FE-520 Intro to Python for Financial Applications

Markowitz model and Portfolio efficient frontier

Final Report Group 17

Sumit Gupta (10441745)
Mukesh S Bengaluru (10436083)
Dushyanth S Nandeesh (10441771)

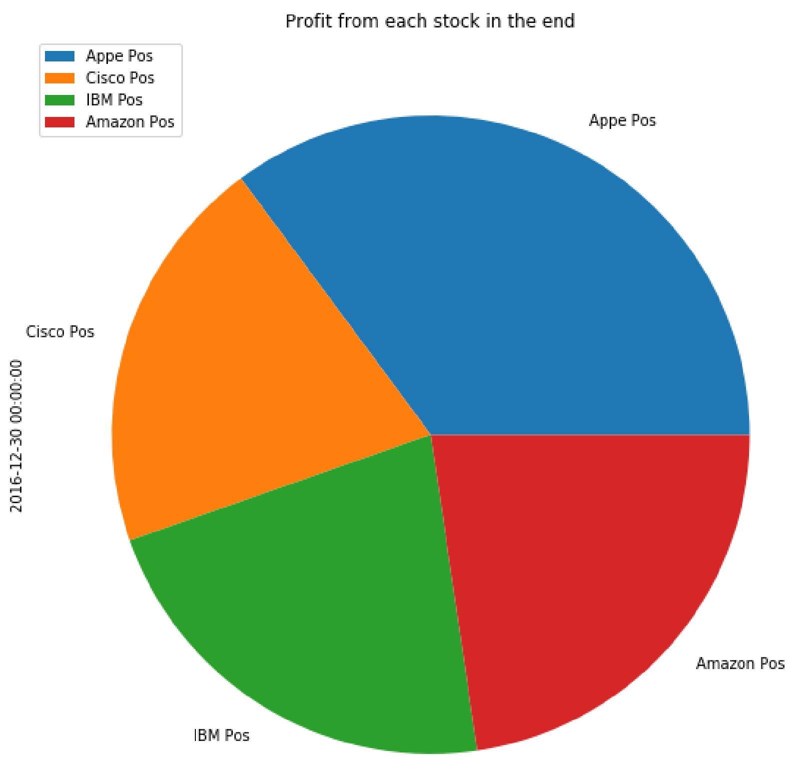
Contents

Introduction

Thinking about managing your own stock portfolio? Markowitz portfolio theory in python to minimize the variance of your portfolio given a set target average return. The higher of a return you want, the higher of a risk (variance) you will need to take on. This optimization problem will find the optimal weights for each asset in the portfolio.

Portfolio

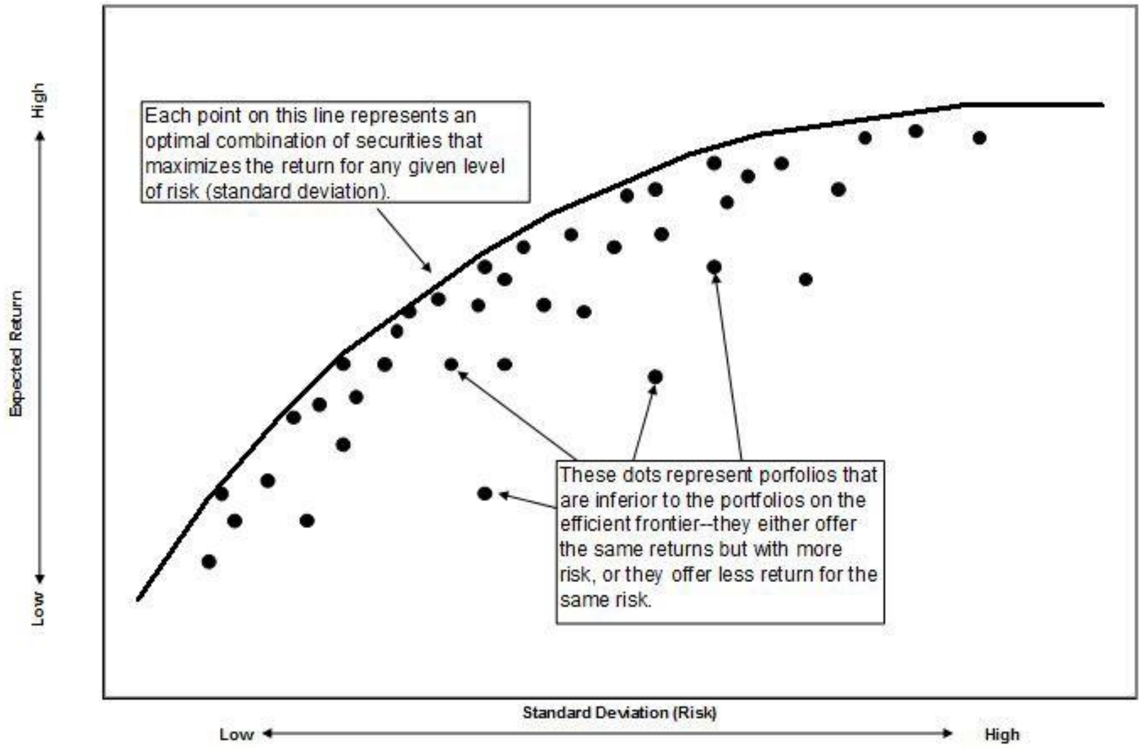
A portfolio is a grouping of financial assets such as stocks, bonds, commodities, currencies and cash equivalents, as well as their fund counterparts, including mutual, exchange-traded and closed funds. A portfolio can also consist of non-publicly tradable securities, like real estate, art, and private investments.



