] == None*: clauses in renderers.

```
class great_expectations.render.renderer.content_block.expectation_string.ExpectationString
    Bases: great_expectations.render.renderer.content_block.content_block,
ContentBlockRenderer
```

```
classmethod _missing_content_block_fn(cls, expectation, styling=None, include_column_name=True)

classmethod expect_column_to_exist(cls, expectation, styling=None, include_column_name=True)

classmethod expect_column_unique_value_count_to_be_between(cls, expectation, styling=None, include_column_name=True)

classmethod expect_column_values_to_be_between(cls, expectation, styling=None, include_column_name=True)

classmethod expect_column_pair_values_A_to_be_greater_than_B(cls, expectation, styling=None, include_column_name=True)

classmethod expect_column_pair_values_to_be_equal(cls, expectation, styling=None, include_column_name=True)

classmethod expect_table_columns_to_match_ordered_list(cls, expectation, styling=None, include_column_name=True)

classmethod expect_multicolumn_values_to_be_unique(cls, expectation, styling=None, include_column_name=True)

classmethod expect_table_column_count_to_equal(cls, expectation, styling=None, include_column_name=True)

classmethod expect_table_column_count_to_be_between(cls, expectation, styling=None, include_column_name=True)

classmethod expect_table_row_count_to_be_between(cls, expectation, styling=None, include_column_name=True)

classmethod expect_table_row_count_to_equal(cls, expectation, styling=None, include_column_name=True)

classmethod expect_column_distinct_values_to_be_in_set(cls, expectation, styling=None, include_column_name=True)

classmethod expect_column_values_to_not_be_null(cls, expectation, styling=None, include_column_name=True)
```

```
classmethod expect_column_values_to_be_null(cls, expectation, styling=None, include_column_name=True)

classmethod expect_column_values_to_be_of_type(cls, expectation, styling=None, include_column_name=True)

classmethod expect_column_values_to_be_in_type_list(cls, expectation, styling=None, include_column_name=True)

classmethod expect_column_values_to_be_in_set(cls, expectation, styling=None, include_column_name=True)

classmethod expect_column_values_to_not_be_in_set(cls, expectation, styling=None, include_column_name=True)

classmethod expect_column_proportion_of_unique_values_to_be_between(cls, expectation, styling=None, include_column_name=True)

classmethod expect_column_values_to_be_increasing(cls, expectation, styling=None, include_column_name=True)

classmethod expect_column_values_to_be_decreasing(cls, expectation, styling=None, include_column_name=True)

classmethod expect_column_value_lengths_to_be_between(cls, expectation, styling=None, include_column_name=True)

classmethod expect_column_value_lengths_to_equal(cls, expectation, styling=None, include_column_name=True)

classmethod expect_column_values_to_match_regex(cls, expectation, styling=None, include_column_name=True)

classmethod expect_column_values_to_not_match_regex(cls, expectation, styling=None, include_column_name=True)

classmethod expect_column_values_to_match_regex_list(cls, expectation, styling=None, include_column_name=True)

classmethod expect_column_values_to_not_match_regex_list(cls, expectation, styling=None, include_column_name=True)

classmethod expect_column_values_to_match_strftime_format(cls, expectation, styling=None, include_column_name=True)

classmethod expect_column_values_to_be_dateutil_parseable(cls, expectation, styling=None, include_column_name=True)
```

```

classmethod expect_column_values_to_be_json_parseable(cls, expectation,
                                                       styling=None, include_column_name=True)

classmethod expect_column_values_to_match_json_schema(cls, expectation,
                                                       styling=None, include_column_name=True)

classmethod expect_column_distinct_values_to_contain_set(cls, expectation,
                                                         styling=None, include_column_name=True)

classmethod expect_column_distinct_values_to_equal_set(cls, expectation,
                                                        styling=None, include_column_name=True)

classmethod expect_column_mean_to_be_between(cls, expectation, styling=None, include_column_name=True)

classmethod expect_column_median_to_be_between(cls, expectation, styling=None, include_column_name=True)

classmethod expect_column_stdev_to_be_between(cls, expectation, styling=None, include_column_name=True)

classmethod expect_column_max_to_be_between(cls, expectation, styling=None, include_column_name=True)

classmethod expect_column_min_to_be_between(cls, expectation, styling=None, include_column_name=True)

classmethod expect_column_sum_to_be_between(cls, expectation, styling=None, include_column_name=True)

classmethod expect_column_most_common_value_to_be_in_set(cls, expectation,
                                                         styling=None, include_column_name=True)

classmethod _get_kl_divergence_partition_object_table(cls, partition_object,
                                                       header=None)

classmethod expect_column_quantile_values_to_be_between(cls, expectation,
                                                         styling=None, include_column_name=True)

classmethod _get_kl_divergence_chart(cls, partition_object, header=None)

classmethod expect_column_kl_divergence_to_be_less_than(cls, expectation,
                                                         styling=None, include_column_name=True)

classmethod expect_column_values_to_be_unique(cls, expectation, styling=None, include_column_name=True)

```

`great_expectations.render.renderer.content_block.profiling_overview_table_content_block`

Module Contents

Classes

`ProfilingOverviewTableContentBlockRenderer()`

```
class great_expectations.render.renderer.content_block.profiling_overview_table_content_block.  
    Bases: great_expectations.render.renderer.content_block.content_block.  
            ContentBlockRenderer  
  
    classmethod render (cls, ge_object, header_row=None)  
        Each expectation method should return a list of rows  
  
    classmethod expect_column_values_to_not_match_regex (cls, ge_object)  
  
    classmethod expect_column_unique_value_count_to_be_between (cls, ge_object)  
  
    classmethod expect_column_proportion_of_unique_values_to_be_between (cls,  
                                                                           ge_object)  
  
    classmethod expect_column_max_to_be_between (cls, ge_object)  
  
    classmethod expect_column_mean_to_be_between (cls, ge_object)  
  
    classmethod expect_column_values_to_not_be_null (cls, ge_object)  
  
    classmethod expect_column_values_to_be_null (cls, ge_object)
```

```
great_expectations.render.renderer.content_block.validation_results_table_content_block
```

Module Contents

Classes

```
ValidationResultsTableContentBlockRenderer()
```

```
great_expectations.render.renderer.content_block.validation_results_table_content_block.log
```

```
class great_expectations.render.renderer.content_block.validation_results_table_content_block.  
    Bases: great_expectations.render.renderer.content_block.expectation_string.  
            ExpectationStringRenderer  
  
    _content_block_type = table  
    _rendered_component_type  
    _rendered_component_default_init_kwargs  
    _default_element_styling  
    _default_content_block_styling  
  
    classmethod _get_status_icon (cls, evr)  
  
    classmethod _get_unexpected_table (cls, evr)  
  
    classmethod _get_unexpected_statement (cls, evr)  
  
    classmethod _get_kl_divergence_observed_value (cls, evr)  
  
    classmethod _get_quantile_values_observed_value (cls, evr)  
  
    classmethod _get_observed_value (cls, evr)  
  
    classmethod _process_content_block (cls, content_block, has_failed_evr)  
  
    classmethod _get_content_block_fn (cls, expectation_type)
```

Package Contents

Classes

<i>ValidationResultsTableContentBlockRenderer()</i>	
<i>ExpectationSuiteBulletListContentBlockRenderer()</i>	
<i>ExceptionListContentBlockRenderer()</i>	Render a bullet list of exception messages raised for provided EVRs
<i>ExpectationStringRenderer()</i>	
<i>ProfilingOverviewTableContentBlockRenderer()</i>	

class great_expectations.render.renderer.content_block.**ValidationResultsTableContentBlockRenderer**
 Bases: *great_expectations.render.renderer.content_block.expectation_string.ExpectationStringRenderer*

```

    _content_block_type = table
    _rendered_component_type
    _rendered_component_default_init_kwargs
    _default_element_styling
    _default_content_block_styling
    classmethod _get_status_icon(cls, evr)
    classmethod _get_unexpected_table(cls, evr)
    classmethod _get_unexpected_statement(cls, evr)
    classmethod _get_kl_divergence_observed_value(cls, evr)
    classmethod _get_quantile_values_observed_value(cls, evr)
    classmethod _get_observed_value(cls, evr)
    classmethod _process_content_block(cls, content_block, has_failed_evr)
    classmethod _get_content_block_fn(cls, expectation_type)

```

class great_expectations.render.renderer.content_block.**ExpectationSuiteBulletListContentBlockRenderer**
 Bases: *great_expectations.render.renderer.content_block.expectation_string.ExpectationStringRenderer*

```

    _rendered_component_type
    _content_block_type = bullet_list
    _default_element_styling

```

class great_expectations.render.renderer.content_block.**ExceptionListContentBlockRenderer**
 Bases: *great_expectations.render.renderer.content_block.content_block.ContentBlockRenderer*

Render a bullet list of exception messages raised for provided EVRs

```

    _rendered_component_type
    _content_block_type = bullet_list
    _default_header = Failed expectations <span class="mr-3 triangle"></span>

```

```
_default_content_block_styling
_default_element_styling
classmethod _missing_content_block_fn(cls, evr, styling=None, include_column_name=True)
class great_expectations.render.renderers.content_block.ExpectationStringRenderer
Bases: great_expectations.render.renderers.content_block.ContentBlockRenderer
classmethod _missing_content_block_fn(cls, expectation, styling=None, include_column_name=True)
classmethod expect_column_to_exist(cls, expectation, styling=None, include_column_name=True)
classmethod expect_column_unique_value_count_to_be_between(cls, expectation, styling=None, include_column_name=True)
classmethod expect_column_values_to_be_between(cls, expectation, styling=None, include_column_name=True)
classmethod expect_column_pair_values_A_to_be_greater_than_B(cls, expectation, styling=None, include_column_name=True)
classmethod expect_column_pair_values_to_be_equal(cls, expectation, styling=None, include_column_name=True)
classmethod expect_table_columns_to_match_ordered_list(cls, expectation, styling=None, include_column_name=True)
classmethod expect_multicolumn_values_to_be_unique(cls, expectation, styling=None, include_column_name=True)
classmethod expect_table_column_count_to_equal(cls, expectation, styling=None, include_column_name=True)
classmethod expect_table_column_count_to_be_between(cls, expectation, styling=None, include_column_name=True)
classmethod expect_table_row_count_to_be_between(cls, expectation, styling=None, include_column_name=True)
classmethod expect_table_row_count_to_equal(cls, expectation, styling=None, include_column_name=True)
classmethod expect_column_distinct_values_to_be_in_set(cls, expectation, styling=None, include_column_name=True)
classmethod expect_column_values_to_not_be_null(cls, expectation, styling=None, include_column_name=True)
classmethod expect_column_values_to_be_null(cls, expectation, styling=None, include_column_name=True)
classmethod expect_column_values_to_be_of_type(cls, expectation, styling=None, include_column_name=True)
```

```

classmethod expect_column_values_to_be_in_type_list (cls, expectation,
                                                    styling=None, include_column_name=True)

classmethod expect_column_values_to_be_in_set (cls, expectation, styling=None, include_column_name=True)

classmethod expect_column_values_to_not_be_in_set (cls, expectation,
                                                    styling=None, include_column_name=True)

classmethod expect_column_proportion_of_unique_values_to_be_between (cls,
                                                                        expectation,
                                                                        styling=None,
                                                                        include_column_name=True)

classmethod expect_column_values_to_be_increasing (cls, expectation,
                                                    styling=None, include_column_name=True)

classmethod expect_column_values_to_be_decreasing (cls, expectation,
                                                    styling=None, include_column_name=True)

classmethod expect_column_value_lengths_to_be_between (cls, expectation,
                                                        styling=None, include_column_name=True)

classmethod expect_column_value_lengths_to_equal (cls, expectation, styling=None,
                                                    include_column_name=True)

classmethod expect_column_values_to_match_regex (cls, expectation, styling=None,
                                                    include_column_name=True)

classmethod expect_column_values_to_not_match_regex (cls, expectation,
                                                       styling=None, include_column_name=True)

classmethod expect_column_values_to_match_regex_list (cls, expectation,
                                                       styling=None, include_column_name=True)

classmethod expect_column_values_to_not_match_regex_list (cls, expectation,
                                                           styling=None, include_column_name=True)

classmethod expect_column_values_to_match_strftime_format (cls, expectation,
                                                           styling=None, include_column_name=True)

classmethod expect_column_values_to_be_dateutil_parseable (cls, expectation,
                                                            styling=None, include_column_name=True)

classmethod expect_column_values_to_be_json_parseable (cls, expectation,
                                                         styling=None, include_column_name=True)

classmethod expect_column_values_to_match_json_schema (cls, expectation,
                                                         styling=None, include_column_name=True)

```

```
classmethod expect_column_distinct_values_to_contain_set (cls, expectation,
                                                         styling=None, include_column_name=True)

classmethod expect_column_distinct_values_to_equal_set (cls, expectation,
                                                         styling=None, include_column_name=True)

classmethod expect_column_mean_to_be_between (cls, expectation, styling=None, include_column_name=True)

classmethod expect_column_median_to_be_between (cls, expectation, styling=None, include_column_name=True)

classmethod expect_column_stdev_to_be_between (cls, expectation, styling=None, include_column_name=True)

classmethod expect_column_max_to_be_between (cls, expectation, styling=None, include_column_name=True)

classmethod expect_column_min_to_be_between (cls, expectation, styling=None, include_column_name=True)

classmethod expect_column_sum_to_be_between (cls, expectation, styling=None, include_column_name=True)

classmethod expect_column_most_common_value_to_be_in_set (cls, expectation,
                                                           styling=None, include_column_name=True)

classmethod _get_kl_divergence_partition_object_table (cls, partition_object,
                                                         header=None)

classmethod expect_column_quantile_values_to_be_between (cls, expectation,
                                                         styling=None, include_column_name=True)

classmethod _get_kl_divergence_chart (cls, partition_object, header=None)

classmethod expect_column_kl_divergence_to_be_less_than (cls, expectation,
                                                         styling=None, include_column_name=True)

classmethod expect_column_values_to_be_unique (cls, expectation, styling=None, include_column_name=True)

class great_expectations.render.renderers.content_block.ProfilingOverviewTableContentBlockRenderer
Bases: great_expectations.render.renderers.content_block.content_block.ContentBlockRenderer

classmethod render (cls, ge_object, header_row=None)
    Each expectation method should return a list of rows

classmethod expect_column_values_to_not_match_regex (cls, ge_object)

classmethod expect_column_unique_value_count_to_be_between (cls, ge_object)

classmethod expect_column_proportion_of_unique_values_to_be_between (cls,
                                                                        ge_object)

classmethod expect_column_max_to_be_between (cls, ge_object)

classmethod expect_column_mean_to_be_between (cls, ge_object)

classmethod expect_column_values_to_not_be_null (cls, ge_object)

classmethod expect_column_values_to_be_null (cls, ge_object)
```


Submodules

`great_expectations.render.renderer.call_to_action_renderer`

Module Contents

Classes

CallToActionRenderer()

class `great_expectations.render.renderer.call_to_action_renderer.CallToActionRenderer`

Bases: `object`

`_document_defaults`

classmethod `render` (*cls*, *cta_object*)

Parameters **`cta_object`** – dict {

 “header”: # optional, can be a string or string template “buttons”: # list of CallToActionButtons

}

Returns

dict {

 “header”: # optional, can be a string or string template “buttons”: # list of CallToActionButtons

}

`great_expectations.render.renderer.column_section_renderer`

Module Contents

Classes

ColumnSectionRenderer()

ProfilingResultsColumnSectionRenderer(*overview_table_renderer=None*,
expectation_string_renderer=None, *run-*
time_environment=None)

ValidationResultsColumnSectionRenderer(*table_renderer=None*)

ExpectationSuiteColumnSectionRenderer(*bullet_list_renderer=None*)

Functions

`convert_to_string_and_escape(var)`

`great_expectations.render.renderer.column_section_renderer.logger``great_expectations.render.renderer.column_section_renderer.convert_to_string_and_escape(var)``class great_expectations.render.renderer.column_section_renderer.ColumnSectionRenderer``Bases: great_expectations.render.renderer.renderer.Renderer``classmethod _get_column_name(cls, ge_object)``class great_expectations.render.renderer.column_section_renderer.ProfilingResultsColumnSectionRenderer``Bases: great_expectations.render.renderer.column_section_renderer.
ColumnSectionRenderer``render(self, evrs, section_name=None, column_type=None)``classmethod _render_header(cls, evrs, column_type=None)``classmethod _render_expectation_types(cls, evrs, content_blocks)``_render_overview_table(self, evrs)``classmethod _render_quantile_table(cls, evrs)``classmethod _render_stats_table(cls, evrs)``classmethod _render_values_set(cls, evrs)``_render_histogram(self, evrs)``classmethod _render_bar_chart_table(cls, evrs)``classmethod _render_failed(cls, evrs)``classmethod _render_unrecognized(cls, evrs, content_blocks)``class great_expectations.render.renderer.column_section_renderer.ValidationResultsColumnSectionRenderer``Bases: great_expectations.render.renderer.column_section_renderer.
ColumnSectionRenderer``classmethod _render_header(cls, validation_results)``_render_table(self, validation_results)``render(self, validation_results)``class great_expectations.render.renderer.column_section_renderer.ExpectationSuiteColumnSectionRenderer``Bases: great_expectations.render.renderer.column_section_renderer.
ColumnSectionRenderer``classmethod _render_header(cls, expectations)``_render_bullet_list(self, expectations)``render(self, expectations)`

`great_expectations.render.renderer.other_section_renderer`

Module Contents

Classes

[ProfilingResultsOverviewSectionRenderer\(\)](#)

class `great_expectations.render.renderer.other_section_renderer.ProfilingResultsOverviewSe`

Bases: *`great_expectations.render.renderer.renderer.Renderer`*

classmethod `render` (*cls*, *evrs*, *section_name=None*)

classmethod `_render_header` (*cls*, *evrs*, *content_blocks*)

classmethod `_render_dataset_info` (*cls*, *evrs*, *content_blocks*)

classmethod `_render_variable_types` (*cls*, *evrs*, *content_blocks*)

classmethod `_render_expectation_types` (*cls*, *evrs*, *content_blocks*)

classmethod `_render_warnings` (*cls*, *evrs*, *content_blocks*)

classmethod `_get_percentage_missing_cells_str` (*cls*, *evrs*)

classmethod `_get_column_types` (*cls*, *evrs*)

`great_expectations.render.renderer.page_renderer`

Module Contents

Classes

[ValidationResultsPageRenderer](#)(*column_section_renderer=None*)

[ExpectationSuitePageRenderer](#)(*column_section_renderer=None*)

[ProfilingResultsPageRenderer](#)(*overview_section_renderer=None*,
column_section_renderer=None)

`great_expectations.render.renderer.page_renderer.logger`

class `great_expectations.render.renderer.page_renderer.ValidationResultsPageRenderer` (*column_s*

Bases: *`great_expectations.render.renderer.renderer.Renderer`*

render (*self*, *validation_results*)

classmethod `_render_validation_header` (*cls*, *validation_results*)

classmethod `_render_validation_info` (*cls*, *validation_results*)

classmethod `_render_nested_table_from_dict` (*cls*, *input_dict*, *header=None*,
sub_table=False)

classmethod `_render_validation_statistics` (*cls*, *validation_results*)

class `great_expectations.render.renderer.page_renderer.ExpectationSuitePageRenderer` (*column_s*

Bases: *`great_expectations.render.renderer.renderer.Renderer`*

render (*self*, *expectations*)

_render_table_level_expectations (*self*, *columns*)

classmethod _render_expectation_suite_header (*cls*)

classmethod _render_expectation_suite_info (*cls*, *expectations*)

classmethod _render_expectation_suite_notes (*cls*, *expectations*)

class great_expectations.render.renderer.page_renderer.**ProfilingResultsPageRenderer** (*overview_col-umn_sect*)

Bases: [great_expectations.render.renderer.renderer.Renderer](#)

render (*self*, *validation_results*)

[great_expectations.render.renderer.renderer](#)

Module Contents

Classes

[Renderer\(\)](#)

class great_expectations.render.renderer.renderer.**Renderer**

Bases: `object`

classmethod render (*cls*, *ge_object*)

classmethod _get_expectation_type (*cls*, *ge_object*)

classmethod _find_evr_by_type (*cls*, *evrs*, *type_*)

classmethod _find_all_evrs_by_type (*cls*, *evrs*, *type_*, *column_=None*)

classmethod _get_column_list_from_evrs (*cls*, *evrs*)

Get list of column names.

If `expect_table_columns_to_match_ordered_list` EVR is present, use it as the list, including the order.

Otherwise, get the list of all columns mentioned in the expectations and order it alphabetically.

Parameters *evrs* –

Returns list of columns with best effort sorting

classmethod _group_evrs_by_column (*cls*, *validation_results*)

classmethod _group_and_order_expectations_by_column (*cls*, *expectations*)

Group expectations by column.

`great_expectations.render.renderer.site_builder`

Module Contents

Classes

<code>SiteBuilder(data_context, store_backend, site_name=None, site_index_builder=None, show_how_to_buttons=True, site_section_builders=None, runtime_environment=None, **kwargs)</code>	SiteBuilder builds data documentation for the project defined by a
<code>DefaultSiteSectionBuilder(name, data_context, target_store, source_store_name, custom_styles_directory=None, custom_views_directory=None, show_how_to_buttons=True, run_name_filter=None, validation_results_limit=None, renderer=None, view=None, data_context_id=None, **kwargs)</code>	
<code>DefaultSiteIndexBuilder(name, site_name, data_context, target_store, custom_styles_directory=None, custom_views_directory=None, show_how_to_buttons=True, validation_results_limit=None, renderer=None, view=None, data_context_id=None, source_stores=None, **kwargs)</code>	
<code>CallToActionButton(title, link)</code>	

`great_expectations.render.renderer.site_builder.logger`

```
class great_expectations.render.renderer.site_builder.SiteBuilder(data_context,
                                                                    store_backend,
                                                                    site_name=None,
                                                                    site_index_builder=None,
                                                                    show_how_to_buttons=True,
                                                                    site_section_builders=None,
                                                                    run-
                                                                    time_environment=None,
                                                                    **kwargs)
```

Bases: `object`

SiteBuilder builds data documentation for the project defined by a `DataContext`.

A data documentation site consists of HTML pages for expectation suites, profiling and validation results, and an `index.html` page that links to all the pages.

The exact behavior of SiteBuilder is controlled by configuration in the `DataContext`'s `great_expectations.yml` file.

Users can specify:

- which datasources to document (by default, all)
- whether to include expectations, validations and profiling results

sections (by default, all) * where the expectations and validations should be read from (filesystem or S3) * where the HTML files should be written (filesystem or S3) * which renderer and view class should be used to render each section

Here is an example of a minimal configuration for a site:

```
local_site:
  class_name: SiteBuilder
  store_backend:
    class_name: TupleS3StoreBackend
    bucket: data_docs.my_company.com
    prefix: /data_docs/
```

A more verbose configuration can also control individual sections and override renderers, views, and stores:

```
local_site:
  class_name: SiteBuilder
  store_backend:
    class_name: TupleS3StoreBackend
    bucket: data_docs.my_company.com
    prefix: /data_docs/
  site_index_builder:
    class_name: DefaultSiteIndexBuilder

# Verbose version:
# index_builder:
#   module_name: great_expectations.render.builder
#   class_name: DefaultSiteIndexBuilder
#   renderer:
#     module_name: great_expectations.render.renderer
#     class_name: SiteIndexPageRenderer
#   view:
#     module_name: great_expectations.render.view
#     class_name: DefaultJinjaIndexPageView

site_section_builders:
  # Minimal specification
  expectations:
    class_name: DefaultSiteSectionBuilder
    source_store_name: expectation_store
  renderer:
    module_name: great_expectations.render.renderer
    class_name: ExpectationSuitePageRenderer

  # More verbose specification with optional arguments
  validations:
    module_name: great_expectations.data_context.render
    class_name: DefaultSiteSectionBuilder
    source_store_name: local_validation_store
    renderer:
      module_name: great_expectations.render.renderer
      class_name: SiteIndexPageRenderer
    view:
      module_name: great_expectations.render.view
      class_name: DefaultJinjaIndexPageView
```

clean_site(self)

build(self, resource_identifiers=None)

Parameters **resource_identifiers** – a list of resource identifiers

(**ExpectationSuiteIdentifier**, **ValidationResultIdentifier**). If specified, rebuild HTML(or other views the

data docs site renders) only for the resources in this list. This supports incremental build of data docs sites (e.g., when a new validation result is created) and avoids full rebuild.

Returns

get_resource_url (*self*, *resource_identifier=None*, *only_if_exists=True*)

Return the URL of the HTML document that renders a resource (e.g., an expectation suite or a validation result).

Parameters *resource_identifier* – ExpectationSuiteIdentifier,

ValidationResultIdentifier or any other type's identifier. The argument is optional - when not supplied, the method returns the URL of the index page. :return: URL (string)

```
class great_expectations.render.renderer.site_builder.DefaultSiteSectionBuilder(name,
                                                                              data_context,
                                                                              tar-
                                                                              get_store,
                                                                              source_store_name,
                                                                              cus-
                                                                              tom_styles_directory,
                                                                              cus-
                                                                              tom_views_directory,
                                                                              show_how_to_run,
                                                                              run_name_filter,
                                                                              val-
                                                                              i-
                                                                              da-
                                                                              tion_results_limit,
                                                                              ren-
                                                                              derer=None,
                                                                              view=None,
                                                                              data_context_identifier,
                                                                              **kwargs)
```

Bases: object

build (*self*, *resource_identifiers=None*)

_resource_key_passes_run_name_filter (*self*, *resource_key*)

```
class great_expectations.render.renderer.site_builder.DefaultSiteIndexBuilder(name,
                                                                              site_name,
                                                                              data_context,
                                                                              tar-
                                                                              get_store,
                                                                              cus-
                                                                              tom_styles_directo
                                                                              cus-
                                                                              tom_views_directo
                                                                              show_how_to_but
                                                                              val-
                                                                              i-
                                                                              da-
                                                                              tion_results_limit=
                                                                              ren-
                                                                              derer=None,
                                                                              view=None,
                                                                              data_context_id=None,
                                                                              source_stores=None,
                                                                              **kwargs)
```

Bases: object

```
add_resource_info_to_index_links_dict(self, index_links_dict, expectation_suite_name,
                                       section_name, batch_identifier=None,
                                       run_id=None, validation_success=None,
                                       run_time=None, run_name=None, as-
                                       set_name=None, batch_kwargs=None)
```

```
get_calls_to_action(self)
```

```
_get_call_to_action_buttons(self, usage_statistics)
```

Build project and user specific calls to action buttons.

This can become progressively smarter about project and user specific calls to action.

```
build(self)
```

```
class great_expectations.render.renderer.site_builder.CallToActionButton(title,
                                                                           link)
```

Bases: object

`great_expectations.render.renderer.site_index_page_renderer`

Module Contents

Classes

`SiteIndexPageRenderer()`

`great_expectations.render.renderer.site_index_page_renderer.logger`

```
class great_expectations.render.renderer.site_index_page_renderer.SiteIndexPageRenderer
```

Bases: `great_expectations.render.renderer.renderer.Renderer`

```
classmethod _generate_expectation_suites_link_table(cls, index_links_dict)
```



```

classmethod _generate_profiling_results_link_table (cls, index_links_dict)
classmethod _generate_validation_results_link_table (cls, index_links_dict)
classmethod _render_expectation_suite_cell (cls, expectation_suite_name, expectation_suite_path)
classmethod _render_batch_id_cell (cls, batch_id, batch_kwargs)
classmethod _get_formatted_datetime (cls, _datetime)
classmethod _get_timestamp (cls, _datetime)
classmethod _render_validation_success_cell (cls, validation_success)
classmethod render (cls, index_links_dict)

```

`great_expectations.render.renderers.slack_renderer`

Module Contents

Classes

SlackRenderer()

```

class great_expectations.render.renderers.slack_renderer.SlackRenderer
    Bases: great_expectations.render.renderers.Renderer
    render (self, validation_result=None)
    _custom_blocks (self, evr)

```

`great_expectations.render.renderers.suite_edit_notebook_renderer`

Module Contents

Classes

SuiteEditNotebookRenderer()

Render a notebook that can re-create or edit a suite.

```

class great_expectations.render.renderers.suite_edit_notebook_renderer.SuiteEditNotebookRenderer
    Bases: great_expectations.render.renderers.Renderer
    Render a notebook that can re-create or edit a suite.
    Use cases: - Make an easy path to edit a suite that a Profiler created. - Make it easy to edit a suite where only JSON exists.
    classmethod _get_expectations_by_column (cls, expectations)
    classmethod _build_kwargs_string (cls, expectation)
    add_header (self, suite_name: str, batch_kwargs)
    add_footer (self)

```

```

add_code_cell (self, code: str, lint: bool = False)
    Add the given code as a new code cell.

add_markdown_cell (self, markdown: str)
    Add the given markdown as a new markdown cell.

add_expectation_cells_from_suite (self, expectations)

_add_column_level_expectations (self, expectations_by_column)

_add_table_level_expectations (self, expectations_by_column)

static _build_meta_arguments (meta)

classmethod write_notebook_to_disk (cls, notebook, notebook_file_path)

render (self, suite: ExpectationSuite, batch_kwargs=None)
    Render a notebook dict from an expectation suite.

render_to_disk (self, suite: ExpectationSuite, notebook_file_path: str, batch_kwargs=None)
    Render a notebook to disk from an expectation suite.

    If batch_kwargs are passed they will override any found in suite citations.

add_authoring_intro (self)

get_batch_kwargs (self, suite: ExpectationSuite, batch_kwargs: Union[dict, BatchKwargs])

static _fix_path_in_batch_kwargs (batch_kwargs)

```

`great_expectations.render.renderer.suite_scaffold_notebook_renderer`

Module Contents

Classes

<code>SuiteScaffoldNotebookRenderer</code> (<i>context</i> : Data-Context, <i>suite</i> : ExpectationSuite, <i>batch_kwargs</i>)	Render a notebook that can re-create or edit a suite.
--	---

```
class great_expectations.render.renderer.suite_scaffold_notebook_renderer.SuiteScaffoldNotebookRenderer
```

Bases: `great_expectations.render.renderer.suite_edit_notebook_renderer.SuiteEditNotebookRenderer`

Render a notebook that can re-create or edit a suite.

Use cases: - Make an easy path to edit a suite that a Profiler created. - Make it easy to edit a suite where only JSON exists.

```

add_header (self)

_add_scaffold_column_list (self)

add_footer (self)

load_batch (self)

render (self, batch_kwargs=None, **kwargs)
    Render a notebook dict from an expectation suite.

render_to_disk (self, notebook_file_path: str)
    Render a notebook to disk from an expectation suite.

    If batch_kwargs are passed they will override any found in suite citations.

_add_scaffold_cell (self)

```

Package Contents

Classes

```

CallToActionRenderer()
ExpectationSuiteColumnSectionRenderer(bullet_list_renderer=None)
ProfilingResultsColumnSectionRenderer(overview_table_renderer=None,
expectation_string_renderer=None,          run-
time_environment=None)
ValidationResultsColumnSectionRenderer(table_renderer=None)
ProfilingResultsOverviewSectionRenderer()
ExpectationSuitePageRenderer(column_section_renderer=None)
ProfilingResultsPageRenderer(overview_section_renderer=None,
column_section_renderer=None)
ValidationResultsPageRenderer(column_section_renderer=None)
SiteIndexPageRenderer()
SlackRenderer()

```

```
class great_expectations.render.renderer.CallToActionRenderer
```

```
    Bases: object
```

```
    _document_defaults
```

```
    classmethod render (cls, cta_object)
```

```
        Parameters cta_object – dict {
```

```
            "header": # optional, can be a string or string template "buttons": # list of CallToAc-
            tionButtons
```

```
        }
```

```
        Returns
```

```
        dict {
```

```
            "header": # optional, can be a string or string template "buttons": # list of CallToAc-
            tionButtons
```

```
        }
```

```
class great_expectations.render.renderer.ExpectationSuiteColumnSectionRenderer (bullet_list_rende
    Bases:          great_expectations.render.renderer.column_section_renderer.
                  ColumnSectionRenderer

    classmethod _render_header (cls, expectations)

    _render_bullet_list (self, expectations)

    render (self, expectations)

class great_expectations.render.renderer.ProfilingResultsColumnSectionRenderer (overview_table_re
    ex-
    pec-
    ta-
    tion_string_rende
    run-
    time_environment

    Bases:          great_expectations.render.renderer.column_section_renderer.
                  ColumnSectionRenderer

    render (self, evrs, section_name=None, column_type=None)

    classmethod _render_header (cls, evrs, column_type=None)

    classmethod _render_expectation_types (cls, evrs, content_blocks)

    _render_overview_table (self, evrs)

    classmethod _render_quantile_table (cls, evrs)

    classmethod _render_stats_table (cls, evrs)

    classmethod _render_values_set (cls, evrs)

    _render_histogram (self, evrs)

    classmethod _render_bar_chart_table (cls, evrs)

    classmethod _render_failed (cls, evrs)

    classmethod _render_unrecognized (cls, evrs, content_blocks)

class great_expectations.render.renderer.ValidationResultsColumnSectionRenderer (table_renderer=
    Bases:          great_expectations.render.renderer.column_section_renderer.
                  ColumnSectionRenderer

    classmethod _render_header (cls, validation_results)

    _render_table (self, validation_results)

    render (self, validation_results)

class great_expectations.render.renderer.ProfilingResultsOverviewSectionRenderer
    Bases: great_expectations.render.renderer.renderer.Renderer

    classmethod render (cls, evrs, section_name=None)

    classmethod _render_header (cls, evrs, content_blocks)

    classmethod _render_dataset_info (cls, evrs, content_blocks)

    classmethod _render_variable_types (cls, evrs, content_blocks)

    classmethod _render_expectation_types (cls, evrs, content_blocks)

    classmethod _render_warnings (cls, evrs, content_blocks)

    classmethod _get_percentage_missing_cells_str (cls, evrs)
```

```

    classmethod _get_column_types (cls, evrs)

class great_expectations.render.renderer.ExpectationSuitePageRenderer (column_section_renderer=None)
    Bases: great_expectations.render.renderer.render.Renderer
    render (self, expectations)
    _render_table_level_expectations (self, columns)
    classmethod _render_expectation_suite_header (cls)
    classmethod _render_expectation_suite_info (cls, expectations)
    classmethod _render_expectation_suite_notes (cls, expectations)

class great_expectations.render.renderer.ProfilingResultsPageRenderer (overview_section_renderer=None,
                                                                    col-
                                                                    umn_section_renderer=None)
    Bases: great_expectations.render.renderer.render.Renderer
    render (self, validation_results)

class great_expectations.render.renderer.ValidationResultsPageRenderer (column_section_renderer=None)
    Bases: great_expectations.render.renderer.render.Renderer
    render (self, validation_results)
    classmethod _render_validation_header (cls, validation_results)
    classmethod _render_validation_info (cls, validation_results)
    classmethod _render_nested_table_from_dict (cls, input_dict, header=None,
                                                sub_table=False)
    classmethod _render_validation_statistics (cls, validation_results)

class great_expectations.render.renderer.SiteIndexPageRenderer
    Bases: great_expectations.render.renderer.render.Renderer
    classmethod _generate_expectation_suites_link_table (cls, index_links_dict)
    classmethod _generate_profiling_results_link_table (cls, index_links_dict)
    classmethod _generate_validation_results_link_table (cls, index_links_dict)
    classmethod _render_expectation_suite_cell (cls, expectation_suite_name, expectation_suite_path)
    classmethod _render_batch_id_cell (cls, batch_id, batch_kwargs)
    classmethod _get_formatted_datetime (cls, _datetime)
    classmethod _get_timestamp (cls, _datetime)
    classmethod _render_validation_success_cell (cls, validation_success)
    classmethod render (cls, index_links_dict)

class great_expectations.render.renderer.SlackRenderer
    Bases: great_expectations.render.renderer.render.Renderer
    render (self, validation_result=None)
    _custom_blocks (self, evr)

```

`great_expectations.render.types`

Package Contents

Classes

<code>RenderedContent()</code>
<code>RenderedComponentContent(content_block_type, styling=None)</code>
<code>RenderedHeaderContent(header, subheader=None, header_row=None, styling=None, content_block_type='header')</code>
<code>RenderedGraphContent(graph, header=None, subheader=None, styling=None, content_block_type='graph')</code>
<code>RenderedTableContent(table, header=None, subheader=None, header_row=None, styling=None, content_block_type='table', table_options=None, header_row_options=None)</code>
<code>RenderedTabsContent(tabs, header=None, subheader=None, styling=None, content_block_type='tabs')</code>
<code>RenderedBootstrapTableContent(table_data, table_columns, title_row=None, table_options=None, header=None, subheader=None, styling=None, content_block_type='bootstrap_table')</code>
<code>RenderedContentBlockContainer(content_blocks, styling=None, content_block_type='content_block_container')</code>
<code>RenderedMarkdownContent(markdown, styling=None, content_block_type='markdown')</code>
<code>RenderedStringTemplateContent(string_template, styling=None, content_block_type='string_template')</code>
<code>RenderedBulletListContent(bullet_list, header=None, subheader=None, styling=None, content_block_type='bullet_list')</code>
<code>ValueListContent(value_list, header=None, subheader=None, styling=None, content_block_type='value_list')</code>
<code>TextContent(text, header=None, subheader=None, styling=None, content_block_type='text')</code>
<code>CollapseContent(collapse, collapse_toggle_link=None, header=None, subheader=None, styling=None, content_block_type='collapse', inline_link=False)</code>
<code>RenderedDocumentContent(sections, data_asset_name=None, full_data_asset_identifier=None, renderer_type=None, page_title=None, utm_medium=None, cta_footer=None, expectation_suite_name=None, batch_kwargs=None)</code>
<code>RenderedSectionContent(content_blocks, section_name=None)</code>

exception `great_expectations.render.types.InvalidRenderedContentError`

Bases: *great_expectations.exceptions.GreatExpectationsTypeError*

Inappropriate argument type.

class `great_expectations.render.types.RenderedContent`

Bases: `object`

to_json_dict (*self*)

__eq__ (*self*, *other*)

Return `self==value`.

classmethod `rendered_content_list_to_json` (*cls*, *list_*, *check_dicts=False*)

classmethod `rendered_content_dict_to_json` (*cls*, *dict_*, *check_list_dicts=True*)

class `great_expectations.render.types.RenderedComponentContent` (*content_block_type*,
styling=None)

Bases: *great_expectations.render.types.RenderedContent*

to_json_dict (*self*)

class `great_expectations.render.types.RenderedHeaderContent` (*header*, *sub-*
header=None,
header_row=None,
styling=None, *con-*
tent_block_type='header')

Bases: *great_expectations.render.types.RenderedComponentContent*

to_json_dict (*self*)

class `great_expectations.render.types.RenderedGraphContent` (*graph*, *header=None*,
subheader=None,
styling=None, *con-*
tent_block_type='graph')

Bases: *great_expectations.render.types.RenderedComponentContent*

to_json_dict (*self*)

class `great_expectations.render.types.RenderedTableContent` (*table*, *header=None*,
subheader=None,
header_row=None,
styling=None, *con-*
tent_block_type='table',
table_options=None,
header_row_options=None)

Bases: *great_expectations.render.types.RenderedComponentContent*

to_json_dict (*self*)

class `great_expectations.render.types.RenderedTabsContent` (*tabs*, *header=None*,
subheader=None,
styling=None, *con-*
tent_block_type='tabs')

Bases: *great_expectations.render.types.RenderedComponentContent*

to_json_dict (*self*)

```
class great_expectations.render.types.RenderedBootstrapTableContent (table_data,
                                                                    ta-
                                                                    ble_columns,
                                                                    ti-
                                                                    tle_row=None,
                                                                    ta-
                                                                    ble_options=None,
                                                                    header=None,
                                                                    sub-
                                                                    header=None,
                                                                    styling=None,
                                                                    con-
                                                                    tent_block_type='bootstrap_table')

    Bases: great_expectations.render.types.RenderedComponentContent
    to_json_dict (self)

class great_expectations.render.types.RenderedContentBlockContainer (content_blocks,
                                                                    styling=None,
                                                                    con-
                                                                    tent_block_type='content_block_')

    Bases: great_expectations.render.types.RenderedComponentContent
    to_json_dict (self)

class great_expectations.render.types.RenderedMarkdownContent (markdown,
                                                                    styling=None,
                                                                    con-
                                                                    tent_block_type='markdown')

    Bases: great_expectations.render.types.RenderedComponentContent
    to_json_dict (self)

class great_expectations.render.types.RenderedStringTemplateContent (string_template,
                                                                    styling=None,
                                                                    con-
                                                                    tent_block_type='string_template')

    Bases: great_expectations.render.types.RenderedComponentContent
    to_json_dict (self)

class great_expectations.render.types.RenderedBulletListContent (bullet_list,
                                                                    header=None,
                                                                    sub-
                                                                    header=None,
                                                                    styling=None,
                                                                    con-
                                                                    tent_block_type='bullet_list')

    Bases: great_expectations.render.types.RenderedComponentContent
    to_json_dict (self)

class great_expectations.render.types.ValueListContent (value_list, header=None,
                                                                    subheader=None,
                                                                    styling=None, content_block_type='value_list')

    Bases: great_expectations.render.types.RenderedComponentContent
    to_json_dict (self)
```



```

class great_expectations.render.types.TextContent (text, header=None, sub-
                                                    header=None, styling=None,
                                                    content_block_type='text')
    Bases: great_expectations.render.types.RenderedComponentContent
    to_json_dict (self)

class great_expectations.render.types.CollapseContent (collapse, col-
                                                         lapse_toggle_link=None,
                                                         header=None, sub-
                                                         header=None,
                                                         styling=None, con-
                                                         tent_block_type='collapse',
                                                         inline_link=False)
    Bases: great_expectations.render.types.RenderedComponentContent
    to_json_dict (self)

class great_expectations.render.types.RenderedDocumentContent (sections,
                                                                data_asset_name=None,
                                                                full_data_asset_identifer=None,
                                                                ren-
                                                                derer_type=None,
                                                                page_title=None,
                                                                utm_medium=None,
                                                                cta_footer=None,
                                                                expecta-
                                                                tion_suite_name=None,
                                                                batch_kwargs=None)
    Bases: great_expectations.render.types.RenderedContent
    to_json_dict (self)

class great_expectations.render.types.RenderedSectionContent (content_blocks,
                                                                sec-
                                                                tion_name=None)
    Bases: great_expectations.render.types.RenderedContent
    to_json_dict (self)

```

great_expectations.render.view

Submodules

great_expectations.render.view.view

Module Contents

Classes

NoOpTemplate()

PrettyPrintTemplate()

DefaultJinjaView(custom_styles_directory=None,
custom_views_directory=None)

Defines a method for converting a document to human-
consumable form

continues on next page

Table 119 – continued from previous page

<code>DefaultJinjaPageView(custom_styles_directory=None, custom_views_directory=None)</code>	Defines a method for converting a document to human-consumable form
<code>DefaultJinjaIndexPageView(custom_styles_directory=None, custom_views_directory=None)</code>	Defines a method for converting a document to human-consumable form
<code>DefaultJinjaSectionView(custom_styles_directory=None, custom_views_directory=None)</code>	Defines a method for converting a document to human-consumable form
<code>DefaultJinjaComponentView(custom_styles_directory=None, custom_views_directory=None)</code>	Defines a method for converting a document to human-consumable form

```
class great_expectations.render.view.view.NoOpTemplate
```

```
    Bases: object
```

```
    render (self, document)
```

```
class great_expectations.render.view.view.PrettyPrintTemplate
```

```
    Bases: object
```

```
    render (self, document, indent=2)
```

```
class great_expectations.render.view.view.DefaultJinjaView (custom_styles_directory=None,
                                                             cus-
                                                             tom_views_directory=None)
```

```
    Bases: object
```

```
    Defines a method for converting a document to human-consumable form
```

- Font Awesome 5.10.1
- Bootstrap 4.3.1
- jQuery 3.2.1
- Vega 5
- Vega-Lite 4
- Vega-Embed 6

```
    _template
```

```
    render (self, document, template=None, **kwargs)
```

```
    _get_template (self, template)
```

```
    add_data_context_id_to_url (self, jinja_context, url, add_datetime=True)
```

```
    render_content_block (self, jinja_context, content_block, index=None, content_block_id=None)
```

```
    render_dict_values (self, context, dict_, index=None, content_block_id=None)
```

```
    render_bootstrap_table_data (self, context, table_data, index=None, con-
                                tent_block_id=None)
```

```
    get_html_escaped_json_string_from_dict (self, source_dict)
```

```
    attributes_dict_to_html_string (self, attributes_dict, prefix="")
```

```
    render_styling (self, styling)
```

```
        Adds styling information suitable for an html tag.
```

```
        Example styling block:
```

```
styling = {
    "classes": ["alert", "alert-warning"],
    "attributes": {
        "role": "alert",
        "data-toggle": "popover",
    },
    "styles": {
        "padding": "10px",
        "border-radius": "2px",
    }
}
```

The above block returns a string similar to:

```
'class="alert alert-warning" role="alert" data-toggle="popover" style=
↪"padding: 10px; border-radius: 2px"'
```

“classes”, “attributes” and “styles” are all optional parameters. If they aren’t present, they simply won’t be rendered.

