Assignment 7

MACS 30000

Jiaxu Han

1. Unit Testing in Python.

Problem 1

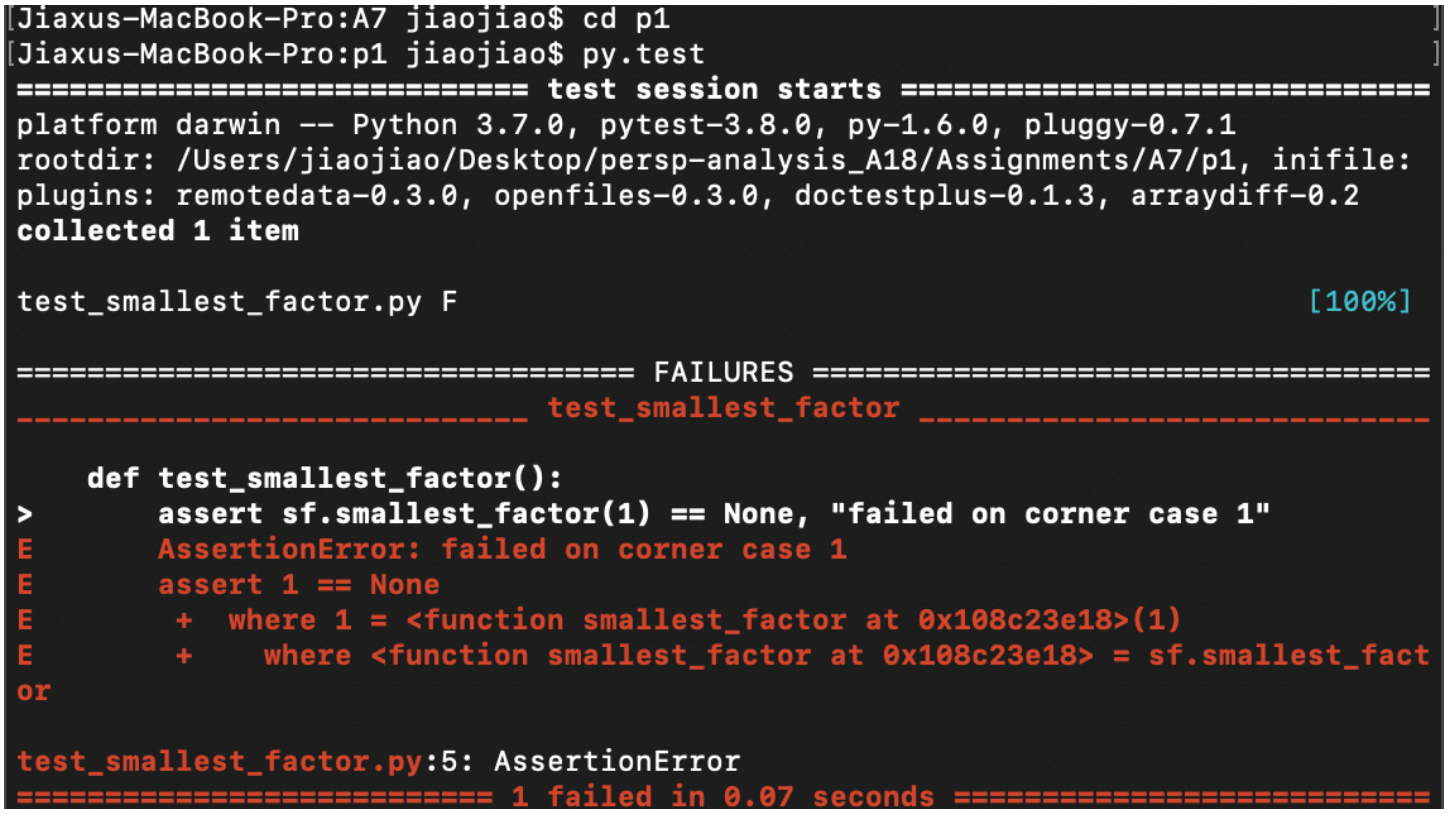
The function that needs to be tested in problem 1 is stored in "smallest_factor.py" that you can find in folder "p1". The code of the original function is:

```
# function needs to be tested in problem 1
def smallest_factor(n):
    """Return the smallest prime factor of the positive integer n."""
    if n == 1: return 1
    for i in range(2, int(n**.5)):
        if n % i == 0: return i
    return n
```

The unit testing for above function is stored in "test_smallest_factor.py" that you can find in folder "p1". It includes six cases. First of all, I included integer 1 and 2 as two special cases that need to be examined. Technically speaking, 1 is not a prime number and it does not have a prime factor either (https://en.wikipedia.org/wiki/Table_of_prime_factors). Therefore, the function should return None if the input is 1. 2, on the other hand, is a prime number and its smallest prime factor is itself. As a result, the function should return 2 if the input is 2. In addition to that, I also included 4 which is the square of the prime number 2 and is an even number; 103 which itself is a prime number; 142 which is a random even number; and 25 which is the square of the prime number 5 and is an odd number. The code I wrote is:

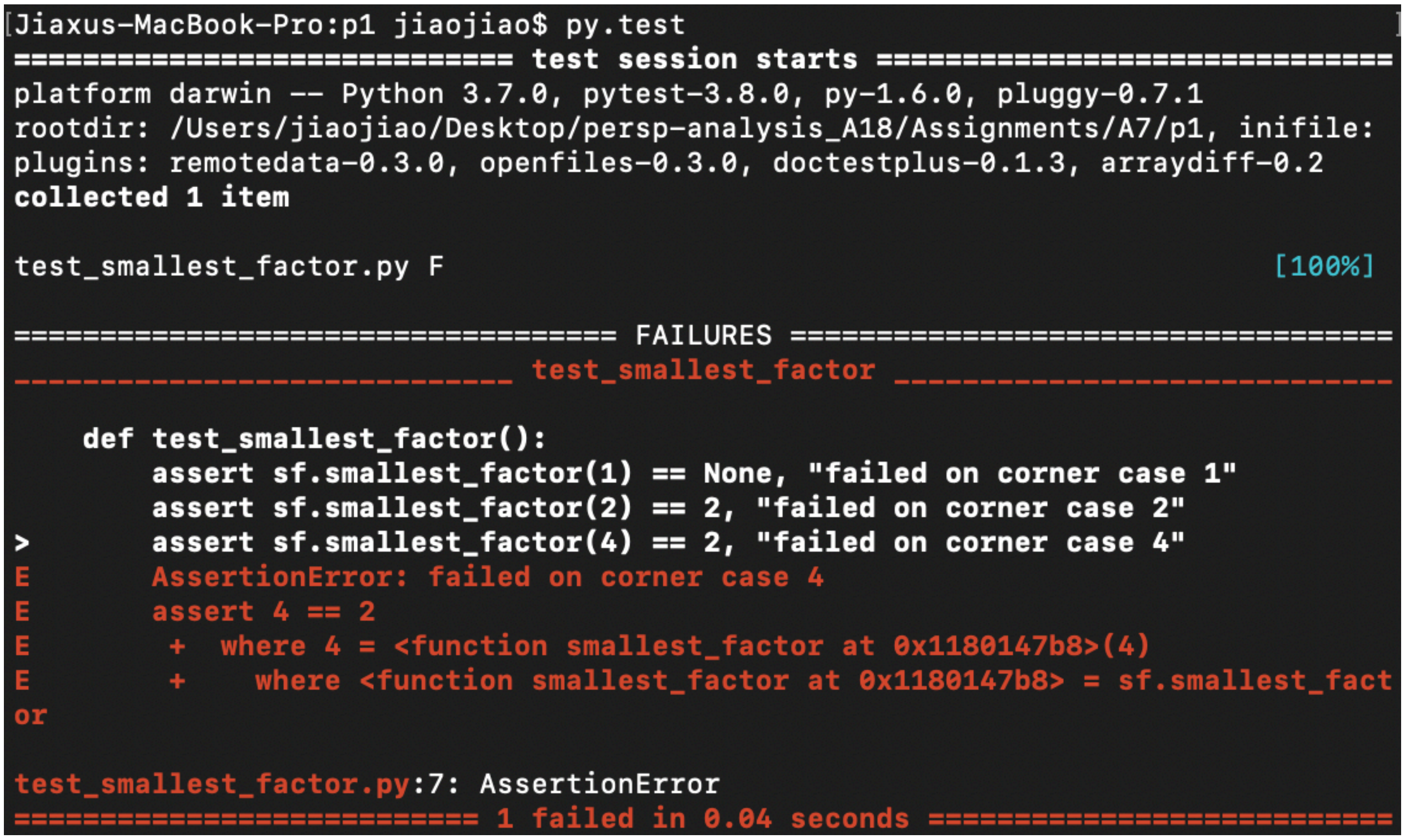
```
#unit testing for smallest_factor in problem 1
import smallest_factor as sf
def test_smallest_factor():
    assert sf.smallest_factor(1) == None, "failed on corner case 1"
    assert sf.smallest_factor(2) == 2, "failed on corner case 2"
    assert sf.smallest_factor(4) == 2, "failed on corner case 4"
    assert sf.smallest_factor(103) == 103, "failed on prime number"
    assert sf.smallest_factor(142) == 2, "failed on even integer"
    assert sf.smallest_factor(25) == 5, "failed on odd integer that is not
a prime number"
```

After running py.test in my terminal, the error message is:



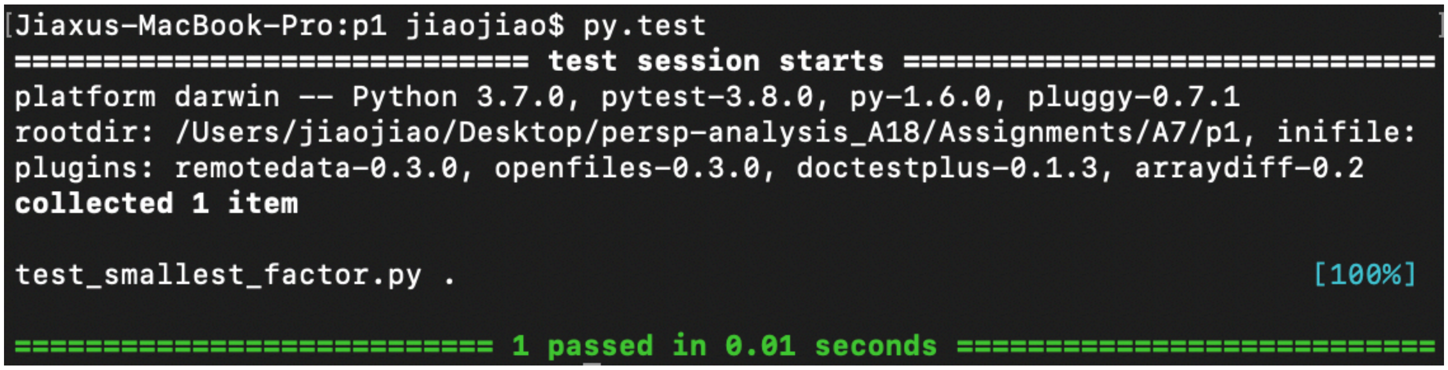
```
[Jiaxus-MacBook-Pro:A7 jiaojiao$ cd p1                                          ]
[Jiaxus-MacBook-Pro:p1 jiaojiao$ py.test                                        ]
============================= test session starts ==============================
platform darwin -- Python 3.7.0, pytest-3.8.0, py-1.6.0, pluggy-0.7.1
rootdir: /Users/jiaojiao/Desktop/persp-analysis_A18/Assignments/A7/p1, inifile:
plugins: remotedata-0.3.0, openfiles-0.3.0, doctestplus-0.1.3, arraydiff-0.2
collected 1 item

test_smallest_factor.py F                                                [100%]

=================================== FAILURES ===================================
_____________________________ test_smallest_factor _____________________________

    def test_smallest_factor():
>       assert sf.smallest_factor(1) == None, "failed on corner case 1"
E       AssertionError: failed on corner case 1
E       assert 1 == None
E        +  where 1 = <function smallest_factor at 0x108c23e18>(1)
E        +    where <function smallest_factor at 0x108c23e18> = sf.smallest_fact
or

test_smallest_factor.py:5: AssertionError
=========================== 1 failed in 0.07 seconds ===========================
```

We can see from above error message that the function failed when n = 1. The function should return None instead of 1. Therefore, I corrected the forth line of original function from “if n==1: return 1” to “if n==1: return None”. After running py.test again in Terminal, the error message became:



```
[Jiaxus-MacBook-Pro:p1 jiaojiao$ py.test                                        ]
============================= test session starts ==============================
platform darwin -- Python 3.7.0, pytest-3.8.0, py-1.6.0, pluggy-0.7.1
rootdir: /Users/jiaojiao/Desktop/persp-analysis_A18/Assignments/A7/p1, inifile:
plugins: remotedata-0.3.0, openfiles-0.3.0, doctestplus-0.1.3, arraydiff-0.2
collected 1 item

test_smallest_factor.py F                                                [100%]

=================================== FAILURES ===================================
_____________________________ test_smallest_factor _____________________________

    def test_smallest_factor():
        assert sf.smallest_factor(1) == None, "failed on corner case 1"
        assert sf.smallest_factor(2) == 2, "failed on corner case 2"
>       assert sf.smallest_factor(4) == 2, "failed on corner case 4"
E       AssertionError: failed on corner case 4
E       assert 4 == 2
E        +  where 4 = <function smallest_factor at 0x1180147b8>(4)
E        +    where <function smallest_factor at 0x1180147b8> = sf.smallest_fact
or

test_smallest_factor.py:7: AssertionError
=========================== 1 failed in 0.04 seconds ===========================
```