Modern Portfolio Theory

Harry Markowitz's contribution to the world of finance and economics cannot be emphasized enough. He is widely regarded as the pioneer of **Modern Portfolio Theory (MPT)** with his ground-breaking paper "*Portfolio Selection*" in 1952. He eventually won a **Nobel Memorial Prize in 1990** in Economic Sciences for his contribution to the field.

Modern Portfolio Theory is a theory about how investors (who are risk averse) construct portfolios that maximise their expected returns for given levels of risk. The breakthrough insight from MPT was the fact that risks and returns characteristics of various investments need not be isolated and analysed but looked at how these investments affected the performance of a portfolio. The assumptions of MPT, thus, emphasise that investors only assume additional risk when there is a possibility of higher expected returns — "High risk, High Reward"



By simply constructing portfolios with different combinations of securities, investors could achieve a maximum expected return given their risk preferences due to the fact that the returns of a portfolio are greatly affected by nature of the relationship between assets and their weights in the portfolio.

Quandl

The world’s most powerful data lives on Quandl The premier source for financial, economic, and alternative datasets, serving investment professionals. Quandl’s platform is used by over 400,000 people, including analysts from the world’s top hedge funds, asset managers and investment banks.

Stocks

I have taken 10 stock into consideration that are performing really well in the market over the past few years. I used quandl to collect the closing price of each day from 2013 to 2018.

```
                Apple
Date
2013-01-02  71.195748
2013-01-03  70.296565
2013-01-04  68.338996
2013-01-07  67.937002
2013-01-08  68.119845
```

Screenshot of one of 10 Stock

I have collected the stock price of following 10 stocks

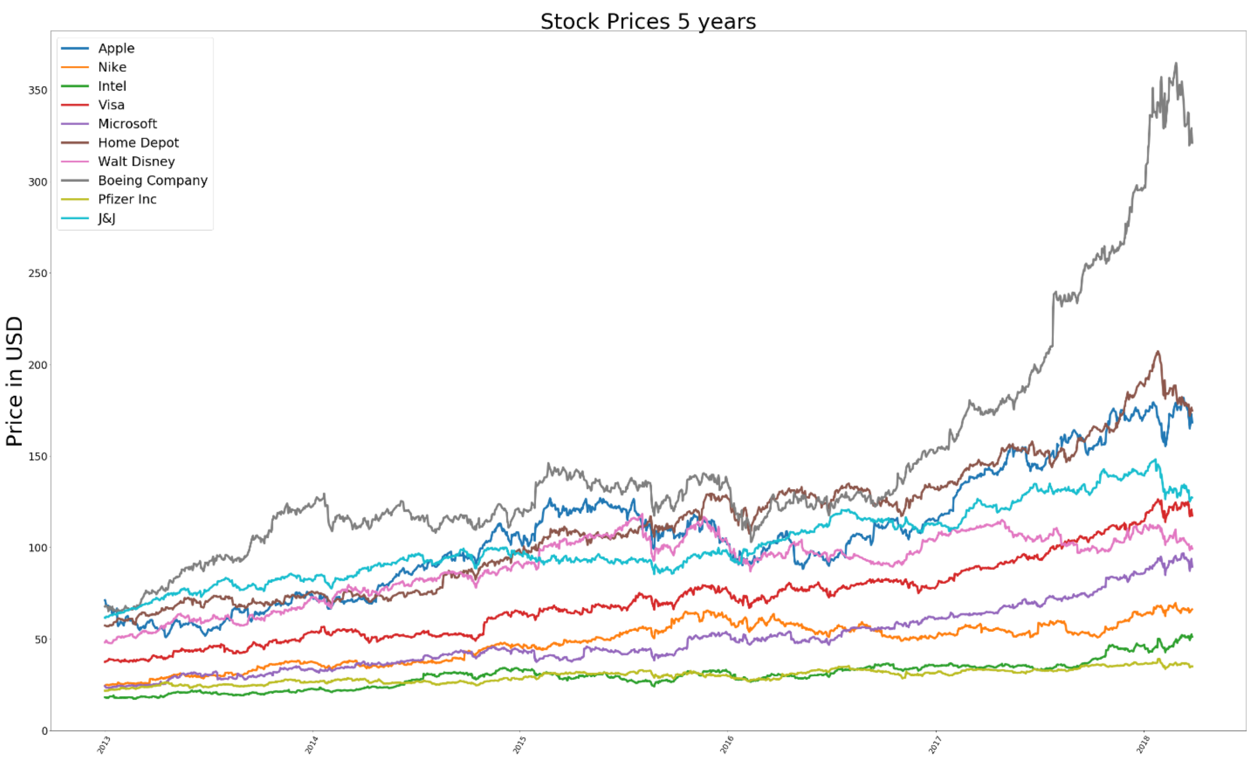
- Apple
- Nike
- Intel
- Visa
- Microsoft
- Home Depot
- Walt Disney
- Boeing Company
- Pfizer Inc
- J&J

Data Pre-processing:

I concatenated all the stock closing price for each day to form one data frame that could be used for performing portfolio optimization.

```
PS C:\Users\sumit\OneDrive\Desktop\project> python .\portfolio.py
                Apple       Nike      Intel       Visa  Microsoft  Home Depot  Walt Disney  Boeing Company  Pfizer Inc        J&J
Date
2013-01-02  71.195748  24.456742  18.101359  37.507891  24.194478   57.369885    48.169335       67.733862   21.715242  61.595109
2013-01-03  70.296565  24.706782  18.050560  37.536859  23.870367   57.207211    48.273026       68.085406   21.664955  61.508159
2013-01-04  68.338996  24.947387  17.915096  37.843430  23.423618   57.098761    49.196821       68.278756   21.757147  62.212451
2013-01-07  67.937002  24.985129  17.991295  38.113792  23.379820   56.792571    48.046790       66.907732   21.773909  62.082027
2013-01-08  68.119845  24.720935  17.855831  38.468642  23.257183   57.134911    47.848834       65.150009   21.807433  62.090722
```

Graph to show the growth of stock over the time period to understand how these stocks were performing in the market.



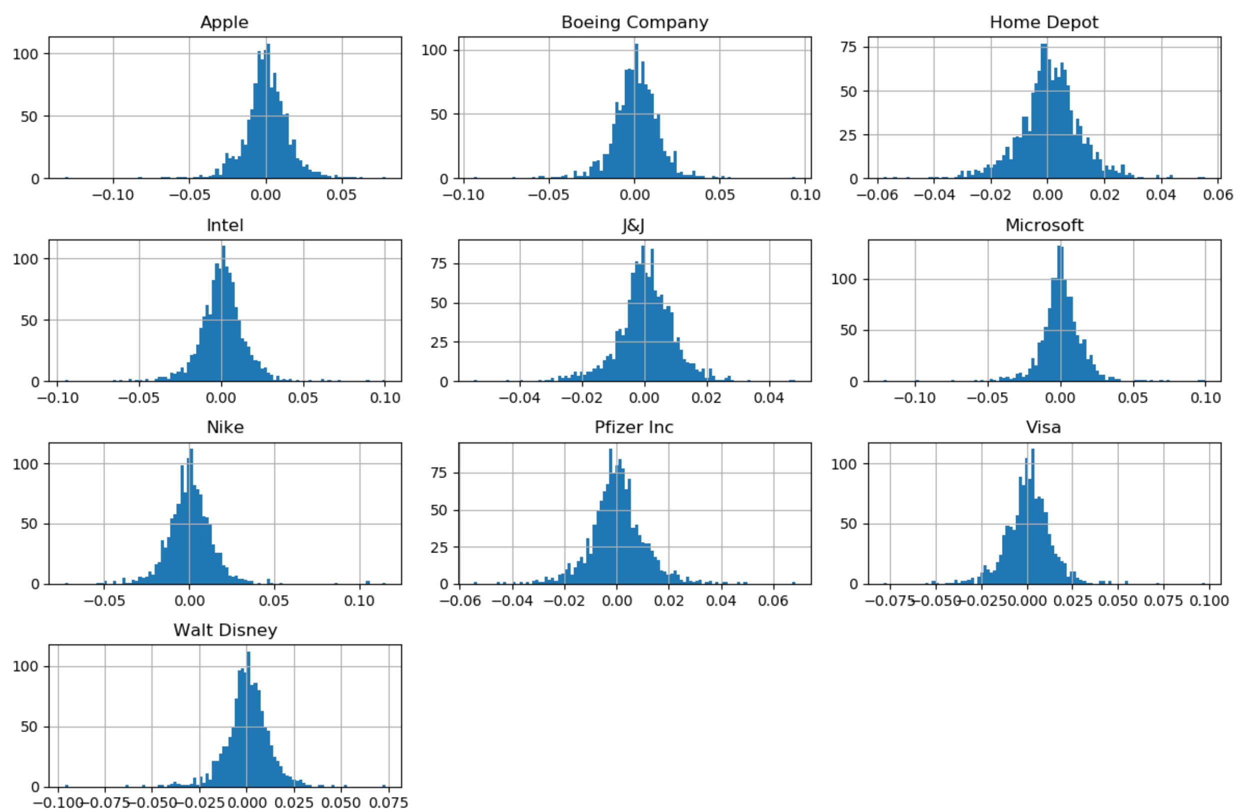
Log Returns vs Arithmetic Returns

We will now switch over to using log returns instead of arithmetic returns, for many of our use cases they are almost the same, but most technical analyses require detrending/normalizing the time series and using log returns is a nice way to do that.

Log returns are convenient to work with Monte Carlo Simulation for Optimization Search and Markowitz's Efficient Frontier

Daily Return

```
---------------------------Daily return--------------------
                 Apple       Nike      Intel       Visa  Microsoft  Home Depot  Walt Disney  Boeing Company  Pfizer Inc       J&J
Date
2013-01-02         NaN        NaN        NaN        NaN        NaN         NaN          NaN             NaN         NaN       NaN
2013-01-03 -0.012710   0.010172  -0.002810   0.000772  -0.013487   -0.002840     0.002150        0.005177   -0.002318 -0.001413
2013-01-04 -0.028242   0.009691  -0.007533   0.008134  -0.018893   -0.001898     0.018956        0.002836    0.004246  0.011385
2013-01-07 -0.005900   0.001512   0.004244   0.007119  -0.001872   -0.005377    -0.023654       -0.020284    0.000770 -0.002099
2013-01-08  0.002688  -0.010630  -0.007558   0.009267  -0.005259    0.006010    -0.004129       -0.026622    0.001538  0.000140
```



