great_expectations Documentation

The Great Expectations Team

Jun 14, 2019

Contents

CHAPTER 1

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.
- Monitor data quality in production data pipelines and data products.
- Simplify debugging data pipelines if (when) they break.
- Codify assumptions used to build models when sharing with distributed teams or other analysts.

1.3 How do I get started?

It's easy! Just use pip install:

```
$ pip install great_expectations
```

You can also clone the repository, which includes examples of using great_expectations.

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

1.4 What expectations are available?

Expectations include:

- `expect_table_row_count_to_equal`
- `expect_column_values_to_be_unique`
- `expect_column_values_to_be_in_set`
- `expect_column_mean_to_be_between`
- . . . and many more

Visit the glossary of expectations for a complete list of expectations that are currently part of the great expectations vocabulary.

1.5 Can I contribute?

Absolutely. Yes, please. Start here, and don't be shy with questions!

1.6 How do I learn more?

For full documentation, visit Great Expectations on readthedocs.io.

Down with Pipeline Debt! explains the core philosophy behind Great Expectations. Please give it a read, and clap, follow, and share while you're at it.

For quick, hands-on introductions to Great Expectations' key features, check out our walkthrough videos:

- Introduction to Great Expectations
- Using Distributional Expectations

1.7 What's the best way to get in touch with the Great Expectations team?

Issues on GitHub. If you have questions, comments, feature requests, etc., opening an issue is definitely the best path forward.

We also have a slack channel, which you can join here: https://tinyurl.com/great-expectations-slack

1.8 Great Expectations doesn't do X. Is it right for my use case?

It depends. If you have needs that the library doesn't meet yet, please upvote an existing issue(s) or open a new issue and we'll see what we can do. Great Expectations is under active development, so your use case might be supported soon.

CHAPTER 2

Data Contexts

Data Contexts manage connections to Great Expectations Datasets. Note: data contexts will be changed significantly during the next release of GE.

To get a data context, simply call *get_data_context()* on the ge object:

```
>> import great_expectations as ge
>> options = { ## my connection options }
>> sql_context = ge.get_data_context('sqlalchemy_context', options)

>> sql_dataset = sql_context.get_dataset('table_name')
```

There are currently four types of data contexts:

- *PandasCSVDataContext*: The PandasCSVDataContext ('PandasCSV') exposes a local directory containing files as datasets.
- *SqlAlchemyDataContext*: The SqlAlchemyDataContext ('SqlAlchemy') exposes tables from a SQL-compliant database as datasets.
- *SparkCSVDataContext*: The SparkCSVDataContext ('SparkCSV') exposes csv files accessible from a SparkSQL context.
- *DatabricksTableContext*: The DatabricksTableContext ('DatabricksTable') exposes tables from a databricks notebook.

All data contexts expose the following methods:

- list_datasets(): lists datasets available in current context
- get_dataset(dataset_name): returns a dataset with the matching name (e.g. filename or tablename)

2.1 *PandasCSVDataContext*

The *options* paramater for a PandasCSVDataContext is simply the glob pattern matching the files to be available.

2.2 *SqlAlchemyDataContext*

The *options* parameter for a SqlAlchemyDataContext is the sqlalchemy connection string to connect to the database.

2.3 *SparkCSVDataContext*

The *options* parameter for a SparkCSVDataContext is a directory from which to read a CSV file, and options to pass to the reader.

2.4 *DatabricksTableContext*

The *options* parameter for a _DatabricksTableContext is a dataase from which to expose tables; get_dataset optionally also accepts a date partition.

CHAPTER 3

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.

3.1 Connect-and-expect

Great Expectations's connect-and-expect API makes it easy to declare Expectations within the tools you already use for data exploration: jupyter notebooks, the ipython console, scratch scripts, etc.

```
>> import great_expectations as ge
>> my_df = ge.read_csv("./tests/examples/titanic.csv")

>> my_df.expect_column_values_to_be_in_set(
    "Sex",
    ["male", "female"]
)
{
    'success': True,
    'summary_obj': {
        'unexpected_count': 0,
        'unexpected_percent': 0.0,
        'unexpected_percent_nonmissing': 0.0,
        'partial_unexpected_list': []
    }
}
```

3.2 Instant feedback

When you invoke an Expectation method from a notebook or console, it will immediately return a dictionary containing the result and a list of exceptions.

For example:

```
>> print my_df.PClass.value_counts()
3rd    711
1st    322
2nd    279
*        1
Name: PClass, dtype: int64

>> my_df.expect_column_values_to_be_in_set(
    "PClass",
    ["1st", "2nd", "3rd"]
)
{
    'success': False,
    'summary_obj': {
        'unexpected_count': 1,
        'unexpected_percent': 0.0007616146230007616,
        'unexpected_percent_nonmissing': 0.0007616146230007616,
        'partial_unexpected_list': ['*']
    }
}
```

