A Crash Course in Software Verification and Unit Tests

Daniel Perrefort

University of Pittsburgh

February 4, 2020

After this talk you should…

#

So you can find things later

1. Understand the differences between validation/verification and how that relates to unit tests
2. Be able to argue why testing your code is important
3. Know how to write a generic test case
4. Know what CI is and how it fits in your work-flow

Validation Vs. Verification

1

Validation is intended to ensure a product, service, or system results in a deliverable that meets the operational needs of the user.

Verification is intended to check that a product, service, or system meets a set of design specifications.

https://en.wikipedia.org/wiki/Verification_and_validation

Validation Vs. Verification

1

Validation is intended to ensure a product, service, or system results in a deliverable that meets the operational needs of the user.

- In English: Are we building the right system?

Verification is intended to check that a product, service, or system meets a set of design specifications.

- In English: Are we building the system right?

https://en.wikipedia.org/wiki/Verification_and_validation

Validation Vs. Verification

1

Validation is intended to ensure a product, service, or system results in a deliverable that meets the operational needs of the user.

- In English: Are we building the right system?

Verification is intended to check that a product, service, or system meets a set of design specifications.

- In English: Are we building the system right?

You add a blue “submit” button to an online form:

- Is it blue? Is “submit” spelled right?
- Does the underlying function raise an error if the input (i.e. the form) is empty?

https://en.wikipedia.org/wiki/Verification_and_validation

Validation Vs. Verification

1

Validation is intended to ensure a product, service, or system results in a deliverable that meets the operational needs of the user.

- In English: Are we building the right system?

Verification is intended to check that a product, service, or system meets a set of design specifications.

- In English: Are we building the system right?

You add a blue “submit” button to an online form:

- Is it blue? Is “submit” spelled right? <- (Validation)
- Does the underlying function raise an error if the input (i.e. the form) is empty? <- (Verification)

https://en.wikipedia.org/wiki/Verification_and_validation

Why do I care about this?

- As scientists, we care about validation…
    - To make sure we simulate the right things
    - To avoid costly mistakes (computation time / funding)
    - Usually care the most for large scale projects

- We also care about verification…
    - To make sure our results are correct
    - To ensure our results are reproducible
    - For every single project!!!

Why do I care about verification?

If you wouldn’t trust a published mathematical result from someone who never double checked their work …

… then you shouldn’t trust results from an untested software pipeline either

… and yes, that includes **YOUR** code too!!!

Pros / Cons of Writing Tests

2

The Pros …

- You want to be sure your code **really** works
- Helps ensure the validity of your results (They eliminate human error)
- Saves time and effort in the long term (Find bugs early)
- Simplifies deployment and reproducibility
- You can refactor your code with confidence
- Forces you to write develop better code quality
- Forces you to be knowledgeable about the behavior and performance of your code
- They make it easier to add features

Pros / Cons of Writing Tests

The Cons …

- They require more time up front
- You end up with more code to maintain
- They don’t cover UI
- It takes a bit of practice to get good at

How do I test my code?

3

“Functional Testing”: A type of software testing that validates the software system against the functional requirements/specifications.

How do I test my code?

3

"Functional Testing": A type of software testing that validates the software system against the functional requirements/specifications.

Unit Testing: A method by which individual units of the source code are tested to determine if they are functionally correct.

How do I test my code?

3

“Functional Testing”: A type of software testing that validates the software system against the functional requirements/specifications.

(Key Concept!!!)

Unit Testing: A method by which individual units of the source code are tested to determine if they are functionally correct.

Let’s start with an example ->

```
from unittest import TestCase

def add(x, y):
    return x + y

class TestAdd(TestCase):
    def test_five_plus_four(self):
        self.assertEqual(add(5, 4), 9)
```

3

Example 1

There are other options to `unittest` (Thanks Brett!):
https://www.slant.co/versus/9148/9149/~unittest_vs_pytest

Step 1: Organizing Your Code

3

- Disorganized code is not testable code. Organized code usually testable

  Use functions/modules (or methods/classes in OOD) to separate code into logical units.

- “You can’t test a for loop” – MWV

- Design before you write
  - Ask what functionality you need
  - Decide how to write it
  - Write a well documented function

Step 1: Organizing Your Code

3

- Disorganized code is not testable code. Organized code usually testable

  Use functions/modules (or methods/classes in OOD) to separate code into logical units.

