

[Home](#)[Courses ▼](#)[Trading Platform](#)[Contact Us](#)[Career Growth ▼](#)[Tutorials ▼](#)

Search in the blog...



# Portfolio Optimization Using Monte Carlo Simulation

Portfolio & Risk Management

📅 Feb 08, 2018

🕒 6 min read

By **Mandeep Kaur**



[Our cookie policy ▲](#)



In the [previous blog](#) of this series, we saw how to compute the mean and the risk (or standard deviation) of a portfolio containing ' $n$ ' number of stocks, each stock ' $i$ ' having a weight of ' $w_i$ '.

In this blog, we will see how to do [portfolio optimization](#) by changing these weights. By portfolio optimization, we mean getting a portfolio that meets any of the three conditions based on the investor's requirements.

## Conditions of Portfolio Optimization

1. A portfolio which has the minimum risk for the desired level of expected return.
2. A portfolio which gives the maximum expected return at the desired level of risk (risk as measured in terms of standard deviation or variance).
3. A portfolio which has the maximum return to risk ratio (or [Sharpe ratio](#)).

## Annual Returns and Standard Deviation

To simplify our analysis in this blog, we will deal with daily returns and standard deviation and will consider only 1 month of [stock data](#) (Dec 2017). However, in practice, we work with annual returns

and standard deviation. The formulae for converting daily returns and standard deviation to an annual basis are as shown (assuming 252 trading days in a year):

$$\text{Annual Return} = \text{Daily Return} * 252 \quad \text{Annual Standard Deviation} = \text{Daily Standard Deviation} * 252$$

Let us consider a portfolio consisting of four stocks in banking/financial services sector, namely: Bank of America (BAC), Goldman Sachs (GS), JP Morgan Chase & Co (JPM) and Morgan Stanley (MS).

To start with, we'll assign random weights to all four stocks, keeping the sum of the weights to be 1. We will compute the return and standard deviation of the portfolio as we did in the previous blog and record it. We will then run Monte Carlo Simulations on our portfolio to get the optimal weights for the stocks. We will use [python](#) to demonstrate how portfolio optimization can be achieved. Before moving on to the step-by-step process, let us quickly have a look at [Monte Carlo Simulation](#).

## 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. For the purpose of this blog, we will restrict ourselves to 10000 such iterations. Out of these 10000 results for returns and corresponding standard deviation, we can then achieve portfolio optimization by identifying a portfolio that satisfies on any of the 3 conditions discussed above.

## Portfolio Optimization Process in Python

Let's start by importing relevant **libraries** and fetching the data for the stocks for Dec 2017.

```
#Import relevant Libraries  
import pandas as pd  
import numpy as np  
import pandas_datareader.data as web  
import matplotlib.pyplot as plt
```

```
#Fetch data from yahoo and save under DataFrame named 'data'  
stock = ['BAC', 'GS', 'JPM', 'MS']  
data = web.DataReader(stock, data_source="yahoo", start='12/01/2017', end='12/31/2017')['Adj C
```

```
#Arrange the data in ascending order  
data=data.iloc[::-1]  
print (data.round(2))
```

The data looks as shown:

	BAC	GS	JPM	MS
Date				
2017-11-30	28.17	247.64	103.98	51.61
2017-12-01	28.10	248.95	104.25	51.95
2017-12-04	29.06	250.65	106.40	52.68
2017-12-05	28.93	248.33	105.17	52.01
2017-12-06	28.64	245.95	104.39	51.69
2017-12-07	28.78	248.56	104.08	52.35
2017-12-08	29.05	250.35	105.38	52.89
2017-12-11	28.94	250.13	105.07	52.77
2017-12-12	29.32	257.68	106.30	53.85
2017-12-13	28.84	255.56	104.96	53.18
2017-12-14	28.73	255.48	104.12	52.64
2017-12-15	29.04	257.17	105.59	53.10
2017-12-18	29.48	260.02	106.41	53.24
2017-12-19	29.45	256.48	105.96	52.93
2017-12-20	29.48	255.18	105.59	52.51
2017-12-21	29.82	261.01	107.27	52.88
2017-12-22	29.88	258.97	106.89	52.72
2017-12-26	29.78	257.72	106.47	52.47
2017-12-27	29.73	255.95	106.66	52.57
2017-12-28	29.80	256.50	107.23	52.65
2017-12-29	29.52	254.76	106.39	52.47