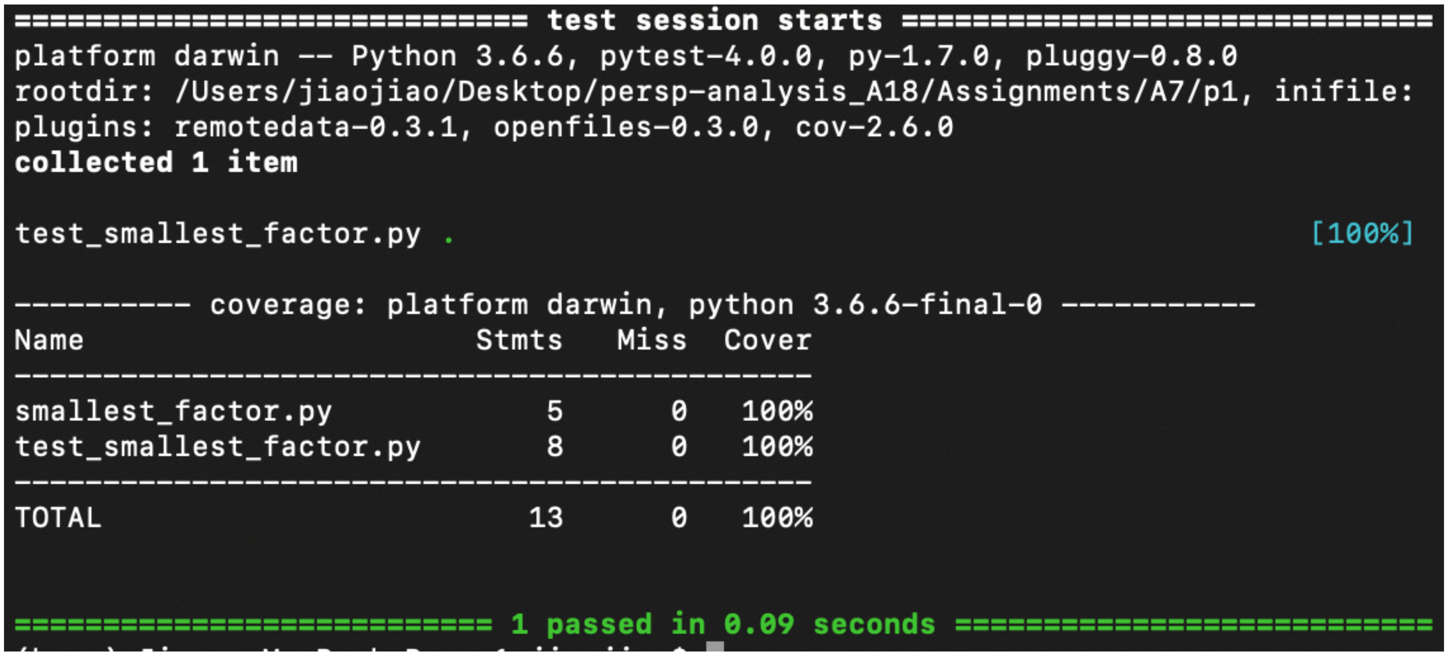
The above error message shows that the function failed again at the case when n = 4. The output should be 2 instead of 4, and this is because 2 is actually not in “range(2, int(n**.5))”. Therefore, I corrected the fifth line of the code from “for i in range(2, int(n**.5)):” to “for i in range(2, int(n**.5) + 1)”. Then I run the py.test again, the function was able to pass the test:



```
Jiaxus-MacBook-Pro:p1 jiaojiao$ py.test
============================== test session starts ===============================
platform darwin -- Python 3.7.0, pytest-3.8.0, py-1.6.0, pluggy-0.7.1
rootdir: /Users/jiaojiao/Desktop/persp-analysis_A18/Assignments/A7/p1, inifile:
plugins: remotedata-0.3.0, openfiles-0.3.0, doctestplus-0.1.3, arraydiff-0.2
collected 1 item

test_smallest_factor.py .                                                  [100%]

=========================== 1 passed in 0.01 seconds ============================
```

Problem 2

(1) After the installation of pytest-cov, I checked the coverage of smallest_factor() from problem 1, and the result is 100% coverage:



```
============================== test session starts ===============================
platform darwin -- Python 3.6.6, pytest-4.0.0, py-1.7.0, pluggy-0.8.0
rootdir: /Users/jiaojiao/Desktop/persp-analysis_A18/Assignments/A7/p1, inifile:
plugins: remotedata-0.3.1, openfiles-0.3.0, cov-2.6.0
collected 1 item

test_smallest_factor.py .                                                  [100%]

---------- coverage: platform darwin, python 3.6.6-final-0 -----------
Name                        Stmts   Miss  Cover
-----------------------------------------------
smallest_factor.py              5      0   100%
test_smallest_factor.py         8      0   100%
-----------------------------------------------
TOTAL                          13      0   100%


=========================== 1 passed in 0.09 seconds ============================
```

(2) You can find the function month_length( ) stored in file “month_length.py” in folder “p2”. The code of this function is:

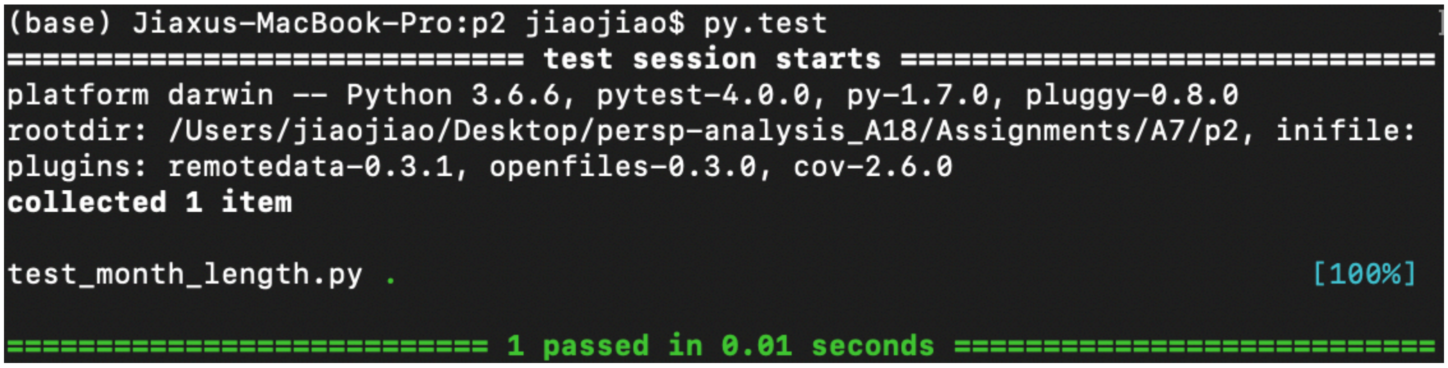
```
def month_length(month, leap_year=False):
    """Return the number of days in the given month."""
    if month in {"September", "April", "June", "November"}:
        return 30
    elif month in {"January", "March", "May", "July",
                        "August", "October", "December"}:
```

```
        return 31
    if month == "February":
        if not leap_year:
            return 28
        else:
            return 29
    else:
        return None
```

