Unit Testing for Data Utilization and Management

Michael Penhallegon

Introduction

As the complexity of software and scrpt development continues withing the Seattle KSR Team, there is a chance to strengten software engineering concepts to improve quality, increase productivity and drive to more design-thinking/behavior driven development

Unit testing is a software engineering concept to write test for individual components of code to check conditions, demostrate base functionality and to codify software requirements. While in more "pure" and theoretical applications, unit tests are seen to confirm invariants and codify mathematical-proof concept, unit tests in pragramtic practice can work well in a grounds-up iterative development style. Finally, unit test can become part of a toolset of data testing to further data quality in addition to software/code.

In this document we will review a few librarie and packages across two languages, R and Python. Part of the rason to include Python in this context is to demonstrate well developed and utilized unti testing framework. Then we will apply those concept to R, which in the author's opinion, is significantly under-developed[1] and the mainstream commonly used package for it leave a lot to be desired.

Unit testing in a classical defintion should be used to test individual components or units of code. Itergration testing, testing entire system/applications and the intergration between that system and other software, is a complentary concept and similar package/libraries cen be used to intergrate test. Pragmatically, in the context of Knowledge Science Research, both can be useful and concorent.

Python Unit Testing

For Python unit testing there are a few different libraries for unit testing:

- unittest – included in the standard libary
- pytest

...and quite a few others. For this demo, the focus will be on pytest.

Writing a Basic Test

Unit testing is built on the concept of `assert` in many programming languages. An assert statement normally will take a conditional expresssion. such as `1 == 2` and will stop the code with an `AssertionError` in Python. This is similar function to R's `stopifnot` function, ultimating stopping execution. Pytest using assert as a way to set conditions for a passing or failing return on a testing function.

As an example, write a `plus4` function that adds 4 to any number inputted into the function:

```python
def plus4(x):
    return 4 + x
```

for a first step in a testing suite, a series of a related tests, just test that inputting 2 `plus4` returns 6 (2 + 4 = 6). Like:

```python
import pytest
def test_plus4():
    assert plus4(2) == 6
```

Then run your tests by:

```
python -m pytest
```

will return:

```
================================== test session starts ===================================
platform linux -- Python 3.8.3, pytest-5.4.3, py-1.9.0, pluggy-0.13.1
rootdir: /home/crimson/Documents/develop/unit_test_training/python, inifile: pytest.ini
collected 1 item

tests/test_basics.py .                                                              [100%]

=================================== 1 passed in 0.01s ====================================
```

This is basic but not super useful, so instead, let's look at an edge case, in this case infinity, `math.inf`. We will write a test we know should fail in this implementation:

```python
import math

def test_inf_plus4():
    assert plus4(math.inf) == 0
```

when pytest is run, the result will include a failure:

```
[...]
tests/test_basics.py .F                                                             [100%]

======================================= FAILURES ========================================
____________________________________ test_max_int ______________________________________

    def test_max_int():
>       assert plus4(math.inf) == 0
E       assert inf == 0
E        +  where inf = plus4(inf)
E        +    where inf = math.inf

tests/test_basics.py:9: AssertionError
================================ short test summary info =================================
FAILED tests/test_basics.py::test_max_int - assert inf == 0
============================== 1 failed, 1 passed in 0.03s ===============================
```

Any test you write should be able to failable, otherwise it is not useful. This seems obvious in this example, but in more complex algorthims, that can be challenging to built.

Any conditional expression can be used so as an advance example (see the code sample for loading data):

```python
import pandas as pd

# Create some fixtures, objects that can be used to test something
df = pd.read_parquet("data/random_norms.parquet")

def test_df_columns():
    expect_column_subset = ["a", "b"]
    # let's use the power of applied math to check columns, not expecting c
    assert set(df.columns).issubset(expect_column_subset)

def test_df_for_empties():
    assert any([
        len(df[column_name].dropna()) == len(df[column_name])
        for column_name in list(df.columns)])
```

R

Unit testing packages seem to have less consensus. Packages include:

- stopifnot[???]
- testthat[2][^1 must use devtools::install_github("hadley/testthat#594") for now]
- assertthat[3]
- assertr[4]
- ensurer[5]
- assertive[6]
- checkmate[7]
- testit[8]
- pointblank[9]
- RUnit
- and more…

More research on our team should be done to evaulate all these options

References

1. Bryan J. Vignette on using testthat outside the context of a package · issue #659 · r-lib/testthat. GitHub. Published October 2017. Accessed July 10, 2020. https://github.com/r-lib/testthat/issues/659

2. Wickham H, RStudio, utils::recover()) RC team (of. *Testthat: Unit Testing for R.*; 2020. Accessed July 10, 2020. https://CRAN.R-project.org/package=testthat

3. Wickham H. *Assertthat: Easy Pre and Post Assertions.*; 2019. Accessed July 10, 2020. https://CRAN.R-project.org/package=assertthat

4. Fischetti T. *Assertr: Assertive Programming for R Analysis Pipelines.*; 2020. Accessed July 10, 2020. https://CRAN.R-project.org/package=assertr

5. Bache SM. *Ensurer: Ensure Values at Runtime.*; 2015. Accessed July 10, 2020. https://CRAN.R-project.org/package=ensurer

6. Cotton R. *Assertive: Readable Check Functions to Ensure Code Integrity.*; 2016. Accessed July 10, 2020. https://CRAN.R-project.org/package=assertive

7. Lang M, Bischl B. *Checkmate: Fast and Versatile Argument Checks.*; 2020. Accessed July 10, 2020. https://CRAN.R-project.org/package=checkmate

8. Xie Y, Mortimer S. *Testit: A Simple Package for Testing R Packages.*; 2019. Accessed July 10, 2020. https://CRAN.R-project.org/package=testit

9. Iannone R. *Pointblank: Validation of Local and Remote Data Tables.*; 2020. Accessed July 10, 2020. https://CRAN.R-project.org/package=pointblank

10. Bioconducter. Unit testing. Bioconducter. Accessed July 10, 2020. http://bioconductor.org/developers/how-to/unitTesting-guidelines/