stock prediction

January 23, 2019

```
In [4]: import pandas as pd
        import pandas_datareader
```

```
In [5]: from pandas_datareader import data
        import matplotlib.pyplot as plt
        %matplotlib inline
```

```
In [6]: pg = data.DataReader('MSFT', data_source='yahoo',start='1995-1-1')
```

```
In [7]: pg.head()
```

```
Out[7]:                 High       Low      Open     Close      Volume  Adj Close
        Date
        1995-01-03  3.843750  3.757812  3.843750  3.761719  39545600.0   2.729909
        1995-01-04  3.796875  3.718750  3.765625  3.789062  51611200.0   2.749754
        1995-01-05  3.812500  3.710938  3.804688  3.726562  39824000.0   2.704397
        1995-01-06  3.828125  3.734375  3.742188  3.789062  46681600.0   2.749754
        1995-01-09  3.812500  3.734375  3.804688  3.765625  46000000.0   2.732745
```

Simple return

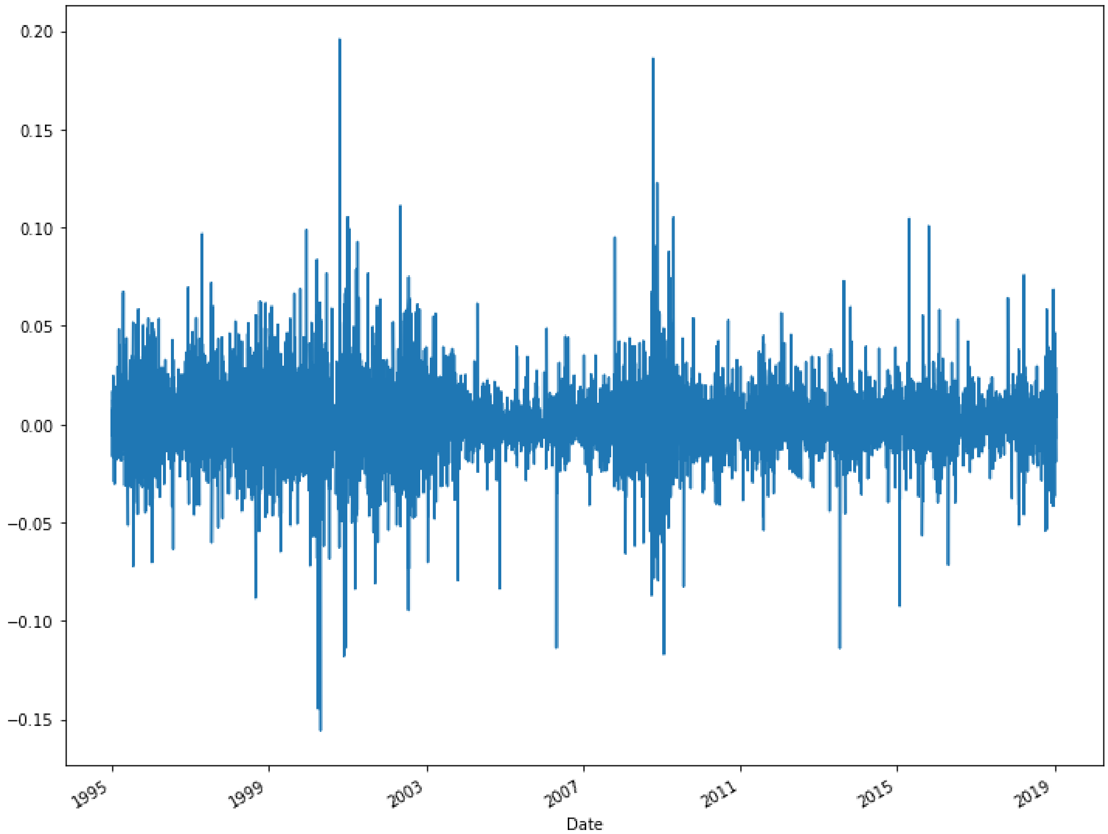
```
In [8]: pg['simple_return'] = (pg['Adj Close']/pg['Adj Close'].shift(1))-1
```

```
In [9]: pg.head()
```

```
Out[9]:                 High       Low      Open     Close      Volume  Adj Close  \
        Date
        1995-01-03  3.843750  3.757812  3.843750  3.761719  39545600.0   2.729909
        1995-01-04  3.796875  3.718750  3.765625  3.789062  51611200.0   2.749754
        1995-01-05  3.812500  3.710938  3.804688  3.726562  39824000.0   2.704397
        1995-01-06  3.828125  3.734375  3.742188  3.789062  46681600.0   2.749754
        1995-01-09  3.812500  3.734375  3.804688  3.765625  46000000.0   2.732745

                    simple_return
        Date
        1995-01-03            NaN
        1995-01-04       0.007269
        1995-01-05      -0.016495
        1995-01-06       0.016771
        1995-01-09      -0.006186
```

```
In [10]: pg['simple_return'].plot(figsize=(12,10))
```

```
Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0xae36438>
```




```
In [11]: average_return = pg['simple_return'].mean()
```

```
In [104]: average_return
```

```
Out[104]: 0.0008017855195413143
```

```
In [178]: average_return = average_return * 250
          average_return
```

```
Out[178]: 0.20125777227339678
```

```
In [179]: round(average_return,4)*100
```

```
Out[179]: 20.13
```

