

# 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`  $\iff$  `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):
    # Always put the argument object
    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()
        ...
```

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

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

    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 the DataCamp interface for an exercise. On the left, there's a sidebar with 'Exercise' and 'Instructions 2/2' (50 XP). The main area displays a code editor for 'script.py' with line numbers 1 through 8. Below the editor is an 'IPython Shell' window. The shell shows 'In [1]:' and a green box highlights the output area, which is currently empty. The 'Run Code' button is visible in the bottom right of the editor area.

# 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



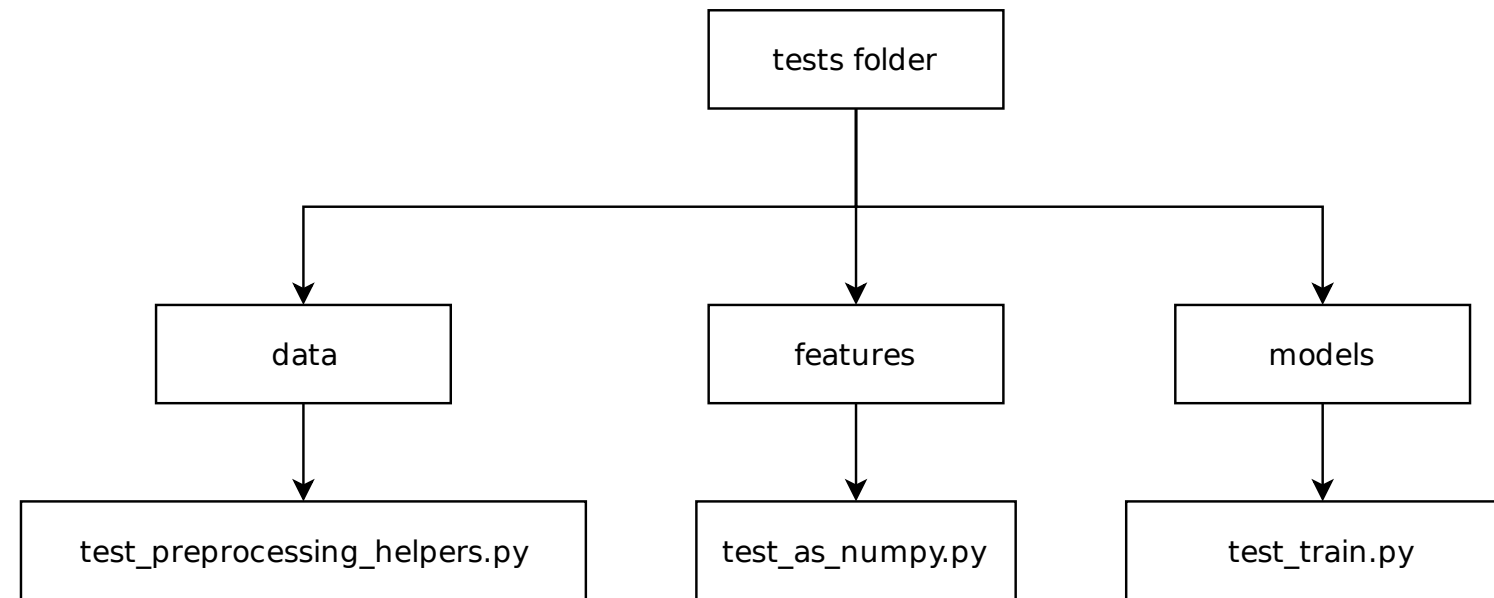
**Dibya Chakravorty**  
Test Automation Engineer

# Test organization

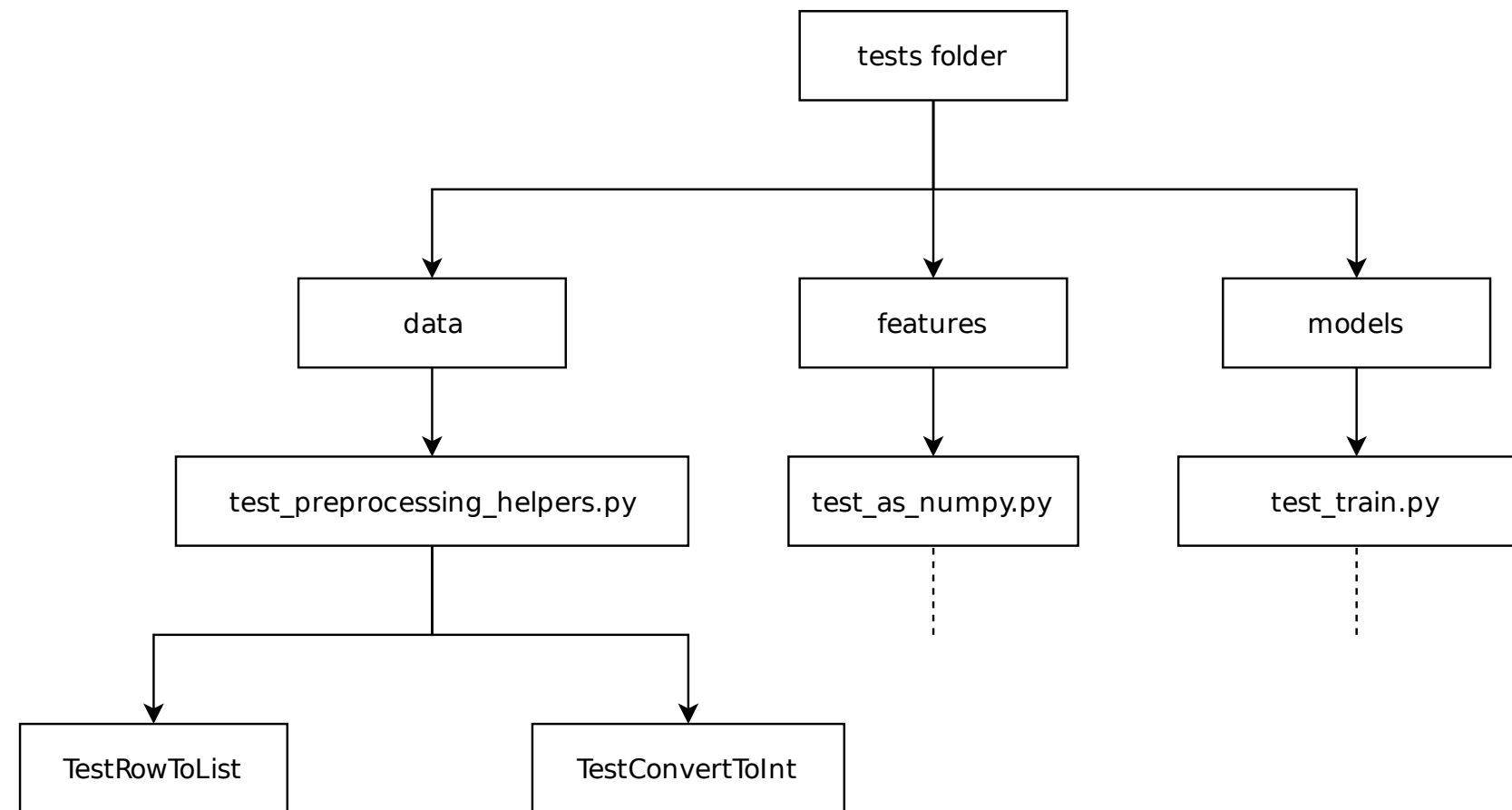
tests folder



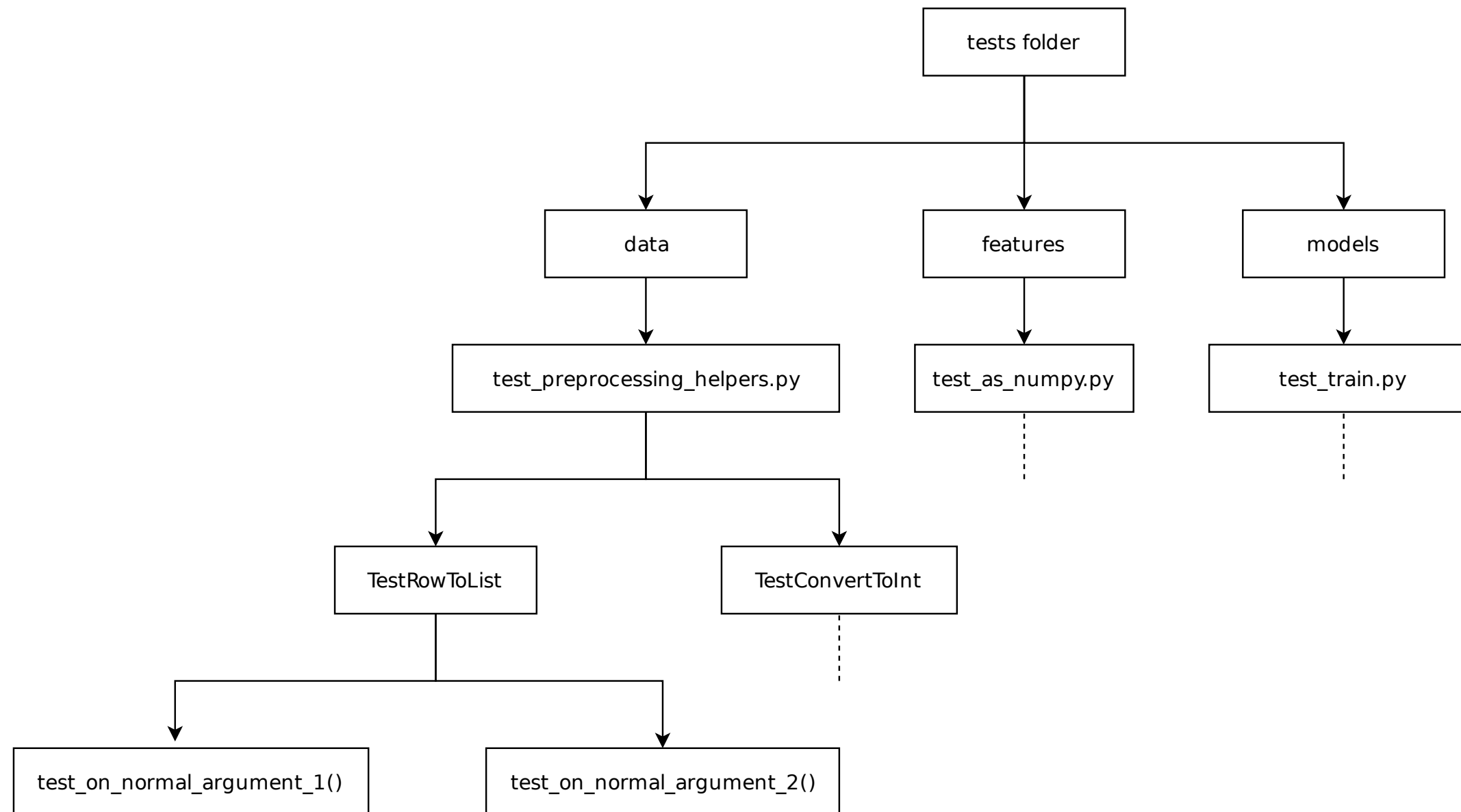
# Test organization



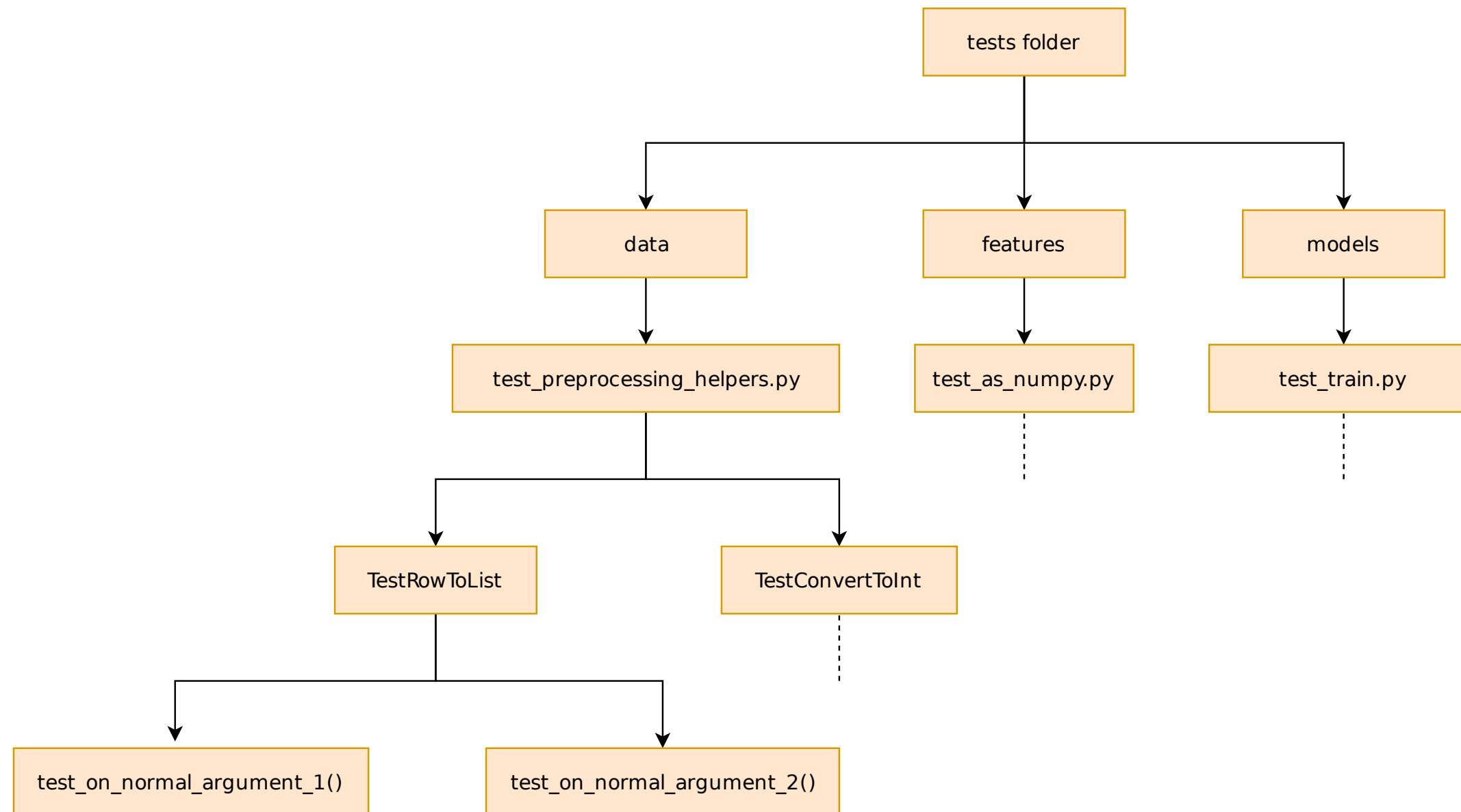
# Test organization



# 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%]

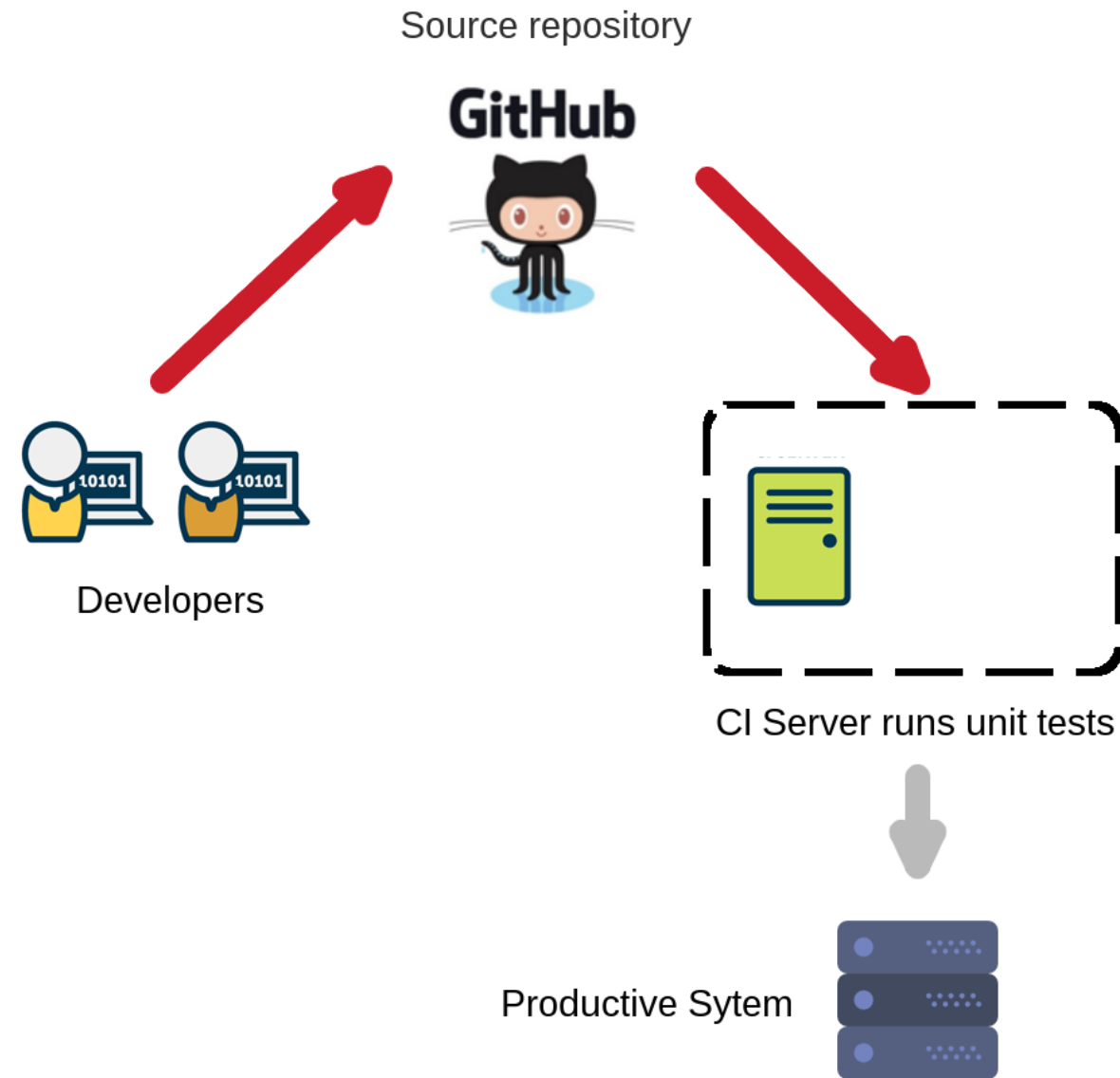