Another example:

```
 >> my_df.expect_column_values_to_match_regex(
     "Name",
     "^[A-Za-z\, \(\)\']+$"
 )
 {
     'success': False,
     'summary_obj': {
         'unexpected_count': 16,
         'unexpected_percent': 0.012185833968012186,
         'unexpected_percent_nonmissing': 0.012185833968012186,
         'partial_unexpected_list': [
             'Bjornstrm-Steffansson, Mr Mauritz Hakan',
             'Brown, Mrs James Joseph (Margaret Molly" Tobin)"',
             'Frolicher-Stehli, Mr Maxmillian',
             'Frolicher-Stehli, Mrs Maxmillian (Margaretha Emerentia Stehli)',
             'Lindeberg-Lind, Mr Erik Gustaf',
             'Roebling, Mr Washington Augustus 2nd',
             'Rothes, the Countess of (Noel Lucy Martha Dyer-Edwardes)',
             'Simonius-Blumer, Col Alfons',
             'Thorne, Mr George (alias of: Mr George Rosenshine)',
             'Downton (?Douton), Mr William James',
             'Aijo-Nirva, Mr Isak',
             'Johannesen-Bratthammer, Mr Bernt',
             'Larsson-Rondberg, Mr Edvard',
             'Nicola-Yarred, Miss Jamila',
             'Nicola-Yarred, Master Elias',
             'Thomas, Mr John (? 1st/2nd class)'
         ]
     }
}
```

This instant feedback helps you zero in on exceptions very quickly, taking a lot of the pain and guesswork out of early data exploration.

3.3 Capture More About Your Data

Build expectations as you conduct exploratory data analysis to ensure insights about data processes and pipelines remain part of your team's knowldege. Great Expectations's library of Expectations has been developed by a broad cross-section of data scientists and engineers. Check out the *Glossary of Expectations*; it covers all kinds of practical use cases:

- Foreign key verification and row-based accounting for ETL
- Form validation and regex pattern-matching for names, URLs, dates, addresses, etc.
- Checks for missing data
- Crosstabs
- Distributions for statistical modeling.
- etc.

You can also add notes or even structured metadata to expectations to describe the intent of an expectation or anything else relevant for understanding it:

```
>> my_df.expect_column_values_to_match_regex(
    "Name",
    "^[A-Za-z\, \(\)\']+$",
    meta = { "notes": "A simple experimental regex for name matching.", "source":
↪"http://great-expectations.readthedocs.io/en/latest/glossary.html" })
```

3.4 Saving Expectations

At the end of your exploration, call *save_expectations* to store all Expectations from your session to your pipeline test files.

This is how you always know what to expect from your data.

```
>> my_df.save_expectations_config("my_titanic_expectations.json")
```

For more detail on how to control expectation output, please see *Standard arguments for expectations* and *result_format*.

3.5 Types of Expectations

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.

CHAPTER 4

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.

4.1 Great Expectations' Philosophy of Distributional Expectations

Great Expectations attempts to provide a simple, expressive framework for describing distributional expectations. The framework generally adopts a nonparametric approach, although it is possible to build expectations from parameterized distributions.

The design is motivated by the following assumptions:

- **Encoding expectations into a simple object that allows for portable data pipeline testing is the top priority.** In many circumstances the loss of precision associated with "compressing" data into an expectation may be beneficial because of its intentional simplicity as well as because it adds a very light layer of obfuscation over the data which may align with privacy preservation goals.
- **While it should be possible to easily extend the framework with more rigorous statistical tests, great expectations** should provide simple, reasonable defaults. Care should be taken in cases where robust statistical guarantees are expected.
- **Building and interpreting expectations should be intuitive: a more precise partition object implies a more precise** expectation.

4.2 Partition Objects

The core constructs of a great expectations distributional expectation are the partition 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 lower_bound_i <= x < upper_bound_i.
- However, following the behavior of numpy.histogram, a data element x is in the largest bin k if x == upper_bound_k.
- A bin may include -Infinity and Infinity as endpoints, however, those endpoints are not supported by the Kolmogorov-Smirnov test.
- Partition weights define the probability of the associated interval. Note that this effectively applies a "piecewise uniform" distribution to the data for the purpose of statistical tests.

Example continuous partition object:

```
{
  "bins": [ 0, 1, 2, 10],
  "weights": [0.3, 0.3, 0.4]
}
```

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]
}
```

4.3 Constructing Partition Objects

Three convenience functions are available to easily construct partition objects from existing data:

- *continuous_partition_data*
- *categorical_partition_data*
- *kde_partition_data*

Convenience functions are also provided to validate that an object is a valid partition density object:

- *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.

4.4 Distributional Expectations Core Tests

Distributional expectations rely on three tests for their work.

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 provided weights are scaled to the size of the data in the tested column at the time of the test.

- `expect_column_chisquare_test_p_value_to_be_greater_than`

4.5 Distributional Expectations Alternatives

The core partition density object used in current expectations focuses on a particular (partition-based) method of "compressing" the data into a testable form, however it may be desireable to use alternative nonparametric approaches (e.g. Fourier transform/wavelets) to describe expected data.

CHAPTER 5

Validation

Once you've constructed and stored Expectations, you can use them to validate new data.