Other dictionary keys are also allowed and ignored.

render_styling_from_string_template (*self*, *template*)

This method is a thin wrapper use to call *render_styling* from within jinja templates.

generate_html_element_uuid (*self*, *prefix=None*)

render_markdown (*self*, *markdown*)

render_string_template (*self*, *template*)

abstract _validate_document (*self*, *document*)

class great_expectations.render.view.view.DefaultJinjaPageView (*custom_styles_directory=None*,
cus-
tom_views_directory=None)

Bases: *great_expectations.render.view.view.DefaultJinjaView*

Defines a method for converting a document to human-consumable form

- Font Awesome 5.10.1
- Bootstrap 4.3.1
- jQuery 3.2.1
- Vega 5
- Vega-Lite 4
- Vega-Embed 6

_template = **page.j2**

_validate_document (*self*, *document*)

class great_expectations.render.view.view.DefaultJinjaIndexPageView (*custom_styles_directory=None*,
cus-
tom_views_directory=None)

Bases: *great_expectations.render.view.view.DefaultJinjaPageView*

Defines a method for converting a document to human-consumable form

- Font Awesome 5.10.1

- Bootstrap 4.3.1
- jQuery 3.2.1
- Vega 5
- Vega-Lite 4
- Vega-Embed 6

_template = index_page.j2

class great_expectations.render.view.view.**DefaultJinjaSectionView** (*custom_styles_directory=None, cus-
tom_views_directory=None*)

Bases: *great_expectations.render.view.view.DefaultJinjaView*

Defines a method for converting a document to human-consumable form

- Font Awesome 5.10.1
- Bootstrap 4.3.1
- jQuery 3.2.1
- Vega 5
- Vega-Lite 4
- Vega-Embed 6

_template = section.j2

_validate_document (*self, document*)

class great_expectations.render.view.view.**DefaultJinjaComponentView** (*custom_styles_directory=None, cus-
tom_views_directory=None*)

Bases: *great_expectations.render.view.view.DefaultJinjaView*

Defines a method for converting a document to human-consumable form

- Font Awesome 5.10.1
- Bootstrap 4.3.1
- jQuery 3.2.1
- Vega 5
- Vega-Lite 4
- Vega-Embed 6

_template = component.j2

_validate_document (*self, document*)

Package Contents

Classes

<code>DefaultJinjaComponentView</code> (<code>custom_styles_directory=None</code> , <code>custom_views_directory=None</code>)	Defines a method for converting a document to human-consumable form
<code>DefaultJinjaIndexPageView</code> (<code>custom_styles_directory=None</code> , <code>custom_views_directory=None</code>)	Defines a method for converting a document to human-consumable form
<code>DefaultJinjaPageView</code> (<code>custom_styles_directory=None</code> , <code>custom_views_directory=None</code>)	Defines a method for converting a document to human-consumable form
<code>DefaultJinjaSectionView</code> (<code>custom_styles_directory=None</code> , <code>custom_views_directory=None</code>)	Defines a method for converting a document to human-consumable form

```
class great_expectations.render.view.DefaultJinjaComponentView(custom_styles_directory=None,
                                                                cus-
                                                                tom_views_directory=None)
```

Bases: `great_expectations.render.view.view.DefaultJinjaView`

Defines a method for converting a document to human-consumable form

- Font Awesome 5.10.1
- Bootstrap 4.3.1
- jQuery 3.2.1
- Vega 5
- Vega-Lite 4
- Vega-Embed 6

`_template = component.j2`

`_validate_document` (*self*, *document*)

```
class great_expectations.render.view.DefaultJinjaIndexPageView(custom_styles_directory=None,
                                                                cus-
                                                                tom_views_directory=None)
```

Bases: `great_expectations.render.view.view.DefaultJinjaPageView`

Defines a method for converting a document to human-consumable form

- Font Awesome 5.10.1
- Bootstrap 4.3.1
- jQuery 3.2.1
- Vega 5
- Vega-Lite 4
- Vega-Embed 6

`_template = index_page.j2`

```
class great_expectations.render.view.DefaultJinjaPageView(custom_styles_directory=None,
                                                           cus-
                                                           tom_views_directory=None)
```

Bases: `great_expectations.render.view.view.DefaultJinjaView`

Defines a method for converting a document to human-consumable form

- Font Awesome 5.10.1
- Bootstrap 4.3.1
- jQuery 3.2.1
- Vega 5
- Vega-Lite 4
- Vega-Embed 6

`_template = page.j2`

`_validate_document (self, document)`

class great_expectations.render.view.DefaultJinjaSectionView (*custom_styles_directory=None, cus-
tom_views_directory=None*)

Bases: *great_expectations.render.view.view.DefaultJinjaView*

Defines a method for converting a document to human-consumable form

- Font Awesome 5.10.1
- Bootstrap 4.3.1
- jQuery 3.2.1
- Vega 5
- Vega-Lite 4
- Vega-Embed 6

`_template = section.j2`

`_validate_document (self, document)`

Submodules

`great_expectations.render.exceptions`

Module Contents

exception great_expectations.render.exceptions.InvalidRenderedContentError

Bases: *great_expectations.exceptions.GreatExpectationsTypeError*

Inappropriate argument type.

`great_expectations.render.util`

Rendering utility

Module Contents

Functions

<code>num_to_str(f, precision=DEFAULT_PRECISION, use_locale=False, no_scientific=False)</code>	Convert the given float to a string, centralizing standards for precision and decisions about scientific
<code>ordinal(num)</code>	Convert a number to ordinal

`great_expectations.render.util.DEFAULT_PRECISION = 4`

`great_expectations.render.util.ctx`

`great_expectations.render.util.prec`

`great_expectations.render.util.num_to_str(f, precision=DEFAULT_PRECISION, use_locale=False, no_scientific=False)`

Convert the given float to a string, centralizing standards for precision and decisions about scientific notation. Adds an approximately equal sign in the event precision loss (e.g. rounding) has occurred.

There's a good discussion of related issues here: <https://stackoverflow.com/questions/38847690/convert-float-to-string-in-positional-format-without-scientific-notation-and-fa>

Parameters

- **f** – the number to format
- **precision** – the number of digits of precision to display
- **use_locale** – if True, use locale-specific formatting (e.g. adding thousands separators)
- **no_scientific** – if True, print all available digits of precision without scientific notation. This may insert leading zeros before very small numbers, causing the resulting string to be longer than *precision* characters

Returns A string representation of the float, according to the desired parameters

`great_expectations.render.util.SUFFIXES`

`great_expectations.render.util.ordinal(num)`
Convert a number to ordinal

Package Contents

Classes

<code>DefaultJinjaPageView(custom_styles_directory=None, custom_views_directory=None)</code>	Defines a method for converting a document to human-consumable form
--	---

class `great_expectations.render.DefaultJinjaPageView(custom_styles_directory=None, custom_views_directory=None)`

Bases: `great_expectations.render.view.view.DefaultJinjaView`

Defines a method for converting a document to human-consumable form

- Font Awesome 5.10.1

- Bootstrap 4.3.1
- jQuery 3.2.1
- Vega 5
- Vega-Lite 4
- Vega-Embed 6

```
_template = page.j2
_validate_document (self, document)
```

`great_expectations.types`

Submodules

`great_expectations.types.base`

Module Contents

Classes

<code>DotDict()</code>	This class provides dot.notation dot.notation access to dictionary attributes.
------------------------	--

`great_expectations.types.base.logger`

`great_expectations.types.base.yaml`

class `great_expectations.types.base.DotDict`

Bases: `dict`

This class provides dot.notation dot.notation access to dictionary attributes.

It is also serializable by the ruamel.yaml library used in Great Expectations for managing configuration objects.

`__setattr__`

`__delattr__`

`_yaml_merge` = []

`__getattr__` (*self, item*)

`__dir__` (*self*)

Default dir() implementation.

`__deepcopy__` (*self, memo*)

classmethod `yaml_anchor` (*cls*)

classmethod `to_yaml` (*cls, representer, node*)

Use dict representation for DotDict (and subtypes by default)

`great_expectations.types.configurations`

Module Contents

Classes

<code>ClassConfig</code> (class_name, module_name=None)	Defines information sufficient to identify a class to be (dynamically) loaded for a DataContext.
<code>ClassConfigSchema</code> (*, only: types.StrSequenceOrSet = None, exclude: types.StrSequenceOrSet = (), many: bool = False, context: typing.Dict = None, load_only: types.StrSequenceOrSet = (), dump_only: types.StrSequenceOrSet = (), partial: typing.Union[bool, types.StrSequenceOrSet] = False, unknown: str = None)	Base schema class with which to define custom schemas.

```
class great_expectations.types.configurations.ClassConfig(class_name, module_name=None)
```

Bases: object

Defines information sufficient to identify a class to be (dynamically) loaded for a DataContext.

property class_name (self)

property module_name (self)

```
class great_expectations.types.configurations.ClassConfigSchema(*, only:
    types.StrSequenceOrSet
    = None,
    exclude:
    types.StrSequenceOrSet
    = (), many:
    bool = False,
    context:
    typing.Dict
    = None,
    load_only:
    types.StrSequenceOrSet
    = (),
    dump_only:
    types.StrSequenceOrSet
    = (), partial:
    typing.Union[bool,
    types.StrSequenceOrSet]
    = False, unknown: str =
    None)
```

Bases: `marshmallow.Schema`

Base schema class with which to define custom schemas.

Example usage:

```
import datetime as dt
from dataclasses import dataclass

from marshmallow import Schema, fields

@dataclass
class Album:
    title: str
    release_date: dt.date

class AlbumSchema(Schema):
    title = fields.Str()
    release_date = fields.Date()

album = Album("Beggars Banquet", dt.date(1968, 12, 6))
schema = AlbumSchema()
data = schema.dump(album)
data # {'release_date': '1968-12-06', 'title': 'Beggars Banquet'}
```

Parameters

- **only** – Whitelist of the declared fields to select when instantiating the Schema. If None, all fields are used. Nested fields can be represented with dot delimiters.
- **exclude** – Blacklist of the declared fields to exclude when instantiating the Schema. If a field appears in both *only* and *exclude*, it is not used. Nested fields can be represented with dot delimiters.
- **many** – Should be set to *True* if *obj* is a collection so that the object will be serialized to a list.
- **context** – Optional context passed to *fields.Method* and *fields.Function* fields.
- **load_only** – Fields to skip during serialization (write-only fields)
- **dump_only** – Fields to skip during deserialization (read-only fields)
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to Nested fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.
- **unknown** – Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*.

Changed in version 3.0.0: *prefix* parameter removed.

Changed in version 2.0.0: *__validators__*, *__preprocessors__*, and *__data_handlers__* are removed in favor of *marshmallow.decorators.validates_schema*, *marshmallow.decorators.pre_load* and *marshmallow.decorators.post_dump*. *__accessor__* and *__error_handler__* are deprecated. Implement the *handle_error* and *get_attribute* methods instead.

class_name

module_name

`great_expectations.types.configurations.classConfigSchema`

`great_expectations.types.expectations`

Package Contents

Classes

<code>ClassConfig</code> (class_name, module_name=None)	Defines information sufficient to identify a class to be (dynamically) loaded for a DataContext.
<code>DictDot</code> ()	

```
class great_expectations.types.ClassConfig (class_name, module_name=None)
    Bases: object

    Defines information sufficient to identify a class to be (dynamically) loaded for a DataContext.

    property class_name (self)
    property module_name (self)
```

```
class great_expectations.types.DictDot
    Bases: object

    __getitem__ (self, item)
    __setitem__ (self, key, value)
    __delitem__ (self, key)
```

`great_expectations.validation_operators`

Subpackages

`great_expectations.validation_operators.types`

Submodules

`great_expectations.validation_operators.types.validation_operator_result`

Module Contents

Classes

<code>ValidationOperatorResult</code> (run_id: RunIdentifier, run_results: Dict[ValidationResultIdentifier, Dict[str, Union[ExpectationSuiteValidationResult, dict, str]]], validation_operator_config, evaluation_parameters: dict = None, success: bool = None)	The run_results property forms the backbone of this type and defines the basic contract for what a validation
---	---

continues on next page

Table 126 – continued from previous page

<i>ValidationOperatorResultSchema</i> (*, only: types.StrSequenceOrSet = None, exclude: types.StrSequenceOrSet = (), many: bool = False, context: typing.Dict = None, load_only: types.StrSequenceOrSet = (), dump_only: types.StrSequenceOrSet = (), partial: typing.Union[bool, types.StrSequenceOrSet] = False, unknown: str = None)	Base schema class with which to define custom schemas.
---	--

```
class great_expectations.validation_operators.types.validation_operator_result.ValidationOperatorResult
```

Bases: *great_expectations.types.DictDot*

The `run_results` property forms the backbone of this type and defines the basic contract for what a validation operator’s `run` method returns. It is a dictionary where the top-level keys are the `ValidationResultIdentifiers` of the validation results generated in the run. Each value is a dictionary having at minimum, a `validation_result` key; this dictionary can contain other keys that are relevant for a specific validation operator implementation. For example, the dictionary from a `WarningAndFailureExpectationSuitesValidationOperator` would have an extra key named `expectation_suite_severity_level` to indicate if the suite is at either a “warning” or “failure” level, as well as an `actions_results` key.

e.g. {

```
    ValidationResultIdentifier: { “validation_result”: ExpectationSuiteValidationResult, “ac-  
                                tions_results”: {}  
    }  
}
```

```
property validation_operator_config (self)  
property run_results (self)
```

```
property run_id(self)
property evaluation_parameters(self)
property success(self)
list_batch_identifiers(self)
list_data_asset_names(self)
list_expectation_suite_names(self)
list_validation_result_identifiers(self)
list_validation_results(self, group_by=None)
_list_validation_results_by_validation_result_identifier(self)
_list_validation_results_by_expectation_suite_name(self)
_list_validation_results_by_data_asset_name(self)
list_data_assets_validated(self, group_by: str = None)
_list_data_assets_validated_by_batch_id(self)
get_statistics(self)
_list_validation_statistics(self)
to_json_dict(self)
__repr__(self)
    Return repr(self).
```

class great_expectations.validation_operators.types.validation_operator_result.ValidationOperatorResult

Bases: `marshmallow.Schema`

Base schema class with which to define custom schemas.

Example usage:

```
import datetime as dt
from dataclasses import dataclass

from marshmallow import Schema, fields


@dataclass
class Album:
    title: str
```

(continues on next page)

(continued from previous page)

```

release_date: dt.date

class AlbumSchema(Schema):
    title = fields.Str()
    release_date = fields.Date()

album = Album("Beggars Banquet", dt.date(1968, 12, 6))
schema = AlbumSchema()
data = schema.dump(album)
data # {'release_date': '1968-12-06', 'title': 'Beggars Banquet'}
```

Parameters

- **only** – Whitelist of the declared fields to select when instantiating the Schema. If None, all fields are used. Nested fields can be represented with dot delimiters.
- **exclude** – Blacklist of the declared fields to exclude when instantiating the Schema. If a field appears in both *only* and *exclude*, it is not used. Nested fields can be represented with dot delimiters.
- **many** – Should be set to *True* if *obj* is a collection so that the object will be serialized to a list.
- **context** – Optional context passed to `fields.Method` and `fields.Function` fields.
- **load_only** – Fields to skip during serialization (write-only fields)
- **dump_only** – Fields to skip during deserialization (read-only fields)
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to Nested fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.
- **unknown** – Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*.

Changed in version 3.0.0: *prefix* parameter removed.

Changed in version 2.0.0: `__validators__`, `__preprocessors__`, and `__data_handlers__` are removed in favor of `marshmallow.decorators.validates_schema`, `marshmallow.decorators.pre_load` and `marshmallow.decorators.post_dump`. `__accessor__` and `__error_handler__` are deprecated. Implement the `handle_error` and `get_attribute` methods instead.

run_id

run_results

evaluation_parameters

validation_operator_config

success

prepare_dump (*self*, *data*, ***kwargs*)

make_expectation_suite_validation_result (*self*, *data*, ***kwargs*)

`great_expectations.validation_operators.types.validation_operator_result.validationOperator`

Submodules

`great_expectations.validation_operators.actions`

An action is a way to take an arbitrary method and make it configurable and runnable within a Data Context.

The only requirement from an action is for it to have a `take_action` method.

Module Contents

Classes

<code>ValidationAction(data_context)</code>	This is the base class for all actions that act on validation results
<code>NoOpAction(data_context)</code>	This is the base class for all actions that act on validation results
<code>SlackNotificationAction(data_context, renderer, slack_webhook, notify_on='all')</code>	SlackNotificationAction sends a Slack notification to a given webhook.
<code>StoreValidationResultAction(data_context, target_store_name=None)</code>	StoreValidationResultAction stores a validation result in the ValidationsStore.
<code>StoreEvaluationParametersAction(data_context, target_store_name=None)</code>	StoreEvaluationParametersAction extracts evaluation parameters from a validation result and stores them in the store
<code>StoreMetricsAction(data_context, requested_metrics, target_store_name='metrics_store')</code>	re-StoreMetricsAction extracts metrics from a Validation Result and stores them
<code>UpdateDataDocsAction(data_context, site_names=None, target_site_names=None)</code>	UpdateDataDocsAction is a validation action that

`great_expectations.validation_operators.actions.logger`

class `great_expectations.validation_operators.actions.ValidationAction(data_context)`
Bases: `object`

This is the base class for all actions that act on validation results and are aware of a Data Context namespace structure.

The Data Context is passed to this class in its constructor.

run (*self*, *validation_result_suite*, *validation_result_suite_identifier*, *data_asset*, ***kwargs*)

Parameters

- **validation_result_suite** –
- **validation_result_suite_identifier** –
- **data_asset** –

Param *kwargs* - any additional arguments the child might use

Returns

_run (*self*, *validation_result_suite*, *validation_result_suite_identifier*, *data_asset*)

class `great_expectations.validation_operators.actions.NoOpAction(data_context)`
Bases: `great_expectations.validation_operators.actions.ValidationAction`

This is the base class for all actions that act on validation results and are aware of a Data Context namespace structure.

The Data Context is passed to this class in its constructor.

```
_run (self, validation_result_suite, validation_result_suite_identifier, data_asset)
```

```
class great_expectations.validation_operators.actions.SlackNotificationAction (data_context,
                                                                              ren-
                                                                              derer,
                                                                              slack_webhook,
                                                                              no-
                                                                              tify_on='all')
```

Bases: `great_expectations.validation_operators.actions.ValidationAction`

SlackNotificationAction sends a Slack notification to a given webhook.

Configuration

```
- name: send_slack_notification_on_validation_result
action:
  class_name: StoreValidationResultAction
  # put the actual webhook URL in the uncommitted/config_variables.yml file
  slack_webhook: ${validation_notification_slack_webhook}
  notify_on: all # possible values: "all", "failure", "success"
  renderer:
    # the class that implements the message to be sent
    # this is the default implementation, but you can
    # implement a custom one
    module_name: great_expectations.render.renderer.slack_renderer
    class_name: SlackRenderer
```

```
_run (self, validation_result_suite, validation_result_suite_identifier, data_asset=None)
```

```
class great_expectations.validation_operators.actions.StoreValidationResultAction (data_context,
                                                                              tar-
                                                                              get_store_name)
```

Bases: `great_expectations.validation_operators.actions.ValidationAction`

StoreValidationResultAction stores a validation result in the ValidationsStore.

Configuration

```
- name: store_validation_result
action:
  class_name: StoreValidationResultAction
  # name of the store where the actions will store validation results
  # the name must refer to a store that is configured in the great_expectations.
  ↪yml file
  target_store_name: validations_store
```

```
_run (self, validation_result_suite, validation_result_suite_identifier, data_asset)
```

```
class great_expectations.validation_operators.actions.StoreEvaluationParametersAction (data_context,
                                                                              tar-
                                                                              get_store_name)
```

Bases: `great_expectations.validation_operators.actions.ValidationAction`

StoreEvaluationParametersAction extracts evaluation parameters from a validation result and stores them in the store configured for this action.

Evaluation parameters allow expectations to refer to statistics/metrics computed in the process of validating other prior expectations.

Configuration

```
- name: store_evaluation_params
action:
  class_name: StoreEvaluationParametersAction
  # name of the store where the action will store the parameters
  # the name must refer to a store that is configured in the great_expectations.
  ↪yaml file
  target_store_name: evaluation_parameter_store
```

```
__run (self, validation_result_suite, validation_result_suite_identifier, data_asset)
```

```
class great_expectations.validation_operators.actions.StoreMetricsAction (data_context,
                                                                              re-
                                                                              requested_metrics,
                                                                              tar-
                                                                              get_store_name='metrics')
```

Bases: *great_expectations.validation_operators.actions.ValidationAction*

StoreMetricsAction extracts metrics from a Validation Result and stores them in a metrics store.

Configuration

```
- name: store_evaluation_params
action:
  class_name: StoreMetricsAction
  # name of the store where the action will store the metrics
  # the name must refer to a store that is configured in the great_expectations.
  ↪yaml file
  target_store_name: my_metrics_store
```

```
__run (self, validation_result_suite, validation_result_suite_identifier, data_asset)
```

```
class great_expectations.validation_operators.actions.UpdateDataDocsAction (data_context,
                                                                              site_names=None,
                                                                              tar-
                                                                              get_site_names=None)
```

Bases: *great_expectations.validation_operators.actions.ValidationAction*

UpdateDataDocsAction is a validation action that notifies the site builders of all the data docs sites of the Data Context that a validation result should be added to the data docs.

Configuration

```
- name: update_data_docs
action:
  class_name: UpdateDataDocsAction
```

You can also instruct UpdateDataDocsAction to build only certain sites by providing a `site_names` key with a list of sites to update:

- name: update_data_docs

action: class_name: UpdateDataDocsAction site_names:

- production_site

```
__run (self, validation_result_suite, validation_result_suite_identifier, data_asset)
```

`great_expectations.validation_operators.util`

Module Contents

Functions

`send_slack_notification(query, slack_webhook)`

`great_expectations.validation_operators.util.logger`

`great_expectations.validation_operators.util.send_slack_notification(query, slack_webhook)`

`great_expectations.validation_operators.validation_operators`

Module Contents

Classes

<code>ValidationOperator()</code>	The base class of all validation operators.
<code>ActionListValidationOperator(data_context, action_list, name, result_format={ 'result_format': 'SUMMARY' })</code>	ActionListValidationOperator validates each batch in its run method's <code>assets_to_validate</code> argument against the Expectation Suite included within that batch.
<code>WarningAndFailureExpectationSuitesValidationOperator(data_context, action_list, name, base_expectation_suite_name=None, expectation_suite_name_suffixes=None, stop_on_first_error=False, slack_webhook=None, notify_on='all', result_format={ 'result_format': 'SUMMARY' })</code>	WarningAndFailureExpectationSuitesValidationOperator is a validation operator

`great_expectations.validation_operators.validation_operators.logger`

`great_expectations.validation_operators.validation_operators.logger`

class `great_expectations.validation_operators.validation_operators.ValidationOperator`
 Bases: `object`

The base class of all validation operators.

It defines the signature of the public `run` method. This method and the `validation_operator_config` property are the only contract re operators' API. Everything else is up to the implementors of validation operator classes that will be the descendants of this base class.

property `validation_operator_config(self)`

This method builds the config dict of a particular validation operator. The “kwargs” key is what really distinguishes different validation operators.

e.g.: {

```

    "class_name": "ActionListValidationOperator",          "module_name":
    "great_expectations.validation_operators", "name": self.name, "kwargs": {
        "action_list": self.action_list

```

```
        },
    }
}

{ "class_name":      "WarningAndFailureExpectationSuitesValidationOperator",    "module_name":
  "great_expectations.validation_operators", "name": self.name, "kwargs": {
    "action_list":      self.action_list,      "base_expectation_suite_name":
    self.base_expectation_suite_name,      "expectation_suite_name_suffixes":
    self.expectation_suite_name_suffixes,    "stop_on_first_error":    self.stop_on_first_error,
    "slack_webhook": self.slack_webhook, "notify_on": self.notify_on,
  },
}

}

abstract run (self,      assets_to_validate,      run_id=None,      evaluation_parameters=None,
               run_name=None, run_time=None)

class great_expectations.validation_operators.validation_operators.ActionListValidationOperator
```

Bases: `great_expectations.validation_operators.validation_operators.ValidationOperator`

ActionListValidationOperator validates each batch in its run method's `assets_to_validate` argument against the Expectation Suite included within that batch.

Then it invokes a list of configured actions on every validation result.

Each action in the list must be an instance of `ValidationAction` class (or its descendants). See the actions included in Great Expectations and how to configure them [here](#). You can also implement your own actions by extending the base class.

The init command includes this operator in the default configuration file.

Configuration

An instance of ActionListValidationOperator is included in the default configuration file `great_expectations.yml` that `great_expectations init` command creates.

```
perform_action_list_operator: # this is the name you will use when you invoke_
→the operator
  class_name: ActionListValidationOperator

  # the operator will call the following actions on each validation result
  # you can remove or add actions to this list. See the details in the actions
  # reference
  action_list:
    - name: store_validation_result
      action:
        class_name: StoreValidationResultAction
        target_store_name: validations_store
    - name: send_slack_notification_on_validation_result
      action:
        class_name: SlackNotificationAction
        # put the actual webhook URL in the uncommitted/config_variables.yml file
```

(continues on next page)

(continued from previous page)

```

slack_webhook: ${validation_notification_slack_webhook}
notify_on: all # possible values: "all", "failure", "success"
renderer:
    module_name: great_expectations.render.renderer.slack_renderer
    class_name: SlackRenderer
- name: update_data_docs
  action:
    class_name: UpdateDataDocsAction

```

Invocation

This is an example of invoking an instance of a Validation Operator from Python:

```

results = context.run_validation_operator(
    assets_to_validate=[batch0, batch1, ...],
    run_id=RunIdentifier(**{
        "run_name": "some_string_that_uniquely_identifies_this_run",
        "run_time": "2020-04-29T10:46:03.197008" # optional run timestamp,
        ↪ defaults to current UTC datetime
    }), # you may also pass in a dictionary with run_name and run_time keys
    validation_operator_name="operator_instance_name",
)

```

- `assets_to_validate` - an iterable that specifies the data assets that the operator will validate. The members of the list can be either batches or triples that will allow the operator to fetch the batch: (data_asset_name, expectation_suite_name, batch_kwargs) using this method: `get_batch()`
- `run_id` - pipeline run id of type `RunIdentifier`, consisting of a `run_time` (always assumed to be UTC time) and `run_name` string that is meaningful to you and will help you refer to the result of this operation later
- `validation_operator_name` you can instances of a class that implements a Validation Operator

The run method returns a `ValidationOperatorResult` object:

```

{
  "run_id": {"run_time": "20200527T041833.074212Z", "run_name": "my_run_name"},
  "success": True,
  "evaluation_parameters": None,
  "validation_operator_config": {
    "class_name": "ActionListValidationOperator",
    "module_name": "great_expectations.validation_operators",
    "name": "action_list_operator",
    "kwargs": {
      "action_list": [
        {
          "name": "store_validation_result",
          "action": {"class_name": "StoreValidationResultAction"},
        },
        {
          "name": "store_evaluation_params",
          "action": {"class_name": "StoreEvaluationParametersAction"},
        },
        {
          "name": "update_data_docs",
          "action": {"class_name": "UpdateDataDocsAction"},
        }
      ]
    }
  }
}

```

(continues on next page)

(continued from previous page)

```

        },
    },
    },
    "run_results": {
        ValidationResultIdentifier: {
            "validation_result": ExpectationSuiteValidationResult object,
            "actions_results": {
                "store_validation_result": {},
                "store_evaluation_params": {},
                "update_data_docs": {},
            },
        },
    },
}

```

property validation_operator_config (self)

This method builds the config dict of a particular validation operator. The “kwargs” key is what really distinguishes different validation operators.

e.g.: {

```

    "class_name": "ActionListValidationOperator",    "module_name":
    "great_expectations.validation_operators", "name": self.name, "kwargs": {
        "action_list": self.action_list
    },
}

```

```

{ "class_name": "WarningAndFailureExpectationSuitesValidationOperator",    "module_name":
  "great_expectations.validation_operators", "name": self.name, "kwargs": {
      "action_list": self.action_list,    "base_expectation_suite_name":
      self.base_expectation_suite_name,    "expectation_suite_name_suffixes":
      self.expectation_suite_name_suffixes,    "stop_on_first_error": self.stop_on_first_error,
      "slack_webhook": self.slack_webhook, "notify_on": self.notify_on,
  },
}

```

_build_batch_from_item (self, item)

Internal helper method to take an asset to validate, which can be either:

- (1) a DataAsset; or
- (2) a tuple of data_asset_name, expectation_suite_name, and batch_kwargs (suitable for passing to get_batch)

Parameters *item* – The item to convert to a batch (see above)

Returns A batch of data

```

run (self, assets_to_validate, run_id=None, evaluation_parameters=None, run_name=None,
     run_time=None, result_format=None)

```

`_run_actions` (*self*, *batch*, *expectation_suite_identifier*, *expectation_suite*, *batch_validation_result*, *run_id*)

Runs all actions configured for this operator on the result of validating one batch against one expectation suite.

If an action fails with an exception, the method does not continue.

Parameters

- **`batch`** –
- **`expectation_suite`** –
- **`batch_validation_result`** –
- **`run_id`** –

Returns a dictionary: {action name -> result returned by the action}

class `great_expectations.validation_operators.validation_operators.WarningAndFailureExpectationSuitesValidationOperator`

Bases: `great_expectations.validation_operators.validation_operators.ActionListValidationOperator`

WarningAndFailureExpectationSuitesValidationOperator is a validation operator that accepts a list batches of data assets (or the information necessary to fetch these batches). The operator retrieves 2 expectation suites for each data asset/batch - one containing the critical expectations (“failure”) and the other containing non-critical expectations (“warning”). By default, the operator assumes that the first is called “failure” and the second is called “warning”, but “base_expectation_suite_name” attribute can be specified in the operator’s configuration to make sure it searched for “{base_expectation_suite_name}.failure” and {base_expectation_suite_name}.warning” expectation suites for each data asset.

The operator validates each batch against its “failure” and “warning” expectation suites and invokes a list of actions on every validation result.

The list of these actions is specified in the operator’s configuration

Each action in the list must be an instance of ValidationAction class (or its descendants).

The operator sends a Slack notification (if “slack_webhook” is present in its config). The “notify_on” config property controls whether the notification should be sent only in the case of failure (“failure”), only in the case of success (“success”), or always (“all”).

Configuration

Below is an example of this operator’s configuration:

```

run_warning_and_failure_expectation_suites:
  class_name: WarningAndFailureExpectationSuitesValidationOperator

  # the following two properties are optional - by default the operator looks_
  ↪for
  # expectation suites named "failure" and "warning".
  # You can use these two properties to override these names.
  # e.g., with expectation_suite_name_prefix=boo_ and
  # expectation_suite_name_suffixes = ["red", "green"], the operator
  # will look for expectation suites named "boo_red" and "boo_green"
  expectation_suite_name_prefix="",
  expectation_suite_name_suffixes=["failure", "warning"],

  # optional - if true, the operator will stop and exit after first failed_
  ↪validation. false by default.
  stop_on_first_error=False,

  # put the actual webhook URL in the uncommitted/config_variables.yml file
  slack_webhook: ${validation_notification_slack_webhook}
  # optional - if "all" - notify always, "success" - notify only on success,
  ↪"failure" - notify only on failure
  notify_on="all"

  # the operator will call the following actions on each validation result
  # you can remove or add actions to this list. See the details in the actions
  # reference
  action_list:
    - name: store_validation_result
      action:
        class_name: StoreValidationResultAction
        target_store_name: validations_store
    - name: store_evaluation_params
      action:
        class_name: StoreEvaluationParametersAction
        target_store_name: evaluation_parameter_store

```

Invocation

This is an example of invoking an instance of a Validation Operator from Python:

```

results = context.run_validation_operator(
    assets_to_validate=[batch0, batch1, ...],
    run_id=RunIdentifier(**{
        "run_name": "some_string_that_uniquely_identifies_this_run",
        "run_time": "2020-04-29T10:46:03.197008" # optional run timestamp,
    ↪defaults to current UTC datetime
    })), # you may also pass in a dictionary with run_name and run_time keys
    validation_operator_name="operator_instance_name",
)

```

- *assets_to_validate* - an iterable that specifies the data assets that the operator will validate. The members of the list can be either batches or triples that will allow the operator to fetch the batch: (data_asset_name, expectation_suite_name, batch_kwargs) using this method: `get_batch()`
- *run_id* - pipeline run id of type `RunIdentifier`, consisting of a *run_time* (always assumed to be UTC time) and *run_name* string that is meaningful to you and will help you refer to the result of this operation later
- *validation_operator_name* you can instances of a class that implements a Validation Operator

The `run` method returns a `ValidationOperatorResult` object.

The value of “success” is `True` if no critical expectation suites (“failure”) failed to validate (non-critical warning”) expectation suites are allowed to fail without affecting the success status of the run.

```
{
  "run_id": {"run_time": "20200527T041833.074212Z", "run_name": "my_run_name"},
  "success": True,
  "evaluation_parameters": None,
  "validation_operator_config": {
    "class_name": "WarningAndFailureExpectationSuitesValidationOperator",
    "module_name": "great_expectations.validation_operators",
    "name": "warning_and_failure_operator",
    "kwargs": {
      "action_list": [
        {
          "name": "store_validation_result",
          "action": {"class_name": "StoreValidationResultAction"},
        },
        {
          "name": "store_evaluation_params",
          "action": {"class_name": "StoreEvaluationParametersAction"},
        },
        {
          "name": "update_data_docs",
          "action": {"class_name": "UpdateDataDocsAction"},
        },
      ],
      "base_expectation_suite_name": ...,
      "expectation_suite_name_suffixes": ...,
      "stop_on_first_error": ...,
      "slack_webhook": ...,
      "notify_on": ...,
    },
  },
  "run_results": {
    ValidationResultIdentifier: {
      "validation_result": ExpectationSuiteValidationResult object,
      "expectation_suite_severity_level": "warning",
      "actions_results": {
        "store_validation_result": {},
        "store_evaluation_params": {},
        "update_data_docs": {},
      },
    },
  }
}
```

property `validation_operator_config`(*self*)

This method builds the config dict of a particular validation operator. The “kwargs” key is what really distinguishes different validation operators.

e.g.: {

```
    "class_name": "ActionListValidationOperator",          "module_name":
    "great_expectations.validation_operators", "name": self.name, "kwargs": {
        "action_list": self.action_list
    },
}
```

```
    }
    { "class_name":      "WarningAndFailureExpectationSuitesValidationOperator",    "module_name":
      "great_expectations.validation_operators", "name": self.name, "kwargs": {
        "action_list":      self.action_list,      "base_expectation_suite_name":
        self.base_expectation_suite_name,      "expectation_suite_name_suffixes":
        self.expectation_suite_name_suffixes,    "stop_on_first_error":    self.stop_on_first_error,
        "slack_webhook": self.slack_webhook, "notify_on": self.notify_on,
      },
    }
  }

  _build_slack_query (self, validation_operator_result: ValidationOperatorResult)

  run (self,    assets_to_validate,    run_id=None,    base_expectation_suite_name=None,    evalua-
    tion_parameters=None, run_name=None, run_time=None, result_format=None)
```

Package Contents

Classes

<i>NoOpAction</i> (data_context)	This is the base class for all actions that act on validation results
<i>SlackNotificationAction</i> (data_context, renderer, slack_webhook, notify_on='all')	SlackNotificationAction sends a Slack notification to a given webhook.
<i>StoreEvaluationParametersAction</i> (data_context, target_store_name=None)	StoreEvaluationParametersAction extracts evaluation parameters from a validation result and stores them in the store
<i>StoreMetricsAction</i> (data_context, requested_metrics, target_store_name='metrics_store')	StoreMetricsAction extracts metrics from a Validation Result and stores them
<i>StoreValidationResultAction</i> (data_context, target_store_name=None)	StoreValidationResultAction stores a validation result in the ValidationsStore.
<i>UpdateDataDocsAction</i> (data_context, site_names=None, target_site_names=None)	UpdateDataDocsAction is a validation action that
<i>ValidationAction</i> (data_context)	This is the base class for all actions that act on validation results
<i>ActionListValidationOperator</i> (data_context, action_list, name, result_format={'result_format': 'SUMMARY'})	ActionListValidationOperator validates each batch in its run method's <code>assets_to_validate</code> argument against the Expectation Suite included within that batch.
<i>ValidationOperator</i> ()	The base class of all validation operators.
<i>WarningAndFailureExpectationSuitesValidationOperator</i> (data_context, action_list, name, base_expectation_suite_name=None, expectation_suite_name_suffixes=None, stop_on_first_error=False, slack_webhook=None, notify_on='all', result_format={'result_format': 'SUMMARY'})	<i>WarningAndFailureExpectationSuitesValidationOperator</i> is a validation operator

Functions

```
send_slack_notification(query, slack_webhook)
```

class great_expectations.validation_operators.NoOpAction(*data_context*)

Bases: *great_expectations.validation_operators.actions.ValidationAction*

This is the base class for all actions that act on validation results and are aware of a Data Context namespace structure.

The Data Context is passed to this class in its constructor.

_run (*self*, *validation_result_suite*, *validation_result_suite_identifier*, *data_asset*)

class great_expectations.validation_operators.SlackNotificationAction(*data_context*,
renderer,
renderer,
slack_webhook,
notify_on=*'all'*)

Bases: *great_expectations.validation_operators.actions.ValidationAction*

SlackNotificationAction sends a Slack notification to a given webhook.

Configuration

```
- name: send_slack_notification_on_validation_result
action:
  class_name: StoreValidationResultAction
  # put the actual webhook URL in the uncommitted/config_variables.yml file
  slack_webhook: ${validation_notification_slack_webhook}
  notify_on: all # possible values: "all", "failure", "success"
  renderer:
    # the class that implements the message to be sent
    # this is the default implementation, but you can
    # implement a custom one
    module_name: great_expectations.render.renderer.slack_renderer
    class_name: SlackRenderer
```

_run (*self*, *validation_result_suite*, *validation_result_suite_identifier*, *data_asset*=None)

class great_expectations.validation_operators.StoreEvaluationParametersAction(*data_context*,
target_store_name)

Bases: *great_expectations.validation_operators.actions.ValidationAction*

StoreEvaluationParametersAction extracts evaluation parameters from a validation result and stores them in the store configured for this action.

Evaluation parameters allow expectations to refer to statistics/metrics computed in the process of validating other prior expectations.

Configuration

```
- name: store_evaluation_params
action:
  class_name: StoreEvaluationParametersAction
  # name of the store where the action will store the parameters
```

(continues on next page)

(continued from previous page)

```
# the name must refer to a store that is configured in the great_expectations.
→yaml file
target_store_name: evaluation_parameter_store
```

```
_run (self, validation_result_suite, validation_result_suite_identifier, data_asset)
```

```
class great_expectations.validation_operators.StoreMetricsAction (data_context,
                                                                    re-
                                                                    requested_metrics,
                                                                    tar-
                                                                    get_store_name='metrics_store')
```

Bases: `great_expectations.validation_operators.actions.ValidationAction`

StoreMetricsAction extracts metrics from a Validation Result and stores them in a metrics store.

Configuration

```
- name: store_evaluation_params
action:
  class_name: StoreMetricsAction
  # name of the store where the action will store the metrics
  # the name must refer to a store that is configured in the great_expectations.
→yaml file
  target_store_name: my_metrics_store
```

```
_run (self, validation_result_suite, validation_result_suite_identifier, data_asset)
```

```
class great_expectations.validation_operators.StoreValidationResultAction (data_context,
                                                                              tar-
                                                                              get_store_name=None)
```

Bases: `great_expectations.validation_operators.actions.ValidationAction`

StoreValidationResultAction stores a validation result in the ValidationsStore.

Configuration

```
- name: store_validation_result
action:
  class_name: StoreValidationResultAction
  # name of the store where the actions will store validation results
  # the name must refer to a store that is configured in the great_expectations.
→yaml file
  target_store_name: validations_store
```

```
_run (self, validation_result_suite, validation_result_suite_identifier, data_asset)
```

```
class great_expectations.validation_operators.UpdateDataDocsAction (data_context,
                                                                      site_names=None,
                                                                      tar-
                                                                      get_site_names=None)
```

Bases: `great_expectations.validation_operators.actions.ValidationAction`

UpdateDataDocsAction is a validation action that notifies the site builders of all the data docs sites of the Data Context that a validation result should be added to the data docs.

Configuration

```
- name: update_data_docs
action:
  class_name: UpdateDataDocsAction
```

You can also instruct `UpdateDataDocsAction` to build only certain sites by providing a `site_names` key with a list of sites to update:

- `name: update_data_docs`

action: `class_name: UpdateDataDocsAction site_names:`

- `production_site`

`_run (self, validation_result_suite, validation_result_suite_identifier, data_asset)`

class `great_expectations.validation_operators.ValidationAction (data_context)`
Bases: `object`

This is the base class for all actions that act on validation results and are aware of a Data Context namespace structure.

The Data Context is passed to this class in its constructor.

run (`self, validation_result_suite, validation_result_suite_identifier, data_asset, **kwargs`)

Parameters

- **validation_result_suite** –
- **validation_result_suite_identifier** –
- **data_asset** –

Param `kwargs` - any additional arguments the child might use

Returns

`_run (self, validation_result_suite, validation_result_suite_identifier, data_asset)`

`great_expectations.validation_operators.logger`

`great_expectations.validation_operators.send_slack_notification (query, slack_webhook)`

class `great_expectations.validation_operators.ActionListValidationOperator (data_context, action_list, name, result_format='result_format', result_format='SUMMARY')`

Bases: `great_expectations.validation_operators.validation_operators.ValidationOperator`

`ActionListValidationOperator` validates each batch in its `run` method's `assets_to_validate` argument against the Expectation Suite included within that batch.

Then it invokes a list of configured actions on every validation result.

Each action in the list must be an instance of `ValidationAction` class (or its descendants). See the actions included in Great Expectations and how to configure them [here](#). You can also implement your own actions by extending the base class.

The `init` command includes this operator in the default configuration file.