===== FAILURES =====
----- TestRowToList.test_on_one_tab_with_missing_value -----

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

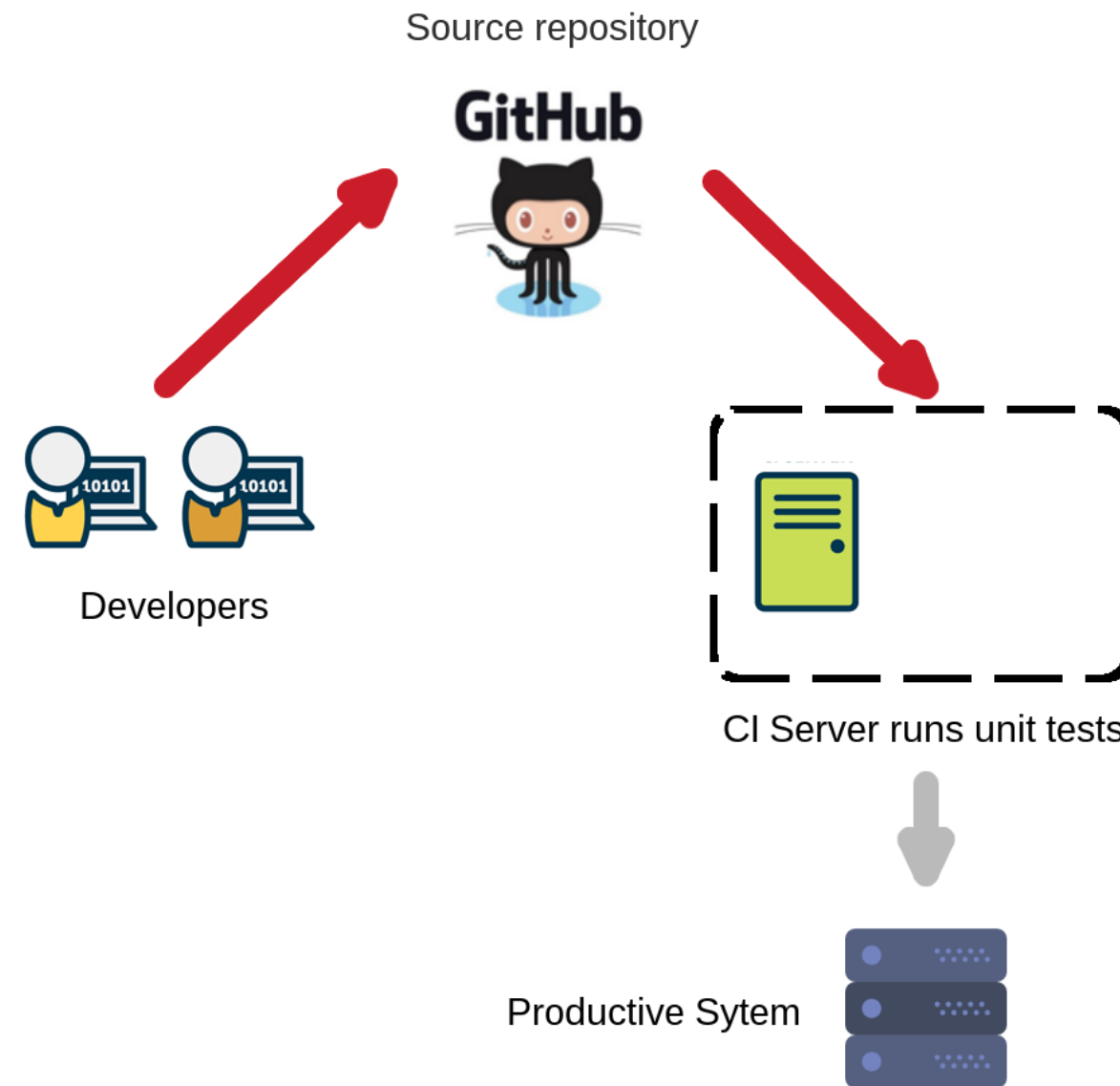
    def test_on_one_tab_with_missing_value(self):    # (1, 1) boundary value
        actual = row_to_list("\t4,567\n")
>       assert actual is None, "Expected: None, Actual: {0}".format(actual)
E       AssertionError: Expected: None, Actual: ['', '4,567']
E       assert ['', '4,567'] is None

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

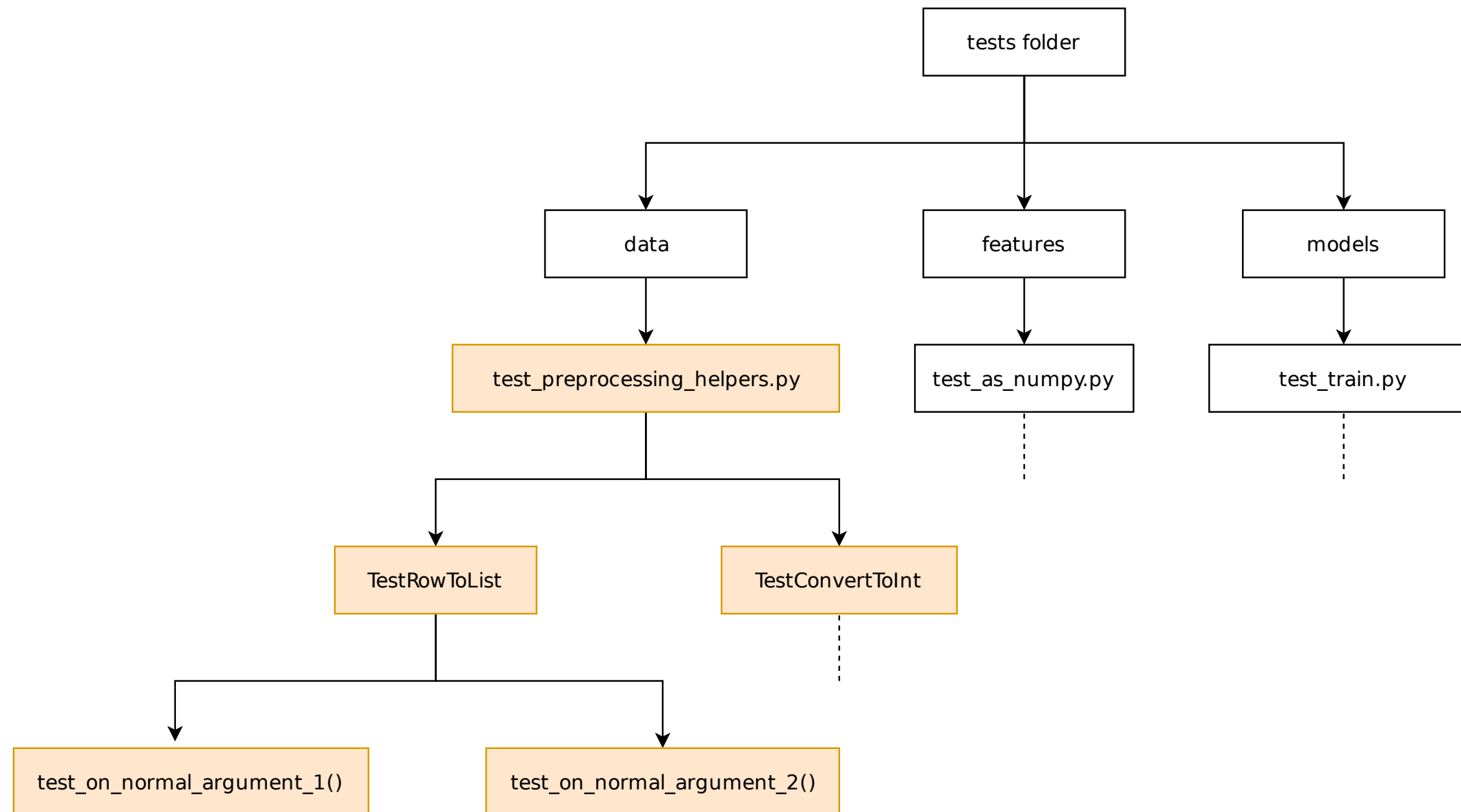
===== FAILURES =====
