# 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`

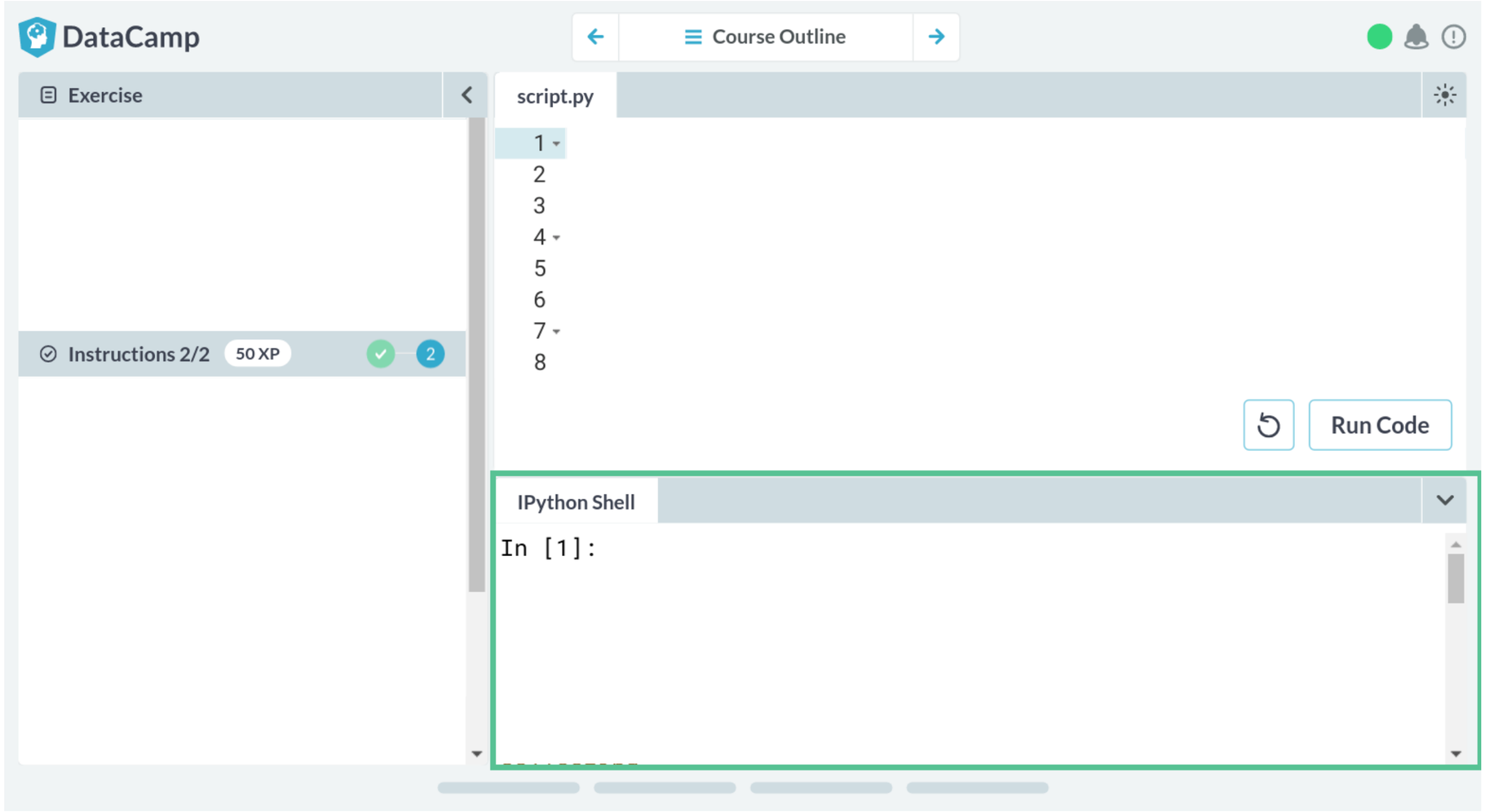
```
import pytest
from data.preprocessing_helpers import row_to_list, convert_to_int

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



# 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

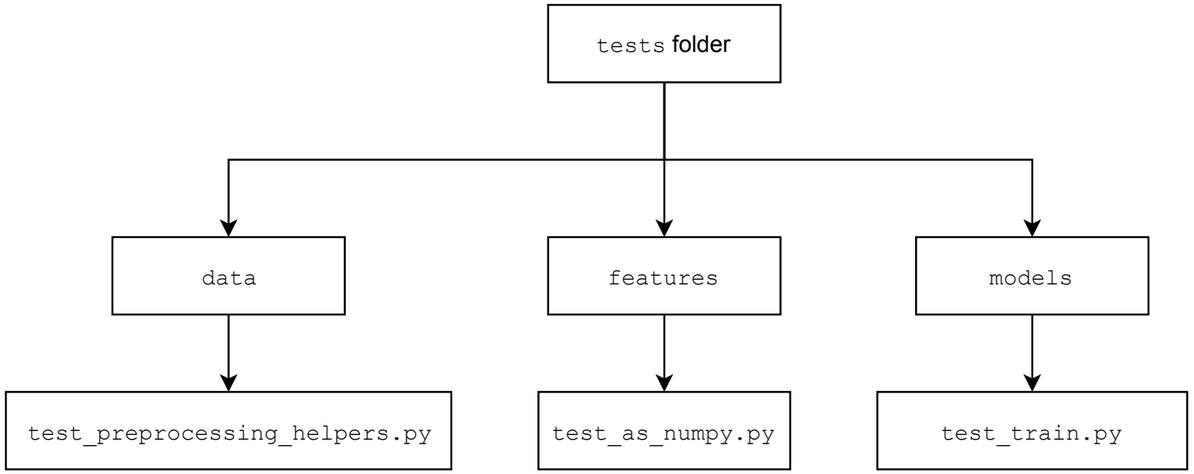
# Mastering test execution

UNIT TESTING FOR DATA SCIENCE IN PYTHON

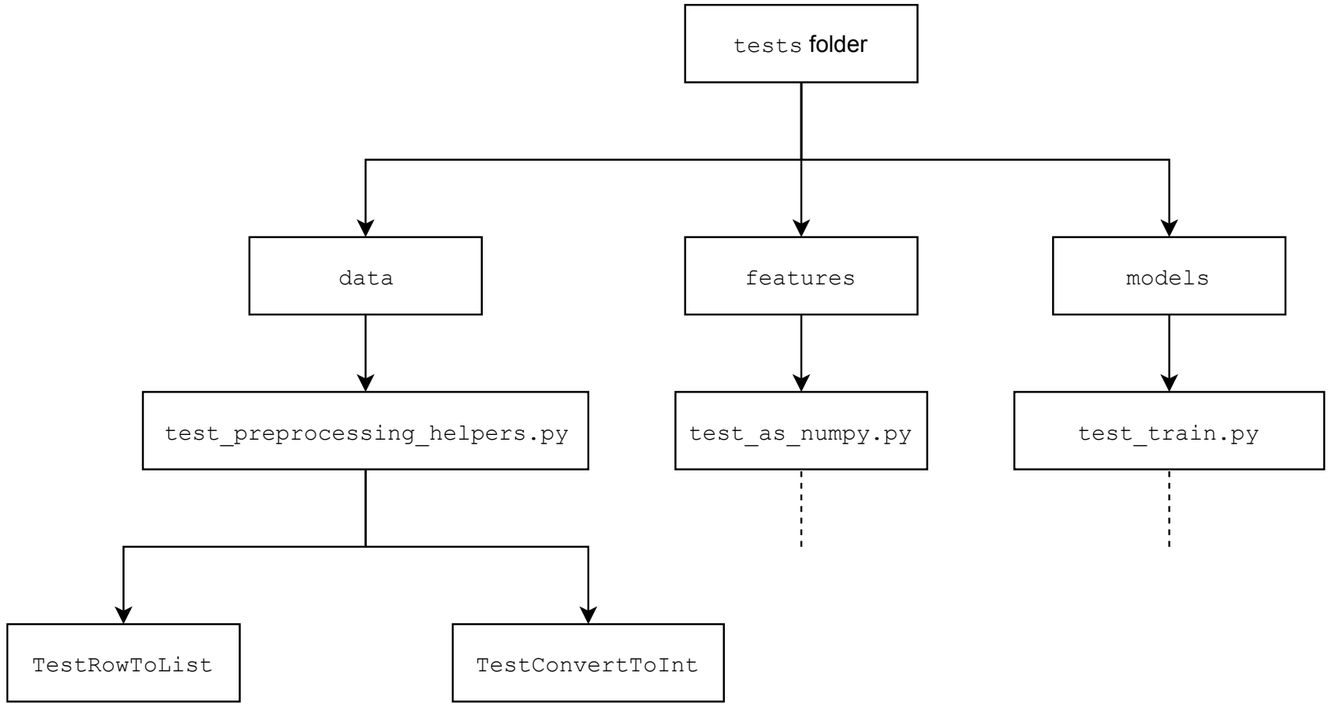
**Dibya Chakravorty**
Test Automation Engineer

