Assignment 7

Fulin Guo

1. **Problem 1:**

   I first stored the code of problem 1 in a file named smallest_factor.py Then, I wrote the unit test code in a file named test_1.py. Here is the test code:

```
# test_1.py
from smallest_factor import smallest_factor

def test_1():
        assert smallest_factor(2)==2
        assert smallest_factor(18)==2
        assert smallest_factor(17)==17
        assert smallest_factor(8)==2
```

   Finally, I ran the unit test by using py.test command and got the outcome as follows:

```
(base) FulindeMacBook-Pro:problem1 fulinguo$ 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/fulinguo/Desktop/problem1, inifile:
plugins: remotedata-0.3.1, openfiles-0.3.0, cov-2.6.0
collected 1 item

test_1.py F                                                              [100%]

=================================== FAILURES ===================================
_________________________________ test_1 _________________________________

    def test_1():
        assert smallest_factor(2)==2
        assert smallest_factor(18)==2
        assert smallest_factor(17)==17
        # assert smallest_factor(1)==1
>       assert smallest_factor(8)==2
E    assert 8 == 2
E     +  where 8 = smallest_factor(8)

test_1.py:8: AssertionError
=========================== 1 failed in 0.06 seconds ===========================
```

   The test failed as the smallest prime factor of 8 should be 2, but the function smallest_factor(8) returned 8. The reason of this wrong outcome was that "*range(2, int(8**.5))*" did not include 2; actually, it included no integrals. Therefore, calling smallest_factor(8) would not go through the for loop, but returned n (i.e., 8) directly.

To correct the code, **we should change "*for i in range(2, int(8**.5))*" to "*for i in range(2, int(8**.5)+1)*" and let all other codes remain unchanged.**

After correcting, testing the code again got the following outcome:

```
(base) FulindeMacBook-Pro:problem1 fulinguo$ 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/fulinguo/Desktop/problem1, inifile:
plugins: remotedata-0.3.1, openfiles-0.3.0, cov-2.6.0
collected 1 item

test_1.py .                                                              [100%]

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

This time, the program passed all test cases.

Problem 2:

Using py.test –cov. The coverage information of problem 1 is showed as follows:

```
(base) FulindeMacBook-Pro:problem1 fulinguo$ 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/fulinguo/Desktop/problem1, inifile:
plugins: remotedata-0.3.1, openfiles-0.3.0, cov-2.6.0
collected 1 item

test_1.py .                                                              [100%]

---------- coverage: platform darwin, python 3.6.6-final-0 -----------
Name                 Stmts   Miss  Cover
----------------------------------------
smallest_factor.py       7      1    86%
test_1.py                6      0   100%
----------------------------------------
TOTAL                   13      1    92%


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

As the result shows, the test covered all lines in test_1.py, but missed 1 line in smallest_factor.py.

Using the cov-report tool, I can find which lines were not covered:

```
1  def smallest_factor(n):
2          if n==1:
3                  return 1
4          for i in range(2,int(n**.5)+1):
5                  if n%i==0:
6                          return i
7          return n
```

As the figure shows, the line "*return 1*" has not been tested. This is because I did not test the case when n equals 1. Therefore, I add the test case "*assert smallest_factor(1)==1"*:

```
# test_1.py
from smallest_factor import smallest_factor

def test_1():
        assert smallest_factor(2)==2
        assert smallest_factor(18)==2
        assert smallest_factor(17)==17
        assert smallest_factor(1)==1
        assert smallest_factor(8)==2
```

Running py.test –cov again. This time all lines of codes were covered.

```
(base) FulindeMacBook-Pro:problem1 fulinguo$ 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/fulinguo/Desktop/problem1, inifile:
plugins: remotedata-0.3.1, openfiles-0.3.0, cov-2.6.0
collected 1 item

test_1.py .                                                              [100%]

---------- coverage: platform darwin, python 3.6.6-final-0 -----------
Name                 Stmts   Miss  Cover
----------------------------------------
smallest_factor.py       7      0   100%
test_1.py                7      0   100%
----------------------------------------
TOTAL                   14      0   100%


========================== 1 passed in 0.04 seconds ===========================
```