----- TestRowToList.test_on_one_tab_with_missing_value -----

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: {0}".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%]

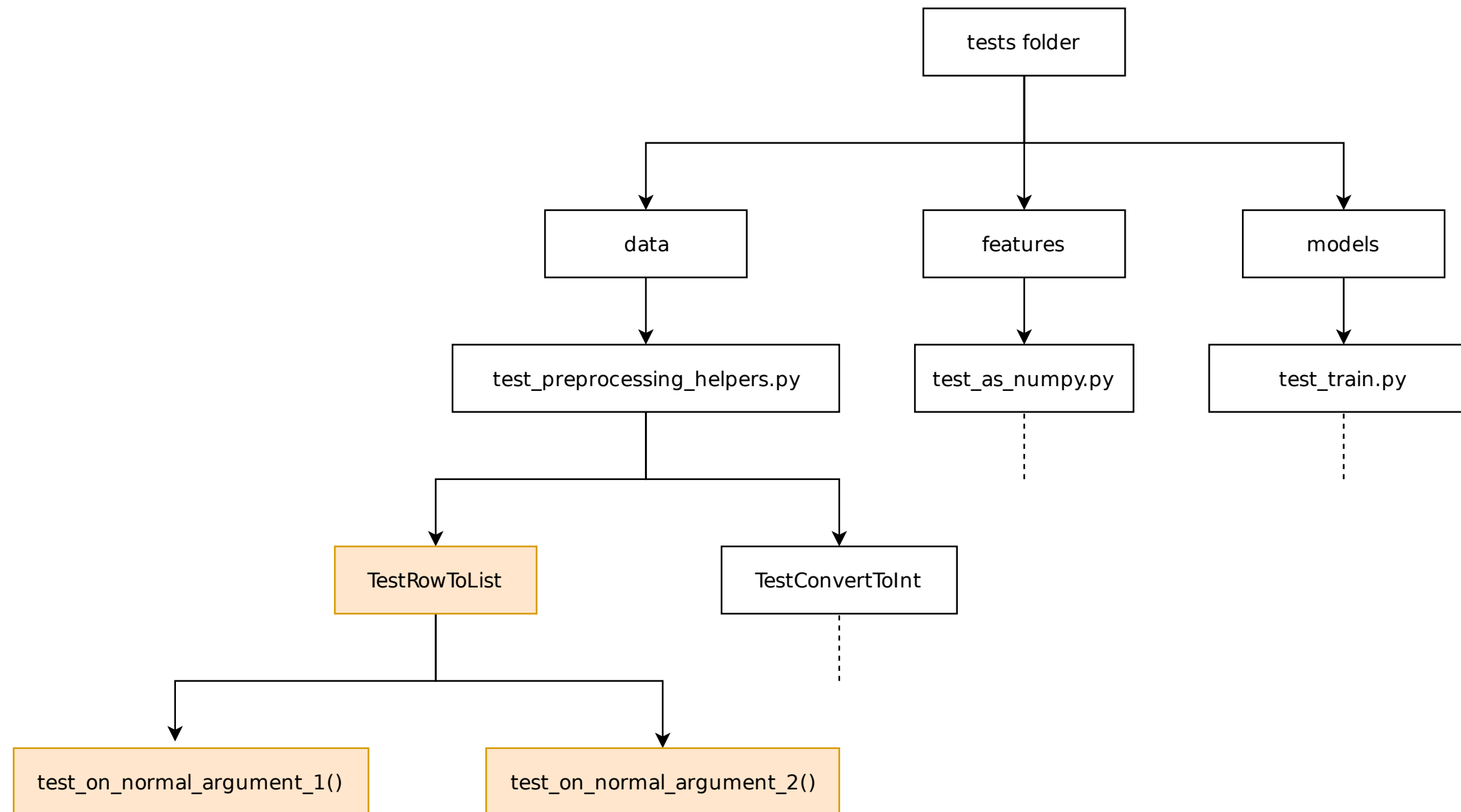
===== FAILURES =====
----- TestRowToList.test_on_one_tab_with_missing_value -----

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: {0}".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%]

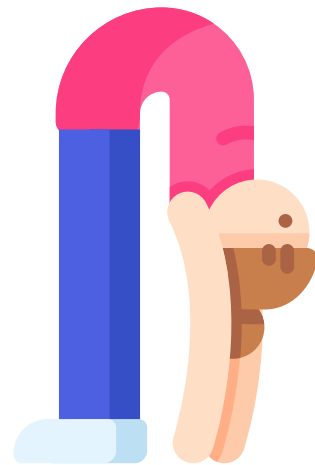
===== FAILURES =====
----- TestRowToList.test_on_one_tab_with_missing_value -----

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: {0}".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%]
```