Logreturns

```
In [106]: import numpy as np
          pg['logreturn'] = np.log(pg['Adj Close']/pg['Adj Close'].shift(1))

In [107]: pg.head()

Out[107]:                 High       Low      Open     Close      Volume  Adj Close  \
          Date
          1995-01-03  3.843750  3.757812  3.843750  3.761719  39545600.0   2.729909
          1995-01-04  3.796875  3.718750  3.765625  3.789062  51611200.0   2.749754
          1995-01-05  3.812500  3.710938  3.804688  3.726562  39824000.0   2.704397
          1995-01-06  3.828125  3.734375  3.742188  3.789062  46681600.0   2.749754
          1995-01-09  3.812500  3.734375  3.804688  3.765625  46000000.0   2.732745

                      simple_return  logreturn
          Date
          1995-01-03            NaN        NaN
          1995-01-04       0.007269   0.007243
          1995-01-05      -0.016495  -0.016632
          1995-01-06       0.016771   0.016632
          1995-01-09      -0.006186  -0.006205

In [182]: average_return_log = pg['logreturn'].mean() * 250 * 100
          average_return_log

Out[182]: 15.176635612959672
```

0.0.1 return of portfolio of security

```
In [183]: player = ['PG','MSFT', 'F', 'GE']
          mydata = pd.DataFrame()
          for individual in player:
              mydata[individual] = data.DataReader(individual, data_source='yahoo',start='1995-

In [184]: mydata.info()

<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 6056 entries, 1995-01-03 to 2019-01-22
Data columns (total 4 columns):
PG      6056 non-null float64
MSFT    6056 non-null float64
F       6056 non-null float64
GE      6056 non-null float64
dtypes: float64(4)
memory usage: 236.6 KB


In [185]: mydata.head()
```

Out[185]:

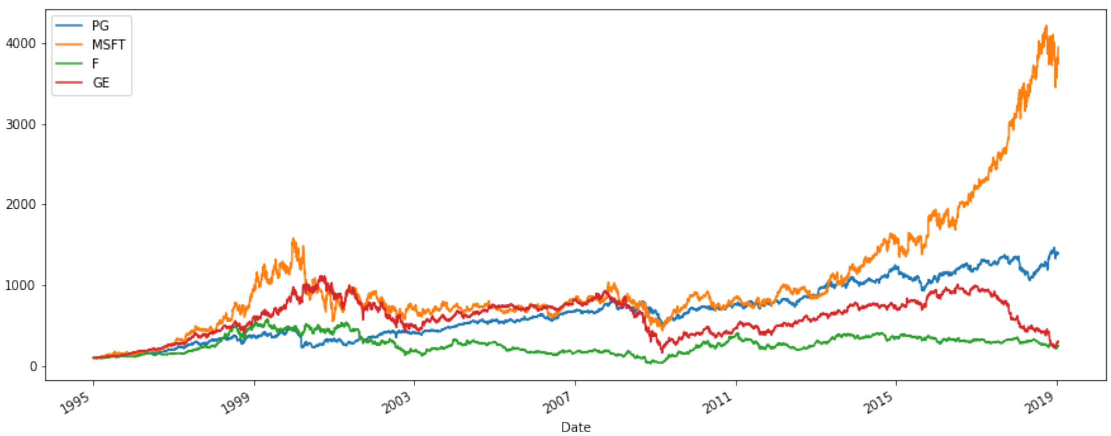
Date	PG	MSFT	F	GE
1995-01-03	6.528558	2.729909	3.531971	2.975797
1995-01-04	6.476228	2.749754	3.626998	2.975797
1995-01-05	6.384644	2.704397	3.595320	2.983092
1995-01-06	6.397724	2.749754	3.595320	2.968507
1995-01-09	6.371559	2.732745	3.658676	2.939330