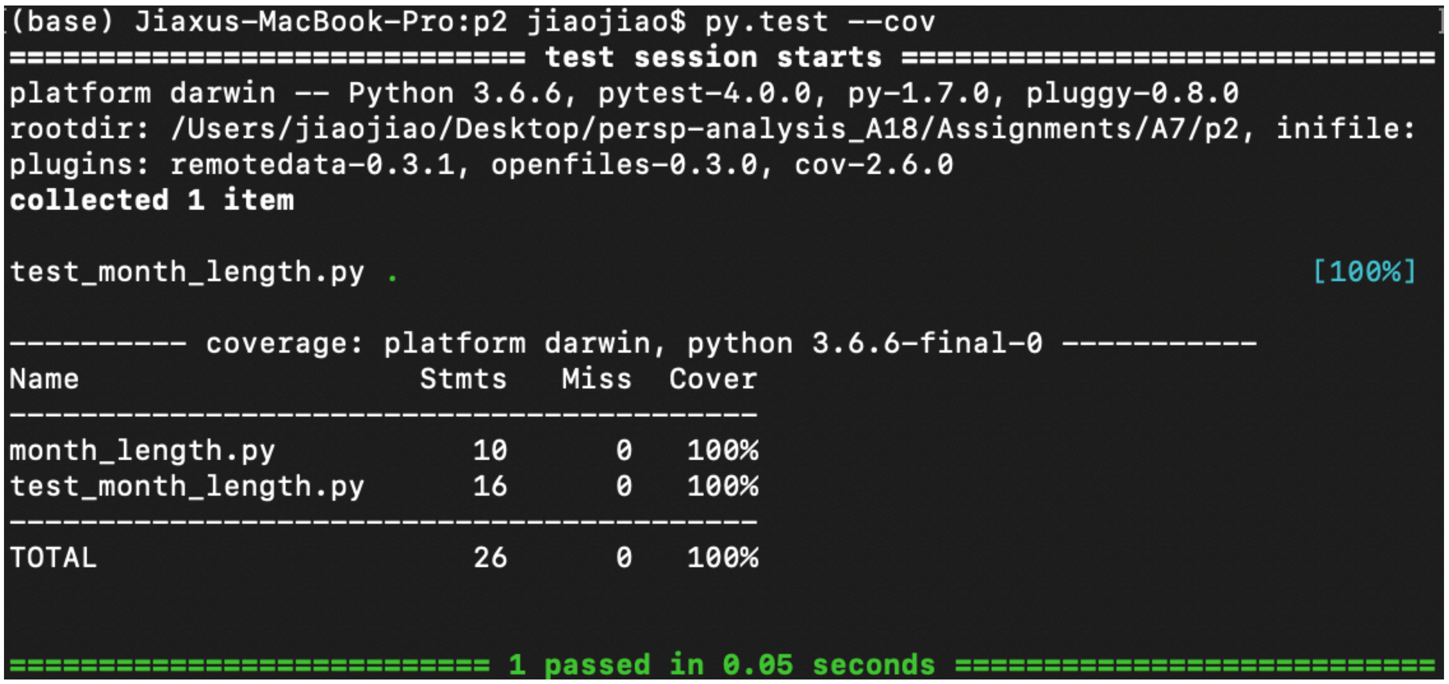
The unit testing for the above function is stored in file "test_month_length" in folder "p2". The code of the unit testing is:

```
# unit testing for function month_length in problem 2
import month_length as ml

def test_month_length():
    assert ml.month_length('January') == 31, "unexpected days for January"
    assert ml.month_length('February') == 28, "unexpected days for 
February, not leap year"
    assert ml.month_length('February', leap_year = True) == 29, 
"unexpected days for February, leap year"
    assert ml.month_length('March') == 31, "unexpected days for March"
    assert ml.month_length('April') == 30, "unexpected days for April"
    assert ml.month_length('May') == 31, "unexpected days for May"
    assert ml.month_length('June') == 30, "unexpected days for June"
    assert ml.month_length('July') == 31, "unexpected days for July"
    assert ml.month_length('August') == 31, "unexpected days for August"
    assert ml.month_length('September') == 30, "unexpected days for 
September"
    assert ml.month_length('October') == 31, "unexpected days for October"
    assert ml.month_length('November') == 30, "unexpected days for 
November"
    assert ml.month_length('December') == 31, "unexpected days for 
December"
    assert ml.month_length('foo') == None, "This is not a month”
```

After running py.test as well as py.test --cov, the function month_length was able to pass the tests and the unit testing has a full coverage.



```
(base) Jiaxus-MacBook-Pro:p2 jiaojiao$ py.test
============================ test session starts =============================
platform darwin -- Python 3.6.6, pytest-4.0.0, py-1.7.0, pluggy-0.8.0
rootdir: /Users/jiaojiao/Desktop/persp-analysis_A18/Assignments/A7/p2, inifile:
plugins: remotedata-0.3.1, openfiles-0.3.0, cov-2.6.0
collected 1 item

test_month_length.py .                                                  [100%]

========================== 1 passed in 0.01 seconds ==========================
```



```
(base) Jiaxus-MacBook-Pro:p2 jiaojiao$ py.test --cov
============================== test session starts ================================
platform darwin -- Python 3.6.6, pytest-4.0.0, py-1.7.0, pluggy-0.8.0
rootdir: /Users/jiaojiao/Desktop/persp-analysis_A18/Assignments/A7/p2, inifile:
plugins: remotedata-0.3.1, openfiles-0.3.0, cov-2.6.0
collected 1 item

test_month_length.py .                                                       [100%]

---------- coverage: platform darwin, python 3.6.6-final-0 -----------
Name                     Stmts   Miss  Cover
--------------------------------------------
month_length.py             10      0   100%
test_month_length.py        16      0   100%
--------------------------------------------
TOTAL                       26      0   100%


=========================== 1 passed in 0.05 seconds ============================
```