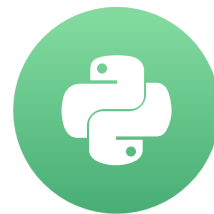
```
===== 1 passed, 15 deselected in 0.36 seconds =====
```

# Let's run some tests!

UNIT TESTING FOR DATA SCIENCE IN PYTHON

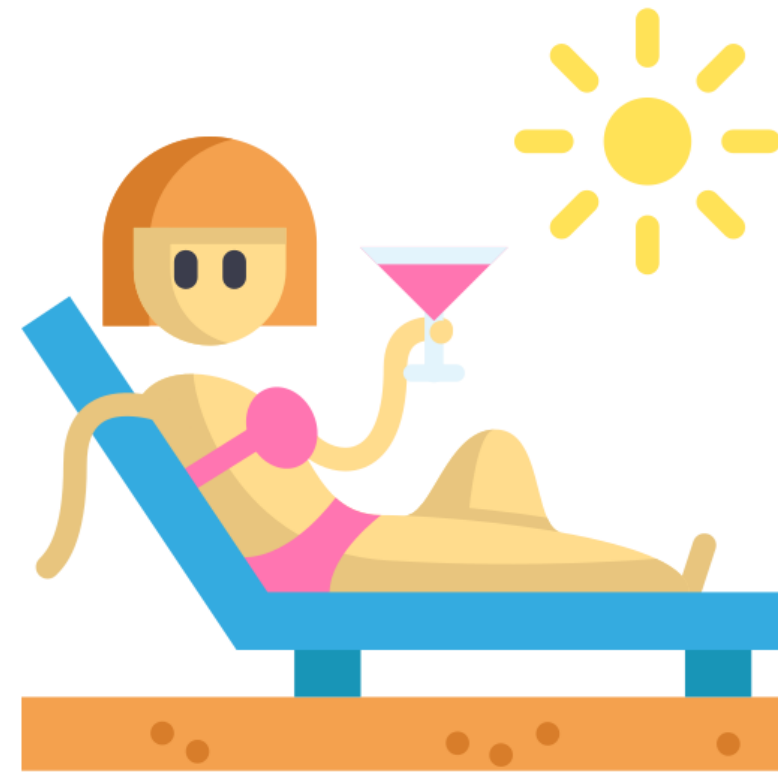
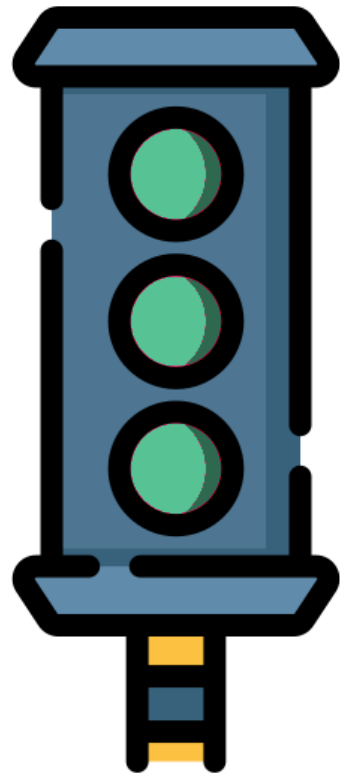
# Expected failures and conditional skipping

UNIT TESTING FOR DATA SCIENCE IN PYTHON

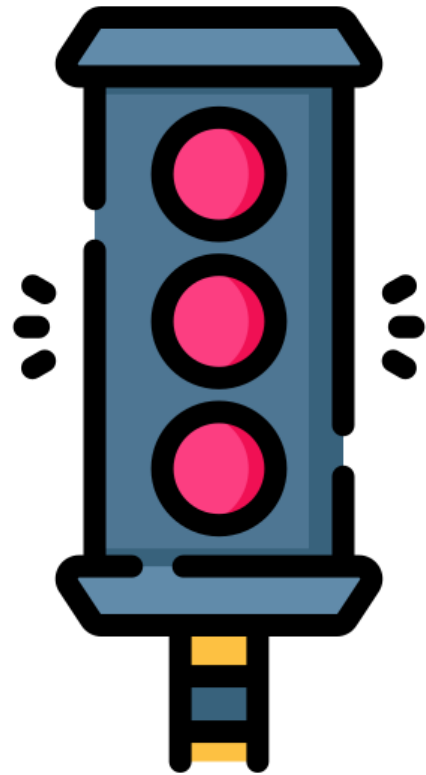


**Dibya Chakravorty**  
Test Automation Engineer

# 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%]

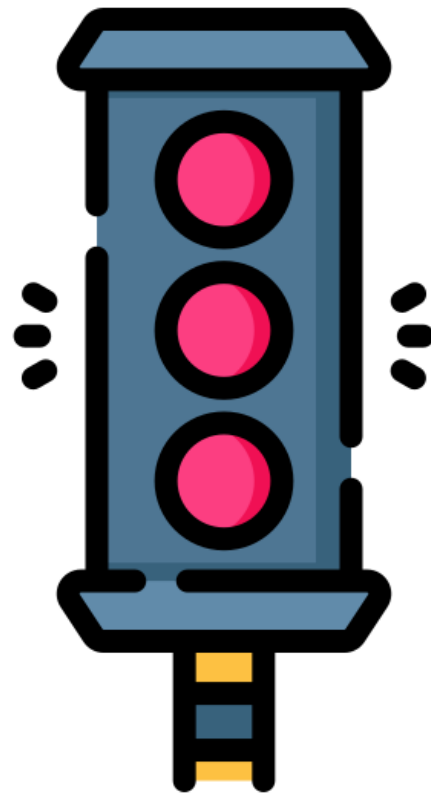
===== FAILURES =====
----- TestTrainModel.test_on_linear_data -----

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

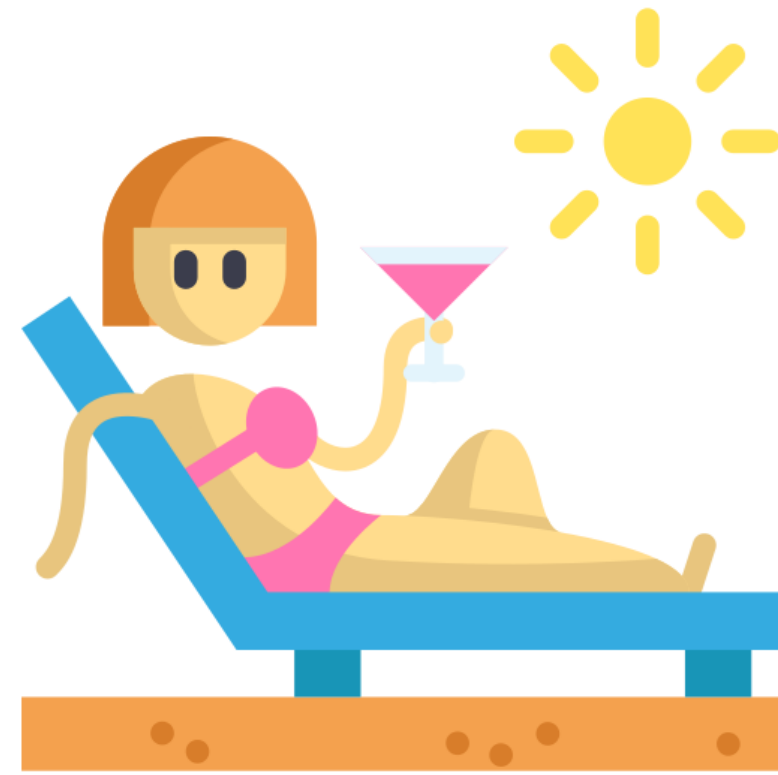
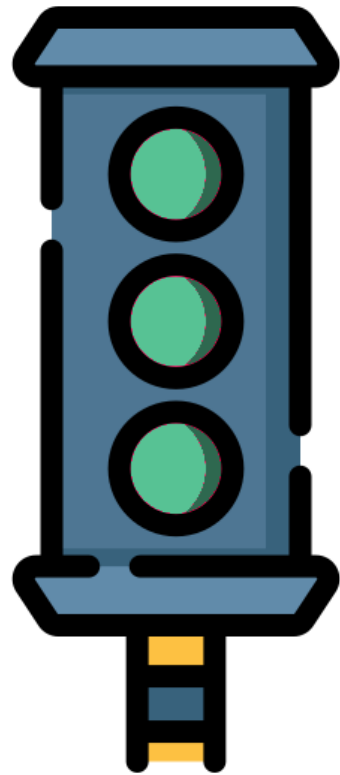
class TestTrainModel(object):
    @pytest.mark.xfail
    def test_on_linear_data(self):
        ...
```

```
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 lower
        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: {}".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
```

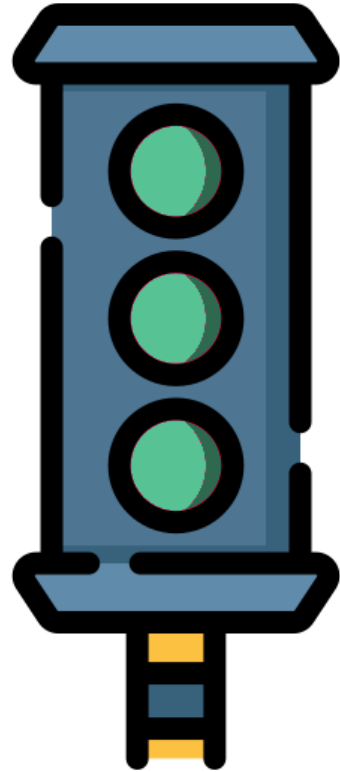
# 1 skipped, 1 xfailed

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/xfailing 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):
```

