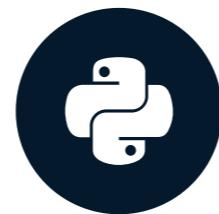


How to organize a growing set of tests?

UNIT TESTING FOR DATA SCIENCE IN PYTHON



Dibya Chakravorty
Test Automation Engineer

What you've done so far

16

- `row_to_list()`
- `convert_to_int()`
- `get_data_as_numpy_array()`
- `split_into_training_and_testing_sets()`

What you've done so far

32

- `row_to_list()`
- `convert_to_int()`
- `get_data_as_numpy_array()`
- `split_into_training_and_testing_sets()`
- `...`

What you've done so far

64

- `row_to_list()`
- `convert_to_int()`
- `get_data_as_numpy_array()`
- `split_into_training_and_testing_sets()`
- `...`

What you've done so far

128

- `row_to_list()`
- `convert_to_int()`
- `get_data_as_numpy_array()`
- `split_into_training_and_testing_sets()`
- `...`

Need a strategy to organize tests

128

Need a strategy to organize tests



Project structure

```
src/
```

```
# All application code lives here
```

Project structure

```
src/                                # All application code lives here
|-- data/                            # Package for data preprocessing
    |-- __init__.py
```

Project structure

```
src/                                # All application code lives here
|-- data/                            # Package for data preprocessing
|   |-- __init__.py
|   |-- preprocessing_helpers.py      # Contains row_to_list(), convert_to_int()
```

Project structure

```
src/                                # All application code lives here
|-- data/                            # Package for data preprocessing
|   |-- __init__.py
|   |-- preprocessing_helpers.py      # Contains row_to_list(), convert_to_int()
|-- features/                         # Package for feature generation from preprocessed data
    |-- __init__.py
```

Project structure

```
src/                                # All application code lives here
|-- data/                            # Package for data preprocessing
|   |-- __init__.py
|   |-- preprocessing_helpers.py      # Contains row_to_list(), convert_to_int()
|-- features/                         # Package for feature generation from preprocessed data
|   |-- __init__.py
|   |-- as_numpy.py                  # Contains get_data_as_numpy_array()
```

Project structure

```
src/                                # All application code lives here
|-- data/                            # Package for data preprocessing
|   |-- __init__.py
|   |-- preprocessing_helpers.py      # Contains row_to_list(), convert_to_int()
|-- features/                         # Package for feature generation from preprocessed data
|   |-- __init__.py
|   |-- as_numpy.py                  # Contains get_data_as_numpy_array()
|-- models/                           # Package for training and testing linear regression model
    |-- __init__.py
```

Project structure

```
src/                                # All application code lives here
|-- data/                            # Package for data preprocessing
|   |-- __init__.py
|   |-- preprocessing_helpers.py      # Contains row_to_list(), convert_to_int()
|-- features/                         # Package for feature generation from preprocessed data
|   |-- __init__.py
|   |-- as_numpy.py                  # Contains get_data_as_numpy_array()
|-- models/                           # Package for training/testing linear regression model
|   |-- __init__.py
|   |-- train.py                     # Contains split_into_training_and_testing_sets()
```

The tests folder

```
src/                                # All application code lives here
|-- data/                            # Package for data preprocessing
|   |-- __init__.py
|   |-- preprocessing_helpers.py      # Contains row_to_list(), convert_to_int()
|-- features/                         # Package for feature generation from preprocessed data
|   |-- __init__.py
|   |-- as_numpy.py                  # Contains get_data_as_numpy_array()
|-- models/                           # Package for training/testing linear regression model
|   |-- __init__.py
|   |-- train.py                     # Contains split_into_training_and_testing_sets()
tests/                                # Test suite: all tests live here
```

The tests folder mirrors the application folder

```
src/                                # All application code lives here
|-- data/                            # Package for data preprocessing
|   |-- __init__.py
|   |-- preprocessing_helpers.py      # Contains row_to_list(), convert_to_int()
|-- features/                         # Package for feature generation from preprocessed data
|   |-- __init__.py
|   |-- as_numpy.py                  # Contains get_data_as_numpy_array()
|-- models/                           # Package for training/testing linear regression model
|   |-- __init__.py
|   |-- train.py                     # Contains split_into_training_and_testing_sets()
tests/                                # Test suite: all tests live here
|-- data/
|   |-- __init__.py
|-- features/
|   |-- __init__.py
|-- models/
    |-- __init__.py
```

Python module and test module correspondence

- `my_module.py` \longleftrightarrow `test_my_module.py`.

```
src/                                # All application code lives here
|-- data/                            # Package for data preprocessing
|   |-- __init__.py
|   |-- preprocessing_helpers.py      # Contains row_to_list(), convert_to_int()
|-- features/                         # Package for feature generation from preprocessed data
|   |-- __init__.py
|   |-- as_numpy.py                  # Contains get_data_as_numpy_array()
|-- models/                           # Package for training/testing linear regression model
|   |-- __init__.py
|   |-- train.py                     # Contains split_into_training_and_testing_sets()
tests/                                # Test suite: all tests live here
|-- data/
|   |-- __init__.py
|   |-- test.preprocessing_helpers.py # Corresponds to module src/data/preprocessing_helpers.py
|-- features/
|   |-- __init__.py
|-- models/
|   |-- __init__.py
```

Structuring tests inside test modules

- Test module: `test_preprocessing_helpers.py`

```
import pytest
from data.preprocessing_helpers import row_to_list, convert_to_int

def test_on_no_tab_no_missing_value():    # A test for row_to_list()
    ...

def test_on_two_tabs_no_missing_value():   # Another test for row_to_list()
    ...

...
def test_with_no_comma():                 # A test for convert_to_int()
    ...

def test_with_one_comma():                # Another test for convert_to_int()
    ...

...
```

Test class

Test class is a container for a single unit's tests



Test class: theoretical structure

- Test module: test_preprocessing_helpers.py

```
import pytest
from data.preprocessing_helpers import row_to_list, convert_to_int

class
```

Test class: theoretical structure

- Test module: test_preprocessing_helpers.py

```
import pytest
from data.preprocessing_helpers import row_to_list, convert_to_int

class TestRowToList():                                # Use CamelCase
```

Test class: theoretical structure

- Test module: test_preprocessing_helpers.py

```
import pytest
from data.preprocessing_helpers import row_to_list, convert_to_int

class TestRowToList(object):
    def test_on_no_tab_no_missing_value():                      # Always put the argument object
        ...                                                       # A test for row_to_list()

    def test_on_two_tabs_no_missing_value():                     # Another test for row_to_list()
        ...
        ...
```

Test class: theoretical structure

- Test module: test_preprocessing_helpers.py

```
import pytest
from data.preprocessing_helpers import row_to_list, convert_to_int

class TestRowToList(object):
    def test_on_no_tab_no_missing_value(self):      # Always put the argument object
        ...
        ...
    def test_on_two_tabs_no_missing_value(self):     # Always put the argument self
        ...
        ...
```

Clean separation

- Test module: `test_preprocessing_helpers.py`

```
import pytest
from data.preprocessing_helpers import row_to_list, convert_to_int

class TestRowToList(object):
    def test_on_no_tab_no_missing_value(self):          # Always put the argument object
        ...
        ...
        def test_on_two_tabs_no_missing_value(self):      # Always put the argument self
            ...
            ...

class TestConvertToInt(object):                      # Test class for convert_to_int()
    def test_with_no_comma(self):                     # A test for convert_to_int()
        ...
        ...
        def test_with_one_comma(self):                  # Another test for convert_to_int()
            ...
            ...
```

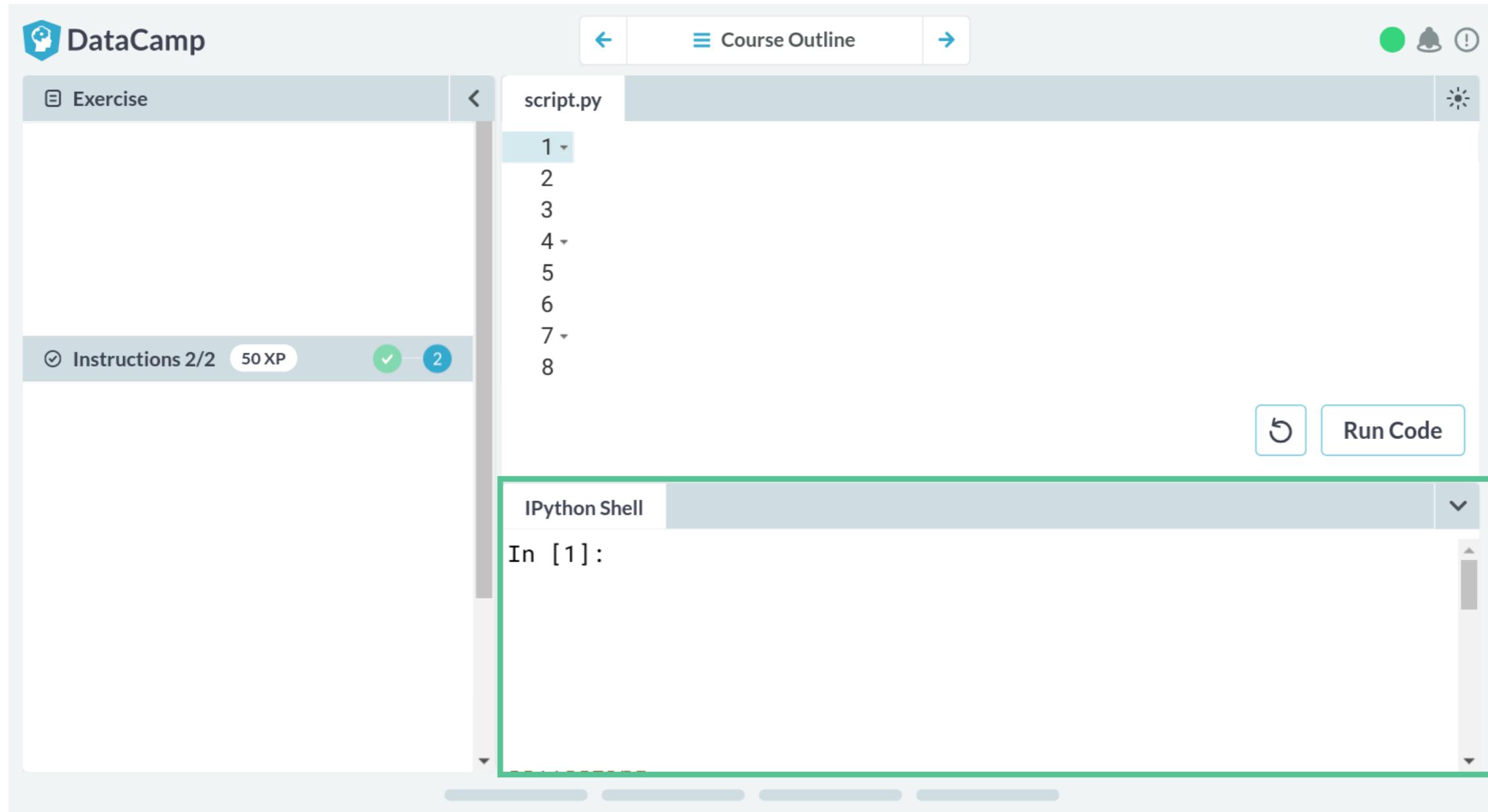
Final test directory structure

```
src/                                # All application code lives here
|-- data/                            # Package for data preprocessing
|   |-- __init__.py
|   |-- preprocessing_helpers.py      # Contains row_to_list(), convert_to_int()
|-- features/                         # Package for feature generation from preprocessed data
|   |-- __init__.py
|   |-- as_numpy.py                  # Contains get_data_as_numpy_array()
|-- models/                           # Package for training/testing linear regression model
|   |-- __init__.py
|   |-- train.py                     # Contains split_into_training_and_testing_sets()
tests/                                # Test suite: all tests live here
|-- data/
|   |-- __init__.py
|   |-- test_preprocessing_helpers.py # Contains TestRowToList, TestConvertToInt
|-- features/
|   |-- __init__.py
|   |-- test_as_numpy.py             # Contains TestGetDataAsNumpyArray
|-- models/
|   |-- __init__.py
|   |-- test_train.py                # Contains TestSplitIntoTrainingAndTestingSets
```

Test directory is well organized!



