# lab1

July 1, 2019

### 0.0.1 About myself

My name is Sumeng Wang. I am currently an MFE student at Anderson. I will be graduating in December. I came from a Computer Science and Math background. My email is sumeng.wang.2019@anderson.ucla.edu. Feel free to email me any questions you have

### 0.0.2 Indroduction to Python

(optional) install Python and VS Code:
Windows: https://phoenixnap.com/kb/how-to-install-python-3-windows
MacOS: should already have python 2.7 built in, which is fine using it.

Python IDE: I personally like VS Code, because when you are doing a big project involving multiple files, VS Code really shows its advantage. A lot of people also use PyCharm, which is also fine. So if you want to use VS code, follow the following steps:
1. download VS Code
2. Go to extensions, download 'Python' extension
3. (optional) install IPython. type "pip install ipython" or "pip3 install ipython" in command line

### 0.0.3 number operations

[19]:
```
a = 321
b = 123
print(a + b)
print(a - b)
print(a * b)
print(a / b)
print(a // b) # floor of the quotient
print(a % b) # modulo
print(a ** b) # exponential
```

```
444
198
39483
2.6097560975609757
2
75
19958090417085894458868346460869407506294310708280902760549834784156160518107427
44266659866237649724398077867647391301933916888873056932956981320084155378897537
```

```
204157938779020746657657362300306222228624983858181182445476681413886437508808968374705345567300143142129513335501229498061195830302261217147108745610
```

#### 0.0.4 variable types

[20]:
```
a = 100
b = 12.345
c = 1 + 5j
d = 'hello, world'
e = True
print(type(a))
print(type(b))
print(type(c))
print(type(d))
print(type(e))
```

```
<class 'int'>
<class 'float'>
<class 'complex'>
<class 'str'>
<class 'bool'>
```

#### 0.0.5 Python Collections (Arrays)

There are four collection data types in the Python programming language:
List is a collection which is ordered and changeable. Allows duplicate members.
Tuple is a collection which is ordered and unchangeable. Allows duplicate members.
Set is a collection which is unordered and unindexed. No duplicate members.
Dictionary is a collection which is unordered, changeable and indexed. No duplicate members.

**list**

[6]:
```
thislist = ["apple", "banana", "cherry"]
print(thislist)
```

```
['apple', 'banana', 'cherry']
```

[7]:
```
thislist = ["apple", "banana", "cherry"]
print(thislist[1])
```

```
banana
```

[8]:
```
thislist = ["apple", "banana", "cherry"]
thislist[1] = "ball"
print(thislist)
```

```
['apple', 'ball', 'cherry']
```

If you want to check if the list contains a specific element, use "in"

[19]:
```
thislist = ["apple", "banana", "cherry"]
"apple" in thislist
```

[19]:
```
True
```

[11]:
```
len(thislist)
```

[11]:
```
3
```

[12]:
```
# insert item at the end of the list
thislist.append("orange")
print(thislist)
```

```
['apple', 'banana', 'cherry', 'orange']
```

[13]:
```
# insert item at specific location
thislist = ["apple", "banana", "cherry"]
thislist.insert(1, "orange")
print(thislist)
```

```
['apple', 'orange', 'banana', 'cherry']
```

[15]:
```
thislist = ["apple", "banana", "cherry"]
del thislist[0]
print(thislist)
del thislist
print(thislist)
```

```
['banana', 'cherry']


      ␣
↪---------------------------------------------------------------------------

      NameError                                 Traceback (most recent call␣
↪last)

      <ipython-input-15-900bda7c3972> in <module>
        3 print(thislist)
        4 del thislist
  ----> 5 print(thislist)


      NameError: name 'thislist' is not defined
```

**Dictionary**

[16]:
```
thisdict =        {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

[20]:
```
# Get the value of a specific key
thisdict["model"]
```

[20]:
```
'Mustang'
```

#### 0.0.6 if statement

Python supports the usual logical conditions from mathematics:
Equals: a == b
Not Equals: a != b
Less than: a < b
Less than or equal to: a <= b
Greater than: a > b
Greater than or equal to: a >= b

[1]:
```
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

```
b is greater than a
```

Everything under if must be indented.