```
>> import json
>> import great_expectations as ge
>> my_expectations_config = json.load(file("my_titanic_expectations.json"))
>> my_df = ge.read_csv(
    "./tests/examples/titanic.csv",
    expectations_config=my_expectations_config
)
>> 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,
        "min_value": 20
      }
    },
    {
      "unexpected_list": [],
      "expectation_type": "expect_column_values_to_be_between",
```

(continues on next page)

(continued from previous page)


```
      "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,
  }
}
```


Calling great_expectations's `validate` method generates a JSON-formatted report. The report contains information about:

- the overall sucess (the *success* field),
- summary statistics of the expectations (the *statistics* field), and
- the detailed results of each expectation (the *results* field).

5.1 Command-line validation

This is especially powerful when combined with great_expectations's command line tool, which lets you validate in a one-line bash script.

```
$ great_expectations validate tests/examples/titanic.csv \
    tests/examples/titanic_expectations.json
{
  "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,
        "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,
```

(continues on next page)

(continued from previous page)

```
      "kwargs": {
        "column": "PClass",
        "value_set": [
          "1st",
          "2nd",
          "3rd"
        ]
      }
    }
  ]
  "success", False,
  "statistics": {
      "evaluated_expectations": 10,
      "successful_expectations": 9,
      "unsuccessful_expectations": 1,
      "success_percent": 90.0
  }
}
```

5.2 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 *Using Evaluation Parameters* for more information.

CHAPTER 6

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 your testing framework early,
2. capture those early learnings while they're still fresh, 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!

CHAPTER 7

Glossary of Expectations

7.1 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.

7.2 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*

7.3 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*

7.4 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*

7.5 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*

7.6 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*

7.7 Aggregate functions

- *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_stdev_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*

7.8 Column pairs

- *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*

7.9 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*

7.10 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*

CHAPTER 8

Advanced

8.1 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 config 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.

8.1.1 *result_format*

See *result_format* for more information.

8.1.2 *include_config*

All Expectations accept a boolean *include_config* parameter. If true, then the expectation config 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,
)

{
    'exception_index_list': [0, 10, 11, 12, 13, 14],
    'exception_list': ['A', 'E', 'E', 'E', 'E', 'E'],
```

(continues on next page)

(continued from previous page)

```
    '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
}
```

8.1.3 *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.

8.1.4 *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.

```
>> my_df.expect_column_values_to_be_in_set(
    "my_column",
    ["a", "b", "c"],
    meta={
        "foo": "bar",
        "baz": [1,2,3,4]
    }
)
{
    "success": False,
    "meta": {
        "foo": "bar",
        "baz": [1,2,3,4]
    }
}
```

8.1.5 *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]
```

(continues on next page)

(continued from previous page)

```
>> 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
    ],
    "unexpected_percent_nonmissing": 0.2,
    "unexpected_count": 2
  }
}
```

8.1.6 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 my_dataset.csv my_expectations.json --result_format=BOOLEAN_ONLY --
↪catch_exceptions=False --include_config=True
```

8.2 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.

8.2.1 *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 *result*	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 *column_map_expectation* 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 *column_aggregate_expectation* type expectations:				
observed_value	no	yes	yes	yes
details (e.g. statistical details)	no	no	yes	yes

result_format 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.

8.2.2 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', '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'}
)
{
    'success': False,
    'result': {
        'unexpected_count': 6,
        'unexpected_percent': 0.1666666666666666,
        'unexpected_percent_nonmissing': 0.1666666666666666,
        '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': 0.1666666666666666,
        'unexpected_percent_nonmissing': 0.1666666666666666,
        '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'}
)
```

(continues on next page)

(continued from previous page)

```
{
    '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'}*.

8.2.3 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
}
```

8.2.4 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
```

(continues on next page)

(continued from previous page)

```
        "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

{
    "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
    }
}
```

8.2.5 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`)
        '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': 0.1666666666666666,
        'unexpected_percent_nonmissing': 0.1666666666666666,
        '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
```

(continues on next page)

(continued from previous page)

```
        'details': {<expectation-specific result justification fields>}
    }
}
```

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
    }
}
```

8.2.6 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': 0.1666666666666666,
        'unexpected_percent_nonmissing': 0.1666666666666666,
        '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
    }
}
```

8.3 Autoinspection

It can be very convenient to have great expectations automatically review a dataset and suggest expectations that may be appropriate. Currently, there's a very basic, but easily extensible, autoinspection capability available.

Dataset objects have an *autoinspect* method which allows you to provide a function that will evaluate a dataset object and add expectations to it. By default *autoinspect* will call the autoinspect function `columns_exist` which will add an *expect_column_to_exist* expectation for each column currently present on the dataset.

To implement additional autoinspection functions, you simply take a single parameter, a Dataset, and evaluate and add expectations to that object.