# Test organization

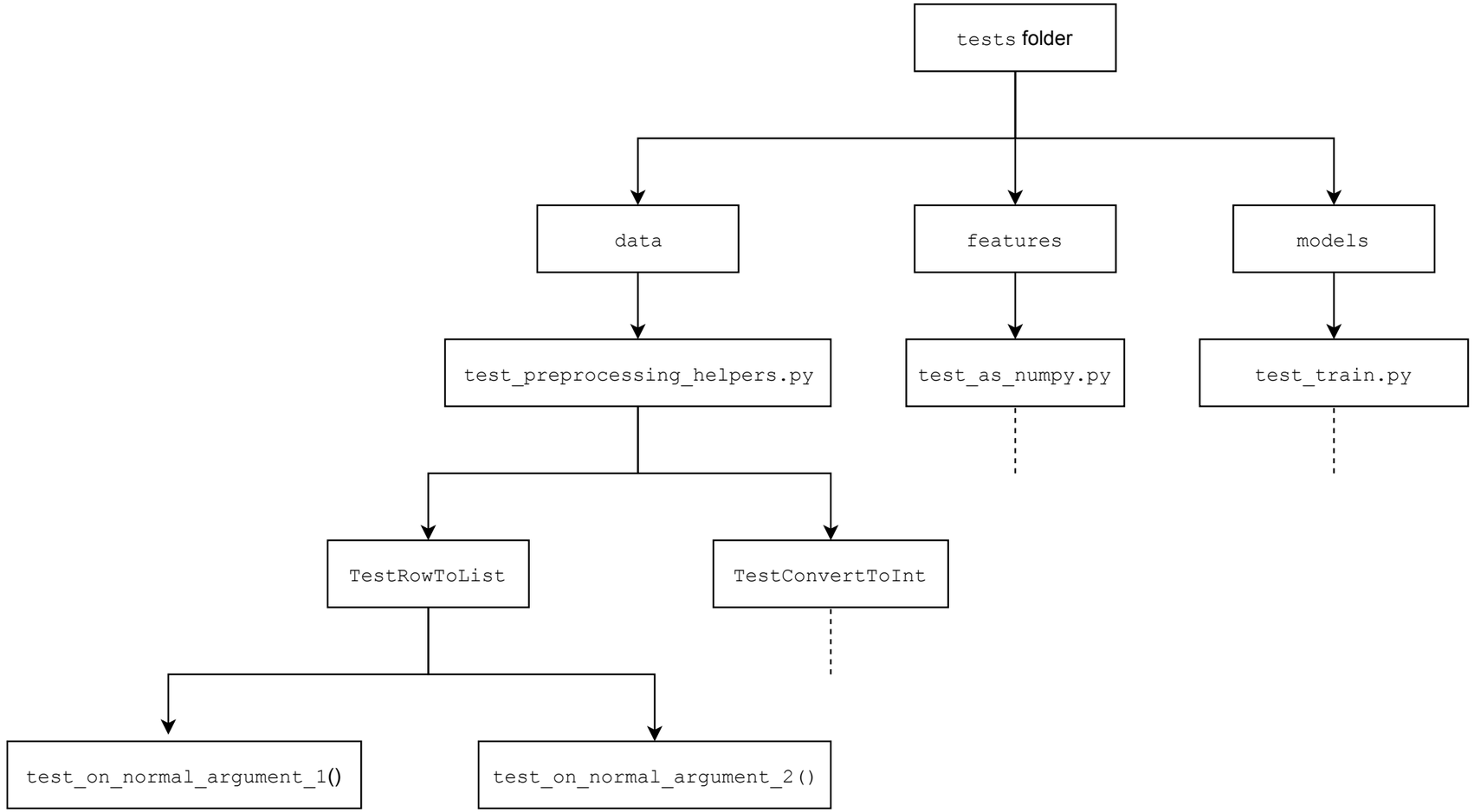
# Test organization



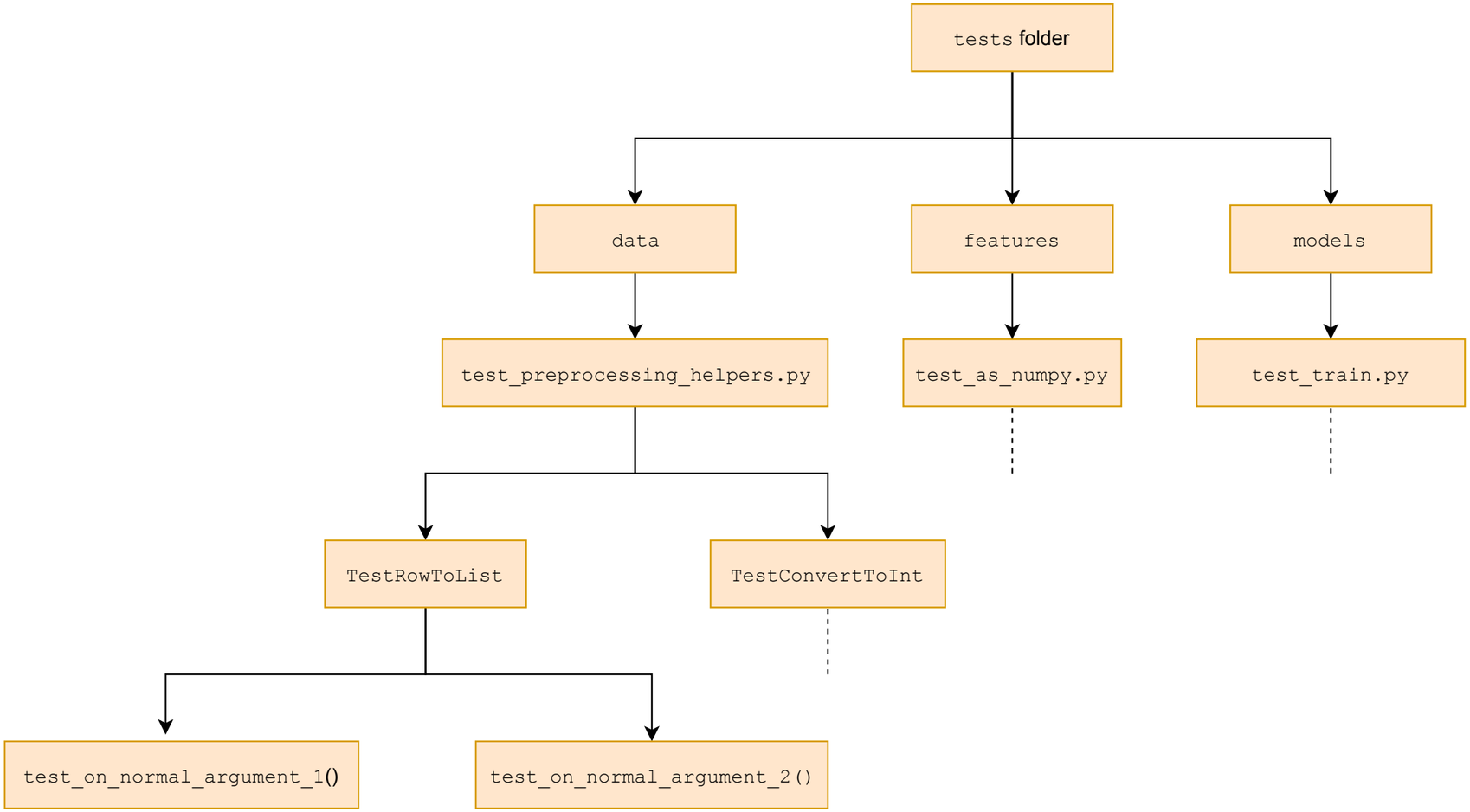
# Test organization



# Test organization



# Running all tests



# Running all tests

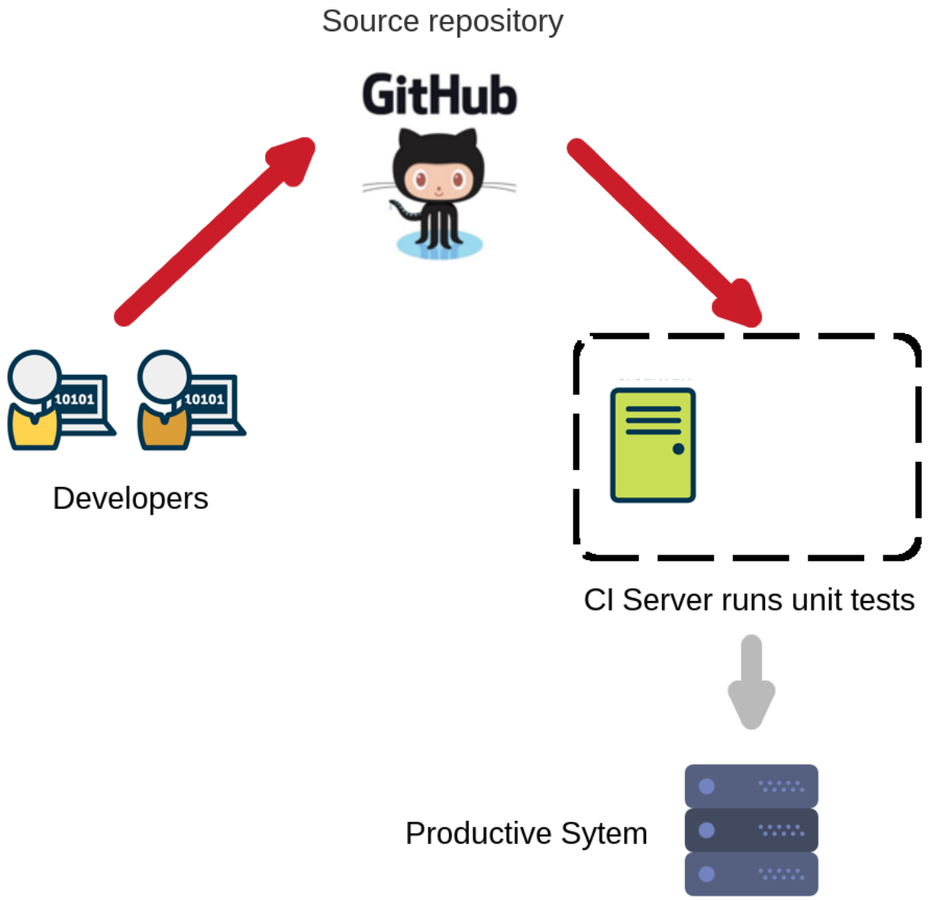
```
cd tests
pytest
```

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

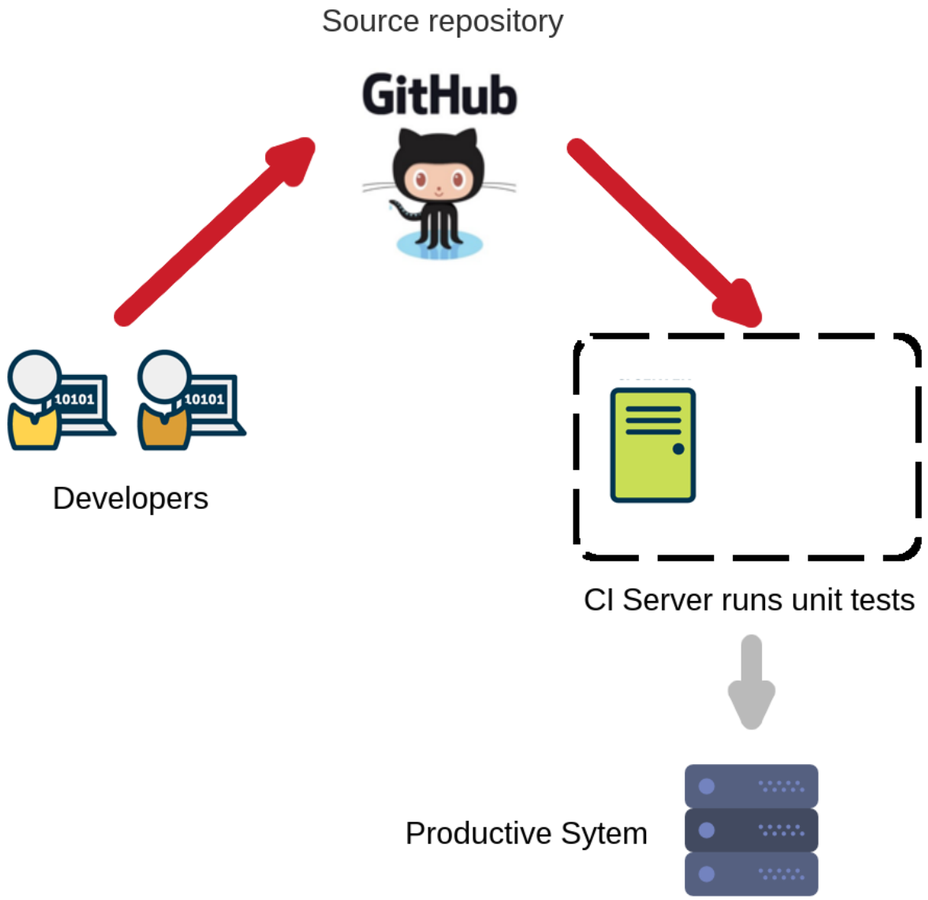
# Running all tests

```
============================================================ 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?