**elif and else**

[2]:
```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

```
a is greater than b
```

**and, or**

[ ]:
```
if a > b and c > a:
  print("Both conditions are True")
if a > b or a > c:
  print("At least one of the conditions is True")
```

#### 0.0.7 for loop

[3]:
```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

```
apple
banana
cherry
```

[4]:
```
fruits = ["apple", "banana", "cherry"]
for idx, val in enumerate(fruits):
  print(str(idx) + ': ' + val)
```

```
0: apple
1: banana
2: cherry
```

[5]:
```
for x in "banana":
  print(x)
```

```
b
a
n
a
n
a
```

[21]:
```
thisdict =        {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x, y in thisdict.items():
  print(x, y)
```

```
brand Ford
model Mustang
year 1964
```

[22]:
```
for x in range(6):
  print(x)
```

```
0
1
2
3
4
5
```

[23]:
```
for x in range(2, 6):
  print(x)
```

```
2
3
4
5
```

[24]:
```
for x in range(2, 30, 3):
  print(x)
```

```
2
5
8
11
14
17
20
23
26
29
```

### 0.0.8 Function

[25]:
```
def my_function():
  print("Hello from a function")

my_function()
```

```
Hello from a function
```

[26]:
```
# function with parameter
def my_function(x):
  print("I like " + x)

my_function("apple")
```

```
my_function("banana")
```

```
I like apple
I like banana
```

#### 0.0.9 Lambda function

A lambda function is a small anonymous function.
A lambda function can take any number of arguments, but can only have one expression.

[27]:
```
x = lambda a : a + 10
print(x(5))
```

```
15
```

[28]:
```
x = lambda a, b : a * b
print(x(5, 6))
```

```
30
```

# 1 Pandas foundation

[ ]:
```
# install required packages
pip install pandas
pip install numpy
pip install matplotlib
```

[1]:
```
import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt
```

[ ]:
```
# read csv files
df = pd.read_csv(filename)
```

**Create a dataframe from scratch**

[48]:
```
# first create a dictionary
d = {'Name': ['Andy', 'Bill','Catherine','David','Emma'],
     'Score': [87, 67, 93, 95, 50],
    'Gender': ['M','M','F','M','F']}
df = pd.DataFrame(d)
df
```

[48]:

| | Name | Score | Gender |
|---|---|---|---|
| 0 | Andy | 87 | M |
| 1 | Bill | 67 | M |
| 2 | Catherine | 93 | F |
| 3 | David | 95 | M |

```
4       Emma     50      F
```

[21]:
```
df2 = pd.DataFrame([['Andy', 87, 'M'],
                    ['Bill', 67, 'M'],
                    ['Catherine', 93, 'F'],
                    ['David', 95, 'M'],
                    ['Emma', 50, 'F']],
                   columns=['Name', 'Score', 'Gender'])
df2
```

[21]:
```
         Name  Score Gender
0       Andy     87      M
1       Bill     67      M
2  Catherine     93      F
3      David     95      M
4       Emma     50      F
```

**Subsetting**

[6]:
```
df['Name']
```

[6]:
```
0         Andy
1         Bill
2    Catherine
3        David
4         Emma
Name: Name, dtype: object
```

[8]:
```
df.loc[:,'Name']
```

[8]:
```
0         Andy
1         Bill
2    Catherine
3        David
4         Emma
Name: Name, dtype: object
```

[9]:
```
df.iloc[:, 0]
```

[9]:
```
0         Andy
1         Bill
2    Catherine
3        David
4         Emma
Name: Name, dtype: object
```

[10]:
```
df.loc[1:2, :]
```

[10]:
```
         Name  Score Gender
1       Bill     67      M
2  Catherine     93      F
```

[11]:
```
df[['Name', 'Gender']]
```

[11]:
```
         Name Gender
0        Andy      M
1        Bill      M
2  Catherine      F
3       David      M
4        Emma      F
```

**Subsetting on condition**

[12]:
```
df[df['Gender'] == 'M']
```

[12]:
```
    Name  Score Gender
0   Andy     87      M
1   Bill     67      M
3  David     95      M
```