Configuration

An instance of `ActionListValidationOperator` is included in the default configuration file `great_expectations.yml` that `great_expectations init` command creates.

```
perform_action_list_operator: # this is the name you will use when you invoke
↳the operator
  class_name: ActionListValidationOperator

  # the operator will call the following actions on each validation result
  # you can remove or add actions to this list. See the details in the actions
  # reference
  action_list:
    - name: store_validation_result
      action:
        class_name: StoreValidationResultAction
        target_store_name: validations_store
    - name: send_slack_notification_on_validation_result
      action:
        class_name: SlackNotificationAction
        # put the actual webhook URL in the uncommitted/config_variables.yml file
        slack_webhook: ${validation_notification_slack_webhook}
        notify_on: all # possible values: "all", "failure", "success"
        renderer:
          module_name: great_expectations.render.renderer.slack_renderer
          class_name: SlackRenderer
    - name: update_data_docs
      action:
        class_name: UpdateDataDocsAction
```

Invocation

This is an example of invoking an instance of a Validation Operator from Python:

```
results = context.run_validation_operator(
    assets_to_validate=[batch0, batch1, ...],
    run_id=RunIdentifier(**{
        "run_name": "some_string_that_uniquely_identifies_this_run",
        "run_time": "2020-04-29T10:46:03.197008" # optional run timestamp,
↳defaults to current UTC datetime
    })), # you may also pass in a dictionary with run_name and run_time keys
    validation_operator_name="operator_instance_name",
)
```

- `assets_to_validate` - an iterable that specifies the data assets that the operator will validate. The members of the list can be either batches or triples that will allow the operator to fetch the batch: (`data_asset_name`, `expectation_suite_name`, `batch_kwargs`) using this method: `get_batch()`
- `run_id` - pipeline run id of type `RunIdentifier`, consisting of a `run_time` (always assumed to be UTC time) and `run_name` string that is meaningful to you and will help you refer to the result of this operation later
- `validation_operator_name` you can instances of a class that implements a Validation Operator

The run method returns a `ValidationOperatorResult` object:

```
{
    "run_id": {"run_time": "20200527T041833.074212Z", "run_name": "my_run_name"},
    "success": True,
    "evaluation_parameters": None,
```

(continues on next page)

(continued from previous page)

```

"validation_operator_config": {
    "class_name": "ActionListValidationOperator",
    "module_name": "great_expectations.validation_operators",
    "name": "action_list_operator",
    "kwargs": {
        "action_list": [
            {
                "name": "store_validation_result",
                "action": {"class_name": "StoreValidationResultAction"},
            },
            {
                "name": "store_evaluation_params",
                "action": {"class_name": "StoreEvaluationParametersAction"},
            },
            {
                "name": "update_data_docs",
                "action": {"class_name": "UpdateDataDocsAction"},
            },
        ],
    },
},
"run_results": {
    ValidationResultIdentifier: {
        "validation_result": ExpectationSuiteValidationResult object,
        "actions_results": {
            "store_validation_result": {},
            "store_evaluation_params": {},
            "update_data_docs": {},
        },
    },
},
}

```

property validation_operator_config (self)

This method builds the config dict of a particular validation operator. The “kwargs” key is what really distinguishes different validation operators.

e.g.: {

```

    "class_name": "ActionListValidationOperator",    "module_name":
    "great_expectations.validation_operators", "name": self.name, "kwargs": {
        "action_list": self.action_list
    },

```

}

```

{ "class_name": "WarningAndFailureExpectationSuitesValidationOperator",    "module_name":
  "great_expectations.validation_operators", "name": self.name, "kwargs": {
    "action_list": self.action_list,    "base_expectation_suite_name":
    self.base_expectation_suite_name,    "expectation_suite_name_suffixes":
    self.expectation_suite_name_suffixes,    "stop_on_first_error": self.stop_on_first_error,
    "slack_webhook": self.slack_webhook, "notify_on": self.notify_on,
  },
}

```

`_build_batch_from_item` (*self*, *item*)

Internal helper method to take an asset to validate, which can be either:

- (1) a DataAsset; or
- (2) a tuple of `data_asset_name`, `expectation_suite_name`, and `batch_kwargs` (suitable for passing to `get_batch`)

Parameters *item* – The item to convert to a batch (see above)

Returns A batch of data

`run` (*self*, *assets_to_validate*, *run_id=None*, *evaluation_parameters=None*, *run_name=None*, *run_time=None*, *result_format=None*)

`_run_actions` (*self*, *batch*, *expectation_suite_identifier*, *expectation_suite*, *batch_validation_result*, *run_id*)

Runs all actions configured for this operator on the result of validating one batch against one expectation suite.

If an action fails with an exception, the method does not continue.

Parameters

- **`batch`** –
- **`expectation_suite`** –
- **`batch_validation_result`** –
- **`run_id`** –

Returns a dictionary: {action name -> result returned by the action}

`class` `great_expectations.validation_operators.ValidationOperator`

Bases: `object`

The base class of all validation operators.

It defines the signature of the public `run` method. This method and the `validation_operator_config` property are the only contract re operators' API. Everything else is up to the implementors of validation operator classes that will be the descendants of this base class.

`property validation_operator_config` (*self*)

This method builds the config dict of a particular validation operator. The “kwargs” key is what really distinguishes different validation operators.

e.g.: {

```
    “class_name”:          “ActionListValidationOperator”,      “module_name”:
    “great_expectations.validation_operators”, “name”: self.name, “kwargs”: {
        “action_list”: self.action_list
    },
```

}

```
{ “class_name”:          “WarningAndFailureExpectationSuitesValidationOperator”,      “module_name”:
  “great_expectations.validation_operators”, “name”: self.name, “kwargs”: {
    “action_list”:          self.action_list,          “base_expectation_suite_name”:
    self.base_expectation_suite_name,          “expectation_suite_name_suffixes”:
    self.expectation_suite_name_suffixes, “stop_on_first_error”:  self.stop_on_first_error,
    “slack_webhook”: self.slack_webhook, “notify_on”: self.notify_on,
```



```

    },
}

abstract run (self, assets_to_validate, run_id=None, evaluation_parameters=None,
              run_name=None, run_time=None)

class great_expectations.validation_operators.WarningAndFailureExpectationSuitesValidationOperator

```

Bases: `great_expectations.validation_operators.validation_operators.ActionListValidationOperator`

WarningAndFailureExpectationSuitesValidationOperator is a validation operator that accepts a list batches of data assets (or the information necessary to fetch these batches). The operator retrieves 2 expectation suites for each data asset/batch - one containing the critical expectations (“failure”) and the other containing non-critical expectations (“warning”). By default, the operator assumes that the first is called “failure” and the second is called “warning”, but “base_expectation_suite_name” attribute can be specified in the operator’s configuration to make sure it searched for “{base_expectation_suite_name}.failure” and {base_expectation_suite_name}.warning” expectation suites for each data asset.

The operator validates each batch against its “failure” and “warning” expectation suites and invokes a list of actions on every validation result.

The list of these actions is specified in the operator’s configuration

Each action in the list must be an instance of ValidationAction class (or its descendants).

The operator sends a Slack notification (if “slack_webhook” is present in its config). The “notify_on” config property controls whether the notification should be sent only in the case of failure (“failure”), only in the case of success (“success”), or always (“all”).

Configuration

Below is an example of this operator’s configuration:

```

run_warning_and_failure_expectation_suites:
  class_name: WarningAndFailureExpectationSuitesValidationOperator

  # the following two properties are optional - by default the operator looks_
  ↳for
  # expectation suites named "failure" and "warning".
  # You can use these two properties to override these names.
  # e.g., with expectation_suite_name_prefix=boo_ and
  # expectation_suite_name_suffixes = ["red", "green"], the operator
  # will look for expectation suites named "boo_red" and "boo_green"

```

(continues on next page)

(continued from previous page)

```

expectation_suite_name_prefix="",
expectation_suite_name_suffixes=["failure", "warning"],

# optional - if true, the operator will stop and exit after first failed_
↪validation. false by default.
stop_on_first_error=False,

# put the actual webhook URL in the uncommitted/config_variables.yml file
slack_webhook: ${validation_notification_slack_webhook}
# optional - if "all" - notify always, "success" - notify only on success,
↪"failure" - notify only on failure
notify_on="all"

# the operator will call the following actions on each validation result
# you can remove or add actions to this list. See the details in the actions
# reference
action_list:
  - name: store_validation_result
    action:
      class_name: StoreValidationResultAction
      target_store_name: validations_store
  - name: store_evaluation_params
    action:
      class_name: StoreEvaluationParametersAction
      target_store_name: evaluation_parameter_store

```

Invocation

This is an example of invoking an instance of a Validation Operator from Python:

```

results = context.run_validation_operator(
    assets_to_validate=[batch0, batch1, ...],
    run_id=RunIdentifier(**{
        "run_name": "some_string_that_uniquely_identifies_this_run",
        "run_time": "2020-04-29T10:46:03.197008" # optional run timestamp,
↪defaults to current UTC datetime
    })), # you may also pass in a dictionary with run_name and run_time keys
    validation_operator_name="operator_instance_name",
)

```

- *assets_to_validate* - an iterable that specifies the data assets that the operator will validate. The members of the list can be either batches or triples that will allow the operator to fetch the batch: (data_asset_name, expectation_suite_name, batch_kwargs) using this method: *get_batch()*
- *run_id* - pipeline run id of type *RunIdentifier*, consisting of a *run_time* (always assumed to be UTC time) and *run_name* string that is meaningful to you and will help you refer to the result of this operation later
- *validation_operator_name* you can instances of a class that implements a Validation Operator

The *run* method returns a *ValidationOperatorResult* object.

The value of “success” is True if no critical expectation suites (“failure”) failed to validate (non-critical warning”) expectation suites are allowed to fail without affecting the success status of the run.

```

{
  "run_id": {"run_time": "20200527T041833.074212Z", "run_name": "my_run_name"},
  "success": True,

```

(continues on next page)

(continued from previous page)

```

"evaluation_parameters": None,
"validation_operator_config": {
    "class_name": "WarningAndFailureExpectationSuitesValidationOperator",
    "module_name": "great_expectations.validation_operators",
    "name": "warning_and_failure_operator",
    "kwargs": {
        "action_list": [
            {
                "name": "store_validation_result",
                "action": {"class_name": "StoreValidationResultAction"},
            },
            {
                "name": "store_evaluation_params",
                "action": {"class_name": "StoreEvaluationParametersAction"},
            },
            {
                "name": "update_data_docs",
                "action": {"class_name": "UpdateDataDocsAction"},
            },
        ],
        "base_expectation_suite_name": ...,
        "expectation_suite_name_suffixes": ...,
        "stop_on_first_error": ...,
        "slack_webhook": ...,
        "notify_on": ...,
    },
},
"run_results": {
    ValidationResultIdentifier: {
        "validation_result": ExpectationSuiteValidationResult object,
        "expectation_suite_severity_level": "warning",
        "actions_results": {
            "store_validation_result": {},
            "store_evaluation_params": {},
            "update_data_docs": {},
        },
    },
}
}

```

property validation_operator_config (*self*)

This method builds the config dict of a particular validation operator. The “kwargs” key is what really distinguishes different validation operators.

e.g.: {

```

    "class_name": "ActionListValidationOperator",    "module_name":
    "great_expectations.validation_operators", "name": self.name, "kwargs": {
        "action_list": self.action_list
    },
}

{ "class_name": "WarningAndFailureExpectationSuitesValidationOperator",    "module_name":
  "great_expectations.validation_operators", "name": self.name, "kwargs": {
    "action_list": self.action_list,    "base_expectation_suite_name":

```

```
        self.base_expectation_suite_name,                "expectation_suite_name_suffixes":
        self.expectation_suite_name_suffixes,  "stop_on_first_error":    self.stop_on_first_error,
        "slack_webhook": self.slack_webhook, "notify_on": self.notify_on,
    },
}

def _build_slack_query (self, validation_operator_result: ValidationOperatorResult)
run (self,  assets_to_validate,  run_id=None,  base_expectation_suite_name=None,  evalua-
    tion_parameters=None, run_name=None, run_time=None, result_format=None)
```

`great_expectations.validator`

Submodules

`great_expectations.validator.validator`

This is currently helping bridge APIs

Module Contents

Classes

`Validator`(batch, expectation_suite, expecta-
tion_engine=None, **kwargs)

```
class great_expectations.validator.validator.Validator (batch,  expectation_suite,
                                                         expectation_engine=None,
                                                         **kwargs)

    Bases: object

    get_dataset (self)
```

Submodules

`great_expectations._version`

Git implementation of _version.py.

Module Contents

Classes

`VersioneerConfig`() Container for Versioneer configuration parameters.

Functions

<code>get_keywords()</code>	Get the keywords needed to look up the version information.
<code>get_config()</code>	Create, populate and return the VersioneerConfig() object.
<code>register_vcs_handler(vcs, method)</code>	Decorator to mark a method as the handler for a particular VCS.
<code>run_command(commands, args, cwd=None, verbose=False, hide_stderr=False, env=None)</code>	Call the given command(s).
<code>versions_from_parentdir(parentdir_prefix, root, verbose)</code>	Try to determine the version from the parent directory name.
<code>git_get_keywords(versionfile_abs)</code>	Extract version information from the given file.
<code>git_versions_from_keywords(keywords, tag_prefix, verbose)</code>	Get version information from git keywords.
<code>git_pieces_from_vcs(tag_prefix, root, verbose, run_command=run_command)</code>	Get version from 'git describe' in the root of the source tree.
<code>plus_or_dot(pieces)</code>	Return a + if we don't already have one, else return a .
<code>render_pep440(pieces)</code>	Build up version string, with post-release "local version identifier".
<code>render_pep440_pre(pieces)</code>	TAG[.post.devDISTANCE] – No -dirty.
<code>render_pep440_post(pieces)</code>	TAG[.postDISTANCE[.dev0]+gHEX] .
<code>render_pep440_old(pieces)</code>	TAG[.postDISTANCE[.dev0]] .
<code>render_git_describe(pieces)</code>	TAG[-DISTANCE-gHEX][-[dirty]].
<code>render_git_describe_long(pieces)</code>	TAG-DISTANCE-gHEX[-[dirty]].
<code>render(pieces, style)</code>	Render the given version pieces into the requested style.
<code>get_versions()</code>	Get version information or return default if unable to do so.

`great_expectations._version.get_keywords()`
Get the keywords needed to look up the version information.

class `great_expectations._version.VersioneerConfig`
Container for Versioneer configuration parameters.

`great_expectations._version.get_config()`
Create, populate and return the VersioneerConfig() object.

exception `great_expectations._version.NotThisMethod`
Bases: Exception

Exception raised if a method is not valid for the current scenario.

`great_expectations._version.LONG_VERSION_PY`

`great_expectations._version.HANDLERS`

`great_expectations._version.register_vcs_handler(vcs, method)`
Decorator to mark a method as the handler for a particular VCS.

`great_expectations._version.run_command(commands, args, cwd=None, verbose=False, hide_stderr=False, env=None)`
Call the given command(s).

`great_expectations._version.versions_from_parentdir(parentdir_prefix, root, verbose)`
Try to determine the version from the parent directory name.

Source tarballs conventionally unpack into a directory that includes both the project name and a version string. We will also support searching up two directory levels for an appropriately named parent directory

`great_expectations._version.git_get_keywords (versionfile_abs)`

Extract version information from the given file.

`great_expectations._version.git_versions_from_keywords (keywords, tag_prefix, verbose)`

Get version information from git keywords.

`great_expectations._version.git_pieces_from_vcs (tag_prefix, root, verbose, run_command=run_command)`

Get version from ‘git describe’ in the root of the source tree.

This only gets called if the git-archive ‘subst’ keywords were *not* expanded, and `_version.py` hasn’t already been rewritten with a short version string, meaning we’re inside a checked out source tree.

`great_expectations._version.plus_or_dot (pieces)`

Return a + if we don’t already have one, else return a .

`great_expectations._version.render_pep440 (pieces)`

Build up version string, with post-release “local version identifier”.

Our goal: TAG[+DISTANCE.gHEX[.dirty]] . Note that if you get a tagged build and then dirty it, you’ll get TAG+0.gHEX.dirty

Exceptions: 1: no tags. git_describe was just HEX. 0+untagged.DISTANCE.gHEX[.dirty]

`great_expectations._version.render_pep440_pre (pieces)`

TAG[.post.devDISTANCE] – No -dirty.

Exceptions: 1: no tags. 0.post.devDISTANCE

`great_expectations._version.render_pep440_post (pieces)`

TAG[.postDISTANCE[.dev0]+gHEX] .

The “.dev0” means dirty. Note that .dev0 sorts backwards (a dirty tree will appear “older” than the corresponding clean one), but you shouldn’t be releasing software with -dirty anyways.

Exceptions: 1: no tags. 0.postDISTANCE[.dev0]

`great_expectations._version.render_pep440_old (pieces)`

TAG[.postDISTANCE[.dev0]] .

The “.dev0” means dirty.

Exceptions: 1: no tags. 0.postDISTANCE[.dev0]

`great_expectations._version.render_git_describe (pieces)`

TAG[*-*DISTANCE-gHEX][*-*dirty].

Like ‘git describe –tags –dirty –always’.

Exceptions: 1: no tags. HEX[*-*dirty] (note: no ‘g’ prefix)

`great_expectations._version.render_git_describe_long (pieces)`

TAG-DISTANCE-gHEX[*-*dirty].

Like ‘git describe –tags –dirty –always -long’. The distance/hash is unconditional.

Exceptions: 1: no tags. HEX[*-*dirty] (note: no ‘g’ prefix)

`great_expectations._version.render (pieces, style)`

Render the given version pieces into the requested style.

`great_expectations._version.get_versions ()`

Get version information or return default if unable to do so.

great_expectations.exceptions

Module Contents

exception great_expectations.exceptions.**GreatExpectationsError** (*message*)

Bases: Exception

Common base class for all non-exit exceptions.

exception great_expectations.exceptions.**GreatExpectationsValidationError** (*message*,
validation_error)

Bases: `marshmallow.ValidationError`, `great_expectations.exceptions.GreatExpectationsError`

Raised when validation fails on a field or schema.

Validators and custom fields should raise this exception.

Parameters

- **message** – An error message, list of error messages, or dict of error messages. If a dict, the keys are subitems and the values are error messages.
- **field_name** – Field name to store the error on. If *None*, the error is stored as schema-level error.
- **data** – Raw input data.
- **valid_data** – Valid (de)serialized data.

exception great_expectations.exceptions.**DataContextError** (*message*)

Bases: `great_expectations.exceptions.GreatExpectationsError`

Common base class for all non-exit exceptions.

exception great_expectations.exceptions.**CheckpointError** (*message*)

Bases: `great_expectations.exceptions.DataContextError`

Common base class for all non-exit exceptions.

exception great_expectations.exceptions.**CheckpointNotFoundError** (*message*)

Bases: `great_expectations.exceptions.CheckpointError`

Common base class for all non-exit exceptions.

exception great_expectations.exceptions.**StoreBackendError** (*message*)

Bases: `great_expectations.exceptions.DataContextError`

Common base class for all non-exit exceptions.

exception great_expectations.exceptions.**UnavailableMetricError** (*message*)

Bases: `great_expectations.exceptions.GreatExpectationsError`

Common base class for all non-exit exceptions.

exception great_expectations.exceptions.**ParserError** (*message*)

Bases: `great_expectations.exceptions.GreatExpectationsError`

Common base class for all non-exit exceptions.

exception `great_expectations.exceptions.InvalidConfigurationYamlError` (*message*)
Bases: `great_expectations.exceptions.GreatExpectationsError`

Common base class for all non-exit exceptions.

exception `great_expectations.exceptions.InvalidTopLevelConfigKeyError` (*message*)
Bases: `great_expectations.exceptions.GreatExpectationsError`

Common base class for all non-exit exceptions.

exception `great_expectations.exceptions.MissingTopLevelConfigKeyError` (*message*,
validation_error)
Bases: `great_expectations.exceptions.GreatExpectationsValidationError`

Raised when validation fails on a field or schema.

Validators and custom fields should raise this exception.

Parameters

- **message** – An error message, list of error messages, or dict of error messages. If a dict, the keys are subitems and the values are error messages.
- **field_name** – Field name to store the error on. If *None*, the error is stored as schema-level error.
- **data** – Raw input data.
- **valid_data** – Valid (de)serialized data.

exception `great_expectations.exceptions.InvalidDataContextConfigError` (*message*,
validation_error)
Bases: `great_expectations.exceptions.GreatExpectationsValidationError`

Raised when validation fails on a field or schema.

Validators and custom fields should raise this exception.

Parameters

- **message** – An error message, list of error messages, or dict of error messages. If a dict, the keys are subitems and the values are error messages.
- **field_name** – Field name to store the error on. If *None*, the error is stored as schema-level error.
- **data** – Raw input data.
- **valid_data** – Valid (de)serialized data.

exception `great_expectations.exceptions.InvalidBatchKwargsError` (*message*)
Bases: `great_expectations.exceptions.GreatExpectationsError`

Common base class for all non-exit exceptions.

exception `great_expectations.exceptions.InvalidBatchIdError` (*message*)
Bases: `great_expectations.exceptions.GreatExpectationsError`

Common base class for all non-exit exceptions.

exception `great_expectations.exceptions.InvalidDataContextKeyError` (*message*)
Bases: `great_expectations.exceptions.DataContextError`

exception `great_expectations.exceptions.InvalidCacheValueError` (*result_dict*)
Bases: `great_expectations.exceptions.GreatExpectationsError`

Common base class for all non-exit exceptions.

exception `great_expectations.exceptions.ConfigNotFoundError`
Bases: `great_expectations.exceptions.DataContextError`

The great_expectations dir could not be found.

exception `great_expectations.exceptions.PluginModuleNotFoundError` (*module_name*)
Bases: `great_expectations.exceptions.GreatExpectationsError`

A module import failed.

exception `great_expectations.exceptions.PluginClassNotFoundError` (*module_name*,
class_name)
Bases: `great_expectations.exceptions.DataContextError`, `AttributeError`

A module import failed.

exception `great_expectations.exceptions.ClassInstantiationError` (*module_name*,
*pack-
age_name*,
class_name)
Bases: `great_expectations.exceptions.GreatExpectationsError`

Common base class for all non-exit exceptions.

exception `great_expectations.exceptions.ExpectationSuiteNotFoundError` (*data_asset_name*)
Bases: `great_expectations.exceptions.GreatExpectationsError`

Common base class for all non-exit exceptions.

exception `great_expectations.exceptions.BatchKwargsError` (*message*,
batch_kwargs=None)
Bases: `great_expectations.exceptions.DataContextError`

Common base class for all non-exit exceptions.

exception `great_expectations.exceptions.DatasourceInitializationError` (*datasource_name*,
*mes-
sage*)
Bases: `great_expectations.exceptions.GreatExpectationsError`

Common base class for all non-exit exceptions.

exception `great_expectations.exceptions.InvalidConfigValueTypeError` (*message*)
Bases: `great_expectations.exceptions.DataContextError`

Common base class for all non-exit exceptions.

`great_expectations.util`

Module Contents

Functions

`verify_dynamic_loading_support`(*module_name*,
package_name=None)

continues on next page

Table 135 – continued from previous page

<code>load_class(class_name, module_name)</code>	
<code>_convert_to_dataset_class(df, dataset_class, expectation_suite=None, profiler=None)</code>	Convert a (pandas) dataframe to a great_expectations dataset, with (optional) expectation_suite
<code>_load_and_convert_to_dataset_class(df, class_name, module_name, expectation_suite=None, profiler=None)</code>	Convert a (pandas) dataframe to a great_expectations dataset, with (optional) expectation_suite
<code>read_csv(filename, class_name='PandasDataset', module_name='great_expectations.dataset', dataset_class=None, expectation_suite=None, profiler=None, *args, **kwargs)</code>	Read a file using Pandas read_csv and return a great_expectations dataset.
<code>read_json(filename, class_name='PandasDataset', module_name='great_expectations.dataset', dataset_class=None, expectation_suite=None, accessor_func=None, profiler=None, *args, **kwargs)</code>	Read a file using Pandas read_json and return a great_expectations dataset.
<code>read_excel(filename, class_name='PandasDataset', module_name='great_expectations.dataset', dataset_class=None, expectation_suite=None, profiler=None, *args, **kwargs)</code>	Read a file using Pandas read_excel and return a great_expectations dataset.
<code>read_table(filename, class_name='PandasDataset', module_name='great_expectations.dataset', dataset_class=None, expectation_suite=None, profiler=None, *args, **kwargs)</code>	Read a file using Pandas read_table and return a great_expectations dataset.
<code>read_parquet(filename, class_name='PandasDataset', module_name='great_expectations.dataset', dataset_class=None, expectation_suite=None, profiler=None, *args, **kwargs)</code>	Read a file using Pandas read_parquet and return a great_expectations dataset.
<code>from_pandas(pandas_df, class_name='PandasDataset', module_name='great_expectations.dataset', dataset_class=None, expectation_suite=None, profiler=None)</code>	Read a Pandas data frame and return a great_expectations dataset.
<code>read_pickle(filename, class_name='PandasDataset', module_name='great_expectations.dataset', dataset_class=None, expectation_suite=None, profiler=None, *args, **kwargs)</code>	Read a file using Pandas read_pickle and return a great_expectations dataset.
<code>validate(data_asset, expectation_suite=None, data_asset_name=None, expectation_suite_name=None, data_context=None, data_asset_class_name=None, data_asset_module_name='great_expectations.dataset', data_asset_class=None, *args, **kwargs)</code>	Validate the provided data asset. Validate can accept an optional data_asset_name to apply, data_context to use
<code>gen_directory_tree_str(startpath)</code>	Print the structure of directory as a tree:
<code>lint_code(code)</code>	Lint strings of code passed in.

```
great_expectations.util.logger
```

```
great_expectations.util.verify_dynamic_loading_support(module_name, package_name=None)
```

```
great_expectations.util.load_class(class_name, module_name)
```

```
great_expectations.util._convert_to_dataset_class(df, dataset_class, expectation_suite=None, profiler=None)
```

Convert a (pandas) dataframe to a great_expectations dataset, with (optional) expectation_suite

Parameters

- **df** – the DataFrame object to convert
- **dataset_class** – the class to which to convert the existing DataFrame
- **expectation_suite** – the expectation suite that should be attached to the resulting dataset
- **profiler** – the profiler to use to generate baseline expectations, if any

Returns A new Dataset object

```
great_expectations.util._load_and_convert_to_dataset_class(df, class_name,
                                                           module_name, expectation_suite=None,
                                                           profiler=None)
```

Convert a (pandas) dataframe to a great_expectations dataset, with (optional) expectation_suite

Parameters

- **df** – the DataFrame object to convert
- **class_name** (*str*) – class to which to convert resulting Pandas df
- **module_name** (*str*) – dataset module from which to try to dynamically load the relevant module
- **expectation_suite** – the expectation suite that should be attached to the resulting dataset
- **profiler** – the profiler to use to generate baseline expectations, if any

Returns A new Dataset object

```
great_expectations.util.read_csv(filename, class_name='PandasDataset', module_name='great_expectations.dataset', dataset_class=None, expectation_suite=None, profiler=None, *args, **kwargs)
```

Read a file using Pandas read_csv and return a great_expectations dataset.

Parameters

- **filename** (*string*) – path to file to read
- **class_name** (*str*) – class to which to convert resulting Pandas df
- **module_name** (*str*) – dataset module from which to try to dynamically load the relevant module
- **dataset_class** (*Dataset*) – If specified, the class to which to convert the resulting Dataset object; if not specified, try to load the class named via the class_name and module_name parameters
- **expectation_suite** (*string*) – path to great_expectations expectation suite file
- **profiler** (*Profiler class*) – profiler to use when creating the dataset (default is None)

Returns great_expectations dataset

```
great_expectations.util.read_json(filename, class_name='PandasDataset', module_name='great_expectations.dataset', dataset_class=None, expectation_suite=None, accessor_func=None, profiler=None, *args, **kwargs)
```

Read a file using Pandas read_json and return a great_expectations dataset.

Parameters

- **filename** (*string*) – path to file to read
- **class_name** (*str*) – class to which to convert resulting Pandas df
- **module_name** (*str*) – dataset module from which to try to dynamically load the relevant module
- **dataset_class** (*Dataset*) – If specified, the class to which to convert the resulting Dataset object; if not specified, try to load the class named via the class_name and module_name parameters
- **expectation_suite** (*string*) – path to great_expectations expectation suite file
- **accessor_func** (*Callable*) – functions to transform the json object in the file
- **profiler** (*Profiler class*) – profiler to use when creating the dataset (default is None)

Returns great_expectations dataset

```
great_expectations.util.read_excel(filename, class_name='PandasDataset',
                                   module_name='great_expectations.dataset',
                                   dataset_class=None, expectation_suite=None, profiler=None, *args, **kwargs)
```

Read a file using Pandas read_excel and return a great_expectations dataset.

Parameters

- **filename** (*string*) – path to file to read
- **class_name** (*str*) – class to which to convert resulting Pandas df
- **module_name** (*str*) – dataset module from which to try to dynamically load the relevant module
- **dataset_class** (*Dataset*) – If specified, the class to which to convert the resulting Dataset object; if not specified, try to load the class named via the class_name and module_name parameters
- **expectation_suite** (*string*) – path to great_expectations expectation suite file
- **profiler** (*Profiler class*) – profiler to use when creating the dataset (default is None)

Returns great_expectations dataset or ordered dict of great_expectations datasets, if multiple worksheets are imported

```
great_expectations.util.read_table(filename, class_name='PandasDataset',
                                   module_name='great_expectations.dataset',
                                   dataset_class=None, expectation_suite=None, profiler=None, *args, **kwargs)
```

Read a file using Pandas read_table and return a great_expectations dataset.

Parameters

- **filename** (*string*) – path to file to read
- **class_name** (*str*) – class to which to convert resulting Pandas df
- **module_name** (*str*) – dataset module from which to try to dynamically load the relevant module
- **dataset_class** (*Dataset*) – If specified, the class to which to convert the resulting Dataset object; if not specified, try to load the class named via the class_name and module_name parameters

- **expectation_suite** (*string*) – path to great_expectations expectation suite file
- **profiler** (*Profiler class*) – profiler to use when creating the dataset (default is None)

Returns great_expectations dataset

```
great_expectations.util.read_parquet(filename, class_name='PandasDataset',
                                     module_name='great_expectations.dataset',
                                     dataset_class=None, expectation_suite=None, profiler=None, *args, **kwargs)
```

Read a file using Pandas read_parquet and return a great_expectations dataset.

Parameters

- **filename** (*string*) – path to file to read
- **class_name** (*str*) – class to which to convert resulting Pandas df
- **module_name** (*str*) – dataset module from which to try to dynamically load the relevant module
- **dataset_class** (*Dataset*) – If specified, the class to which to convert the resulting Dataset object; if not specified, try to load the class named via the class_name and module_name parameters
- **expectation_suite** (*string*) – path to great_expectations expectation suite file
- **profiler** (*Profiler class*) – profiler to use when creating the dataset (default is None)

Returns great_expectations dataset

```
great_expectations.util.from_pandas(pandas_df, class_name='PandasDataset',
                                    module_name='great_expectations.dataset',
                                    dataset_class=None, expectation_suite=None, profiler=None)
```

Read a Pandas data frame and return a great_expectations dataset.

Parameters

- **pandas_df** (*Pandas df*) – Pandas data frame
- **class_name** (*str*) – class to which to convert resulting Pandas df
- **module_name** (*str*) – dataset module from which to try to dynamically load the relevant module
- **dataset_class** (*Dataset*) – If specified, the class to which to convert the resulting Dataset object; if not specified, try to load the class named via the class_name and module_name parameters
- **expectation_suite** (*string*) – path to great_expectations expectation suite file
- **profiler** (*profiler class*) – The profiler that should be run on the dataset to establish a baseline expectation suite.

Returns great_expectations dataset

```
great_expectations.util.read_pickle(filename, class_name='PandasDataset',
                                    module_name='great_expectations.dataset',
                                    dataset_class=None, expectation_suite=None, profiler=None, *args, **kwargs)
```

Read a file using Pandas read_pickle and return a great_expectations dataset.

Parameters

- **filename** (*string*) – path to file to read
- **class_name** (*str*) – class to which to convert resulting Pandas df
- **module_name** (*str*) – dataset module from which to try to dynamically load the relevant module
- **dataset_class** (*Dataset*) – If specified, the class to which to convert the resulting Dataset object; if not specified, try to load the class named via the class_name and module_name parameters
- **expectation_suite** (*string*) – path to great_expectations expectation suite file
- **profiler** (*Profiler class*) – profiler to use when creating the dataset (default is None)

Returns great_expectations dataset

```
great_expectations.util.validate (data_asset, expectation_suite=None,
                                   data_asset_name=None, expectation_suite_name=None,
                                   data_context=None, data_asset_class_name=None,
                                   data_asset_module_name='great_expectations.dataset',
                                   data_asset_class=None, *args, **kwargs)
```

Validate the provided data asset. Validate can accept an optional data_asset_name to apply, data_context to use to fetch an expectation_suite if one is not provided, and data_asset_class_name/data_asset_module_name or data_asset_class to use to provide custom expectations.

Parameters

- **data_asset** – the asset to validate
- **expectation_suite** – the suite to use, or None to fetch one using a DataContext
- **data_asset_name** – the name of the data asset to use
- **expectation_suite_name** – the name of the expectation_suite to use
- **data_context** – data context to use to fetch an an expectation suite, or the path from which to obtain one
- **data_asset_class_name** – the name of a class to dynamically load a DataAsset class
- **data_asset_module_name** – the name of the module to dynamically load a DataAsset class
- **data_asset_class** – a class to use. overrides data_asset_class_name/data_asset_module_name if provided
- ***args** –
- ****kwargs** –

Returns:

```
great_expectations.util.gen_directory_tree_str (startpath)
```

Print the structure of directory as a tree:

Ex: project_dir0/

AAA/ BBB/

aaa.txt bbb.txt

#Note: files and directories are sorted alphabetically, so that this method can be used for testing.

`great_expectations.util.lint_code (code)`
Lint strings of code passed in.

Package Contents

Classes

<code>DataContext(context_root_dir=None, time_environment=None)</code>	run-	A DataContext represents a Great Expectations project. It organizes storage and access for
--	------	--

Functions

<code>get_versions()</code>	Get version information or return default if unable to do so.
-----------------------------	---

`great_expectations.get_versions()`
Get version information or return default if unable to do so.

`great_expectations.__version__`

class `great_expectations.DataContext` (*context_root_dir=None, runtime_environment=None*)
Bases: `great_expectations.data_context.data_context.BaseDataContext`

A DataContext represents a Great Expectations project. It organizes storage and access for expectation suites, datasources, notification settings, and data fixtures.

The DataContext is configured via a yml file stored in a directory called `great_expectations`; the configuration file as well as managed expectation suites should be stored in version control.

Use the *create* classmethod to create a new empty config, or instantiate the DataContext by passing the path to an existing data context root directory.

DataContexts use data sources you're already familiar with. BatchKwargGenerators help introspect data stores and data execution frameworks (such as airflow, Nifi, dbt, or dagster) to describe and produce batches of data ready for analysis. This enables fetching, validation, profiling, and documentation of your data in a way that is meaningful within your existing infrastructure and work environment.

DataContexts use a datasource-based namespace, where each accessible type of data has a three-part normalized *data_asset_name*, consisting of *datasource/generator/data_asset_name*.

- The datasource actually connects to a source of materialized data and returns Great Expectations DataAssets connected to a compute environment and ready for validation.
- The BatchKwargGenerator knows how to introspect datasources and produce identifying “batch_kwargs” that define particular slices of data.
- The *data_asset_name* is a specific name – often a table name or other name familiar to users – that batch kwargs generators can slice into batches.

An expectation suite is a collection of expectations ready to be applied to a batch of data. Since in many projects it is useful to have different expectations evaluate in different contexts—profiling vs. testing; warning vs. error; high vs. low compute; ML model or dashboard—suites provide a namespace option for selecting which expectations a DataContext returns.

In many simple projects, the datasource or batch kwargs generator name may be omitted and the DataContext will infer the correct name when there is no ambiguity.

Similarly, if no expectation suite name is provided, the DataContext will assume the name “default”.

classmethod create (*cls*, *project_root_dir=None*, *usage_statistics_enabled=True*, *runtime_environment=None*)

Build a new great_expectations directory and DataContext object in the provided *project_root_dir*.

create will not create a new “great_expectations” directory in the provided folder, provided one does not already exist. Then, it will initialize a new DataContext in that folder and write the resulting config.

Parameters

- **project_root_dir** – path to the root directory in which to create a new great_expectations directory
- **runtime_environment** – a dictionary of config variables that
- **both those set in config_variables.yml and the environment (override) –**

Returns DataContext

classmethod all_uncommitted_directories_exist (*cls*, *ge_dir*)

Check if all uncommitted directories exist.

classmethod config_variables_yaml_exist (*cls*, *ge_dir*)

Check if all config_variables.yml exists.

classmethod write_config_variables_template_to_disk (*cls*, *uncommitted_dir*)

classmethod write_project_template_to_disk (*cls*, *ge_dir*, *usage_statistics_enabled=True*)

classmethod scaffold_directories (*cls*, *base_dir*)

Safely create GE directories for a new project.

classmethod scaffold_custom_data_docs (*cls*, *plugins_dir*)

Copy custom data docs templates

classmethod scaffold_notebooks (*cls*, *base_dir*)

Copy template notebooks into the notebooks directory for a project.

_load_project_config (*self*)

Reads the project configuration from the project configuration file. The file may contain \${SOME_VARIABLE} variables - see *self._project_config_with_variables_substituted* for how these are substituted.

Returns the configuration object read from the file

list_checkpoints (*self*)

List checkpoints. (Experimental)

get_checkpoint (*self*, *checkpoint_name: str*)

Load a checkpoint. (Experimental)

_list_ymls_in_checkpoints_directory (*self*)

_save_project_config (*self*)

Save the current project to disk.

add_store (*self*, *store_name*, *store_config*)

Add a new Store to the DataContext and (for convenience) return the instantiated Store object.

Parameters

- **store_name** (*str*) – a key for the new Store in *self._stores*

- **store_config** (*dict*) – a config for the Store to add

Returns store (Store)

add_datasource (*self*, *name*, ***kwargs*)

Add a new datasource to the data context, with configuration provided as kwargs. :param name: the name for the new datasource to add :param initialize: if False, add the datasource to the config, but do not

initialize it, for example if a user needs to debug database connectivity.

Parameters **kwargs** (*keyword arguments*) – the configuration for the new datasource

Returns datasource (Datasource)

classmethod **find_context_root_dir** (*cls*)

classmethod **get_ge_config_version** (*cls*, *context_root_dir=None*)

classmethod **set_ge_config_version** (*cls*, *config_version*, *context_root_dir=None*, *validate_config_version=True*)

classmethod **find_context_yaml_file** (*cls*, *search_start_dir=None*)

Search for the yaml file starting here and moving upward.

classmethod **does_config_exist_on_disk** (*cls*, *context_root_dir*)

Return True if the great_expectations.yml exists on disk.

classmethod **is_project_initialized** (*cls*, *ge_dir*)

Return True if the project is initialized.

To be considered initialized, all of the following must be true: - all project directories exist (including uncommitted directories) - a valid great_expectations.yml is on disk - a config_variables.yml is on disk - the project has at least one datasource - the project has at least one suite

classmethod **does_project_have_a_datasource_in_config_file** (*cls*, *ge_dir*)

classmethod **_does_context_have_at_least_one_datasource** (*cls*, *ge_dir*)

classmethod **_does_context_have_at_least_one_suite** (*cls*, *ge_dir*)

classmethod **_attempt_context_instantiation** (*cls*, *ge_dir*)

static **_validate_checkpoint** (*checkpoint: dict*, *checkpoint_name: str*)

great_expectations.rtd_url_ge_version

PYTHON MODULE INDEX

g

great_expectations, 233	great_expectations.core.usage_statistics.anonymizers, 254
great_expectations._version, 648	great_expectations.core.usage_statistics.anonymizers, 255
great_expectations.cli, 233	great_expectations.core.usage_statistics.anonymizers, 255
great_expectations.cli.checkpoint, 235	great_expectations.core.usage_statistics.anonymizers, 255
great_expectations.cli.checkpoint_script_template, 237	great_expectations.core.usage_statistics.anonymizers, 255
great_expectations.cli.cli, 238	great_expectations.core.usage_statistics.anonymizers, 255
great_expectations.cli.cli_logging, 238	great_expectations.core.usage_statistics.anonymizers, 256
great_expectations.cli.cli_messages, 239	great_expectations.core.usage_statistics.anonymizers, 256
great_expectations.cli.datasource, 240	great_expectations.core.usage_statistics.anonymizers, 256
great_expectations.cli.docs, 244	great_expectations.core.usage_statistics.anonymizers, 256
great_expectations.cli.init, 245	great_expectations.core.usage_statistics.anonymizers, 257
great_expectations.cli.mark, 245	great_expectations.core.usage_statistics.anonymizers, 257
great_expectations.cli.project, 246	great_expectations.core.usage_statistics.anonymizers, 257
great_expectations.cli.store, 246	great_expectations.core.usage_statistics.anonymizers, 257
great_expectations.cli.suite, 247	great_expectations.core.usage_statistics.anonymizers, 257
great_expectations.cli.tap_template, 248	great_expectations.core.usage_statistics.schemas, 258
great_expectations.cli.toolkit, 248	great_expectations.core.usage_statistics.usage_statistics, 258
great_expectations.cli.upgrade_helpers, 233	great_expectations.core.util, 266
great_expectations.cli.upgrade_helpers.base_upgrade_helper, 233	great_expectations.data_asset, 283
great_expectations.cli.upgrade_helpers.upgrade_helper_vii, 234	great_expectations.data_asset.data_asset, 283
great_expectations.cli.util, 251	great_expectations.data_asset.file_data_asset, 289
great_expectations.cli.validation_operator, 252	great_expectations.data_asset.util, 294
great_expectations.core, 254	great_expectations.data_context, 305
great_expectations.core.batch, 260	great_expectations.data_context.data_context, 338
great_expectations.core.data_context_key, 261	great_expectations.data_context.store, 305
great_expectations.core.evaluation_parameters, 262	great_expectations.data_context.store.database_store, 305
great_expectations.core.id_dict, 263	great_expectations.data_context.store.expectations, 306
great_expectations.core.metric, 264	great_expectations.data_context.store.html_site_store, 306
great_expectations.core.urn, 266	
great_expectations.core.usage_statistics, 254	
great_expectations.core.usage_statistics.anonymizers, 254	
great_expectations.core.usage_statistics.anonymizers, 254	

[306](#)
[307](#)
[308](#)
[308](#)
[309](#)
[310](#)
[313](#)
[350](#)
[320](#)
[320](#)
[329](#)
[329](#)
[351](#)
[364](#)
[364](#)
[410](#)
[432](#)
[448](#)
[462](#)
[534](#)
[534](#)
[534](#)
[536](#)
[536](#)
[538](#)
[539](#)
[540](#)
[542](#)
[543](#)
[559](#)
[562](#)
[564](#)
[565](#)
[554](#)
[554](#)
[557](#)
[567](#)
[651](#)
[574](#)
[574](#)
[579](#)
[579](#)
[580](#)
[581](#)
[583](#)
[584](#)
[584](#)
[586](#)
[618](#)
[597](#)
[597](#)
[586](#)
[587](#)
[587](#)
[588](#)
[588](#)
[591](#)
[592](#)
[599](#)
[great_expectations.data_context.store.metadata_store](#)
[great_expectations.data_context.store.queries_store](#)
[great_expectations.data_context.store.storage_backend](#)
[great_expectations.data_context.store.storage_backend](#)
[great_expectations.data_context.store.tuple_store_backend](#)
[great_expectations.data_context.store.validations_store](#)
[great_expectations.data_context.templates](#)
[great_expectations.data_context.types](#)
[great_expectations.data_context.types.base](#)
[great_expectations.data_context.types.base](#)
[great_expectations.data_context.types.regional_identifiers](#)
[great_expectations.data_context.util](#)
[great_expectations.dataset](#)
[great_expectations.dataset.dataset](#)
[great_expectations.dataset.pandas_dataset](#)
[great_expectations.dataset.sparkdf_dataset](#)
[great_expectations.dataset.sqlalchemy_dataset](#)
[great_expectations.dataset.util](#)
[great_expectations.datasource](#)
[great_expectations.datasource.batch_kwargs](#)
[great_expectations.datasource.batch_kwargs](#)
[great_expectations.datasource.batch_kwargs](#)
[great_expectations.datasource.batch_kwargs](#)
[great_expectations.datasource.batch_kwargs](#)
[great_expectations.datasource.batch_kwargs](#)
[great_expectations.datasource.batch_kwargs](#)
[great_expectations.datasource.batch_kwargs](#)
[great_expectations.datasource.batch_kwargs](#)
[great_expectations.datasource.batch_kwargs](#)
[great_expectations.datasource.datasource](#)
[great_expectations.datasource.pandas_datasource](#)
[great_expectations.datasource.sparkdf_datasource](#)
[great_expectations.datasource.sqlalchemy_datasource](#)
[great_expectations.datasource.types](#)
[great_expectations.datasource.types.batch_kwargs](#)
[great_expectations.datasource.types.reader_methods](#)
[great_expectations.datasource.util](#)
[great_expectations.exceptions](#)
[great_expectations.jupyter_ux](#)
[great_expectations.jupyter_ux.expectation_explorer](#)
[great_expectations.profile](#)
[great_expectations.profile.base](#)
[great_expectations.profile.basic_dataset_profiler](#)
[great_expectations.profile.basic_suite_builder_profiler](#)
[great_expectations.profile.columns_exist](#)
[great_expectations.profile.metrics_utils](#)
[great_expectations.profile.multi_batch_validation_runner](#)
[great_expectations.render](#)
[great_expectations.render.exceptions](#)
[great_expectations.render.renderer](#)
[great_expectations.render.renderer.call_to_action_renderer](#)
[great_expectations.render.renderer.column_section_renderer](#)
[great_expectations.render.renderer.content_block](#)
[great_expectations.render.renderer.databricks_batch_kwargs_generator_block](#)
[great_expectations.render.renderer.dask_batch_kwargs_generator_block](#)
[great_expectations.render.renderer.global_reader_batch_kwargs_generator_block](#)
[great_expectations.render.renderer.minimal_batch_kwargs_generator_block](#)
[great_expectations.render.renderer.queries_batch_kwargs_generator_block](#)
[great_expectations.render.renderer.s3_batch_kwargs_generator_block](#)
[great_expectations.render.renderer.sdb_batch_kwargs_generator_block](#)
[great_expectations.render.renderer.table_batch_kwargs_generator_block](#)
[great_expectations.render.renderer.section_renderer](#)
[great_expectations.render.renderer.page_renderer](#)