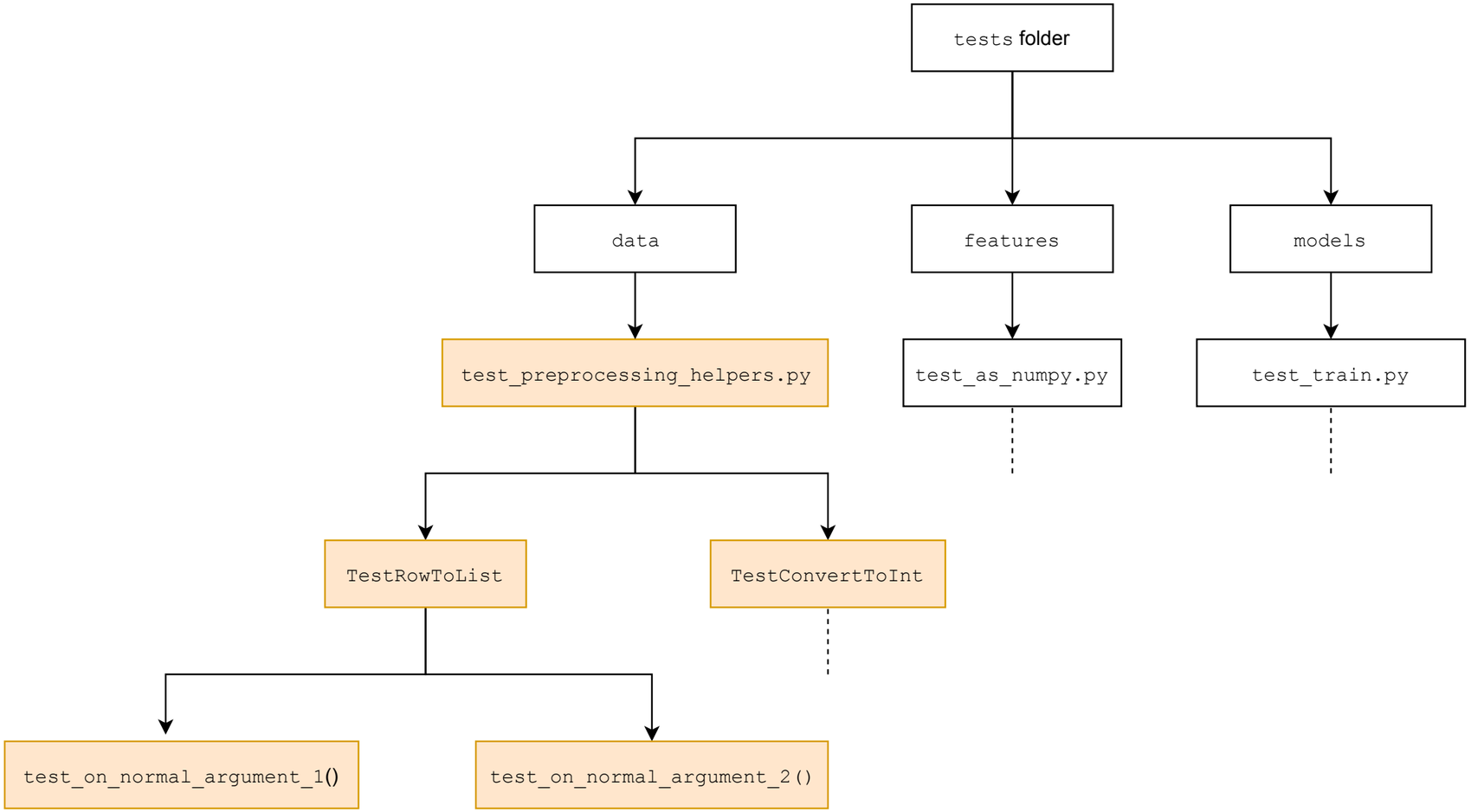


# The -x flag: stop after first failure

```
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

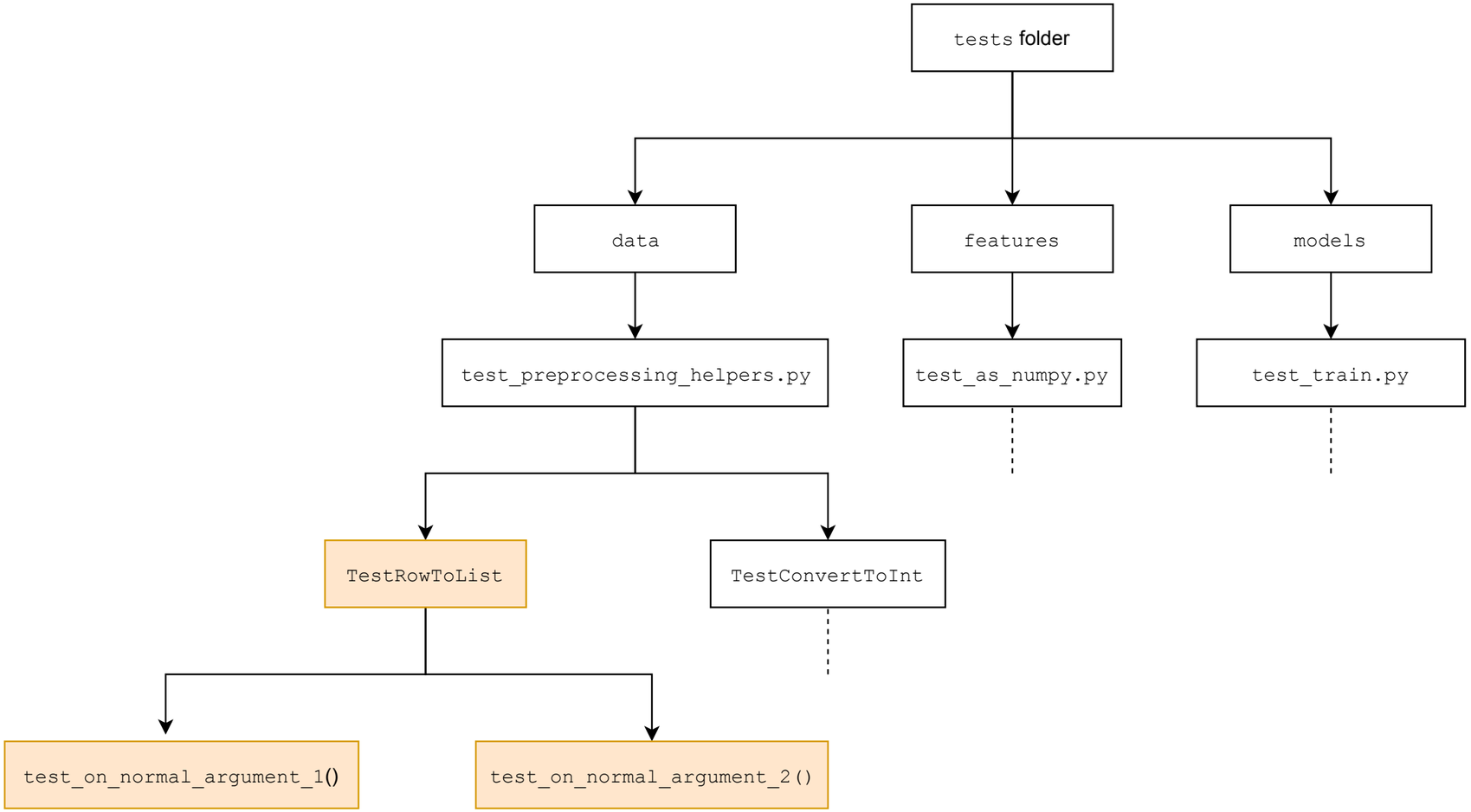


# Running tests in a test module

```
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: {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, 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 ==================================
```

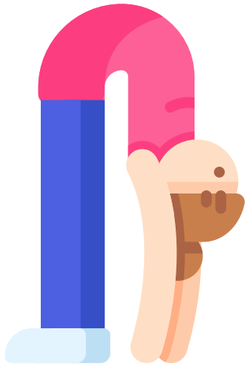
# Running tests using node ID

- 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!

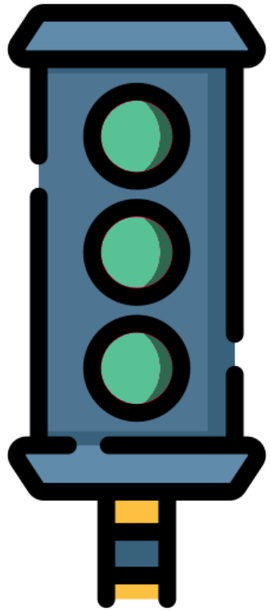
UNIT TESTING FOR DATA SCIENCE IN PYTHON

# Expected failures and conditional skipping

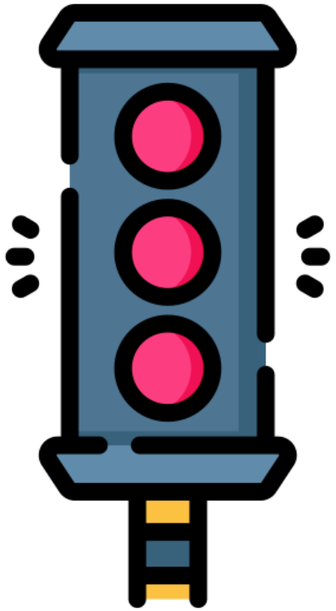
UNIT TESTING FOR DATA SCIENCE IN PYTHON

**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.

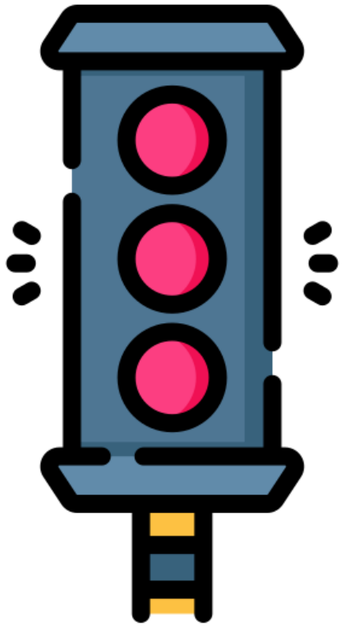
```
import pytest