Problem 3

You can find the function operate( ) that needs to be tested in file "operate.py" in folder "p3". The code for this function is:

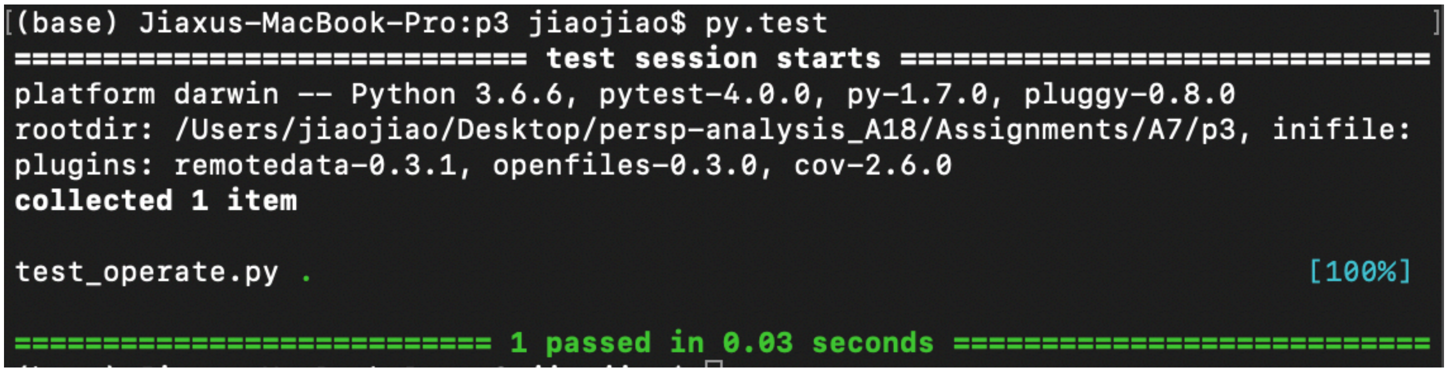
```python
#problem 3
def operate(a, b, oper):
    """Apply an arithmetic operation to a and b."""
    if type(oper) is not str:
        raise TypeError("oper must be a string")
    elif oper == '+':
        return a + b
    elif oper == '-':
        return a - b
    elif oper == '*':
        return a * b
    elif oper == '/':
        if b == 0:
            raise ZeroDivisionError("division by zero is undefined")
        return a / b
    raise ValueError("oper must be one of '+', '/', '-', or '*'")
```

The unit testing for above function is stored in file "test_operate.py" in folder "p3". The code of the unit testing is:

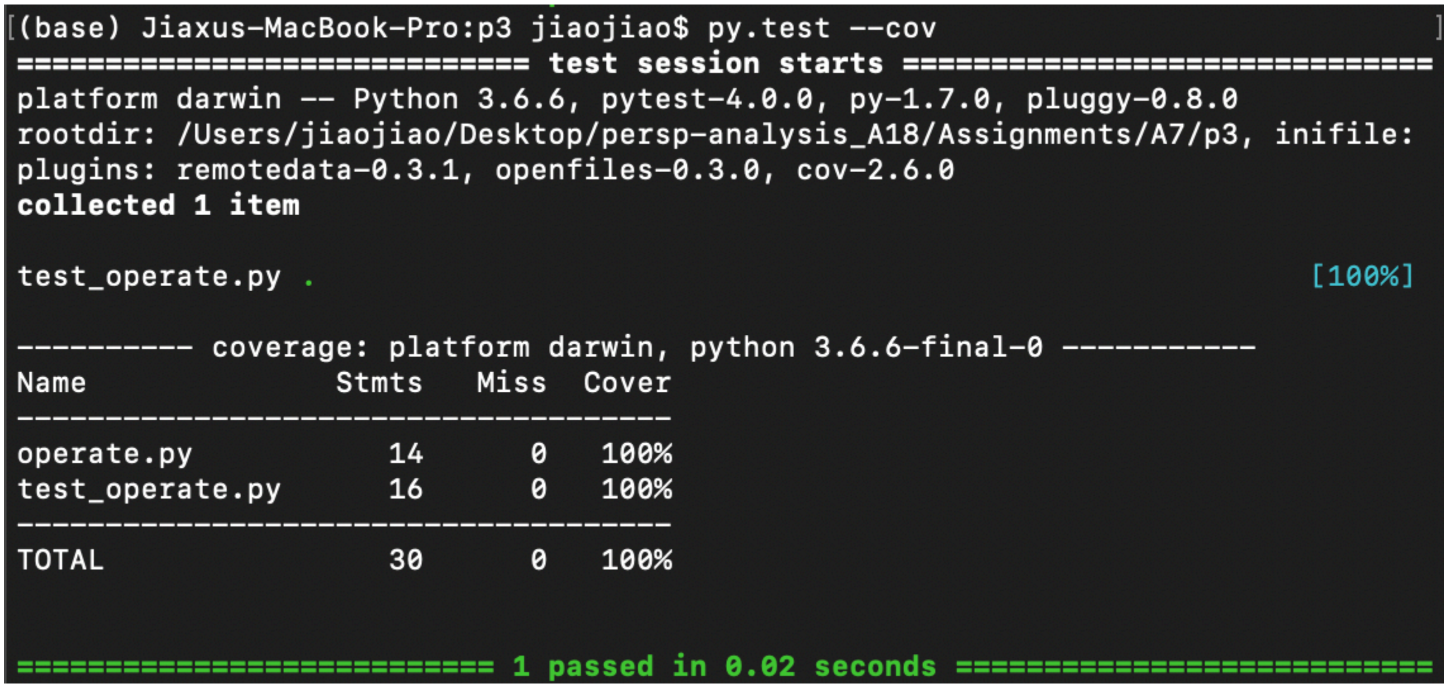
```python
#unit testing for function operate

import pytest
import operate as op
```

```
def test_operate():
    with pytest.raises(TypeError) as excinfo:
        op.operate(1, 1, 7)
    assert excinfo.value.args[0] == "oper must be a string"
    assert op.operate(3, 5, '+') == 8, "unexpected value using addition"
    assert op.operate(7, 8, '-') == -1, "unexpected value using
subtraction"
    assert op.operate(3, 0, '*') == 0, "unexpected value using
mutiplication"
    assert op.operate(5, 2, '/') == 2.5, "unexpected value using division"
    with pytest.raises(ZeroDivisionError) as excinfo:
        op.operate(2, 0, '/')
    assert excinfo.value.args[0] == "division by zero is undefined"
    with pytest.raises(ValueError) as excinfo:
        op.operate(1, 4, '^')
    assert excinfo.value.args[0] == "oper must be one of '+', '/', '-', or
'*'"
```

After running py.test as well as py.test --cov and its cov-report tool, the function operate( ) was able to pass the tests and the unit testing had a comprehensive coverage.