```
>> import great_expectations as ge
>> df = ge.dataset.PandasDataset({"col": [1, 2, 3, 4, 5]})
>> df.autoinspect(ge.dataset.autoinspect.columns_exist)
>> df.get_expectations_config()
    {'dataset_name': None,
     'meta': {'great_expectations.__version__': '0.4.4__develop'},
     'expectations': [
         {'expectation_type': 'expect_column_to_exist',
          'kwargs': {'column': 'col'}
         }]
    }
```

8.4 Using 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, but we may still want to ensure that we can use the same expectation configuration object.

Great Expectations makes working with parameters of that kind easy! When declaring an expectation, you can specify that a particular argument is an evaluation parameter that should be substituted at evaluation time, and 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
}
```

You can also store parameter values in a special dictionary called evaluation_parameters that is stored in the expectations_config 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"})
```

When validating expectations, you can provide evaluation parameters based on upstream results:

```
>> my_df.validate(expectations_config=my_dag_step_config, evaluation_parameters={
↪"upstream_row_count": upstream_row_count})
```

Finally, the command-line tool also allows you to provide a JSON file that contains parameters to use during evaluation:

```
>> cat my_parameters_file.json
{
    "upstream_row_count": 10
}
>> great_expectations validate --evaluation_paramters=my_parameters_file.json dataset_
↪file.csv expectations_config.json
```

8.5 Custom expectations

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_medical_diagnosis_code
expect_column_value_to_be_be_unicode_encodable
```

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.

Side note: in future versions, Great Expectations will probably grow to include additional Expectations. If you have an Expectation that could be universally useful, please make the case on the Great Expectations issue tracker on github.

8.5.1 The easy way

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 *Naming conventions* 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:

@MetaPandasDataset.column_map_expectation decorates a custom function, wrapping it 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.

To work with these decorators, your custom function must accept two arguments: *self* and *column*. When your function is called, *column* will contain all the non-null values in the given column. Your function must return a series of boolean values in the same order, with the same index.

@MetaPandasDataset.column_aggregate_expectation accepts *self* and *column*. It must return a dictionary containing a boolean *success* value, and a nested dictionary called *result* which contains an *observed_value* argument.

```
from great_expectations.dataset import PandasDataset, MetaPandasDataset

class CustomPandasDataset(PandasDataset):

    @MetaPandasDataset.column_map_expectation
    def expect_column_values_to_equal_2(self, column):
```

(continues on next page)

(continued from previous page)

```
        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, the decorators work slightly differently. See the MetaSqlAlchemy class docstrings for more information.

```
import sqlalchemy as sa
from great_expectations.dataset import SqlAlchemyDataset, MetaSqlAlchemyDataset

class CustomSqlAlchemyDataset(SqlAlchemyDataset):

    @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,
            }
        }
```

8.5.2 The hard way

1. Create a subclass from the dataset class of your choice
2. Write the whole expectation yourself
3. Decorate it with the *@expectation* decorator

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*. Malformed result objects can break Great Expectations in subtle and unanticipated ways.

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

class CustomPandasDataset(PandasDataset):
```

(continues on next page)

(continued from previous page)

```
    @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,
                    '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,
                "unexpected_list" : unexpected_values[:20],
                "unexpected_index_list" : list(self.index[result==False])[:20],
            }
        else:
            return {
                "success" : len(unexpected_values) == 0,
                "unexpected_list" : unexpected_values[:20],
                "unexpected_index_list" : list(self.index[result==False])[:20],
            }
```

8.5.3 The quick way

For rapid prototyping, you can use the following syntax to quickly iterate on the logic for expectations.

```
>> DataAsset.test_expectation_function(my_func)

>> Dataset.test_column_map_expectation_function(my_map_func, column='my_column')

>> Dataset.test_column_aggregate_expectation_function(my_agg_func, column='my_column')
```

These functions will return output just like regular expectations. However, they will NOT save a copy of the expectation to the config.