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):
        ...
```

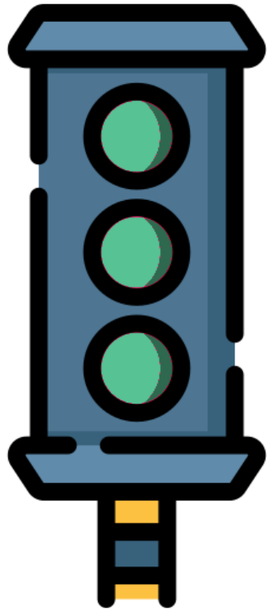
# xfail: marking tests as "expected to fail"

```
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 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
```

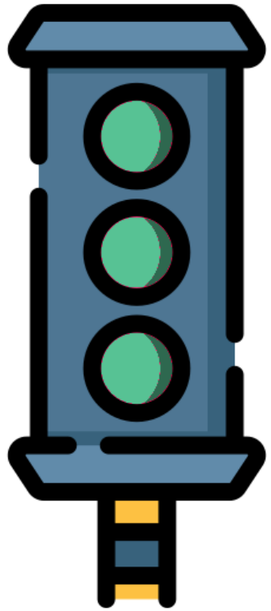
# The reason argument

```
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/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



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

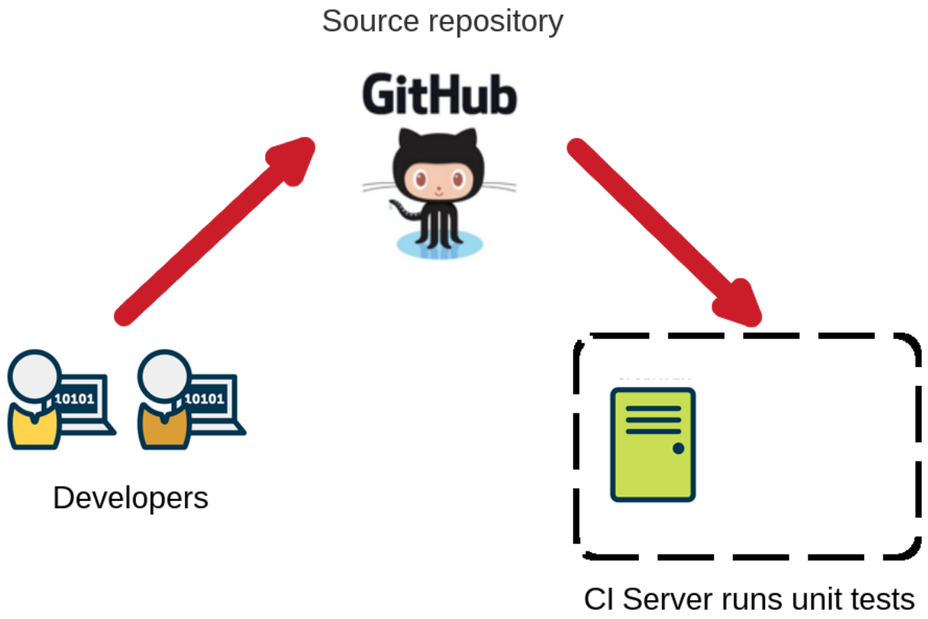


NumPy is the fundamental package needed for scientific computing with Python.

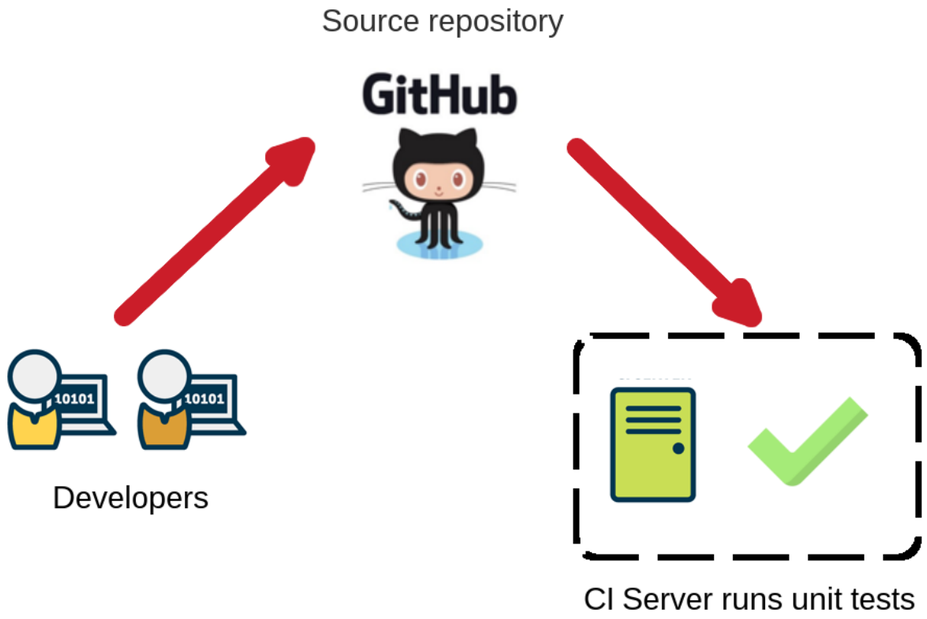
- **Website (including documentation):** https://www.numpy.org
- **Mailing list:** https://mail.python.org/mailman/listinfo/numpy-discussion

# The build status badge

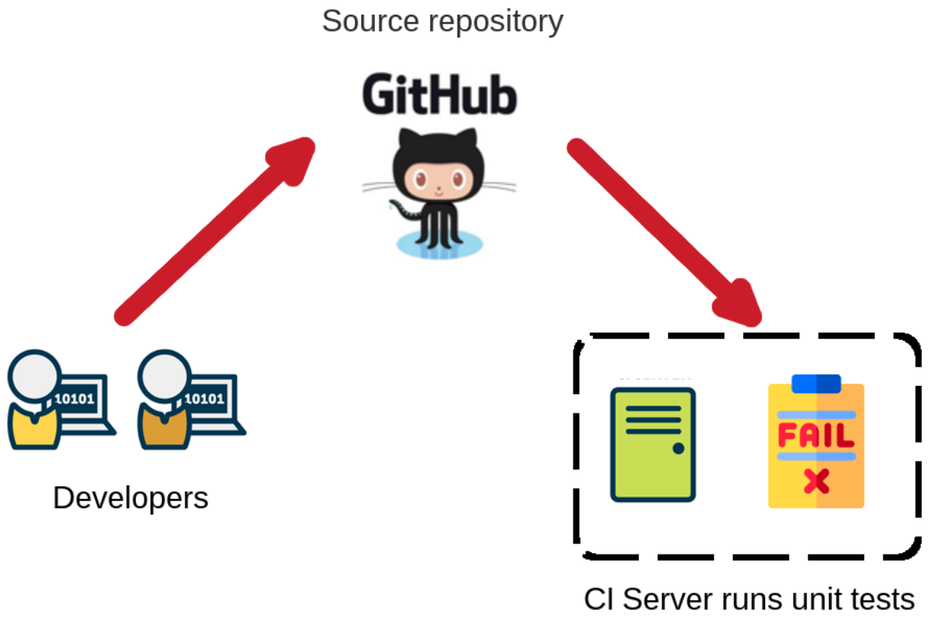
# The build status badge



# Build passing = Stable project




# Build failing = Unstable project




# CI server

# Step 1: Create a configuration file

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

# Step 1: Create a configuration file

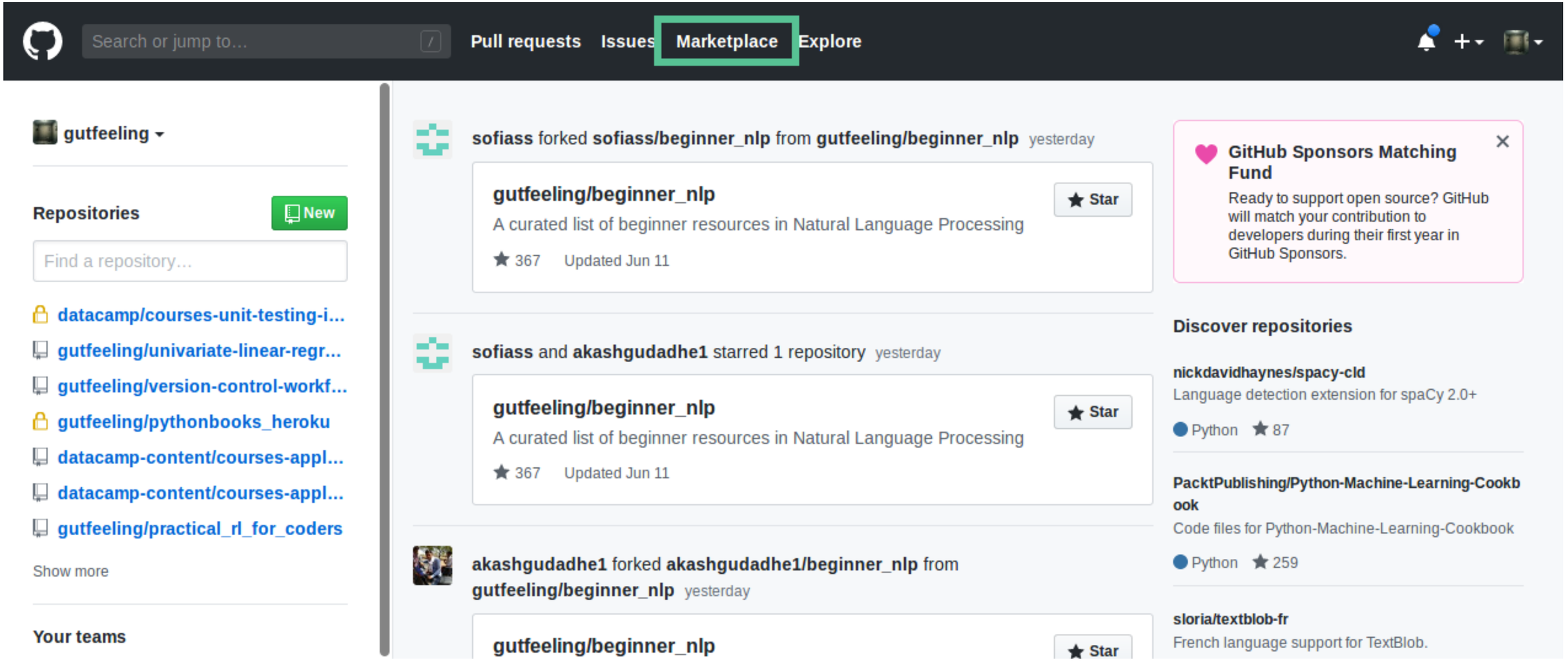
- Contents of `.travis.yml` .