[13]:
```
df[df['Score'] > 70]
```

[13]:
```
         Name  Score Gender
0        Andy     87      M
2  Catherine     93      F
3       David     95      M
```

[18]:
```
df[df['Gender'] != 'M']
```

[18]:
```
         Name  Score Gender
2  Catherine     93      F
4        Emma     50      F
```

[16]:
```
df[(df['Score'] > 70) & (df['Gender'] == 'M')]
```

[16]:
```
    Name  Score Gender
0   Andy     87      M
3  David     95      M
```

[17]:
```
df[(df['Score'] > 70) | (df['Gender'] == 'M')]
```

[17]:
```
         Name  Score Gender
0        Andy     87      M
1        Bill     67      M
2  Catherine     93      F
3       David     95      M
```

**Some functions**

[28]:
```
print(df['Score'].max())
print(df['Score'].min())
```

```
95
50
```

[25]:
```
df['Score'].sum()
```

[25]:
```
392
```

[29]:
```
df['Score'].mean()
```

[29]:
```
78.4
```

[30]:
```
df['Score'].std()
```

[30]:
```
19.3597520645281
```

[31]:
```
df['Score'].skew()
```

[31]:
```
-0.9225286167901775
```

[32]:
```
df['Score'].kurtosis()
```

[32]:
```
-0.9322923132821286
```

**Insert into dataframe**

[36]:
```
# insert row
df3 = pd.DataFrame([['Fiona', 78, 'F'],
                    ['George', 84, 'M']],
                   columns=['Name', 'Score', 'Gender'])
df.append(df3)
```

[36]:
```
         Name  Score Gender
0        Andy     87      M
1        Bill     67      M
2  Catherine     93      F
3      David     95      M
4        Emma     50      F
0      Fiona     78      F
1     George     84      M
```

[38]:
```
df.append(df3, ignore_index = True)
```

[38]:
```
         Name  Score Gender
0        Andy     87      M
1        Bill     67      M
2  Catherine     93      F
3      David     95      M
4        Emma     50      F
5      Fiona     78      F
6     George     84      M
```

[49]:
```
df.loc[5] = ['Fiona', 78, 'F']
df.loc[6] = ['George', 84, 'M']
df
```

[49]:
```
         Name  Score Gender
0        Andy     87      M
```

```
1       Bill     67      M
2  Catherine     93      F
3      David     95      M
4       Emma     50      F
5      Fiona     78      F
6     George     84      M
```

[77]:
```
# create new column
df['new Score'] = df['Score'] + 1
df
```

[77]:
```
         Name  Score Gender  new Score       mean     gpmean
0       Andy     87      M         88  79.142857  83.250000
1       Bill     67      M         68  79.142857  83.250000
2  Catherine     93      F         94  79.142857  73.666667
3      David     95      M         96  79.142857  83.250000
4       Emma     50      F         51  79.142857  73.666667
5      Fiona     78      F         79  79.142857  73.666667
6     George     84      M         85  79.142857  83.250000
```

[52]:
```
df['mean'] = df['Score'].mean()
df
```

[52]:
```
         Name  Score Gender  new Score       mean
0       Andy     87      M         88  79.142857
1       Bill     67      M         68  79.142857
2  Catherine     93      F         94  79.142857
3      David     95      M         96  79.142857
4       Emma     50      F         51  79.142857
5      Fiona     78      F         79  79.142857
6     George     84      M         85  79.142857
```

### Grouping

[58]:
```
df.groupby('Gender')[['Score']].mean()
```

[58]:
```
            Score
Gender
F       73.666667
M       83.250000
```

[65]:
```
df.groupby('Gender')[['Score']].mean().reset_index()
```

[65]:
```
  Gender      Score
0      F  73.666667
1      M  83.250000
```

[156]:
```
# more complex aggregation, use lambda function
myfunc = lambda x: np.mean(x * 2 + 1)
f = {'Score' : myfunc}
df.groupby('Gender').agg(f).reset_index()
```

[156]:
```
  Gender       Score
0      F  148.333333
1      M  167.500000
```

[64]:
```
df['gpmean'] = df.groupby('Gender')[['Score']].transform(np.mean)
df
```