599
great_expectations.render.renderer.renderer,
600
great_expectations.render.renderer.site_builder,
601
great_expectations.render.renderer.site_index_page_renderer,
604
great_expectations.render.renderer.slack_renderer,
605
great_expectations.render.renderer.suite_edit_notebook_renderer,
605
great_expectations.render.renderer.suite_scaffold_notebook_renderer,
606
great_expectations.render.types, 610
great_expectations.render.util, 618
great_expectations.render.view, 613
great_expectations.render.view.view, 613
great_expectations.types, 620
great_expectations.types.base, 620
great_expectations.types.configurations,
621
great_expectations.types.expectations,
623
great_expectations.util, 654
great_expectations.validation_operators,
623
great_expectations.validation_operators.actions,
628
great_expectations.validation_operators.types,
623
great_expectations.validation_operators.types.validation_operator_result,
623
great_expectations.validation_operators.util,
631
great_expectations.validation_operators.validation_operators,
631
great_expectations.validator, 648
great_expectations.validator.validator,
648

Symbols

<code>__deepcopy__()</code>	(<i>great_expectations.types.base.DotDict</i> method), 620	<code>__ne__()</code>	(<i>great_expectations.core.ExpectationConfiguration</i> method), 272
<code>__delattr__()</code>	(<i>great_expectations.types.base.DotDict</i> attribute), 620	<code>__ne__()</code>	(<i>great_expectations.core.ExpectationSuite</i> method), 274
<code>__delitem__()</code>	(<i>great_expectations.core.DictDot</i> method), 269	<code>__ne__()</code>	(<i>great_expectations.core.data_context_key.DataContextKey</i> method), 261
<code>__delitem__()</code>	(<i>great_expectations.types.DictDot</i> method), 623	<code>__repr__()</code>	(<i>great_expectations.core.DataContextKey</i> method), 268
<code>__dir__()</code>	(<i>great_expectations.types.base.DotDict</i> method), 620	<code>__repr__()</code>	(<i>great_expectations.core.ExpectationConfiguration</i> method), 272
<code>__eq__()</code>	(<i>great_expectations.core.DataContextKey</i> method), 268	<code>__repr__()</code>	(<i>great_expectations.core.ExpectationKwargs</i> method), 272
<code>__eq__()</code>	(<i>great_expectations.core.ExpectationConfiguration</i> method), 272	<code>__repr__()</code>	(<i>great_expectations.core.ExpectationSuite</i> method), 274
<code>__eq__()</code>	(<i>great_expectations.core.ExpectationSuite</i> method), 274	<code>__repr__()</code>	(<i>great_expectations.core.ExpectationSuiteValidationResult</i> method), 280
<code>__eq__()</code>	(<i>great_expectations.core.ExpectationSuiteValidationResult</i> method), 280	<code>__repr__()</code>	(<i>great_expectations.core.ExpectationValidationResult</i> method), 278
<code>__eq__()</code>	(<i>great_expectations.core.ExpectationValidationResult</i> method), 278	<code>__repr__()</code>	(<i>great_expectations.core.RunIdentifier</i> method), 270
<code>__eq__()</code>	(<i>great_expectations.core.data_context_key.DataContextKey</i> method), 261	<code>__repr__()</code>	(<i>great_expectations.core.data_context_key.DataContextKey</i> method), 261
<code>__eq__()</code>	(<i>great_expectations.render.types.RenderedContent</i> method), 611	<code>__repr__()</code>	(<i>great_expectations.data_context.types.resource_identifiers.ResourceIdentifiers</i> method), 330
<code>__finalize__()</code>	(<i>great_expectations.dataset.PandasDataset</i> method), 512	<code>__repr__()</code>	(<i>great_expectations.validation_operators.types.validation_operator.ValidationOperator</i> method), 625
<code>__finalize__()</code>	(<i>great_expectations.dataset.pandas_dataset.PandasDataset</i> method), 411	<code>__setattr__()</code>	(<i>great_expectations.types.base.DotDict</i> attribute), 620
<code>__get__()</code>	(<i>great_expectations.data_asset.util.DocInheritanceAdapter</i> method), 294	<code>__setitem__()</code>	(<i>great_expectations.core.DictDot</i> method), 269
<code>__getattr__()</code>	(<i>great_expectations.types.base.DotDict</i> method), 620	<code>__setitem__()</code>	(<i>great_expectations.types.DictDot</i> method), 623
<code>__getitem__()</code>	(<i>great_expectations.core.DictDot</i> method), 269	<code>__str__()</code>	(<i>great_expectations.core.ExpectationConfiguration</i> method), 272
<code>__getitem__()</code>	(<i>great_expectations.types.DictDot</i> method), 623	<code>__str__()</code>	(<i>great_expectations.core.ExpectationKwargs</i> method), 272
<code>__hash__()</code>	(<i>great_expectations.core.DataContextKey</i> method), 268	<code>__str__()</code>	(<i>great_expectations.core.ExpectationSuite</i> method), 274
<code>__hash__()</code>	(<i>great_expectations.core.data_context_key.DataContextKey</i> method), 261	<code>__str__()</code>	(<i>great_expectations.core.ExpectationSuiteValidationResult</i> method), 280
<code>__ne__()</code>	(<i>great_expectations.core.DataContextKey</i> method), 268	<code>__str__()</code>	(<i>great_expectations.core.ExpectationValidationResult</i> method), 278

<code>method</code>), 278	<code>method</code>), 552
<code>__str__()</code> (<code>great_expectations.core.RunIdentifier</code> <code>method</code>), 270	<code>_build_asset_iterator()</code> (<code>great_expectations.datasource.batch_kwarg_generator.s3_batch</code> <code>method</code>), 542
<code>__version__</code> (in module <code>great_expectations</code>), 660	<code>_build_batch_from_item()</code> (<code>great_expectations.validation_operators.ActionListValidationOp</code> <code>method</code>), 643
<code>_add_column_level_expectations()</code> (<code>great_expectations.render.renderer.suite_edit_notebook_renderer</code> <code>method</code>), 606	<code>_build_batch_from_item()</code> (<code>great_expectations.validation_operators.validation_operators.A</code> <code>method</code>), 634
<code>_add_pandas_datasource()</code> (in module <code>great_expectations.cli.datasource</code>), 242	<code>_build_batch_from_item()</code> (<code>great_expectations.validation_operators.validation_operators.A</code> <code>method</code>), 634
<code>_add_scaffold_cell()</code> (<code>great_expectations.render.renderer.suite_scaffold_notebook_renderer.SuiteScaffoldNotebookRenderer</code> <code>method</code>), 607	<code>_build_batch_from_item()</code> (<code>great_expectations.validation_operators.validation_operators.A</code> <code>method</code>), 634
<code>_add_scaffold_column_list()</code> (<code>great_expectations.render.renderer.suite_scaffold_notebook_renderer.SuiteScaffoldNotebookRenderer</code> <code>method</code>), 607	<code>_build_batch_from_item()</code> (<code>great_expectations.validation_operators.validation_operators.A</code> <code>method</code>), 634
<code>_add_spark_datasource()</code> (in module <code>great_expectations.cli.datasource</code>), 243	<code>_build_batch_from_item()</code> (<code>great_expectations.validation_operators.validation_operators.A</code> <code>method</code>), 634
<code>_add_sqlalchemy_datasource()</code> (in module <code>great_expectations.cli.datasource</code>), 242	<code>_build_batch_from_item()</code> (<code>great_expectations.validation_operators.validation_operators.A</code> <code>method</code>), 634
<code>_add_table_level_expectations()</code> (<code>great_expectations.render.renderer.suite_edit_notebook_renderer</code> <code>method</code>), 606	<code>_build_batch_from_item()</code> (<code>great_expectations.validation_operators.validation_operators.A</code> <code>method</code>), 634
<code>_anonymizers</code> (in module <code>great_expectations.core.usage_statistics.usage_statistics</code>), 259	<code>_build_batch_from_item()</code> (<code>great_expectations.validation_operators.validation_operators.A</code> <code>method</code>), 634
<code>_append_expectation()</code> (<code>great_expectations.data_asset.DataAsset</code> <code>method</code>), 296	<code>_build_batch_from_item()</code> (<code>great_expectations.validation_operators.validation_operators.A</code> <code>method</code>), 634
<code>_append_expectation()</code> (<code>great_expectations.data_asset.data_asset.DataAsset</code> <code>method</code>), 284	<code>_build_batch_from_item()</code> (<code>great_expectations.validation_operators.validation_operators.A</code> <code>method</code>), 634
<code>_apply_dateutil_parse()</code> (<code>great_expectations.dataset.sparkdf_dataset.SparkDFDataset</code> <code>static method</code>), 434	<code>_build_batch_from_item()</code> (<code>great_expectations.validation_operators.validation_operators.A</code> <code>method</code>), 634
<code>_apply_global_config_overrides()</code> (<code>great_expectations.data_context.BaseDataContext</code> <code>method</code>), 353	<code>_build_batch_from_item()</code> (<code>great_expectations.validation_operators.validation_operators.A</code> <code>method</code>), 634
<code>_apply_global_config_overrides()</code> (<code>great_expectations.data_context.data_context.BaseDataContext</code> <code>method</code>), 339	<code>_build_batch_from_item()</code> (<code>great_expectations.validation_operators.validation_operators.A</code> <code>method</code>), 634
<code>_attempt_context_instantiation()</code> (<code>great_expectations.DataContext</code> class <code>method</code>), 662	<code>_build_batch_from_item()</code> (<code>great_expectations.validation_operators.validation_operators.A</code> <code>method</code>), 634
<code>_attempt_context_instantiation()</code> (<code>great_expectations.data_context.DataContext</code> class <code>method</code>), 363	<code>_build_batch_from_item()</code> (<code>great_expectations.validation_operators.validation_operators.A</code> <code>method</code>), 634
<code>_attempt_context_instantiation()</code> (<code>great_expectations.data_context.data_context.DataContext</code> class <code>method</code>), 349	<code>_build_batch_from_item()</code> (<code>great_expectations.validation_operators.validation_operators.A</code> <code>method</code>), 634
<code>_batch_kwarg_type</code> (<code>great_expectations.datasource.batch_kwarg_generator.batch_kwarg_generator</code> attribute), 535	<code>_build_batch_from_item()</code> (<code>great_expectations.validation_operators.validation_operators.A</code> <code>method</code>), 634
<code>_build_asset_iterator()</code> (<code>great_expectations.datasource.batch_kwarg_generator.S3GlobalReaderBatchKwargGenerator</code> <code>method</code>), 552	<code>_build_batch_from_item()</code> (<code>great_expectations.validation_operators.validation_operators.A</code> <code>method</code>), 634

`method), 542`
`_build_batch_kwarg_from_path()`
`(great_expectations.datasource.batch_kwarg_generator.GlobReaderBatchKwargGenerator`
`method), 549`
`_build_batch_kwarg_from_path()`
`(great_expectations.datasource.batch_kwarg_generator.SubdirReaderBatchKwargGenerator`
`method), 553`
`_build_batch_kwarg_from_path()`
`(great_expectations.datasource.batch_kwarg_generator.GlobReaderBatchKwargGenerator`
`method), 538`
`_build_batch_kwarg_from_path()`
`(great_expectations.datasource.batch_kwarg_generator.SubdirReaderBatchKwargGenerator`
`method), 543`
`_build_batch_kwarg_generator()`
`(great_expectations.datasource.DataSource`
`method), 569`
`_build_batch_kwarg_generator()`
`(great_expectations.datasource.datasource.DataSource`
`method), 561`
`_build_batch_kwarg_path_iter()`
`(great_expectations.datasource.batch_kwarg_generator.GlobReaderBatchKwargGenerator`
`method), 549`
`_build_batch_kwarg_path_iter()`
`(great_expectations.datasource.batch_kwarg_generator.S3GlobReaderBatchKwargGenerator`
`method), 552`
`_build_batch_kwarg_path_iter()`
`(great_expectations.datasource.batch_kwarg_generator.SubdirReaderBatchKwargGenerator`
`method), 553`
`_build_batch_kwarg_path_iter()`
`(great_expectations.datasource.batch_kwarg_generator.GlobReaderBatchKwargGenerator`
`method), 538`
`_build_batch_kwarg_path_iter()`
`(great_expectations.datasource.batch_kwarg_generator.S3GlobReaderBatchKwargGenerator`
`method), 542`
`_build_batch_kwarg_path_iter()`
`(great_expectations.datasource.batch_kwarg_generator.SubdirReaderBatchKwargGenerator`
`method), 543`
`_build_column_description_metadata()`
`(great_expectations.profile.BasicSuiteBuilderProfiler`
`class method), 586`
`_build_column_description_metadata()`
`(great_expectations.profile.basic_suite_builder_profiler.BasicSuiteBuilderProfiler`
`class method), 583`
`_build_datasource_from_config()`
`(great_expectations.data_context.BaseDataContext`
`method), 357`
`_build_datasource_from_config()`
`(great_expectations.data_context.data_context.BaseDataContext`
`method), 343`
`_build_datasource_intro_string()` (in module `great_expectations.cli.datasource`), 242
`_build_generators()`
`(great_expectations.datasource.DataSource`
`method), 569`
`_build_generators()`
`(great_expectations.datasource.datasource.DataSource`
`method), 549`
`_build_intro_string()` (in module `great_expectations.cli.docs`), 245
`_build_notebook_renderers()`
`(great_expectations.render.renderers.suite_edit_notebook_renderers`
`class method), 605`
`_build_notebook_renderers()`
`(great_expectations.render.renderers.suite_edit_notebook_renderers`
`static method), 606`
`_build_slack_query()`
`(great_expectations.validation_operators.validation_operators.W`
`method), 638`
`_build_store()` (in module `great_expectations.data_context.BaseDataContext`), 353
`_build_store()` (in module `great_expectations.data_context.data_context.BaseD`), 343
`_build_table_column_expectations()`
`(great_expectations.profile.BasicSuiteBuilderProfiler`
`method), 586`
`_build_table_column_expectations()`
`(great_expectations.profile.basic_suite_builder_profiler.BasicSuiteBuilderProfiler`
`method), 583`
`_build_table_row_count_expectation()`
`(great_expectations.profile.BasicSuiteBuilderProfiler`
`method), 586`
`_build_table_row_count_expectation()`
`(great_expectations.profile.basic_suite_builder_profiler.BasicSuiteBuilderProfiler`
`method), 583`
`_calc_map_expectation_success()`
`(great_expectations.data_asset.DataAsset`
`method), 280`
`_calc_map_expectation_success()`
`(great_expectations.data_asset.data_asset.DataAsset`
`method), 288`
`_calc_validation_statistics()` (in module `great_expectations.data_asset.data_asset`), 289
`_calc_validation_statistics_opt_out()`
`(great_expectations.data_context.BaseDataContext`
`method), 353`
`_check_global_usage_statistics_opt_out()`
`(great_expectations.data_context.data_context.BaseDataContext`
`method), 339`
`_check_that_columns_exist()` (in module `great_expectations.profile.basic_suite_builder_profiler`), 583
`_check_that_expectations_are_available()`
`(in module great_expectations.profile.basic_suite_builder_profiler), 583`
`_close_worker()` (in module `great_expectations.core.usage_statistics.usage_stat`), 583

method), 259

`_collect_bigquery_credentials()` (in module `great_expectations.cli.datasource`), 242

`_collect_mysql_credentials()` (in module `great_expectations.cli.datasource`), 242

`_collect_postgres_credentials()` (in module `great_expectations.cli.datasource`), 242

`_collect_redshift_credentials()` (in module `great_expectations.cli.datasource`), 243

`_collect_snowflake_credentials()` (in module `great_expectations.cli.datasource`), 242

`_compile_evaluation_parameter_dependencies()` (`great_expectations.data_context.BaseDataContext` method), 358

`_compile_evaluation_parameter_dependencies()` (`great_expectations.data_context.data_context.BaseDataContext` method), 344

`_constructor()` (`great_expectations.dataset.PandasDataset` property), 512

`_constructor()` (`great_expectations.dataset.pandas_dataset.PandasDataset` property), 411

`_content_block_type` (`great_expectations.render.renderer.content_block.ExceptionListContentBlockRenderer` attribute), 593

`_content_block_type` (`great_expectations.render.renderer.content_block.ExpectationSuiteBulletListContentBlockRenderer` attribute), 593

`_content_block_type` (`great_expectations.render.renderer.content_block.ValidationResultsTableContentBlockRenderer` attribute), 593

`_content_block_type` (`great_expectations.render.renderer.content_block.bullet_list_content_block.ExpectationSuiteBulletListContentBlockRenderer` attribute), 587

`_content_block_type` (`great_expectations.render.renderer.content_block.exception_list_content_block.ExceptionListContentBlockRenderer` attribute), 588

`_content_block_type` (`great_expectations.render.renderer.content_block.validation_results_table_content_block.ValidationResultsTableContentBlockRenderer` attribute), 592

`_convert_filepath_to_key()` (`great_expectations.data_context.store.TupleStoreBackend` method), 320

`_convert_filepath_to_key()` (`great_expectations.data_context.store.tuple_store_backend.TupleStoreBackend` method), 311

`_convert_key()` (`great_expectations.data_context.store.query_store_backend.QueryStore` method), 308

`_convert_key_to_filepath()` (`great_expectations.data_context.store.TupleStoreBackend` method), 320

`_convert_key_to_filepath()` (`great_expectations.data_context.store.tuple_store_backend.TupleStoreBackend` method), 311

`_convert_to_dataset_class()` (in module `great_expectations.util`), 655

`_copy_and_clean_up_expectation()` (`great_expectations.core.ExpectationSuite` method), 275

`_copy_and_clean_up_expectation()` (`great_expectations.data_asset.DataAsset` method), 296

`_copy_and_clean_up_expectation()` (`great_expectations.data_asset.data_asset.DataAsset` method), 284

`_copy_and_clean_up_expectations_from_indexes()` (`great_expectations.core.ExpectationSuite` method), 275

`_copy_and_clean_up_expectations_from_indexes()` (`great_expectations.data_asset.DataAsset` method), 296

`_copy_and_clean_up_expectations_from_indexes()` (`great_expectations.data_asset.data_asset.DataAsset` method), 284

`_create_expectations_for_datetime_column()` (`great_expectations.profile.BasicSuiteBuilderProfiler` class method), 586

`_create_expectations_for_datetime_column()` (`great_expectations.profile.basic_suite_builder_profiler.BasicSuiteBuilderProfiler` class method), 583

`_create_expectations_for_datetime_column()` (`great_expectations.profile.BasicSuiteBuilderProfiler` class method), 586

`_create_expectations_for_integer_column()` (`great_expectations.profile.basic_suite_builder_profiler.BasicSuiteBuilderProfiler` class method), 583

`_create_expectations_for_integer_column()` (`great_expectations.profile.basic_suite_builder_profiler.BasicSuiteBuilderProfiler` class method), 583

`_create_expectations_for_string_column()` (`great_expectations.profile.basic_suite_builder_profiler.BasicSuiteBuilderProfiler` class method), 583

`_create_expectations_for_string_column()` (`great_expectations.profile.basic_suite_builder_profiler.BasicSuiteBuilderProfiler` class method), 583

`_create_expectations()` (`great_expectations.profile.BasicSuiteBuilderProfiler` class method), 586

`_create_non_nullity_expectations()` (`great_expectations.profile.basic_suite_builder_profiler.BasicSuiteBuilderProfiler` class method), 583

`_custom_blocks()` (`great_expectations.render.renderer.SlackRenderer` method), 609

`_custom_blocks()` (`great_expectations.render.renderer.slack_renderer.SlackRenderer` method), 605

`_data_asset_type` (`great_expectations.data_asset.DataAsset`

[attribute](#)), 295
[_data_asset_type](#) ([great_expectations.data_asset.FileDataAsset](#) [DataAsset](#) [_reader_options](#) [attribute](#)), 301
[_data_asset_type](#) ([great_expectations.data_asset.data_asset.DataAsset](#) [DataAsset](#) [_reader_options](#) [attribute](#)), 283
[_data_asset_type](#) ([great_expectations.data_asset.file_data_asset.FileDataAsset](#) [DataAsset](#) [_reader_options](#) [attribute](#)), 290
[_data_asset_type](#) ([great_expectations.dataset.Dataset](#) [demo_profile\(\)](#) ([great_expectations.profile.BasicSuiteBuilderProfile](#) [dataset](#) [_reader_options](#) [attribute](#)), 467
[_data_asset_type](#) ([great_expectations.dataset.dataset.Dataset](#) [demo_profile\(\)](#) ([great_expectations.profile.basic_suite_builder_profile](#) [dataset](#) [_reader_options](#) [attribute](#)), 365
[_deduplicate_evaluation_parameter_dependencies](#) ([in module great_expectations.core](#)), 272
[_default_content_block_styling](#) ([great_expectations.render.renderer.content_block.ExceptionListContentBlockRenderer](#) [attribute](#)), 593
[_default_content_block_styling](#) ([great_expectations.render.renderer.content_block.ValidationResultsTableContentBlockRenderer](#) [attribute](#)), 593
[_default_content_block_styling](#) ([great_expectations.render.renderer.content_block.content_block.ContentBlockRenderer](#) [attribute](#)), 587
[_default_content_block_styling](#) ([great_expectations.render.renderer.content_block.exception_list_content_block.ExceptionListContentBlockRenderer](#) [attribute](#)), 588
[_default_content_block_styling](#) ([great_expectations.render.renderer.content_block.validation_results_table_content_block.ValidationResultsTableContentBlockRenderer](#) [attribute](#)), 592
[_default_element_styling](#) ([great_expectations.render.renderer.content_block.ExceptionListContentBlockRenderer](#) [attribute](#)), 594
[_default_element_styling](#) ([great_expectations.render.renderer.content_block.ExceptionListContentBlockRenderer](#) [attribute](#)), 593
[_default_element_styling](#) ([great_expectations.render.renderer.content_block.ValidationResultsTableContentBlockRenderer](#) [attribute](#)), 593
[_default_element_styling](#) ([great_expectations.render.renderer.content_block.bullet_list_content_block.ExceptionListContentBlockRenderer](#) [attribute](#)), 587
[_default_element_styling](#) ([great_expectations.render.renderer.content_block.content_block.ContentBlockRenderer](#) [attribute](#)), 587
[_default_element_styling](#) ([great_expectations.render.renderer.content_block.exception_list_content_block.ExceptionListContentBlockRenderer](#) [attribute](#)), 588
[_default_element_styling](#) ([great_expectations.render.renderer.content_block.validation_results_table_content_block.ValidationResultsTableContentBlockRenderer](#) [attribute](#)), 592
[_default_header](#) ([great_expectations.render.renderer.content_block.ExceptionListContentBlockRenderer](#) [attribute](#)), 593
[_default_header](#) ([great_expectations.render.renderer.content_block.validation_results_table_content_block.ValidationResultsTableContentBlockRenderer](#) [attribute](#)), 587
[_default_header](#) ([great_expectations.render.renderer.content_block.bullet_list_content_block.ExceptionListContentBlockRenderer](#) [attribute](#)), 587

(*great_expectations.dataset.PandasDataset*
 method), 516

_expect_column_values_to_be_of_type__aggregate ((*great_expectations.render.renderer.SiteIndexPageRenderer*
 (*great_expectations.dataset.pandas_dataset.PandasDataset* class method), 609
 method), 414

_expect_column_values_to_be_of_type__map ((*great_expectations.render.renderer.site_index_page_renderer.SiteIndexPageRenderer*
 (*great_expectations.dataset.PandasDataset*
 class method), 604
 method), 516

_expect_column_values_to_be_of_type__map ((*great_expectations.render.renderer.SiteIndexPageRenderer*
 (*great_expectations.dataset.pandas_dataset.PandasDataset* class method), 609
 method), 414

_explicit_url (*great_expectations.data_context.types.base.AnnotatedExpectationSuiteConfigSchema*
 attribute), 324

_filter_citations ((*great_expectations.core.ExpectationSuite*
 static method), 274

_find_all_evrs_by_type ((*great_expectations.render.renderer.Renderer*
 class method), 600

_find_evr_by_type ((*great_expectations.render.renderer.Renderer*
 class method), 600

_find_next_datetime_column ((*great_expectations.profile.BasicSuiteBuilderProfiler*
 class method), 586

_find_next_datetime_column ((*great_expectations.profile.basic_suite_builder_profiler.BasicSuiteBuilderProfiler*
 class method), 583

_find_next_low_card_column ((*great_expectations.profile.BasicSuiteBuilderProfiler*
 class method), 586

_find_next_low_card_column ((*great_expectations.profile.basic_suite_builder_profiler.BasicSuiteBuilderProfiler*
 class method), 583

_find_next_numeric_column ((*great_expectations.profile.BasicSuiteBuilderProfiler*
 class method), 586

_find_next_numeric_column ((*great_expectations.profile.basic_suite_builder_profiler.BasicSuiteBuilderProfiler*
 class method), 583

_find_next_string_column ((*great_expectations.profile.BasicSuiteBuilderProfiler*
 class method), 586

_find_next_string_column ((*great_expectations.profile.basic_suite_builder_profiler.BasicSuiteBuilderProfiler*
 class method), 583

_fix_path_in_batch_kwargs ((*great_expectations.render.renderer.suite_edit_notebook_renderer.SuiteEditNotebookRenderer*
 static method), 606

_format_map_output ((*great_expectations.data_asset.DataAsset*
 method), 300

_format_map_output ((*great_expectations.data_asset.data_asset.DataAsset*
 method), 300

method), 288

_generate_expectation_suites_link_table ((*great_expectations.render.renderer.SiteIndexPageRenderer*
 class method), 604

_generate_expectation_suites_link_table ((*great_expectations.render.renderer.site_index_page_renderer.SiteIndexPageRenderer*
 class method), 604

_generate_profiling_results_link_table ((*great_expectations.render.renderer.SiteIndexPageRenderer*
 class method), 604

_generate_profiling_results_link_table ((*great_expectations.render.renderer.site_index_page_renderer.SiteIndexPageRenderer*
 class method), 604

_generate_upgrade_checklist ((*great_expectations.cli.upgrade_helpers.UpgradeHelperV11*
 method), 235

_generate_upgrade_checklist ((*great_expectations.cli.upgrade_helpers.upgrade_helper_v11.UpgradeHelperV11*
 method), 234

_generate_upgrade_report ((*great_expectations.cli.upgrade_helpers.UpgradeHelperV11*
 method), 235

_generate_upgrade_report ((*great_expectations.cli.upgrade_helpers.upgrade_helper_v11.UpgradeHelperV11*
 method), 234

_generate_validation_results_link_table ((*great_expectations.render.renderer.SiteIndexPageRenderer*
 class method), 609

_generate_validation_results_link_table ((*great_expectations.render.renderer.site_index_page_renderer.SiteIndexPageRenderer*
 class method), 605

_get ((*great_expectations.data_context.store.DatabaseStoreBackend*
 method), 317

_get ((*great_expectations.data_context.store.InMemoryStoreBackend*
 method), 317

_get ((*great_expectations.data_context.store.StoreBackend*
 method), 317

_get ((*great_expectations.data_context.store.TupleFilesystemStoreBackend*
 method), 319

_get ((*great_expectations.data_context.store.TupleGCSSStoreBackend*
 method), 319

_get ((*great_expectations.data_context.store.TupleS3StoreBackend*
 method), 319

_get ((*great_expectations.data_context.store.database_store_backend.DatabaseStoreBackend*
 method), 310

_get ((*great_expectations.data_context.store.store_backend.InMemoryStoreBackend*
 method), 310

_get ((*great_expectations.data_context.store.store_backend.StoreBackend*
 method), 310

_get ((*great_expectations.data_context.store.tuple_store_backend.TupleFilesystemStoreBackend*
 method), 311

_get ((*great_expectations.data_context.store.tuple_store_backend.TupleGCSSStoreBackend*
 method), 313

_get ((*great_expectations.data_context.store.tuple_store_backend.TupleS3StoreBackend*
 method), 313

method), 312
 _get_asset_options()
 (great_expectations.datasource.batch_kwarg_generator.S3GlobalReaderBatchKwargGenerator.content_block.content_block.
 method), 552
 _get_asset_options()
 (great_expectations.datasource.batch_kwarg_generator.s3(batch_kwarg_generator.S3GlobalReaderBatchKwargGenerator.
 method), 542
 _get_batch_kwarg_for_sqlalchemy_datasource_get_data_asset_config()
 (in module great_expectations.cli.datasource),
 244
 _get_batch_kwarg_from_generator_or_from_file_path_asset_config()
 (in module great_expectations.cli.datasource),
 243
 _get_call_to_action_buttons()
 (great_expectations.render.renderer.site_builder.DefaultSiteIndexBuilder.great_expectations.datasource.batch_kwarg_generator.manual.
 method), 604
 _get_column_cardinality()
 (great_expectations.profile.basic_dataset_profiler.BasicDatasetProfiler.get_profile_data_options.datasource.batch_kwarg_generator.GlobRea
 class method), 581
 _get_column_cardinality_with_caching()
 (great_expectations.profile.BasicSuiteBuilderProfiler (great_expectations.datasource.batch_kwarg_generator.glob_rea
 class method), 585
 _get_column_cardinality_with_caching()
 (great_expectations.profile.basic_suite_builder_profiler.BasicSuiteBuilderProfiler.great_expectations.cli.toolkit), 250
 class method), 582
 _get_column_list_from_evrs()
 (great_expectations.render.renderer.renderer.Renderer (great_expectations.dataset.sqlalchemy_dataset.SqlAlchemyData
 class method), 600
 _get_column_name()
 (great_expectations.render.renderer.column_section_renderer.ColumnSectionRenderer (great_expectations.dataset.sqlalchemy_dataset.SqlAlchemyData
 class method), 598
 _get_column_type()
 (great_expectations.profile.basic_dataset_profiler.BasicDatasetProfiler.get_profile_data_options), 587
 class method), 581
 _get_column_type_with_caching()
 (great_expectations.profile.BasicSuiteBuilderProfiler (great_expectations.render.renderer.renderer.Renderer
 class method), 585
 _get_column_type_with_caching()
 (great_expectations.profile.basic_suite_builder_profiler.BasicSuiteBuilderProfiler (great_expectations.render.renderer.suite_edit_notebook_renderer
 class method), 582
 _get_column_types()
 (great_expectations.render.renderer.ProfilingResultsOverviewSectionRenderer (great_expectations.render.renderer.site_index_page_renderer.Site
 class method), 608
 _get_column_types()
 (great_expectations.render.renderer.other_section_renderer.ProfilingResultsOverviewSectionRenderer (great_expectations.render.renderer.suite_edit_notebook_renderer
 class method), 599
 _get_content_block_fn()
 (great_expectations.render.renderer.content_block_validation_results_table_content_block_renderer (great_expectations.data_context.BaseDataContext
 class method), 593
 _get_content_block_fn()
 (great_expectations.render.renderer.content_block_validation_results_table_content_block_renderer (great_expectations.data_context.data_context.BaseDataContext
 class method), 587
 _get_content_block_fn()
 (great_expectations.render.renderer.content_block_validation_results_table_content_block_renderer (great_expectations.data_context.data_context.BaseDataContext
 method), 339
 (great_expectations.render.renderer.content_block_validation_results_table_content_block_renderer (great_expectations.data_context.data_context.BaseDataContext
 method), 339

`_get_tuple_filesystem_store_backend_run_time()` `key()` (`great_expectations.data_context.store.database_store_backend` (`great_expectations.cli.upgrade_helpers.UpgradeHelperV1` method), 306
`method`), 235 `_has_key()` (`great_expectations.data_context.store.store_backend.InMemoryStoreBackend` method), 317
`_get_tuple_filesystem_store_backend_run_time()` `method`), 310
`(great_expectations.cli.upgrade_helpers.upgrade_helper_v1.UpgradeHelperV1` (`great_expectations.data_context.store.store_backend.StoreBackend` method), 234 `method`), 310
`_get_tuple_gcs_store_backend_run_time()` `_has_key()` (`great_expectations.data_context.store.tuple_store_backend.TupleStoreBackend` (`great_expectations.cli.upgrade_helpers.UpgradeHelperV1` method), 312
`method`), 235 `_has_key()` (`great_expectations.data_context.store.tuple_store_backend.TupleStoreBackend` method), 313
`_get_tuple_gcs_store_backend_run_time()` `method`), 313
`(great_expectations.cli.upgrade_helpers.upgrade_helper_v1.UpgradeHelperV1` (`great_expectations.data_context.store.tuple_store_backend.TupleStoreBackend` method), 234 `method`), 312
`_get_tuple_s3_store_backend_run_time()` `_id_ignore_keys` (`great_expectations.core.IDDict` (`great_expectations.cli.upgrade_helpers.UpgradeHelperV1` attribute), 269
`method`), 235 `_id_ignore_keys` (`great_expectations.core.id_dict.IDDict` attribute), 264
`_get_tuple_s3_store_backend_run_time()` `attribute`), 264
`(great_expectations.cli.upgrade_helpers.upgrade_helper_v1.UpgradeHelperV1` (`great_expectations.data_context.BaseDataContext` method), 234 `method`), 353
`_get_unexpected_statement()` `_init_stores()` (`great_expectations.data_context.data_context.BaseDataContext` (`great_expectations.render.renderer.content_block.ValidationResultsTableContentBlockRenderer` class method), 593
`class method`), 593 `_initialize_expectations()` (`great_expectations.data_asset.DataAsset` (`great_expectations.render.renderer.content_block.validation_results_table_content_block.ValidationResultsTableContentBlockRenderer` class method), 592
`class method`), 592 `_initialize_expectations()` (`great_expectations.data_asset.data_asset.DataAsset` (`great_expectations.render.renderer.content_block.ValidationResultsTableContentBlockRenderer` class method), 593
`class method`), 593 `_initialize_usage_statistics()` (`great_expectations.data_context.BaseDataContext` (`great_expectations.render.renderer.content_block.validation_results_table_content_block.ValidationResultsTableContentBlockRenderer` class method), 592
`class method`), 592 `_initialize_usage_statistics()` (`great_expectations.data_context.data_context.BaseDataContext` (`great_expectations.datasource.batch_kwarg_generator.SubdirReaderBatchKwargGenerator` method), 553
`method`), 553 `_internal_names` (`great_expectations.dataset.PandasDataset` attribute), 512
`_get_valid_file_options()` `attribute`), 512
`(great_expectations.datasource.batch_kwarg_generator.subdir_readers.batch_kwarg_generator.SubdirReaderBatchKwargGenerator` method), 543 `attribute`), 411
`_group_and_order_expectations_by_column()` `internal_names_set` (`great_expectations.render.renderer.renderer.Renderer` (`great_expectations.dataset.PandasDataset` class method), 600
`class method`), 600 `attribute`), 512
`_group_evrs_by_column()` `_internal_names_set` (`great_expectations.render.renderer.renderer.Renderer` (`great_expectations.dataset.pandas_dataset.PandasDataset` class method), 600
`class method`), 600 `attribute`), 411
`_has_key()` (`great_expectations.data_context.store.DatabaseStoreBackend` (in module `great_expectations.profile.basic_suite_builder_profiler`), `method`), 315
`_has_key()` (`great_expectations.data_context.store.InMemoryStoreBackend` `method`), 317
`_has_key()` (`great_expectations.data_context.store.StoreBackend` attribute), 315
`method`), 318 `_key_class` (`great_expectations.data_context.store.ExpectationsStore` attribute), 315
`_has_key()` (`great_expectations.data_context.store.TupleFilesystemStoreBackend` `method`), 318
`method`), 318 `_key_class` (`great_expectations.data_context.store.MetricStore` attribute), 315
`_has_key()` (`great_expectations.data_context.store.TupleGCSSStoreBackend` `method`), 319
`method`), 319 `_key_class` (`great_expectations.data_context.store.Store` attribute), 315
`_has_key()` (`great_expectations.data_context.store.TupleS3StoreBackend` `method`), 319
`method`), 319 `_key_class` (`great_expectations.data_context.store.ValidationsStore` attribute), 315

attribute), 320
 _key_class(great_expectations.data_context.store.expectations_store.ExpectationsStore() (in module
 attribute), 306
 _key_class(great_expectations.data_context.store.html_site_store.HtmlSiteStore.from_site_config()
 attribute), 307
 _key_class(great_expectations.data_context.store.metric_store.MetricStore) 354
 attribute), 307
 _key_class(great_expectations.data_context.store.query_store.SqlAlchemyQueryStore) 354
 attribute), 308
 _key_class(great_expectations.data_context.store.store.Store) 354
 attribute), 309
 _key_class(great_expectations.data_context.store.validations_store.ValidationsStore) 354
 attribute), 314
 _list_data_assets_validated_by_batch_id() (great_expectations.render.renderer.content_block.ExpectationStr
 (great_expectations.validation_operators.types.validation_operator_classmethod), 625
 method), 625
 _list_validation_results_by_data_asset_name() (great_expectations.render.renderer.content_block.content_block.
 (great_expectations.validation_operators.types.validation_operator_classmethod), 625
 method), 625
 _list_validation_results_by_expectation_suite_name() (great_expectations.render.renderer.content_block.exception_list
 (great_expectations.validation_operators.types.validation_operator_classmethod), 625
 method), 625
 _list_validation_results_by_validation_result_id(great_expectations.render.renderer.content_block.expectation_str
 (great_expectations.validation_operators.types.validation_operator_classmethod), 625
 method), 625
 _list_validation_statistics() (great_expectations.validation_operators.types.validation_operator_classmethod), 625
 method), 625
 _list_ymls_in_checkpoints_directory() (great_expectations.DataContext method), 661
 _list_ymls_in_checkpoints_directory() (great_expectations.data_context.DataContext
 method), 362
 _list_ymls_in_checkpoints_directory() (great_expectations.data_context.data_context.DataContext
 method), 348
 _load_and_convert_to_dataset_class() (in module great_expectations.util), 656
 _load_checkpoint_yaml_template() (in mod-
 ule great_expectations.cli.checkpoint), 236
 _load_config_variables_file() (great_expectations.data_context.BaseDataContext
 method), 355
 _load_config_variables_file() (great_expectations.data_context.data_context.BaseDataContext
 method), 341
 _load_project_config() (great_expectations.DataContext method),
 661
 _load_project_config() (great_expectations.data_context.DataContext
 method), 362
 _load_project_config() (great_expectations.data_context.data_context.DataContextstatic method), 414
 method), 348
 great_expectations.cli.checkpoint), 237
 (great_expectations.data_context.BaseDataContext
 _load_site_builder_from_site_config()
 (great_expectations.data_context.data_context.BaseDataContext
 method), 340
 _missing_content_block_fn() (great_expectations.render.renderer.content_block.ExceptionListC
 _missing_content_block_fn() (great_expectations.render.renderer.content_block.ExpectationStr
 _missing_content_block_fn() (great_expectations.render.renderer.content_block.content_block.
 _missing_content_block_fn() (great_expectations.render.renderer.content_block.exception_list
 _missing_content_block_fn() (great_expectations.render.renderer.content_block.expectation_str
 _move() (great_expectations.data_context.store.DatabaseStoreBackend
 method), 315
 _move() (great_expectations.data_context.store.InMemoryStoreBackend
 method), 317
 _move() (great_expectations.data_context.store.StoreBackend
 method), 318
 _move() (great_expectations.data_context.store.TupleFilesystemStoreBack
 method), 318
 _move() (great_expectations.data_context.store.TupleGCSSStoreBackend
 method), 319
 _move() (great_expectations.data_context.store.TupleS3StoreBackend
 method), 319
 _move() (great_expectations.data_context.store.database_store_backend.
 method), 305
 _move() (great_expectations.data_context.store.store_backend.InMemory
 method), 310
 _move() (great_expectations.data_context.store.store_backend.StoreBack
 method), 310
 _move() (great_expectations.data_context.store.tuple_store_backend.Tup
 method), 312
 _move() (great_expectations.data_context.store.tuple_store_backend.Tup
 method), 313
 _move() (great_expectations.data_context.store.tuple_store_backend.Tup
 method), 312
 _native_type_type_map() (great_expectations.dataset.PandasDataset
 static method), 516
 _native_type_type_map() (great_expectations.dataset.pandas_dataset.PandasDataset

_normalize_absolute_or_relative_path() (great_expectations.data_context.BaseDataContext class method), 584
 _normalize_absolute_or_relative_path() (great_expectations.data_context.data_context.BaseDataContext class method), 354
 _normalize_absolute_or_relative_path() (great_expectations.data_context.data_context.BaseDataContext class method), 340
 _normalize_store_path() (great_expectations.data_context.BaseDataContext class method), 354
 _normalize_store_path() (great_expectations.data_context.data_context.BaseDataContext class method), 340
 _normalize_store_path() (great_expectations.data_context.data_context.BaseDataContext class method), 340
 _parse_value_set() (great_expectations.dataset.Dataset static method), 511
 _parse_value_set() (great_expectations.dataset.dataset.Dataset static method), 410
 _partitioner() (great_expectations.datasource.batch_kwarg_generator.GlobReaderBatchKwargsGenerator class method), 549
 _partitioner() (great_expectations.datasource.batch_kwarg_generator.GlobReaderBatchKwargsGenerator class method), 552
 _partitioner() (great_expectations.datasource.batch_kwarg_generator.GlobReaderBatchKwargsGenerator class method), 538
 _partitioner() (great_expectations.datasource.batch_kwarg_generator.GlobReaderBatchKwargsGenerator class method), 542
 _process_content_block() (great_expectations.render.render_content_block.ValidationRulesTableConceptBlockRenderer class method), 593
 _process_content_block() (great_expectations.render.render_content_block.ValidationRulesTableConceptBlockRenderer class method), 587
 _process_content_block() (great_expectations.render.render_content_block.ValidationRulesTableConceptBlockRenderer class method), 592
 _process_docs_site_for_checklist() (great_expectations.cli.upgrade_helpers.UpgradeHelperV1 class method), 235
 _process_docs_site_for_checklist() (great_expectations.cli.upgrade_helpers.upgrade_helper_v1.UpgradeHelperV1 class method), 234
 _process_metrics_store_for_checklist() (great_expectations.cli.upgrade_helpers.UpgradeHelperV1 class method), 235
 _process_metrics_store_for_checklist() (great_expectations.cli.upgrade_helpers.upgrade_helper_v1.UpgradeHelperV1 class method), 234
 _process_validations_store_for_checklist() (great_expectations.cli.upgrade_helpers.UpgradeHelperV1 class method), 235
 _process_validations_store_for_checklist() (great_expectations.cli.upgrade_helpers.upgrade_helper_v1.UpgradeHelperV1 class method), 234
 _profile() (great_expectations.profile.BasicDatasetProfiler class method), 584
 _profile() (great_expectations.profile.BasicSuiteBuilderProfiler class method), 586
 _profile() (great_expectations.profile.ColumnsExistProfiler class method), 586
 _profile() (great_expectations.profile.base.DatasetProfiler class method), 580
 _profile() (great_expectations.profile.basic_dataset_profiler.BasicDatasetProfiler class method), 581
 _profile() (great_expectations.profile.basic_suite_builder_profiler.BasicSuiteBuilderProfiler class method), 583
 _profile() (great_expectations.profile.columns_exist.ColumnsExistProfiler class method), 583
 _profile_to_create_a_suite() (in module great_expectations.cli.toolkit), 250
 _project_config_with_variables_substituted() (great_expectations.data_context.BaseDataContext property), 355
 _project_config_with_variables_substituted() (great_expectations.data_context.data_context.BaseDataContext property), 330
 _raise_profiling_errors() (in module great_expectations.cli.toolkit), 250
 _remove_column_expectations() (in module great_expectations.render.render_content_block.ValidationRulesTableConceptBlockRenderer class method), 583
 _remove_table_expectations() (in module great_expectations.render.render_content_block.ValidationRulesTableConceptBlockRenderer class method), 583
 _render_bar_chart_table() (great_expectations.render.render_content_block.ValidationRulesTableConceptBlockRenderer class method), 577
 _render_bar_chart_table() (great_expectations.render.render_content_block.ValidationRulesTableConceptBlockRenderer class method), 592
 _render_bar_chart_table() (great_expectations.render.render_content_block.ValidationRulesTableConceptBlockRenderer class method), 608
 _render_bar_chart_table() (great_expectations.render.render_content_block.ValidationRulesTableConceptBlockRenderer class method), 598
 _render_batch_id_cell() (great_expectations.render.render_content_block.ValidationRulesTableConceptBlockRenderer class method), 609
 _render_batch_id_cell() (great_expectations.render.render_content_block.ValidationRulesTableConceptBlockRenderer class method), 605
 _render_bullet_list() (great_expectations.render.render_content_block.ValidationRulesTableConceptBlockRenderer class method), 576
 _render_bullet_list() (great_expectations.render.render_content_block.ValidationRulesTableConceptBlockRenderer class method), 608
 _render_bullet_list() (great_expectations.render.render_content_block.ValidationRulesTableConceptBlockRenderer class method), 598
 _render_dataset_info() (great_expectations.render.render_content_block.ValidationRulesTableConceptBlockRenderer class method), 598

```

(great_expectations.render.renderer.ProfilingResultsOverviewSectionRenderer.jupyter_ux), 578
class method), 608
_render_dataset_info()
(great_expectations.render.renderer.other_section_renderer.ProfilingResultsOverviewSectionRenderer.jupyter_ux.ProfilingResultsColumnSectionRenderer), 576
class method), 599
_render_expectation_meta_notes()
(great_expectations.render.renderer.content_block.content_block_content_block_renderer), 577
class method), 587
_render_expectation_suite_cell()
(great_expectations.render.renderer.SiteIndexPageRenderer), 608
class method), 609
_render_expectation_suite_cell()
(great_expectations.render.renderer.site_index_page_renderer.SiteIndexPageRenderer), 608
class method), 605
_render_expectation_suite_header()
(great_expectations.render.renderer.ExpectationSuitePageRenderer), 598
class method), 609
_render_expectation_suite_header()
(great_expectations.render.renderer.page_renderer.ExpectationSuitePageRenderer), 608
class method), 600
_render_expectation_suite_info()
(great_expectations.render.renderer.ExpectationSuitePageRenderer), 598
class method), 609
_render_expectation_suite_info()
(great_expectations.render.renderer.page_renderer.ExpectationSuitePageRenderer.jupyter_ux.ProfilingResultsColumnSectionRenderer), 577
class method), 600
_render_expectation_suite_notes()
(great_expectations.render.renderer.ExpectationSuitePageRenderer), 608
class method), 609
_render_expectation_suite_notes()
(great_expectations.render.renderer.page_renderer.ExpectationSuitePageRenderer), 598
class method), 600
_render_expectation_types()
(great_expectations.jupyter_ux.ProfilingResultsColumnSectionRenderer), 609
class method), 577
_render_expectation_types()
(great_expectations.render.renderer.ProfilingResultsColumnSectionRenderer), 599
class method), 608
_render_expectation_types()
(great_expectations.render.renderer.ProfilingResultsOverviewSectionRenderer.jupyter_ux.ProfilingResultsColumnSectionRenderer), 577
class method), 608
_render_expectation_types()
(great_expectations.render.renderer.column_section_renderer.ProfilingResultsColumnSectionRenderer), 608
class method), 598
_render_expectation_types()
(great_expectations.render.renderer.other_section_renderer.ProfilingResultsOverviewSectionRenderer.column_section_renderer.ProfilingResultsColumnSectionRenderer), 598
class method), 599
_render_failed() (great_expectations.jupyter_ux.ProfilingResultsColumnSectionRenderer), 577
class method), 577
_render_failed() (great_expectations.render.renderer.ProfilingResultsColumnSectionRenderer), 608
class method), 608
_render_failed() (great_expectations.render.renderer.column_section_renderer.ProfilingResultsColumnSectionRenderer), 608
class method), 598
_render_for_jupyter() (in module _render_quantile_table())