In [30]:
```
mydata.iloc[0]
```

Out[30]:
```
PG      6.528558
MSFT    2.729909
F       3.531971
GE      2.975797
Name: 1995-01-03 00:00:00, dtype: float64
```

In [31]:
```
(mydata/mydata.iloc[0]*100).plot(figsize=(15,6))
```

Out[31]: <matplotlib.axes._subplots.AxesSubplot at 0xf193860>



In [32]:
```
weight = np.array([0.25,0.25,0.25,0.25])
```

In [33]:
```
returns = (mydata/mydata.shift(1)) - 1
```

In [34]:
```
returns.head()
```

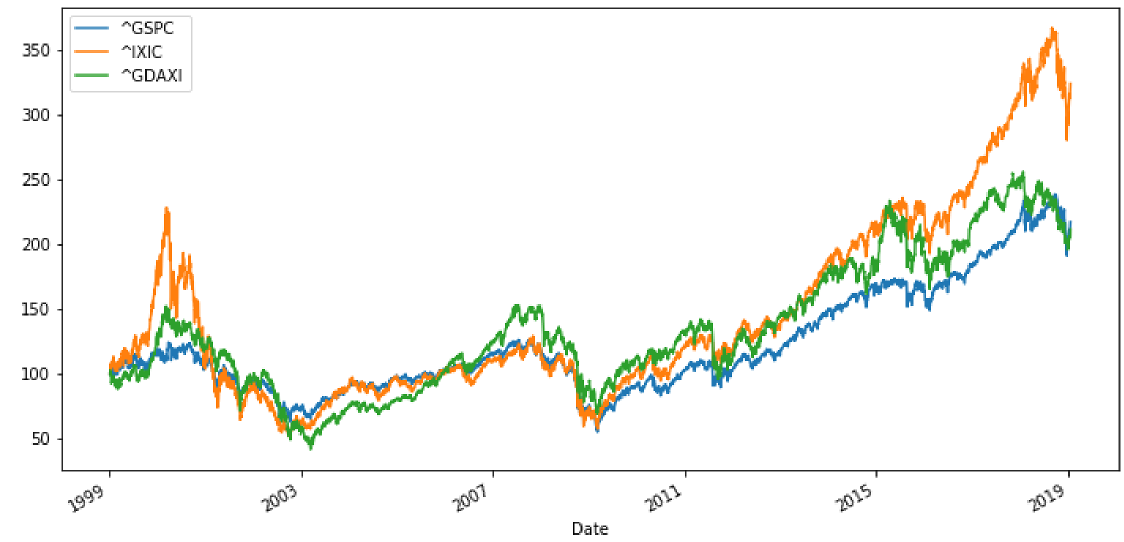
Out[34]:

Date	PG	MSFT	F	GE
1995-01-03	NaN	NaN	NaN	NaN
1995-01-04	-0.008015	0.007269	0.026905	0.000000
1995-01-05	-0.014142	-0.016495	-0.008734	0.002451
1995-01-06	0.002049	0.016771	0.000000	-0.004889
1995-01-09	-0.004090	-0.006186	0.017622	-0.009829

```
In [35]: averagereturns = returns.mean()*250
         averagereturns

Out[35]: PG      0.134034
         MSFT    0.201258
         F       0.115024
         GE      0.090257
         dtype: float64

In [36]: type(averagereturns)

Out[36]: pandas.core.series.Series

In [37]: np.dot(averagereturns,weight)

Out[37]: 0.13514310628800663

In [38]: # Assigning different weight
         weight2 = np.array([0.40,0.40,0.10,0.10])

In [39]: np.dot(averagereturns,weight2)*100

Out[39]: 15.46447061443043
```

0.0.2 Calculating return of index

```
In [12]: stockindex = ['^GSPC', '^IXIC','^GDAXI']

In [13]: indexdata = pd.DataFrame()
         for i in stockindex:
             indexdata[i] = data.DataReader(i, data_source='yahoo',start='1999-01-04')['Adj Cl

In [14]: indexdata.head()

Out[14]:                   ^GSPC         ^IXIC        ^GDAXI
         Date
         1999-01-04  1228.099976  2208.050049  5290.359863
         1999-01-05  1244.780029  2251.270020  5263.410156
         1999-01-06  1272.339966  2320.860107  5442.899902
         1999-01-07  1269.729980  2326.090088  5345.709961
         1999-01-08  1275.089966  2344.409912  5370.509766

In [43]: (indexdata/indexdata.iloc[0]*100).plot(figsize=(12,6))

Out[43]: <matplotlib.axes._subplots.AxesSubplot at 0xefef630>
```




```
In [44]: indexdatareturn  = (indexdata/indexdata.shift(1)-1)
         indexdatareturn.tail()

Out[44]:                ^GSPC     ^IXIC    ^GDAXI
         Date
         2019-01-14 -0.005258 -0.009404 -0.002898
         2019-01-15  0.010722  0.017074  0.003305
         2019-01-16  0.002222  0.001546  0.003622
         2019-01-17  0.007591  0.007075 -0.001155
         2019-01-18  0.013183  0.010272  0.026278

In [45]: anualdatareturn = indexdatareturn.mean()*250

In [46]: anualdatareturn.head()

Out[46]: ^GSPC     0.056628
         ^IXIC     0.090044
         ^GDAXI    0.053166
         dtype: float64

In [17]: indexdata2 = ['MSFT','^GSPC','^DJI']
         indexdat2 = pd.DataFrame()
         for i in indexdata2:
             indexdat2[i] = data.DataReader(i, data_source='yahoo',start='2007-01-01')['Adj Cl

In [18]: indexdat2.head()

Out[18]:                  MSFT        ^GSPC          ^DJI
         Date
         2007-01-03  22.574831  1416.599976  12474.519531
```

```
2007-01-04  22.537027  1418.339966  12480.690430
2007-01-05  22.408501  1409.709961  12398.009766
2007-01-08  22.627748  1412.839966  12423.490234
2007-01-09  22.650431  1412.109985  12416.599609
```

In [19]:
```
(indexdat2/indexdat2.iloc[0]*100).plot(figsize=(12,6))
```

Out[19]:
```
<matplotlib.axes._subplots.AxesSubplot at 0xe7c4a20>
```

Calculating the risk of security

In [77]:
```
company = ['MSFT','AAPL']
secdata = pd.DataFrame()
for i in company:
    secdata[i] = data.DataReader(i, data_source='yahoo',start='2008-01-11')['Adj Close
```

In [78]:
```
secdata.head()
```

Out[78]:
```
                 MSFT       AAPL
Date
2008-01-10  26.306889  17.030483
2008-01-11  25.985041  16.520582
2008-01-14  26.352856  17.103188
2008-01-15  26.054005  16.171404
2008-01-16  25.463963  15.272138
```

In [80]:
```
import numpy as np
secreturn =  np.log(secdata/secdata.shift(1))
```