8.5.4 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)
```

(continues on next page)

(continued from previous page)

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

A similar approach works for the command-line tool.

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

8.6 Naming conventions

*expect_*_to_**

8.7 Extending Great Expectations

When implementing an expectation defined in the base *Dataset* for a new backend, add the *@DocInherit* decorator first to use the default dataset documentation for the expectation. That can help users of your dataset see consistent documentation no matter which backend is implementing the great_expectations API.

@DocInherit overrides your function's __get__ method with one that will replace the local docstring with the docstring from its parent. It is defined in *Dataset.util*.

8.8 Changelog and Roadmap

8.8.1 Planned Features

- Improved project initialization and data contexts
- Improved variable typing
- Support for non-tabular datasources (e.g. JSON, XML, AVRO)

v.0.6.1__develop

v.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]*)

8.8.2 v.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

8.8.3 v.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)

8.8.4 v.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).

8.8.5 v.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)

8.8.6 v.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)

8.8.7 v.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_lenghts_to_[be_between|equal] for SQAlchemy (thanks @ccnobbli)
- Fix PandasDataset subclasses to inherit child class

8.8.8 v.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)

8.8.9 v.0.4.1

- Correct inclusion of new data_context module in source distribution

8.8.10 v.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 * the output formats are explicitly hierarchical now, with BOOLEAN_ONLY < BASIC < SUMMARY < COMPLETE. ‘column_aggregate_expectation‘s 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_gr (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

8.8.11 v.0.3.2

- Include requirements file in source dist to support conda

8.8.12 v.0.3.1

- Fix infinite recursion error when building custom expectations
- Catch dateutil parsing overflow errors

8.8.13 v.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

8.9 Implemented Expectations

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
expect_column_to_exist	True	True	True
expect_table_columns_to_match_ordered_list	True	True	True
expect_table_row_count_to_be_between	True	True	True
expect_table_row_count_to_equal	True	True	True
expect_column_values_to_be_unique	True	True	True
expect_column_values_to_not_be_null	True	True	True
expect_column_values_to_be_null	True	True	True
expect_column_values_to_be_of_type	True	False	False
expect_column_values_to_be_in_type_list	True	False	False
expect_column_values_to_be_in_set	True	True	True
expect_column_values_to_not_be_in_set	True	True	True
expect_column_values_to_be_between	True	True	False
expect_column_values_to_be_increasing	True	False	False
expect_column_values_to_be_decreasing	True	False	False
expect_column_value_lengths_to_be_between	True	True	False
expect_column_value_lengths_to_equal	True	True	True
expect_column_values_to_match_regex	True	False	True
expect_column_values_to_not_match_regex	True	False	True
expect_column_values_to_match_regex_list	True	False	False
expect_column_values_to_not_match_regex_list	True	False	False
expect_column_values_to_match_strftime_format	True	False	False
expect_column_values_to_be_dateutil_parseable	True	False	False
expect_column_values_to_be_json_parseable	True	False	False
expect_column_values_to_match_json_schema	True	False	False
expect_column_parameterized_distribution_ks_test_p_value_to_be_greater_than	True	False	False
expect_column_distinct_values_to_equal_set	True	True	True
expect_column_distinct_values_to_contain_set	True	True	True
expect_column_mean_to_be_between	True	True	True
expect_column_median_to_be_between	True	True	False
expect_column_stdev_to_be_between	True	False	True
expect_column_unique_value_count_to_be_between	True	True	True

Continued on next page

Table 1 – continued from previous page

expect_column_proportion_of_unique_values_to_be_between	True	True	True
expect_column_most_common_value_to_be_in_set	True	False	True
expect_column_sum_to_be_between	True	True	True
expect_column_min_to_be_between	True	True	True
expect_column_max_to_be_between	True	True	True
expect_column_chisquare_test_p_value_to_be_greater_than	True	True	True
expect_column_bootstrapped_ks_test_p_value_to_be_greater_than	True	False	False
expect_column_kl_divergence_to_be_less_than	True	True	True
expect_column_pair_values_to_be_equal	True	False	False
expect_column_pair_values_A_to_be_greater_than_B	True	False	False
expect_column_pair_values_to_be_in_set	True	False	False
expect_multicolumn_values_to_be_unique	True	False	False

CHAPTER 9

Module Docs

9.1 Data Asset Module

9.1.1 great_expectations.data_asset.base

class great_expectations.data_asset.base.**DataAsset**(**args*, ***kwargs*)

class great_expectations.data_asset.base.**ValidationStatistics**(*evaluated_expectations*, *successful_expectations*, *unsuccessful_expectations*, *success_percent*, *success*)

Bases: tuple

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

__getstate__()
Exclude the OrderedDict from pickling

__repr__()
Return a nicely formatted representation string

evaluated_expectations
Alias for field number 0

success
Alias for field number 4

success_percent
Alias for field number 3

successful_expectations
Alias for field number 1

unsuccessful_expectations
Alias for field number 2

9.1.2 great_expectations.data_asset.file_data_asset

class great_expectations.data_asset.file_data_asset.**MetaFileDataAsset**(**args*, ***kwargs*)

Bases: *great_expectations.data_asset.base.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(*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 kwarg _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*

expect_file_line_regex_match_count_to_be_between(**args*, ***kwargs*)
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 (**args*, ***kwargs*)

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.
- **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_hash_to_equal (**args*, ***kwargs*)

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

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

Checks to see if a file specified by the user actually exists

Parameters **filepath** (*str or None*) – The filepath to evalutate. 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 (**args*, ***kwargs*)

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

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*.

class great_expectations.data_asset.file_data_asset.**MetaFileDataAsset**(**args*, ***kwargs*)

Bases: *great_expectations.data_asset.base.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(*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 kwarg _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

9.1.3 great_expectations.data_asset.util

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. :param partition_object: The partition_object to evaluate :return: 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:

```
{
    "partition": (list) The categorical values present in the data
    "weights": (list) The weights of the values in the partition.
}
```

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:

```
{
    "partition": (list) The endpoints of the partial partition of␣
↪reals,
    "weights": (list) The densities of the bins implied by the␣
↪partition.
}
```

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*)

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.

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.
}
```

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.**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.

9.2 Dataset Module

9.2.1 great_expectations.dataset.dataset

class great_expectations.dataset.dataset.**MetaDataset**(**args*, ***kwargs*)
Bases: *great_expectations.data_asset.base.DataAsset*

Holds expectation decorators.

classmethod column_aggregate_expectation(*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*

hashable_getters = ['get_column_max', 'get_column_mean', 'get_column_median', 'get_col

get_row_count()
Returns: int, table row count

get_table_columns()
Returns: List[str], list of column names

get_column_nonnull_count(*column*)
Returns: int

get_column_mean(*column*)
Returns: float

get_column_value_counts(*column*)
Returns: pd.Series of value counts for a column, sorted by value

get_column_sum(*column*)
Returns: float

get_column_max(*column*, *parse_strings_as_datetimes=False*)
Returns: any

get_column_min(*column*, *parse_strings_as_datetimes=False*)
Returns: any

get_column_unique_count (*column*)
Returns: int

get_column_modes (*column*)
Returns: List[any], list of modes (ties OK)

get_column_median (*column*)
Returns: any

get_column_stdev (*column*)
Returns: float

get_column_hist (*column*, *bins*)
Returns: List[int], a list of counts corresponding to bins

get_column_count_in_range (*column*, *min_val=None*, *max_val=None*, *min_strictly=False*, *max_strictly=True*)
Returns: int

classmethod column_map_expectation (*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 percent 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_unique` for an example of a column_map_expectation

test_column_map_expectation_function (*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* for more information.