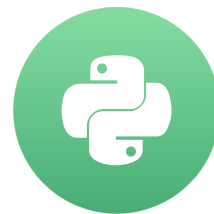
```
...
```

# 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



---

Travis CI **passing** AppVeyor **passing**  Azure Pipelines **succeeded**  codecov **85%**

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>



# Code coverage and build status badges



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>

# Code coverage and build status badges



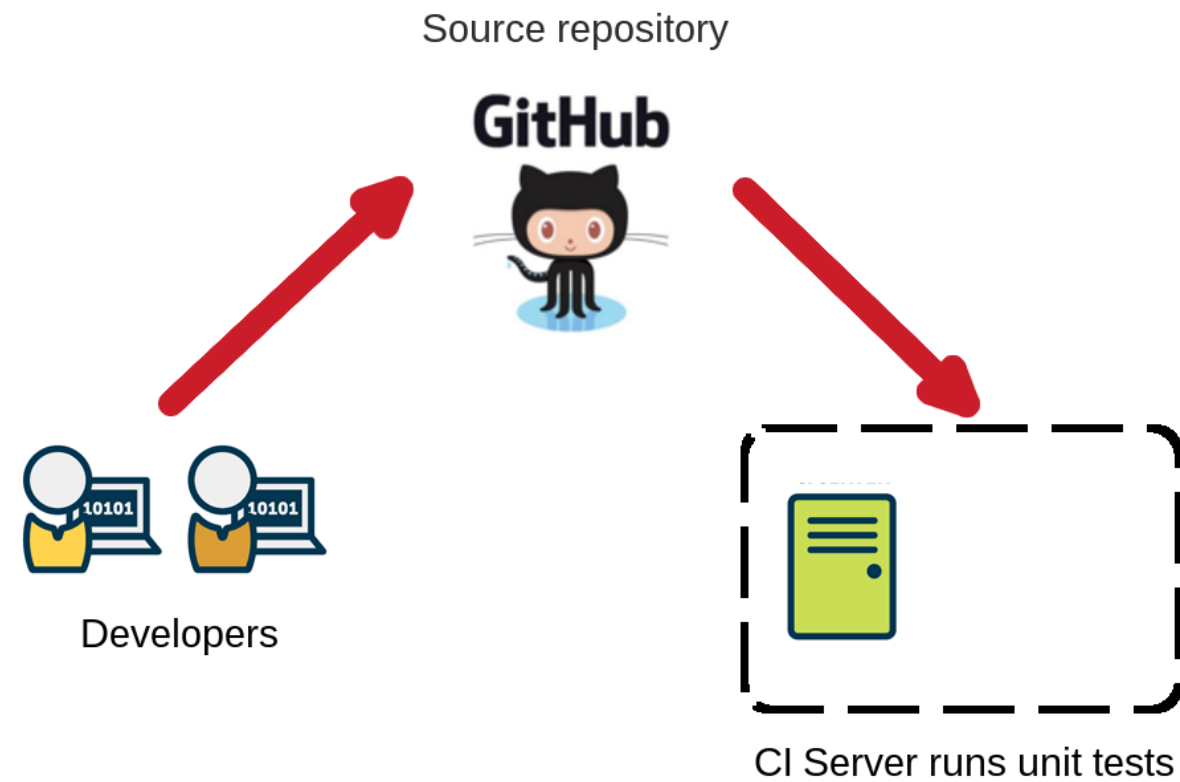
NumPy is the fundamental package needed for scientific computing with Python.