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

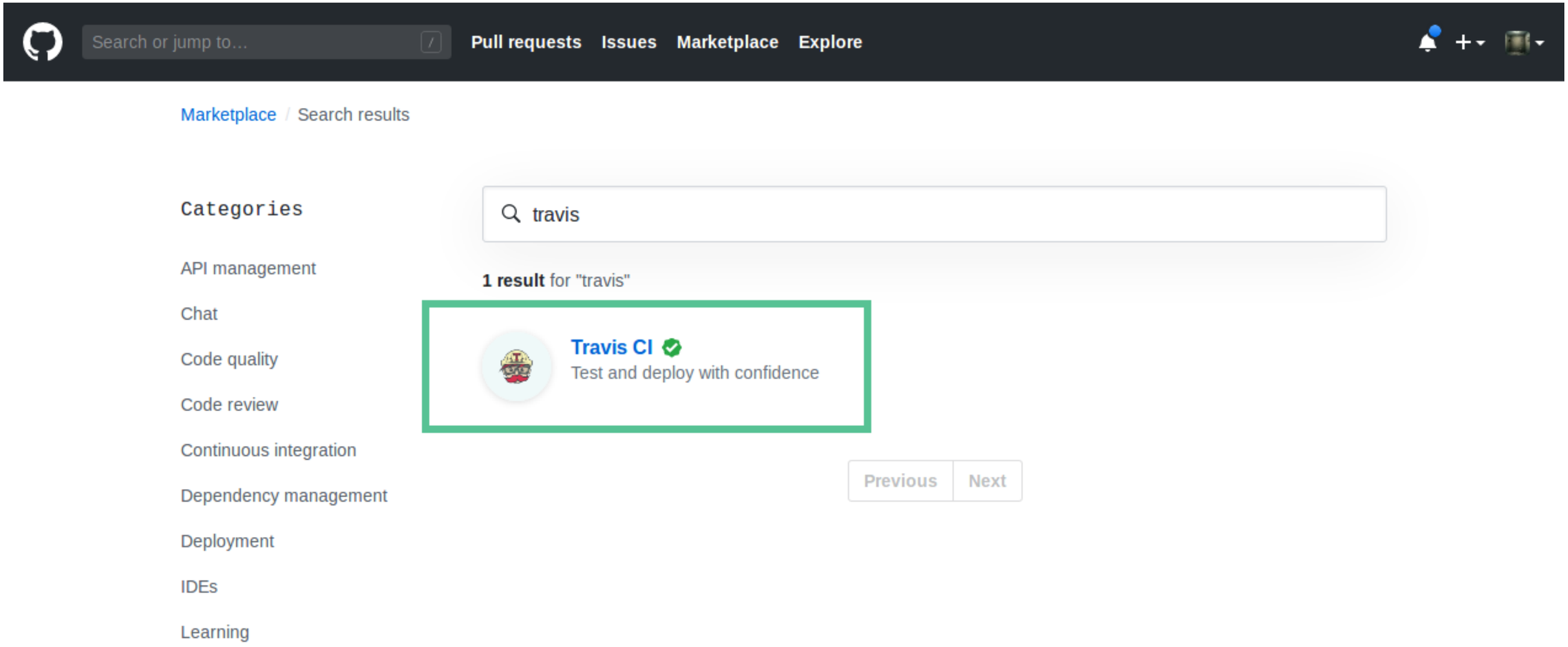
# Step 2: Push the file to GitHub

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

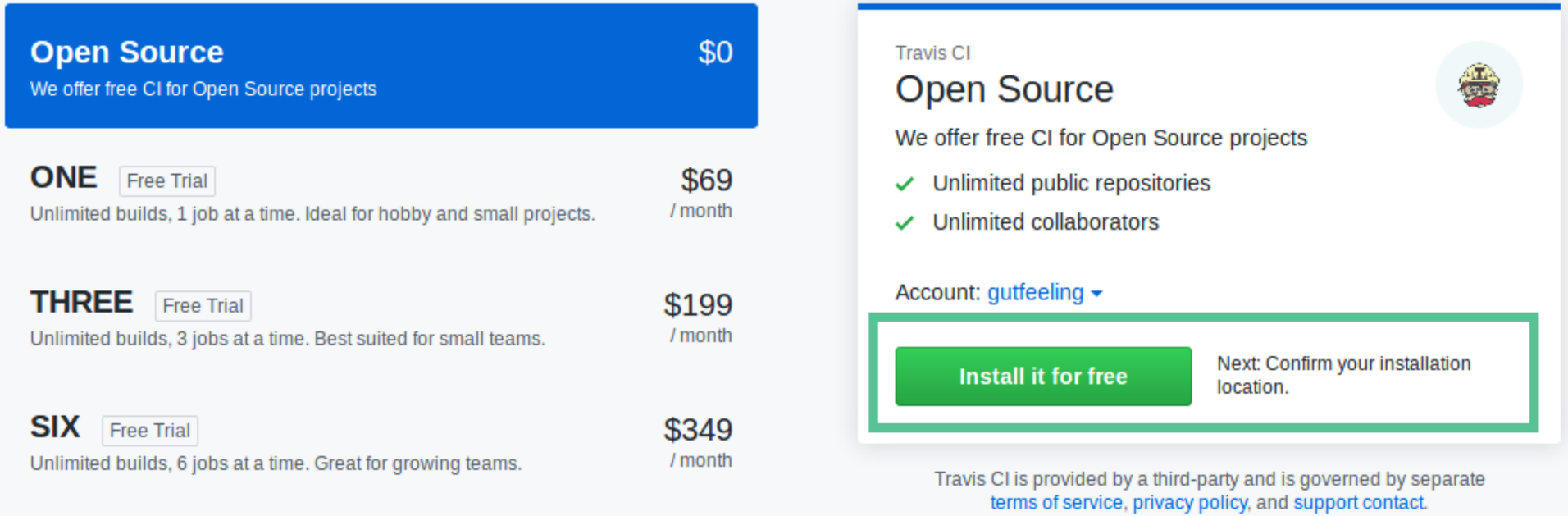
# Step 3: Install the Travis CI app



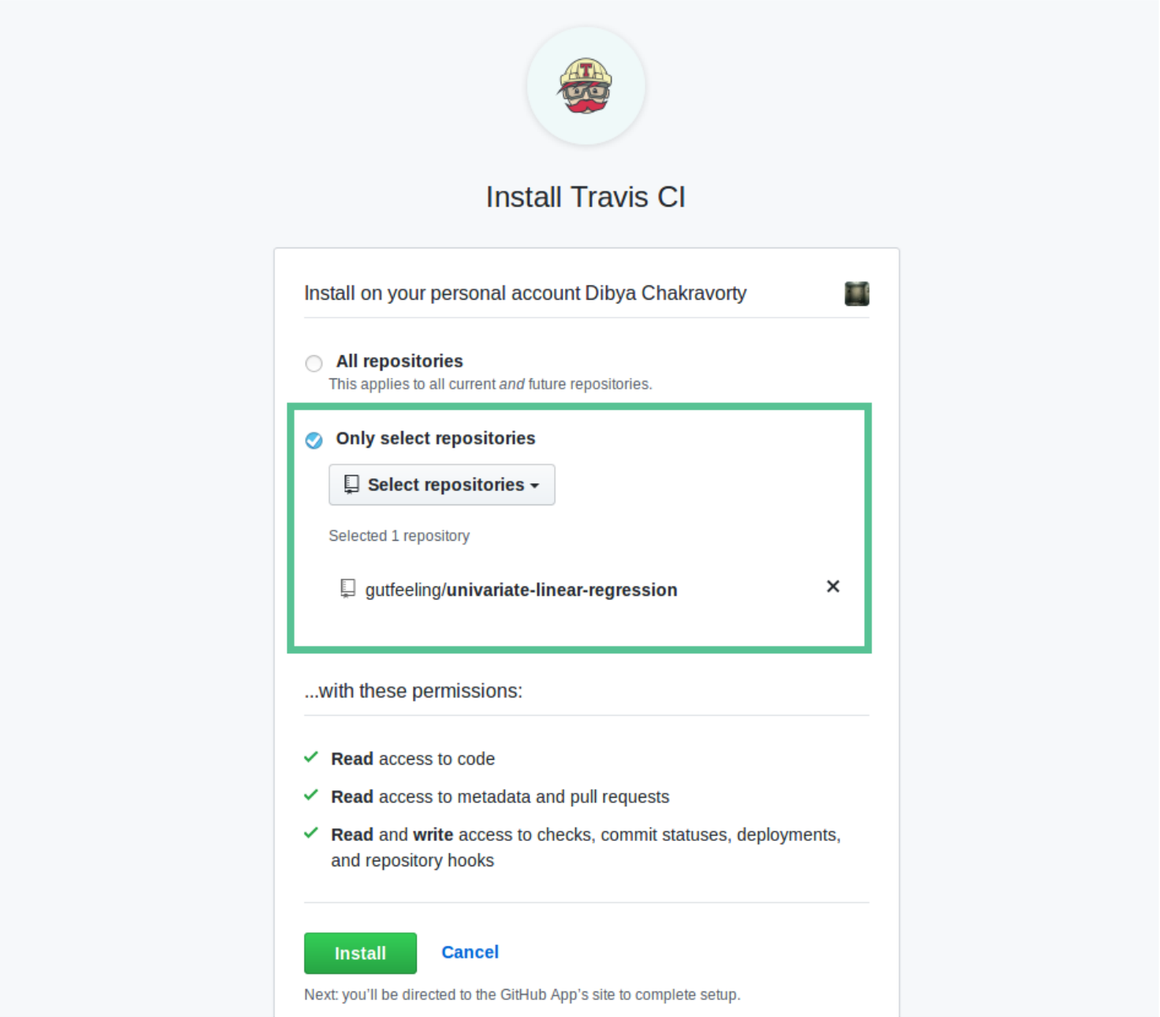
# Step 3: Install the Travis CI app



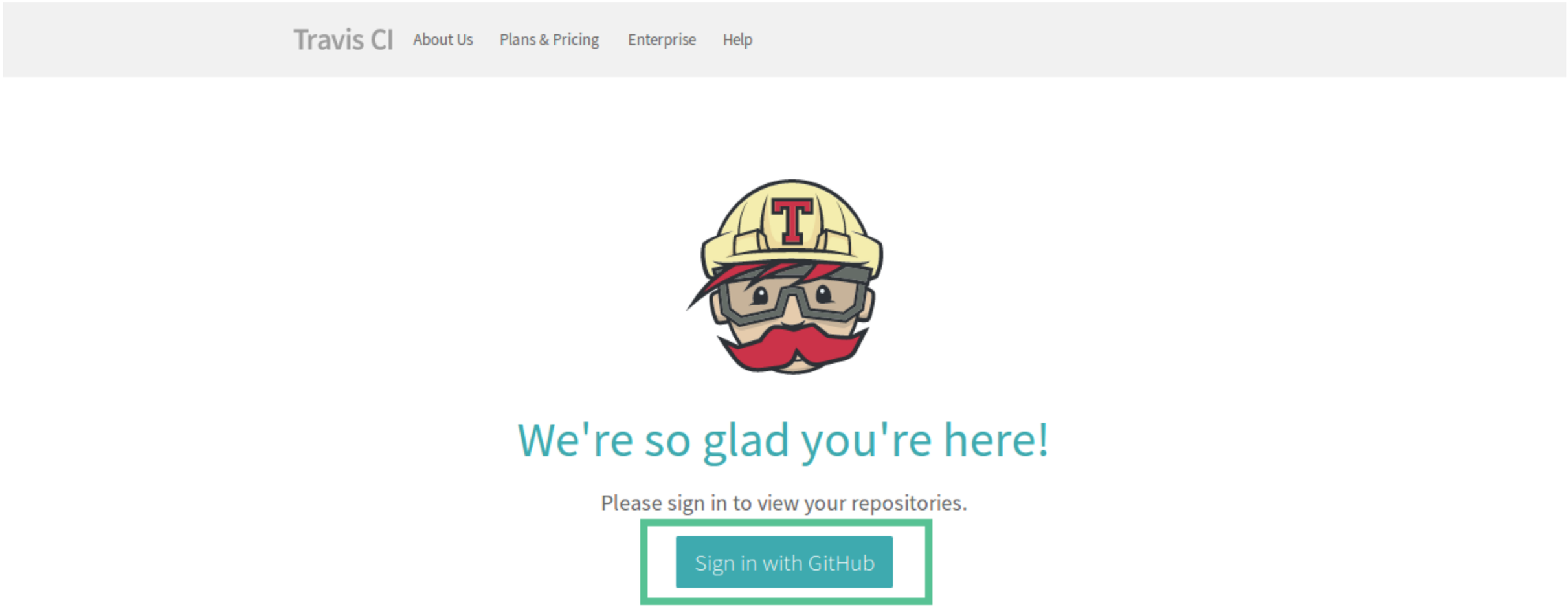
# Step 3: Install the Travis CI app



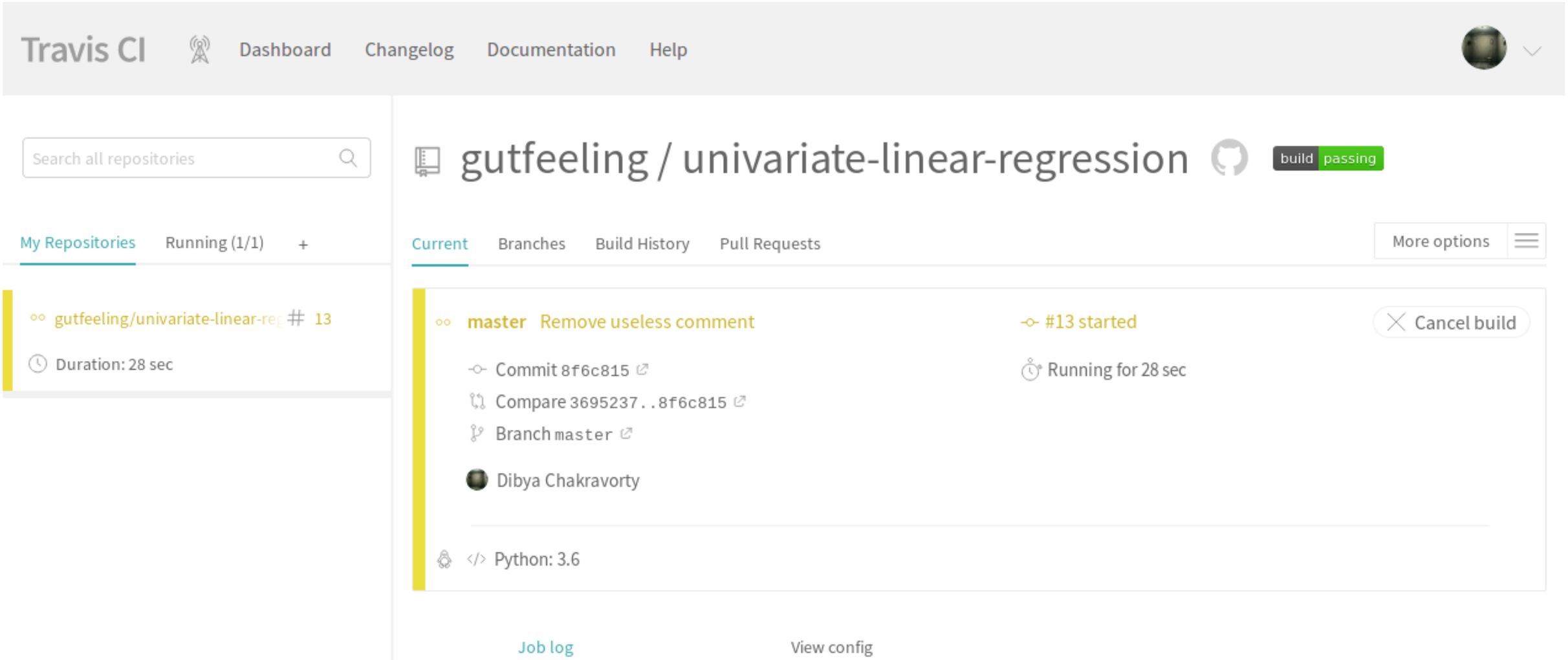