[64]:
```
        Name  Score Gender  new Score       mean     gpmean
0       Andy     87      M         88  79.142857  83.250000
1       Bill     67      M         68  79.142857  83.250000
2  Catherine     93      F         94  79.142857  73.666667
3      David     95      M         96  79.142857  83.250000
4       Emma     50      F         51  79.142857  73.666667
5      Fiona     78      F         79  79.142857  73.666667
6     George     84      M         85  79.142857  83.250000
```

**Plotting**

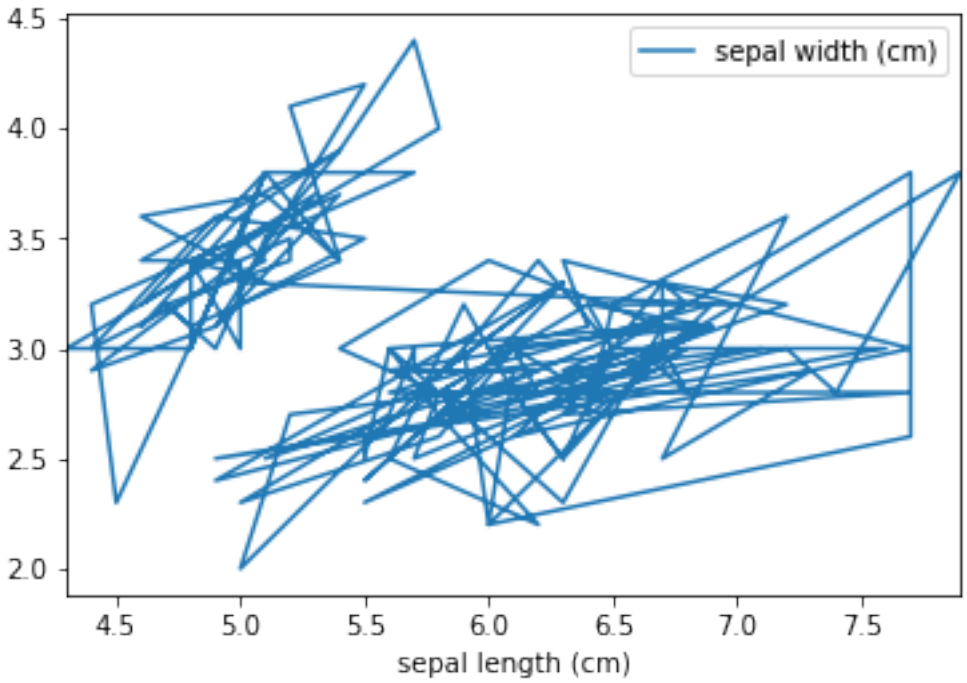
[112]:
```
from sklearn import datasets
iris = datasets.load_iris()
X = pd.DataFrame(iris.data, columns = iris.feature_names)
X['type'] = iris.target
X.head()
```

[112]:
```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

   type
0     0
1     0
2     0
3     0
4     0
```