- “You can’t test a for loop” – MWV

- Design before you write
  - Ask what functionality you need
  - Decide how to write it
  - Write a well documented function

- Better quality code
- Fewer code revisions
- Faster development
- Less headaches

Things We Should Talk About… but Won't

3

SOLID: See https://medium.com/feedzaitech/writing-testable-code-b3201d4538eb for a great overview!

- **Single Responsibility Principle (SRP)**
  - Each software module should only have one reason to change.
- **Open/Closed Principle (OCP)**
  - Your classes should be open for extension but closed to modifications.
- **Liskov Substitution Principle (LSP)**
  - Objects of a superclass shall be replaceable with objects of its subclasses without breaking the application.
- **Interface Segregation Principle (ISP)**
  - No client should be forced to depend on methods it does not use.
- **Dependency Inversion Principle (DIP)**
  - High-level modules should not depend on low-level modules; both should depend on abstractions.

Step 2: Organizing Your Tests

3

One possible solution is to:

- Put tests in a “tests” directory located in the same place as your source code
- Give each module its own script
- Give each function its own class

```
# tests/my_module.py

class TestAdd(TestCase):
    def test_five_plus_four(self):
        # Magic

class TestSubtract(TestCase):
    # More magic

class TestNobelPrizePhysicsSimulation(testCase):
    def test_with_air_resistance(self):
        # Even more magic

    def test_in_a_vacuum(self):
        # A crazy, incredible amount of magic
```

3

Example 2

```
from unittest import TestCase

class TestStringMethods(TestCase):
    def test_upper(self):
        self.assertEqual('foo'.upper(), ‘FOO’)

    def test_isupper(self):
        self.assertTrue(‘FOO’.isupper())
        self.assertFalse(‘Foo’.isupper())

    def test_split(self):
        s = 'hello world’
        self.assertListEqual(s.split(), [‘hello’,’ ‘world’])
        # check that s.split fails when the separator is not a string
        self.assertRaises(TypeError, s.plit, 2)
```

By inheriting from the TestCase class, you get access to a comprehensive collection of prebuilt testing criteria!
This is all listed in the official docs!

3

Method Name	Checks that
assertEqual(a, b)	a == b
assertNotEqual(a, b)	a != b
assertTrue(x)	bool(x) is True
assertFalse(x)	bool(x) is False
assertIs(a, b)	a is b
assertIsNot(a, b)	a is not b
assertIsNone(x)	x is None
assertIsNotNone(x)	x is not None
assertIn(a, b)	a in b
assertNotIn(a, b)	a not in b
assertIsInstance(a, b)	isinstance(a, b)
assertNotIsInstance(a, b)	not isinstance(a, b)

Method Name	Checks that
assertAlmostEqual(a, b)	round(a-b, 7) == 0
assertNotAlmostEqual(a, b)	round(a-b, 7) != 0
assertGreater(a, b)	a > b
assertGreaterEqual(a, b)	a >= b
assertLess(a, b)	a < b
assertLessEqual(a, b)	a <= b
assertRegex(s, r)	r.search(s)
assertNotRegex(s, r)	not r.search(s)
assertCountEqual(a, b)	*a* and *b* have the same elements in the same number, regardless of their order.

"Production" style tests can be used to test the creation of exceptions, warnings, and log messages

Method Name	Checks that
assertRaises(exc, fun, *args, **kwds)	fun(*args, **kwds) raises *exc*
assertRaisesRegex(exc, r, fun, *args, **kwds)	fun(*args, **kwds) raises *exc* and the message matches regex *r*
assertWarns(warn, fun, *args, **kwds)	fun(*args, **kwds) raises *warn*
assertWarnsRegex(warn, r, fun, *args, **kwds)	fun(*args, **kwds) raises *warn* and the message matches regex *r*
assertLogs(logger, level)	The with block logs on *logger* with minimum *level*

Some general utilities

Method Name	Checks that
pass()	def setup(self)
fail()	def setupClass(self)
	@skipif

3

Type specific tests can be used to handle certain special cases.