```

Index	679
--------------	------------


```

attribute), 618
_template (great_expectations.render.view.view.DefaultJinjaComponentView
attribute), 616
_template (great_expectations.render.view.view.DefaultJinjaIndexRegionView
attribute), 616
_template (great_expectations.render.view.view.DefaultJinjaPageView
attribute), 615
_template (great_expectations.render.view.view.DefaultJinjaSectionView
attribute), 616
_template (great_expectations.render.view.view.DefaultJinjaView
attribute), 614
_update_upgrade_log ()
(great_expectations.cli.upgrade_helpers.UpgradeHelperV1
method), 235
_update_upgrade_log ()
(great_expectations.cli.upgrade_helpers.upgrade_helper_v1
method), 234
_update_validation_result_json ()
(great_expectations.cli.upgrade_helpers.UpgradeHelperV1
method), 235
_update_validation_result_json ()
(great_expectations.cli.upgrade_helpers.upgrade_helper_v1
method), 234
_validate_at_least_one_suite_is_listed ()
(in module great_expectations.cli.checkpoint),
237
_validate_checkpoint ()
(great_expectations.DataContext
method), 662
_validate_checkpoint ()
(great_expectations.data_context.DataContext
static method), 363
_validate_checkpoint ()
(great_expectations.data_context.data_context.DataContext
static method), 349
_validate_document ()
(great_expectations.jupyter_ux.DefaultJinjaSectionView
method), 577
_validate_document ()
(great_expectations.render.DefaultJinjaPageView
method), 620
_validate_document ()
(great_expectations.render.view.DefaultJinjaComponentView
method), 617
_validate_document ()
(great_expectations.render.view.DefaultJinjaPageView
method), 618
_validate_document ()
(great_expectations.render.view.DefaultJinjaSectionView
method), 618
_validate_document ()
(great_expectations.render.view.view.DefaultJinjaComponentView
method), 616
validate_document ()
(great_expectations.render.view.view.DefaultJinjaPageView
method), 615
validate_document ()
(great_expectations.render.view.view.DefaultJinjaSectionView
method), 616
validate_document ()
(great_expectations.render.view.view.DefaultJinjaView
method), 316
_validate_key () (great_expectations.data_context.store.HtmlSiteStore
method), 317
_validate_key () (great_expectations.data_context.store.StoreBackend
method), 317
_validate_key () (great_expectations.data_context.store.TupleStoreBackend
method), 317
_validate_key () (great_expectations.data_context.store.html_site_store
method), 317
_validate_key () (great_expectations.data_context.store.store.StoreBackend
method), 309
_validate_key () (great_expectations.data_context.store.store_backend
method), 317
_validate_key () (great_expectations.data_context.store.tuple_store_backend
method), 311
_validate_valdiation_config () (in module
great_expectations.cli.validation_operator),
253
_validate_value ()
(great_expectations.data_context.store.MetricStore
method), 316
_validate_value ()
(great_expectations.data_context.store.StoreBackend
method), 317
_validate_value ()
(great_expectations.data_context.store.TupleStoreBackend
method), 320
_validate_value ()
(great_expectations.data_context.store.metric_store.MetricStore
method), 307
_validate_value ()
(great_expectations.data_context.store.store_backend.StoreBackend
method), 310
_validate_value ()
(great_expectations.data_context.store.tuple_store_backend.TupleStoreBackend
method), 311
_verify_checkpoint_does_not_exist () (in
module great_expectations.cli.checkpoint), 236
_write_checkpoint_script_to_disk () (in
module great_expectations.cli.checkpoint), 237
_write_checkpoint_to_disk () (in module
great_expectations.cli.checkpoint), 236
_verify (great_expectations.types.base.DotDict
attribute), 620

```

A

action_list_to_string() (in module `great_expectations.cli.util`), 252
 ActionAnonymizer (class in `great_expectations.core.usage_statistics.anonymizers.action_anonymizer`), 254
 ActionListValidationOperator (class in `great_expectations.validation_operators`), 641
 ActionListValidationOperator (class in `great_expectations.validation_operators.validation_operators`), 632
 add_authoring_intro() (in `great_expectations.render.renderer.suite_edit_notebook_renderer`), 606
 add_batch_kwargs_generator() (in `great_expectations.data_context.BaseDataContext`), 357
 add_batch_kwargs_generator() (in `great_expectations.data_context.data_context.BaseDataContext`), 343
 add_batch_kwargs_generator() (in `great_expectations.datasource.Datasource`), 569
 add_batch_kwargs_generator() (in `great_expectations.datasource.datasource.Datasource`), 561
 add_citation() (in `great_expectations.core.ExpectationSuite`), 274
 add_citation() (in `great_expectations.data_asset.data_asset.DataAsset`), 288
 add_citation() (in `great_expectations.data_asset.DataAsset`), 300
 add_code_cell() (in `great_expectations.render.renderer.suite_edit_notebook_renderer.SuiteEditNotebookRenderer`), 605
 add_data_context_id_to_url() (in `great_expectations.render.view.view.DefaultJinjaView`), 614
 add_datasource() (in `great_expectations.data_context.BaseDataContext`), 356
 add_datasource() (in `great_expectations.data_context.data_context.BaseDataContext`), 343
 add_datasource() (in `great_expectations.data_context.data_context.DataContext`), 348
 add_datasource() (in `great_expectations.data_context.DataContext`), 362
 add_datasource() (in `great_expectations.DataContext`), 662
 add_datasource() (in module `great_expectations.cli.datasource`), 242
 add_datasource_usage_statistics() (in module `great_expectations.core.usage_statistics.usage_statistics`), 260
 add_expectation_cells_from_suite() (in `great_expectations.render.renderer.suite_edit_notebook_renderer.SuiteEditNotebookRenderer`), 606
 add_expectation_meta() (in `great_expectations.profile.base.DatasetProfiler`), 580
 add_footer() (in `great_expectations.render.renderer.suite_edit_notebook_renderer.SuiteEditNotebookRenderer`), 605
 add_footer() (in `great_expectations.render.renderer.suite_scaffold_notebook_renderer.SuiteScaffoldNotebookRenderer`), 607
 add_header() (in `great_expectations.render.renderer.suite_edit_notebook_renderer.SuiteEditNotebookRenderer`), 605
 add_header() (in `great_expectations.render.renderer.suite_scaffold_notebook_renderer.SuiteScaffoldNotebookRenderer`), 606
 add_markdown_cell() (in `great_expectations.render.renderer.suite_edit_notebook_renderer.SuiteEditNotebookRenderer`), 606
 add_meta() (in `great_expectations.profile.base.DatasetProfiler`), 580
 add_query() (in `great_expectations.datasource.batch_kwargs_generator.batch_kwargs_generator`), 540
 add_query() (in `great_expectations.datasource.batch_kwargs_generator.batch_kwargs_generator`), 551
 add_resource_info_to_index_links_dict() (in `great_expectations.render.renderer.site_builder.DefaultSiteIndexBuilder`), 604
 add_store() (in `great_expectations.data_context.BaseDataContext`), 353
 add_store() (in `great_expectations.data_context.data_context.BaseDataContext`), 340
 add_store() (in `great_expectations.data_context.data_context.DataContext`), 348
 add_store() (in `great_expectations.data_context.DataContext`), 362
 add_store() (in `great_expectations.DataContext`), 661
 add_validation_operator() (in `great_expectations.data_context.BaseDataContext`), 354
 add_validation_operator() (in `great_expectations.data_context.data_context.BaseDataContext`), 340
 all_uncommitted_directories_exist() (in `great_expectations.data_context.data_context.DataContext`), 348
 all_uncommitted_directories_exist() (in `great_expectations.data_context.DataContext`), 362
 all_uncommitted_directories_exist() (in `great_expectations.DataContext`), 661
 AmbiguousDataAssetNameError, 653
 anonymize() (in `great_expectations.core.usage_statistics.anonymizers.action_anonymizer`), 254
 anonymize_action_info() (in `great_expectations.core.usage_statistics.anonymizers.action_anonymizer`), 254

<i>method</i>), 254	258
<code>anonymize_batch_info()</code> (<i>great_expectations.core.usage_statistics.anonymizers.batch_info</i> module, <i>great_expectations.data_context.templates</i>), <i>method</i>), 255	<code>ANONYMIZED_USAGE_STATISTICS_DISABLED</code> (in <i>great_expectations.data_context.templates</i>), 351
<code>anonymize_batch_kwargs()</code> (<i>great_expectations.core.usage_statistics.anonymizers.batch_info</i> module, <i>great_expectations.data_context.templates</i>), <i>method</i>), 255	<code>ANONYMIZED_USAGE_STATISTICS_ENABLED</code> (in <i>great_expectations.data_context.templates</i>), 351
<code>anonymize_data_docs_site_info()</code> (<i>great_expectations.core.usage_statistics.anonymizers.data_docs_site_info</i> module, <i>great_expectations.data_context.templates</i>), <i>method</i>), 255	<code>anonymized_validation_operator_schema</code> (<i>great_expectations.data_context.templates</i>), 258
<code>anonymize_datasource_info()</code> (<i>great_expectations.core.usage_statistics.anonymizers.datasource_info</i> module, <i>great_expectations.data_context.templates</i>), <i>method</i>), 256	<code>AnonymizedUsageStatisticsConfig</code> (class in <i>great_expectations.data_context.templates</i>), 322
<code>anonymize_expectation_suite_info()</code> (<i>great_expectations.core.usage_statistics.anonymizers.expectation_suite_info</i> module, <i>great_expectations.data_context.templates</i>), <i>method</i>), 256	<code>AnonymizedUsageStatisticsConfigSchema</code> (<i>great_expectations.data_context.templates</i>), 322
<code>anonymize_object_info()</code> (<i>great_expectations.core.usage_statistics.anonymizers.anonymize_object_info</i> module, <i>great_expectations.data_context.templates</i>), <i>method</i>), 254	<code>anonymizedUsageStatisticsSchema</code> (in <i>mod-</i> <i>great_expectations.data_context.templates</i>), 329
<code>anonymize_site_builder_info()</code> (<i>great_expectations.core.usage_statistics.anonymizers.site_builder_info</i> module, <i>great_expectations.data_context.templates</i>), <i>method</i>), 256	<code>Anonymizer</code> (class in <i>great_expectations.data_context.templates</i>), 254
<code>anonymize_store_backend_info()</code> (<i>great_expectations.core.usage_statistics.anonymizers.store_backend_info</i> module, <i>great_expectations.data_context.templates</i>), <i>method</i>), 257	<code>anonymous_usage_statistics</code> (<i>great_expectations.data_context.templates</i>), 329
<code>anonymize_store_info()</code> (<i>great_expectations.core.usage_statistics.anonymizers.store_info</i> module, <i>great_expectations.data_context.templates</i>), <i>method</i>), 257	<code>anonymous_usage_statistics()</code> (<i>great_expectations.data_context.templates</i>), 355
<code>anonymize_validation_operator_info()</code> (<i>great_expectations.core.usage_statistics.anonymizers.validation_operator_info</i> module, <i>great_expectations.data_context.templates</i>), <i>method</i>), 257	<code>anonymous_usage_statistics()</code> (<i>great_expectations.data_context.templates</i>), 341
<code>anonymized_action_schema</code> (in <i>module</i> <i>append_expectation()</i> <i>great_expectations.core.usage_statistics.schemas</i>), 258	(<i>great_expectations.core.ExpectationSuite</i> <i>method</i>), 275
<code>anonymized_batch_schema</code> (in <i>module</i> <i>append_expectation()</i> <i>great_expectations.core.usage_statistics.schemas</i>), 258	(<i>great_expectations.data_asset.data_asset.DataAsset</i> <i>method</i>), 284
<code>anonymized_class_info_schema</code> (in <i>module</i> <i>append_expectation()</i> <i>great_expectations.core.usage_statistics.schemas</i>), 258	(<i>great_expectations.data_asset.DataAsset</i> <i>method</i>), 296
<code>anonymized_data_docs_site_schema</code> (in <i>module</i> <i>append_expectation()</i> <i>great_expectations.core.usage_statistics.schemas</i>), 258	<code>asset_globs()</code> (<i>great_expectations.datasource.batch_kwargs_generator</i> <i>property</i>), 537
<code>anonymized_datasource_schema</code> (in <i>module</i> <i>append_expectation()</i> <i>great_expectations.core.usage_statistics.schemas</i>), 258	<code>asset_globs()</code> (<i>great_expectations.datasource.batch_kwargs_generator</i> <i>property</i>), 549
<code>anonymized_expectation_suite_schema</code> (in <i>module</i> <i>append_expectation()</i> <i>great_expectations.core.usage_statistics.schemas</i>), 258	<code>AssetConfiguration</code> (class in <i>great_expectations.datasource.batch_kwargs_generator.table_batch_configuration</i>), 546
<code>anonymized_store_schema</code> (in <i>module</i> <i>append_expectation()</i> <i>great_expectations.core.usage_statistics.schemas</i>), 258	<code>AssetConfigurationSchema</code> (class in <i>great_expectations.datasource.batch_kwargs_generator.table_batch_configuration</i>), 544
<code>anonymized_string_schema</code> (in <i>module</i> <i>append_expectation()</i> <i>great_expectations.core.usage_statistics.schemas</i>), 258	<code>asset_globs()</code> (<i>great_expectations.datasource.batch_kwargs_generator.table_batch_configuration</i>), 546
	<code>assets()</code> (<i>great_expectations.datasource.batch_kwargs_generator.manual_asset_configuration</i>), 546

property), 539

assets() (great_expectations.datasource.batch_kwargs_generator.MutableBatchKwargsGenerator (class in great_expectations.profile), 584

assets() (great_expectations.datasource.batch_kwargs_generator.S3GlobReaderBatchKwargsGenerator (class in great_expectations.profile.basic_suite_builder_profiler), 541

assets() (great_expectations.datasource.batch_kwargs_generator.S3GlobReaderBatchKwargsGenerator (class in great_expectations.profile.basic_suite_builder_profiler), 552

attempt_allowing_relative_error() (great_expectations.dataset.Dataset method), 511

attempt_allowing_relative_error() (great_expectations.dataset.dataset.Dataset method), 410

attempt_allowing_relative_error() (great_expectations.dataset.sqlalchemy_dataset.SqlAlchemyDataset method), 449

attempt_to_open_validation_results_in_data_docs (in module great_expectations.cli.toolkit), 250

attributes_dict_to_html_string() (great_expectations.render.view.view.DefaultJinjaView method), 614

autoinspect() (great_expectations.data_asset.data_asset.DataAsset method), 283

autoinspect() (great_expectations.data_asset.DataAsset method), 295

B

batch_kwargs (in module great_expectations.cli.checkpoint_script_template), 238

BASE_DIRECTORIES (great_expectations.data_context.BaseDataContext attribute), 353

BASE_DIRECTORIES (great_expectations.data_context.data_context.DataContext attribute), 339

base_directory() (great_expectations.datasource.batch_kwargs_generator.GlobReaderBatchKwargsGenerator (class in great_expectations.profile), 537

base_directory() (great_expectations.datasource.batch_kwargs_generator.GlobReaderBatchKwargsGenerator (class in great_expectations.profile), 549

base_directory() (great_expectations.datasource.batch_kwargs_generator.GlobReaderBatchKwargsGenerator (class in great_expectations.profile), 543

base_directory() (great_expectations.datasource.batch_kwargs_generator.GlobReaderBatchKwargsGenerator (class in great_expectations.profile), 553

BaseDataContext (class in great_expectations.data_context), 352

BaseDataContext (class in great_expectations.data_context.data_context), 338

BaseUpgradeHelper (class in great_expectations.cli.upgrade_helpers.base_upgrade_helpers), 233

BasicDatasetProfiler (class in great_expectations.profile), 584

BasicDatasetProfiler (class in great_expectations.profile.basic_dataset_profiler), 581

BasicDatasetProfilerBase (class in great_expectations.profile.basic_dataset_profiler), 580

Batch (class in great_expectations.core.batch), 260

batch (in module great_expectations.cli.tap_template), 248

batch_id() (great_expectations.data_asset.data_asset.DataAsset property), 285

batch_id() (great_expectations.data_asset.DataAsset property), 297

batch_identifier(great_expectations.data_context.types.resource_identifiers.BatchIdentifier), 334

batch_identifier(great_expectations.data_context.types.resource_identifiers.BatchIdentifier), 337

batch_identifier() (great_expectations.core.metric.BatchMetric property), 265

batch_identifier() (great_expectations.data_context.types.resource_identifiers.BatchIdentifier), 333

batch_identifier() (great_expectations.data_context.types.resource_identifiers.BatchIdentifier), 335

batch_kwargs (in module great_expectations.cli.checkpoint_script_template), 238

batch_kwargs (in module great_expectations.cli.tap_template), 248

batch_kwargs_generator(great_expectations.datasource.batch_kwargs_generator.GlobReaderBatchKwargsGenerator (class in great_expectations.profile), 260

batch_kwargs_generator(great_expectations.datasource.batch_kwargs_generator.GlobReaderBatchKwargsGenerator (class in great_expectations.profile), 285

batch_kwargs_generator(great_expectations.datasource.batch_kwargs_generator.GlobReaderBatchKwargsGenerator (class in great_expectations.profile), 297

batch_kwargs_generator(great_expectations.datasource.batch_kwargs_generator.GlobReaderBatchKwargsGenerator (class in great_expectations.profile), 326

batch_markers() (great_expectations.core.batch.Batch property), 261

batch_markers() (great_expectations.data_asset.data_asset.DataAsset property), 285

batch_markers() (great_expectations.data_asset.DataAsset property), 297

batch_parameters() (great_expectations.core.batch.Batch property), 261

batch_parameters() (great_expectations.data_asset.data_asset.DataAsset property), 285

batch_parameters() (great_expectations.data_asset.DataAsset property), 297

property), 297

BatchAnonymizer (class in method), 604
 great_expectations.core.usage_statistics.anonymizer.batch_anonymizer, 255
 method), 603

batches_to_validate (in module build() (great_expectations.render.renderer.site_builder.SiteBuilder
 great_expectations.cli.checkpoint_script_template), method), 602
 238 build_batch_kwargs()

BatchIdentifier (class in (great_expectations.data_context.BaseDataContext
 great_expectations.data_context.types.resource_identifiers), method), 356
 332 build_batch_kwargs()

BatchIdentifierSchema (class in (great_expectations.data_context.data_context.BaseDataContext
 great_expectations.data_context.types.resource_identifiers), method), 342
 333 build_batch_kwargs()

batchIdentifierSchema (in module (great_expectations.datasource.batch_kwargs_generator.batch_kv
 great_expectations.data_context.types.resource_identifiers), method), 535
 338 build_batch_kwargs()

BatchKwargs (class in (great_expectations.datasource.Datasource
 great_expectations.core.id_dict), 264 method), 570

BatchKwargs (class in build_batch_kwargs() (great_expectations.datasource.datasource.Datasource
 great_expectations.datasource.types), 557 method), 562

BatchKwargsAnonymizer (class in method), 562
 great_expectations.core.usage_statistics.anonymizer.batch_kwargs_anonymizer, 255
 build_continuous_partition_object() (in
 module great_expectations.dataset.util), 464

BatchKwargsError, 654 build_configuration()

BatchKwargsGenerator (class in (great_expectations.datasource.Datasource
 great_expectations.datasource.batch_kwargs_generator.batch_kwargs_generator), 534
 build_configuration()

BatchMarkers (class in (great_expectations.datasource.datasource.Datasource
 great_expectations.datasource.types), 558 class method), 560

BatchMarkers (class in build_configuration() (great_expectations.datasource.pandas_datasource.PandasData
 great_expectations.datasource.types.batch_kwargs), (great_expectations.datasource.pandas_datasource.PandasData
 555 class method), 563

BatchMetric (class in build_configuration() (great_expectations.datasource.PandasDatasource
 great_expectations.core.metric), 265 (great_expectations.datasource.PandasDatasource
 class method), 571

BIGQUERY (great_expectations.cli.datasource.SupportedDatabases class method), 571
 attribute), 241 build_configuration()

bigquery_types_tuple (in module (great_expectations.datasource.sparkdf_datasource.SparkDFData
 great_expectations.dataset.sqlalchemy_dataset), class method), 564
 448 build_configuration()

BOOLEAN (great_expectations.profile.base.ProfilerDataType (great_expectations.datasource.SparkDFDatasource
 attribute), 579 class method), 572

BOOLEAN_TYPE_NAMES build_configuration() (great_expectations.profile.basic_dataset_profiler.BasicDatasetProfilerBasic
 attribute), 581 class method), 566

bootstrap_link_element (in module build_configuration() (great_expectations.datasource.SqlAlchemyDatasource
 great_expectations.jupyter_ux), 578 (great_expectations.datasource.SqlAlchemyDatasource
 class method), 536

boto3 (in module great_expectations.datasource.batch_kwargs_generator.batch_kwargs_generator),
 540 build_continuous_partition_object() (in
 module great_expectations.datasource.batch_kwargs_generator.batch_kwargs_generator), 536

bucket() (great_expectations.datasource.batch_kwargs_generator.batch_kwargs_generator, 536
 property), 541 build_data_docs()

bucket() (great_expectations.datasource.batch_kwargs_generator.batch_kwargs_generator, 536
 property), 552 method), 359

bucket() (great_expectations.datasource.util.S3Url build_data_docs() (great_expectations.data_context.data_context.BaseDataContext
 property), 567 (great_expectations.data_context.data_context.BaseDataContext
 class method), 359

method), 345

build_docs() (in module *great_expectations.cli.docs*), 245

BUILD_DOCS_PROMPT (in module *great_expectations.cli.cli_messages*), 239

build_envelope() (*great_expectations.core.usage_statistics.usage_statistics.UsageStatisticsHandler* method), 259

build_evaluation_parameters() (in module *great_expectations.core.evaluation_parameters*), 262

build_init_payload() (*great_expectations.core.usage_statistics.usage_statistics.UsageStatisticsHandler* method), 259

C

CallToActionButton (class in *great_expectations.render.renderer.site_builder*), 604

CallToActionRenderer (class in *great_expectations.render.renderer*), 607

CallToActionRenderer (class in *great_expectations.render.renderer.call_to_action_renderer*), 597

categorical_partition_data() (in module *great_expectations.dataset.util*), 463

check_sql_engine_dialect() (in module *great_expectations.dataset.util*), 466

checkpoint (in module *great_expectations.cli.checkpoint_script_template*), 238

checkpoint() (in module *great_expectations.cli.checkpoint*), 236

checkpoint_list() (in module *great_expectations.cli.checkpoint*), 236

checkpoint_new() (in module *great_expectations.cli.checkpoint*), 236

checkpoint_run() (in module *great_expectations.cli.checkpoint*), 236

checkpoint_script() (in module *great_expectations.cli.checkpoint*), 236

CheckpointError, 651

CheckpointNotFoundError, 651

CHECKPOINTS_DIR (*great_expectations.data_context.BaseDataContext* attribute), 353

CHECKPOINTS_DIR (*great_expectations.data_context.data_context.BaseDataContext* attribute), 339

class_name (*great_expectations.data_context.types.base.DataSourceConfigSchema* attribute), 326

class_name (*great_expectations.types.configurations.ClassConfigSchema* attribute), 622

class_name() (*great_expectations.data_context.types.base.DataSourceConfig* property), 322

class_name() (*great_expectations.types.ClassConfig* property), 623

class_name() (*great_expectations.types.configurations.ClassConfig* property), 621

ClassConfig (class in *great_expectations.types*), 623

ClassConfig (class in *great_expectations.types.configurations*), 621

ClassConfigSchema (class in *great_expectations.types.configurations*), 621

classConfigSchema (in module *great_expectations.types.configurations*), 621

ClassInstantiationError, 654

clean_data_docs() (*great_expectations.data_context.BaseDataContext* method), 359

clean_data_docs() (*great_expectations.data_context.data_context.BaseDataContext* method), 345

clean_data_docs() (in module *great_expectations.cli.docs*), 245

clean_empty() (*great_expectations.core.ExpectationSuiteSchema* method), 278

clean_site() (*great_expectations.data_context.store.html_site_store.HtmlSiteStore* method), 307

clean_site() (*great_expectations.data_context.store.HtmlSiteStore* method), 316

clean_site() (*great_expectations.render.renderer.site_builder.SiteBuilder* method), 602

clear_stack() (*great_expectations.core.evaluation_parameters.EvaluationParameters* method), 262

cli() (in module *great_expectations.cli*), 253

cli() (in module *great_expectations.cli.cli*), 238

cli_as_beta() (*great_expectations.cli.mark.Mark* static method), 246

cli_as_deprecation() (*great_expectations.cli.mark.Mark* static method), 246

cli_as_experimental() (*great_expectations.cli.mark.Mark* static method), 245

cli_colorize_string() (in module *great_expectations.cli.util*), 252

cli_message() (in module *great_expectations.cli.util*), 252

cli_message_dict() (in module *great_expectations.cli.util*), 252

cli_message_list() (in module *great_expectations.cli.util*), 252

cli_new_ds_choice_payload (in module *great_expectations.core.usage_statistics.schemas*), 258

CollapseContent (class in *great_expectations.render.types*), 613

colored (in module `great_expectations.cli.util`), 252

`column_aggregate_expectation()` (`great_expectations.dataset.dataset.MetaDataset` class method), 365

`column_map_expectation()` (`great_expectations.dataset.dataset.MetaDataset` class method), 364

`column_map_expectation()` (`great_expectations.dataset.MetaPandasDataset` class method), 511

`column_map_expectation()` (`great_expectations.dataset.pandas_dataset.MetaPandasDataset` class method), 410

`column_map_expectation()` (`great_expectations.dataset.sparkdf_dataset.MetaSparkDFDataset` class method), 433

`column_map_expectation()` (`great_expectations.dataset.sqlalchemy_dataset.MetaSqlAlchemyDataset` class method), 448

`column_pair_map_expectation()` (`great_expectations.dataset.MetaPandasDataset` class method), 512

`column_pair_map_expectation()` (`great_expectations.dataset.pandas_dataset.MetaPandasDataset` class method), 410

`column_pair_map_expectation()` (`great_expectations.dataset.sparkdf_dataset.MetaSparkDFDataset` class method), 433

`column_reflection_fallback()` (`great_expectations.dataset.sqlalchemy_dataset.SqlAlchemyDataset` class method), 450

`ColumnSectionRenderer` (class in `great_expectations.render.renderer.column_section_renderer`), 598

`ColumnsExistProfiler` (class in `great_expectations.render.renderer.column_section_renderer`), 586

`ColumnsExistProfiler` (class in `great_expectations.profile.columns_exist`), 583

`commented_map()` (`great_expectations.data_context.types.base.DataContextConfig` property), 322

`COMPLETE_ONBOARDING_PROMPT` (in module `great_expectations.cli.cli_messages`), 239

`config()` (`great_expectations.datasource.Datasource` property), 569

`config()` (`great_expectations.datasource.datasource.Datasource` property), 561

`config_variables_file_path` (`great_expectations.data_context.types.base.DataContextConfig` attribute), 329

`CONFIG_VARIABLES_INTRO` (in module `great_expectations.data_context.templates`), 350

`CONFIG_VARIABLES_TEMPLATE` (in module `great_expectations.data_context.templates`), 351

`config_variables_yaml_exist()` (`great_expectations.data_context.data_context.DataContext` class method), 348

`config_variables_yaml_exist()` (`great_expectations.data_context.DataContext` class method), 362

`config_variables_yaml_exist()` (`great_expectations.DataContext` class method), 661

`create_expectation_suite()` (`great_expectations.data_context.types.base.DataContext` attribute), 328

`create_expectation_suite()` (`great_expectations.data_context.types.base.DataContext` class method), 322

`ConfigNotFoundError`, 654

`confirm_proceed_or_exit()` (in module `great_expectations.cli.toolkit`), 251

`ContentBlockRenderer` (class in `great_expectations.render.renderer.content_block.content_block`), 587

`context` (in module `great_expectations.cli.checkpoint_script_template`), 238

`context` (in module `great_expectations.cli.tap_template`), 248

`create_partition_data()` (in module `great_expectations.dataset.util`), 463

`convert_result_to_serializable()` (`great_expectations.core.ExpectationValidationResultSchema` method), 280

`convert_to_json_serializable()` (in module `great_expectations.core`), 270

`convert_to_string_and_escape()` (in module `great_expectations.render.renderer.column_section_renderer`), 598

`cooltip_style_element` (in module `great_expectations.jupyter_ux`), 578

`copy_static_assets()` (`great_expectations.data_context.store.html_site_store.HtmlSiteStore` method), 307

`copy_static_assets()` (`great_expectations.data_context.store.HtmlSiteStore` method), 316

`create()` (`great_expectations.data_context.data_context.DataContext` class method), 347

`create()` (`great_expectations.data_context.DataContext` class method), 361

`create_empty_suite()` (in module `great_expectations.cli.toolkit`), 250

`create_expectation_suite()` (`great_expectations.data_context.BaseDataContext` class method), 661

method), 357

create_expectation_suite() (great_expectations.data_context.data_context.BaseDataContext method), 343

create_expectation_suite() (in module DataAsset (class in great_expectations.data_asset), great_expectations.cli.toolkit), 249

create_expectation_widget() (great_expectations.jupyter UX expectation_explorer.ExpectationExplorer method), 576

create_multiple_expectations() (in module DataAssetProfiler (class in great_expectations.dataset.util), 466

create_temporary_table() (great_expectations.dataset.sqlalchemy_dataset.SqlAlchemyDataset method), 450

credentials (great_expectations.data_context.types.base.DataSourceConfigSchema attribute), 326

ctx (in module great_expectations.render.util), 619

CURRENT_CONFIG_VERSION (in module great_expectations.data_context.types.base), 321

D

data() (great_expectations.core.batch.Batch property), 261

data_asset_name (great_expectations.data_context.types.resource_identifiers.BatchIdentifier attribute), 334

data_asset_name() (great_expectations.core.metric.ValidationMetric property), 265

data_asset_name() (great_expectations.core.metric.ValidationMetricIdentifier property), 266

data_asset_name() (great_expectations.data_context.types.resource_identifiers.BatchIdentifier property), 333

data_asset_type (great_expectations.core.ExpectationSuiteSchema attribute), 278

data_asset_type (great_expectations.data_context.types.resource_identifiers.BatchIdentifier attribute), 326

data_context() (great_expectations.core.batch.Batch property), 261

data_context() (great_expectations.datasource.DataSource property), 569

data_context() (great_expectations.datasource.DataSource property), 561

data_context_id (great_expectations.data_context.types.base.AnonymizedUsageStatisticsConfigSchema attribute), 324

data_context_id() (great_expectations.data_context.BaseDataContext property), 355

data_context_id() (great_expectations.data_context.data_context.BaseDataContext property), 341

data_context_id() (great_expectations.data_context.data_context.BaseDataContext property), 341

(great_expectations.data_context.types.base.AnonymizedUsageStatisticsConfigSchema property), 322

data_context_sites (great_expectations.data_context.types.base.DataContext attribute), 328

DataAsset (class in great_expectations.data_asset), 295

DataAsset (class in great_expectations.data_asset.data_asset), 283

DataAssetProfiler (class in great_expectations.profile.base), 580

DatabaseStoreBackend (class in great_expectations.data_context.store), 315

DatabaseStoreBackend (class in great_expectations.data_context.store.database_store_backend), 305

DatabricksTableBatchKwargsGenerator (class in great_expectations.datasource.batch_kwargs_generator), 548

DatabricksTableBatchKwargsGenerator (class in great_expectations.datasource.batch_kwargs_generator), 536

DataContext (class in great_expectations), 660

DataContext (class in great_expectations), 661

DataContext (class in great_expectations), 661

DataContext (class in great_expectations), 661

DataContextConfig (class in great_expectations.data_context.types.base), 321

DataContextConfigSchema (class in great_expectations.data_context.types.base), 327

DataContextConfigSchema (in module great_expectations.data_context.types.base), 329

DataSourceConfigSchema (class in great_expectations.core), 268

DataContextKey (class in great_expectations.core), 261

DataContextKey (class in great_expectations.core), 261

DataDocsAnonymizer (class in great_expectations.core.usage_statistics.anonymizers.data_docs_anonymizer), 555

Dataset (class in great_expectations.dataset), 466

Dataset (class in great_expectations.dataset.dataset), 365

dataset() (great_expectations.datasource.types.batch_kwargs.InMemoryBatchKwargs property), 556

dataset() (great_expectations.datasource.types.InMemoryBatchKwargs property), 559

DatasetProfiler (class in great_expectations.datasource.types.batch_kwargs.InMemoryBatchKwargs property), 559

[great_expectations.profile.base](#)), 580
 Datasource (class in [great_expectations.datasource](#)), 568
 Datasource (class in [great_expectations.datasource.datasource](#)), 560
 datasource () (in module [great_expectations.cli.datasource](#)), 241
 datasource_initialization_exceptions (in module [great_expectations.datasource.sqlalchemy_datasource](#)), 565
 datasource_list () (in module [great_expectations.cli.datasource](#)), 242
 datasource_name () (in [great_expectations.core.batch.Batch](#) property), 260
 datasource_new () (in module [great_expectations.cli.datasource](#)), 241
 datasource_profile () (in module [great_expectations.cli.datasource](#)), 242
 datasource_sqlalchemy_connect_payload (in module [great_expectations.core.usage_statistics.schemas](#)), 258
 DATASOURCE_TYPE_BY_DATASOURCE_CLASS (in module [great_expectations.cli.datasource](#)), 241
 DatasourceAnonymizer (class in [great_expectations.core.usage_statistics.anonymizers.datasource_anonymizers](#)), 256
 DatasourceConfig (class in [great_expectations.data_context.types.base](#)), 322
 DatasourceConfigSchema (class in [great_expectations.data_context.types.base](#)), 324
 datasourceConfigSchema (in module [great_expectations.data_context.types.base](#)), 329
 DatasourceConfigSchema.Meta (class in [great_expectations.data_context.types.base](#)), 326
 DatasourceInitializationError, 654
 datasources ([great_expectations.data_context.types.base.DataContextConfigSchema](#) attribute), 328
 datasources () ([great_expectations.data_context.BaseDataContext](#) method), 355
 datasources () ([great_expectations.data_context.data_context.BaseDataContext](#) property), 341
 DatasourceTypes (class in [great_expectations.cli.datasource](#)), 241
 DATETIME ([great_expectations.profile.base.ProfilerDataSet](#) attribute), 580
 DATETIME_TYPE_NAMES (in [great_expectations.profile.basic_dataset_profiler.BasicDatasetProfilerBase](#) attribute), 581
 default_flow_style (in module [great_expectations.cli.toolkit](#)), 249
 default_flow_style (in module [great_expectations.data_context.data_context](#)), 338
 default_flow_style (in module [great_expectations.datasource.datasource](#)), 559
 DEFAULT_PRECISION (in module [great_expectations.render.util](#)), 619
 DEFAULT_USAGE_STATISTICS_URL (in module [great_expectations.data_context.types.base](#)), 321
 DefaultJinjaComponentView (class in [great_expectations.render.view](#)), 617
 DefaultJinjaComponentView (class in [great_expectations.render.view.view](#)), 616
 DefaultJinjaIndexPageView (class in [great_expectations.render.view](#)), 617
 DefaultJinjaIndexPageView (class in [great_expectations.render.view.view](#)), 615
 DefaultJinjaPageView (class in [great_expectations.render](#)), 619
 DefaultJinjaPageView (class in [great_expectations.render.view](#)), 617
 DefaultJinjaPageView (class in [great_expectations.render.view.view](#)), 615
 DefaultJinjaSectionView (class in [great_expectations.jupyter_ux](#)), 577
 DefaultJinjaSectionView (class in [great_expectations.render.view](#)), 618
 DefaultJinjaSectionView (class in [great_expectations.render.view.view](#)), 616
 DefaultJinjaView (class in [great_expectations.render.view.view](#)), 614
 DefaultSiteIndexBuilder (class in [great_expectations.render.renderer.site_builder](#)), 603
 DefaultSiteSectionBuilder (class in [great_expectations.render.renderer.site_builder](#)), 603
 DataContextConfigSchema (class in [great_expectations.data_context.BaseDataContext](#) attribute), 355
 delete_datasource () (in module [great_expectations.cli.datasource](#)), 241
 delete_expectation_suite () (in [great_expectations.data_context.BaseDataContext](#) method), 357
 BasicDatasetProfilerBase (class in [great_expectations.data_context.data_context.BaseDataContext](#) attribute), 581

method), 344

deserialize() (great_expectations.data_context.store.expectations_store.ExpectationsStore method), 306

deserialize() (great_expectations.data_context.store.ExpectationsStore method), 315

deserialize() (great_expectations.data_context.store.metric_store.MetricStore method), 307

deserialize() (great_expectations.data_context.store.MetricStore method), 316

deserialize() (great_expectations.data_context.store.Store method), 317

deserialize() (great_expectations.data_context.store.store.Store method), 309

deserialize() (great_expectations.data_context.store.validations_store.ValidationsStore method), 314

deserialize() (great_expectations.data_context.store.ValidationsStore method), 320

DictDot (class in great_expectations.core), 269

DictDot (class in great_expectations.types), 623

discard_failing_expectations() (great_expectations.data_asset.data_asset.DataAsset method), 285

discard_failing_expectations() (great_expectations.data_asset.DataAsset method), 297

display_column_evrs_as_section() (in module great_expectations.jupyter_ux), 579

display_column_expectations_as_section() (in module great_expectations.jupyter_ux), 578

display_profiled_column_evrs_as_section() (in module great_expectations.jupyter_ux), 578

do_config_check() (in module great_expectations.cli.project), 246

DocInherit (class in great_expectations.data_asset.util), 294

docs() (in module great_expectations.cli.docs), 244

docs_build() (in module great_expectations.cli.docs), 244

docs_list() (in module great_expectations.cli.docs), 245

does_config_exist_on_disk() (great_expectations.data_context.data_context.DataContext class method), 349

does_config_exist_on_disk() (great_expectations.data_context.DataContext class method), 363

does_config_exist_on_disk() (great_expectations.DataContext class method), 662

does_project_have_a_datasource_in_config_file() (great_expectations.data_context.data_context.DataContext class method), 349

does_project_have_a_datasource_in_config_file() (great_expectations.data_context.DataContext class method), 363

deserialize() (great_expectations.data_context.store.expectations_store.ExpectationsStore method), 662

DONE (in module great_expectations.cli.cli_messages), 244

DotDict (class in great_expectations.types.base), 620

MyYAML (great_expectations.cli.toolkit.MyYAML method), 249

E

edit_expectation_suite() (great_expectations.data_asset.data_asset.DataAsset method), 295

edit_expectation_suite() (great_expectations.jupyter_ux.expectation_explorer.ExpectationExplorer method), 576

edit_expectation_suite_usage_statistics() (in module great_expectations.core.usage_statistics.usage_statistics), 260

emit() (great_expectations.core.usage_statistics.usage_statistics.UsageStatistics method), 259

empty_payload_schema (in module great_expectations.core.usage_statistics.schemas), 258

enabled (great_expectations.data_context.types.base.AnonymizedUsageStatistics attribute), 324

enabled() (great_expectations.data_context.types.base.AnonymizedUsageStatistics property), 322

ensure_json_serializable() (in module great_expectations.core), 270

evaluate_stack() (great_expectations.core.evaluation_parameters.EvaluationParameters method), 262

evaluation_parameter_store() (great_expectations.data_context.BaseDataContext property), 358

evaluation_parameter_store() (great_expectations.data_context.data_context.BaseDataContext property), 344

evaluation_parameter_store_name (great_expectations.data_context.types.base.DataContextConfigSchema attribute), 328

evaluation_parameter_store_name() (great_expectations.data_context.BaseDataContext property), 358

evaluation_parameter_store_name() (great_expectations.data_context.data_context.BaseDataContext property), 344

evaluation_parameters (great_expectations.core.ExpectationSuiteSchema attribute), 278