# Step 3: Install the Travis CI app



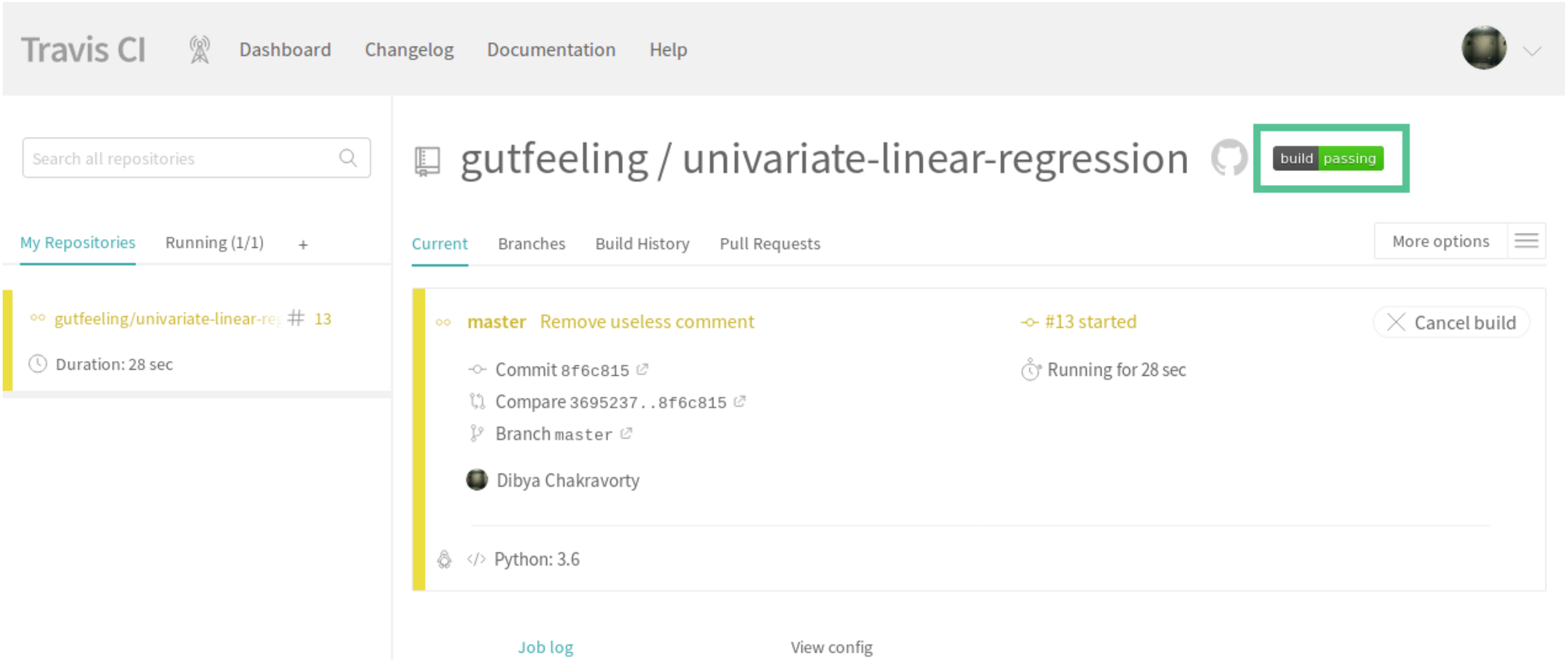
# Step 3: Install the Travis CI app



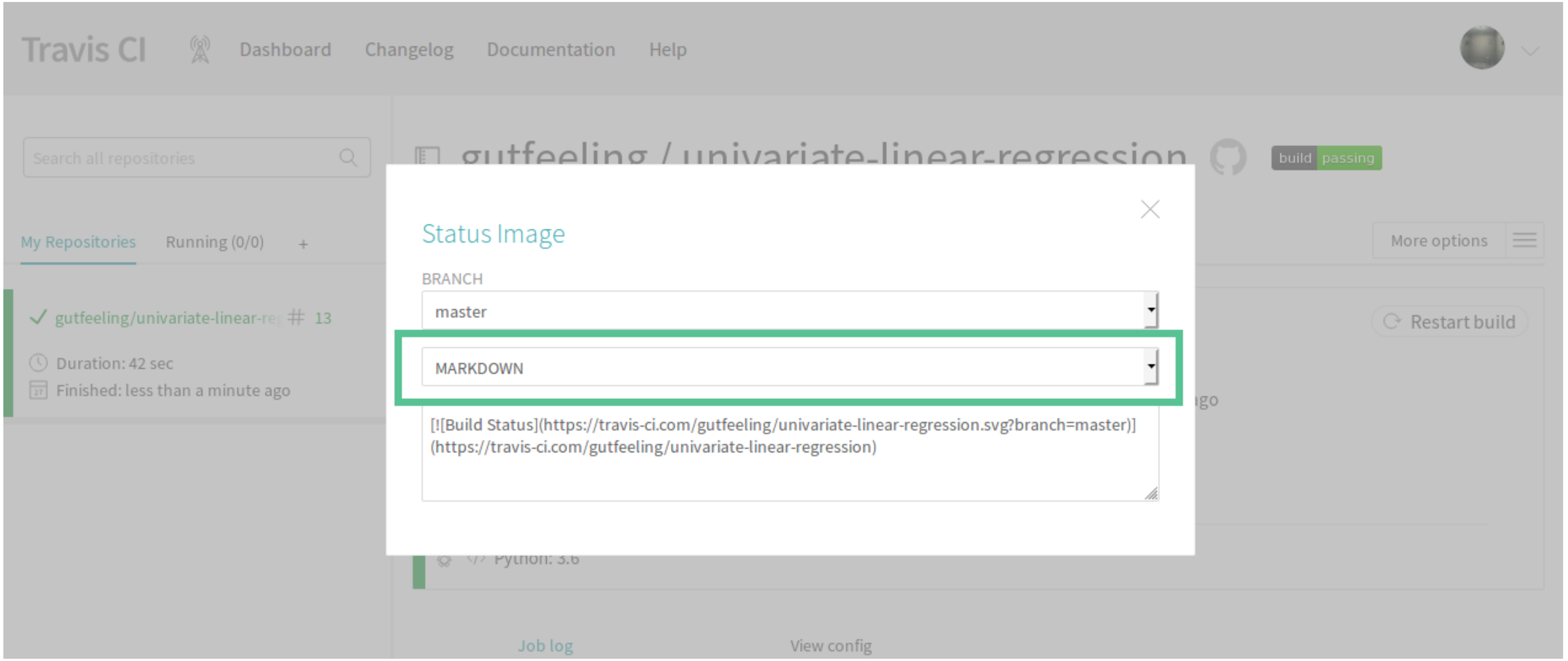
# Every commit leads to a build



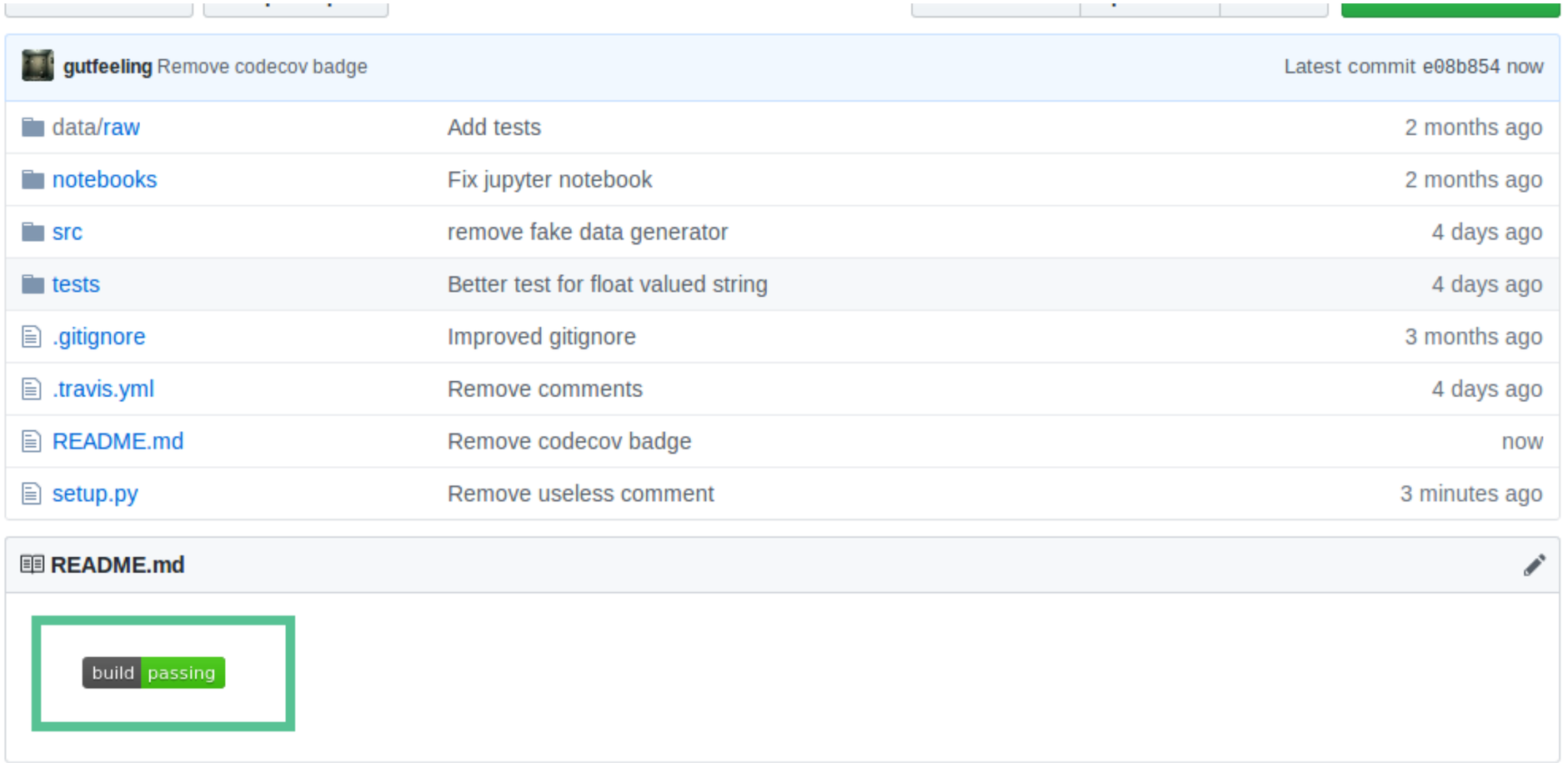
# Step 4: Showing the build status badge



# Step 4: Showing the build status badge



# Step 4: Showing the build status badge



gutfeeling Remove codecov badge — Latest commit e08b854 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 |

README.md

build passing

# 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`

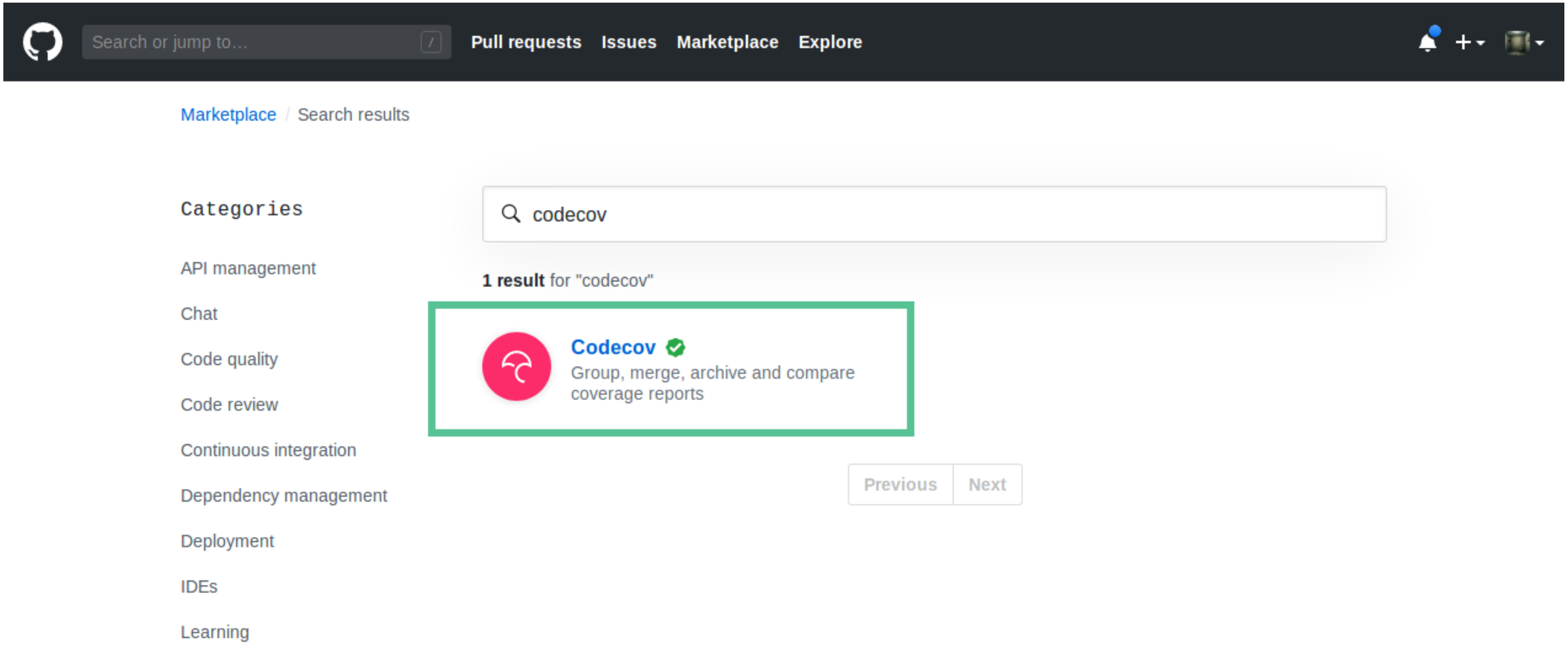
```
language: python
python:
  - "3.6"
install:
  - pip install -e .
  - pip install pytest-cov codecov    # Install packages for code coverage report
script:
  - pytest --cov=src tests            # Point to the source directory
```