Method Name	Checks that
assertMultiLineEqual(a, b)	strings
assertSequenceEqual(a, b)	sequences
assertListEqual(a, b)	lists
assertTupleEqual(a, b)	tuples
assertSetEqual(a, b)	sets or frozensets
assertDictEqual(a, b)	dicts

```python
def zp_bias(ref_temp: float, cal_temp : float, band: tuple, pwv : float):
    """Calculate the residual error in the photometric zero point due to PWV


    Args:
        ref_temp: The temperature of the star used to calibrate the image in Kelvin
        cal_temp: The temperature of another star in the same image
        band: An array specifying a photometric bandpass
        pwv:: The PWV concentration along line of sight in mm

    Returns:
        The error in magnitudes for the photometric zero point of the given band
    """


    # Values for reference star
    ref_mag = magnitude(ref_temp, band, 0)
    ref_mag_atm = magnitude(ref_temp, band, pwv)
    ref_zero_point = ref_mag - ref_mag_atm

    # Values for star being calibrated
    cal_mag = magnitude(cal_temp, band, 0)
    cal_mag_atm = magnitude(cal_temp, band, pwv)
    cal_zero_point = cal_mag - cal_mag_atm

    return cal_zero_point - ref_zero_point
```

3

Example 3

```python
def zp_bias(ref_temp: float, cal_temp : float, band: tuple, pwv : float):
    """Calculate the residual error in the photometric zero point due to PWV


    Args:
        ref_temp: The temperature of the star used to calibrate the image in Kelvin
        cal_temp: The temperature of another star in the same image
        band: An array specifying a photometric bandpass
        pwv:: The PWV concentration along line of sight in mm

    Returns:
        The error in magnitudes for the photometric zero point of the given band
    """

    # Values for reference star
    ref_mag = magnitude(ref_temp, band, 0)
    ref_mag_atm = magnitude(ref_temp, band, pwv)
    ref_zero_point = ref_mag - ref_mag_atm

    # Values for star being calibrated
    cal_mag = magnitude(cal_temp, band, 0)
    cal_mag_atm = magnitude(cal_temp, band, pwv)
    cal_zero_point = cal_mag - cal_mag_atm

    return cal_zero_point - ref_zero_point
```

3

Example 3

3

```
class ZeroPointBias(TestCase):
    """Tests for the function blackbody.zp_bias"""

    def test_same_temperature(self):
        """Tests that bias is zero for stars of same temperature"""

        msg = "Returned bias was non-zero"
        bias_3000 = zp_bias(3000, 3000, (7000, 8500), 13)
        self.assertEqual(0, bias_3000, msg)

        bias_6000 = zp_bias(6000, 6000, (8500, 10000), 13)
        self.assertEqual(0, bias_6000, msg)


    def test_returned_sign(self):
        """Tests that bias has expected sign"""

        msg = "Returned bias has incorrect sign"
        bias_3_6 = zp_bias(3000, 6000, (7000, 8500), 13)
        self.assertLess(0, bias_3_6, msg)

        bias_6_3 = zp_bias(6000, 3000, (7000, 8500), 13)
        self.assertGreater(0, bias_6_3, msg)
```

Example 3

Good Testing Practice

3

Try to …

- Emphasize the usage of test **UNITS**
- **Avoid test interdependence**
- **Keep tests short**
- **Hard setup = bad unit**

Try not to …

- Rely on network access
- Perform I/O tasks
- Rely on the file system / Hit a database
- Repeat yourself

Incorporating Tests in Your Workflow

4

Option 1: Run tests directly from a dedicated test script (Not Ideal)

Option 1: Have a dedicated test script

```
import unittest


<… some unit tests here… >


if __name__ == ‘__main__’:
    unittest.main()
```