IPython console's working directory is tests



The screenshot shows a DataCamp exercise interface. At the top, there's a navigation bar with the DataCamp logo, course outline links, and user status indicators. Below the navigation bar, the main area is divided into two sections: an "Exercise" panel on the left and a code editor on the right.

The "Exercise" panel contains:

- A title bar with "Exercise" and a back/forward arrow.
- A progress bar indicating "Instructions 2/2" and "50 XP".
- A green checkmark icon and a blue button with the number "2".

The code editor section has:

- A title bar with "script.py" and line numbers 1 through 8.
- Two buttons at the bottom right: a circular refresh icon and a "Run Code" button.

Below the code editor is an "IPython Shell" window, which is highlighted with a green border. It displays the text "In [1]:".

IPython console's working directory is tests

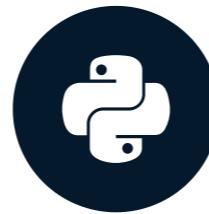
```
src/
|-- data/
|   |-- __init__.py
|   |-- preprocessing_helpers.py
|-- features/
|   |-- __init__.py
|   |-- as_numpy.py
|-- models/
|   |-- __init__.py
|   |-- train.py
tests/                               # This is IPython console's working directory from now on
|-- data/
|   |-- __init__.py
|   |-- test.preprocessing_helpers.py
|-- features/
|   |-- __init__.py
|   |-- test_as_numpy.py
|-- models/
    |-- __init__.py
    |-- test_train.py
```

Let's practice structuring tests!