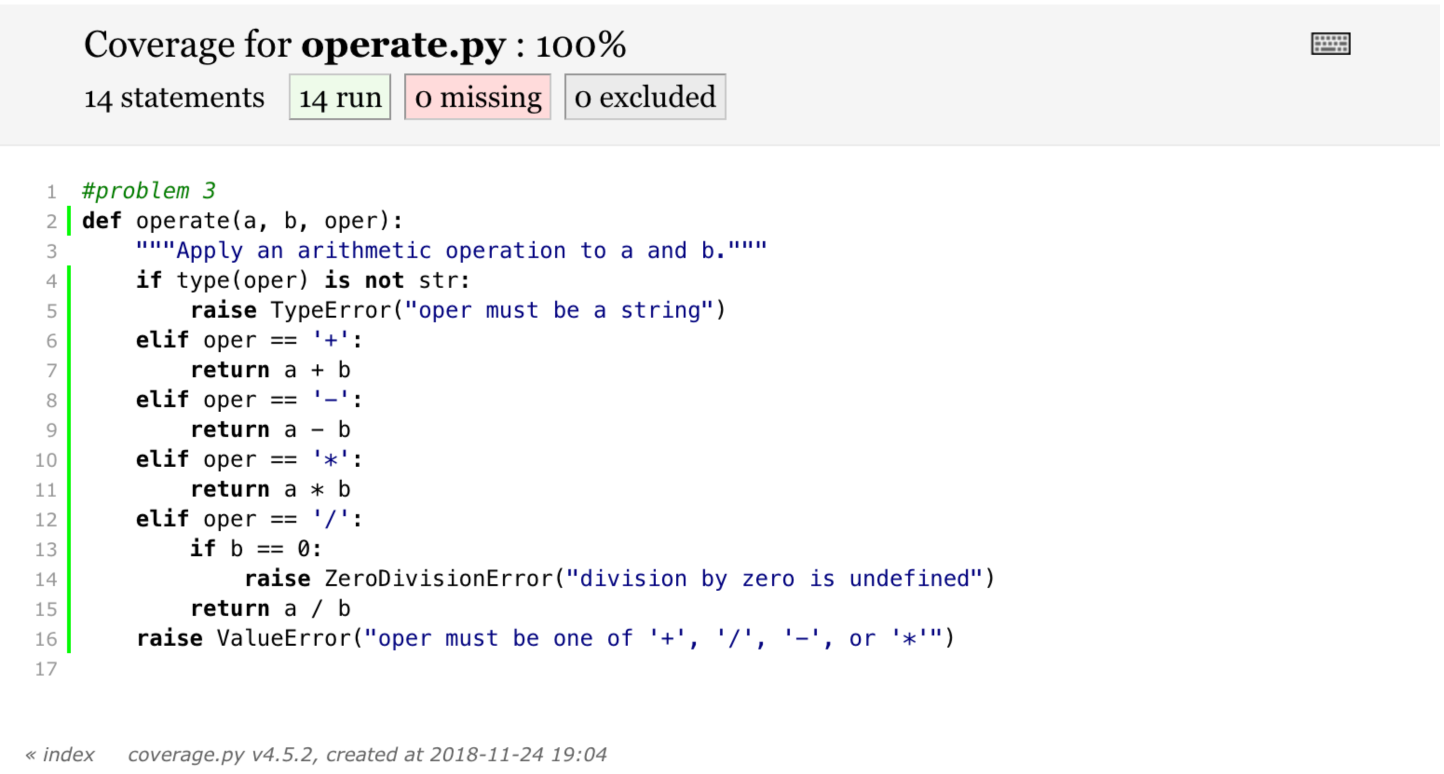


```
(base) Jiaxus-MacBook-Pro:p3 jiaojiao$ py.test
============================== test session starts ===============================
platform darwin -- Python 3.6.6, pytest-4.0.0, py-1.7.0, pluggy-0.8.0
rootdir: /Users/jiaojiao/Desktop/persp-analysis_A18/Assignments/A7/p3, inifile:
plugins: remotedata-0.3.1, openfiles-0.3.0, cov-2.6.0
collected 1 item

test_operate.py .                                                          [100%]

============================ 1 passed in 0.03 seconds ============================
```



```
(base) Jiaxus-MacBook-Pro:p3 jiaojiao$ py.test --cov
============================== test session starts ===============================
platform darwin -- Python 3.6.6, pytest-4.0.0, py-1.7.0, pluggy-0.8.0
rootdir: /Users/jiaojiao/Desktop/persp-analysis_A18/Assignments/A7/p3, inifile:
plugins: remotedata-0.3.1, openfiles-0.3.0, cov-2.6.0
collected 1 item

test_operate.py .                                                          [100%]

---------- coverage: platform darwin, python 3.6.6-final-0 -----------
Name                 Stmts   Miss  Cover
----------------------------------------
operate.py              14      0   100%
test_operate.py         16      0   100%
----------------------------------------
TOTAL                   30      0   100%


============================ 1 passed in 0.02 seconds ============================
```

You can also find the following reports in htmlcov folder in p3.