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

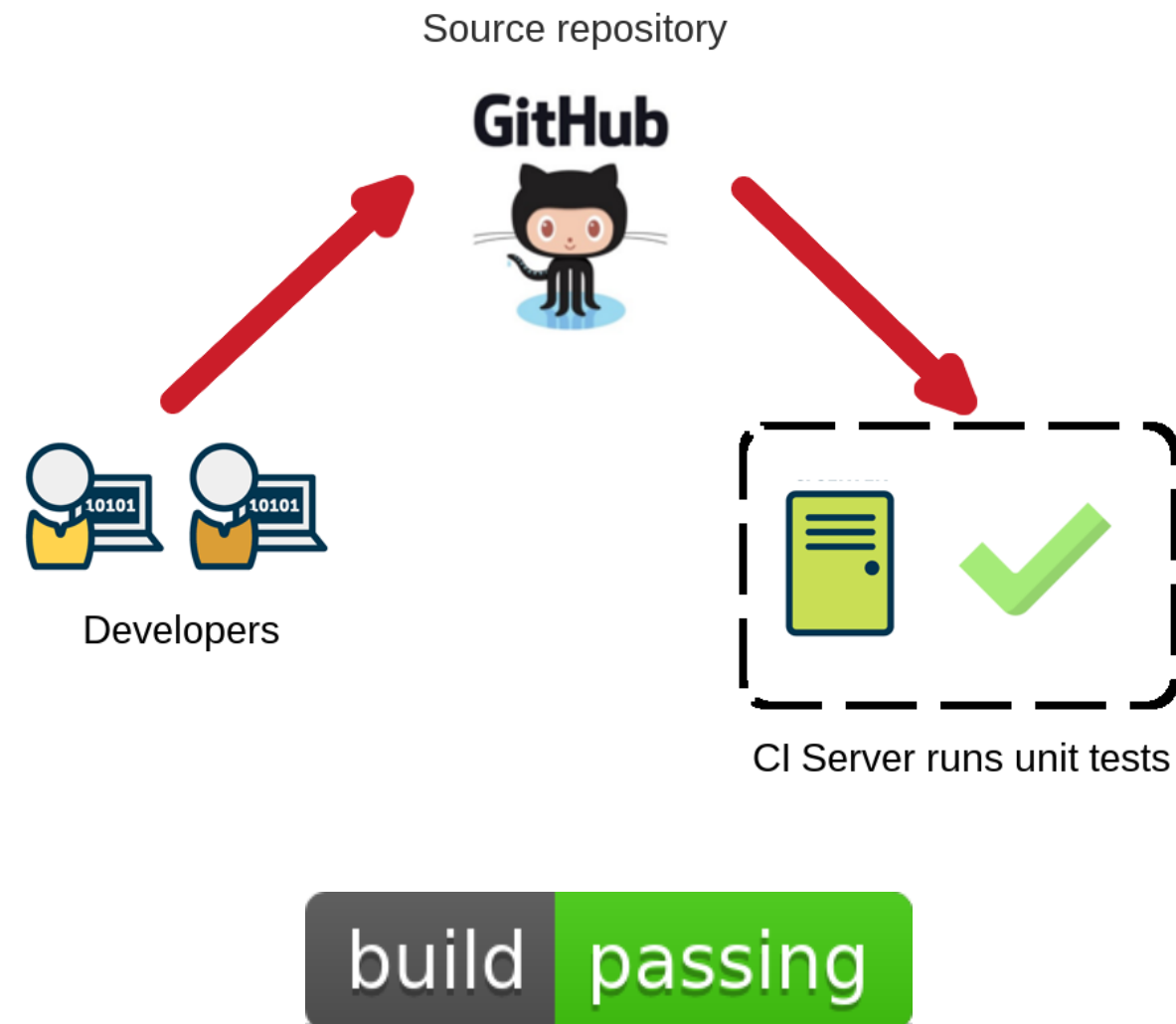
# The build status badge



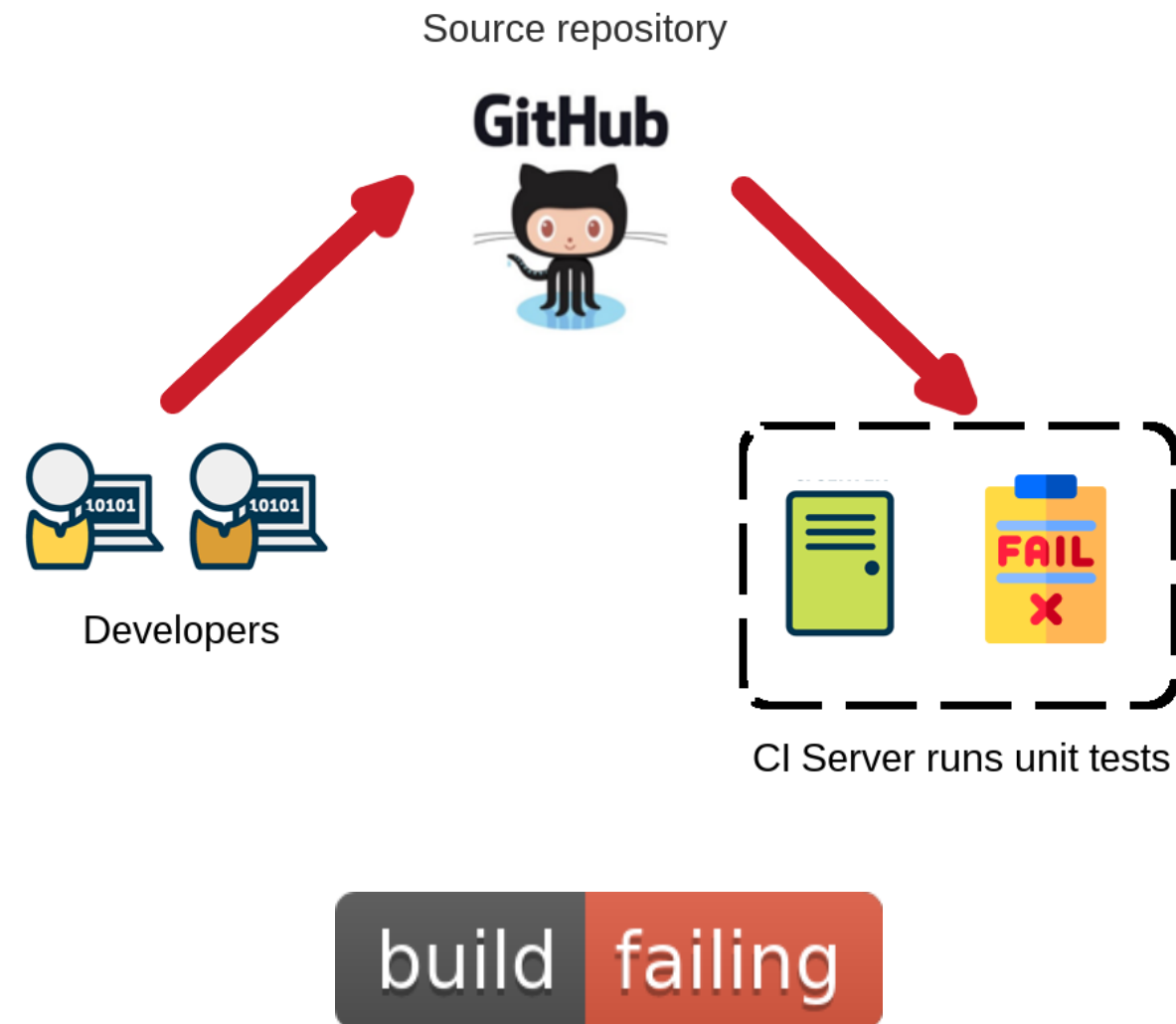
# The build status badge



# Build passing = Stable project



# Build failing = Unstable project



# 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 the GitHub interface with the 'Marketplace' tab selected. The main content area displays a list of repositories, with 'gutfeeling/beginner\_nlp' highlighted. The sidebar on the left shows the user 'gutfeeling' and a list of repositories. The right sidebar features a 'Discover repositories' section with recommendations like 'spacy-cld' and 'Python-Machine-Learning-Cookbook'.

**Repositories**

- datacamp/courses-unit-testing-i...
- gutfeeling/univariate-linear-regr...
- gutfeeling/version-control-workf...
- gutfeeling/pythonbooks\_heroku
- datacamp-content/courses-appl...
- datacamp-content/courses-appl...
- gutfeeling/practical\_rl\_for\_coders


Show more

**Your teams**

**Discover repositories**

- nickdavidhaynes/spacy-cld  
Language detection extension for spaCy 2.0+  
Python ★ 87
- PacktPublishing/Python-Machine-Learning-Cookbook  
Code files for Python-Machine-Learning-Cookbook  
Python ★ 259
- sloria/textblob-fr  
French language support for TextBlob.

# Step 3: Install the Travis CI app

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

Marketplace / Search results

Categories

API management

Chat

Code quality

Code review


Continuous integration

Dependency management

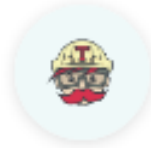
Deployment


IDEs

Learning

 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

We offer free CI for Open Source projects

\$0

#### ONE

Free Trial

Unlimited builds, 1 job at a time. Ideal for hobby and small projects.

\$69

/ month

#### THREE

Free Trial

Unlimited builds, 3 jobs at a time. Best suited for small teams.

\$199

/ month

#### SIX

Free Trial

Unlimited builds, 6 jobs at a time. Great for growing teams.

\$349

/ month

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



## Install Travis CI


Install on your personal account Dibya Chakravorty 

☐ **All repositories**  
This applies to all current *and* future repositories.

☒ **Only select repositories**

Select repositories ▾

Selected 1 repository

 gutfeeling/univariate-linear-regression ×

...with these permissions:

- ✓ **Read** access to code
- ✓ **Read** access to metadata and pull requests
- ✓ **Read** and **write** access to checks, commit statuses, deployments, and repository hooks

[Install](#) [Cancel](#)

Next: you'll be directed to the GitHub App's site to complete setup.

# 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

Travis CI

[Dashboard](#) [Changelog](#) [Documentation](#) [Help](#)



Search all repositories

My Repositories


Running (1/1)

+

gutfeeling/univariate-linear-reg # 13

Duration: 28 sec

gutfeeling / univariate-linear-regression



build passing

Current

Branches

Build History

Pull Requests

More options

Remove useless comment

#13 started

Cancel build

Commit 8f6c815

Compare 3695237..8f6c815

Branch master

Dibya Chakravorty

Python: 3.6

Running for 28 sec

Job log

View config



# Step 4: Showing the build status badge

Travis CI

Dashboard

Changelog

Documentation

Help

Search all repositories

My Repositories

Running (1/1)