UNIT TESTING FOR DATA SCIENCE IN PYTHON

Mastering test execution

UNIT TESTING FOR DATA SCIENCE IN PYTHON

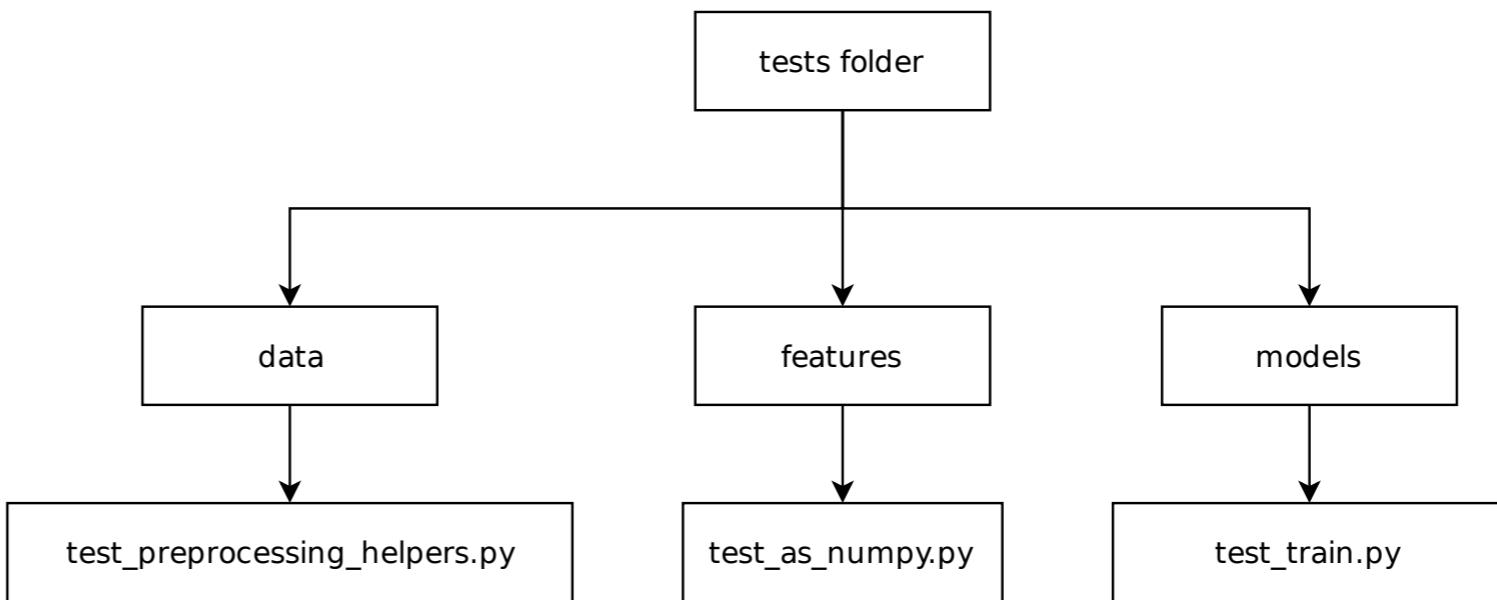


Dibya Chakravorty
Test Automation Engineer

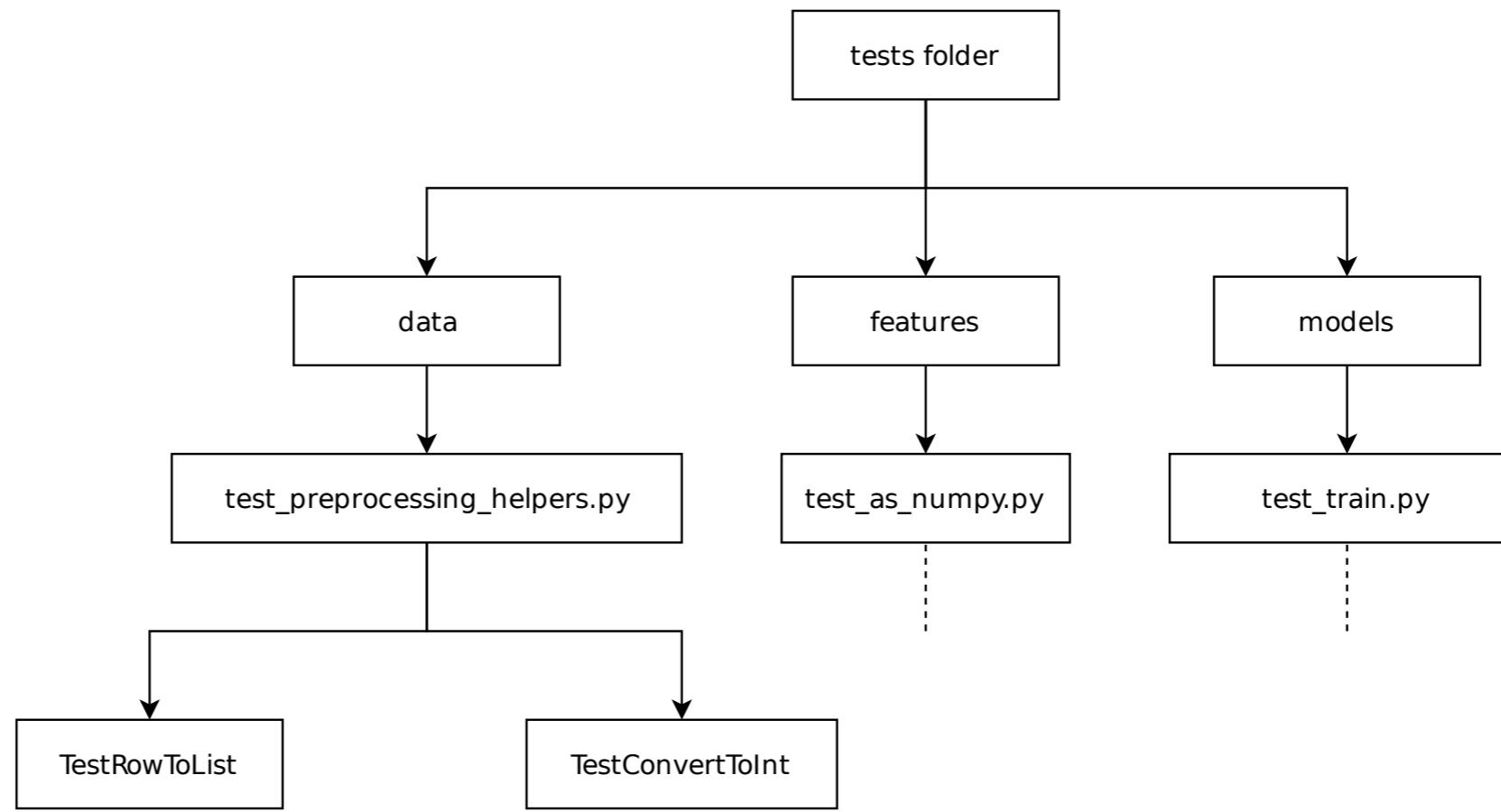
Test organization

tests folder

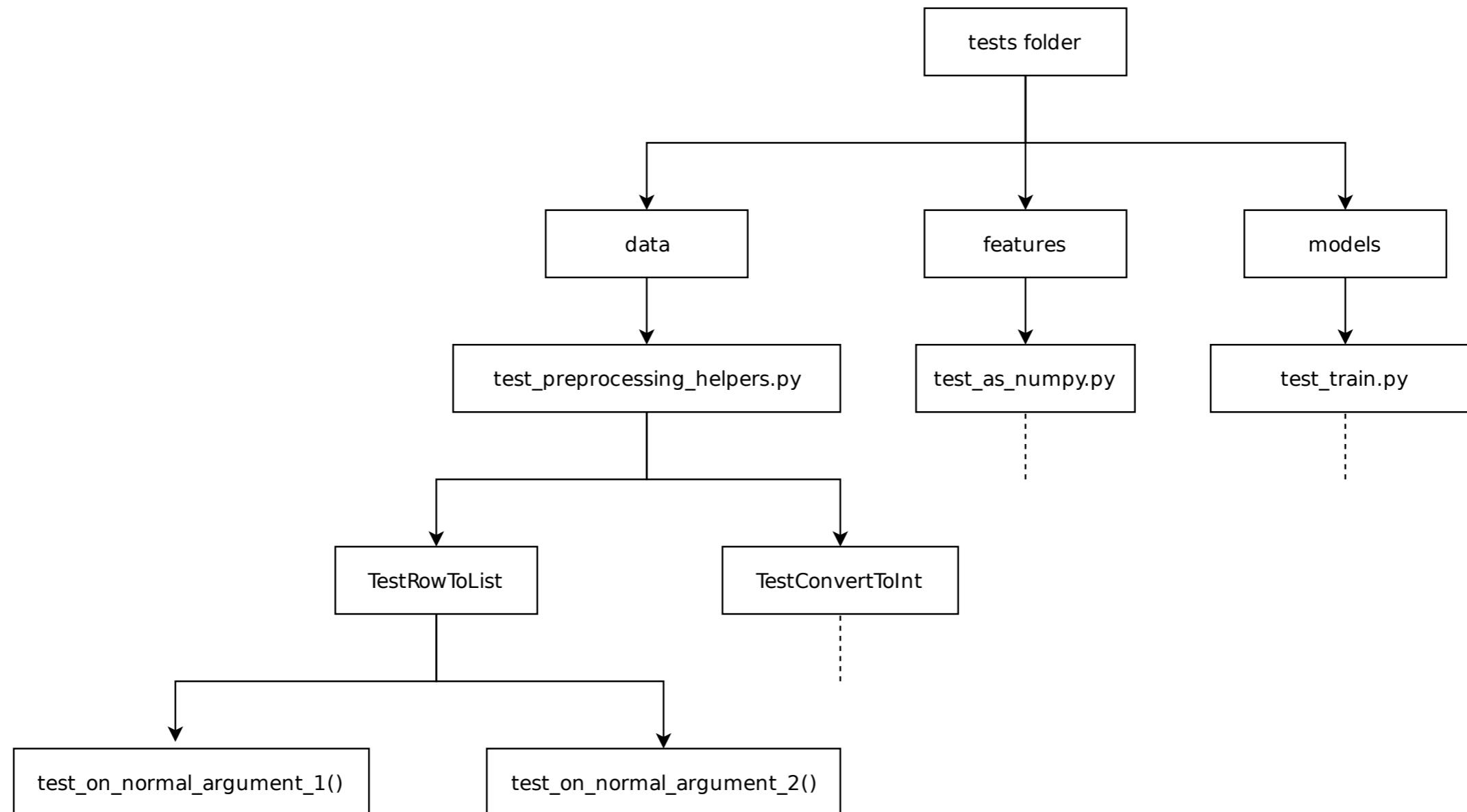
Test organization



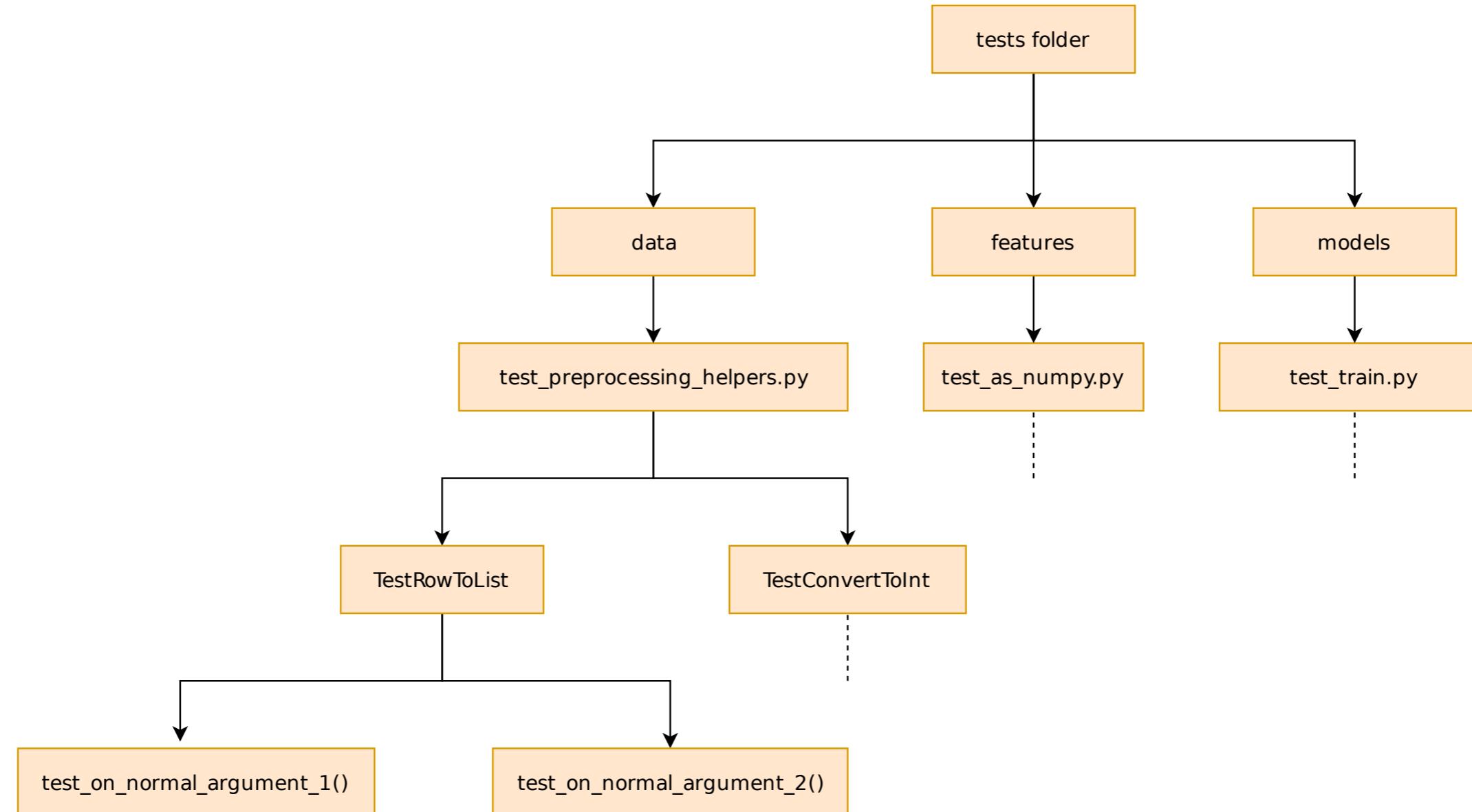
Test organization



Test organization



Running all tests



Running all tests

```
cd tests  
pytest
```

- Recurses into directory subtree of `tests/`.
 - Filenames starting with `test_` → test module.
 - Classnames starting with `Test` → test class.
 - Function names starting with `test_` → unit test.

Running all tests

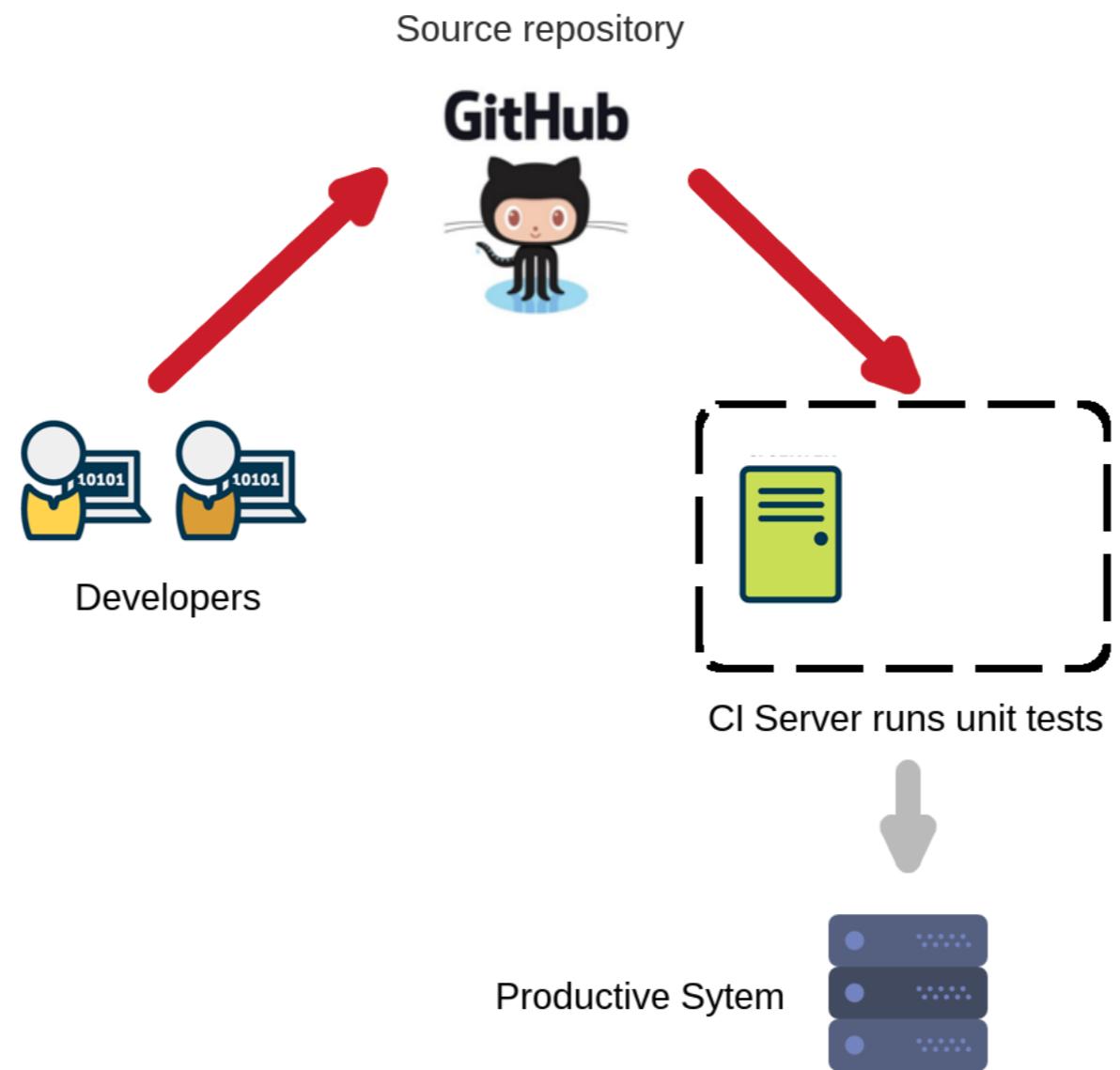
```
===== test session starts =====
data/test_preprocessing_helpers.py .....F.... [ 81%]
features/test_as_numpy.py . [ 87%]
models/test_train.py .. [100%]

===== FAILURES =====
----- TestRowToList.test_on_one_tab_with_missing_value -----
self = <tests.data.test_preprocessing_helpers.TestRowToList object at 0x7f6205475240>

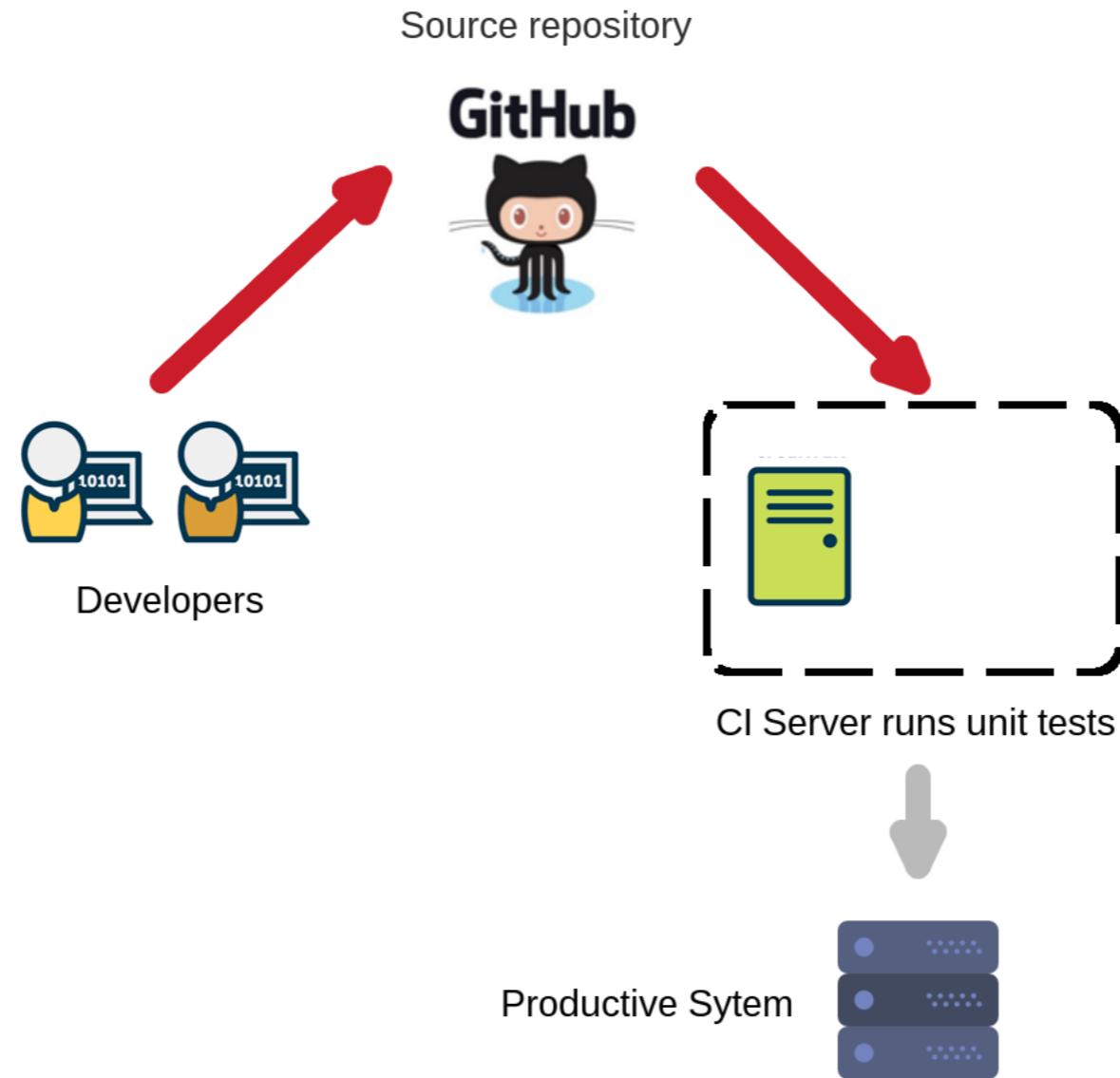
    def test_on_one_tab_with_missing_value(self):    # (1, 1) boundary value
        actual = row_to_list("\t4,567\n")
>       assert actual is None, "Expected: None, Actual: {}".format(actual)
E       AssertionError: Expected: None, Actual: ['', '4,567']
E       assert ['', '4,567'] is None

data/test_preprocessing_helpers.py:55: AssertionError
===== 1 failed, 15 passed in 0.46 seconds =====
```

Typical scenario: CI server



Binary question: do all unit tests pass?