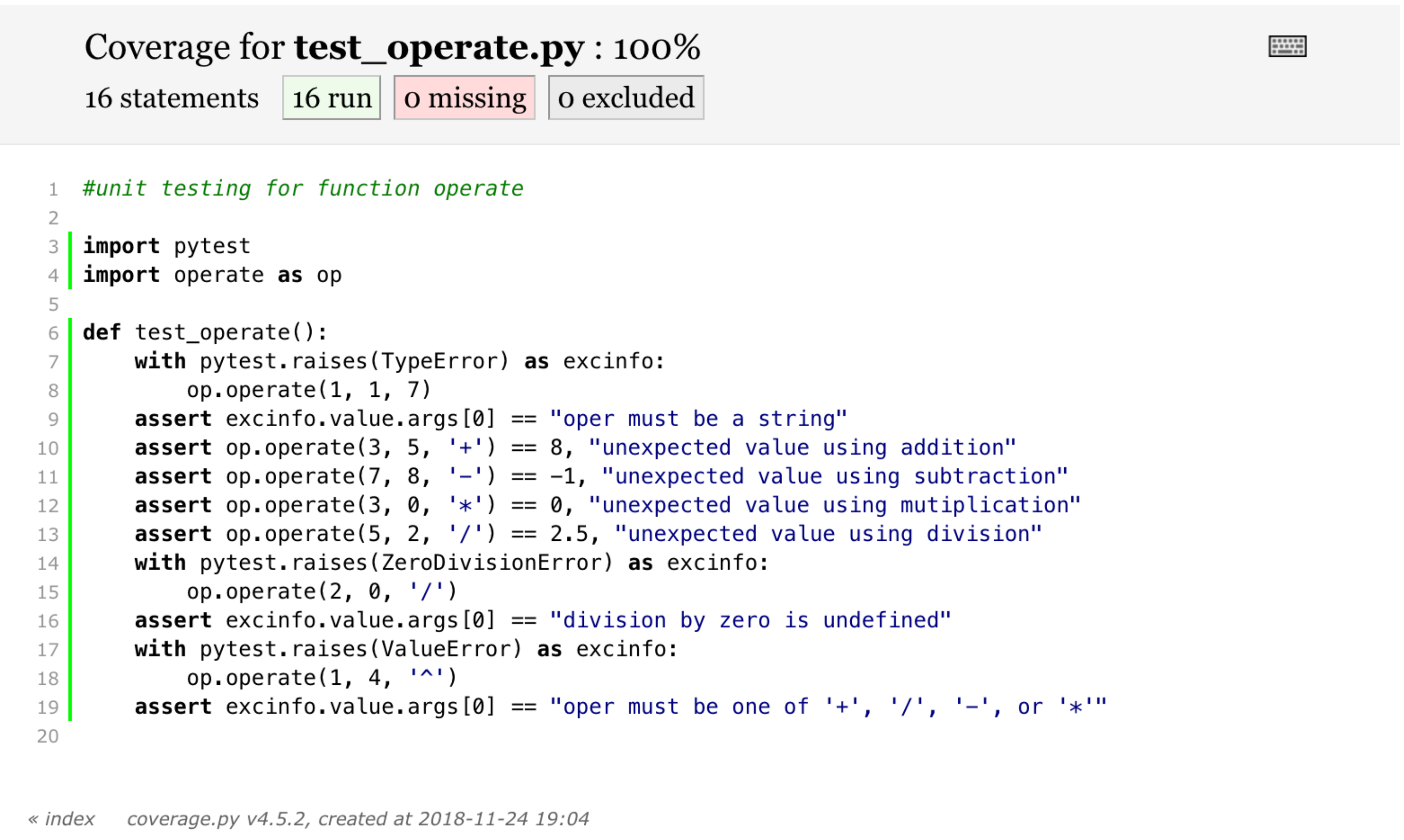


Coverage for **operate.py** : 100%

14 statements | 14 run | 0 missing | 0 excluded

```python
#problem 3
def operate(a, b, oper):
    """Apply an arithmetic operation to a and b."""
    if type(oper) is not str:
        raise TypeError("oper must be a string")
    elif oper == '+':
        return a + b
    elif oper == '-':
        return a - b
    elif oper == '*':
        return a * b
    elif oper == '/':
        if b == 0:
            raise ZeroDivisionError("division by zero is undefined")
        return a / b
    raise ValueError("oper must be one of '+', '/', '-', or '*'")

```

« index   coverage.py v4.5.2, created at 2018-11-24 19:04



Coverage for **test_operate.py** : 100%

16 statements | 16 run | 0 missing | 0 excluded

```python
#unit testing for function operate

import pytest
import operate as op

def test_operate():
    with pytest.raises(TypeError) as excinfo:
        op.operate(1, 1, 7)
    assert excinfo.value.args[0] == "oper must be a string"
    assert op.operate(3, 5, '+') == 8, "unexpected value using addition"
    assert op.operate(7, 8, '-') == -1, "unexpected value using subtraction"
    assert op.operate(3, 0, '*') == 0, "unexpected value using mutiplication"
    assert op.operate(5, 2, '/') == 2.5, "unexpected value using division"
    with pytest.raises(ZeroDivisionError) as excinfo:
        op.operate(2, 0, '/')
    assert excinfo.value.args[0] == "division by zero is undefined"
    with pytest.raises(ValueError) as excinfo:
        op.operate(1, 4, '^')
    assert excinfo.value.args[0] == "oper must be one of '+', '/', '-', or '*'"

```

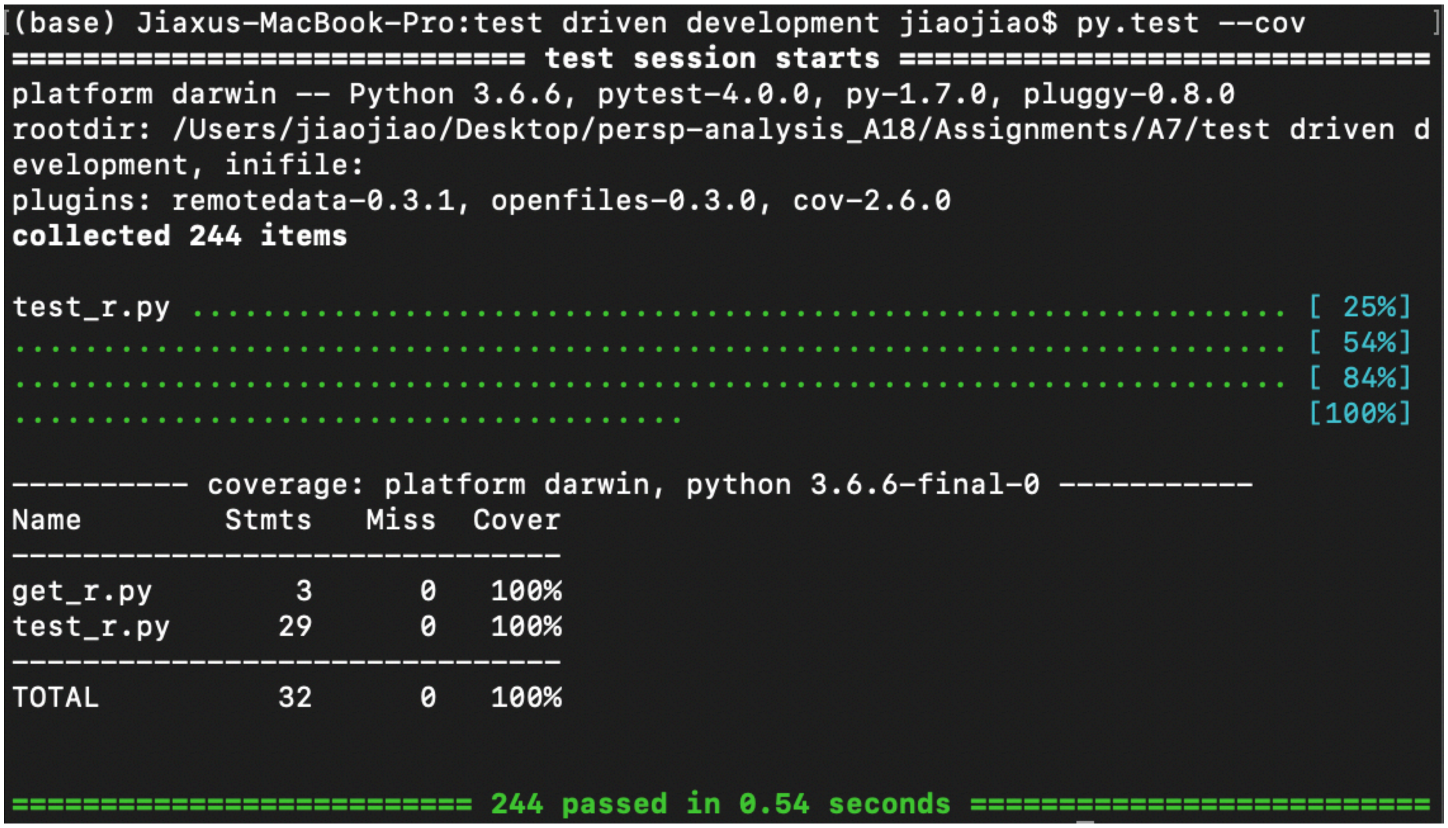
« index   coverage.py v4.5.2, created at 2018-11-24 19:04