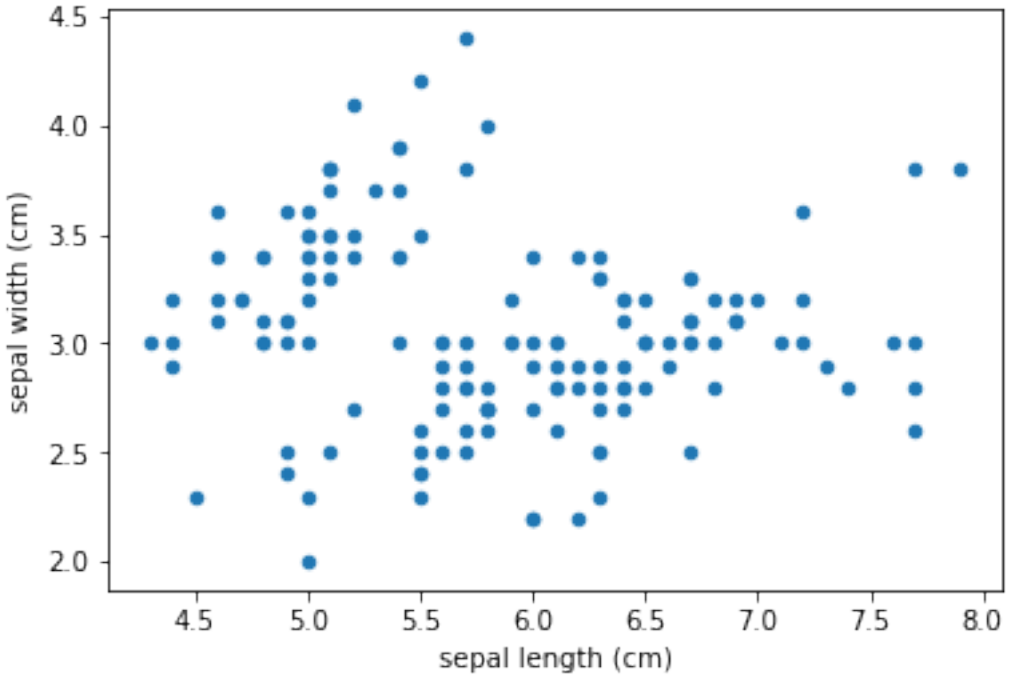
[108]:
```
X.plot(x = 'sepal length (cm)', y = 'sepal width (cm)')
```

[108]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x21ff4ddb9b0>
```



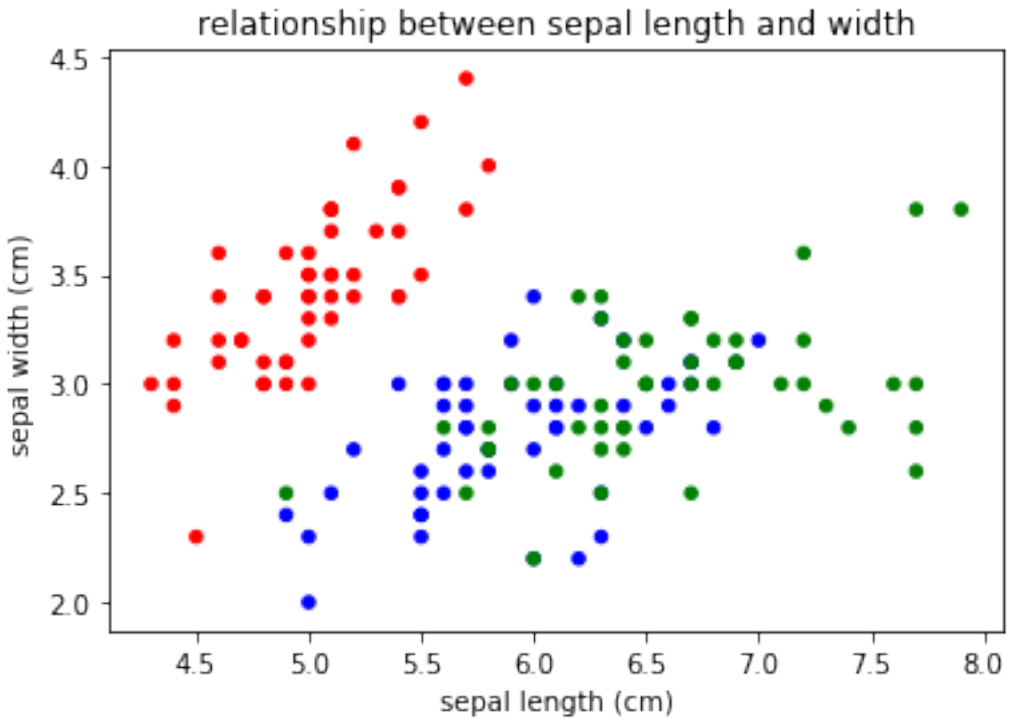

[109]:
```
X.plot(x = 'sepal length (cm)', y = 'sepal width (cm)', kind = 'scatter')
```

[109]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x21ff4e40c88>
```



[125]:
```
colors = {0:'red', 1:'blue', 2:'green'}
X.plot(x = 'sepal length (cm)', y = 'sepal width (cm)', kind = 'scatter',
       title = 'relationship between sepal length and width', c = X['type'].
 ↪apply(lambda l: colors[l]))
```