The -x flag: stop after first failure

```
pytest -x
```

```
===== test session starts =====
data/test_preprocessing_helpers.py .....F

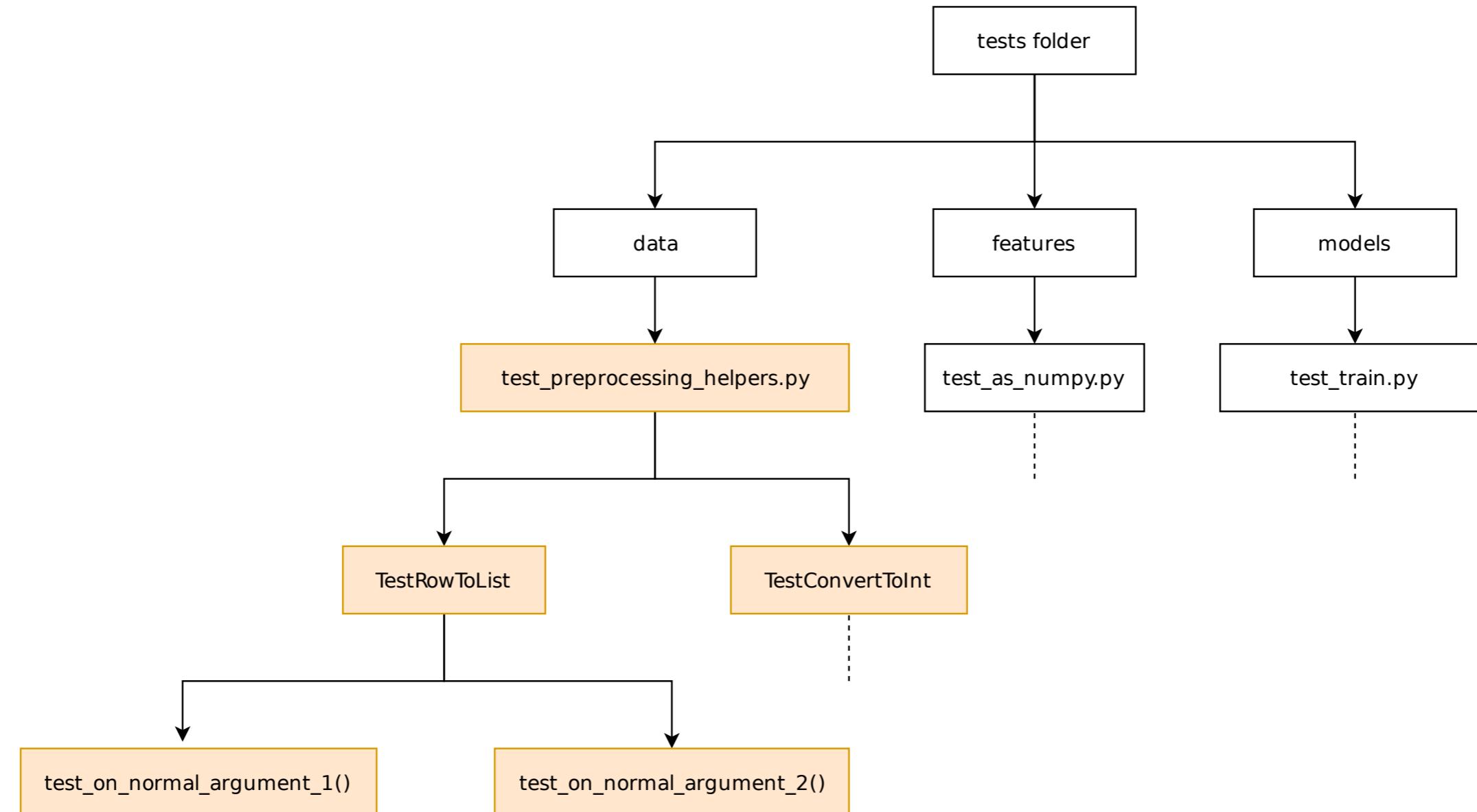
===== FAILURES =====
---- TestRowToList.test_on_one_tab_with_missing_value ----

self = <tests.data.test_preprocessing_helpers.TestRowToList object at 0x7f6309f17198>

    def test_on_one_tab_with_missing_value(self):    # (1, 1) boundary value
        actual = row_to_list("\t4,567\n")
>       assert actual is None, "Expected: None, Actual: {}".format(actual)
E       AssertionError: Expected: None, Actual: ['', '4,567']
E       assert ['', '4,567'] is None

data/test_preprocessing_helpers.py:55: AssertionError
===== 1 failed, 8 passed in 0.45 seconds =====
```

Running tests in a test module



Running tests in a test module

```
pytest data/test_preprocessing_helpers.py
```

```
data/test_preprocessing_helpers.py .....F.... [100%]
```

```
===== FAILURES =====
```

```
----- TestRowToList.test_on_one_tab_with_missing_value -----
```

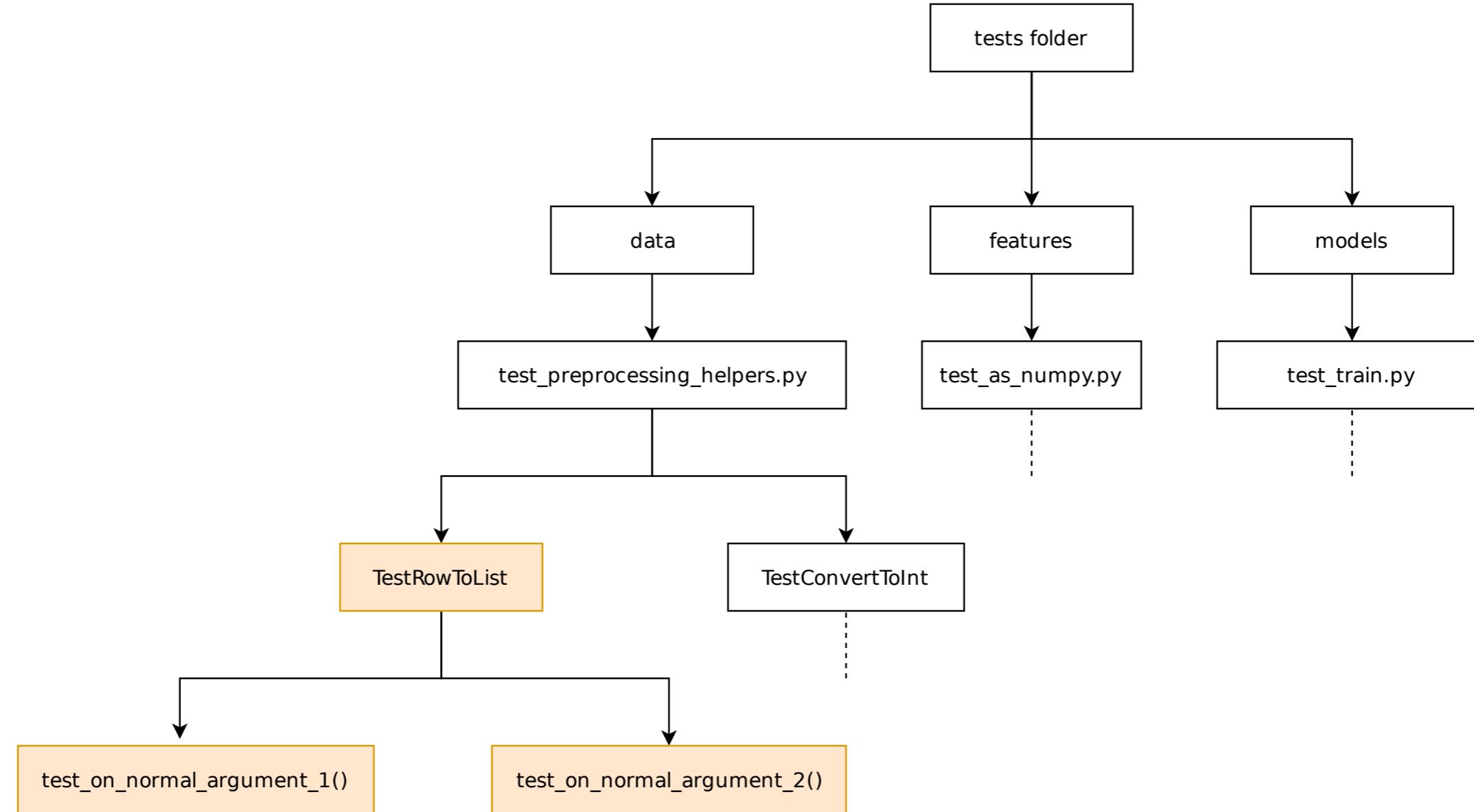
```
self = <tests.data.test_preprocessing_helpers.TestRowToList object at 0x7f435947f198>
```

```
def test_on_one_tab_with_missing_value(self):    # (1, 1) boundary value
    actual = row_to_list("\t4,567\n")
>   assert actual is None, "Expected: None, Actual: {}".format(actual)
E   AssertionError: Expected: None, Actual: ['', '4,567']
E   assert ['', '4,567'] is None
```

```
data/test_preprocessing_helpers.py:55: AssertionError
```

```
===== 1 failed, 12 passed in 0.07 seconds =====
```

Running only a particular test class



Node ID

- Node ID of a test class: <path to test module>::<test class name>
- Node ID of an unit test: <path to test module>::<test class name>::<unit test name>

Running tests using node ID

- Run the test class `TestRowToList`.

```
pytest data/test_preprocessing_helpers.py::TestRowToList
```

```
data/test_preprocessing_helpers.py ..F.... [100%]

=====
 FAILURES =====
----- TestRowToList.test_on_one_tab_with_missing_value -----
----- self = <tests.data.test_preprocessing_helpers.TestRowToList object at 0x7ffb3bac4da0>

----- def test_on_one_tab_with_missing_value(self):      # (1, 1) boundary value
-----     actual = row_to_list("\t4,567\n")
>       assert actual is None, "Expected: None, Actual: {}".format(actual)
E       AssertionError: Expected: None, Actual: ['', '4,567']
E       assert ['', '4,567'] is None

----- data/test_preprocessing_helpers.py:55: AssertionError
===== 1 failed, 6 passed in 0.06 seconds =====
```

Running tests using node ID

- Run the unit test `test_on_one_tab_with_missing_value()`.

```
pytest data/test_preprocessing_helpers.py::TestRowToList::test_on_one_tab_with_missing_value
```

```
data/test_preprocessing_helpers.py F [100]
```

```
===== FAILURES =====
----- TestRowToList.test_on_one_tab_with_missing_value -----

```

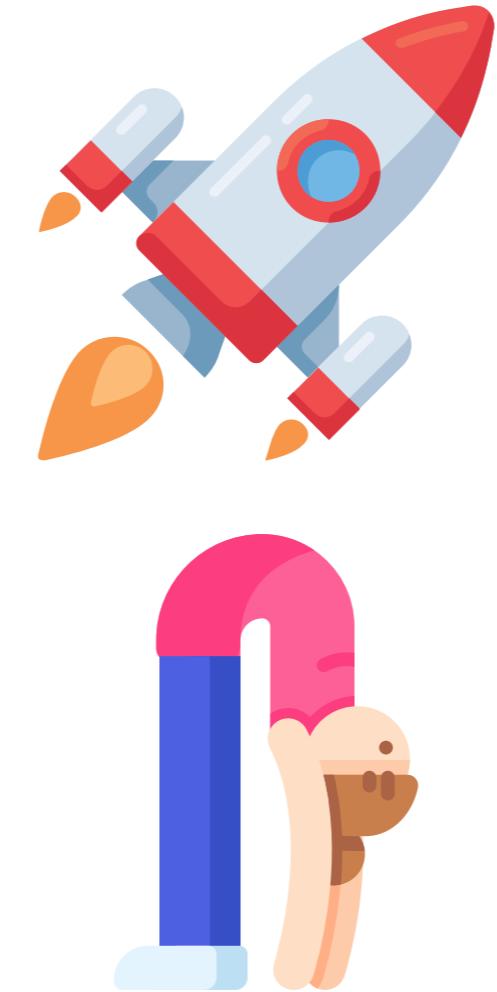
```
self = <tests.data.test_preprocessing_helpers.TestRowToList object at 0x7f4eece33b00>
```

```
def test_on_one_tab_with_missing_value(self):    # (1, 1) boundary value
    actual = row_to_list("\t4,567\n")
>   assert actual is None, "Expected: None, Actual: {}".format(actual)
E   AssertionError: Expected: None, Actual: ['', '4,567']
E   assert ['', '4,567'] is None
```

```
data/test_preprocessing_helpers.py:55: AssertionError
```

```
===== 1 failed in 0.06 seconds =====
```

Running tests using keyword expressions



The -k option

```
pytest -k "pattern"
```

- Runs all tests whose node ID matches the pattern.