+

gutfeeling/univariate-linear-reg # 13

Duration: 28 sec

gutfeeling / univariate-linear-regression

build passing

Current

Branches

Build History

Pull Requests

More options

master Remove useless comment

#13 started

Cancel build

Commit 8f6c815

Compare 3695237..8f6c815

Branch master

Dibya Chakravorty

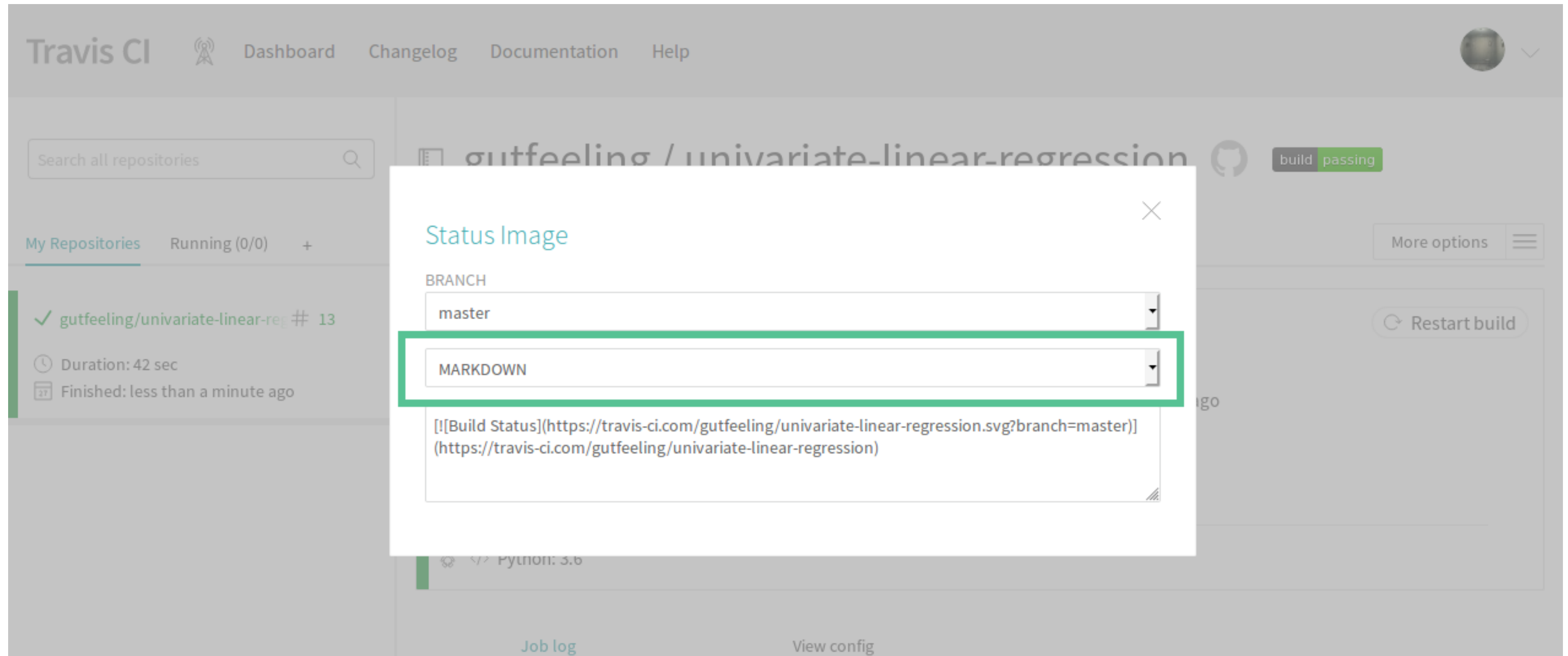
Python: 3.6

Running for 28 sec

Job log


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 'Travis CI', a signal icon, and links for 'Dashboard', 'Changelog', 'Documentation', and 'Help'. Below this is a search bar labeled 'Search all repositories'. The main content area shows a repository 'gutfeeling / univariate-linear-regression' with a 'build passing' status. A modal window titled 'Status Image' is open, showing a dropdown menu with 'master' selected. Below the dropdown, the 'MARKDOWN' option is highlighted with a green border. The modal also displays the generated markdown code: `[![Build Status](https://travis-ci.com/gutfeeling/univariate-linear-regression.svg?branch=master)](https://travis-ci.com/gutfeeling/univariate-linear-regression)`. In the background, a build log for 'gutfeeling/univariate-linear-reg # 13' is visible, showing a duration of 42 seconds and a status of 'Finished: less than a minute ago'. A 'Restart build' button is also present.


# Step 4: Showing the build status badge

 **gutfeeling** Remove codecov badge

Latest commit e08b854 now

 <a href="#">data/raw</a>	Add tests	2 months ago
 <a href="#">notebooks</a>	Fix jupyter notebook	2 months ago
 <a href="#">src</a>	remove fake data generator	4 days ago
 <a href="#">tests</a>	Better test for float valued string	4 days ago
 <a href="#">.gitignore</a>	Improved gitignore	3 months ago
 <a href="#">.travis.yml</a>	Remove comments	4 days ago
 <a href="#">README.md</a>	Remove codecov badge	now
 <a href="#">setup.py</a>	Remove useless comment	3 minutes ago

 **README.md** 



# Code coverage



- $\text{code coverage} = \frac{\text{num lines of application code that ran during testing}}{\text{total num lines of application code}} \times 100$
- 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



[Pull requests](#) [Issues](#) [Marketplace](#) [Explore](#)



  

[Marketplace](#) / Search results

### Categories

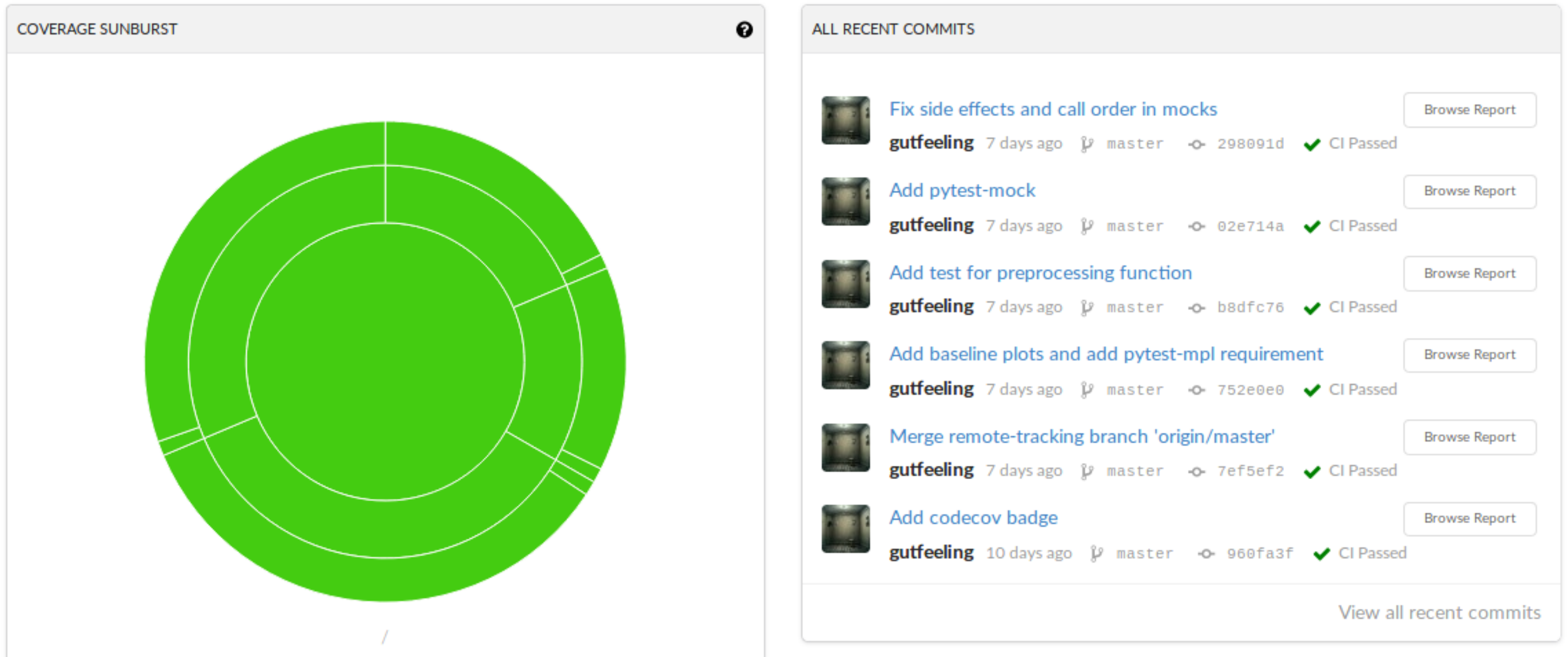
- API management
- Chat
- Code quality
- Code review
- Continuous integration
- Dependency management
- Deployment
- IDEs
- Learning

**1 result** for "codecov"

**Codecov**   
Group, merge, archive and compare coverage reports

[Previous](#) [Next](#)

# Commits lead to coverage report at codecov.io



# Step 3: Showing the badge in GitHub

The screenshot shows the GitHub repository settings page for the repository 'gutfeeling/univariate-linear-regression'. The 'Settings' tab is selected in the top navigation bar. On the left sidebar, the 'Badge' section is highlighted. The main content area displays three tabs: 'Markdown', 'HTML', and 'RST'. The 'Markdown' tab is active and highlighted with a green border. It contains a code snippet for a Codecov badge, which is partially visible. The 'HTML' and 'RST' tabs also show their respective code snippets. Each tab has a 'Copy' button in the top right corner.

General








Yaml



Badge



Markdown

```
[![codecov](https://codecov.io/gh/gutfeeling/univariate-linear-regression/branch/master/graphl
```

HTML

```
<a href="https://codecov.io/gh/gutfeeling/univariate-linear-regression">  
   <a href="#">data</a>        | Match the course code | 12 minutes ago |
|  <a href="#">notebooks</a>   | Match the course code | 12 minutes ago |
|  <a href="#">src</a>         | Match the course code | 12 minutes ago |
|  <a href="#">tests</a>       | Match the course code | 12 minutes ago |
|  <a href="#">.gitignore</a>  | Improved gitignore    | 4 months ago   |
|  <a href="#">.travis.yml</a> | Remove comments       | 2 months ago   |
|  <a href="#">README.md</a>   | Update README.md      | 4 minutes ago  |
|  <a href="#">setup.py</a>  | Add pytest-mock       | 2 months ago   |

 **README.md** 

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