Index	691
--------------	------------

`expect_column_pair_values_to_be_in_set()` `expect_column_stdev_to_be_between()`
`(great_expectations.dataset.pandas_dataset.PandasDataset (great_expectations.render.renderer.content_block.expectation_str`
`method), 431` `class method), 591`
`expect_column_pair_values_to_be_in_set()` `expect_column_stdev_to_be_between()`
`(great_expectations.dataset.PandasDataset (great_expectations.render.renderer.content_block.ExpectationStr`
`method), 532` `class method), 596`
`expect_column_parameterized_distributionekspetestobumalsemtobbebetweenthan()`
`(great_expectations.dataset.Dataset method), (great_expectations.dataset.Dataset method),`
`488` `501`
`expect_column_parameterized_distributionekspetestobumalsemtobbebetweenthan()`
`(great_expectations.dataset.dataset.Dataset (great_expectations.dataset.dataset.Dataset`
`method), 387` `method), 399`
`expect_column_parameterized_distributionekspetestobumalsemtobbebetweenthan()`
`(great_expectations.dataset.pandas_dataset.PandasDataset (great_expectations.render.renderer.content_block.expectation_str`
`method), 427` `class method), 591`
`expect_column_parameterized_distributionekspetestobumalsemtobbebetweenthan()`
`(great_expectations.dataset.PandasDataset (great_expectations.render.renderer.content_block.ExpectationStr`
`method), 528` `class method), 596`
`expect_column_proportion_of_unique_valuesexpectbebetweenend)exist()`
`(great_expectations.dataset.Dataset method), (great_expectations.dataset.Dataset method),`
`498` `469`
`expect_column_proportion_of_unique_valuesexpectbebetweenend)exist()`
`(great_expectations.dataset.dataset.Dataset (great_expectations.dataset.dataset.Dataset`
`method), 397` `method), 367`
`expect_column_proportion_of_unique_valuesexpectbebetweenend)exist()`
`(great_expectations.render.renderer.content_block.expectation_str (great_expectations.render.renderer.content_block.expectation_str`
`class method), 590` `class method), 589`
`expect_column_proportion_of_unique_valuesexpectbebetweenend)exist()`
`(great_expectations.render.renderer.content_block.ExpectationStr (great_expectations.render.renderer.content_block.ExpectationStr`
`class method), 595` `class method), 594`
`expect_column_proportion_of_unique_valuesexpectbebetweenunique_value_count_to_be_between()`
`(great_expectations.render.renderer.content_block.profiling (great_expectations.render.renderer.content_block.profiling_overv`
`class method), 592` `497`
`expect_column_proportion_of_unique_valuesexpectbebetweenunique_value_count_to_be_between()`
`(great_expectations.render.renderer.content_block.ProfilingOverViewTableContentBlock (great_expectations.render.renderer.content_block.ProfilingOver`
`class method), 596` `method), 396`
`expect_column_quantile_values_to_be_betweenexpect_column_unique_value_count_to_be_between()`
`(great_expectations.dataset.Dataset method), (great_expectations.render.renderer.content_block.expectation_str`
`495` `class method), 589`
`expect_column_quantile_values_to_be_betweenexpect_column_unique_value_count_to_be_between()`
`(great_expectations.dataset.dataset.Dataset (great_expectations.render.renderer.content_block.ExpectationStr`
`method), 393` `class method), 594`
`expect_column_quantile_values_to_be_betweenexpect_column_unique_value_count_to_be_between()`
`(great_expectations.render.renderer.content_block.expectation_str (great_expectations.render.renderer.content_block.expectation_str`
`class method), 591` `class method), 592`
`expect_column_quantile_values_to_be_betweenexpect_column_unique_value_count_to_be_between()`
`(great_expectations.render.renderer.content_block.ExpectationStr (great_expectations.render.renderer.content_block.ProfilingOverv`
`class method), 596` `class method), 596`
`expect_column_stdev_to_be_between()` `expect_column_value_lengths_to_be_between()`
`(great_expectations.dataset.Dataset method), (great_expectations.dataset.Dataset method),`
`496` `481`
`expect_column_stdev_to_be_between()` `expect_column_value_lengths_to_be_between()`
`(great_expectations.dataset.dataset.Dataset (great_expectations.dataset.dataset.Dataset`
`method), 395` `method), 380`

<code>expect_column_value_lengths_to_be_between()</code> (<i>great_expectations.dataset.pandas_dataset.PandasDataset</i> method), 420	<code>expect_column_values_to_be_between()</code> (<i>great_expectations.dataset.sparkdf_dataset.SparkDFDataset</i> method), 436
<code>expect_column_value_lengths_to_be_between()</code> (<i>great_expectations.dataset.PandasDataset</i> method), 521	<code>expect_column_values_to_be_between()</code> (<i>great_expectations.dataset.sqlalchemy_dataset.SqlAlchemyDataset</i> method), 455
<code>expect_column_value_lengths_to_be_between()</code> (<i>great_expectations.dataset.sparkdf_dataset.SparkDFDataset</i> method), 437	<code>expect_column_values_to_be_between()</code> (<i>great_expectations.render.renderer.content_block.expectation_string_renderer.ExpectationStringRenderer</i> class method), 589
<code>expect_column_value_lengths_to_be_between()</code> (<i>great_expectations.dataset.sqlalchemy_dataset.SqlAlchemyDataset</i> method), 457	<code>expect_column_values_to_be_between()</code> (<i>great_expectations.render.renderer.content_block.ExpectationStringRenderer</i> class method), 594
<code>expect_column_value_lengths_to_be_between()</code> (<i>great_expectations.render.renderer.content_block.expectation_string_renderer.ExpectationStringRenderer</i> class method), 590	<code>expect_column_values_to_be_dateutil_parseable()</code> (<i>great_expectations.dataset.pandas_dataset.PandasDataset</i> method), 486
<code>expect_column_value_lengths_to_be_between()</code> (<i>great_expectations.render.renderer.content_block.ExpectationStringRenderer</i> class method), 595	<code>expect_column_values_to_be_dateutil_parseable()</code> (<i>great_expectations.render.renderer.content_block.ExpectationStringRenderer</i> class method), 385
<code>expect_column_value_lengths_to_equal()</code> (<i>great_expectations.dataset.Dataset</i> method), 482	<code>expect_column_values_to_be_dateutil_parseable()</code> (<i>great_expectations.dataset.pandas_dataset.PandasDataset</i> method), 425
<code>expect_column_value_lengths_to_equal()</code> (<i>great_expectations.dataset.dataset.Dataset</i> method), 381	<code>expect_column_values_to_be_dateutil_parseable()</code> (<i>great_expectations.dataset.PandasDataset</i> method), 526
<code>expect_column_value_lengths_to_equal()</code> (<i>great_expectations.dataset.pandas_dataset.PandasDataset</i> method), 421	<code>expect_column_values_to_be_dateutil_parseable()</code> (<i>great_expectations.render.renderer.content_block.expectation_string_renderer.ExpectationStringRenderer</i> class method), 590
<code>expect_column_value_lengths_to_equal()</code> (<i>great_expectations.dataset.PandasDataset</i> method), 522	<code>expect_column_values_to_be_dateutil_parseable()</code> (<i>great_expectations.render.renderer.content_block.ExpectationStringRenderer</i> class method), 595
<code>expect_column_value_lengths_to_equal()</code> (<i>great_expectations.dataset.sparkdf_dataset.SparkDFDataset</i> method), 439	<code>expect_column_values_to_be_decreasing()</code> (<i>great_expectations.dataset.Dataset</i> method), 480
<code>expect_column_value_lengths_to_equal()</code> (<i>great_expectations.dataset.sqlalchemy_dataset.SqlAlchemyDataset</i> method), 456	<code>expect_column_values_to_be_decreasing()</code> (<i>great_expectations.dataset.dataset.Dataset</i> method), 379
<code>expect_column_value_lengths_to_equal()</code> (<i>great_expectations.render.renderer.content_block.expectation_string_renderer.ExpectationStringRenderer</i> class method), 590	<code>expect_column_values_to_be_decreasing()</code> (<i>great_expectations.render.renderer.content_block.ExpectationStringRenderer</i> class method), 419
<code>expect_column_value_lengths_to_equal()</code> (<i>great_expectations.render.renderer.content_block.ExpectationStringRenderer</i> class method), 595	<code>expect_column_values_to_be_decreasing()</code> (<i>great_expectations.render.renderer.content_block.ExpectationStringRenderer</i> class method), 521
<code>expect_column_values_to_be_between()</code> (<i>great_expectations.dataset.Dataset</i> method), 478	<code>expect_column_values_to_be_decreasing()</code> (<i>great_expectations.render.renderer.content_block.expectation_string_renderer.ExpectationStringRenderer</i> class method), 590
<code>expect_column_values_to_be_between()</code> (<i>great_expectations.dataset.dataset.Dataset</i> method), 377	<code>expect_column_values_to_be_decreasing()</code> (<i>great_expectations.render.renderer.content_block.ExpectationStringRenderer</i> class method), 595
<code>expect_column_values_to_be_between()</code> (<i>great_expectations.dataset.pandas_dataset.PandasDataset</i> method), 417	<code>expect_column_values_to_be_in_set()</code> (<i>great_expectations.dataset.Dataset</i> method), 476
<code>expect_column_values_to_be_between()</code> (<i>great_expectations.dataset.PandasDataset</i> method), 518	<code>expect_column_values_to_be_in_set()</code> (<i>great_expectations.dataset.dataset.Dataset</i> method), 375

<code>expect_column_values_to_be_in_set()</code> (<i>great_expectations.dataset.pandas_dataset.PandasDataset</i> <i>great_expectations.render.renderer.content_block.expectation_str</i> <i>method</i>), 415	<code>expect_column_values_to_be_increasing()</code> <i>class method</i>), 590
<code>expect_column_values_to_be_in_set()</code> (<i>great_expectations.dataset.PandasDataset</i> <i>method</i>), 516	<code>expect_column_values_to_be_increasing()</code> (<i>great_expectations.render.renderer.content_block.ExpectationStr</i> <i>class method</i>), 595
<code>expect_column_values_to_be_in_set()</code> (<i>great_expectations.dataset.sparkdf_dataset.SparkDFDataset</i> <i>great_expectations.dataset.Dataset</i> <i>method</i>), 434	<code>expect_column_values_to_be_json_parseable()</code> <i>great_expectations.render.renderer.content_block.expectation_str</i> <i>class method</i>), 487
<code>expect_column_values_to_be_in_set()</code> (<i>great_expectations.dataset.sqlalchemy_dataset.SqlAlchemyDataset</i> <i>great_expectations.dataset.dataset.Dataset</i> <i>method</i>), 453	<code>expect_column_values_to_be_json_parseable()</code> <i>great_expectations.render.renderer.content_block.expectation_str</i> <i>class method</i>), 385
<code>expect_column_values_to_be_in_set()</code> (<i>great_expectations.render.renderer.content_block.expectation_str</i> <i>great_expectations.render.renderer.content_block.expectation_str</i> <i>class method</i>), 590	<code>expect_column_values_to_be_json_parseable()</code> (<i>great_expectations.render.renderer.content_block.expectation_str</i> <i>great_expectations.render.renderer.content_block.expectation_str</i> <i>class method</i>), 425
<code>expect_column_values_to_be_in_set()</code> (<i>great_expectations.render.renderer.content_block.ExpectationStr</i> <i>great_expectations.render.renderer.content_block.expectation_str</i> <i>class method</i>), 595	<code>expect_column_values_to_be_json_parseable()</code> (<i>great_expectations.render.renderer.content_block.ExpectationStr</i> <i>great_expectations.render.renderer.content_block.expectation_str</i> <i>class method</i>), 527
<code>expect_column_values_to_be_in_type_list()</code> (<i>great_expectations.dataset.Dataset</i> <i>method</i>), 475	<code>expect_column_values_to_be_json_parseable()</code> (<i>great_expectations.render.renderer.content_block.expectation_str</i> <i>class method</i>), 590
<code>expect_column_values_to_be_in_type_list()</code> (<i>great_expectations.dataset.dataset.Dataset</i> <i>method</i>), 374	<code>expect_column_values_to_be_json_parseable()</code> (<i>great_expectations.render.renderer.content_block.ExpectationStr</i> <i>class method</i>), 595
<code>expect_column_values_to_be_in_type_list()</code> (<i>great_expectations.dataset.pandas_dataset.PandasDataset</i> <i>great_expectations.dataset.Dataset</i> <i>method</i>), 414	<code>expect_column_values_to_be_null()</code> (<i>great_expectations.dataset.Dataset</i> <i>method</i>), 474
<code>expect_column_values_to_be_in_type_list()</code> (<i>great_expectations.dataset.PandasDataset</i> <i>method</i>), 516	<code>expect_column_values_to_be_null()</code> (<i>great_expectations.dataset.dataset.Dataset</i> <i>method</i>), 372
<code>expect_column_values_to_be_in_type_list()</code> (<i>great_expectations.dataset.sparkdf_dataset.SparkDFDataset</i> <i>great_expectations.dataset.pandas_dataset.PandasDataset</i> <i>method</i>), 443	<code>expect_column_values_to_be_null()</code> (<i>great_expectations.render.renderer.content_block.expectation_str</i> <i>great_expectations.render.renderer.content_block.expectation_str</i> <i>class method</i>), 413
<code>expect_column_values_to_be_in_type_list()</code> (<i>great_expectations.dataset.sqlalchemy_dataset.SqlAlchemyDataset</i> <i>great_expectations.dataset.PandasDataset</i> <i>method</i>), 452	<code>expect_column_values_to_be_null()</code> (<i>great_expectations.render.renderer.content_block.expectation_str</i> <i>great_expectations.render.renderer.content_block.expectation_str</i> <i>class method</i>), 515
<code>expect_column_values_to_be_in_type_list()</code> (<i>great_expectations.render.renderer.content_block.expectation_str</i> <i>great_expectations.render.renderer.content_block.expectation_str</i> <i>class method</i>), 590	<code>expect_column_values_to_be_null()</code> (<i>great_expectations.render.renderer.content_block.expectation_str</i> <i>great_expectations.render.renderer.content_block.expectation_str</i> <i>class method</i>), 441
<code>expect_column_values_to_be_in_type_list()</code> (<i>great_expectations.render.renderer.content_block.ExpectationStr</i> <i>great_expectations.render.renderer.content_block.expectation_str</i> <i>class method</i>), 595	<code>expect_column_values_to_be_null()</code> (<i>great_expectations.render.renderer.content_block.ExpectationStr</i> <i>great_expectations.render.renderer.content_block.expectation_str</i> <i>class method</i>), 450
<code>expect_column_values_to_be_increasing()</code> (<i>great_expectations.dataset.Dataset</i> <i>method</i>), 479	<code>expect_column_values_to_be_null()</code> (<i>great_expectations.render.renderer.content_block.expectation_str</i> <i>class method</i>), 589
<code>expect_column_values_to_be_increasing()</code> (<i>great_expectations.dataset.dataset.Dataset</i> <i>method</i>), 378	<code>expect_column_values_to_be_null()</code> (<i>great_expectations.render.renderer.content_block.ExpectationStr</i> <i>class method</i>), 594
<code>expect_column_values_to_be_increasing()</code> (<i>great_expectations.dataset.pandas_dataset.PandasDataset</i> <i>great_expectations.render.renderer.content_block.profilng_overv</i> <i>method</i>), 418	<code>expect_column_values_to_be_null()</code> <i>class method</i>), 592
<code>expect_column_values_to_be_increasing()</code> (<i>great_expectations.dataset.PandasDataset</i> <i>method</i>), 520	<code>expect_column_values_to_be_null()</code> (<i>great_expectations.render.renderer.content_block.ProfilngOverv</i> <i>class method</i>), 596

<code>expect_column_values_to_be_of_type()</code> (<i>great_expectations.dataset.Dataset</i> method), 474	<code>expect_column_values_to_match_json_schema()</code> (<i>great_expectations.dataset.pandas_dataset.PandasDataset</i> method), 426
<code>expect_column_values_to_be_of_type()</code> (<i>great_expectations.dataset.dataset.Dataset</i> method), 373	<code>expect_column_values_to_match_json_schema()</code> (<i>great_expectations.dataset.PandasDataset</i> method), 528
<code>expect_column_values_to_be_of_type()</code> (<i>great_expectations.dataset.pandas_dataset.PandasDataset</i> method), 414	<code>expect_column_values_to_match_json_schema()</code> (<i>great_expectations.dataset.sparkdf_dataset.SparkDFDataset</i> method), 441
<code>expect_column_values_to_be_of_type()</code> (<i>great_expectations.dataset.PandasDataset</i> method), 515	<code>expect_column_values_to_match_json_schema()</code> (<i>great_expectations.render.renderer.content_block.expectation_str</i> class method), 591
<code>expect_column_values_to_be_of_type()</code> (<i>great_expectations.dataset.sparkdf_dataset.SparkDFDataset</i> method), 442	<code>expect_column_values_to_match_json_schema()</code> (<i>great_expectations.render.renderer.content_block.ExpectationStr</i> class method), 595
<code>expect_column_values_to_be_of_type()</code> (<i>great_expectations.dataset.sqlalchemy_dataset.SqlAlchemyDataset</i> method), 452	<code>expect_column_values_to_match_regex()</code> (<i>great_expectations.dataset.Dataset</i> method), 483
<code>expect_column_values_to_be_of_type()</code> (<i>great_expectations.render.renderer.content_block.expectation_str</i> class method), 590	<code>expect_column_values_to_match_regex()</code> (<i>great_expectations.render.renderer.content_block.ExpectationStr</i> class method), 594
<code>expect_column_values_to_be_of_type()</code> (<i>great_expectations.render.renderer.content_block.ExpectationStr</i> class method), 594	<code>expect_column_values_to_match_regex()</code> (<i>great_expectations.render.renderer.content_block.ExpectationStr</i> class method), 594
<code>expect_column_values_to_be_unique()</code> (<i>great_expectations.dataset.Dataset</i> method), 472	<code>expect_column_values_to_match_regex()</code> (<i>great_expectations.render.renderer.content_block.ExpectationStr</i> class method), 594
<code>expect_column_values_to_be_unique()</code> (<i>great_expectations.dataset.dataset.Dataset</i> method), 371	<code>expect_column_values_to_match_regex()</code> (<i>great_expectations.render.renderer.content_block.ExpectationStr</i> class method), 594
<code>expect_column_values_to_be_unique()</code> (<i>great_expectations.dataset.pandas_dataset.PandasDataset</i> method), 412	<code>expect_column_values_to_match_regex()</code> (<i>great_expectations.render.renderer.content_block.ExpectationStr</i> class method), 594
<code>expect_column_values_to_be_unique()</code> (<i>great_expectations.dataset.PandasDataset</i> method), 513	<code>expect_column_values_to_match_regex()</code> (<i>great_expectations.render.renderer.content_block.ExpectationStr</i> class method), 594
<code>expect_column_values_to_be_unique()</code> (<i>great_expectations.dataset.sparkdf_dataset.SparkDFDataset</i> method), 438	<code>expect_column_values_to_match_regex()</code> (<i>great_expectations.render.renderer.content_block.ExpectationStr</i> class method), 594
<code>expect_column_values_to_be_unique()</code> (<i>great_expectations.dataset.sqlalchemy_dataset.SqlAlchemyDataset</i> method), 458	<code>expect_column_values_to_match_regex_list()</code> (<i>great_expectations.render.renderer.content_block.ExpectationStr</i> class method), 591
<code>expect_column_values_to_be_unique()</code> (<i>great_expectations.render.renderer.content_block.expectation_str</i> class method), 591	<code>expect_column_values_to_match_regex_list()</code> (<i>great_expectations.render.renderer.content_block.ExpectationStr</i> class method), 591
<code>expect_column_values_to_be_unique()</code> (<i>great_expectations.render.renderer.content_block.ExpectationStr</i> class method), 596	<code>expect_column_values_to_match_regex_list()</code> (<i>great_expectations.render.renderer.content_block.ExpectationStr</i> class method), 596
<code>expect_column_values_to_match_json_schema()</code> (<i>great_expectations.dataset.Dataset</i> method), 487	<code>expect_column_values_to_match_regex_list()</code> (<i>great_expectations.render.renderer.content_block.ExpectationStr</i> class method), 596
<code>expect_column_values_to_match_json_schema()</code> (<i>great_expectations.dataset.dataset.Dataset</i> method), 386	<code>expect_column_values_to_match_regex_list()</code> (<i>great_expectations.render.renderer.content_block.ExpectationStr</i> class method), 596

<code>expect_column_values_to_match_regex_list()</code> (<code>great_expectations.dataset.sqlalchemy_dataset.SqlAlchemyDataset</code> method), 460	<code>expect_column_values_to_not_be_null()</code> (<code>great_expectations.dataset.Dataset</code> method), 473
<code>expect_column_values_to_match_regex_list()</code> (<code>great_expectations.render.renderer.content_block.expectation_group_renderer.StringRendererDataset</code> class method), 590	<code>expect_column_values_to_not_be_null()</code> (<code>great_expectations.render.renderer.content_block.expectation_group_renderer.StringRendererDataset</code> class method), 372
<code>expect_column_values_to_match_regex_list()</code> (<code>great_expectations.render.renderer.content_block.ExpectationGroupRendererPandasDataset</code> class method), 595	<code>expect_column_values_to_not_be_null()</code> (<code>great_expectations.render.renderer.content_block.ExpectationGroupRendererPandasDataset</code> class method), 413
<code>expect_column_values_to_match_strftime_format()</code> (<code>great_expectations.dataset.Dataset</code> method), 486	<code>expect_column_values_to_not_be_null()</code> (<code>great_expectations.dataset.PandasDataset</code> method), 514
<code>expect_column_values_to_match_strftime_format()</code> (<code>great_expectations.dataset.dataset.Dataset</code> method), 384	<code>expect_column_values_to_not_be_null()</code> (<code>great_expectations.dataset.sparkdf_dataset.SparkDFDataset</code> method), 440
<code>expect_column_values_to_match_strftime_format()</code> (<code>great_expectations.dataset.pandas_dataset.PandasDataset</code> method), 424	<code>expect_column_values_to_not_be_null()</code> (<code>great_expectations.dataset.sqlalchemy_dataset.SqlAlchemyDataset</code> method), 451
<code>expect_column_values_to_match_strftime_format()</code> (<code>great_expectations.dataset.PandasDataset</code> method), 526	<code>expect_column_values_to_not_be_null()</code> (<code>great_expectations.render.renderer.content_block.expectation_group_renderer.StringRendererDataset</code> class method), 589
<code>expect_column_values_to_match_strftime_format()</code> (<code>great_expectations.dataset.sparkdf_dataset.SparkDFDataset</code> method), 440	<code>expect_column_values_to_not_be_null()</code> (<code>great_expectations.render.renderer.content_block.ExpectationGroupRendererPandasDataset</code> class method), 594
<code>expect_column_values_to_match_strftime_format()</code> (<code>great_expectations.render.renderer.content_block.expectation_group_renderer.StringRendererDataset</code> class method), 590	<code>expect_column_values_to_not_be_null()</code> (<code>great_expectations.render.renderer.content_block.ExpectationGroupRendererPandasDataset</code> class method), 592
<code>expect_column_values_to_match_strftime_format()</code> (<code>great_expectations.render.renderer.content_block.ExpectationGroupRendererPandasDataset</code> class method), 595	<code>expect_column_values_to_not_be_null()</code> (<code>great_expectations.render.renderer.content_block.ProfilingOverviewRendererDataset</code> class method), 596
<code>expect_column_values_to_not_be_in_set()</code> (<code>great_expectations.dataset.Dataset</code> method), 477	<code>expect_column_values_to_not_match_regex()</code> (<code>great_expectations.dataset.Dataset</code> method), 483
<code>expect_column_values_to_not_be_in_set()</code> (<code>great_expectations.dataset.dataset.Dataset</code> method), 376	<code>expect_column_values_to_not_match_regex()</code> (<code>great_expectations.dataset.dataset.Dataset</code> method), 382
<code>expect_column_values_to_not_be_in_set()</code> (<code>great_expectations.dataset.pandas_dataset.PandasDataset</code> method), 416	<code>expect_column_values_to_not_match_regex()</code> (<code>great_expectations.dataset.pandas_dataset.PandasDataset</code> method), 422
<code>expect_column_values_to_not_be_in_set()</code> (<code>great_expectations.dataset.PandasDataset</code> method), 517	<code>expect_column_values_to_not_match_regex()</code> (<code>great_expectations.dataset.PandasDataset</code> method), 524
<code>expect_column_values_to_not_be_in_set()</code> (<code>great_expectations.dataset.sparkdf_dataset.SparkDFDataset</code> method), 435	<code>expect_column_values_to_not_match_regex()</code> (<code>great_expectations.dataset.sparkdf_dataset.SparkDFDataset</code> method), 444
<code>expect_column_values_to_not_be_in_set()</code> (<code>great_expectations.dataset.sqlalchemy_dataset.SqlAlchemyDataset</code> method), 454	<code>expect_column_values_to_not_match_regex()</code> (<code>great_expectations.dataset.sqlalchemy_dataset.SqlAlchemyDataset</code> method), 459
<code>expect_column_values_to_not_be_in_set()</code> (<code>great_expectations.render.renderer.content_block.expectation_group_renderer.StringRendererDataset</code> class method), 590	<code>expect_column_values_to_not_match_regex()</code> (<code>great_expectations.render.renderer.content_block.expectation_group_renderer.StringRendererDataset</code> class method), 590
<code>expect_column_values_to_not_be_in_set()</code> (<code>great_expectations.render.renderer.content_block.ExpectationGroupRendererPandasDataset</code> class method), 595	<code>expect_column_values_to_not_match_regex()</code> (<code>great_expectations.render.renderer.content_block.ExpectationGroupRendererPandasDataset</code> class method), 595

`expect_table_column_count_to_equal()` (great_expectations.render.renderer.content_block.expectation_string.ExpectationStringRenderer class method), 589
`expect_table_column_count_to_equal()` (great_expectations.render.renderer.content_block.expectation_string.ExpectationStringRenderer class method), 594
`expect_table_columns_to_match_ordered_list()` (great_expectations.dataset.Dataset method), 469
`expect_table_columns_to_match_ordered_list()` (great_expectations.dataset.Dataset method), 368
`expect_table_columns_to_match_ordered_list()` (great_expectations.render.renderer.content_block.expectation_string.ExpectationStringRenderer class method), 589
`expect_table_columns_to_match_ordered_list()` (great_expectations.render.renderer.content_block.expectation_string.ExpectationStringRenderer class method), 594
`expect_table_row_count_to_be_between()` (great_expectations.dataset.Dataset method), 471
`expect_table_row_count_to_be_between()` (great_expectations.dataset.Dataset method), 370
`expect_table_row_count_to_be_between()` (great_expectations.render.renderer.content_block.expectation_string.ExpectationStringRenderer class method), 589
`expect_table_row_count_to_be_between()` (great_expectations.render.renderer.content_block.expectation_string.ExpectationStringRenderer class method), 594
`expect_table_row_count_to_equal()` (great_expectations.dataset.Dataset method), 472
`expect_table_row_count_to_equal()` (great_expectations.dataset.Dataset method), 370
`expect_table_row_count_to_equal()` (great_expectations.render.renderer.content_block.expectation_string.ExpectationStringRenderer class method), 589
`expect_table_row_count_to_equal()` (great_expectations.render.renderer.content_block.expectation_string.ExpectationStringRenderer class method), 594
`expectation()` (great_expectations.data_asset.data_asset.DataAsset class method), 283
`expectation()` (great_expectations.data_asset.DataAsset class method), 295
`expectation_config` (great_expectations.core.ExpectationValidationResultSchema attribute), 279
`expectation_kwarg_dict_to_ge_kwargs()` (great_expectations.jupyter UX expectation_explorer.ExpectationExplorer class method), 574
`expectation_suite_identifier` (great_expectations.data_context.types.resource_identifiers.ValidationResultIdentifierSchema attribute), 315
`attribute`, 337
`ExpectationString`, `ExpectationStringRenderer` (great_expectations.core.metric.ValidationMetric property), 265
`ExpectationStringRenderer`.`identifier()` (great_expectations.core.metric.ValidationMetricIdentifier property), 266
`expectation_suite_identifier()` (great_expectations.data_context.types.resource_identifiers.ValidationMetricIdentifier property), 335
`expectation_suite_name` (great_expectations.core.ExpectationSuiteSchema attribute), 277
`ExpectationString`, `ExpectationStringRenderer` (great_expectations.data_context.types.resource_identifiers.ExpectationSuiteIdentifier property), 332
`ExpectationStringRenderer`.`name()` (great_expectations.data_asset.data_asset.DataAsset property), 288
`expectation_suite_name()` (great_expectations.data_asset.DataAsset property), 300
`expectation_suite_name()` (great_expectations.data_context.types.resource_identifiers.ExpectationSuiteIdentifier property), 330
`ExpectationString`, `ExpectationStringRenderer` (great_expectations.core.ExpectationConfiguration attribute), 274
`expectation_type()` (great_expectations.core.ExpectationConfiguration property), 272
`ExpectationConfiguration` (class in `great_expectations.core`), 272
`ExpectationConfigurationSchema` (class in `great_expectations.core`), 272
`expectationConfigurationSchema` (in module `great_expectations.core`), 282
`ExpectationExplorer` (class in `great_expectations.render.renderer.content_block.expectation_string`), 574
`ExpectationKwargs` (class in `great_expectations.render.renderer.content_block.expectation_string`), 272
`expectations` (great_expectations.core.ExpectationSuiteSchema attribute), 277
`expectations_store_name` (great_expectations.data_context.types.base.DataContextConfiguration attribute), 328
`expectations_store_name()` (great_expectations.data_context.BaseDataContext property), 355
`expectations_store_name()` (great_expectations.data_context.data_context.BaseDataContext property), 341
`ExpectationsStore` (class in `great_expectations.data_context.types.resource_identifiers.ValidationResultIdentifierSchema`), 315

ExpectationsStore (class in [great_expectations.core](#)), 278
[great_expectations.data_context.store.expectations_store](#) (class in [great_expectations.core](#)), 278
 306

ExpectationStringRenderer (class in [expectationValidationResultSchema](#) (in [great_expectations.core](#)), 282
[great_expectations.render.renderer.content_block](#)),
 594

ExpectationStringRenderer (class in [property](#)), 322
[great_expectations.render.renderer.content_block.ExpectationStringContext](#) (class in [great_expectations.data_context](#)), 363
 589

ExpectationSuite (class in [ExplorerDataContext](#) (class in [great_expectations.data_context.data_context](#)),
[great_expectations.core](#)), 274

ExpectationSuiteAnonymizer (class in [349](#)
[great_expectations.core.usage_statistics.anonymizer.expectationSuiteAnonymizer](#) (in module [great_expectations.core.evaluation_parameters](#)),
 256 263

ExpectationSuiteBulletListContentBlockRenderer (class in [great_expectations.render.renderer.content_block](#)),
 593

ExpectationSuiteBulletListContentBlockRenderer (class in [great_expectations.render.renderer.content_block.bullet_list_content_block](#) (in module [great_expectations.data_context.data_context.BaseDataContext](#) attribute), 353
[great_expectations.render.renderer.content_block.bullet_list_content_block](#) (in module [great_expectations.data_context.data_context.BaseDataContext](#) attribute), 339
 587

ExpectationSuiteColumnSectionRenderer (class in [great_expectations.jupyter_ux](#)), 576
[FEW](#) ([great_expectations.profile.base.ProfilerCardinality](#) attribute), 580

ExpectationSuiteColumnSectionRenderer (class in [great_expectations.render.renderer](#)),
[file_lines_map_expectation\(\)](#) ([great_expectations.data_asset.file_data_asset.MetaFileDataAsset](#) class method), 289
 607

ExpectationSuiteColumnSectionRenderer (class in [great_expectations.render.renderer.column_section_renderer](#) (in module [great_expectations.data_context.util](#)), 352
[file_relative_path\(\)](#) (in module [great_expectations.data_context.util](#)), 352
 598

ExpectationSuiteIdentifier (class in [great_expectations.data_asset](#)), 301
[great_expectations.data_context.types.resource_identifiers](#) (class in [great_expectations.data_asset](#)), 301
 330

ExpectationSuiteIdentifierSchema (class in [great_expectations.data_asset.file_data_asset](#)),
 290

ExpectationSuiteIdentifierSchema (class in [great_expectations.data_context.types.resource_identifiers](#)),
[implicit\(\)](#) ([great_expectations.data_context.types.base.AnonymizedUsageStatistics](#) method), 324
 330

expectationSuiteIdentifierSchema (in module [great_expectations.data_context.types.resource_identifiers](#)),
[find_context_root_dir\(\)](#) ([great_expectations.data_context.data_context.DataContext](#) class method), 349
 338

ExpectationSuiteNotFoundError, 654

ExpectationSuitePageRenderer (class in [find_context_root_dir\(\)](#) ([great_expectations.data_context.DataContext](#) class method), 362
[great_expectations.render.renderer](#)), 609

ExpectationSuitePageRenderer (class in [find_context_root_dir\(\)](#) ([great_expectations.DataContext](#) class method), 662
[great_expectations.render.renderer.page_renderer](#)), 599

ExpectationSuiteSchema (class in [great_expectations.core](#)), 276
[find_context_yaml_file\(\)](#) ([great_expectations.data_context.data_context.DataContext](#) class method), 349

expectationSuiteSchema (in module [great_expectations.core](#)), 282

ExpectationSuiteValidationResult (class in [great_expectations.core](#)), 280
[find_context_yaml_file\(\)](#) ([great_expectations.data_context.DataContext](#) class method), 363

ExpectationSuiteValidationResultSchema (class in [great_expectations.core](#)), 280
[find_context_yaml_file\(\)](#) ([great_expectations.DataContext](#) class method), 662

expectationSuiteValidationResultSchema (in module [great_expectations.core](#)), 282

ExpectationValidationResult (class in

`find_evaluation_parameter_dependencies()` `from_fixed_length_tuple()`
 (in module `great_expectations.core`), 268 (`great_expectations.core.data_context_key.DataContextKey`
`find_evaluation_parameter_dependencies()` class method), 261
 (in module `great_expectations.core.evaluation_parameters`), 263
`find_expectation_indexes()` `from_fixed_length_tuple()`
 (`great_expectations.core.ExpectationSuite` class method), 261
 method), 276 (`great_expectations.core.DataContextKey`
`find_expectation_indexes()` class method), 268
 (`great_expectations.data_asset.data_asset.DataAsset` `from_fixed_length_tuple()`
 method), 284 (`great_expectations.core.metric.MetricIdentifier`
`find_expectation_indexes()` class method), 265
 (`great_expectations.data_asset.DataAsset` `from_fixed_length_tuple()`
 method), 296 (`great_expectations.core.metric.ValidationMetricIdentifier`
`find_expectations()` class method), 266
 (`great_expectations.core.ExpectationSuite` `from_fixed_length_tuple()`
 method), 276 (`great_expectations.core.RunIdentifier` class
`find_expectations()` method), 270
 (`great_expectations.data_asset.data_asset.DataAsset` `from_fixed_length_tuple()`
 method), 285 (`great_expectations.data_context.types.resource_identifiers.ExpectationIdentifier`
`find_expectations()` class method), 330
 (`great_expectations.data_asset.DataAsset` `from_fixed_length_tuple()`
 method), 296 (`great_expectations.data_context.types.resource_identifiers.ValidationMetricIdentifier`
`fixed_length_key()` class method), 335
 (`great_expectations.data_context.store.store_backend.StoreBackend` `from_fixed_length_tuple()`
 property), 309 (`great_expectations.core.metric.MetricIdentifier`
 class method), 265
`fixed_length_key()` `from_object()` (`great_expectations.core.metric.ValidationMetricIdentifier`
 (`great_expectations.data_context.store.StoreBackend` class method), 266
 property), 317 `from_object()` (`great_expectations.data_context.types.resource_identifiers.ExpectationIdentifier`
 class method), 335
`FLOAT` (`great_expectations.profile.base.ProfilerDataType` `from_pandas()` (in module `great_expectations.util`),
 attribute), 579
`FLOAT_TYPE_NAMES` (`great_expectations.profile.basic_dataset_profiler.BasicDatasetProfilerBase`
 attribute), 581 `from_tuple()` (`great_expectations.core.data_context_key.DataContextKey`
 class method), 261
`fn` (`great_expectations.core.evaluation_parameters.EvaluationParameters` `from_tuple()` (`great_expectations.core.DataContextKey`
 attribute), 262 class method), 268
`format_dict_for_error_message()` (in mod- `from_tuple()` (`great_expectations.core.metric.MetricIdentifier`
 ule `great_expectations.data_context.util`), 351 class method), 265
`from_commented_map()` `from_tuple()` (`great_expectations.core.metric.ValidationMetricIdentifier`
 (`great_expectations.data_context.types.base.DataContextConfig` class method), 266
 class method), 322
`from_configuration()` `from_tuple()` (`great_expectations.core.RunIdentifier`
 (`great_expectations.datasource.Datasource` class method), 270
 class method), 568 `from_tuple()` (`great_expectations.data_context.types.resource_identifiers.ExpectationIdentifier`
 class method), 333
`from_configuration()` `from_tuple()` (`great_expectations.data_context.types.resource_identifiers.ValidationMetricIdentifier`
 (`great_expectations.datasource.datasource.Datasource` class method), 330
 class method), 560
`from_dataset()` (`great_expectations.dataset.Dataset` `from_tuple()` (`great_expectations.data_context.types.resource_identifiers.ExpectationIdentifier`
 class method), 467 class method), 338
`from_dataset()` (`great_expectations.dataset.dataset.Dataset` `from_tuple()` (`great_expectations.data_context.types.resource_identifiers.ValidationMetricIdentifier`
 class method), 365 class method), 335
`from_dataset()` (`great_expectations.dataset.sparkdf_dataset.SparkDFDataset`
 class method), 433
`from_dataset()` (`great_expectations.dataset.sqlalchemy_dataset.SqlAlchemyDataset`
 class method), 449 `GE_IDFR` (`great_expectations.data_context.BaseDataContext`
 attribute), 353

G

Index	703
--------------	------------

<i>method</i>), 513	<i>method</i>), 433
get_column_count_in_range() (<i>great_expectations.dataset.sparkdf_dataset.SparkDFDataset</i> <i>method</i>), 434	get_column_mean() (<i>great_expectations.dataset.sqlalchemy_dataset.SqlAlchemyData</i> <i>method</i>), 450
get_column_count_in_range() (<i>great_expectations.dataset.sqlalchemy_dataset.SqlAlchemyDataset</i> <i>method</i>), 450	get_column_median() (<i>great_expectations.dataset.Dataset</i> <i>method</i>), 467
get_column_expectations() (<i>great_expectations.core.ExpectationSuite</i> <i>method</i>), 274	get_column_median() (<i>great_expectations.dataset.dataset.Dataset</i> <i>method</i>), 366
get_column_hist() (<i>great_expectations.dataset.Dataset</i> <i>method</i>), 468	get_column_median() (<i>great_expectations.dataset.pandas_dataset.PandasDataset</i> <i>method</i>), 412
get_column_hist() (<i>great_expectations.dataset.dataset.Dataset</i> <i>method</i>), 366	get_column_median() (<i>great_expectations.dataset.PandasDataset</i> <i>method</i>), 513
get_column_hist() (<i>great_expectations.dataset.pandas_dataset.PandasDataset</i> <i>method</i>), 412	get_column_median() (<i>great_expectations.dataset.sparkdf_dataset.SparkDFDataset</i> <i>method</i>), 434
get_column_hist() (<i>great_expectations.dataset.PandasDataset</i> <i>method</i>), 513	get_column_median() (<i>great_expectations.dataset.sqlalchemy_dataset.SqlAlchemyData</i> <i>method</i>), 450
get_column_hist() (<i>great_expectations.dataset.sparkdf_dataset.SparkDFDataset</i> <i>method</i>), 434	get_column_min() (<i>great_expectations.dataset.Dataset</i> <i>method</i>), 467
get_column_hist() (<i>great_expectations.dataset.sqlalchemy_dataset.SqlAlchemyDataset</i> <i>method</i>), 450	get_column_min() (<i>great_expectations.dataset.dataset.Dataset</i> <i>method</i>), 366
get_column_max() (<i>great_expectations.dataset.Dataset</i> <i>method</i>), 467	get_column_min() (<i>great_expectations.dataset.pandas_dataset.Panda</i> <i>method</i>), 411
get_column_max() (<i>great_expectations.dataset.dataset.Dataset</i> <i>method</i>), 366	get_column_min() (<i>great_expectations.dataset.PandasDataset</i> <i>method</i>), 513
get_column_max() (<i>great_expectations.dataset.pandas_dataset.PandasDataset</i> <i>method</i>), 411	get_column_min() (<i>great_expectations.dataset.sparkdf_dataset.Spark</i> <i>method</i>), 433
get_column_max() (<i>great_expectations.dataset.PandasDataset</i> <i>method</i>), 513	get_column_min() (<i>great_expectations.dataset.sqlalchemy_dataset.Sq</i> <i>method</i>), 449
get_column_max() (<i>great_expectations.dataset.sparkdf_dataset.SparkDFDataset</i> <i>method</i>), 433	get_column_modes() (<i>great_expectations.dataset.Dataset</i> <i>method</i>),
get_column_max() (<i>great_expectations.dataset.sqlalchemy_dataset.SqlAlchemyDataset</i> <i>method</i>), 449	get_column_modes() (<i>great_expectations.dataset.dataset.Dataset</i> <i>method</i>), 366
get_column_mean() (<i>great_expectations.dataset.Dataset</i> <i>method</i>), 467	get_column_modes() (<i>great_expectations.dataset.pandas_dataset.PandasDataset</i> <i>method</i>), 412
get_column_mean() (<i>great_expectations.dataset.dataset.Dataset</i> <i>method</i>), 365	get_column_modes() (<i>great_expectations.dataset.PandasDataset</i> <i>method</i>), 513
get_column_mean() (<i>great_expectations.dataset.pandas_dataset.PandasDataset</i> <i>method</i>), 411	get_column_modes() (<i>great_expectations.dataset.sparkdf_dataset.SparkDFDataset</i> <i>method</i>), 434
get_column_mean() (<i>great_expectations.dataset.PandasDataset</i> <i>method</i>), 513	get_column_names() (<i>great_expectations.jupyter UX expectation_explorer.Expectation</i> <i>method</i>), 576
get_column_mean() (<i>great_expectations.dataset.sparkdf_dataset.SparkDFDataset</i> <i>method</i>),	get_column_nonnull_count() (<i>great_expectations.dataset.Dataset</i> <i>method</i>),

467	
<code>get_column_nonnull_count()</code> (<i>great_expectations.dataset.dataset.Dataset</i> <i>method</i>), 365	<i>method</i>), 434
<code>get_column_nonnull_count()</code> (<i>great_expectations.dataset.pandas_dataset.PandasDataset</i> <i>method</i>), 467	<code>get_column_stdev()</code> (<i>great_expectations.dataset.sqlalchemy_dataset.SqlAlchemyData</i> <i>method</i>), 450
<code>get_column_nonnull_count()</code> (<i>great_expectations.dataset.PandasDataset</i> <i>method</i>), 513	<code>get_column_sum()</code> (<i>great_expectations.dataset.Dataset</i> <i>method</i>), 366
<code>get_column_nonnull_count()</code> (<i>great_expectations.dataset.sparkdf_dataset.SparkDFDataset</i> <i>method</i>), 512	<code>get_column_sum()</code> (<i>great_expectations.dataset.pandas_dataset.Panda</i> <i>method</i>), 411
<code>get_column_nonnull_count()</code> (<i>great_expectations.dataset.sqlalchemy_dataset.SqlAlchemyDataset</i> <i>method</i>), 449	<code>get_column_sum()</code> (<i>great_expectations.dataset.PandasDataset</i> <i>method</i>), 512
<code>get_column_partition()</code> (<i>great_expectations.dataset.Dataset</i> <i>method</i>), 468	<code>get_column_sum()</code> (<i>great_expectations.dataset.sparkdf_dataset.Spark</i> <i>method</i>), 433
<code>get_column_partition()</code> (<i>great_expectations.dataset.dataset.Dataset</i> <i>method</i>), 366	<code>get_column_sum()</code> (<i>great_expectations.dataset.sqlalchemy_dataset.Sq</i> <i>method</i>), 449
<code>get_column_quantiles()</code> (<i>great_expectations.dataset.Dataset</i> <i>method</i>), 467	<code>get_column_unique_count()</code> (<i>great_expectations.dataset.Dataset</i> <i>method</i>), 467
<code>get_column_quantiles()</code> (<i>great_expectations.dataset.dataset.Dataset</i> <i>method</i>), 366	<code>get_column_unique_count()</code> (<i>great_expectations.dataset.dataset.Dataset</i> <i>method</i>), 366
<code>get_column_quantiles()</code> (<i>great_expectations.dataset.pandas_dataset.PandasDataset</i> <i>method</i>), 412	<code>get_column_unique_count()</code> (<i>great_expectations.dataset.pandas_dataset.PandasDataset</i> <i>method</i>), 412
<code>get_column_quantiles()</code> (<i>great_expectations.dataset.PandasDataset</i> <i>method</i>), 513	<code>get_column_unique_count()</code> (<i>great_expectations.dataset.PandasDataset</i> <i>method</i>), 513
<code>get_column_quantiles()</code> (<i>great_expectations.dataset.sparkdf_dataset.SparkDFDataset</i> <i>method</i>), 434	<code>get_column_unique_count()</code> (<i>great_expectations.dataset.sqlalchemy_dataset.SqlAlchemyData</i> <i>method</i>), 450
<code>get_column_quantiles()</code> (<i>great_expectations.dataset.sqlalchemy_dataset.SqlAlchemyDataset</i> <i>method</i>), 450	<code>get_column_value_counts()</code> (<i>great_expectations.dataset.Dataset</i> <i>method</i>), 467
<code>get_column_stdev()</code> (<i>great_expectations.dataset.Dataset</i> <i>method</i>), 468	<code>get_column_value_counts()</code> (<i>great_expectations.dataset.dataset.Dataset</i> <i>method</i>), 365
<code>get_column_stdev()</code> (<i>great_expectations.dataset.dataset.Dataset</i> <i>method</i>), 366	<code>get_column_value_counts()</code> (<i>great_expectations.dataset.pandas_dataset.PandasDataset</i> <i>method</i>), 411
<code>get_column_stdev()</code> (<i>great_expectations.dataset.pandas_dataset.PandasDataset</i> <i>method</i>), 412	<code>get_column_value_counts()</code> (<i>great_expectations.dataset.PandasDataset</i> <i>method</i>), 513
<code>get_column_stdev()</code> (<i>great_expectations.dataset.PandasDataset</i> <i>method</i>), 513	<code>get_column_value_counts()</code> (<i>great_expectations.dataset.sparkdf_dataset.SparkDFDataset</i> <i>method</i>), 433
<code>get_column_stdev()</code> (<i>great_expectations.dataset.sparkdf_dataset.SparkDFDataset</i> <i>method</i>), 357	<code>get_column_value_counts()</code> (<i>great_expectations.dataset.sqlalchemy_dataset.SqlAlchemyData</i> <i>method</i>), 449
	<code>get_config()</code> (<i>great_expectations.data_context.BaseDataContext</i> <i>method</i>), 357