The -k option

- Run the test class `TestSplitIntoTrainingAndTestingSets`.

```
pytest -k "TestSplitIntoTrainingAndTestingSets"
```

```
models/test_train.py .. [100
```

```
===== 2 passed, 14 deselected in 0.36 seconds =====
```

```
pytest -k "TestSplit"
```

```
models/test_train.py .. [100
```

```
===== 2 passed, 14 deselected in 0.36 seconds =====
```

Supports Python logical operators

```
pytest -k "TestSplit and not test_on_one_row"
```

```
models/test_train.py .
```

[100%]

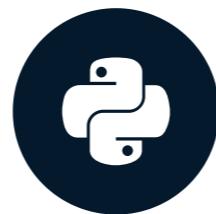
```
===== 1 passed, 15 deselected in 0.36 seconds =====
```

Let's run some tests!

UNIT TESTING FOR DATA SCIENCE IN PYTHON

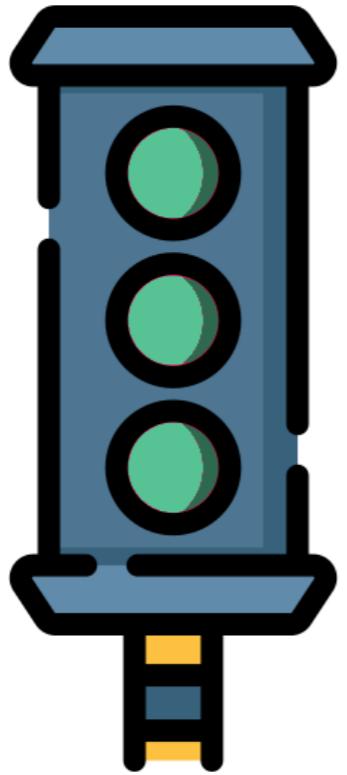
Expected failures and conditional skipping

UNIT TESTING FOR DATA SCIENCE IN PYTHON

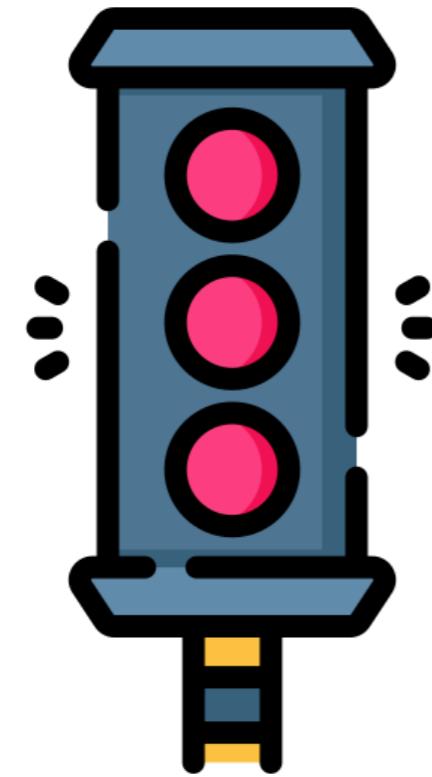


Dibya Chakravorty
Test Automation Engineer

Test suite is green when all tests pass



Test suite is red when any test fails



Implementing a function using TDD

- `train_model()` : Returns best fit line given training data.

```
import pytest

class TestTrainModel(object):
    def test_on_linear_data(self):
        ...
```

The test fails, of course!

```
pytest
```

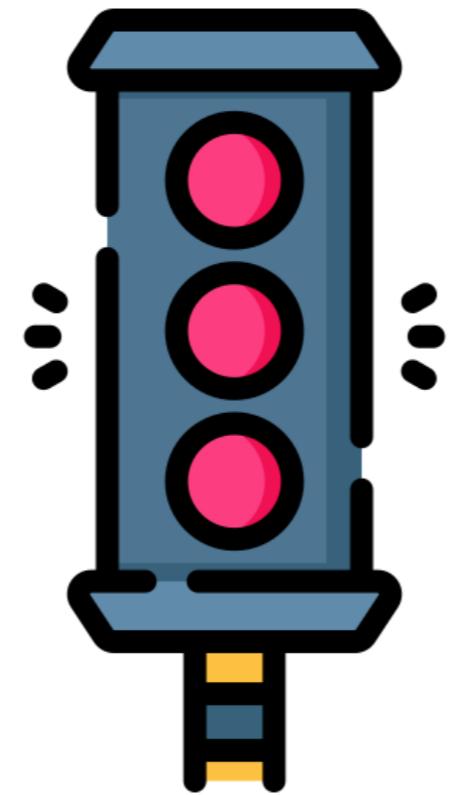
```
===== test session starts =====
data/test_preprocessing_helpers.py ..... [ 76%]
features/test_as_numpy.py . [ 82%]
models/test_train.py ..F [100%]

===== FAILURES =====
----- TestTrainModel.test_on_linear_data -----
self = <tests.models.test_train.TestTrainModel object at 0x7f5fc0f31978>

def test_on_linear_data(self):
    test_input = np.array([[1.0, 3.0], [2.0, 5.0], [3.0, 7.0]])
    expected_slope = 2.0
    expected_intercept = 1.0
>   actual_slope, actual_intercept = train_model(test_input)
E   NameError: name 'train_model' is not defined

models/test_train.py:39: NameError
===== 1 failed, 16 passed in 0.22 seconds =====
```

False alarm



xfail: marking tests as "expected to fail"

```
import pytest

class TestTrainModel(object):
    @
    def test_on_linear_data(self):
        ...
```

xfail: marking tests as "expected to fail"

```
import pytest

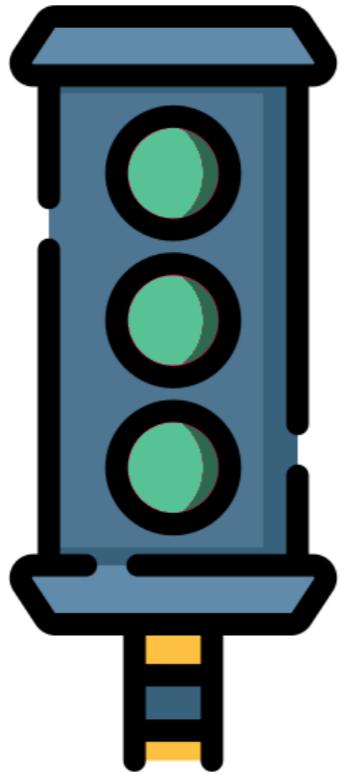
class TestTrainModel(object):
    @pytest.mark.xfail
    def test_on_linear_data(self):
        ...
```

pytest

```
===== test session starts =====
data/test_preprocessing_helpers.py ....., [ 76%]
features/test_as_numpy.py ., [ 82%]
models/test_train.py ..x, [100%]

===== 16 passed, 1 xfailed in 0.21 seconds =====
```

Test suite stays green



Expected failures, but conditionally

Tests that are expected to fail

- on certain Python versions.
- on certain platforms like Windows.

```
class TestConvertToInt(object):
    def test_with_no_comma(self):
        """Only runs on Python 2.7 or lower"""
        test_argument = "756"
        expected = 756
        actual = convert_to_int(test_argument)
        message = unicode("Expected: 2081, Actual: {0}".format(actual)) # Requires Python 2.7 or lo
        assert actual == expected, message
```

Test suite goes red on Python 3

pytest

```
===== test session starts =====
platform linux -- Python 3.6.8, pytest-4.3.1, py-1.8.0, pluggy-0.9.0

===== FAILURES =====
__ TestConvertToInt.test_with_no_comma __

self = <tests.data.test_preprocessing_helpers.TestConvertToInt object at 0x7f2c479a76a0>

def test_with_no_comma(self):
    test_argument = "756"
    expected = 756
    actual = convert_to_int(test_argument)
>     message = unicode("Expected: 2081, Actual: {0}".format(actual))
E     NameError: name 'unicode' is not defined

data/test_preprocessing_helpers.py:12: NameError
===== 1 failed, 15 passed, 1 xfailed in 0.23 seconds =====
```

skipif: skip tests conditionally

```
class TestConvertToInt(object):  
    @pytest.mark.skipif  
    def test_with_no_comma(self):  
        """Only runs on Python 2.7 or lower"""  
        test_argument = "756"  
        expected = 756  
        actual = convert_to_int(test_argument)  
        message = unicode("Expected: 2081, Actual: {0}".format(actual))  
        assert actual == expected, message
```

skipif: skip tests conditionally

```
class TestConvertToInt(object):  
    @pytest.mark.skipif(boolean_expression)  
    def test_with_no_comma(self):  
        """Only runs on Python 2.7 or lower"""  
        test_argument = "756"  
        expected = 756  
        actual = convert_to_int(test_argument)  
        message = unicode("Expected: 2081, Actual: {0}".format(actual))  
        assert actual == expected, message
```

- If `boolean_expression` is `True`, then test is skipped.

skipif when Python version is higher than 2.7

```
import sys

class TestConvertToInt(object):
    @pytest.mark.skipif(sys.version_info > (2, 7))
    def test_with_no_comma(self):
        """Only runs on Python 2.7 or lower"""
        test_argument = "756"
        expected = 756
        actual = convert_to_int(test_argument)
        message = unicode("Expected: 2081, Actual: {0}".format(actual))
        assert actual == expected, message
```

The reason argument

```
import sys

class TestConvertToInt(object):
    @pytest.mark.skipif(sys.version_info > (2, 7), reason="requires Python 2.7")
    def test_with_no_comma(self):
        """Only runs on Python 2.7 or lower"""
        test_argument = "756"
        expected = 756
        actual = convert_to_int(test_argument)
        message = unicode("Expected: 2081, Actual: {0}".format(actual))
        assert actual == expected, message
```

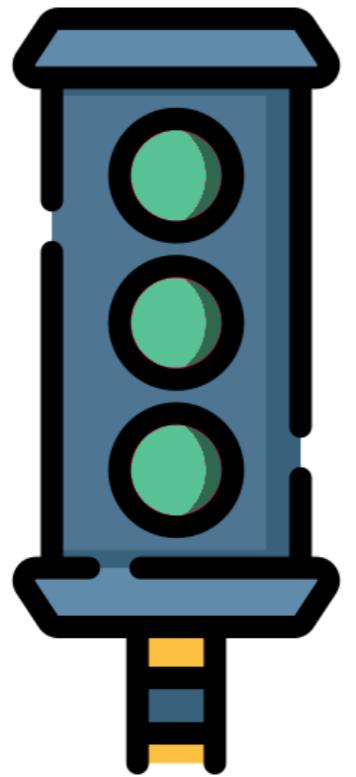
1 skipped, 1 xfailed

pytest

```
===== test session starts =====
platform linux -- Python 3.6.8, pytest-4.3.1, py-1.8.0, pluggy-0.9.0
collected 17 items

data/test_preprocessing_helpers.py s..... [ 76%]
features/test_as_numpy.py . [ 82%]
models/test_train.py ..x [100%]

===== 15 passed, 1 skipped, 1 xfailed in 0.21 seconds =====
```