In [81]:
```
secreturn.head()
```

```
Out[81]:                 MSFT      AAPL
         Date
         2008-01-10       NaN       NaN
         2008-01-11 -0.012310 -0.030398
         2008-01-14  0.014056  0.034658
         2008-01-15 -0.011405 -0.056020
         2008-01-16 -0.022907 -0.057214
```

```
In [82]: secreturn['MSFT'].std()*250**0.5
```

```
Out[82]: 0.27649352837857893
```

```
In [83]: secreturn['AAPL'].std()*250**0.5
```

```
Out[83]: 0.31067214309514274
```

0.0.3 Calculating portfolio risk

```
In [27]: ## Equal weight
         weight = np.array([0.5,0.5])
```

```
In [84]: ## Expected return
         pfolioreturn = np.dot( secreturn,weight.T)*250
         pfolioreturn.mean()
```

```
Out[84]: nan
```

```
In [86]: # Portfolio variance
         pfolio = np.dot(weight.T, np.dot(secreturn.cov()*250,weight))
         pfolio
```

```
Out[86]: 0.06351417158332734
```

```
In [103]: # Portfolio variability
          pfoliovar = np.dot(weight.T, np.dot(secreturn.cov()*250,weight))**0.5
          pfoliovar
```

```
Out[103]: 0.2520201809048778
```

Effecient frontier

```
In [31]: assets = ['MSFT','^GSPC']
         pfdata = pd.DataFrame()
```

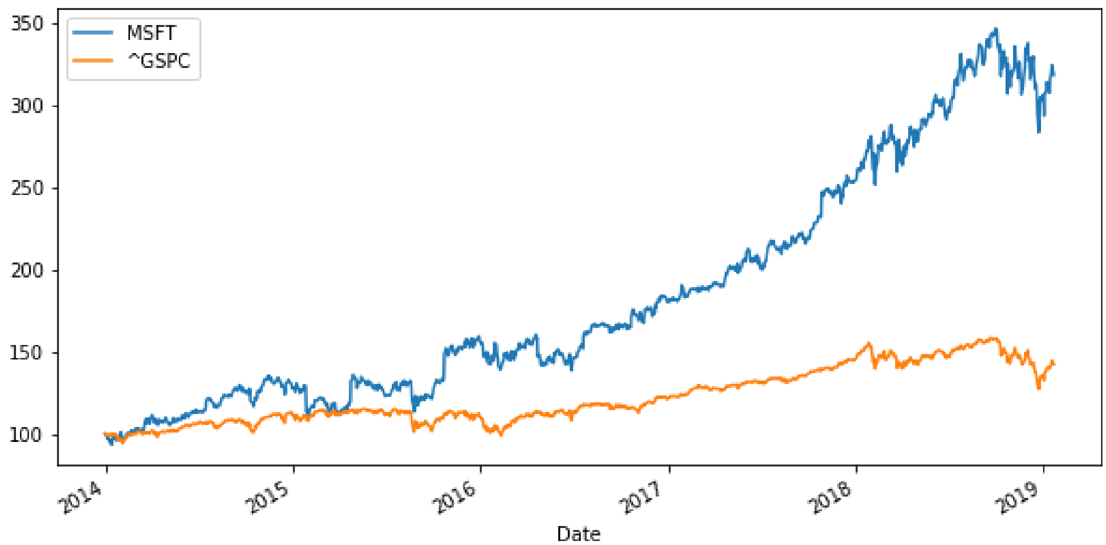
```
In [32]: for i in assets:
             pfdata[i] = data.DataReader(i,data_source='yahoo',start='2014-01-01')['Adj Close'
```

```
In [33]: pfdata.tail()
```

Out[33]:
```
                   MSFT        ^GSPC
Date
2019-01-15  105.010002  2610.300049
2019-01-16  105.379997  2616.100098
2019-01-17  106.120003  2635.959961
2019-01-18  107.709999  2670.709961
2019-01-22  105.680000  2632.899902
```