We will then convert these stock prices into returns and will save this under the name 'stock\_ret'.

```
#Compute stock returns and print the returns in percentage format
```

```
stock_ret = data.pct_change()
```

```
print (stock_ret.round(4)*100)
```

	BAC	GS	JPM	MS
Date				
2017-11-30	NaN	NaN	NaN	NaN
2017-12-01	-0.25	0.53	0.26	0.66
2017-12-04	3.42	0.68	2.06	1.41
2017-12-05	-0.45	-0.93	-1.15	-1.27
2017-12-06	-1.00	-0.96	-0.75	-0.62
2017-12-07	0.49	1.06	-0.30	1.28
2017-12-08	0.94	0.72	1.25	1.03
2017-12-11	-0.38	-0.09	-0.29	-0.23
2017-12-12	1.31	3.02	1.16	2.05
2017-12-13	-1.64	-0.82	-1.25	-1.24
2017-12-14	-0.38	-0.03	-0.81	-1.02
2017-12-15	1.08	0.66	1.41	0.87
2017-12-18	1.52	1.11	0.77	0.26
2017-12-19	-0.10	-1.36	-0.42	-0.58
2017-12-20	0.10	-0.51	-0.35	-0.79
2017-12-21	1.15	2.28	1.59	0.70
2017-12-22	0.20	-0.78	-0.35	-0.30
2017-12-26	-0.33	-0.48	-0.40	-0.47
2017-12-27	-0.17	-0.69	0.19	0.19
2017-12-28	0.24	0.21	0.53	0.15
2017-12-29	-0.94	-0.68	-0.79	-0.34

We will now calculate the mean return of all stocks and the covariance matrix.

```
#Calculate mean returns and covariances of all four the stocks
mean_returns = stock_ret.mean()
cov_matrix = stock_ret.cov()
print (mean_returns)
print (cov_matrix)
```

```
BAC    0.002405
GS     0.001470
JPM    0.001195
MS     0.000870
dtype: float64
```

	BAC	GS	JPM	MS
BAC	0.000124	0.000081	0.000095	0.000078
GS	0.000081	0.000128	0.000081	0.000086
JPM	0.000095	0.000081	0.000094	0.000075
MS	0.000078	0.000086	0.000075	0.000087

Let us define an array to hold the result of each iteration. The array will hold the returns, standard deviation, **Sharpe ratio** and weights for each step in the iteration. We will define one result array initially containing all zeroes and will save the simulation results in this array. The number of columns in the array is 7 to hold portfolio return, standard deviation, Sharpe Ratio and the weights of all stocks. The number of columns will change with the number of stocks in the **portfolio** as we have to store the weights for all the stocks. That's why we use the 'len function' while defining the array. The number of rows in the array is equal to the number of iterations.

```
#Set the number of iterations to 10000 and define an array to hold the simulation results;  
num_iterations = 10000  
simulation_res = np.zeros((4+len(stock)-1,num_iterations))
```

Let's now move on to the iterations.

```
for i in range(num_iterations):  
#Select random weights and normalize to set the sum to 1  
    weights = np.array(np.random.random(4))  
    weights /= np.sum(weights)
```

```
#Calculate the return and standard deviation for every step  
    portfolio_return = np.sum(mean_returns * weights)  
    portfolio_std_dev = np.sqrt(np.dot(weights.T, np.dot(cov_matrix, weights)))
```

```
#Store all the results in a defined array  
    simulation_res[0,i] = portfolio_return  
    simulation_res[1,i] = portfolio_std_dev
```



```
#Calculate Sharpe ratio and store it in the array
simulation_res[2,i] = simulation_res[0,i] / simulation_res[1,i]
```

```
#Save the weights in the array
for j in range(len(weights)):
    simulation_res[j+3,i] = weights[j]
```