2. Test driven development.

The function get_r( ) is stored in file get_r.py in folder “test driven development”, and the code is:

```python
def get_r(K, L, alpha, Z, delta):
    '''
    This function generates the interest rate or vector of interest rates
    '''
    r = alpha * Z * ((L / K) ** (1 - alpha)) - delta
    return r
```

After running py.test --cov, the above function was able to pass the tests:



```
(base) Jiaxus-MacBook-Pro:test driven development jiaojiao$ py.test --cov
============================ test session starts =============================
platform darwin -- Python 3.6.6, pytest-4.0.0, py-1.7.0, pluggy-0.8.0
rootdir: /Users/jiaojiao/Desktop/persp-analysis_A18/Assignments/A7/test driven d
evelopment, inifile:
plugins: remotedata-0.3.1, openfiles-0.3.0, cov-2.6.0
collected 244 items

test_r.py ............................................................. [ 25%]
....................................................................... [ 54%]
....................................................................... [ 84%]
......................................                                  [100%]

---------- coverage: platform darwin, python 3.6.6-final-0 -----------
Name        Stmts   Miss  Cover
-------------------------------
get_r.py        3      0   100%
test_r.py      29      0   100%
-------------------------------
TOTAL          32      0   100%


========================= 244 passed in 0.54 seconds ==========================
```

3. Watts (2014).

In the paper “Common Sense and Sociological Explanations”, Watts (2014) discussed an important issue with great insight that sociologists often unconsciously rely on common sense and unavoidably “conflate understandability with causality” (Watts, 2014, p313), which cause many problems.

To support his argument, the author illustrated the case of Rational Choice Theory which was introduced into sociology in the late 1960s (Watts, 2014, p319). Though it has a “clear scientific aspiration”, as mentioned by Watts (2014, p319), it was heavily criticized by scholars from various fields. The main criticism was focused on the invalid assumptions underlying this theory about “the preferences, knowledge, and computational capabilities of the actors" (Watts, 2014, p320), as well as the inconsistency between the prediction and empirical results. In response to those criticisms, the advocates of this theory have been making adjustments in ways of interpreting the theory. However, as Watts (2014, p321) criticized, the proponents have shifted from a scientific approach to an empathic explanation approach over the course of defending the rational choice theory.

The author mentioned other examples that echo such paradox as the Rational Choice Theory: an intention for scientific rationality but fall into empathic reasoning eventually. Watts pointed out that the main reason underlying such paradox is using mental simulation to theorize human behavior. Watts pointed out that people often use mental simulation and rationalization of some behaviors interchangeably to make predictions without awareness, which render researchers to jump from commonsense to sociological theories without check (2014, p327). However, the problem is that the assumption of understandability is equivalent to causality is invalid (Watts, 2014, p327). In fact, just because an explanation makes sense, it doesn't necessarily lead to any "generalizable causal mechanisms" (Watts, 2014, p327), nor make accurate predictions. Thus, applying those theories formulated by mental simulation more broadly can "undermine the scientific validity of the resulting explanations" (Watts, 2014, p328).

To tackle those issues, the author summarized several approaches to produce more scientific rigorous explanations. Watts proposed that sociologists should rely more on experimental methods especially field experiments in addition to natural experiments, quasi-experiments, and laboratory experiments. An alternative to experiments is to use the model of causal inference coupled with computational methods, statistics, and econometrics on non-experimental data (Watts, 2014, p336). Additionally, Watts suggested that an alternative both to experiments and statistical models is to evaluate explanations more by their predictability, though prediction itself may "not be a necessary condition for causality" (Watts, 2014, p337).

[Addendum]
Watts cited a wide range of researches to formulate his concerns for conflating common sense and prediction with causality. However, he also expressed his disdain for theory in the social field built on "intuitive" assumptions. In my opinion, I agree with Watts in terms of the importance of scientific rigorous explanations, but sometimes it is very difficult to achieve that without some level of theoretical guidance. Much scientific research starts with proposing a model to describe what researchers have observed. Those theories may build on intuitions of the researchers but I would argue that most theories if not all are built on some level of common sense because researchers are humans too. Even in physics, one of the most scientifically rigorous fields, scientists have proposed many theories that were later proved wrong such as atom was once believed to be the fundamental building blocks of nature. However, those "wrong" models are still very important in advancing the area of the research, because theories are meant to be changed, edited, tested, improved and hopefully one day it will be able to objectively describe the reality. In social science, empathic explanation of a theory is dangerous but could be helpful sometimes. For example, there are many social psychologists studying human emotions. At the beginning of this research topic, there were no fMRI, EEG, or other tools to measure physiological reactions of the human body or look at the brain, researchers start by examining themselves and propose theories which were very flawed and intuitive. It is inevitable that researchers themselves become the convenient and immediate resource for studying emotions. However, some of the theories provided a direction or idea for testing. Basic emotion theories, for example, argues that a finite number of emotions are psychologically indivisible and are building blocks for more complex emotions, which has already been proved very problematic with modern technology. But this theory motivates social psychologists to redefine meaningful building blocks of emotion such as "core affect" to study emotions (Russell, 2003). In sum, Watts brought up very insightful and constructive opinions for sociology and it is very important to apply more rigorous and scientific methodologies in research. At the same time, I think that theories are important for science as well especially if it is formulated at the

beginning of a research paradigm and its later improvements are not based on understandability but more on scientific experiments.

References

Watts, D. (2014). Common Sense and Sociological Explanations. *American Journal of Sociology, 120 (2),* 313-351.

Russell, J. A. (2003). Core affect and the psychological construction of emotion. *Psychological Review, 110*(1), 145-172.