[125]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x21ff50bd390>
```




### 1.0.1 Lab1

Write programs to investigate a first trading strategy. Start with a "BV/MV" strategy. Assume six months lagged availabilty. Calculate only the simpler arithmetic performance. Plot the average monthly performance based on 100 / 500 / 1000 largest stocks only. Consider time-series and subsets: last 10 years, last 30 years, last 60 years, and compare to market rate of return.

[126]:
```
# first import all the required modules
import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt
```

[132]:
```
# read data
ccm = pd.read_csv('D://UCLA//431QAM//hw4//bm.csv')
ccm.head()
```

[132]:
```
   PERMNO       jdate     lme    retadj  bm  year  month
0   10000  1986-01-31     NaN  0.000000 NaN  1986      1
1   10000  1986-02-28  16.100 -0.257143 NaN  1986      2
2   10000  1986-03-31  11.960  0.365385 NaN  1986      3
3   10000  1986-04-30  16.330 -0.098592 NaN  1986      4
4   10000  1986-05-31  15.172 -0.222656 NaN  1986      5
```

[138]:
```
# find 100 companies with largest BV/MV value each year
subset1 = ccm.groupby(by = ['jdate'])[['PERMNO','lme','retadj',
                                       'bm','year','month']].apply(lambda df:␣
 ↪df.nlargest(100, 'bm')).reset_index()
subset1.head()
```

[138]:
```
        jdate  level_1  PERMNO        lme    retadj        bm  year  month
0  1963-07-31  1037107   34948  22.113000  0.012346       inf  1963      7
1  1963-07-31   836602   27887  68.385625 -0.054187  8.974675  1963      7
2  1963-07-31   787183   26569  70.325000 -0.034483  8.759663  1963      7
3  1963-07-31   510437   18112  18.914500  0.009569  7.464767  1963      7
4  1963-07-31   962815   32328   1.348313  0.098039  6.828445  1963      7
```

[151]:
```
# subsetting from end of 2008 to end of 2018
subset1 = subset1[(subset1['year']> (2008))&(subset1['year']<= 2018)]
subset1.head()
```

[151]:
```
array([ 0.056992,  0.752577, -0.099647,  0.647059,  0.107605])
```

[158]:
```
weighted_avg = lambda x: np.average(x, weights=subset1.loc[x.index, 'lme'])
func = {'retadj' :weighted_avg}
vwretd = subset1.groupby(by = 'jdate').agg(func).reset_index()
vwretd['cumret'] = (vwretd['retadj'] + 1).cumprod() - 1
vwretd.head()
```

[158]:
```
        jdate    retadj    cumret
0  2009-01-31 -0.187244 -0.187244
1  2009-02-28 -0.224939 -0.370065
2  2009-03-31  0.176934 -0.258608