We then save the output in a '*pandas data frame*' for easy analysis and plotting of data.

```
sim_frame = pd.DataFrame(simulation_res.T,columns=['ret','stdev','sharpe',stock[0],stock[1]
print (sim_frame.head())
print (sim_frame.tail())
```

	ret	stdev	sharpe	BAC	GS	JPM	MS
0	0.001131	0.009189	0.123067	0.027700	0.054452	0.571253	0.346595
1	0.001397	0.009474	0.147459	0.142838	0.329737	0.338161	0.189263
2	0.001675	0.009681	0.173042	0.355449	0.293747	0.256548	0.094256
3	0.001503	0.009491	0.158379	0.239681	0.258662	0.338860	0.162797
4	0.001528	0.009553	0.159959	0.239959	0.304541	0.329268	0.126231
	ret	stdev	sharpe	BAC	GS	JPM	MS
9995	0.001602	0.009583	0.167130	0.316478	0.230561	0.330747	0.122214
9996	0.001333	0.009402	0.141731	0.115071	0.307778	0.311639	0.265512
9997	0.001571	0.009606	0.163501	0.277284	0.353731	0.192777	0.176208
9998	0.001517	0.009439	0.160689	0.312987	0.104245	0.319382	0.263386
9999	0.001602	0.009772	0.163964	0.296160	0.453444	0.017170	0.233226

The above output shows some rows of the simulation results. We can now compute the portfolios having maximum Sharpe ratio or minimum risk.

```
#Spot the position of the portfolio with highest Sharpe Ratio  
max_sharpe = sim_frame.iloc[sim_frame['sharpe'].idxmax()]
```

```
#Spot the position of the portfolio with minimum Standard Deviation  
min_std = sim_frame.iloc[sim_frame['stdev'].idxmin()]  
print ("The portfolio for max Sharpe Ratio:\n", max_sharpe)  
print ("The portfolio for min risk:\n", min_std)
```

The portfolio for max Sharpe Ratio:

```
ret      0.002224
stdev    0.010542
sharpe   0.210910
BAC      0.822040
GS       0.147847
JPM      0.009119
MS       0.020994
```

Name: 9294, dtype: float64

The portfolio for min risk:

```
ret      0.001010
stdev    0.009090
sharpe   0.111160
BAC      0.008306
GS       0.019566
JPM      0.356755
MS       0.615373
```

Name: 2260, dtype: float64

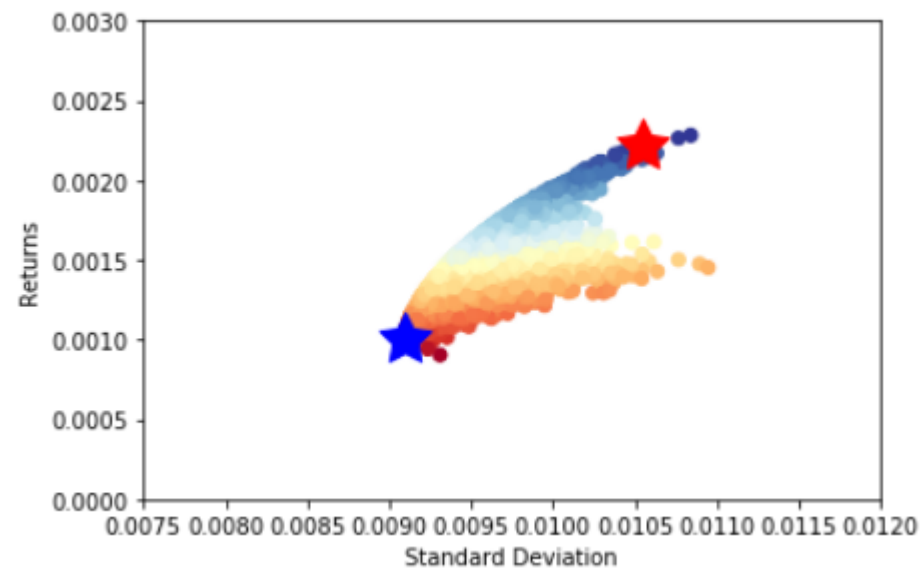
The output can be plotted using the matplotlib library as the relevant points can be highlighted as shown:

```
#Create a scatter plot coloured by various Sharpe Ratios with standard deviation on the x-axis
plt.scatter(sim_frame.stdev,sim_frame.ret,c=sim_frame.sharpe,cmap='RdYlBu')
plt.xlabel('Standard Deviation')
plt.ylabel('Returns')
plt.ylim(0,.003)
plt.xlim(0.0075,0.012)
```

```
#Plot a red star to highlight position of the portfolio with highest Sharpe Ratio  
plt.scatter(max_sharpe[1],max_sharpe[0],marker=(5,1,0),color='r',s=600)
```

```
#Plot a blue star to highlight position of the portfolio with minimum Variance  
plt.scatter(min_std[1],min_std[0],marker=(5,1,0),color='b',s=600)  
plt.show()
```