test_column_aggregate_expectation_function(*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* for more information.

expect_column_to_exist(***kwargs*)
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(***kwargs*)

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_row_count_to_be_between(***kwargs*)

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 (***kwargs*)

Expect the number of rows to equal a value.

expect_table_row_count_to_equal is a basic `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

expect_column_values_to_be_unique (*column*, *mostly=None*, *result_format=None*, *include_config=False*, *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=0.6.*

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 percent 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(*column*, *mostly=None*, *result_format=None*, *include_config=False*, *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 percent 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(*column*, *mostly=None*, *result_format=None*, *include_config=False*, *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 percent 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(*column*, *type_*, *mostly=None*, *result_format=None*, *include_config=False*, *catch_exceptions=None*, *meta=None*)

Expect each column entry to be a specified data type.

expect_column_values_to_be_of_type is a `column_map_expectation`.

Parameters

- **column** (*str*) – The column name.
- **type_** (*str*) – A string representing the data type that each column should have as entries. For example, "double integer" refers to an integer with double precision.

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return *"success": True* if at least mostly percent 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*.

Warning: expect_column_values_to_be_of_type is slated for major changes in future versions of great_expectations.

As of v0.3, great_expectations is exclusively based on pandas, which handles typing in its own peculiar way. Future versions of great_expectations will allow for Datasets in SQL, spark, etc. When we make that change, we expect some breaking changes in parts of the codebase that are based strongly on pandas notions of typing.

See also:

expect_column_values_to_be_in_type_list

expect_column_values_to_be_in_type_list(*column*, *type_list*, *mostly=None*, *result_format=None*, *include_config=False*, *catch_exceptions=None*, *meta=None*)

Expect each column entry to match a list of specified data types.

expect_column_values_to_be_in_type_list is a `column_map_expectation`.

Parameters

- **column** (*str*) – The column name.
- **type_list** (*list of str*) – A list of strings representing the data type that each column should have as entries. For example, "double integer" refers to an integer with double precision.

Keyword Arguments **mostly** (*None or a float between 0 and 1*) – Return *"success": True* if at least mostly percent 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*.

Warning: expect_column_values_to_be_in_type_list is slated for major changes in future versions of great_expectations.

As of v0.3, great_expectations is exclusively based on pandas, which handles typing in its own peculiar way. Future versions of great_expectations will allow for Datasets in SQL, spark, etc. When we make

that change, we expect some breaking changes in parts of the codebase that are based strongly on pandas notions of typing.

See also:

expect_column_values_to_be_of_type

expect_column_values_to_be_in_set(*column*, *value_set*, *mostly=None*, *parse_strings_as_datetimes=None*, *result_format=None*, *include_config=False*, *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": 0.16666666666666666,
    "unexpected_percent_nonmissing": 0.16666666666666666,
    "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 percent 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(*column*, *value_set*, *mostly=None*, *parse_strings_as_datetimes=None*, *result_format=None*, *include_config=False*, *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": 0.5,
    "unexpected_percent_nonmissing": 0.5,
    "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 percent 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(*column*, *min_value=None*, *max_value=None*, *allow_cross_type_comparisons=None*, *parse_strings_as_datetimes=False*, *output_strftime_format=None*, *mostly=None*, *result_format=None*, *include_config=False*, *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

- **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 strfime 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 percent 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_be_between

expect_column_values_to_be_increasing(*column*, *strictly=None*, *parse_strings_as_datetimes=False*, *mostly=None*, *result_format=None*, *include_config=False*, *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 percent 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(*column*, *strictly=None*, *parse_strings_as_datetimes=False*, *mostly=None*, *result_format=None*, *include_config=False*, *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 percent 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(*column*, *min_value=None*, *max_value=None*, *mostly=None*, *result_format=None*, *include_config=False*, *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 percent 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`(*column*, *value*, *mostly=None*, *result_format=None*, *include_config=False*, *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 percent 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(*column*, *regex*, *mostly=None*, *result_format=None*, *include_config=False*, *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 percent 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(*column*, *regex*, *mostly=None*, *result_format=None*, *include_config=False*, *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 percent 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(*column*, *regex_list*, *match_on='any'*, *mostly=None*, *result_format=None*, *include_config=False*, *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 percent 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(*column*, *regex_list*, *mostly=None*, *result_format=None*, *include_config=False*, *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 percent 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(*column*, *strftime_format*, *mostly=None*, *result_format=None*, *include_config=False*, *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 percent 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(*column*, *mostly=None*, *result_format=None*, *include_config=False*, *catch_exceptions=None*, *meta=None*)

Expect column entries to be parseable 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 percent 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 (*column*, *mostly=None*, *result_format=None*, *include_config=False*, *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 percent 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(*column*, *json_schema*, *mostly=None*, *result_format=None*, *include_config=False*, *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 percent 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 at: http://json-schema.org/

expect_column_parameterized_distribution_ks_test_p_value_to_be_greater_than(*column*, *distribution*, *p_value=0.05*, *params=None*, *result_format=None*, *include_config=False*, *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 parameteric 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
- **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_equal_set (***kwargs*)

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.