Daily return Graph

Summary of the log return:

--------------------Description of Log Return-------------

	count	mean	std	min	25%	50%	75%	max
Apple	1314.0	0.000637	0.015163	-0.131875	-0.006357	0.000425	0.008700	0.078794
Nike	1316.0	0.000756	0.013743	-0.073114	-0.006416	0.000505	0.007643	0.115342
Intel	1314.0	0.000789	0.014198	-0.095432	-0.006452	0.000872	0.007880	0.100315
Visa	1316.0	0.000867	0.012812	-0.078368	-0.005440	0.001174	0.007949	0.097527
Microsoft	1316.0	0.000994	0.014375	-0.121033	-0.005856	0.000589	0.007781	0.099413
Home Depot	1316.0	0.000846	0.011222	-0.057616	-0.004633	0.000868	0.006957	0.055384
Walt Disney	1316.0	0.000550	0.011692	-0.096190	-0.005182	0.000849	0.006794	0.073531
Boeing Company	1316.0	0.001183	0.013726	-0.093531	-0.005886	0.001488	0.009008	0.094214
Pfizer Inc	1316.0	0.000363	0.010744	-0.054447	-0.005217	0.000000	0.005562	0.068282
J&J	1316.0	0.000551	0.009133	-0.054402	-0.003795	0.000497	0.005708	0.048395

Yearly Covariance:

------------Yearly covariance-----------

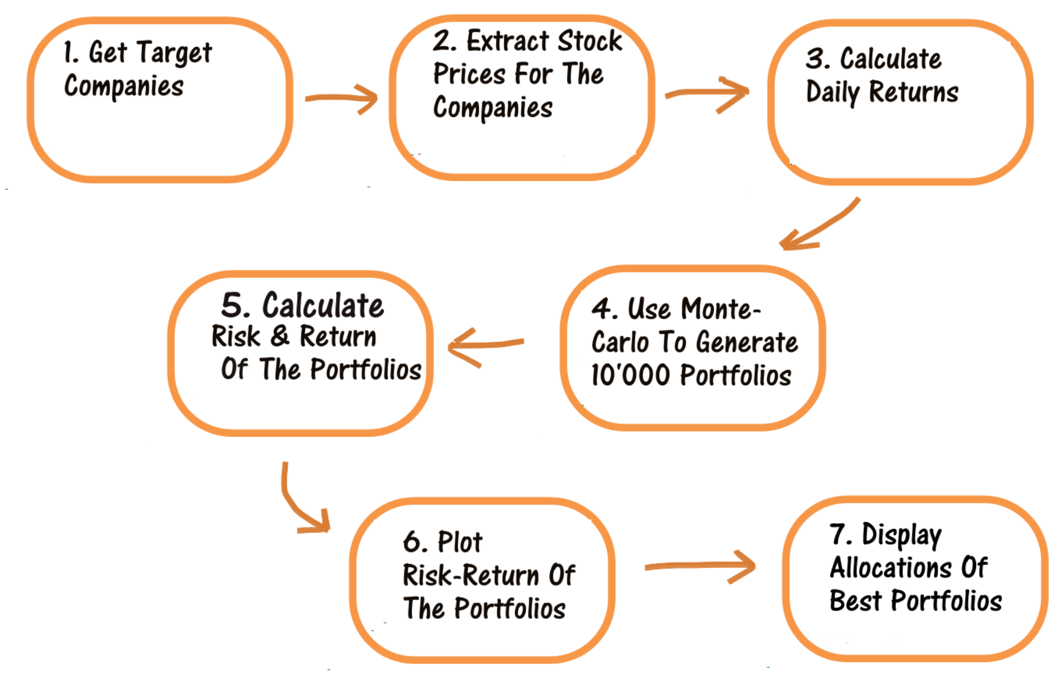
	Apple	Nike	Intel	Visa	Microsoft	Home Depot	Walt Disney	Boeing Company	Pfizer Inc	J&J
Apple	0.057940	0.012465	0.018291	0.015858	0.020504	0.012310	0.012392	0.015189	0.008950	0.008517
Nike	0.012465	0.047595	0.012364	0.017703	0.015016	0.017146	0.016156	0.015784	0.011189	0.010418
Intel	0.018291	0.012364	0.050800	0.017664	0.026261	0.014368	0.015586	0.016986	0.013406	0.011359
Visa	0.015858	0.017703	0.017664	0.041362	0.021087	0.016053	0.016111	0.018418	0.014680	0.013402
Microsoft	0.020504	0.015016	0.026261	0.021087	0.052070	0.015474	0.014904	0.016240	0.012157	0.011809
Home Depot	0.012310	0.017146	0.014368	0.016053	0.015474	0.031735	0.014750	0.014822	0.012232	0.010851
Walt Disney	0.012392	0.016156	0.015586	0.016111	0.014904	0.014750	0.034451	0.016587	0.012106	0.010569
Boeing Company	0.015189	0.015784	0.016986	0.018418	0.016240	0.014822	0.016587	0.047478	0.011911	0.012451
Pfizer Inc	0.008950	0.011189	0.013406	0.014680	0.012157	0.012232	0.012106	0.011911	0.029088	0.013486
J&J	0.008517	0.010418	0.011359	0.013402	0.011809	0.010851	0.010569	0.012451	0.013486	0.021019

Monte Carlo Simulation

This simulation is extensively used in portfolio optimization. In this simulation, we will assign random weights to the stocks. One important point to keep in mind is that the sum of the weights should always sum up to 1. At every particular combination of these weights, we will compute the return and standard deviation of the portfolio and save it. We'll then change the weights and assign some random values and repeat the above procedure.