3  2009-04-30  0.171723 -0.131294
4  2009-05-31  0.049646 -0.088166
```

[161]:
```
ewretd = subset1.groupby(by = 'jdate')[['jdate','retadj']].mean().reset_index()
ewretd['cumret'] = (ewretd['retadj'] + 1).cumprod() - 1
ewretd.head()
```

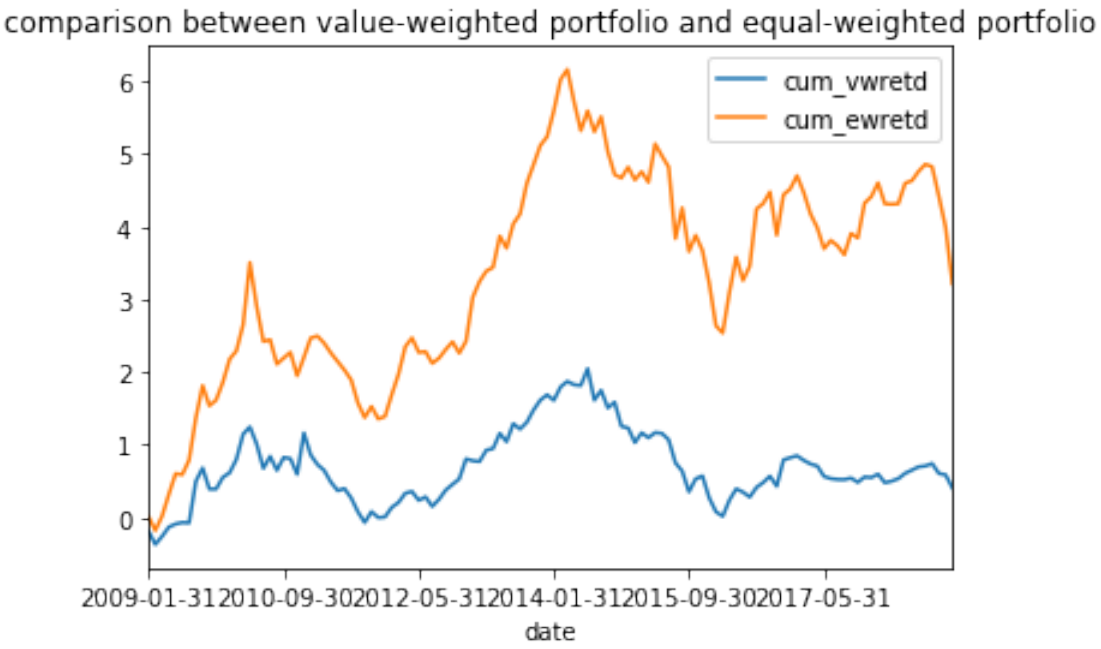
[161]:
```
        jdate    retadj    cumret
0  2009-01-31  0.012437  0.012437
1  2009-02-28 -0.187790 -0.177688
2  2009-03-31  0.240682  0.020227
3  2009-04-30  0.293348  0.319509
4  2009-05-31  0.215318  0.603622
```

[178]:
```
merged = pd.merge(vwretd, ewretd, on = 'jdate')
merged = merged.rename(index = str, columns = {'jdate': 'date',
                                               'cumret_x' : 'cum_vwretd',
                                               'cumret_y' : 'cum_ewretd',
                                               'retadj_x' : 'vwretd',
                                               'retadj_y' : 'ewretd'})
merged.head()
```

[178]:

| | date | vwretd | cum_vwretd | ewretd | cum_ewretd |
|---|---|---|---|---|---|
| 0 | 2009-01-31 | -0.187244 | -0.187244 | 0.012437 | 0.012437 |
| 1 | 2009-02-28 | -0.224939 | -0.370065 | -0.187790 | -0.177688 |
| 2 | 2009-03-31 | 0.176934 | -0.258608 | 0.240682 | 0.020227 |
| 3 | 2009-04-30 | 0.171723 | -0.131294 | 0.293348 | 0.319509 |
| 4 | 2009-05-31 | 0.049646 | -0.088166 | 0.215318 | 0.603622 |

[179]:
```
merged.plot(x = 'date', y = ['cum_vwretd', 'cum_ewretd'],
            title = 'comparison between value-weighted portfolio and␣
 ↪equal-weighted portfolio')
plt.show()
```




[180]:
```
annualized_mean = merged['vwretd'].mean()*12
annualized_std = merged['vwretd'].std()*math.sqrt(12)
SR = annualized_mean/annualized_std
print('annualized mean is: ' + str(annualized_mean))
print('annualized std is: ' + str(annualized_std))
print('Sharpe Ratio is: ' + str(SR))
```

```
annualized mean is: 0.10570953605564334
annualized std is: 0.3942919255553273
Sharpe Ratio is: 0.26809967236016885
```

[181]:
```
annualized_mean = merged['ewretd'].mean()*12
annualized_std = merged['ewretd'].std()*math.sqrt(12)
SR = annualized_mean/annualized_std
print('annualized mean is: ' + str(annualized_mean))
print('annualized std is: ' + str(annualized_std))
print('Sharpe Ratio is: ' + str(SR))
```

```
annualized mean is: 0.19120506259307693
annualized std is: 0.3125954822318652
Sharpe Ratio is: 0.6116693089353483
```

[ ]:
```

```