In the output, the red star shows the portfolio with the maximum Sharpe ratio and the blue star depicts the point with the minimum standard deviation.



From the above curve, we can get the composition for the required optimal portfolio based on any of the three conditions as discussed above. We can select the portfolio with maximum return for a given risk or a portfolio with minimum risk for a given return or we can simply select the portfolio with maximum Sharpe ratio.

## Summary

Just to summarize our complete analysis, we first downloaded the stock price data for one month and computed the mean return of all the stocks and the covariance matrix (which is used in computing the standard deviation of the portfolio). We then ran a Monte Carlo Simulation to compute the risk and return of the portfolio by randomly selecting the weights of the portfolio. We then identify the optimal portfolio based on the Sharpe ratio or other conditions.

## Next Step

Our next blog covers various aspects of the portfolio performance evaluation and portfolio performance measurement. It starts with why evaluation and measurement are necessary. Then explains how to compute and analyse the returns generated by the portfolio after a particular time

## Update

*We have noticed that some users are facing challenges while downloading the market data from Yahoo and Google Finance platforms. In case you are looking for an alternative source for market data, you can use [Quandl](#) for the same.*

***Disclaimer:*** *All investments and trading in the stock market involve risk. Any decisions to place trades in the financial markets, including trading in stock or options or other financial instruments is a personal decision that should only be made after thorough research, including a personal risk and financial assessment and the engagement of professional assistance to the extent you believe necessary. The trading strategies or related information mentioned in this article is for informational purposes only.*

## Download Python Code

- Portfolio Optimization Using Monte Carlo Simulation - Python Code



**Download**



quantinsti | NSE Academy | MCX METAL & ENERGY Trade with Trust

Online Courses on

# Algorithmic Trading for Everyone

**START FOR FREE**

Share Article:



<https://blog.quantinsti.com/portfolio-optimization-maximum-return-risk-ratio-python/>



Jan 29, 2018

## Portfolio Analysis: Calculating Risk and Returns

Apr 27, 2018

## Calculating Value At Risk In Excel & Python

### 3 Comments

Type Comment Here (at least 3 chars)

Submit

Lop 3D BanPHHS • Last Year

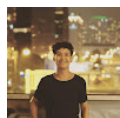
[Our cookie policy](#) ▲





Greate Method !

Reply



Tanapat Kamsaiin • Last Year

Cannot download the python code.

Reply



QuantInsti • Last Year

Hi! Thank you for bringing it to our notice. We have fixed the same.

Reply

Search in the blog...



## Career Growth

Industry

Jobs & Skills

Trading Desk Setup

Success Stories

[Our cookie policy](#) ▲ [Projects](#)

## Tutorials

Automated Trading

Excel & R For Trading

Forex & Crypto Trading

Machine Learning

Maths and Econometrics

Mean Reversion & Statistical Arbitrage

Momentum Trading

More Trading Strategies

Options Trading

Portfolio & Risk Management

Python For Trading

Sentiment Trading

Technical Indicators



## About

QuantInsti

Quantra

Blueshift

## Events & Announcements

Announcements

Webinars

Our cookie policy ▲ 



[Contact Us](#)

[Privacy Policy](#)

[Blog Contribution](#)

Copyright © 2021 QuantInsti.com All Rights Reserved.

[Our cookie policy](#) ▲