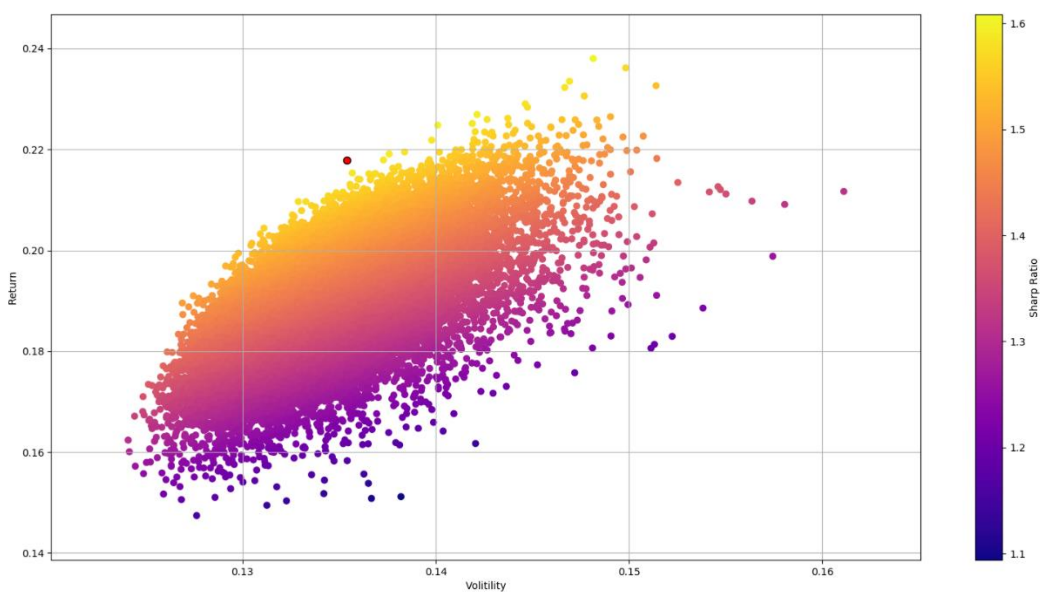
The number of iterations depends on the error that the trader is willing to accept. Higher the number of iterations, higher will be the accuracy of the optimization but at the cost of computation and time.

Steps:

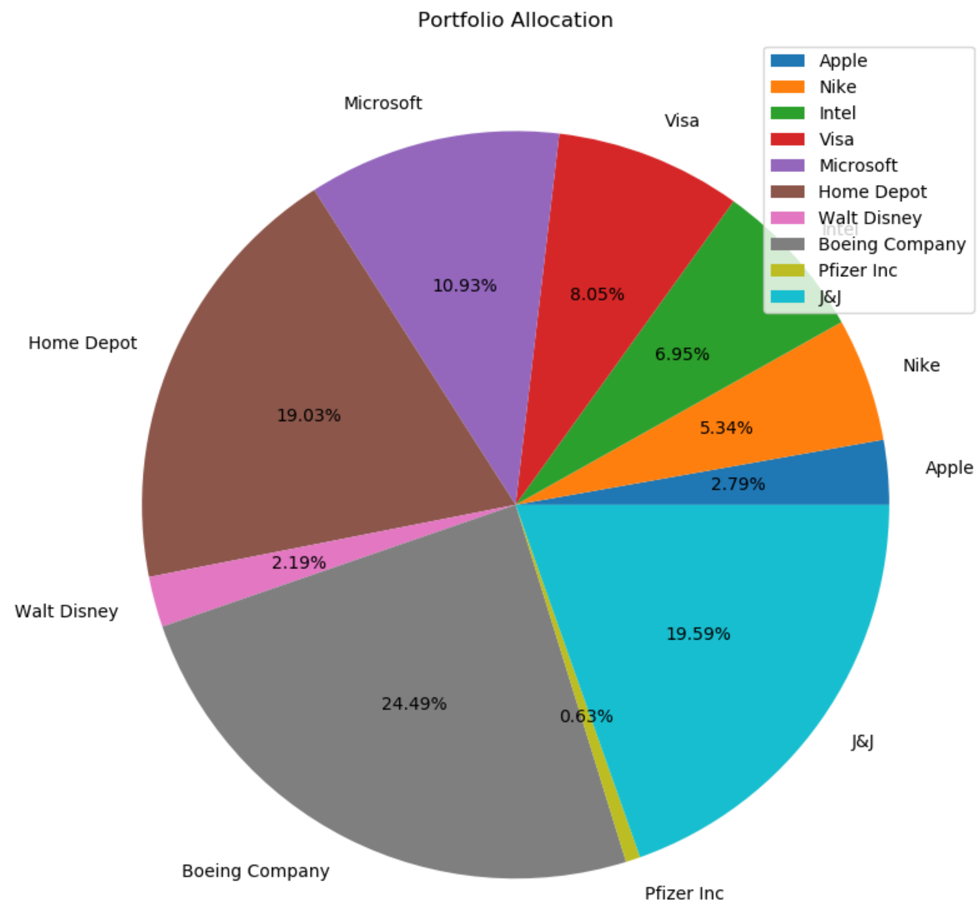


Maximum sharp ratio achieved and portfolio allocation for this.

```
====================Random generated=================================

Maximum Sharp Ratio(using random number generation) : 1.608817368819721
Maximum Sharpe Ratio Portfolio Allocation

                allocation
Apple                 2.79
Nike                  5.34
Intel                 6.95
Visa                  8.05
Microsoft            10.93
Home Depot           19.03
Walt Disney           2.19
Boeing Company       24.49
Pfizer Inc            0.63
J&J                  19.59
Optimization using random ssampling graph generate...

=========================================================================
```



Portfolio Allocation Pie chart:



Mathematically optimized

There are much better ways to find good allocation weights than just guess and check!

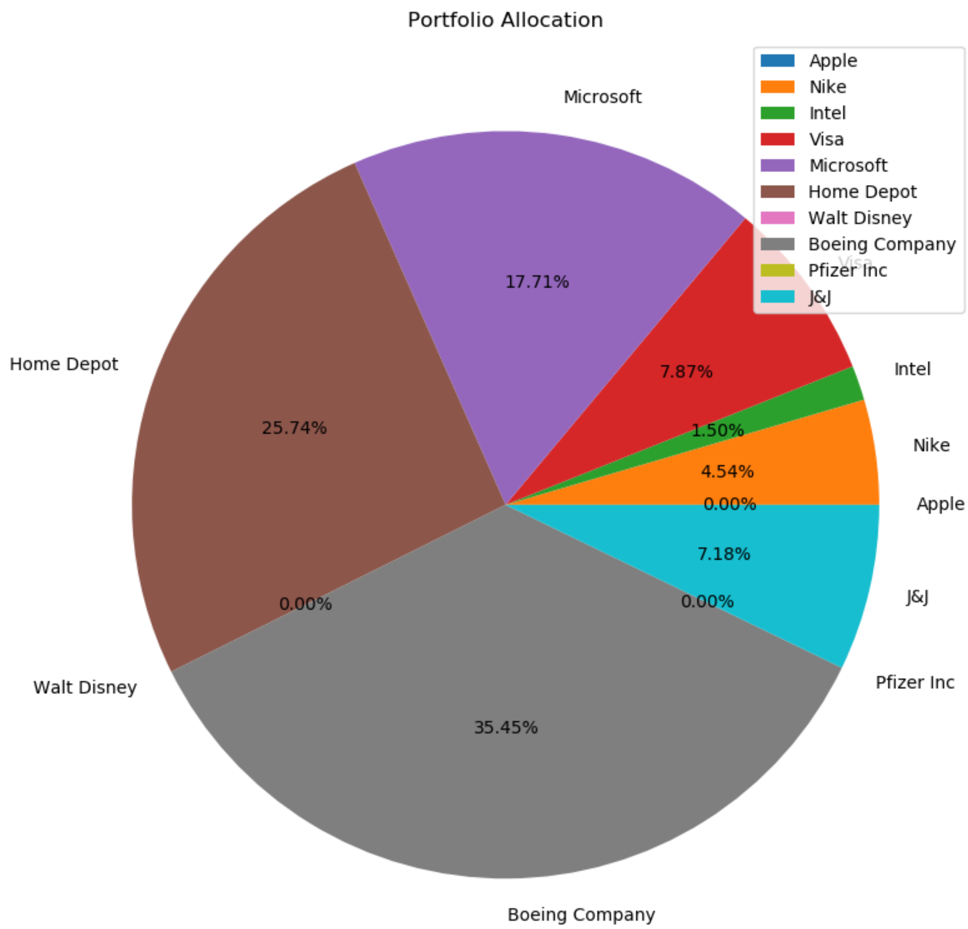
We can use optimization functions to find the ideal weights mathematically!

Optimization works as a minimization function, since we actually want to maximize the Sharpe Ratio, we will need to turn it negative so we can minimize the negative Sharpe (same as maximizing the positive Sharpe)

```
-------Mathematical Optimized Result set---------

     fun: -1.6449962611572726
     jac: array([ 3.19207013e-02, -4.72798944e-04,  4.73111868e-05,  5.46053052e-04,
       -3.39001417e-04,  2.58386135e-05,  2.11826891e-01,  9.77218151e-05,
        3.08924764e-01, -4.81456518e-05])
 message: 'Optimization terminated successfully.'
    nfev: 97
     nit: 8
    njev: 8
  status: 0
 success: True
       x: array([2.15998543e-17, 4.54312129e-02, 1.50290440e-02, 7.87281837e-02,
       1.77136337e-01, 2.57398502e-01, 6.03604739e-17, 3.54493409e-01,
       6.63321610e-17, 7.17833118e-02])
```

```
=======================Mathematically Maximized==============================

Mathematically Maximized Sharp ratio :  1.6449962611572726
Maximum Sharpe Ratio Portfolio Allocation

                allocation
Apple                 0.00
Nike                  4.54
Intel                 1.50
Visa                  7.87
Microsoft            17.71
Home Depot           25.74
Walt Disney           0.00
Boeing Company       35.45
Pfizer Inc            0.00
J&J                   7.18

Mathematical Optimization and Efficient forointier graph generated...

=========================================================================
```