In [34]:
```
(pfdata/pfdata.iloc[0]*100).plot(figsize=(10,5))
```

Out[34]:
```
<matplotlib.axes._subplots.AxesSubplot at 0xf2a8550>
```



In [35]:
```
import numpy as np
logreturn = np.log(pfdata/pfdata.shift(1))
```

In [36]:
```
logreturn.corr()
```

Out[36]:
```
           MSFT     ^GSPC
MSFT   1.000000  0.731257
^GSPC  0.731257  1.000000
```

In [37]:
```
weight = np.random.random(2)
weight /= np.sum(weight)
weight
```

Out[37]:
```
array([0.1771861, 0.8228139])
```

In [38]:
```
#### Expected portfolio return
import numpy as np
np.sum(weight * logreturn.mean()) * 250
```

Out[38]: 0.09756306473017294

In [39]:
```
##### Expected portfolio variance
np.dot(weight.T, np.dot(logreturn.cov()*250,weight))
```

Out[39]: 0.020232287747446728

In [40]:
```
pfolioreturn = []
pfoliovolatile = []

for i in range(1000):
    weight = np.random.random(2)
    weight /= np.sum(weight)
    pfolioreturn.append(np.sum(weight * logreturn.mean()) * 250)
    pfoliovolatile.append(np.sqrt(np.dot(weight.T, np.dot(logreturn.cov()*250,weight)

pfolioreturn = np.array(pfolioreturn)
pfoliovolatile = np.array(pfoliovolatile)
```

In [41]:
```
pfolio = pd.DataFrame({'Return':pfolioreturn,'Volatility':pfoliovolatile})
```

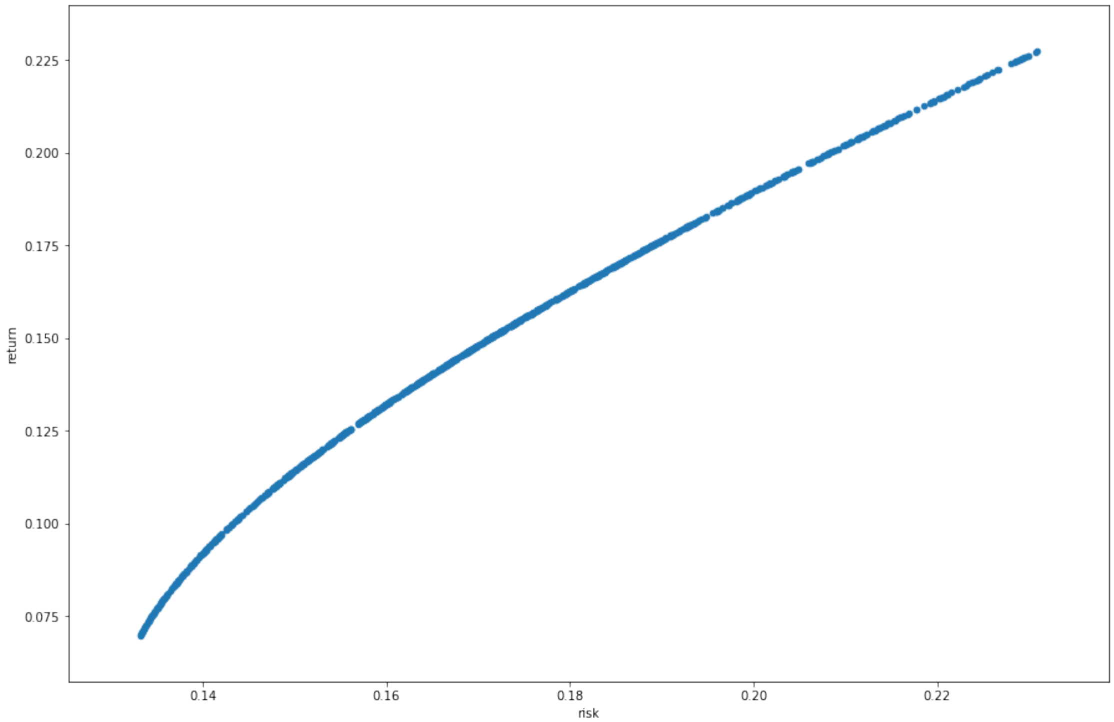
In [42]:
```
pfolio.head()
```

Out[42]:

	Return	Volatility
0	0.160392	0.178414
1	0.086939	0.138153
2	0.107055	0.146478
3	0.225568	0.229303
4	0.087708	0.138424

