jupyterlab_celltests Documentation

Release 0.1.2

Tim Paine

Feb 19, 2020

Contents

Cell-by-cell testing for production Jupyter notebooks in JupyterLab

CHAPTER 1

Overview

`Celltests` is designed for writing tests for linearly executed notebooks. Its primary use is for report unit tests.

1.1 "Linearly executed notebooks?"

When converting notebooks into html/pdf/email reports, they are executed from top-to-bottom one time, and are expected contain as little code as reasonably possible, focusing primarily on the plotting and markdown bits. Libraries for this type of thing include Papermill, JupyterLab Emails, etc.

1.2 Doesn't this already exist?

Nbval is a great product and I recommend using it for notebook regression tests. But it compares the executed notebook's outputs to its existing outputs, which doesn't align well with dynamic reports which might be run everyday with different input/output data.

1.3 So why do I want this again?

This doesn't necessarily help you if your data sources go down, but its likely you'll notice this anyway. Where this comes in handy is:

- when the environment (e.g. package versions) are changing in your system
- when you play around in the notebook (e.g. nonlinear execution) but aren't sure if your reports will still generate
- when your software lifecycle systems have a hard time dealing with notebooks (can't lint/audit them as code unless integrated nbdime/nbconvert to script, tough to test, tough to ensure what works today works tomorrow)

1.4 So what does this do?

Given a notebook, you can write mocks and assertions for individual cells. You can then generate a testing script for this notebook, allowing you to hook it into your testing system and thereby provide unittests of your report.

1.4.1 Writing tests

When you write tests for a cell, we create a new method on a `unittest` class corresponding to the index of your cell, and including the cumulative tests for all previous cells (to mimic what has happened so far in the notebook's linear execution). You can write whatever mocking and asserts you like, and can call `%cell` to inject the contents of the cell into your test. The tests themselves are stored in the cell metadata, similar to celltags, slide information, etc.

1.4.2 Running tests

You can run the tests offline from an `.ipynb` file, or you can execute them from the browser and view the results of `pytest-html`'s html plugin.

1.4.3 Extra Tests

- Max number of lines per cell
- Max number of cells per notebook
- Max number of function definitions per notebook
- Max number of class definitions per notebook
- Percentage of cells tested

1.4.4 Example

In the committed `Untitled.ipynb` notebook, but modified so that cell 0 has its import statement copied 10 times (to trigger test and lint failures):

1.5 Tests

```
Untitled_test.py::TestExtension::test_cell0 PASSED                                  ␣
↪                                                    [  8%]
Untitled_test.py::TestExtension::test_cell1 PASSED                                  ␣ ␣
↪                                                    [ 16%]
Untitled_test.py::TestExtension::test_cell2 PASSED                                  ␣ ␣
↪                                                    [ 25%]
Untitled_test.py::TestExtension::test_cell3 PASSED                                  ␣ ␣ ␣
↪                                                    [ 33%]
Untitled_test.py::TestExtension::test_cell_coverage PASSED                          ␣ ␣ ␣
↪                                                    [ 41%]
Untitled_test.py::TestExtension::test_cells_per_notebook PASSED                     ␣ ␣ ␣
↪                                                    [ 50%]
Untitled_test.py::TestExtension::test_class_definition_count PASSED                 ␣ ␣ ␣
↪                                                    [ 58%]
```

(continues on next page)

(continued from previous page)

```
Untitled_test.py::TestExtension::test_function_definition_count PASSED
↪                                                               [ 66%]
Untitled_test.py::TestExtension::test_lines_per_cell_0 FAILED
↪                                                               [ 75%]
Untitled_test.py::TestExtension::test_lines_per_cell_1 PASSED
↪                                                               [ 83%]
Untitled_test.py::TestExtension::test_lines_per_cell_2 PASSED
↪                                                               [ 91%]
Untitled_test.py::TestExtension::test_lines_per_cell_3 PASSED
↪                                                               [100%]
```

1.6 Lint

```
Checking lines in cell 0:   FAILED
Checking lines in cell 1:   PASSED
Checking lines in cell 2:   PASSED
Checking lines in cell 3:   PASSED
Checking cells per notebook <= 10:  PASSED
Checking functions per notebook <= 10:  PASSED
Checking classes per notebook <= 10:    PASSED
Checking cell test coverage >= 50:  PASSED
```