Showing reason in the test result report

```
pytest -r
```

The -r option

```
pytest -r[set_of_characters]
```

Showing reason for skipping

```
pytest -rs
```

```
===== test session starts =====
platform linux -- Python 3.6.8, pytest-4.3.1, py-1.8.0, pluggy-0.9.0
collected 17 items

data/test_preprocessing_helpers.py s..... [ 76%]
features/test_as_numpy.py . [ 82%]
models/test_train.py ..x [100%]

===== short test summary info =====
SKIPPED [1] tests/data/test_preprocessing_helpers.py:8: Requires Python 2.7 or lower

===== 15 passed, 1 skipped, 1 xfailed in 0.21 seconds =====
```

Optional reason argument to xfail

```
import pytest

class TestTrainModel(object):

    @pytest.mark.xfail
    def test_on_linear_data(self):
        ...
```

Optional reason argument to xfail

```
import pytest

class TestTrainModel(object):
    @pytest.mark.xfail(reason="Using TDD, train_model() is not implemented")
    def test_on_linear_data(self):
        ...
```

Showing reason for xfail

```
pytest -rx
```

```
===== test session starts =====
platform linux -- Python 3.6.8, pytest-4.3.1, py-1.8.0, pluggy-0.9.0
collected 17 items

data/test_preprocessing_helpers.py s..... [ 76%]
features/test_as_numpy.py . [ 82%]
models/test_train.py ..x [100%]

===== short test summary info =====
XFAIL models/test_train.py::TestTrainModel::test_on_linear_data
Using TDD, train_model() is not implemented

===== 15 passed, 1 skipped, 1 xfailed in 0.28 seconds =====
```

Showing reason for both skipped and xfail

```
pytest -rsx
```

```
===== test session starts =====
platform linux -- Python 3.6.8, pytest-4.3.1, py-1.8.0, pluggy-0.9.0
rootdir: /home/dibya/startup-code/datacamp/univariate_linear_regression, inifile:
collected 17 items

data/test_preprocessing_helpers.py s..... [ 76%]
features/test_as_numpy.py . [ 82%]
models/test_train.py ..x [100%]

===== short test summary info =====
SKIPPED [1] tests/data/test_preprocessing_helpers.py:8: Requires Python 2.7 or lower
XFAIL models/test_train.py::TestTrainModel::test_on_linear_data
    Using TDD, train_model() is not implemented

===== 15 passed, 1 skipped, 1 xfailed in 0.22 seconds =====
```

Skipping/x failing entire test classes

```
@pytest.mark.xfail(reason="Using TDD, train_model() is not implemented")  
class TestTrainModel(object):
```

...

```
@pytest.mark.skipif(sys.version_info > (2, 7), reason="requires Python 2.7")  
class TestConvertToInt(object):
```

...

**Let's practice
xfailing and
skipping!**

UNIT TESTING FOR DATA SCIENCE IN PYTHON

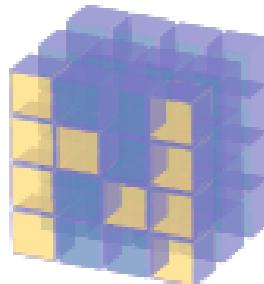
Continuous integration and code coverage

UNIT TESTING FOR DATA SCIENCE IN PYTHON



Dibya Chakravorty
Test Automation Engineer

Code coverage and build status badges



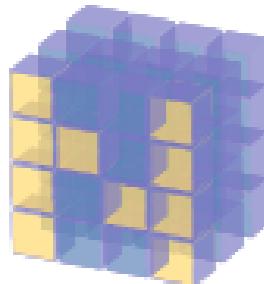
NumPy



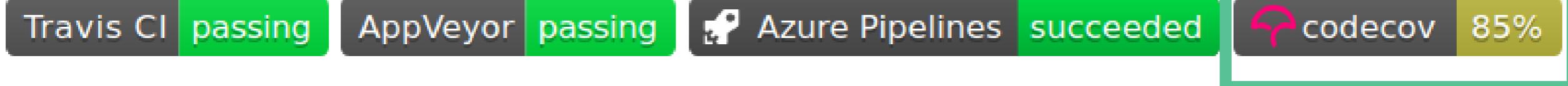
NumPy is the fundamental package needed for scientific computing with Python.

- **Website (including documentation):** <https://www.numpy.org>
- **Mailing list:** <https://mail.python.org/mailman/listinfo/numpy-discussion>
- **Source:** <https://github.com/numpy/numpy>

Code coverage and build status badges



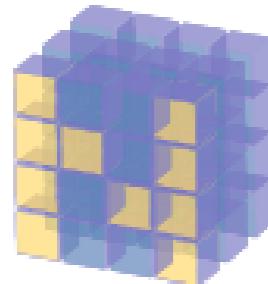
NumPy



NumPy is the fundamental package needed for scientific computing with Python.

- **Website (including documentation):** <https://www.numpy.org>
- **Mailing list:** <https://mail.python.org/mailman/listinfo/numpy-discussion>
- **Source:** <https://github.com/numpy/numpy>

Code coverage and build status badges



NumPy



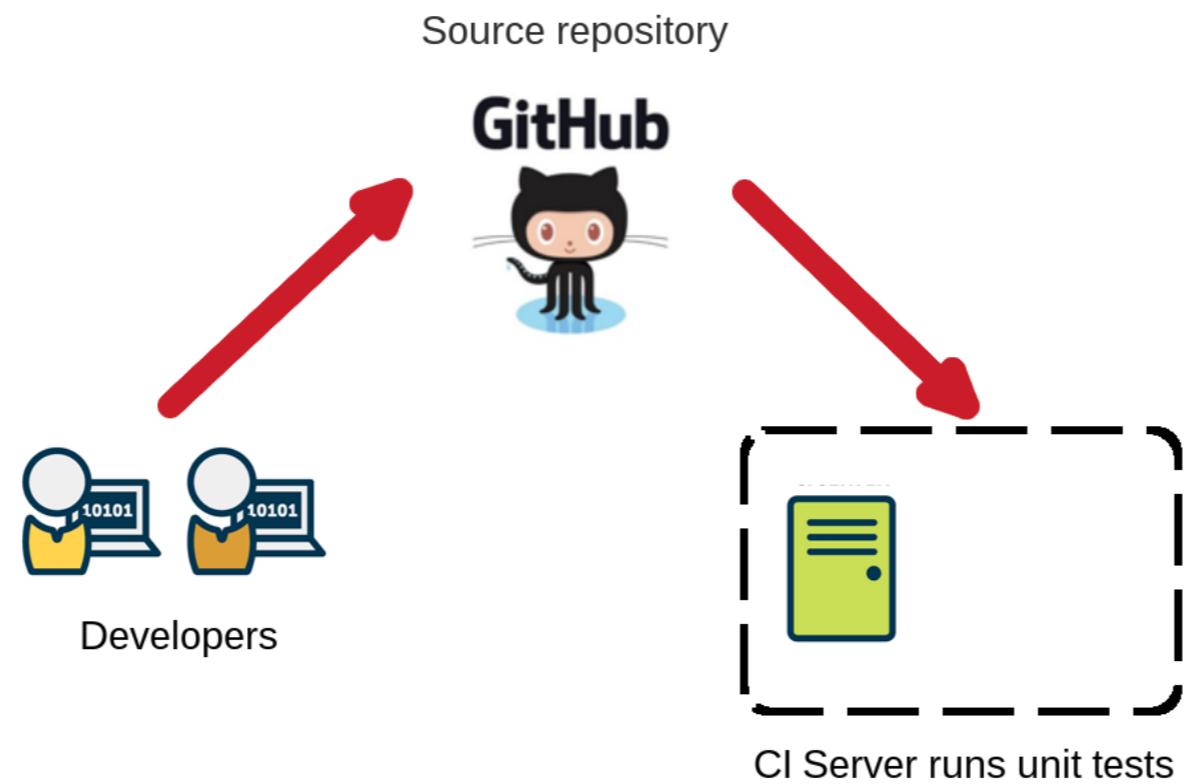
NumPy is the fundamental package needed for scientific computing with Python.

- **Website (including documentation):** <https://www.numpy.org>
- **Mailing list:** <https://mail.python.org/mailman/listinfo/numpy-discussion>
- **Source:** <https://github.com/numpy/numpy>