# Step 1: Modify the Travis CI configuration file

- File: `.travis.yml`

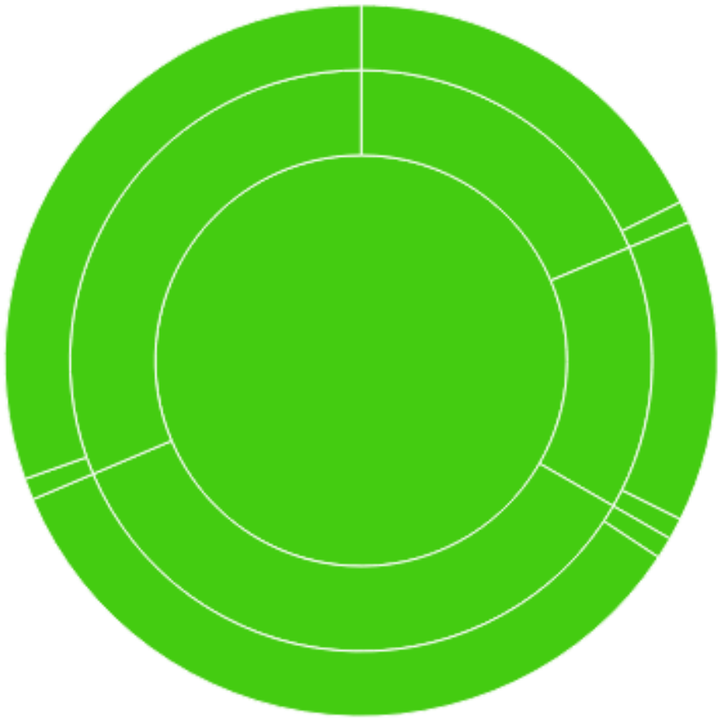
```
language: python
python:
  - "3.6"
install:
  - pip install -e .
  - pip install pytest-cov codecov    # Install packages for code coverage report
script:
  - pytest --cov=src tests            # Point to the source directory
after_success:
  - codecov                           # uploads report to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

# Step 2: Install Codecov



# 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

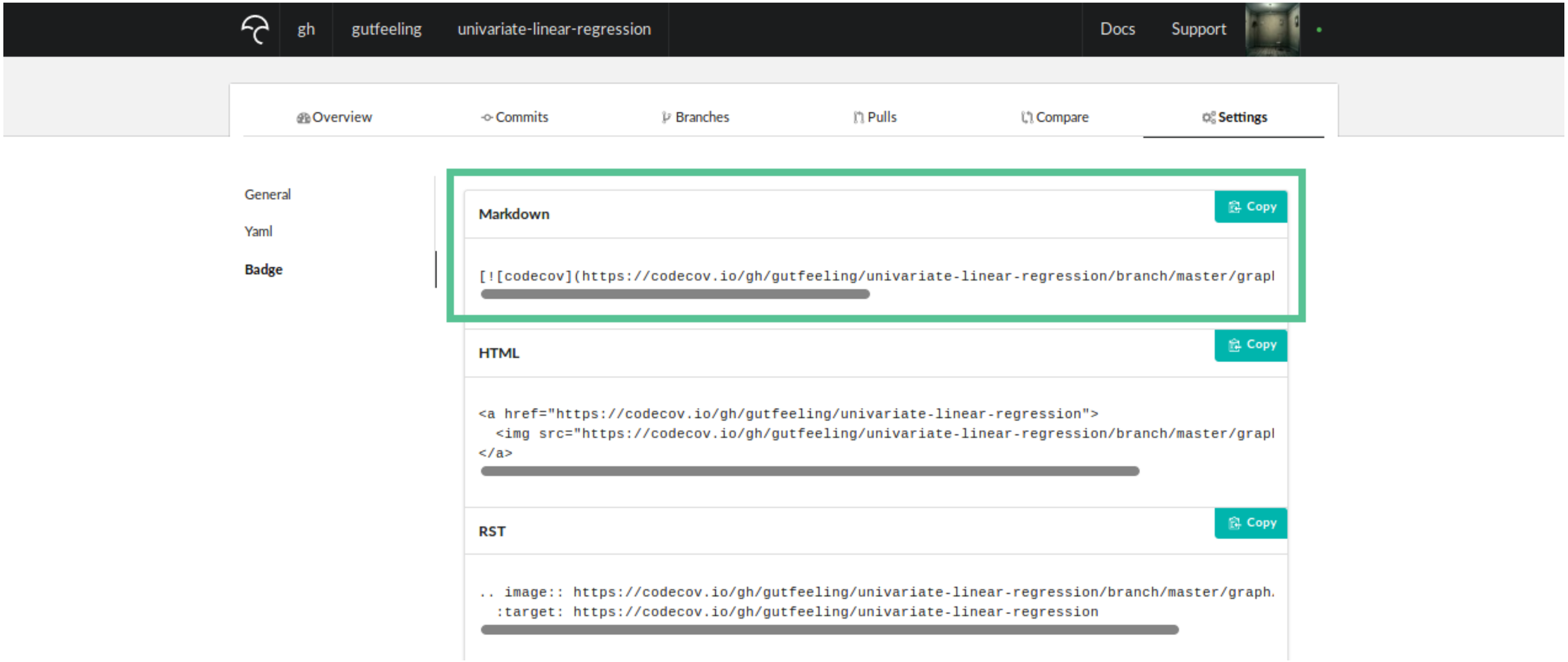
**gutfeeling** 7 days ago `master` `7ef5ef2` CI Passed

Add codecov badge — Browse Report

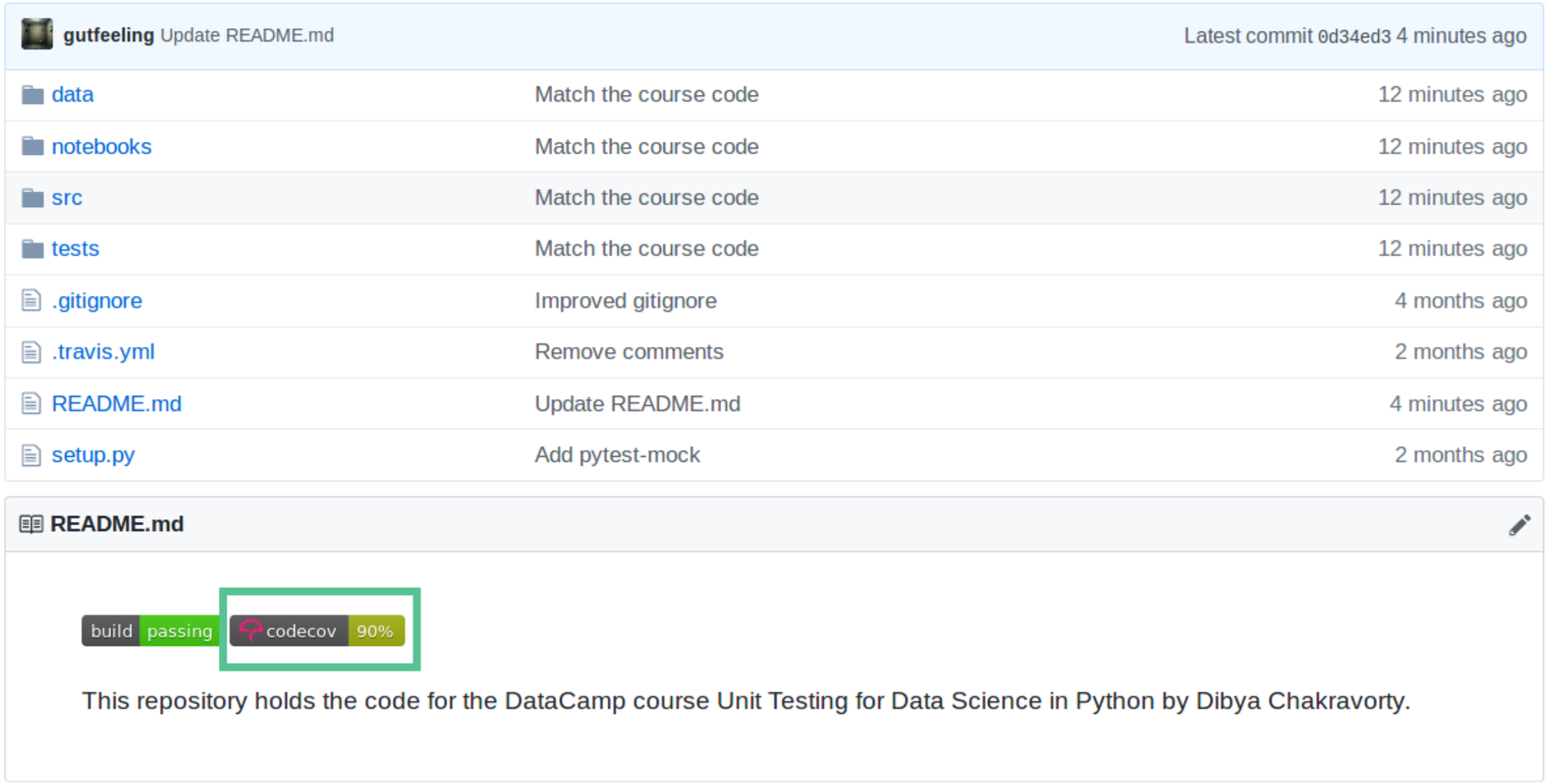
**gutfeeling** 10 days ago `master` `960fa3f` CI Passed

View all recent commits

# Step 3: Showing the badge in GitHub



# Step 3: Showing the badge in GitHub



# Let's practice CI and code coverage!

UNIT TESTING FOR DATA SCIENCE IN PYTHON