`get_config()` (`great_expectations.data_context.data_context.BaseDataContext` suite()
 method), 343
`get_config()` (`great_expectations.datasource.batch_kwarg_generator.BatchKwargsGenerator`
 method), 535
`get_config()` (in module (`great_expectations.data_context.BaseDataContext`
`great_expectations._version`), 649 method), 358
`get_config_value()` (`great_expectations.data_asset.data_asset.DataAsset` (`great_expectations.data_context.data_context.BaseDataContext`
 method), 285 method), 344
`get_config_value()` (`great_expectations.data_asset.DataAsset` (`great_expectations.jupyter_ux.expectation_explorer.Expectation`
 method), 296 method), 576
`get_config_with_variables_substituted()` `get_expectations_config()`
 (`great_expectations.data_context.BaseDataContext` (`great_expectations.data_asset.data_asset.DataAsset`
 method), 355 method), 285
`get_config_with_variables_substituted()` `get_expectations_config()`
 (`great_expectations.data_context.data_context.BaseDataContext` (`great_expectations.data_asset.DataAsset`
 method), 341 method), 297
`get_dataset()` (`great_expectations.validator.validator.Validator` `get_config_version()`
 method), 648 (`great_expectations.data_context.data_context.DataContext`
 method), 349
`get_datasource()` (`great_expectations.data_context.BaseDataContext` method), 349
`get_datasource()` (`great_expectations.data_context.data_context.BaseDataContext` (`great_expectations.data_context.DataContext`
 method), 343 class method), 362
`get_default_expectation_arguments()` `get_ge_config_version()`
 (`great_expectations.data_asset.data_asset.DataAsset` (`great_expectations.DataContext` class
 method), 285 method), 662
`get_default_expectation_arguments()` `get_html_escaped_json_string_from_dict()`
 (`great_expectations.data_asset.DataAsset` (`great_expectations.render.view.view.DefaultJinjaView`
 method), 297 method), 614
`get_docs_sites_urls()` `get_init_kwarg()`
 (`great_expectations.data_context.BaseDataContext` (`great_expectations.dataset.sqlalchemy_dataset.SqlAlchemyBatch`
 method), 354 method), 448
`get_docs_sites_urls()` `get_iterator()` (`great_expectations.datasource.batch_kwarg_generator`
 (`great_expectations.data_context.data_context.BaseDataContext` method), 535
 method), 340
`get_evaluation_parameter()` `get_keywords()` (in module
 (`great_expectations.data_asset.data_asset.DataAsset` `great_expectations._version`), 649
 method), 288 method), 280
`get_evaluation_parameter()` `get_metric()` (`great_expectations.core.ExpectationSuiteValidationResult`
 (`great_expectations.data_asset.DataAsset` method), 278
 method), 300
`get_evaluation_parameter_dependencies()` `get_metric_kwarg_id()` (in module
 (`great_expectations.core.ExpectationConfiguration` `great_expectations.core`), 270
 method), 272 method), 262
`get_evaluation_parameter_dependencies()` `get_parser()` (`great_expectations.core.evaluation_parameters.Evaluation`
 (`great_expectations.core.ExpectationSuite` method), 274
 method), 274
`get_expectation_state()` `get_query_result()`
 (`great_expectations.jupyter_ux.expectation_explorer.ExpectationExplorer` (`great_expectations.data_context.store.query_store.SqlAlchemyQ`
 method), 574 method), 308
`get_expectation_suite()` `get_resource_url()`
 (`great_expectations.data_asset.data_asset.DataAsset` (`great_expectations.render.renderer.site_builder.SiteBuilder`
 method), 285 method), 603
`get_row_count()` (`great_expectations.dataset.Dataset`
 method), 467
`get_row_count()` (`great_expectations.dataset.dataset.Dataset`
 method), 285

module, 235
 great_expectations.cli.checkpoint_script_template
 module, 237
 great_expectations.cli.cli
 module, 238
 great_expectations.cli.cli_logging
 module, 238
 great_expectations.cli.cli_messages
 module, 239
 great_expectations.cli.datasource
 module, 240
 great_expectations.cli.docs
 module, 244
 great_expectations.cli.init
 module, 245
 great_expectations.cli.mark
 module, 245
 great_expectations.cli.project
 module, 246
 great_expectations.cli.store
 module, 246
 great_expectations.cli.suite
 module, 247
 great_expectations.cli.tap_template
 module, 248
 great_expectations.cli.toolkit
 module, 248
 great_expectations.cli.upgrade_helpers
 module, 233
 great_expectations.cli.upgrade_helpers.batch_upgrade_helpers
 module, 233
 great_expectations.cli.upgrade_helpers.upgrade_checkpoint_versions
 module, 234
 great_expectations.cli.util
 module, 251
 great_expectations.cli.validation_operator
 module, 252
 great_expectations.core
 module, 254
 great_expectations.core.batch
 module, 260
 great_expectations.core.data_context_keygreat_expectations.core.data_context_key
 module, 261
 great_expectations.core.evaluation_parameters
 module, 262
 great_expectations.core.id_dict
 module, 263
 great_expectations.core.metric
 module, 264
 great_expectations.core.urn
 module, 266
 great_expectations.core.usage_statisticsgreat_expectations.core.usage_statistics
 module, 254
 great_expectations.core.usage_statisticsgreat_expectations.core.usage_statistics
 module, 254
 great_expectations.core.usage_statistics.anonymizer
 module, 254
 great_expectations.core.usage_statistics.anonymizer
 module, 254
 great_expectations.core.usage_statistics.anonymizer
 module, 255
 great_expectations.core.usage_statistics.anonymizer
 module, 255
 great_expectations.core.usage_statistics.anonymizer
 module, 255
 great_expectations.core.usage_statistics.anonymizer
 module, 256
 great_expectations.core.usage_statistics.anonymizer
 module, 256
 great_expectations.core.usage_statistics.anonymizer
 module, 256
 great_expectations.core.usage_statistics.anonymizer
 module, 257
 great_expectations.core.usage_statistics.anonymizer
 module, 257
 great_expectations.core.usage_statistics.anonymizer
 module, 257
 great_expectations.core.usage_statistics.schemas
 module, 258
 great_expectations.core.usage_statistics.usage_statistics
 module, 258
 great_expectations.core.util
 module, 266
 great_expectations.core.utilgreat_expectations.core.util
 module, 283
 great_expectations.data_asset
 module, 283
 great_expectations.data_asset.data_asset
 module, 283
 great_expectations.data_asset.data_asset
 module, 289
 great_expectations.data_asset.util
 module, 294
 great_expectations.data_context
 module, 305
 great_expectations.data_context.data_context
 module, 338
 great_expectations.data_context.store
 module, 305
 great_expectations.data_context.store.database_store
 module, 305
 great_expectations.data_context.store.expectations
 module, 306
 great_expectations.data_context.store.html_site_store
 module, 306
 great_expectations.data_context.store.metric_store
 module, 307
 great_expectations.data_context.store.query_store
 module, 308
 great_expectations.data_context.store.store
 module, 308

module, 308	module, 562
great_expectations.data_context.store.stgreat_expectations.datasource.sparkdf_datasource	module, 564
module, 309	module, 564
great_expectations.data_context.store.tugreat_expectations.datasource.sqlalchemy_datasource	module, 565
module, 310	module, 565
great_expectations.data_context.store.validation_expectations.datasource.types	module, 554
module, 313	module, 554
great_expectations.data_context.templategreat_expectations.datasource.types.batch_kwargs	module, 554
module, 350	module, 557
great_expectations.data_context.typesgreat_expectations.datasource.types.reader_methods	module, 557
module, 320	module, 567
great_expectations.data_context.types.baggreat_expectations.datasource.util	module, 567
module, 320	module, 651
great_expectations.data_context.types.baggreat_expectations.exceptions	module, 651
module, 329	module, 574
great_expectations.data_context.types.reggreat_expectations.jupyter UX	module, 574
module, 329	module, 574
great_expectations.data_context.utilgreat_expectations.jupyter UX.expectation_explorer	module, 579
module, 351	module, 579
great_expectations.datasetgreat_expectations.profile	module, 580
module, 364	module, 580
great_expectations.dataset.datasetgreat_expectations.profile.base	module, 580
module, 364	module, 580
great_expectations.dataset.pandas_datasetgreat_expectations.profile.basic_dataset_profiler	module, 581
module, 410	module, 581
great_expectations.dataset.sparkdf_datasgreat_expectations.profile.basic_suite_builder_pro	module, 583
module, 432	module, 583
great_expectations.dataset.sqlalchemy_datgreat_expectations.profile.columns_exist	module, 584
module, 448	module, 584
great_expectations.dataset.utilgreat_expectations.profile.metrics_utils	module, 584
module, 462	module, 584
great_expectations.datasourcegreat_expectations.profile.multi_batch_validation_r	module, 584
module, 534	module, 586
great_expectations.datasource.batch_kwargsgreat_expectations.render	module, 586
module, 534	module, 618
great_expectations.datasource.batch_kwargsgreat_expectations.batch_kwargs.generator	module, 586
module, 534	module, 597
great_expectations.datasource.batch_kwargsgreat_expectations.batch_kwargs.generator	module, 597
module, 536	module, 597
great_expectations.datasource.batch_kwargsgreat_expectations.batch_kwargs.generator	module, 586
module, 536	module, 587
great_expectations.datasource.batch_kwargsgreat_expectations.batch_kwargs.generator	module, 587
module, 538	module, 588
great_expectations.datasource.batch_kwargsgreat_expectations.batch_kwargs.generator	module, 588
module, 539	module, 588
great_expectations.datasource.batch_kwargsgreat_expectations.batch_kwargs.generator	module, 588
module, 540	module, 588
great_expectations.datasource.batch_kwargsgreat_expectations.batch_kwargs.generator	module, 588
module, 542	module, 588
great_expectations.datasource.batch_kwargsgreat_expectations.batch_kwargs.generator	module, 588
module, 543	module, 588
great_expectations.datasource.datasourcegreat_expectations.render.renderer.content_block.ex	module, 588
module, 559	module, 588
great_expectations.datasource.pandas_datgreat_expectations.render.renderer.content_block.pr	

module, 591
 great_expectations.render.renderer.content_block_renderer (in module great_expectations.render.renderer), 592
 great_expectations.render.renderer.other_section_renderer (in module great_expectations.render.renderer), 599
 great_expectations.render.renderer.page_renderer (in module great_expectations.render.renderer), 599
 great_expectations.render.renderer.renderer (in module great_expectations.render.renderer), 600
 great_expectations.render.renderer.site_builder_renderer (in module great_expectations.render.renderer), 601
 great_expectations.render.renderer.site_index_renderer (in module great_expectations.render.renderer), 604
 great_expectations.render.renderer.slack_renderer (in module great_expectations.render.renderer), 605
 great_expectations.render.renderer.suite_edit_renderer (in module great_expectations.render.renderer), 605
 great_expectations.render.renderer.suite_scaffold_notebook_renderer (in module great_expectations.render.renderer), 606
 great_expectations.render.types (in module great_expectations.render), 610
 great_expectations.render.util (in module great_expectations.render), 618
 great_expectations.render.view (in module great_expectations.render), 613
 great_expectations.render.view.view (in module great_expectations.render.view), 613
 great_expectations.types (in module great_expectations), 620
 great_expectations.types.base (in module great_expectations.types), 620
 great_expectations.types.configurations (in module great_expectations.types), 621
 great_expectations.types.expectations (in module great_expectations.types), 623
 great_expectations.util (in module great_expectations), 654
 great_expectations.validation_operators (in module great_expectations), 623
 great_expectations.validation_operators.actions (in module great_expectations.validation_operators), 628
 great_expectations.validation_operators.types (in module great_expectations.validation_operators), 623
 great_expectations.validation_operators.types.validation_operator_result (in module great_expectations.validation_operators.types), 623
 great_expectations.validation_operators.util (in module great_expectations.validation_operators), 631
 great_expectations.validation_operators.validation_operators (in module great_expectations.validation_operators), 631
 great_expectations.validator (in module great_expectations.validator), 648
 great_expectations.validator.validator (in module great_expectations.validator), 648
 GreatExpectationsError, 651
 GreatExpectationsTypeError, 653
 GreatExpectationsValidationError, 651
 GREETING (in module great_expectations.cli.cli_messages), 239
 guess_reader_method_from_path() (great_expectations.datasource.pandas_datasource.PandasDatasource static method), 563
 guess_reader_method_from_path() (great_expectations.datasource.PandasDatasource static method), 571
 guess_reader_method_from_path() (great_expectations.datasource.sparkdf_datasource.SparkDFDatasource static method), 565
 guess_reader_method_from_path() (great_expectations.datasource.SparkDFDatasource static method), 572
H
 handle_error() (great_expectations.data_context.types.base.DataContext static method), 329
 HANDLERS (in module great_expectations._version), 649
 has_key() (great_expectations.data_context.store.Store static method), 317
 has_key() (great_expectations.data_context.store.store.Store static method), 309
 has_key() (great_expectations.data_context.store.store_backend.StoreBackend static method), 310
 has_key() (great_expectations.data_context.store.StoreBackend static method), 317
 hash_pandas_dataframe() (in module great_expectations.datasource.util), 567
 HASH_THRESHOLD (in module great_expectations.datasource.pandas_datasource), 562
 hashable_getters (great_expectations.dataset.Dataset attribute), 467
 hashable_getters (great_expectations.dataset.dataset.Dataset attribute), 365
 head() (great_expectations.dataset.sparkdf_dataset.SparkDFDataset static method), 433
 head() (great_expectations.dataset.sqlalchemy_dataset.SqlAlchemyDataset static method), 449
 HtmlSiteStore (class in great_expectations.data_context.store), 315
 HtmlSiteStore (class in great_expectations.data_context.store.html_site_store), 306
 IDDict (class in great_expectations.core), 269
 IDDict (class in great_expectations.core.id_dict), 263
 IGNORED_FILES (great_expectations.data_context.store.store_backend.StoreBackend attribute), 309

IGNORED_FILES (*great_expectations.data_context.store.StoreBackend* attribute), 317
 ignored_keys (*great_expectations.core.ExpectationKwargs* attribute), 272
 in_jupyter_notebook() (in module *great_expectations.core*), 270
 infer_distribution_parameters() (in module *great_expectations.dataset.util*), 465
 init() (in module *great_expectations.cli.init*), 245
 init_payload_schema (in module *great_expectations.core.usage_statistics.schemas*), 258
 initialize_data_asset_state() (*great_expectations.jupyter_ux.expectation_explorer.ExpectedDataAsset* method), 574
 InMemoryBatchKwargs (class in *great_expectations.datasource.types*), 558
 InMemoryBatchKwargs (class in *great_expectations.datasource.types.batch_kwargs*), 556
 InMemoryStoreBackend (class in *great_expectations.data_context.store*), 317
 InMemoryStoreBackend (class in *great_expectations.data_context.store.store_backend*), 310
 instance_id() (*great_expectations.data_context.BaseDataContext* property), 355
 instance_id() (*great_expectations.data_context.data_context.BaseDataContext* property), 341
 instantiate_class_from_config() (in module *great_expectations.data_context.util*), 351
 INT (*great_expectations.profile.base.ProfilerDataType* attribute), 579
 INT_TYPE_NAMES (*great_expectations.profile.basic_dataset_profiler.BasicDatasetProfilerBase* attribute), 581
 InvalidBatchIdError, 558, 652
 InvalidBatchKwargsError, 558, 652
 InvalidCacheValueError, 269, 653
 InvalidConfigError, 653
 InvalidConfigurationYamlError, 651
 InvalidConfigValueTypeError, 654
 InvalidDataContextConfigError, 652
 InvalidDataContextKeyError, 652
 InvalidExpectationConfigurationError, 269, 653
 InvalidExpectationKwargsError, 269, 653
 InvalidRenderedContentError, 610, 618
 InvalidTopLevelConfigKeyError, 652
 InvalidValidationResultError, 653
 is_ignored_key() (*great_expectations.data_context.store.store_backend.StoreBackend* method), 310
 is_ignored_key() (*great_expectations.data_context.store.StoreBackend* method), 318
 is_project_initialized() (*great_expectations.data_context.data_context.DataContext* class method), 349
 is_project_initialized() (*great_expectations.data_context.DataContext* class method), 363
 is_project_initialized() (*great_expectations.DataContext* class method), 662
 is_sane_slack_webhook() (in module *great_expectations.cli.util*), 252
 is_valid_categorical_partition_object() (in module *great_expectations.dataset.util*), 462
 is_valid_partition_object() (in module *great_expectations.dataset.util*), 462
 is_valid_partition_object() (in module *great_expectations.dataset.util*), 462
 isEquivalentTo() (*great_expectations.core.ExpectationConfiguration* method), 272
 isEquivalentTo() (*great_expectations.core.ExpectationKwargs* method), 272
 isEquivalentTo() (*great_expectations.core.ExpectationSuite* method), 274
 json_parse_exception (in module *great_expectations.cli.suite*), 247
 json_parse_exception (in module *great_expectations.cli.validation_operator*), 252

K

key() (*great_expectations.datasource.util.S3Url* property), 567
 key_to_tuple() (*great_expectations.data_context.store.Store* method), 317
 key_to_tuple() (*great_expectations.data_context.store.store.Store* method), 309
 KNOWN_EXTENSIONS (in module *great_expectations.datasource.batch_kwargs_generator.subdir_replacer*), 542
 known_extensions() (*great_expectations.datasource.batch_kwargs_generator.subdir_replacer* property), 543
 known_extensions() (*great_expectations.datasource.batch_kwargs_generator.SubdirReplacer* property), 543
 kwargs (*great_expectations.core.ExpectationConfigurationSchema* attribute), 274
 kwargs() (*great_expectations.core.ExpectationConfiguration* property), 272

kwargs_to_tuple() (in module *great_expectations.data_context.BaseDataContext*
great_expectations.profile.metrics_utils), method), 358
 584

L

launch_jupyter_notebook() (in module *great_expectations.validation_operators.types.validation_operator*
great_expectations.cli.toolkit), 250 method), 625

LETS_BEGIN_PROMPT (in module *great_expectations.data_context.data_context.BaseDataContext*
great_expectations.cli.cli_messages), 239 method), 344

limit() (*great_expectations.datasource.types.batch_kwargs.SqlAlchemyDataSourceBatchKwargs*
 property), 555 method), 357

limit() (*great_expectations.datasource.types.SqlAlchemyDataSourceBatchKwargs*
 property), 558 method), 343

lint_code() (in module *great_expectations.util*), 659

list_available_expectation_types() *list_keys()* (*great_expectations.data_context.store.database_store_backend.DatabaseStoreBackend*
great_expectations.data_asset.data_asset.DataAsset method), 306
 method), 283

list_available_expectation_types() *list_keys()* (*great_expectations.data_context.store.DatabaseStoreBackend*
great_expectations.data_asset.DataAsset method), 295 method), 315

list_available_expectations() *list_keys()* (*great_expectations.data_context.store.html_site_store.HtmlSiteStore*
great_expectations.render.renderer.content_block.content_block.ContentBlockRenderer method), 587
 class method), 587

list_batch_identifiers() *list_keys()* (*great_expectations.data_context.store.InMemoryStoreBackend*
great_expectations.validation_operators.types.validation_operator.ResultValidationOperatorResult method), 625
 method), 317

list_batch_kwargs_generators() *list_keys()* (*great_expectations.data_context.store.Store*
great_expectations.datasource.DataSource method), 569
 method), 309

list_batch_kwargs_generators() *list_keys()* (*great_expectations.data_context.store.store_backend.InMemoryStoreBackend*
great_expectations.datasource.datasource.DataSource method), 561
 method), 310

list_checkpoints() *list_keys()* (*great_expectations.data_context.store.StoreBackend*
great_expectations.data_context.data_context.DataContext method), 348
 method), 318

list_checkpoints() *list_keys()* (*great_expectations.data_context.store.tuple_store_backend.TupleStoreBackend*
great_expectations.data_context.DataContext method), 362
 method), 312

list_checkpoints() *list_keys()* (*great_expectations.data_context.store.tuple_store_backend.TupleStoreBackend*
great_expectations.DataContext method), 661
 method), 313

list_data_asset_names() *list_keys()* (*great_expectations.data_context.store.tuple_store_backend.TupleStoreBackend*
great_expectations.validation_operators.types.validation_operator.ResultValidationOperatorResult method), 625
 method), 319

list_data_assets_validated() *list_keys()* (*great_expectations.data_context.store.TupleS3StoreBackend*
great_expectations.validation_operators.types.validation_operator.ResultValidationOperatorResult method), 625
 method), 319

list_datasources() *list_stores()* (*great_expectations.data_context.BaseDataContext*
great_expectations.data_context.BaseDataContext method), 357
 method), 357

list_datasources() *list_stores()* (*great_expectations.data_context.data_context.BaseDataContext*
great_expectations.data_context.data_context.BaseDataContext method), 343
 method), 343

list_expectation_suite_names() *list_validation_operator_names()*
great_expectations.data_context.BaseDataContext method), 356
 method), 356

list_validation_operator_names()

`(great_expectations.data_context.data_context.BaseDataContext`
`method)`, 342
`list_validation_operators()`
`(great_expectations.data_context.BaseDataContext`
`method)`, 357
`list_validation_operators()`
`(great_expectations.data_context.data_context.BaseDataContext`
`method)`, 343
`list_validation_result_identifiers()`
`(great_expectations.validation_operators.types.validation_op`
`method)`, 625
`list_validation_results()`
`(great_expectations.validation_operators.types.validation_op`
`method)`, 625
`load_batch()` (`great_expectations.render.renderer.suite_scaffold` (in module `great_expectations.render.suite_scaffold`), 607
`method)`, 607
`load_batch()` (in module `great_expectations.cli.toolkit`), 250
`load_checkpoint()` (in module `great_expectations.cli.toolkit`), 251
`load_class()` (in module `great_expectations.data_context.util`), 351
`load_class()` (in module `great_expectations.util`), 655
`load_data_context_with_error_handling()`
`(in module great_expectations.cli.toolkit)`, 251
`load_expectation_suite()` (in module `great_expectations.cli.toolkit`), 250
`load_library()` (in module `great_expectations.cli.datasource`), 242
`logger` (in module `great_expectations.cli.cli_logging`), 238
`logger` (in module `great_expectations.cli.datasource`), 241
`logger` (in module `great_expectations.core`), 269
`logger` (in module `great_expectations.core.evaluation_parameters`), 262
`logger` (in module `great_expectations.core.usage_statistics_logger`), 254
`logger` (in module `great_expectations.core.usage_statistics_logger`), 259
`logger` (in module `great_expectations.data_asset.data_asset`), 283
`logger` (in module `great_expectations.data_context.data_context`), 338
`logger` (in module `great_expectations.data_context.store.database_backend`), 305
`logger` (in module `great_expectations.data_context.store.hive_store`), 306
`logger` (in module `great_expectations.data_context.store.query_store`), 308
`logger` (in module `great_expectations.data_context.store.store`), 308
`logger` (in module `great_expectations.data_context.store.tuple_store`), 308
`logger` (in module `great_expectations.data_context.types.base`), 321
`logger` (in module `great_expectations.data_context.types.resource_identifier`), 330
`logger` (in module `great_expectations.data_context.util`), 354
`logger` (in module `great_expectations.dataset`), 534
`logger` (in module `great_expectations.dataset.pandas_dataset`), 534
`logger` (in module `great_expectations.dataset.pandas_dataset`), 534
`logger` (in module `great_expectations.dataset.sparkdf_dataset`), 432
`logger` (in module `great_expectations.dataset.sparkdf_dataset`), 432
`logger` (in module `great_expectations.dataset.sqlalchemy_dataset`), 448
`logger` (in module `great_expectations.datasource.batch_kwargs_generator`), 534
`logger` (in module `great_expectations.datasource.batch_kwargs_generator`), 536
`logger` (in module `great_expectations.datasource.batch_kwargs_generator`), 536
`logger` (in module `great_expectations.datasource.batch_kwargs_generator`), 536
`logger` (in module `great_expectations.datasource.batch_kwargs_generator`), 538
`logger` (in module `great_expectations.datasource.batch_kwargs_generator`), 539
`logger` (in module `great_expectations.datasource.batch_kwargs_generator`), 540
`logger` (in module `great_expectations.datasource.batch_kwargs_generator`), 542
`logger` (in module `great_expectations.datasource.batch_kwargs_generator`), 544
`logger` (in module `great_expectations.datasource.datasource`), 559
`logger` (in module `great_expectations.datasource.pandas_datasource`), 562
`logger` (in module `great_expectations.datasource.sparkdf_datasource`), 564
`logger` (in module `great_expectations.datasource.sqlalchemy_datasource`), 565
`logger` (in module `great_expectations.datasource.types`), 558
`logger` (in module `great_expectations.datasource.types.batch_kwargs`), 555
`logger` (in module `great_expectations.jupyter_ux`), 579
`logger` (in module `great_expectations.jupyter_ux.expectation_explorer`), 579
`logger` (in module `great_expectations.profile.base`), 579
`logger` (in module `great_expectations.profile.basic_dataset_profiler`), 580
`logger` (in module `great_expectations.render.renderer.column_section_renderer`), 598
`logger` (in module `great_expectations.render.renderer.content_block.content_block`), 587
`logger` (in module `great_expectations.render.renderer.content_block.valid`), 587

[592](#) [\(great_expectations.data_context.types.base.AnonymizedUsageStatistics\)](#)
[logger](#) ([in module great_expectations.render.renderer.page_renderer](#)), [324](#)
[599](#) [make_validation_result_identifier\(\)](#)
[logger](#) ([in module great_expectations.render.renderer.site_builder](#)), ([great_expectations.data_context.types.resource_identifiers.ValidatedResourceIdentifier](#)), [337](#)
[601](#) [method](#), [337](#)
[logger](#) ([in module great_expectations.render.renderer.site_builder](#)), ([great_expectations.render.renderer.site_builder.RendererClasses](#) ([in module great_expectations.cli.datasource](#)), [241](#)
[604](#) [ManualBatchKwargsGenerator](#) ([class in great_expectations.datasource.batch_kwargs_generator](#)),
[logger](#) ([in module great_expectations.types.base](#)), [620](#) [ManualBatchKwargsGenerator](#) ([class in great_expectations.datasource.batch_kwargs_generator](#)),
[logger](#) ([in module great_expectations.util](#)), [655](#)
[logger](#) ([in module great_expectations.validation_operators](#)), [549](#)
[641](#) [ManualBatchKwargsGenerator](#) ([class in great_expectations.datasource.batch_kwargs_generator](#)),
[logger](#) ([in module great_expectations.validation_operators.actions](#)), [great_expectations.datasource.batch_kwargs_generator.manual_batch_kwargs_generator](#), [538](#)
[628](#) [ManualBatchKwargsGenerator](#), ([great_expectations.profile.base.ProfilerCardinality](#) attribute), [580](#)
[logger](#) ([in module great_expectations.validation_operators.actions](#)), ([great_expectations.profile.base.ProfilerCardinality](#) attribute), [580](#)
[631](#) [ManualBatchKwargsGenerator](#), ([great_expectations.profile.base.ProfilerCardinality](#) attribute), [580](#)
[logger](#) ([in module great_expectations.validation_operators.actions](#)), ([great_expectations.profile.base.ProfilerCardinality](#) attribute), [580](#)
[631](#) [ManualBatchKwargsGenerator](#), ([great_expectations.profile.base.ProfilerCardinality](#) attribute), [580](#)
[LONG_VERSION_PY](#) ([in module great_expectations._version](#)), [649](#)
[meta](#) ([great_expectations.core.ExpectationConfigurationSchema](#) attribute), [274](#)
[meta](#) ([great_expectations.core.ExpectationSuiteSchema](#) attribute), [278](#)
[meta](#) ([great_expectations.core.ExpectationSuiteValidationResultSchema](#) attribute), [282](#)
[main\(\)](#) ([in module great_expectations.cli](#)), [253](#)
[main\(\)](#) ([in module great_expectations.cli.cli](#)), [238](#)
[make_asset_configuration\(\)](#) ([great_expectations.datasource.batch_kwargs_generator.batch_kwargs_generator](#)), [546](#)
[make_batch_identifier\(\)](#) ([great_expectations.data_context.types.resource_identifiers.ValidatedResourceIdentifier](#)), [334](#)
[make_datasource_config\(\)](#) ([great_expectations.data_context.types.base.DataSourceConfigurationSchema](#)), [327](#)
[make_expectation_configuration\(\)](#) ([great_expectations.core.ExpectationConfigurationSchema](#)), [274](#)
[make_expectation_suite\(\)](#) ([great_expectations.core.ExpectationSuiteSchema](#)), [278](#)
[make_expectation_suite_identifier\(\)](#) ([great_expectations.data_context.types.resource_identifiers.ValidatedResourceIdentifier](#)), [332](#)
[make_expectation_suite_validation_result\(\)](#) ([great_expectations.core.ExpectationSuiteValidationResultSchema](#)), [282](#)
[make_expectation_suite_validation_result\(\)](#) ([great_expectations.validation_operators.types.validation_operator_result.ValidationOperatorResultSchema](#)), [627](#)
[make_expectation_validation_result\(\)](#) ([great_expectations.core.ExpectationValidationResultSchema](#)), [280](#)
[make_run_identifier\(\)](#) ([great_expectations.core.RunIdentifierSchema](#)), [272](#)
[make_usage_statistics_config\(\)](#) ([great_expectations.data_context.types.base.AnonymizedUsageStatistics](#)), [592](#)

M

great_expectations.core.metric), 265	great_expectations.core.urn , 266
MetricKwargs (class in great_expectations.core.id_dict), 264	great_expectations.core.usage_statistics , 254
MetricStore (class in great_expectations.data_context.store), 316	great_expectations.core.usage_statistics.anonymized , 254
MetricStore (class in great_expectations.data_context.store.metric_store), 307	great_expectations.core.usage_statistics.anonymized_batch , 254
MINIMUM_SUPPORTED_CONFIG_VERSION (in module great_expectations.data_context.types.base), 321	great_expectations.core.usage_statistics.anonymized_data_asset , 255
MissingConfigVariableError , 653	great_expectations.core.usage_statistics.anonymized_expectations , 255
MissingTopLevelConfigKeyError , 652	great_expectations.core.usage_statistics.anonymized_report , 255
module	great_expectations.core.usage_statistics.anonymized_report_batch , 255
great_expectations , 233	great_expectations.core.usage_statistics.anonymized_report_data_asset , 256
great_expectations._version , 648	great_expectations.core.usage_statistics.anonymized_report_expectations , 256
great_expectations.cli , 233	great_expectations.core.usage_statistics.anonymized_report_report , 257
great_expectations.cli.checkpoint , 235	great_expectations.core.usage_statistics.anonymized_report_report_batch , 257
great_expectations.cli.checkpoint_script_template , 237	great_expectations.core.usage_statistics.anonymized_report_report_data_asset , 258
great_expectations.cli.cli , 238	great_expectations.core.usage_statistics.anonymized_report_report_expectations , 258
great_expectations.cli.cli_logging , 238	great_expectations.core.util , 266
great_expectations.cli.cli_messages , 239	great_expectations.data_asset , 283
great_expectations.cli.datasource , 240	great_expectations.data_asset.data_asset , 283
great_expectations.cli.docs , 244	great_expectations.data_asset.file_data_asset , 289
great_expectations.cli.init , 245	great_expectations.data_asset.util , 294
great_expectations.cli.mark , 245	great_expectations.data_context , 305
great_expectations.cli.project , 246	great_expectations.data_context.data_context , 305
great_expectations.cli.store , 246	great_expectations.data_context.store , 305
great_expectations.cli.suite , 247	great_expectations.data_context.store.database_store , 305
great_expectations.cli.tap_template , 248	great_expectations.data_context.store.expectations_store , 306
great_expectations.cli.toolkit , 248	great_expectations.data_context.store.html_site_store , 306
great_expectations.cli.upgrade_helpers , 233	great_expectations.data_context.store.metric_store , 307
great_expectations.cli.upgrade_helpers.batch_upgrader_helpers , 233	great_expectations.data_context.store.query_store , 308
great_expectations.cli.upgrade_helpers.upgrader_helper_v11 , 234	
great_expectations.cli.util , 251	
great_expectations.cli.validation_operator , 252	
great_expectations.core , 254	
great_expectations.core.batch , 260	
great_expectations.core.data_context_keyframe , 261	
great_expectations.core.evaluation_parameters , 262	
great_expectations.core.id_dict , 263	
great_expectations.core.metric , 264	

[great_expectations.data_context.store.store](#), 564
[308](#) [great_expectations.datasource.sqlalchemy_datasource](#)
[great_expectations.data_context.store.store](#), 565
[309](#) [great_expectations.datasource.types](#),
[great_expectations.data_context.store.tuple](#), 554
[310](#) [store_backend](#),
[great_expectations.datasource.types.batch_kwargs](#),
[great_expectations.data_context.store.validations_store](#), 554
[313](#) [great_expectations.datasource.types.reader_method](#),
[great_expectations.data_context.templates](#), 557
[350](#) [great_expectations.datasource.util](#),
[great_expectations.data_context.types](#), 567
[320](#) [great_expectations.exceptions](#), 651
[great_expectations.data_context.types.base](#), 574
[320](#) [great_expectations.jupyter_ux](#), 574
[great_expectations.data_context.types.base](#), 574
[329](#) [great_expectations.jupyter_ux.expectation_explore](#),
[great_expectations.data_context.types.base](#), 574
[329](#) [source_identifiers](#),
[great_expectations.data_context.types.resource_identifiers](#), 579
[329](#) [great_expectations.profile](#), 579
[great_expectations.data_context.util](#), 580
[351](#) [great_expectations.profile.base](#), 579
[great_expectations.dataset](#), 364
[great_expectations.dataset.dataset](#), 581
[364](#) [great_expectations.profile.columns_exist](#),
[great_expectations.dataset.pandas_dataset](#), 583
[410](#) [great_expectations.profile.metrics_utils](#),
[great_expectations.dataset.sparkdf_dataset](#), 584
[432](#) [great_expectations.profile.multi_batch_validation](#),
[great_expectations.dataset.sqlalchemy_datasource](#), 586
[448](#) [great_expectations.render](#), 586
[great_expectations.dataset.util](#), 462
[great_expectations.datasource](#), 534
[great_expectations.datasource.batch_kwargs_generator](#), 586
[534](#) [great_expectations.render.exceptions](#),
[great_expectations.datasource.batch_kwargs_generator](#), 587
[534](#) [great_expectations.render.render](#), 587
[great_expectations.datasource.batch_kwargs_generator](#), 587
[536](#) [great_expectations.render.render.render_call_to_action](#),
[great_expectations.datasource.batch_kwargs_generator](#), 586
[536](#) [great_expectations.render.render.render_column_section](#),
[great_expectations.datasource.batch_kwargs_generator](#), 587
[538](#) [great_expectations.render.render.render_content_block](#),
[great_expectations.datasource.batch_kwargs_generator](#), 587
[539](#) [great_expectations.render.render.render_content_block](#),
[great_expectations.datasource.batch_kwargs_generator](#), 588
[540](#) [great_expectations.render.render.render_content_block](#),
[great_expectations.datasource.batch_kwargs_generator](#), 588
[542](#) [great_expectations.render.render.render_content_block](#),
[great_expectations.datasource.batch_kwargs_generator](#), 589
[543](#) [great_expectations.render.render.render_content_block](#),
[great_expectations.datasource.datasource](#), 592
[559](#) [great_expectations.render.render.render_other_section](#),
[great_expectations.datasource.pandas_datasource](#), 592
[562](#) [great_expectations.render.render.render_page_renderer](#),
[great_expectations.datasource.sparkdf_datasource](#), 599

[great_expectations.render.renderer.renderer](#), 244
[great_expectations.render.renderer.site_builder](#), 244
[great_expectations.render.renderer.site_indexer](#), 244
[great_expectations.render.renderer.slack_renderer](#), 244
[great_expectations.render.renderer.suite_editor](#), 244
[great_expectations.render.renderer.suite_scanner](#), 244
[great_expectations.render.types](#), 610
[great_expectations.render.util](#), 618
[great_expectations.render.view](#), 613
[great_expectations.render.view.view](#), 613
[great_expectations.types](#), 620
[great_expectations.types.base](#), 620
[great_expectations.types.configurations](#), 621
[great_expectations.types.expectations](#), 623
[great_expectations.util](#), 654
[great_expectations.validation_operators](#), 623
[great_expectations.validation_operators.actions](#), 628
[great_expectations.validation_operators.types](#), 623
[great_expectations.validation_operators.types.validation_operator_result](#), 623
[great_expectations.validation_operators.util](#), 631
[great_expectations.validation_operators.validation_operators](#), 631
[great_expectations.validator](#), 648
[great_expectations.validator.validator](#), 648
[module_name](#) ([great_expectations.data_context.types.base.DataSourceConfigSchema](#) (class in [great_expectations.validation_operators](#)), 326
[module_name](#) ([great_expectations.types.configurations.ClassConfigSchema](#) (class in [great_expectations.validation_operators](#)), 622
[module_name](#) ([great_expectations.data_context.types.base.DataSourceConfig](#) (class in [great_expectations.validation_operators.actions](#)), 322
[module_name](#) ([great_expectations.types.ClassConfig](#) (class in [great_expectations.render.view.view](#)), 623
[module_name](#) ([great_expectations.types.configurations.ClassConfig](#) (class in [great_expectations.render.view.view](#)), 621
[move](#) ([great_expectations.data_context.store.store_backend.StoreBackend](#) (class in [great_expectations.data_context.store.store_backend](#)), 310
[move](#) ([great_expectations.data_context.store.StoreBackend](#) (class in [great_expectations.data_context.store.store_backend](#)), 317
[msg_db_config](#) (in [great_expectations.cli.datasource](#)), 249

num_to_str() (in module great_expectations.render.util), 619

O

ONBOARDING_COMPLETE (in module great_expectations.cli.cli_messages), 239

ONE (great_expectations.profile.base.ProfilerCardinality attribute), 580

open_data_docs() (great_expectations.data_context.BaseDataContext method), 354

open_data_docs() (great_expectations.data_context.data_context.BaseDataContext method), 340

OperationalError (in module great_expectations.profile.basic_dataset_profiler), 580

opn (great_expectations.core.evaluation_parameters.EvaluationParameters attribute), 262

ordinal() (in module great_expectations.render.util), 619

OTHER (great_expectations.cli.datasource.SupportedDatabases attribute), 241

P

PANDAS (great_expectations.cli.datasource.DatasourceTypes attribute), 241

PandasDataset (class in great_expectations.dataset), 512

PandasDataset (class in great_expectations.dataset.pandas_dataset), 410

PandasDatasource (class in great_expectations.datasource), 570

PandasDatasource (class in great_expectations.datasource.pandas_datasource), 562

PandasDatasourceBatchKwargs (class in great_expectations.datasource.types), 558

PandasDatasourceBatchKwargs (class in great_expectations.datasource.types.batch_kwargs), 555

PandasDatasourceInMemoryBatchKwargs (class in great_expectations.datasource.types), 559

PandasDatasourceInMemoryBatchKwargs (class in great_expectations.datasource.types.batch_kwargs), 556

parse_evaluation_parameter() (in module great_expectations.core.evaluation_parameters), 263

parse_result_format() (in module great_expectations.data_asset.util), 294

ParserError, 269, 651

partition_data() (in module great_expectations.dataset.util), 463

path() (great_expectations.datasource.types.batch_kwargs.PathBatchKwargs property), 556

path() (great_expectations.datasource.types.PathBatchKwargs property), 558

PathBatchKwargs (class in great_expectations.datasource.types), 558

PathBatchKwargs (class in great_expectations.datasource.types.batch_kwargs), 555

PluginClassNotFoundError, 654

PluginModuleNotFoundError, 654

plugins_directory (great_expectations.data_context.types.base.DataContextConfig attribute), 328

plugins_directory() (great_expectations.data_context.BaseDataContext property), 354

plugins_directory() (great_expectations.data_context.data_context.BaseDataContext property), 341

plus_or_dot() (in module great_expectations._version), 650

POSTGRES (great_expectations.cli.datasource.SupportedDatabases attribute), 241

prec (in module great_expectations.render.util), 619

prepare_dump() (great_expectations.core.ExpectationSuiteSchema method), 278

prepare_dump() (great_expectations.core.ExpectationSuiteValidationR method), 282

prepare_dump() (great_expectations.validation_operators.types.valida method), 627

PrettyPrintTemplate (class in great_expectations.render.view.view), 614

process_batch_parameters() (great_expectations.datasource.Datasource method), 569

process_batch_parameters() (great_expectations.datasource.datasource.Datasource method), 561

process_batch_parameters() (great_expectations.datasource.pandas_datasource.PandasDatas method), 563

process_batch_parameters() (great_expectations.datasource.PandasDatasource method), 571

process_batch_parameters() (great_expectations.datasource.sparkdf_datasource.SparkDFData method), 564

process_batch_parameters() (great_expectations.datasource.SparkDFDatasource method), 572

process_batch_parameters() (great_expectations.datasource.sqlalchemy_datasource.SqlAlchem method), 566