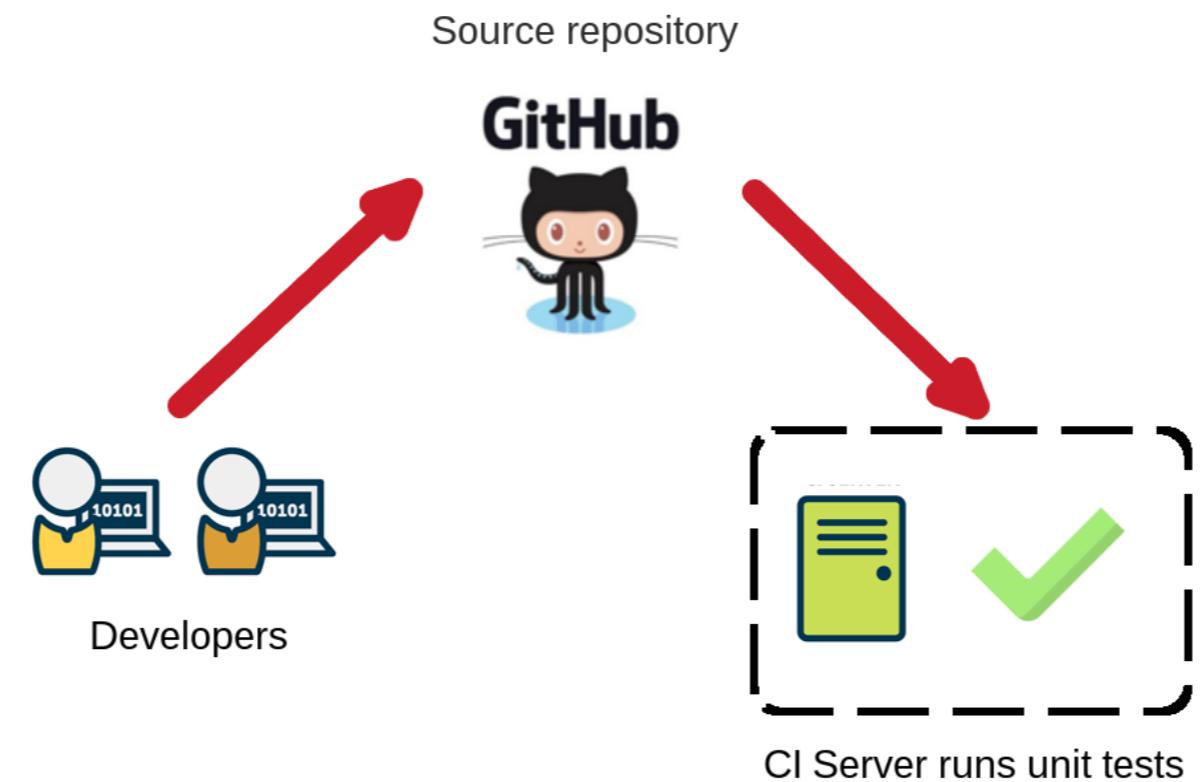
The build status badge



The build status badge

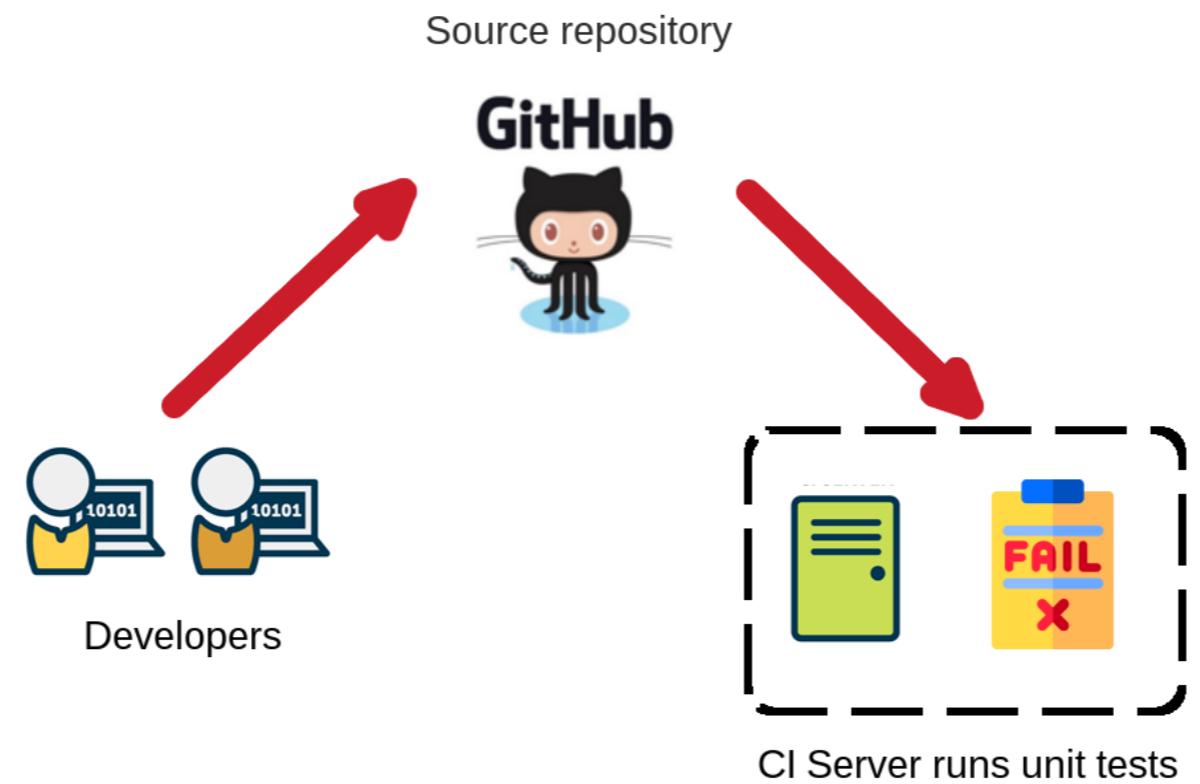


Build passing = Stable project



build passing

Build failing = Unstable project



build failing

CI server



Travis CI

Step 1: Create a configuration file

```
repository root  
|-- src  
|-- tests  
|-- .travis.yml
```

Step 1: Create a configuration file

- Contents of `.travis.yml`.

```
language: python
python:
  - "3.6"
install:
  - pip install -e .
script:
  - pytest tests
```

Step 2: Push the file to GitHub

```
git add .travis.yml  
git push origin master
```

Step 3: Install the Travis CI app

The screenshot shows a GitHub user profile for 'gutfeeling'. The top navigation bar includes 'Pull requests', 'Issues', 'Marketplace' (which is highlighted with a green border), and 'Explore'. The left sidebar lists repositories such as 'datacamp/courses-unit-testing-i...', 'gutfeeling/univariate-linear-regr...', and 'gutfeeling/practical_rl_for_coders'. The main content area displays three recent activity items:

- sofiass forked sofiass/beginner_nlp from gutfeeling/beginner_nlp** yesterday
- sofiass and akashgudadhe1 starred 1 repository** yesterday
- akashgudadhe1 forked akashgudadhe1/beginner_nlp from gutfeeling/beginner_nlp** yesterday

Each item shows the repository details: 'gutfeeling/beginner_nlp', 'A curated list of beginner resources in Natural Language Processing', '★ 367 Updated Jun 11', and a 'Star' button. A GitHub Sponsors Matching Fund banner is visible on the right, and a 'Discover repositories' section lists other projects like 'nickdavidhaynes/spacy-cld' and 'PacktPublishing/Python-Machine-Learning-Cookbook'.

Step 3: Install the Travis CI app

The screenshot shows the GitHub Marketplace search results for the term "travis". The search bar at the top contains the query "travis". To the left, there is a sidebar with categories: API management, Chat, Code quality, Code review, Continuous integration, Dependency management, Deployment, IDEs, Learning, and Localization. The main search results area displays one result: "Travis CI" with a green checkmark icon, followed by the tagline "Test and deploy with confidence". This result is highlighted with a green border. At the bottom right of the results area, there are "Previous" and "Next" navigation buttons.

Marketplace / Search results

Categories

API management

Chat

Code quality

Code review

Continuous integration

Dependency management

Deployment

IDEs

Learning

Localization

Search or jump to... / Pull requests Issues Marketplace Explore

travis

1 result for "travis"

Travis CI Test and deploy with confidence

Previous Next

Step 3: Install the Travis CI app

Pricing and setup

Open Source	\$0
We offer free CI for Open Source projects	
ONE <small>Free Trial</small>	\$69 / month
Unlimited builds, 1 job at a time. Ideal for hobby and small projects.	
THREE <small>Free Trial</small>	\$199 / month
Unlimited builds, 3 jobs at a time. Best suited for small teams.	
SIX <small>Free Trial</small>	\$349 / month
Unlimited builds, 6 jobs at a time. Great for growing teams.	

Travis CI
Open Source 

We offer free CI for Open Source projects

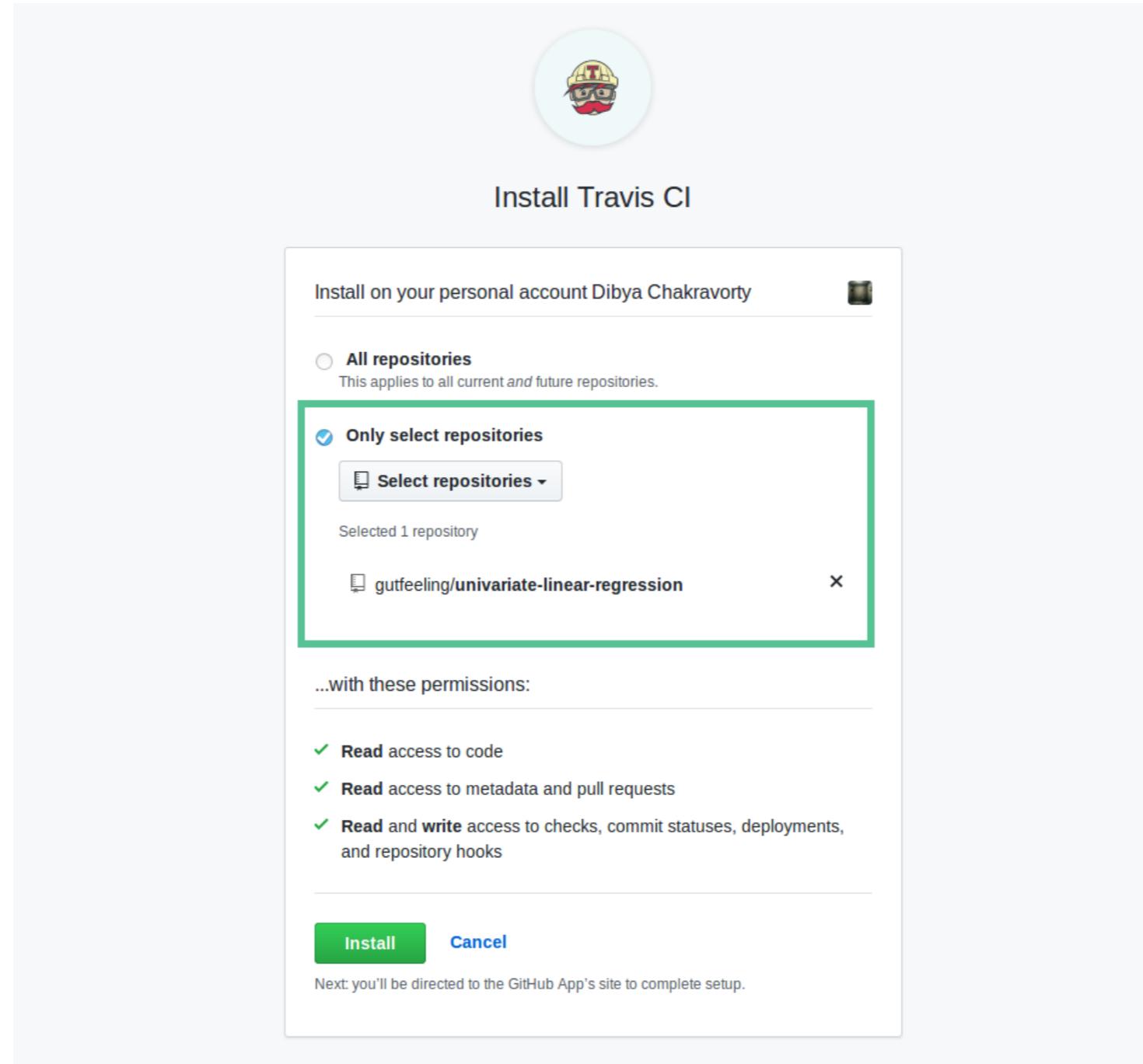
- ✓ Unlimited public repositories
- ✓ Unlimited collaborators

Account: [gutfeeling](#) ▾

[Install it for free](#) Next: Confirm your installation location.

Travis CI is provided by a third-party and is governed by separate [terms of service](#), [privacy policy](#), and [support contact](#).

Step 3: Install the Travis CI app



Step 3: Install the Travis CI app

Travis CI About Us Plans & Pricing Enterprise Help



We're so glad you're here!

Please sign in to view your repositories.

Sign in with GitHub

Every commit leads to a build

The screenshot shows the Travis CI web interface. At the top, there's a navigation bar with links for "Travis CI", "Dashboard", "Changelog", "Documentation", and "Help". On the far right, there's a user profile icon and a dropdown menu.

In the main area, there's a search bar with the placeholder "Search all repositories" and a magnifying glass icon. Below it, a section titled "My Repositories" shows one repository: "Running (1/1)".

The central part of the screen displays the repository "gutfeeling / univariate-linear-regression". The repository name is followed by a green "build passing" badge with a small circular icon. Below the repository name, there are tabs for "Current", "Branches", "Build History", and "Pull Requests", with "Current" being the active tab.

The "Current" tab shows a single build entry:

- Job ID: #13
- Status: started
- Duration: 28 sec
- Commit: 8f6c815
- Compare: 3695237..8f6c815
- Branch: master
- Author: Dibya Chakravorty
- Python version: 3.6

At the bottom of the build card, there are two buttons: "Job log" and "View config".

Step 4: Showing the build status badge

The screenshot shows the Travis CI dashboard interface. At the top, there is a navigation bar with links for "Travis CI", "Dashboard", "Changelog", "Documentation", and "Help". On the far right, there is a user profile icon and a dropdown menu.

In the main area, there is a search bar labeled "Search all repositories" with a magnifying glass icon. Below it, a section titled "My Repositories" shows one repository: "gutfeeling / univariate-linear-regression".

The repository page has tabs for "Current", "Branches", "Build History", and "Pull Requests". The "Current" tab is selected. A green box highlights the "build passing" badge, which consists of a green circle with a white checkmark and the text "build passing".

The build details for job #13 are shown. The commit message is "master Remove useless comment". The build status is "started" and is currently "Running for 28 sec". A "Cancel build" button is available. The build is run on Python 3.6 by Dibya Chakravorty. The commit hash is 8f6c815. The branch is master. The comparison is between commits 3695237 and 8f6c815.

At the bottom of the build card, there are links for "Job log" and "View config".