This is cumbersome …

You will never, ever bring yourself to actually run this script …

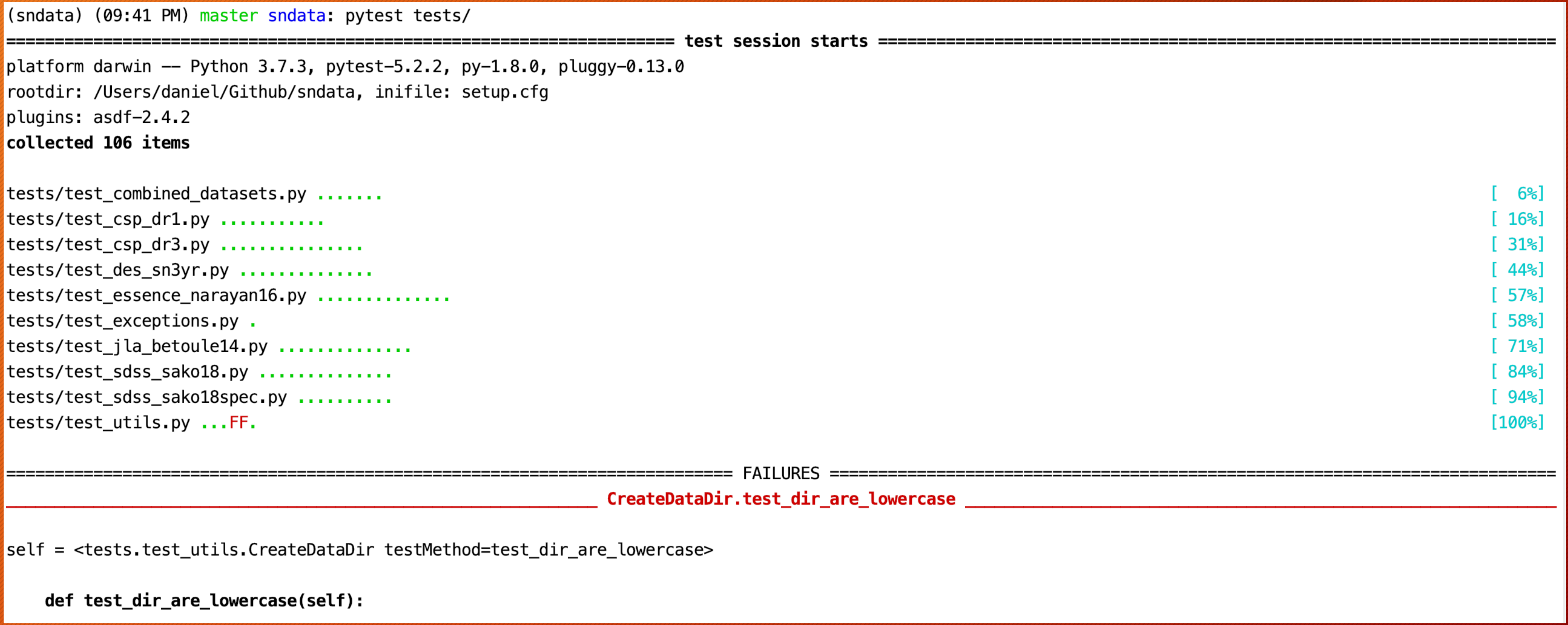
Incorporating Tests in Your Workflow

4

Option 1: Run tests directly from a dedicated test script (Not Ideal)

Option 2: Run your test suite from the command line (A better option)

Incorporating Tests in Your Workflow



```
(sndata) (09:41 PM) master sndata: pytest tests/
===================================================================== test session starts ======================================================================
platform darwin -- Python 3.7.3, pytest-5.2.2, py-1.8.0, pluggy-0.13.0
rootdir: /Users/daniel/Github/sndata, inifile: setup.cfg
plugins: asdf-2.4.2
collected 106 items

tests/test_combined_datasets.py .......                                                                                                                  [  6%]
tests/test_csp_dr1.py ...........                                                                                                                        [ 16%]
tests/test_csp_dr3.py ...............                                                                                                                    [ 31%]
tests/test_des_sn3yr.py ..............                                                                                                                   [ 44%]
tests/test_essence_narayan16.py ..............                                                                                                           [ 57%]
tests/test_exceptions.py .                                                                                                                               [ 58%]
tests/test_jla_betoule14.py ..............                                                                                                               [ 71%]
tests/test_sdss_sako18.py ..............                                                                                                                 [ 84%]
tests/test_sdss_sako18spec.py ..........                                                                                                                 [ 94%]
tests/test_utils.py ...FF.                                                                                                                               [100%]

=========================================================================== FAILURES ===========================================================================
_____________________________________________________________ CreateDataDir.test_dir_are_lowercase _____________________________________________________________

self = <tests.test_utils.CreateDataDir testMethod=test_dir_are_lowercase>

    def test_dir_are_lowercase(self):
```