In [46]:
```
pfolio.plot(x='Volatility', y='Return',kind='scatter',figsize=(15,10))
plt.xlabel('risk')
plt.ylabel('return')
```

Out[46]: Text(0,0.5,'return')



0.0.4 Calculating the beta stocks

```
In [47]: pfdata.head()
```

```
Out[47]:                  MSFT        ^GSPC
         Date
         2013-12-31  33.173367  1848.359985
         2014-01-02  32.951675  1831.979980
         2014-01-03  32.729988  1831.369995
         2014-01-06  32.038334  1826.770020
         2014-01-07  32.286613  1837.880005
```

```
In [108]: cov = pfdata.cov()*250
```

```
In [110]: cov
```

```
Out[110]:                MSFT         ^GSPC
          MSFT   1.385049e+05  1.772464e+06
          ^GSPC  1.772464e+06  2.446858e+07
```

```
In [112]: marketvariance = pfdata['^GSPC'].var()*250
          marketvariance
```

```
Out[112]: 24468582.788682017
```

In [113]: covmarket = cov.iloc[0,1]
covmarket

```
Out[113]: 1772463.9293368359
```

```
In [114]: betaofpg =  covmarket/marketvarianc e
```

```
In [115]: betaofpg
```

```
Out[115]: 0.0724383567550423
```

Expected retrun (CAPM)

```
In [116]: pgexpret = 0.025 + betaofpg*0.25
```

```
In [117]: pgexpret
```

```
Out[117]: 0.04310958918876058
```

```
In [73]: import statsmodels.api as sm
```

```
In [74]: df = sm.datasets.macrodata.load_pandas().data
```

```
In [75]: df.head()
```

```
Out[75]:      year  quarter   realgdp  realcons  realinv  realgovt  realdpi    cpi  \
        0  1959.0      1.0  2710.349    1707.4  286.898   470.045   1886.9  28.98
        1  1959.0      2.0  2778.801    1733.7  310.859   481.301   1919.7  29.15
        2  1959.0      3.0  2775.488    1751.8  289.226   491.260   1916.4  29.35
        3  1959.0      4.0  2785.204    1753.7  299.356   484.052   1931.3  29.37
        4  1960.0      1.0  2847.699    1770.5  331.722   462.199   1955.5  29.54

              m1  tbilrate  unemp      pop  infl  realint
        0  139.7      2.82    5.8  177.146  0.00     0.00
        1  141.7      3.08    5.1  177.830  2.34     0.74
        2  140.5      3.82    5.3  178.657  2.74     1.09
        3  140.0      4.33    5.6  179.386  0.27     4.06
        4  139.6      3.50    5.2  180.007  2.31     1.19
```

```
In [76]: print(sm.datasets.macrodata.NOTE)
```

```
::
    Number of Observations - 203

    Number of Variables - 14

    Variable name definitions::

        year      - 1959q1 - 2009q3
        quarter   - 1-4
```

```
realgdp   - Real gross domestic product (Bil. of chained 2005 US$,
            seasonally adjusted annual rate)
realcons  - Real personal consumption expenditures (Bil. of chained
            2005 US$, seasonally adjusted annual rate)
realinv   - Real gross private domestic investment (Bil. of chained
            2005 US$, seasonally adjusted annual rate)
realgovt  - Real federal consumption expenditures & gross investment
            (Bil. of chained 2005 US$, seasonally adjusted annual rate)
realdpi   - Real private disposable income (Bil. of chained 2005
            US$, seasonally adjusted annual rate)
cpi       - End of the quarter consumer price index for all urban
            consumers: all items (1982-84 = 100, seasonally adjusted).
m1        - End of the quarter M1 nominal money stock (Seasonally
            adjusted)
tbilrate  - Quarterly monthly average of the monthly 3-month
            treasury bill: secondary market rate
unemp     - Seasonally adjusted unemployment rate (%)
pop       - End of the quarter total population: all ages incl. armed
            forces over seas
infl      - Inflation rate (ln(cpi_{t}/cpi_{t-1}) * 400)
realint   - Real interest rate (tbilrate - infl)
```