Step 4: Showing the build status badge

The screenshot shows the Travis CI web interface. At the top, there's a navigation bar with links for "Travis CI", "Dashboard", "Changelog", "Documentation", and "Help". On the far right, there's a user profile icon. Below the navigation, a search bar says "Search all repositories" with a magnifying glass icon. The main area shows "My Repositories" with one repository listed: "gutfeeling/univariate-linear-reg" (branch: master, build #13, duration: 42 sec, finished less than a minute ago). To the right of this repository card is a green "build passing" badge with a small cat icon. A modal window titled "Status Image" is open over the repository card. It has a dropdown menu set to "master" under "BRANCH". Another dropdown menu is set to "MARKDOWN". Below these dropdowns is a code block containing a Travis CI build status badge configuration:

```
[![Build Status](https://travis-ci.com/gutfeeling/univariate-linear-reg.svg?branch=master)](https://travis-ci.com/gutfeeling/univariate-linear-reg)
```

At the bottom of the modal, there are two buttons: "Job log" and "View config".

Step 4: Showing the build status badge

The screenshot shows a GitHub repository page for a user named 'gutfeeling'. The repository has a green progress bar at the top indicating the status of the latest commit. Below the progress bar, the commit history is listed:

Commit	Message	Time Ago
Latest commit e08b854	Remove codecov badge	now
data/raw	Add tests	2 months ago
notebooks	Fix jupyter notebook	2 months ago
src	remove fake data generator	4 days ago
tests	Better test for float valued string	4 days ago
.gitignore	Improved gitignore	3 months ago
.travis.yml	Remove comments	4 days ago
README.md	Remove codecov badge	now
setup.py	Remove useless comment	3 minutes ago

Below the commit history, the 'README.md' file is shown. It contains a green rectangular badge with a white border and the text 'build passing' inside it.

Code coverage



- code coverage = $\frac{\text{num lines of application code that ran during testing}}{\text{total num lines of application code}} \times 100$
- Higher percentages (75% and above) indicate well tested code.

Codecov



Step 1: Modify the Travis CI configuration file

- File: `.travis.yml`

```
language: python
python:
  - "3.6"
install:
  - pip install -e .
script:
  - pytest tests
```

Step 1: Modify the Travis CI configuration file

- File: `.travis.yml`

```
language: python
python:
  - "3.6"
install:
  - pip install -e .
  - pip install pytest-cov codecov      # Install packages for code coverage report
script:
  - pytest tests
```

Step 1: Modify the Travis CI configuration file

- File: `.travis.yml`

```
language: python
python:
  - "3.6"
install:
  - pip install -e .
  - pip install pytest-cov codecov      # Install packages for code coverage report
script:
  - pytest --cov=src tests             # Point to the source directory
```

Step 1: Modify the Travis CI configuration file

- File: `.travis.yml`

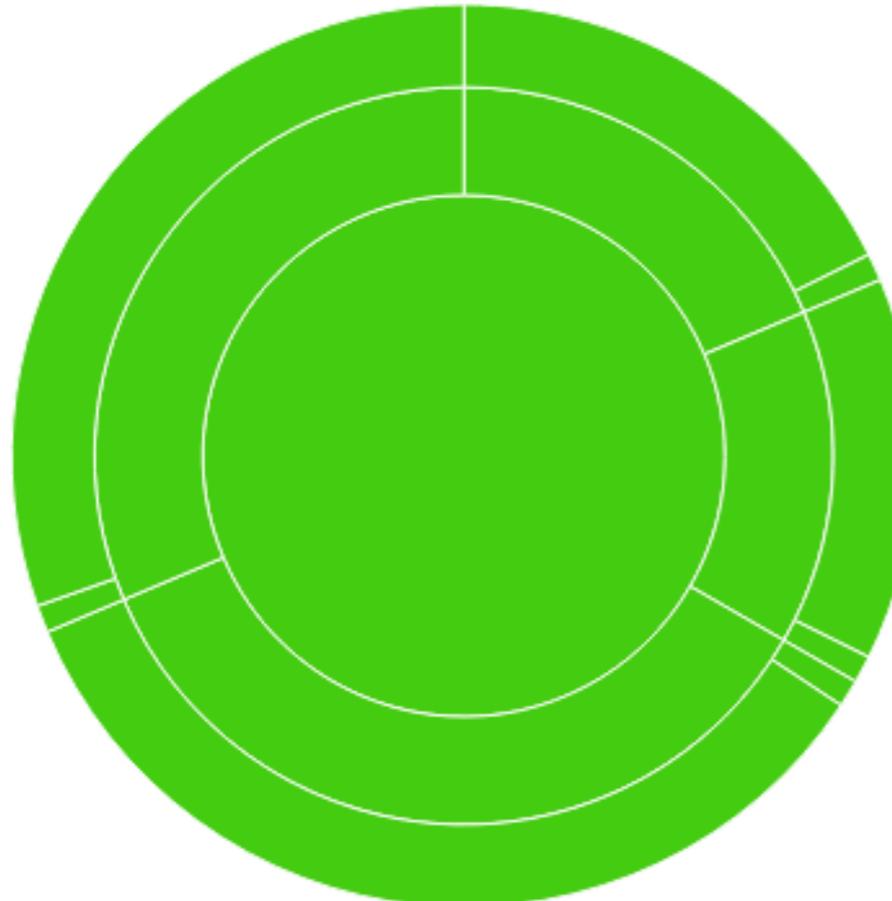
```
language: python
python:
  - "3.6"
install:
  - pip install -e .
  - pip install pytest-cov codecov      # Install packages for code coverage report
script:
  - pytest --cov=src tests             # Point to the source directory
after_success:
  - codecov                         # uploads report to codecov.io
```

Step 2: Install codecov

A screenshot of the GitHub Marketplace search results page. At the top, there is a navigation bar with icons for GitHub, a search bar containing "Search or jump to...", and links for Pull requests, Issues, Marketplace, and Explore. On the right side of the navigation bar are icons for notifications, a plus sign, and a user profile. Below the navigation bar, the URL "Marketplace / Search results" is visible. To the left, a sidebar lists categories: API management, Chat, Code quality, Code review, Continuous integration, Dependency management, Deployment, IDEs, Learning, and Localization. In the center, a search bar contains the query "codecov". Below the search bar, the text "1 result for 'codecov'" is displayed. A single result card is shown, highlighted with a green border. The card features the GitHub logo, the text "Codecov ✅", and the description "Group, merge, archive and compare coverage reports". At the bottom of the result card are "Previous" and "Next" buttons.

Commits lead to coverage report at codecov.io

COVERAGE SUNBURST



/

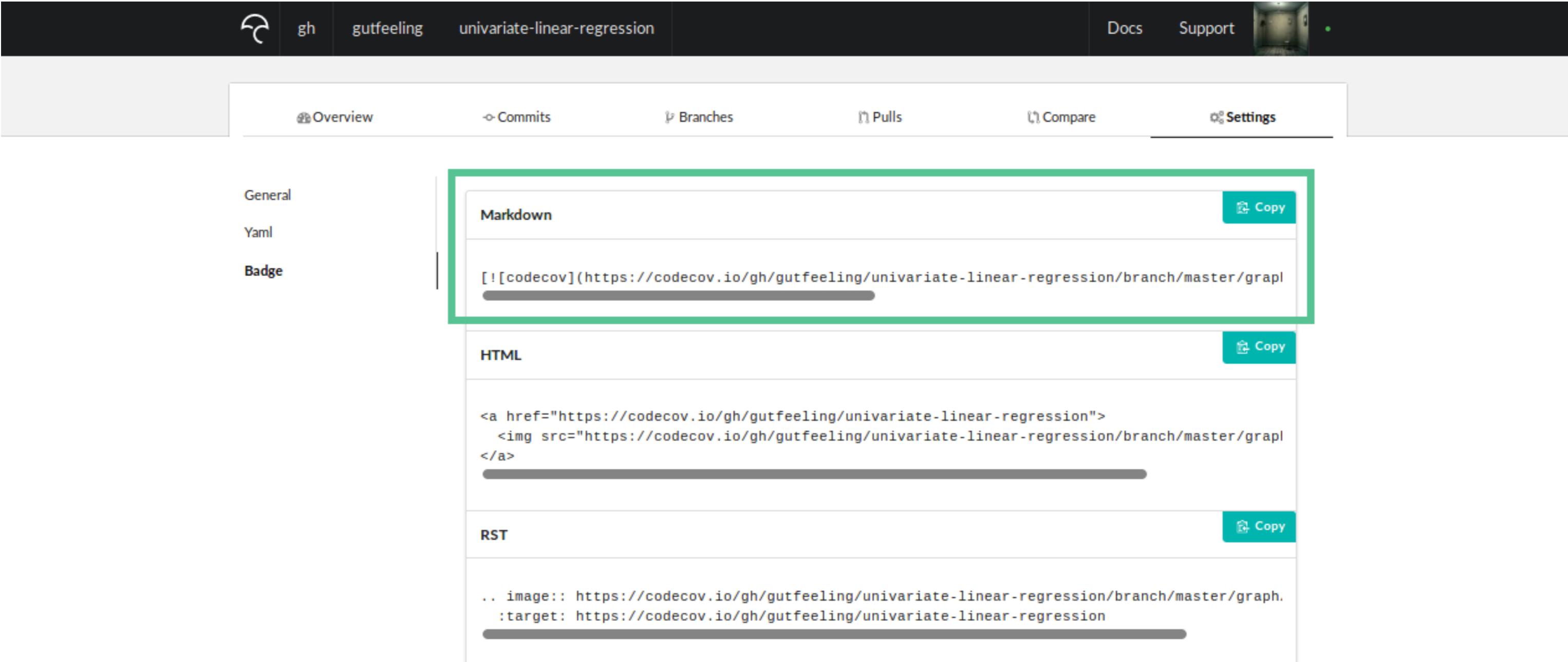
?

ALL RECENT COMMITS

-  [Fix side effects and call order in mocks](#) Browse Report
gutfeeling 7 days ago ⌛ master ⚙ 298091d ✓ CI Passed
-  [Add pytest-mock](#) Browse Report
gutfeeling 7 days ago ⌛ master ⚙ 02e714a ✓ CI Passed
-  [Add test for preprocessing function](#) Browse Report
gutfeeling 7 days ago ⌛ master ⚙ b8dfc76 ✓ CI Passed
-  [Add baseline plots and add pytest-mpl requirement](#) Browse Report
gutfeeling 7 days ago ⌛ master ⚙ 752e0e0 ✓ CI Passed
-  [Merge remote-tracking branch 'origin/master'](#) Browse Report
gutfeeling 7 days ago ⌛ master ⚙ 7ef5ef2 ✓ CI Passed
-  [Add codecov badge](#) Browse Report
gutfeeling 10 days ago ⌛ master ⚙ 960fa3f ✓ CI Passed

[View all recent commits](#)

Step 3: Showing the badge in GitHub



The screenshot shows a GitHub repository page for 'univariate-linear-regression' owned by 'gutfeeling'. The 'Settings' tab is selected. On the left, there's a sidebar with 'General', 'Yaml', and 'Badge' sections. The 'Badge' section is highlighted with a green box. It contains three copy buttons: 'Markdown', 'HTML', and 'RST'. The 'Markdown' section shows the code: `[![codecov](https://codecov.io/gh/gutfeeling/univariate-linear-regression/branch/master/graph)](https://codecov.io/gh/gutfeeling/univariate-linear-regression)`. The 'HTML' section shows: ``. The 'RST' section shows: `.. image:: https://codecov.io/gh/gutfeeling/univariate-linear-regression/branch/master/graph.
:target: https://codecov.io/gh/gutfeeling/univariate-linear-regression`.

Step 3: Showing the badge in GitHub

gutfeeling Update README.md Latest commit 0d34ed3 4 minutes ago

data Match the course code 12 minutes ago

notebooks Match the course code 12 minutes ago

src Match the course code 12 minutes ago

tests Match the course code 12 minutes ago

.gitignore Improved gitignore 4 months ago

.travis.yml Remove comments 2 months ago

README.md Update README.md 4 minutes ago

setup.py Add pytest-mock 2 months ago

README.md 

build passing  codecov 90%

This repository holds the code for the DataCamp course Unit Testing for Data Science in Python by Dibya Chakravorty.

Let's practice CI and code coverage!

UNIT TESTING FOR DATA SCIENCE IN PYTHON