Incorporating Tests in Your Workflow

4

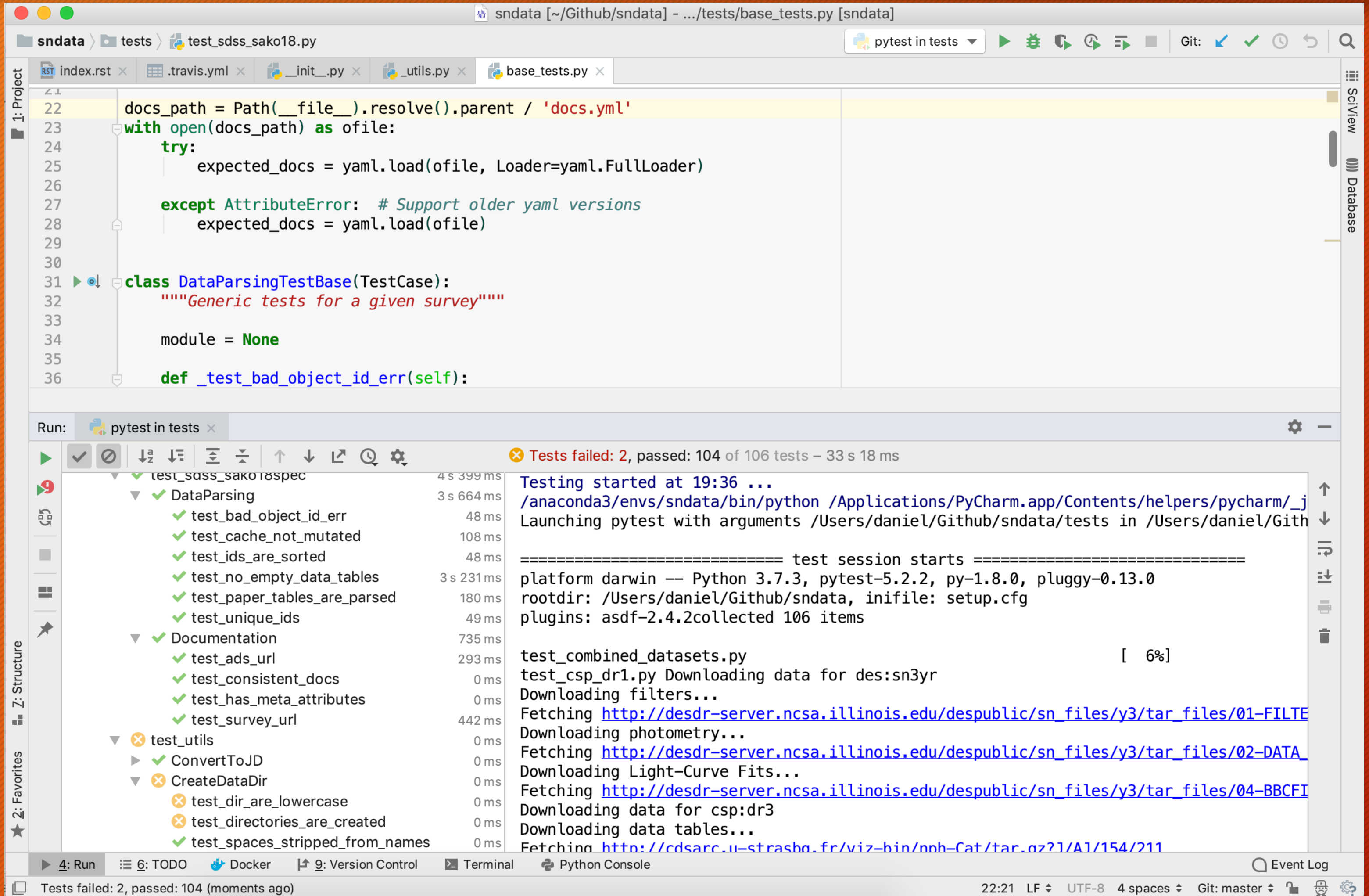
Option 1: Run tests directly from a dedicated test script (Not Ideal)

Option 2: Run your test suite using PyTest (A better option)

Option 3: Work within an IDE (A great option!)