[process_batch_parameters\(\)](#) [596](#)
 ([great_expectations.datasource.SqlAlchemyDataSource](#) [method](#)), [573](#) [ProfilingOverviewTableContentBlockRenderer](#)
 ([class in great_expectations.render.renderer.content_block.profiling](#))
[profile\(\)](#) ([great_expectations.data_asset.data_asset.DataAsset](#) [method](#)), [283](#) [591](#)
[profile\(\)](#) ([great_expectations.data_asset.DataAsset](#) [method](#)), [295](#) [ProfilingResultsColumnSectionRenderer](#)
 ([class in great_expectations.jupyter UX](#)), [577](#)
[profile\(\)](#) ([great_expectations.profile.base.DatasetProfiler](#) [class method](#)), [580](#) [ProfilingResultsColumnSectionRenderer](#)
 ([class in great_expectations.render.renderer](#)), [608](#)
[profile_data_asset\(\)](#) [ProfilingResultsColumnSectionRenderer](#)
 ([great_expectations.data_context.BaseDataContext](#) [method](#)), [360](#) ([class in great_expectations.render.renderer.column_section_renderer](#)) [598](#)
[profile_data_asset\(\)](#) [ProfilingResultsOverviewSectionRenderer](#)
 ([great_expectations.data_context.data_context.BaseDataContext](#) [method](#)), [346](#) ([class in great_expectations.render.renderer](#)), [608](#)
[profile_datasource\(\)](#) [ProfilingResultsOverviewSectionRenderer](#)
 ([great_expectations.data_context.BaseDataContext](#) [method](#)), [359](#) ([class in great_expectations.render.renderer.other_section_renderer](#)) [599](#)
[profile_datasource\(\)](#) [ProfilingResultsPageRenderer](#) ([class in](#)
 ([great_expectations.data_context.data_context.BaseDataContext](#) [method](#)), [345](#) [great_expectations.render.renderer](#)), [609](#)
[profile_datasource\(\)](#) ([in module](#) [great_expectations.render.renderer.page_renderer](#)), [600](#)
 ([great_expectations.cli.datasource](#)), [244](#)
[ProfilerCardinality](#) ([class in project\(\)](#) ([in module great_expectations.cli.project](#)), [246](#)
 ([great_expectations.profile.base](#)), [580](#)
[ProfilerDataType](#) ([class in project_check_config\(\)](#) ([in module](#)
 ([great_expectations.profile.base](#)), [579](#) [great_expectations.cli.project](#)), [246](#)
[ProfilerError](#), [653](#) [PROJECT_HELP_COMMENT](#) ([in module](#)
[PROFILING_ERROR_CODE_MULTIPLE_BATCH_KWARGS_GENERATORS_FOUND](#) ([great_expectations.data_context.templates](#)), [350](#)
 ([great_expectations.data_context.BaseDataContext](#) [attribute](#)), [353](#) [PROJECT_IS_COMPLETE](#) ([in module](#)
[PROFILING_ERROR_CODE_MULTIPLE_BATCH_KWARGS_GENERATORS_FOUND](#) ([great_expectations.cli.cli_messages](#)), [239](#)
 ([great_expectations.data_context.data_context.BaseDataContext](#) [attribute](#)), [339](#) [PROFILING_ERROR_CODE_UNCONDITIONAL_CONFIG_COMMENT](#) ([in module](#)
 ([great_expectations.data_context.templates](#)), [351](#)
[PROFILING_ERROR_CODE_NO_BATCH_KWARGS_GENERATORS_FOUND](#) [351](#)
 ([great_expectations.data_context.BaseDataContext](#) [attribute](#)), [353](#) [PROJECT_TEMPLATE_USAGE_STATISTICS_DISABLED](#)
 ([in module great_expectations.data_context.templates](#)), [351](#)
[PROFILING_ERROR_CODE_NO_BATCH_KWARGS_GENERATORS_FOUND](#) [351](#)
 ([great_expectations.data_context.data_context.BaseDataContext](#) [attribute](#)), [339](#) [PROJECT_TEMPLATE_USAGE_STATISTICS_ENABLED](#)
 ([in module great_expectations.data_context.templates](#)), [351](#)
[PROFILING_ERROR_CODE_SPECIFIED_DATA_ASSETS_NOT_FOUND](#) [351](#)
 ([great_expectations.data_context.BaseDataContext](#) [attribute](#)), [353](#) [project_upgrade\(\)](#) ([in module](#)
 ([great_expectations.cli.project](#)), [246](#)
[PROFILING_ERROR_CODE_SPECIFIED_DATA_ASSETS_NOT_FOUND](#) ([great_expectations.core.evaluation_parameters.EvaluationParameters](#)) [262](#)
 ([great_expectations.data_context.data_context.BaseDataContext](#) [method](#)), [262](#)
 ([great_expectations.data_context.data_context.BaseDataContext](#) [attribute](#)), [339](#) [push_unary_minus\(\)](#)
 ([great_expectations.core.evaluation_parameters.EvaluationParameters](#) [method](#)), [262](#)
 ([great_expectations.data_context.BaseDataContext](#) [attribute](#)), [353](#)
[PROFILING_ERROR_CODE_TOO_MANY_DATA_ASSETS](#) [Q](#)
 ([great_expectations.data_context.data_context.BaseDataContext](#) [attribute](#)), [339](#) [query\(\)](#) ([great_expectations.datasource.types.batch_kwargs.SparkDFDataSource](#)
 ([property](#)), [557](#)
[ProfilingOverviewTableContentBlockRenderer](#) [query\(\)](#) ([great_expectations.datasource.types.batch_kwargs.SqlAlchemyDataSource](#)
 ([class in great_expectations.render.renderer.content_block](#)), [property](#)), [556](#)

[query\(\) \(great_expectations.datasource.types.SparkDFDatasource.QueryBatchKwargsGenerator property\), 559](#)
[query\(\) \(great_expectations.datasource.types.SqlAlchemyDatasource.QueryBatchKwargsGenerator property\), 559](#)
[query_parameters\(\) \(great_expectations.datasource.types.batch_kwargs_generator.SqlAlchemyDatasource.QueryBatchKwargsGenerator property\), 556](#)
[query_parameters\(\) \(great_expectations.datasource.types.SqlAlchemyDatasource.QueryBatchKwargsGenerator property\), 559](#)
[QueryBatchKwargsGenerator \(class in great_expectations.datasource.batch_kwargs_generator\), 550](#)
[QueryBatchKwargsGenerator \(class in great_expectations.datasource.batch_kwargs_generator\), 539](#)
R
[read_csv\(\) \(in module great_expectations.util\), 656](#)
[read_excel\(\) \(in module great_expectations.util\), 657](#)
[read_json\(\) \(in module great_expectations.util\), 656](#)
[read_parquet\(\) \(in module great_expectations.util\), 658](#)
[read_pickle\(\) \(in module great_expectations.util\), 658](#)
[read_table\(\) \(in module great_expectations.util\), 657](#)
[reader_method\(\) \(great_expectations.datasource.batch_kwargs_generator.GlobReaderBatchKwargsGenerator property\), 537](#)
[reader_method\(\) \(great_expectations.datasource.batch_kwargs_generator.GlobReaderBatchKwargsGenerator property\), 549](#)
[reader_method\(\) \(great_expectations.datasource.batch_kwargs_generator.SubdirReaderBatchKwargsGenerator property\), 543](#)
[reader_method\(\) \(great_expectations.datasource.batch_kwargs_generator.SubdirReaderBatchKwargsGenerator property\), 553](#)
[reader_method\(\) \(great_expectations.datasource.types.batch_kwargs_generator.PathBatchKwargsGenerator property\), 556](#)
[reader_method\(\) \(great_expectations.datasource.types.batch_kwargs_generator.S3BatchKwargsGenerator property\), 556](#)
[reader_method\(\) \(great_expectations.datasource.types.PathBatchKwargsGenerator property\), 558](#)
[reader_method\(\) \(great_expectations.datasource.types.S3BatchKwargsGenerator property\), 558](#)
[reader_options\(\) \(great_expectations.datasource.batch_kwargs_generator.GlobReaderBatchKwargsGenerator property\), 537](#)
[reader_options\(\) \(great_expectations.datasource.batch_kwargs_generator.GlobReaderBatchKwargsGenerator property\), 549](#)
[reader_options\(\) \(great_expectations.datasource.batch_kwargs_generator.GlobReaderBatchKwargsGenerator property\), 541](#)
[reader_options\(\) \(great_expectations.datasource.batch_kwargs_generator.S3GlobReaderBatchKwargsGenerator property\), 552](#)
[reader_options\(\) \(great_expectations.datasource.batch_kwargs_generator.SubdirReaderBatchKwargsGenerator property\), 543](#)

`(great_expectations.datasource.sparkdf_datasource.SparkDFDatasource.render_attribute), 564`
`recognized_batch_parameters` (`great_expectations.datasource.SparkDFDatasource.render_attribute`), 572
`recognized_batch_parameters` (`great_expectations.datasource.sqlalchemy_datasource.SqlAlchemyDatasource.render_attribute`), 566
`recognized_batch_parameters` (`great_expectations.datasource.SqlAlchemyDatasource.render_attribute`), 573
`recursively_convert_to_json_serializable` (`in module great_expectations.data_asset.util`), 294
`REDSHIFT` (`great_expectations.cli.datasource.SupportedDatabases.render_attribute`), 241
`register_vcs_handler()` (`in module great_expectations._version`), 649
`remove_expectation()` (`great_expectations.core.ExpectationSuite.render_method`), 276
`remove_expectation()` (`great_expectations.data_asset.data_asset.DataAsset.render_method`), 285
`remove_expectation()` (`great_expectations.data_asset.DataAsset.render_method`), 296
`remove_key()` (`great_expectations.data_context.store.database_store.DatabaseStoreBackend.render_method`), 306
`remove_key()` (`great_expectations.data_context.store.DatabaseStoreBackend.render_method`), 315
`remove_key()` (`great_expectations.data_context.store.expectations_store.ExpectationsStore.render_method`), 306
`remove_key()` (`great_expectations.data_context.store.ExpectationsStore.render_method`), 315
`remove_key()` (`great_expectations.data_context.store.InMemoryStoreBackend.render_method`), 317
`remove_key()` (`great_expectations.data_context.store.store_checkpoint.InMemoryStoreBackend.render_method`), 310
`remove_key()` (`great_expectations.data_context.store.store_checkpoint.SmallBackend.render_method`), 310
`remove_key()` (`great_expectations.data_context.store.StoreBackend.render_method`), 318
`remove_key()` (`great_expectations.data_context.store.tuple_store.HadoopTupleFilesystemStoreBackend.render_method`), 312
`remove_key()` (`great_expectations.data_context.store.tuple_store.HadoopTupleFilesystemStoreBackend.render_method`), 313
`remove_key()` (`great_expectations.data_context.store.tuple_store.HadoopTupleFilesystemStoreBackend.render_method`), 312
`remove_key()` (`great_expectations.data_context.store.TupleFilesystemStoreBackend.render_method`), 318
`remove_key()` (`great_expectations.data_context.store.TupleGCSStoreBackend.render_method`), 319
`remove_key()` (`great_expectations.data_context.store.TupleS3StoreBackend.render_method`), 319

run() (great_expectations.validation_operators.ActionListValidationOperator (class in great_expectations.core), method), 644

run() (great_expectations.validation_operators.actions.ValidationActionIdentifier (class in great_expectations.core), method), 628

run() (great_expectations.validation_operators.validation_operators.ActionListValidationOperator (class in great_expectations.core), method), 634

run() (great_expectations.validation_operators.validation_operators.ValidationOperator (in module great_expectations.core), method), 632

run() (great_expectations.validation_operators.validation_operators.WarningAndFailureExpectationSuitesValidationOperator (in module great_expectations.core), method), 638

run() (great_expectations.validation_operators.ValidationAction (in module great_expectations.core), method), 641

run() (great_expectations.validation_operators.ValidationOperator (in module great_expectations.core), method), 645

run() (great_expectations.validation_operators.WarningAndFailureExpectationSuitesValidationOperator (in module great_expectations.core), method), 648

run_command() (in module great_expectations, property), 649

run_id(great_expectations.data_context.types.resource_identifiers.ValidationResultIdentifierSchema (in module great_expectations.core), attribute), 337

run_id(great_expectations.validation_operators.types.validation_operator_result_validation_operator_result_schema (in module great_expectations.core), attribute), 627

run_id() (great_expectations.core.metric.ValidationMetric (in module great_expectations.core), property), 265

run_id() (great_expectations.core.metric.ValidationMetricIdentifier (in module great_expectations.core), property), 266

run_id() (great_expectations.data_context.types.resource_identifiers.ValidationResultIdentifier (in module great_expectations.core), property), 335

run_id() (great_expectations.validation_operators.types.validation_operator_result_validation_operator_result_schema (in module great_expectations.core), property), 624

RUN_INIT_AGAIN (in module great_expectations.cli, property), 239

run_name(great_expectations.core.RunIdentifierSchema (in module great_expectations.core), attribute), 271

run_name() (great_expectations.core.RunIdentifier (in module great_expectations.core), property), 270

run_results(great_expectations.validation_operators.types.validation_operator_result_validation_operator_result_schema (in module great_expectations.core), attribute), 627

run_results() (great_expectations.validation_operators.types.validation_operator_result_validation_operator_result_schema (in module great_expectations.core), property), 624

run_time(great_expectations.core.RunIdentifierSchema (in module great_expectations.core), attribute), 272

run_time() (great_expectations.core.RunIdentifier (in module great_expectations.core), property), 270

run_validation_operator() (great_expectations.data_context.BaseDataContext (in module great_expectations.core), method), 356

run_validation_operator() (great_expectations.data_context.data_context.BaseDataContext (in module great_expectations.core), method), 342

run_validation_operator_payload_schema (in module great_expectations.core, property), 258

run_validation_operator_usage_statistics() (in module great_expectations.core, property), 259

s3() (great_expectations.datasource.types.batch_kwargs.S3BatchKwargs (class in great_expectations.datasource.batch_kwargs_generator), property), 558

s3() (great_expectations.datasource.types.S3BatchKwargs (class in great_expectations.datasource.batch_kwargs_generator), property), 558

s3_batch_kwargs_generator (class in great_expectations.datasource.batch_kwargs_generator), 540

salt() (great_expectations.core.usage_statistics.anonymizers.anonymizer (in module great_expectations.core), property), 254

save_config_variable() (great_expectations.data_context.BaseDataContext (in module great_expectations.core), method), 355

save_config_variable() (great_expectations.data_context.data_context.BaseDataContext (in module great_expectations.core), method), 344

save_expectation_suite() (great_expectations.data_context.BaseDataContext (in module great_expectations.core), method), 358

save_expectation_suite() (great_expectations.data_context.data_context.BaseDataContext (in module great_expectations.core), method), 344

save_expectation_suite_usage_statistics() (in module great_expectations.core, property), 259

save_or_edit_expectation_suite_payload_schema (in module great_expectations.core, property), 258

258

scaffold_custom_data_docs() (great_expectations.data_context.data_context.DataContext class method), 348

scaffold_custom_data_docs() (great_expectations.data_context.DataContext class method), 362

scaffold_custom_data_docs() (great_expectations.DataContext class method), 661

scaffold_directories() (great_expectations.data_context.data_context.DataContext class method), 348

scaffold_directories() (great_expectations.data_context.DataContext class method), 362

scaffold_directories() (great_expectations.DataContext class method), 661

scaffold_notebooks() (great_expectations.data_context.data_context.DataContext class method), 348

scaffold_notebooks() (great_expectations.data_context.DataContext class method), 362

scaffold_notebooks() (great_expectations.DataContext class method), 661

schema (great_expectations.datasource.batch_kwargs_generator attribute), 546

schema() (great_expectations.datasource.batch_kwargs_generator.AssetConfiguration property), 546

schema() (great_expectations.datasource.types.batch_kwargs.SqlAlchemyDataSourceBatchKwargs property), 555

schema() (great_expectations.datasource.types.SqlAlchemyDataSourceBatchKwargs property), 558

SECTION_SEPARATOR (in module great_expectations.cli.cli_messages), 240

select_batch_kwargs_generator() (in module great_expectations.cli.datasource), 243

select_datasource() (in module great_expectations.cli.toolkit), 251

send_slack_notification() (in module great_expectations.validation_operators), 641

send_slack_notification() (in module great_expectations.validation_operators.util), 631

send_usage_message() (great_expectations.core.usage_statistics.usage_statistics.UsageStatisticsHandler method), 259

send_usage_message() (in module great_expectations.core.usage_statistics.usage_statistics), 260

serialize() (great_expectations.data_context.store.expectations_store.ExpectationsStore method), 306

serialize() (great_expectations.data_context.store.ExpectationsStore method), 315

serialize() (great_expectations.data_context.store.metric_store.MetricStore method), 307

serialize() (great_expectations.data_context.store.MetricStore method), 316

serialize() (great_expectations.data_context.store.Store method), 317

serialize() (great_expectations.data_context.store.store.Store method), 309

serialize() (great_expectations.data_context.store.validations_store.ValidationsStore method), 314

serialize() (great_expectations.data_context.store.ValidationsStore method), 320

set() (great_expectations.data_context.store.html_site_store.HtmlSiteStore method), 307

set() (great_expectations.data_context.store.HtmlSiteStore method), 316

set() (great_expectations.data_context.store.query_store.SqlAlchemyQueryStore method), 308

set() (great_expectations.data_context.store.Store method), 317

set() (great_expectations.data_context.store.store.Store method), 309

set() (great_expectations.data_context.store.store_backend.StoreBackend method), 310

set_config_value() (great_expectations.data_context.store.store_backend.StoreBackend method), 317

set_config_value() (great_expectations.data_asset.data_asset.DataAsset method), 285

set_config_value() (great_expectations.data_asset.DataAsset method), 296

set_data_source() (in module great_expectations.jupyter_ux), 577

set_default_expectation_argument() (great_expectations.data_asset.data_asset.DataAsset method), 285

set_default_expectation_argument() (great_expectations.data_asset.DataAsset method), 297

set_evaluation_parameter() (great_expectations.data_asset.data_asset.DataAsset method), 288

set_evaluation_parameter() (great_expectations.data_asset.DataAsset method), 308

set_expectation_state() (great_expectations.jupyter_ux.expectation_explorer.ExpectationExplorer method), 574

set_ge_config_version()

<code>(great_expectations.data_context.data_context.DataContext class method), 349</code>	<code>SNOWFLAKE (great_expectations.cli.datasource.SupportedDatabases attribute), 241</code>
<code>set_ge_config_version()</code> <code>(great_expectations.data_context.DataContext class method), 363</code>	<code>snowflake (in module great_expectations.dataset.sqlalchemy_dataset), 448</code>
<code>set_ge_config_version()</code> <code>(great_expectations.DataContext class method), 662</code>	<code>SPARK (great_expectations.cli.datasource.DatasourceTypes attribute), 241</code>
<code>setup_notebook_logging()</code> (in module <code>great_expectations.jupyter_ux</code>), 578	<code>SparkDFDataset (class in great_expectations.dataset.sparkdf_dataset), 433</code>
<code>SETUP_SUCCESS</code> (in module <code>great_expectations.cli.cli_messages</code>), 240	<code>SparkDFDatasource (class in great_expectations.datasource), 572</code>
<code>show_available_data_asset_names()</code> (in module <code>great_expectations.jupyter_ux</code>), 578	<code>SparkDFDatasource (class in great_expectations.datasource.sparkdf_datasource), 564</code>
<code>site_section_name()</code> <code>(great_expectations.data_context.types.resource_identifiers.SiteSectionIdentifier property), 338</code>	<code>SparkDFDatasourceBatchKwargs (class in great_expectations.datasource.types), 558</code>
<code>SiteBuilder</code> (class in <code>great_expectations.render.renderers.site_builder</code>), 601	<code>SparkDFDatasourceBatchKwargs (class in great_expectations.datasource.types.batch_kwargs), 555</code>
<code>SiteBuilderAnonymizer</code> (class in <code>great_expectations.core.usage_statistics.anonymizers.site_builder_anonymizer</code>), 256	<code>SparkDFDatasourceInMemoryBatchKwargs (class in great_expectations.datasource.types), 559</code>
<code>SiteIndexPageRenderer</code> (class in <code>great_expectations.render.renderer</code>), 609	<code>SparkDFDatasourceInMemoryBatchKwargs (class in great_expectations.datasource.types.batch_kwargs), 556</code>
<code>SiteIndexPageRenderer</code> (class in <code>great_expectations.render.renderers.site_index_page_renderer</code>), 604	<code>SparkDFDatasourceQueryBatchKwargs (class in great_expectations.datasource.types), 559</code>
<code>SiteSectionIdentifier</code> (class in <code>great_expectations.data_context.types.resource_identifiers</code>), 337	<code>SparkDFDatasourceQueryBatchKwargs (class in great_expectations.datasource.types.batch_kwargs), 556</code>
<code>SLACK_DOC_LINK</code> (in module <code>great_expectations.cli.cli_messages</code>), 239	<code>SparkSession (in module great_expectations.datasource.sparkdf_datasource), 564</code>
<code>SLACK_LATER</code> (in module <code>great_expectations.cli.cli_messages</code>), 239	<code>SQL (great_expectations.cli.datasource.DatasourceTypes attribute), 241</code>
<code>SLACK_SETUP_COMPLETE</code> (in module <code>great_expectations.cli.cli_messages</code>), 239	<code>sql_engine_dialect()</code> <code>(great_expectations.dataset.sqlalchemy_dataset.SqlAlchemyDataset property), 449</code>
<code>SLACK_SETUP_INTRO</code> (in module <code>great_expectations.cli.cli_messages</code>), 239	<code>sqlalchemy (in module great_expectations.data_context.store.database_store_backend), 305</code>
<code>SLACK_SETUP_PROMPT</code> (in module <code>great_expectations.cli.cli_messages</code>), 239	<code>sqlalchemy (in module great_expectations.data_context.store.query_store), 308</code>
<code>SLACK_WEBHOOK_PROMPT</code> (in module <code>great_expectations.cli.cli_messages</code>), 239	<code>sqlalchemy (in module great_expectations.datasource.batch_kwargs_generator.query_batch_kwargs_generator), 539</code>
<code>SlackNotificationAction</code> (class in <code>great_expectations.validation_operators</code>), 639	<code>sqlalchemy (in module great_expectations.datasource.batch_kwargs_generator.table_batch_kwargs_generator), 544</code>
<code>SlackNotificationAction</code> (class in <code>great_expectations.validation_operators.actions</code>), 629	<code>sqlalchemy (in module great_expectations.datasource.sqlalchemy_datasource), 565</code>
<code>SlackRenderer</code> (class in <code>great_expectations.render.renderer</code>), 609	
<code>SlackRenderer</code> (class in <code>great_expectations.render.renderers.slack_renderer</code>), 605	

sqlalchemy_psycopg2	(in module <i>great_expectations.dataset.sqlalchemy_dataset</i>), 448	Store (class in <i>great_expectations.data_context.store</i>), 316
sqlalchemy_redshift	(in module <i>great_expectations.dataset.sqlalchemy_dataset</i>), 448	Store (class in <i>great_expectations.data_context.store.store</i>), 308
SqlAlchemyBatchReference	(class in <i>great_expectations.dataset.sqlalchemy_dataset</i>), 448	store () (in module <i>great_expectations.cli.store</i>), 246
SqlAlchemyDataset	(class in <i>great_expectations.dataset.sqlalchemy_dataset</i>), 449	store_backend () (<i>great_expectations.data_context.store.Store</i> property), 316
SqlAlchemyDatasource	(class in <i>great_expectations.datasource</i>), 573	store_backend () (<i>great_expectations.data_context.store.store.Store</i> property), 309
SqlAlchemyDatasource	(class in <i>great_expectations.datasource.sqlalchemy_datasource</i>), 565	store_evaluation_parameters () (<i>great_expectations.data_context.BaseDataContext</i> method), 358
SqlAlchemyDatasourceBatchKwargs	(class in <i>great_expectations.datasource.types</i>), 558	store_evaluation_parameters () (<i>great_expectations.data_context.data_context.BaseDataContext</i> method), 344
SqlAlchemyDatasourceBatchKwargs	(class in <i>great_expectations.datasource.types.batch_kwargs</i>), 555	store_list () (in module <i>great_expectations.cli.store</i>), 246
SqlAlchemyDatasourceQueryBatchKwargs	(class in <i>great_expectations.datasource.types</i>), 559	store_validation_result_metrics () (<i>great_expectations.data_context.BaseDataContext</i> method), 358
SqlAlchemyDatasourceQueryBatchKwargs	(class in <i>great_expectations.datasource.types.batch_kwargs</i>), 556	store_validation_result_metrics () (<i>great_expectations.data_context.data_context.BaseDataContext</i> method), 344
SqlAlchemyDatasourceTableBatchKwargs	(class in <i>great_expectations.datasource.types</i>), 559	StoreAnonymizer (class in <i>great_expectations.core.usage_statistics.anonymizers.store_anonymizer</i>), 257
SqlAlchemyDatasourceTableBatchKwargs	(class in <i>great_expectations.datasource.types.batch_kwargs</i>), 556	StoreBackend (class in <i>great_expectations.data_context.store</i>), 317
SqlAlchemyDatasourceTableBatchKwargs	(class in <i>great_expectations.datasource.types.batch_kwargs</i>), 556	StoreBackend (class in <i>great_expectations.data_context.store.store_backend</i>), 309
SQLAlchemyError	(in module <i>great_expectations.cli.checkpoint</i>), 236	StoreBackendAnonymizer (class in <i>great_expectations.core.usage_statistics.anonymizers.store_backend_anonymizer</i>), 257
SQLAlchemyError	(in module <i>great_expectations.cli.init</i>), 245	StoreBackendError, 651
SQLAlchemyError	(in module <i>great_expectations.cli.suite</i>), 247	StoreConfigurationError, 653
SQLAlchemyError	(in module <i>great_expectations.cli.validation_operator</i>), 252	StoreError, 653
SQLAlchemyError	(in module <i>great_expectations.data_context.data_context</i>), 338	StoreEvaluationParametersAction (class in <i>great_expectations.validation_operators</i>), 639
SqlAlchemyQueryStore	(class in <i>great_expectations.data_context.store.query_store</i>), 308	StoreEvaluationParametersAction (class in <i>great_expectations.validation_operators.actions</i>), 629
statistics	(<i>great_expectations.core.ExpectationSuiteValidationResult</i> attribute), 282	StoreMetricsAction (class in <i>great_expectations.validation_operators</i>), 640
STOP_SIGNAL	(in module <i>great_expectations.core.usage_statistics.usage_statistics</i>), 259	StoreMetricsAction (class in <i>great_expectations.validation_operators.actions</i>), 630
		stores (<i>great_expectations.data_context.types.base.DataContextConfiguration</i> attribute), 282
		stores () (<i>great_expectations.data_context.BaseDataContext</i> property), 355
		stores () (<i>great_expectations.data_context.data_context.BaseDataContext</i> property), 341

StoreValidationResultAction (class in [great_expectations.validation_operators](#)), [640](#)
 StoreValidationResultAction (class in [great_expectations.validation_operators.actions](#)), [629](#)
 STRING ([great_expectations.profile.base.ProfilerDataType](#) attribute), [579](#)
 STRING_TYPE_NAMES ([great_expectations.profile.basic_dataset_profiler.BasicDatasetProfilerBase](#) attribute), [581](#)
 StringKey (class in [great_expectations.core.data_context_key](#)), [261](#)
 SubdirReaderBatchKwargsGenerator (class in [great_expectations.datasource.batch_kwargs_generator](#)), [552](#)
 SubdirReaderBatchKwargsGenerator (class in [great_expectations.datasource.batch_kwargs_generator.subdir_reader_batch_kwargs_generator](#)), [542](#)
 substitute_all_config_variables() (in module [great_expectations.data_context.util](#)), [352](#)
 substitute_config_variable() (in module [great_expectations.data_context.util](#)), [351](#)
 substitute_none_for_missing() (in module [great_expectations.render.renderer.content_block.expectation_string](#)), [588](#)
 success ([great_expectations.core.ExpectationSuiteValidationResultSchema](#) attribute), [282](#)
 success ([great_expectations.core.ExpectationValidationResultSchema](#) attribute), [279](#)
 success ([great_expectations.validation_operators.types.validation_operator_result.ValidationOperatorResultSchema](#) attribute), [627](#)
 success() ([great_expectations.validation_operators.types.validation_operator_result.ValidationOperatorResult](#) property), [625](#)
 SUFFIXES (in module [great_expectations.render.util](#)), [619](#)
 suite (in module [great_expectations.cli.tap_template](#)), [248](#)
 suite() (in module [great_expectations.cli.suite](#)), [247](#)
 suite_delete() (in module [great_expectations.cli.suite](#)), [248](#)
 suite_demo() (in module [great_expectations.cli.suite](#)), [247](#)
 suite_edit() (in module [great_expectations.cli.suite](#)), [247](#)
 suite_list() (in module [great_expectations.cli.suite](#)), [248](#)
 suite_new() (in module [great_expectations.cli.suite](#)), [247](#)
 suite_scaffold() (in module [great_expectations.cli.suite](#)), [248](#)
 SuiteEditNotebookRenderer (class in [great_expectations.render.renderer.suite_edit_notebook_renderer](#)), [261](#)
 SuiteScaffoldNotebookRenderer (class in [great_expectations.render.renderer.suite_scaffold_notebook_renderer](#)), [261](#)
 SupportedDatabases (class in [great_expectations.cli.datasource](#)), [241](#)
 T
 table_batch_kwargs_generator ([great_expectations.datasource.batch_kwargs_generator.table_batch_kwargs_generator](#) attribute), [546](#)
 table() ([great_expectations.datasource.batch_kwargs_generator.table_batch_kwargs_generator](#) property), [546](#)
 table() ([great_expectations.datasource.types.batch_kwargs.SqlAlchemyBatchKwargsGenerator](#) property), [556](#)
 table() ([great_expectations.datasource.types.SqlAlchemyBatchKwargsGenerator](#) property), [559](#)
 TableBatchKwargsGenerator (class in [great_expectations.datasource.batch_kwargs_generator.table_batch_kwargs_generator](#)), [553](#)
 TableBatchKwargsGenerator (class in [great_expectations.datasource.batch_kwargs_generator.table_batch_kwargs_generator](#)), [546](#)
 tell_user_suite_exists() (in module [great_expectations.cli.toolkit](#)), [250](#)
 test_column_aggregate_expectation_function() ([great_expectations.dataset.Dataset](#) method), [468](#)
 test_column_aggregate_expectation_function() ([great_expectations.dataset.Dataset](#) method), [367](#)
 test_column_map_expectation_function() ([great_expectations.dataset.Dataset](#) method), [468](#)
 test_column_map_expectation_function() ([great_expectations.dataset.Dataset](#) method), [367](#)
 test_expectation_function() ([great_expectations.data_asset.data_asset.DataAsset](#) method), [289](#)
 test_expectation_function() ([great_expectations.data_asset.DataAsset](#) method), [300](#)
 TextContent (class in [great_expectations.render.types](#)), [612](#)
 to_evaluation_parameter_urn() ([great_expectations.core.metric.ValidationMetricIdentifier](#) method), [266](#)
 to_fixed_length_tuple() ([great_expectations.core.data_context_key.DataContextKey](#) method), [261](#)
 to_fixed_length_tuple() ([great_expectations.core.data_context_key.StringKey](#) method), [261](#)
 to_fixed_length_tuple() ([great_expectations.core.data_context_key.StringKey](#) method), [261](#)

(`great_expectations.core.DataContextKey`
 method), 268
`to_fixed_length_tuple()`
 (`great_expectations.core.metric.MetricIdentifier`
 method), 265
`to_fixed_length_tuple()`
 (`great_expectations.core.metric.ValidationMetricIdentifier`
 method), 266
`to_fixed_length_tuple()`
 (`great_expectations.core.RunIdentifier`
 method), 270
`to_fixed_length_tuple()`
 (`great_expectations.data_context.types.resource_identifiers.ExpectationIdentifier`
 method), 330
`to_fixed_length_tuple()`
 (`great_expectations.data_context.types.resource_identifiers.ResultIdentifier`
 method), 335
`to_id()` (`great_expectations.core.id_dict.IDDict`
 method), 264
`to_id()` (`great_expectations.core.IDDict` method), 269
`to_json_dict()` (`great_expectations.core.ExpectationConfiguration`
 method), 272
`to_json_dict()` (`great_expectations.core.ExpectationKwargs`
 method), 272
`to_json_dict()` (`great_expectations.core.ExpectationSuite`
 method), 274
`to_json_dict()` (`great_expectations.core.ExpectationSuiteValidationResult`
 method), 280
`to_json_dict()` (`great_expectations.core.ExpectationValidationResult`
 method), 278
`to_json_dict()` (`great_expectations.core.RunIdentifier`
 method), 270
`to_json_dict()` (`great_expectations.render.types.CollapseContent`
 method), 613
`to_json_dict()` (`great_expectations.render.types.RenderedBootstrapTableContent`
 method), 612
`to_json_dict()` (`great_expectations.render.types.RenderedBulletListContent`
 method), 612
`to_json_dict()` (`great_expectations.render.types.RenderedCompoundContent`
 method), 611
`to_json_dict()` (`great_expectations.render.types.RenderedContentBlock`
 method), 611
`to_json_dict()` (`great_expectations.render.types.RenderedContentBlockContainer`
 method), 612
`to_json_dict()` (`great_expectations.render.types.RenderedDocumentationContent`
 method), 613
`to_json_dict()` (`great_expectations.render.types.RenderedGraphContent`
 method), 611
`to_json_dict()` (`great_expectations.render.types.RenderedHeaderSectionContent`
 method), 611
`to_json_dict()` (`great_expectations.render.types.RenderedMarkdownContent`
 method), 612
`to_json_dict()` (`great_expectations.render.types.RenderedSectionContent`
 method), 613
`to_json_dict()` (`great_expectations.render.types.RenderedStringTemplate`
 method), 612
`to_json_dict()` (`great_expectations.render.types.RenderedTableContent`
 method), 611
`to_json_dict()` (`great_expectations.render.types.RenderedTabsContent`
 method), 611
`to_json_dict()` (`great_expectations.render.types.TextContent`
 method), 613
`to_json_dict()` (`great_expectations.render.types.ValueListContent`
 method), 612
`to_json_dict()` (`great_expectations.validation_operators.types.validate_expectations`
 method), 625
`to_json_dict()` (`great_expectations.core.data_context_key.DataContextKey`
 method), 261
`to_tuple()` (`great_expectations.core.data_context_key.StringKey`
 method), 268
`to_tuple()` (`great_expectations.core.DataContextKey`
 method), 265
`to_tuple()` (`great_expectations.core.metric.MetricIdentifier`
 method), 265
`to_tuple()` (`great_expectations.core.metric.ValidationMetricIdentifier`
 method), 266
`to_tuple()` (`great_expectations.core.RunIdentifier`
 method), 270
`to_tuple()` (`great_expectations.data_context.types.resource_identifiers.ExpectationIdentifier`
 method), 333
`to_tuple()` (`great_expectations.data_context.types.resource_identifiers.ResultIdentifier`
 method), 330
`to_tuple()` (`great_expectations.data_context.types.resource_identifiers.ValidationResult`
 method), 338
`to_tuple()` (`great_expectations.data_context.types.resource_identifiers.RunIdentifier`
 method), 335
`to_tuple()` (`great_expectations.data_context.types.base.DataContextConfiguration`
 method), 322
`to_tuple()` (`great_expectations.types.base.DotDict`
 class method), 620
 (in module
`great_expectations.profile.metrics_utils`),
 584
`tuple_to_key()` (`great_expectations.data_context.store.Store`
 method), 317
`tuple_to_key()` (`great_expectations.data_context.store.store.Store`
 method), 317
`TupleFilesSystemStoreBackend` (class in
`great_expectations.data_context.store`), 318
`TupleFilesSystemStoreBackend` (class in
`great_expectations.data_context.store.tuple_store_backend`),
 311
`TupleS3StoreBackend` (class in
`great_expectations.data_context.store`), 318
`TupleS3StoreBackend` (class in
`great_expectations.data_context.store.tuple_store_backend`),
 311

[great_expectations.data_context.store](#), 319
[TupleS3StoreBackend](#) (class in [method](#)), 233
[great_expectations.data_context.store.tuple_store_backend](#), 312
[TupleStoreBackend](#) (class in [method](#)), 234
[great_expectations.data_context.store](#), 319
[TupleStoreBackend](#) (class in [method](#)), 235
[great_expectations.data_context.store.tuple_store_backend](#), 311
[upgrade_project\(\)](#) (in module [great_expectations.cli.upgrade_helpers.upgrade_helper_v11](#)), 251
[upgrade_store_backend\(\)](#) (in module [great_expectations.cli.upgrade_helpers.upgrade_helper_v11](#)), 234
[UnavailableMetricError](#), 269, 651
[UNCOMMITTED_DIRECTORIES](#) (in module [great_expectations.cli.upgrade_helpers.upgrade_helper_v11](#)), 235
[great_expectations.data_context.BaseDataContext](#) (class in [great_expectations.cli.upgrade_helpers](#)), 353
[UNCOMMITTED_DIRECTORIES](#) (in module [great_expectations.cli.upgrade_helpers.upgrade_helper_v11](#)), 235
[great_expectations.data_context.data_context.BaseDataContext](#) (class in [great_expectations.cli.upgrade_helpers.upgrade_helper_v11](#)), 339
[UNIQUE](#) (in module [great_expectations.profile.base.ProfilerCardinality](#)), 580
[unknown](#) (in module [great_expectations.data_context.types.base.DataSourceConnector](#)), 326
[UNKNOWN](#) (in module [great_expectations.profile.base.ProfilerDataType](#)), 580
[urn_word](#) (in module [great_expectations.core.urn](#)), 266
[usage_statistics_enabled](#) (in module [great_expectations.core.usage_statistics.usage_statistics](#)), 259
[UnsupportedConfigVersionError](#), 653
[update_expectation_state\(\)](#) (in module [great_expectations.jupyter_ux.expectation_explorer.ExpectationExplorer](#)), 574
[update_kwarg_widget_dict\(\)](#) (in module [great_expectations.jupyter_ux.expectation_explorer.ExpectationExplorer](#)), 574
[update_result\(\)](#) (in module [great_expectations.jupyter_ux.expectation_explorer.ExpectationExplorer](#)), 574
[update_return_obj\(\)](#) (in module [great_expectations.data_context.types.base.AnonymizedUsageStatisticsHandler](#)), 322
[update_return_obj\(\)](#) (in module [great_expectations.core.usage_statistics.usage_statistics](#)), 259
[update_return_obj\(\)](#) (in module [great_expectations.data_context.data_context.BaseDataContext](#)), 345

V

[great_expectations.data_context.data_context.ExplorerDataContext](#) (class in [great_expectations.data_asset.data_asset.DataAsset](#)), 287
[validate\(\)](#) (in module [great_expectations.data_asset.DataAsset](#)), 298
[validate\(\)](#) (in module [great_expectations.profile.base.DataAssetProfiler](#)), 580
[validate\(\)](#) (in module [great_expectations.profile.base.DatasetProfiler](#)), 580
[validate_config\(\)](#) (in module [great_expectations.util](#)), 659
[validate_config\(\)](#) (in module [great_expectations.data_context.BaseDataContext](#)), 353

validate_config() (great_expectations.data_context.data_context.BaseDataContext.Action (class in
 class method), 339
 validate_distribution_parameters() (in module great_expectations.dataset.util), 465
 validate_input() (great_expectations.render.renderer.content_block.expect_block.CreditBlock.Renderer.actions),
 class method), 587
 validate_message() ValidationMetric (class in
 (great_expectations.core.usage_statistics.usage_statistics.UsageStatisticsHandler.core.metric), 265
 method), 259
 ValidationMetricIdentifier (class in
 validate_result_dict() great_expectations.core.metric), 265
 (great_expectations.core.ExpectationValidationResult), ValidationOperator (class in
 method), 278
 great_expectations.validation_operators),
 validate_schema() 644
 (great_expectations.data_context.types.base.DataContextConfigSchema (class in
 method), 329
 great_expectations.validation_operators.validation_operators),
 validate_schema() 631
 (great_expectations.data_context.types.base.DataContextConfigSchema (class in
 method), 326
 great_expectations.core.usage_statistics.anonymizers.validation_
 validation_operator() (in module 257
 great_expectations.cli.validation_operator), ValidationOperatorResult (class in
 252
 great_expectations.validation_operators.types.validation_operato
 validation_operator_config 624
 (great_expectations.validation_operators.types.validation_operator_result.ValidationOperatorResultSchema
 attribute), 627
 great_expectations.validation_operators.types.validation_operato
 validation_operator_config() 625
 (great_expectations.validation_operators.ActionListValidationOperatorResultSchema (in module
 property), 643
 great_expectations.validation_operators.types.validation_operato
 validation_operator_config() 627
 (great_expectations.validation_operators.types.validation_operator_result.ValidationOperatorResult in
 property), 624
 great_expectations.data_context.types.resource_identifiers),
 validation_operator_config() 334
 (great_expectations.validation_operators.validation_operators.ActionListValidationOperatorSchema (class in
 property), 634
 great_expectations.data_context.types.resource_identifiers),
 validation_operator_config() 335
 (great_expectations.validation_operators.validation_operators.ValidationOperatorIdentifierSchema (in mod-
 property), 631
 ule great_expectations.data_context.types.resource_identifiers),
 validation_operator_config() 338
 (great_expectations.validation_operators.validation_operators.WarningAndFailureExpectationSuiteValidationOperator
 property), 637
 (class in great_expectations.jupyter_ux), 577
 validation_operator_config() ValidationResultsColumnSectionRenderer
 (great_expectations.validation_operators.ValidationOperator class in great_expectations.render.renderer),
 property), 644
 608
 validation_operator_config() ValidationResultsColumnSectionRenderer
 (great_expectations.validation_operators.WarningAndFailureExpectationSuiteValidationOperator class in
 property), 647
 598
 validation_operator_list() (in module ValidationResultsPageRenderer (class in
 great_expectations.cli.validation_operator), great_expectations.render.renderer), 609
 252
 ValidationResultsPageRenderer (class in
 validation_operator_run() (in module great_expectations.render.renderer.page_renderer),
 great_expectations.cli.validation_operator), 599
 252
 ValidationResultsTableContentBlockRenderer
 validation_operators (class in great_expectations.render.renderer.content_block),
 (great_expectations.data_context.types.base.DataContextConfigSchema 648

[ValidationResultsTableContentBlockRender](#) ([class in `great_expectations.render.renderer.content_block.validation_results_table_content_block`](#)), [context.DataContext](#) [592](#)
[write_config_variables_template_to_disk\(\)](#) ([class method](#)), [348](#)
[validations_store\(\)](#) ([great_expectations.data_context.BaseDataContext](#) [property](#)), [358](#)
[write_config_variables_template_to_disk\(\)](#) ([great_expectations.data_context.DataContext](#) [class method](#)), [362](#)
[validations_store\(\)](#) ([great_expectations.data_context.data_context.BaseDataContext](#) [property](#)), [344](#)
[write_config_variables_template_to_disk\(\)](#) ([great_expectations.DataContext](#) [class method](#)), [661](#)
[validations_store_name](#) ([great_expectations.data_context.types.base.DataContextConfig](#) [attribute](#)), [328](#)
[write_index_page\(\)](#) ([great_expectations.data_context.store.html_site_store.HtmlSiteStore](#) [method](#)), [307](#)
[validations_store_name\(\)](#) ([great_expectations.data_context.BaseDataContext](#) [property](#)), [358](#)
[write_index_page\(\)](#) ([great_expectations.data_context.store.HtmlSiteStore](#) [method](#)), [316](#)
[validations_store_name\(\)](#) ([great_expectations.data_context.data_context.BaseDataContext](#) [property](#)), [344](#)
[write_notebook_to_disk\(\)](#) ([great_expectations.render.renderer.suite_edit_notebook_renderer](#) [class method](#)), [606](#)
[ValidationsStore](#) ([class in `great_expectations.data_context.store`](#)), [320](#)
[write_project_template_to_disk\(\)](#) ([great_expectations.data_context.data_context.DataContext](#) [class method](#)), [348](#)
[ValidationsStore](#) ([class in `great_expectations.data_context.store.validations_store`](#)), [313](#)
[write_project_template_to_disk\(\)](#) ([great_expectations.data_context.DataContext](#) [class method](#)), [362](#)
[ValidationStatistics](#) ([in `great_expectations.data_asset.data_asset`](#)), [289](#)
[write_project_template_to_disk\(\)](#) ([great_expectations.DataContext](#) [class method](#)), [661](#)
[Validator](#) ([class in `great_expectations.validator.validator`](#)), [648](#)
[ValueListContent](#) ([class in `great_expectations.render.types`](#)), [612](#)
[verify_dynamic_loading_support\(\)](#) ([in `module great_expectations.util`](#)), [655](#)
[yaml \(in `module great_expectations.cli.checkpoint`\)](#), [236](#)
[yaml \(in `module great_expectations.cli.toolkit`\)](#), [249](#)
[verify_that_key_to_filepath_operation_is_reversible\(\)](#) ([in `module great_expectations.data_context.data_context`](#)), [338](#)
[verify_that_key_to_filepath_operation_is_reversible\(\)](#) ([great_expectations.data_context.store.tuple_store_backend.TupleStoreBackend](#) [method](#)), [311](#)
[verify_that_key_to_filepath_operation_is_reversible\(\)](#) ([in `module great_expectations.data_context.types.base`](#)), [321](#)
[verify_that_key_to_filepath_operation_is_reversible\(\)](#) ([great_expectations.datasource.datasource](#) [method](#)), [559](#)
[yaml \(in `module great_expectations.types.base`\)](#), [620](#)
[VersioneerConfig](#) ([class in `great_expectations.version`](#)), [649](#)
[yaml_anchor\(\)](#) ([great_expectations.types.base.DotDict](#) [class method](#)), [620](#)
[versions_from_parentdir\(\)](#) ([in `module great_expectations.version`](#)), [649](#)
[yield_batch_kwargs\(\)](#) ([great_expectations.datasource.batch_kwargs_generator.batch_kwargs_generator](#) [method](#)), [536](#)
[VERY_FEW](#) ([great_expectations.profile.base.ProfilerCardinality](#) [attribute](#)), [580](#)
[VERY_MANY](#) ([great_expectations.profile.base.ProfilerCardinality](#) [attribute](#)), [580](#)

W

[WarningAndFailureExpectationSuitesValidationOperator](#) ([class in `great_expectations.validation_operators`](#)), [645](#)
[WarningAndFailureExpectationSuitesValidationOperator](#) ([class in `great_expectations.validation_operators.validation_operators`](#)), [635](#)