Portfolio Allocation Pie chart:

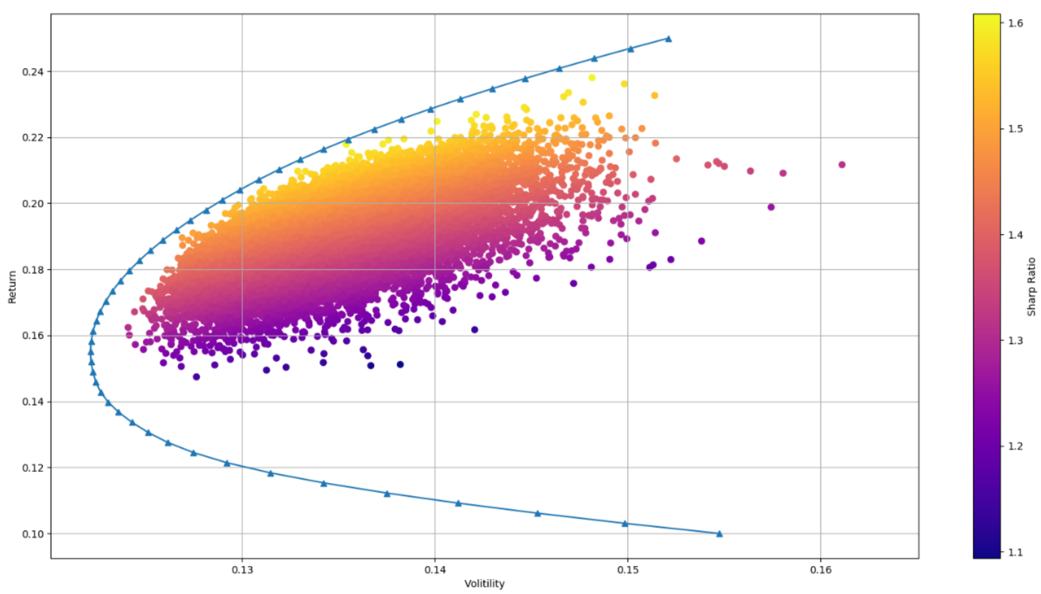


STEVENS INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY®

Markowitz's Efficient Frontier

All Optimal Portfolios (Efficient Frontier)

The efficient frontier is the set of optimal portfolios that offers the highest expected return for a defined level of risk or the lowest risk for a given level of expected return. Portfolios that lie below the efficient frontier are sub-optimal, because they do not provide enough return for the level of risk. Portfolios that cluster to the right of the efficient frontier are also sub-optimal, because they have a higher level of risk for the defined rate of return.