Incorporating Tests in Your Workflow

4



```python
docs_path = Path(__file__).resolve().parent / 'docs.yml'
with open(docs_path) as ofile:
    try:
        expected_docs = yaml.load(ofile, Loader=yaml.FullLoader)

    except AttributeError:  # Support older yaml versions
        expected_docs = yaml.load(ofile)


class DataParsingTestBase(TestCase):
    """Generic tests for a given survey"""

    module = None

    def _test_bad_object_id_err(self):
```

Tests failed: 2, passed: 104 of 106 tests – 33 s 18 ms

```
▼ ✓ test_sdss_sako18spec                 4 s 399 ms
  ▼ ✓ DataParsing                        3 s 664 ms
      ✓ test_bad_object_id_err               48 ms
      ✓ test_cache_not_mutated              108 ms
      ✓ test_ids_are_sorted                  48 ms
      ✓ test_no_empty_data_tables        3 s 231 ms
      ✓ test_paper_tables_are_parsed        180 ms
      ✓ test_unique_ids                      49 ms
  ▼ ✓ Documentation                         735 ms
      ✓ test_ads_url                        293 ms
      ✓ test_consistent_docs                  0 ms
      ✓ test_has_meta_attributes              0 ms
      ✓ test_survey_url                     442 ms
▼ ☒ test_utils                                0 ms
  ▶ ✓ ConvertToJD                             0 ms
  ▼ ☒ CreateDataDir                           0 ms
      ☒ test_dir_are_lowercase                0 ms
      ☒ test_directories_are_created          0 ms
      ✓ test_spaces_stripped_from_names       0 ms
```

```
Testing started at 19:36 ...
/anaconda3/envs/sndata/bin/python /Applications/PyCharm.app/Contents/helpers/pycharm/_j
Launching pytest with arguments /Users/daniel/Github/sndata/tests in /Users/daniel/Gith

============================= test session starts ==============================
platform darwin -- Python 3.7.3, pytest-5.2.2, py-1.8.0, pluggy-0.13.0
rootdir: /Users/daniel/Github/sndata, inifile: setup.cfg
plugins: asdf-2.4.2collected 106 items

test_combined_datasets.py                                           [  6%]
test_csp_dr1.py Downloading data for des:sn3yr
Downloading filters...
Fetching http://desdr-server.ncsa.illinois.edu/despublic/sn_files/y3/tar_files/01-FILTE
Downloading photometry...
Fetching http://desdr-server.ncsa.illinois.edu/despublic/sn_files/y3/tar_files/02-DATA_
Downloading Light-Curve Fits...
Fetching http://desdr-server.ncsa.illinois.edu/despublic/sn_files/y3/tar_files/04-BBCFI
Downloading data for csp:dr3
Downloading data tables...
Fetching http://cdsarc.u-strasbg.fr/viz-bin/nph-Cat/tar.gz?J/AJ/154/211
```

Tests failed: 2, passed: 104 (moments ago)

- Runs at a key stroke
- Highlights test coverage from within the editor
- Makes it easy to run frequently as you commit

Incorporating Tests in Your Workflow

4

Option 1: Run tests directly from a dedicated test script (Not Ideal)

