Bulwark Documentation

Release 0.3.0

Zax Rosenberg

Aug 24, 2019

Contents

Bulwark is a package for convenient property-based testing of pandas dataframes, supported for Python 3.5+.

This project was heavily influenced by the no-longer-supported Engarde. library by Tom Augspurger (thanks for the head start, Tom!), which itself was modeled after the R library assertr.

CHAPTER 1

Why?

Data are messy, and pandas is one of the go-to libraries for analyzing tabular data. In the real world, data analysts and scientists often feel like they don't have the time or energy to think of and write tests for their data. Bulwark's goal is to let you check that your data meets your assumptions of what it should look like at any (and every) step in your code, without making you work too hard.

1.1 Installation

```
pip install bulwark
```

CHAPTER 2

Usage

Bulwark comes with checks for many of the common assumptions you might want to validate for the functions that make up your ETL pipeline, and lets you toss those checks as decorators on the functions you're already writing:

```
import bulwark.decorators as dc

@dc.IsShape(-1, 10)
@dc.IsMonotonic(strict=True)
@dc.HasNoNans()
def compute(df):
    # complex operations to determine result
    ...
    return result_df
```

Still want to have more robust test files? Bulwark's got you covered there, too, with importable functions.

```
import bulwark.checks as ck

df.pipe(ck.has_no_nans())
```

Won't I have to go clean up all those decorators when I'm ready to go to production? Nope - just toggle the built-in debug_mode flag available for every decorator.

```
@dc.IsShape((3, 2), enabled=False)
def compute(df):
    # complex operations to determine result
    ...
    return result_df
```

What if the test I want isn't part of the library? Use the built-in *CustomCheck* to use your own custom function!

```
def len_longer_than(df, l):
    if len(df) <= l:
        raise AssertionError("df is not as long as expected.")
    return df
```

(continues on next page)

(continued from previous page)

```
@dc.CustomCheck(len_longer_than, df=df, l=6)
def append_a_df(df, df2):
    return df.append(df2, ignore_index=True)

df = pd.DataFrame({"a": [1, 2, 3], "b": [4, 5, 6]})
df2 = pd.DataFrame({"a": [1, np.nan, 3, 4], "b": [4, 5, 6, 7]})

append_a_df(df, df2)
```

What if I want to run a lot of tests and want to see all the errors at once? You can use the built-in *MultiCheck*. It will collect all of the errors and either display a warning message of throw an exception based on the *warn* flag. You can even use custom functions with MultiCheck:

```
def len_longer_than(df, l):
    if len(df) <= l:
        raise AssertionError("df is not as long as expected.")
    return df

# `checks` takes a dict of function: dict of params for that function.
# Note that those function params EXCLUDE df.
# Also note that when you use MultiCheck, there's no need to use CustomCheck - just feed in the function.
@dc.MultiCheck(checks={ck.has_no_nans: {"columns": None},
                       len_longer_than: {"l": 6}},
               warn=False)
def append_a_df(df, df2):
    return df.append(df2, ignore_index=True)

df = pd.DataFrame({"a": [1, 2, 3], "b": [4, 5, 6]})
df2 = pd.DataFrame({"a": [1, np.nan, 3, 4], "b": [4, 5, 6, 7]})

append_a_df(df, df2)
```

Check out Examples to see more advanced usage.

CHAPTER 3

Contents

3.1 Changelog

[0.3.0] - 2019-05-30 Added - Add *exact_order* param to *has_columns*

Changed - Hotfix for reversed *has_columns* error messages for missing and unexpected columns - Breaking change to *has_columns* parameter name *exact*, which is now *exact_cols*

[0.2.0] - 2019-05-29 Added - Add *has_columns* check, which asserts that the given columns are contained within the df or exactly match the df's columns. - Add changelog

Changed - Breaking change to rename unique_index to has_unique_index for consistency

[0.1.2] - 2019-01-13 Changed - Improve code base to automatically generate decorators for each check - Hotfix multi_check and unit tests

[0.1.1] - 2019-01-12 Changed - Hotfix to setup.py for the sphinx.setup_command.BuildDoc requirement.

[0.1.0] - 2019-01-12 Changed - Breaking change to rename unique_index to has_unique_index for consistency

3.2 Installation

```
pip install bulwark
```

3.3 Quickstart

Coming soon!

3.4 Design

Coming soon!

3.5 Examples

Coming soon!

3.6 Decorators

3.7 Checks

3.8 Contributing

To contribute, start by cloning this repo:

```
git clone https://github.com/ZaxR/bulwark.git
```

Create a feature branch off of the develop branch:

```
git checkout develop
git checkout -b feature/<snake_case_feature_name>
```

Docstrings and tests required for any new functions/classes/modules. Plesae use Google-formatted docstrings.

Rebuild the docs in your local version using:

```
sphinx-apidoc -o ./docs/_source ./bulwark
cd docs
make html
```

Create a PR to the develop branch.

Tests will be be triggered to run via Travis CI.