Appendix

Code:

```
import pandas as pd
import numpy as np
import quandl
import matplotlib.pyplot as plt
from pandas import DataFrame
from scipy.optimize import minimize
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
# Dates for which stock data is collected
start = pd.to_datetime('2013-01-01')
end = pd.to_datetime('2019-01-01')
# Collecting stock data using QUANDL
aapl = quandl.get('WIKI/AAPL.11',start_date=start,end_date=end)
nike = quandl.get('WIKI/NKE.11',start_date=start,end_date=end)
intel = quandl.get('WIKI/INTC.11',start_date=start,end_date=end)
visa = quandl.get('WIKI/V.11',start_date=start,end_date=end)
msft = quandl.get('WIKI/MSFT.11',start_date=start,end_date=end)
hodp = quandl.get('WIKI/HD.11',start_date=start,end_date=end)
disc = quandl.get('WIKI/DIS.11',start_date=start,end_date=end)
ba = quandl.get('WIKI/BA.11',start_date=start,end_date=end)
pfizer = quandl.get('WIKI/PFE.11',start_date=start,end_date=end)
jnj = quandl.get('WIKI/JNJ.11',start_date=start,end_date=end)
# Merging all the stock to for one file.
stock = pd.concat([aapl,nike,intel,visa,msft,hodp,disc,ba,pfizer,jnj],axis=1)
stock.columns = ['Apple','Nike', 'Intel', 'Visa','Microsoft','Home Depot','Walt D
isney','Boeing Company', 'Pfizer Inc','J&J']
# DIsplay the head of the data frame.
#display(stock.head())
print(stock.head())
#graph of 10 Stocks
f=plt.figure(figsize=(50,30))
plt.plot(stock,linewidth=5)
plt.title('Stock Prices over the years',fontsize=50)
plt.xticks(fontsize=18,rotation=60)
plt.yticks(fontsize=24)
plt.ylabel('Price in USD',fontsize=50)
plt.legend(stock.columns ,loc=2, prop={'size': 30})
print('\nStock Prices over the years Generated...............-\n')
# reating daily return.
print("----------------------------Daily return--------------------")
```

```
log_ret =np.log(stock/stock.shift(1))
print(log_ret.head())
log_ret.hist(bins=100,figsize=(12,8))
g = plt.tight_layout()
print('\nDaily Return graph generated.............\n')
print('--------------------Description of Log Return-------------')
print(log_ret.describe().transpose())
print("\n-------------Yearly covariance-----------")
# Compute pairwise covariance of columns
print(log_ret.cov()*252)
# predicting charp ratio using random values of weight and scaling it to 1
np.random.seed(1276)
# Finding optimum in 25000 repititions
num_ports = 25000
all_weight = np.zeros((num_ports,len(stock.columns)))
ret_arr = np.zeros(num_ports)
vol_arr = np.zeros(num_ports)
sharp_arr = np.zeros(num_ports)
for i in range(num_ports):
    weight = np.array(np.random.random(10))
    weight = weight/np.sum(weight)
    #Save the weight
    all_weight[i,:]=weight
    # Expected Return
    ret_arr[i] = np.sum( (log_ret.mean()* weight)*252)
    #Expected Volitility
    vol_arr[i] = np.sqrt(np.dot(weight,np.dot(log_ret.cov()*252,weight)))
    #Sharp Ratio
    sharp_arr[i]= ret_arr[i]/vol_arr[i]

print("\n====================Random generated===============================\n")
print("Maximum Sharp Ratio(using random number generation) :",sharp_arr.max())
print("Maximum Sharpe Ratio Portfolio Allocation\n")
print('Portfolio allocation graph generate...')
max_sharpe_allocation = pd.DataFrame(all_weight[sharp_arr.argmax()],index=stock.c
olumns,columns=['allocation'])
max_sharpe_allocation.allocation = [round(i*100,2)for i in max_sharpe_allocation.
allocation]
print(max_sharpe_allocation)
max_sr_ret = ret_arr[sharp_arr.argmax()]
max_sr_vol = vol_arr[sharp_arr.argmax()]
l=plt.figure(figsize=(20,10))
plt.scatter(vol_arr,ret_arr,c=sharp_arr,cmap='plasma')
plt.colorbar(label='Sharp Ratio')
plt.xlabel('Volitility')
```

```
plt.ylabel('Return')
plt.grid(True)
plt.scatter(max_sr_vol,max_sr_ret,c='red',s=50,edgecolors='black')
print('Optimization using random sampling graph generate...')
print("\n=====================================================================\n")
# Generate pie charf for east understanding of portfolio
df = DataFrame (max_sharpe_allocation)
p = plt.figure(figsize=(10,10))
plt.pie(df['allocation'],labels=df.index,autopct='%1.2f%%')
plt.legend()
plt.title("Portfolio Allocation")
#Mathematical Optimization
def get_ret_vol_sr(weight):
    weight = np.array(weight)
    ret = np.sum(log_ret.mean()*weight) *252
    vol = np.sqrt(np.dot(weight.T,np.dot(log_ret.cov()*252,weight)))
    sr=ret/vol
    return np.array([ret,vol,sr])
def neg_sharp(weight):
    return get_ret_vol_sr(weight)[2]*-1
def check_sum(weight):
    # if sum is one it returns zero
    return np.sum(weight)-1
cons = ({'type':'eq','fun':check_sum})
bound = ((0,1),(0,1),(0,1),(0,1),(0,1),(0,1),(0,1),(0,1),(0,1),(0,1))
init_guess = [0.10,0.10,0.10,0.10,.10,0.10,0.10,0.10,0.10,.10]
opt_result = minimize(neg_sharp,init_guess,method='SLSQP',bounds=bound,constraint
s=cons)
print("\n-------Mathematical Optimized Result set---------\n")
print(opt_result)
print("\n================Mathematically Maximized===========================\n")
print("Mathematically Maximized Sharp ratio : ",list(get_ret_vol_sr(opt_result.x)
)[2])
print("Maximum Sharpe Ratio Portfolio Allocation\n")
max_sharpe_allocation_new = pd.DataFrame(list(opt_result.x),index=stock.columns,c
olumns=['allocation'])
max_sharpe_allocation_new.allocation = [round(i*100,2)for i in max_sharpe_allocat
ion_new.allocation]
print(max_sharpe_allocation_new)
print('Portfolio allocation graph generate...')
df = DataFrame (max_sharpe_allocation_new)
o = plt.figure(figsize=(10,10))
plt.pie(df['allocation'],labels=df.index,autopct='%1.2f%%')
plt.legend()
plt.title("Portfolio Allocation")
```

```
#Efficient forointier
frointier_y = np.linspace(.10,.25,50)
def minimize_vol(weight):
    return get_ret_vol_sr(weight)[1]
frointier_vol = []
for possible_return in frointier_y:
    cons = ({'type':'eq','fun':check_sum},
       {'type':'eq','fun': lambda w: get_ret_vol_sr(w)[0]-possible_return})
    result = minimize(minimize_vol,init_guess,method='SLSQP',bounds=bound,constra
ints=cons)
    frointier_vol.append(result['fun'])
p=plt.figure(figsize=(20,10))
plt.scatter(vol_arr,ret_arr,c=sharp_arr,cmap='plasma')
plt.colorbar(label='Sharp Ratio')
plt.xlabel('Volitility')
plt.ylabel('Return')
plt.grid(True)
plt.plot(frointier_vol,frointier_y,marker='^')
print("\nEfficient forointier graph generated... ")
print("\n=================================================================\n")
plt.show()
input()
```

Reference:

https://medium.com/python-data/effient-frontier-in-python-34b0c3043314

https://medium.com/python-data/efficient-frontier-portfolio-optimization-with-python-part-2-2-2fe23413ad94

https://www.pythonforfinance.net/2017/01/21/investment-portfolio-optimisation-with-python/

https://www.linkedin.com/pulse/markowitz-portfolio-optimization-python-f%C3%A1bio-neves/