Then, I wrote the unit test that can test all lines of codes in problem 2.
I first stored the code of problem 2 in a file named month_length.py
Then, I wrote test code in a file named test_2.py. Here is the code:

```
# test_2.py

from month_length import month_length
def test_2():
        assert month_length('April')==30
        assert month_length('January')==31
        assert month_length('February')==28
        assert month_length('February',leap_year=True)==29
        assert month_length(2018)==None
```

using py.test –cov command, I got the outcome as follows:

```
FulindeMacBook-Pro:problem2 fulinguo$ 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/fulinguo/Desktop/problem2, inifile:
plugins: remotedata-0.3.1, openfiles-0.3.0, cov-2.6.0
collected 1 item

test_2.py .                                                              [100%]

---------- coverage: platform darwin, python 3.6.6-final-0 -----------
Name              Stmts   Miss  Cover
-------------------------------------
month_length.py      10      0   100%
test_2.py             7      0   100%
-------------------------------------
TOTAL                17      0   100%


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

As the figure shows, all test cases were passed and the test covered all lines in the code of month_length.py

Problem 3:

I first stored the code of problem 3 in a file named operate.py.
Then, I wrote the test in a file named test_3.py. Here is the code:

```
# test_3.py
from operate import operate
import pytest
def test_3():
```

```
assert operate(3,5,'+')==8
assert operate(3,5,'-')==-2
assert operate(3,5,'*')==15
assert operate(3,5,'/')==0.6
with pytest.raises(TypeError) as typinfo:
        operate(3,5,3)
assert typinfo.value.args[0]=='open must be a string'
with pytest.raises(ZeroDivisionError) as zeroinfo:
        operate(3,0,'/')
assert zeroinfo.value.args[0]=='division by zero is undefined'
with pytest.raises(ValueError) as valinfo:
        operate(3,5,'&')
assert valinfo.value.args[0]=="open must be one of '+','/','-', or '*'"
```

Using the py.test –cov command, I got the test outcome as follows:

```
[FulindeMacBook-Pro:problem3 fulinguo$ 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/fulinguo/Desktop/problem3, inifile:
plugins: remotedata-0.3.1, openfiles-0.3.0, cov-2.6.0
collected 1 item

test_3.py .                                                              [100%]

---------- coverage: platform darwin, python 3.6.6-final-0 -----------
Name         Stmts   Miss  Cover
------------------------------
operate.py      14      0   100%
test_3.py       16      0   100%
------------------------------
TOTAL           30      0   100%


========================== 1 passed in 0.04 seconds ===========================
```

As the figure shows, all test cases were passed and the test covered the entire program.

I could also use the cov-report tool to confirm that all lines of operate.py and test_3.py has been covered:

Coverage for **test_3.py** : 100%

16 statements | 16 run | 0 missing | 0 excluded

```
1  from operate import operate
2  import pytest
3  def test_3():
4          assert operate(3,5,'+')==8
5          assert operate(3,5,'-')==-2
6          assert operate(3,5,'*')==15
7          assert operate(3,5,'/')==0.6
8          with pytest.raises(TypeError) as typinfo:
9                  operate(3,5,3)
10         assert typinfo.value.args[0]=='open must be a string'
11         with pytest.raises(ZeroDivisionError) as zeroinfo:
12                 operate(3,0,'/')
13         assert zeroinfo.value.args[0]=='division by zero is undefined'
14         with pytest.raises(ValueError) as valinfo:
15                 operate(3,5,'&')
16         assert valinfo.value.args[0]=="open must be one of '+','/','-', or '*'"
```

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

14 statements | 14 run | 0 missing | 0 excluded

```
1  def operate(a,b,oper):
2          if type(oper) is not str:
3                  raise TypeError('open must be a string')
4          elif oper=='+':
5                  return a+b
6          elif oper =='-':
7                  return a-b
8          elif oper=='*':
9                  return a*b
10         elif oper =='/':
11                 if b==0:
12                         raise ZeroDivisionError('division by zero is undefined')
13                 return a/b
14         raise ValueError("open must be one of '+','/','-', or '*'")
```

The HTML file shows that there were no missing lines.

2. (b) I wrote the python module get_r.py and put it into a folder named get_r. Here is the code:

```
# get_r.py
import numpy as np
def get_r(K,L,alpha,Z,delta):
        K=np.array(K)
        L=np.array(L)
        r=alpha*Z*(L/K)**(1-alpha)-delta
        return r
```

(c) I put the file test_r.py into the same folder as the get_r.py file and use the pytest-cov tool to test the codes in get_r.py. Here is the outcome:

```
FulindeMacBook-Pro:get_r fulinguo$ 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/fulinguo/Desktop/get_r, 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         6      0   100%
test_r.py       29      0   100%
------------------------------
TOTAL           35      0   100%


========================== 244 passed in 0.50 seconds ==========================
```

As the result shows, all test cases in test_r.py were passed and the test covered the entire program in the get_r module.

3. The rational choice theory is an example of “rationalization” the author discussed in the paper. The theory had a lot of criticisms as it was not based on valid and correct assumptions regarding agent’s knowledge, preferences, etc. (Watts, 2014). In addition, the rational choice theory did not have consistence predictions with empirical observations. Although the theory changed a lot over time, like allowing limited knowledge, giving up the utility maximization method, etc. (Watts, 2014), the author argued that the explanations based on rational choice theory had changed from causal explanation to “empathetic explanation”. That is, the supports of the theory now focused less on deduction and prediction than on making the explanation understandable (Watts, 2014). Although Farmer (1992) argued that the rational choice theory was constructed for the sake of better explaining some social outcomes, which is similar to the energy conservation in physics, the rational choice theory did not be tested regarding their predictability (Farmer, 1992; Watts, 2014).

   The main drawback that Watts proposed about utilizing commonsense theories was that social scientists combined causal explanation and

"empathetic explanation" when they constructed theories and this combination was not effective because a valid explanation for some outcomes does not indicate that it is effective for more general causal mechanisms (Watts, 2014). In addition, Watts said that the reasons based on commonsense theories in advance might not result in accurate predictions, while the reasons after the observations might produce more accurate predictions (Watts, 2014). Therefore, using commonsense theories has a negative effect on scientific validity (Watts, 2014).

Facing the issues regarding causal explanations of rational choice theory, Watts proposed that social scientists could provide more scientific explanation by giving up satisfying explanation. That is, researchers can honestly concede that answering some questions is impossible and provide explanation in a more scientific method (Watts, 2014). The authors also proposed that for social scientists, causal explanations could be obtained based on experiments (like field experiments, natural experiments, quasi-experiments, laboratory experiments, etc.), causal inference, and out-of-sample testing (Watts,2014).

We could construct econometrical models based on theoretical models (with some simplification and assumptions) and then utilize these econometrical models to conduct hypothesis tests to detect causal effect and make prediction as well (conditional on some explanatory variables). With theoretical models, we could have some conclusions or conjectures from these theoretical models. This would provide us some selections or choices of what causal inference we could conduct and what causal effects we might be able to detect if it has. (This could be regarded as theory-driven paradigm, which is having a theory first and then making causal inference based on the theory). Without those theoretical models, it would be difficult for us to find where to start our causal inference because the real world is too complex. Also, we might not be able to decide what variables, what factors should be included in the prediction and causal inference process without theoretical models. For example, in order to predict people's consumption behavior, the consumer theory might provide a possible reference for us to construct statistical models (That is, predict agents' consumption behavior based on their income and the price of goods). Without the consumer theory, there are too many variables in the world, we do not know what variables should be included in the prediction. In addition, we could also test the validity of

consumer theory using existing data (the existing data of individuals' consumption behavior, income and the price of goods they bought).

References:

Farmer, Mary K. 1992. "On the Need to Make a Better Job of Justifying Rational Choice Theory." Rationality and Society 4(4): 411-20

Humpherys, Jeffrey and Tyler J. Jarvis, "Labs for Foundations of Applied Mathematics: Python Essentials," creative commons, open access 2018.

Watts, Duncan J., "Common Sense and Sociological Explanations," American Journal of Sociology, September 2014, 120 (2), 313-351.