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]
  },
}
```

expect_column_distinct_values_to_equal_set is a `column_aggregate_expectation`.

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 (***kwargs*)

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

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
```

(continues on next page)

(continued from previous page)

```
"result": {
    "observed_value": [1,2,3]
},
}
```

expect_column_distinct_values_to_contain_set is a `column_aggregate_expectation`.

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 (***kwargs*)

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.

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.
- 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 (***kwargs*)

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.

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.
- 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_stdev_to_be_between

expect_column_stdev_to_be_between (***kwargs*)

Expect the column standard deviation to be between a minimum value and a maximum value.

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.

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.
- 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 (***kwargs*)

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 (***kwargs*)
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.

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)

expect_column_unique_value_count_to_be_between is a `column_aggregate_expectation`.

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.
- 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 (***kwargs*)
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 (***kwargs*)

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 minimum number of unique values allowed.
- **max_value** (*comparable type 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": (list) The actual column sum
}
```

- 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

expect_column_min_to_be_between (***kwargs*)

Expect the column to sum 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 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_values, and all non-null column values to datetimes before making comparisons.
- **output_strftime_format** (*str or None*) – A valid strfime 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.
- 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 (***kwargs*)

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_values, and all non-null column values to datetimes before making comparisons.
- **output_strftime_format** (*str or None*) – A valid strfime 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 max
}
```

- 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

expect_column_chisquare_test_p_value_to_be_greater_than (***kwargs*)

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
    }
}
```

expect_column_bootstrapped_ks_test_p_value_to_be_greater_than(*column*, *partition_object=None*, *p=0.05*, *bootstrap_samples=None*, *bootstrap_sample_size=None*, *result_format=None*, *include_config=False*, *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 (***kwargs*)

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 mechanims 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.

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
```

(continues on next page)

(continued from previous page)

```
    "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_

expect_column_pair_values_to_be_equal(*column_A*, *column_B*, *ignore_row_if='both_values_are_missing'*, *result_format=None*, *include_config=False*, *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(*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=False*, *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(*column_A*, *column_B*, *value_pairs_set*, *ignore_row_if='both_values_are_missing'*, *result_format=None*, *include_config=False*, *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(*column_list*, *ignore_row_if='all_values_are_missing'*, *result_format=None*, *include_config=False*, *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*.

9.2.2 great_expectations.dataset.pandas_dataset

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(*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(*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(*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.core.frame.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).

get_row_count()

Returns: int, table row count

get_table_columns()

Returns: List[str], list of column names

get_column_sum(*column*)

Returns: float

get_column_max(*column*, *parse_strings_as_datetimes=False*)
Returns: any

get_column_min(*column*, *parse_strings_as_datetimes=False*)
Returns: any

get_column_mean(*column*)
Returns: float

get_column_nonnull_count(*column*)
Returns: int

get_column_value_counts(*column*)
Returns: pd.Series of value counts for a column, sorted by value

get_column_unique_count(*column*)
Returns: int

get_column_modes(*column*)
Returns: List[any], list of modes (ties OK)

get_column_median(*column*)
Returns: any

get_column_stdev(*column*)
Returns: float

get_column_hist(*column*, *bins*)
Returns: List[int], a list of counts corresponding to bins

get_column_count_in_range(*column*, *min_val=None*, *max_val=None*, *min_strictly=False*, *max_strictly=True*)
Returns: int

expect_column_values_to_not_match_regex_list(***kwargs*)
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`.

Args: column (str): The column name. regex_list (list): The list of regular expressions which the column entries should not match

Keyword Args: mostly (None or a float between 0 and 1): Return *"success": True* if at least mostly percent 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 *SUMM*
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_parameterized_distribution_ks_test_p_value_to_be_greater_than(***kwargs*)

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 parameteric 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
- **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_pair_values_to_be_equal(***kwargs*)
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(***kwargs*)
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(***kwargs*)

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(***kwargs*)

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*.

9.2.3 great_expectations.dataset.sqlalchemy_dataset

class great_expectations.dataset.sqlalchemy_dataset.**MetaSqlAlchemyDataset**(**args*, ***kwargs*)

Bases: *great_expectations.dataset.dataset.Dataset*

classmethod column_map_expectation(*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*, *engine=None*, *connection_string=None*, *custom_sql=None*, *schema=None*, **args*, ***kwargs*)

Bases: *great_expectations.dataset.sqlalchemy_dataset.MetaSqlAlchemyDataset*

get_row_count()

Returns: int, table row count

get_table_columns()

Returns: List[str], list of column names

get_column_nonnull_count(*column*)

Returns: int

get_column_sum(*column*)

Returns: float

get_column_max(*column*, *parse_strings_as_datetimes=False*)

Returns: any

get_column_min(*column*, *parse_strings_as_datetimes=False*)

Returns: any

get_column_value_counts(*column*)

Returns: pd.Series of value counts for a column, sorted by value

get_column_mean(*column*)

Returns: float

get_column_unique_count(*column*)

Returns: int

get_column_median(*column*)
Returns: any

get_column_hist(*column*, *bins*)
return a list of counts corresponding to bins

get_column_count_in_range(*column*, *min_val=None*, *max_val=None*, *min_strictly=False*, *max_strictly=True*)
Returns: int

create_temporary_table(*table_name*, *custom_sql*)
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()
If we can't reflect the table, use a query to at least get column names.

expect_column_values_to_not_match_regex_list(**args*, ***kwargs*)
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 percent 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

9.2.4 great_expectations.dataset.sparkdf_dataset

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(*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.

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.

get_row_count()

Returns: int, table row count

get_table_columns()

Returns: List[str], list of column names

get_column_nonnull_count(*column*)

Returns: int

get_column_mean(*column*)

Returns: float

get_column_sum(*column*)

Returns: float

get_column_max(*column*, *parse_strings_as_datetimes=False*)

Returns: any

get_column_min(*column*, *parse_strings_as_datetimes=False*)

Returns: any

get_column_value_counts(*column*)

Returns: pd.Series of value counts for a column, sorted by value

get_column_unique_count(*column*)

Returns: int

get_column_modes(*column*)

leverages computation done in _get_column_value_counts

get_column_median(*column*)

Returns: any

get_column_stdev(*column*)

Returns: float

get_column_hist(*column*, *bins*)
return a list of counts corresponding to bins

get_column_count_in_range(*column*, *min_val=None*, *max_val=None*, *min_strictly=False*, *max_strictly=True*)
Returns: int

9.2.5 great_expectations.dataset.util

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. :param partition_object: The partition_object to evaluate :return: 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:

```
{
    "partition": (list) The categorical values present in the data
    "weights": (list) The weights of the values in the partition.
}
```

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:

```
{
    "partition": (list) The endpoints of the partial partition of␣
↪reals,
    "weights": (list) The densities of the bins implied by the␣
↪partition.
}
```

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*)

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.

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.
}
```

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.**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.

9.2.6 great_expectations.dataset.autoinspect

Autoinspect utilities to automatically generate expectations by evaluating a data_asset.

exception great_expectations.dataset.autoinspect.**AutoInspectError**(*message*)

Bases: exceptions.Exception

Exception raised for errors in autoinspection.

message -- explanation of the error

great_expectations.dataset.autoinspect.**columns_exist**(*inspect_dataset*)

This function will take a dataset and add expectations that each column present exists.

Parameters **inspect_dataset** (*great_expectations.dataset*) – The dataset to inspect and to which to add expectations.

9.3 Data Context Module

great_expectations.data_context.**get_data_context**(*context_type*, *options*, **args*, ***kwargs*)

Return a data_context object which exposes options to list datasets and get a dataset from that context. This is a new API in Great Expectations 0.4, and is subject to rapid change.

Parameters

- **context_type** – (string) one of "SqlAlchemy", "PandasCSV", "SparkCSV", or "DatabricksTable"
- **options** – options to be passed to the data context's connect method.

Returns a new DataContext object

9.3.1 great_expectations.data_context.base

class great_expectations.data_context.base.**DataContext**(*options*, **args*, ***kwargs*)

Bases: object

A generic DataContext, exposing the base API including constructor with *options* parameter, list_datasets, and get_dataset.

Warning: this feature is new in v0.4 and may change based on community feedback.

connect(*options*)

list_datasets()

get_dataset(*dataset_name*, *caching=False*, ***kwargs*)

9.3.2 great_expectations.data_context.PandasCSVDataContext

class great_expectations.data_context.pandas_context.**PandasCSVDataContext**(**args*, ***kwargs*)

Bases: *great_expectations.data_context.base.DataContext*

A PandasCSVDataContext makes it easy to get a list of files available in the list_datasets method. Its get_dataset method returns a new Pandas dataset with the provided name.

Warning: this feature is new in v0.4 and may change based on community feedback.

connect(*options*)

list_datasets()

get_dataset(*dataset_name*, *caching=False*, ***kwargs*)

9.3.3 great_expectations.data_context.SqlAlchemyDataContext

class great_expectations.data_context.sqlalchemy_context.**SqlAlchemyDataContext**(**args*, ***kwargs*)

Bases: *great_expectations.data_context.base.DataContext*

A SqlAlchemyDataContext creates a SQLAlchemy engine and provides a list of tables available in the list_datasets method. Its get_dataset method returns a new SqlAlchemy dataset with the provided name.

Warning: this feature is new in v0.4 and may change based on community feedback.

connect(*options*, **args*, ***kwargs*)

list_datasets()

get_dataset(*dataset_name*, *custom_sql=None*, *schema=None*, *caching=False*)

CHAPTER 10

Indices and tables

- genindex
- modindex
- search

Python Module Index

g

Index

Symbols

A

C

D

E

F

G

H

I

K

L

M

P

S

T

U

V