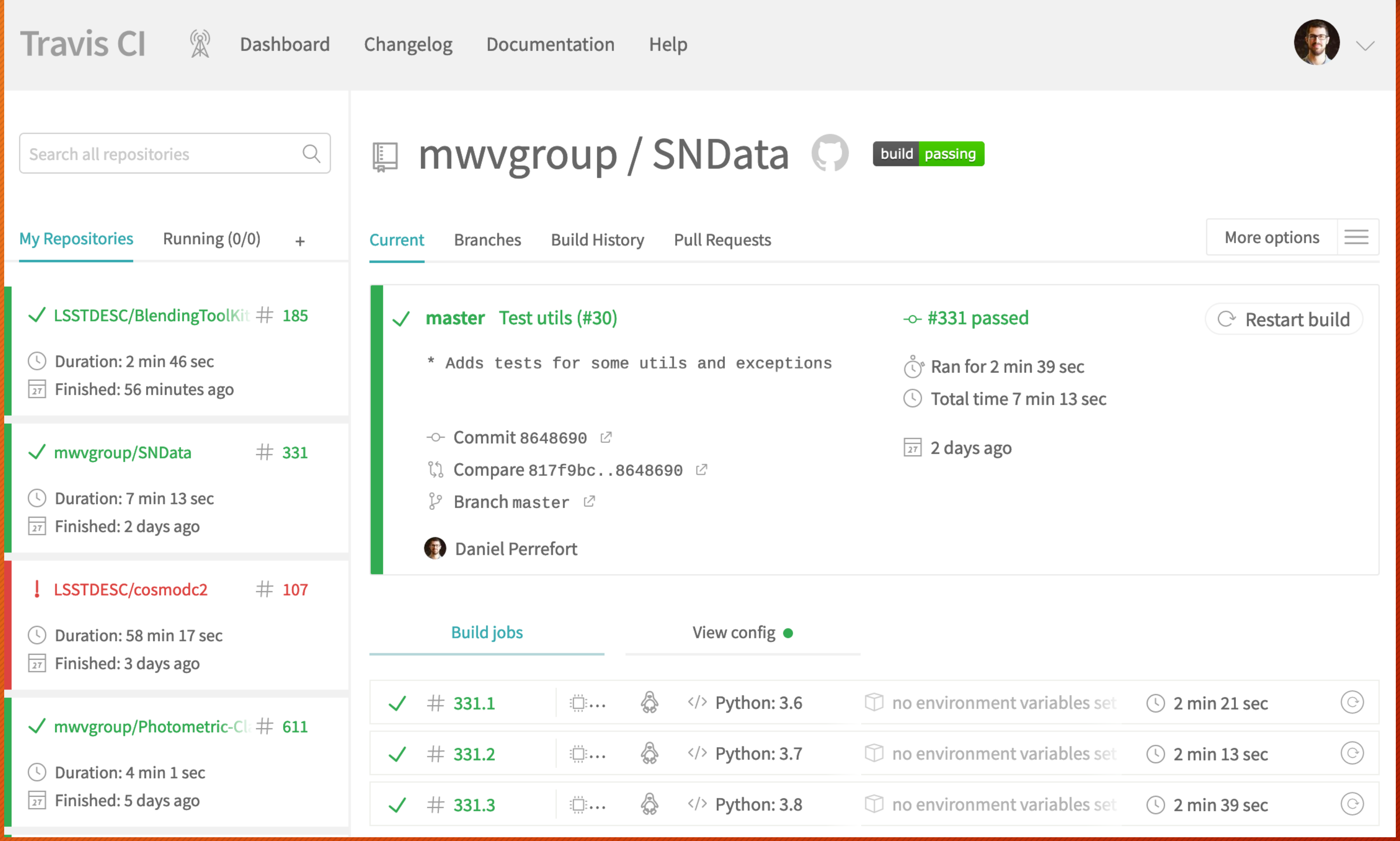
Option 2: Run your test suite using a CLI like PyTest (A better option)

Option 3: Work within an IDE (A great option!)

Option 4: Run tests with a continuous integration tool like Travis (Now we’re talking!!!)

Incorporating Tests in Your Workflow

4



Travis CI · Dashboard · Changelog · Documentation · Help

Search all repositories

My Repositories · Running (0/0) · +

✓ LSSTDESC/BlendingToolKit # 185
Duration: 2 min 46 sec
Finished: 56 minutes ago

✓ mwvgroup/SNData # 331
Duration: 7 min 13 sec
Finished: 2 days ago

! LSSTDESC/cosmodc2 # 107
Duration: 58 min 17 sec
Finished: 3 days ago

✓ mwvgroup/Photometric-Cl # 611
Duration: 4 min 1 sec
Finished: 5 days ago

mwvgroup / SNData

build passing

Current · Branches · Build History · Pull Requests · More options

✓ master Test utils (#30)

```
* Adds tests for some utils and exceptions
```

Commit 8648690
Compare 817f9bc..8648690
Branch master
Daniel Perrefort

#331 passed
Ran for 2 min 39 sec
Total time 7 min 13 sec
2 days ago

Restart build

Build jobs · View config ●

✓ # 331.1	Python: 3.6	no environment variables set	2 min 21 sec
✓ # 331.2	Python: 3.7	no environment variables set	2 min 13 sec
✓ # 331.3	Python: 3.8	no environment variables set	2 min 39 sec

- Runs automatically on every branch, every time!
- Keeps you focused on developing without stopping to run tests
- Easily customized to your needs via a config file
- Email alerts once something goes wrong
- Can run multiple OS / Python combos

In Summary…

1. Building the right code / building code right (validation/verification)

2. Tests provide multiple benefits, but do require some time commitment
   - Ensures the desired behavior
   - Trades off upfront development time for savings down the road
   - Faster bug identification and correction

3. `unittest` is built into Python and provides extensive, prebuilt functionality

4. Look for ways to incorporate tests into your DE – Travis can help