# Mean Variance Optimization (MVO)

Alex Ramirez

January 1, 2023

**Abstract**

Mean Variance Optimization is a form of technical analysis for informed portfolio asset allocation that attempts to maximize expected returns while minimizing the risk. The theory of this analysis can be applied to any portfolio if each asset in the portfolio has a history of it's value such as a stock price. This theory expect to return higher expected percent return at the same or lower level of risk when applied to an investment portfolio. The mean variance optimization model uses historical data on asset value to estimate average expected return and variance of different assets to construct an optimal return vs risk portfolio. This analysis model is is commonly used in finance and has been shown to be effective in maintaining portfolio asset allocation for optimal return and minimized risk.

## Contents



## Appendices

## List of Tables



## List of Figures

## Section 1: Introduction to MVO

The purpose of Mean Variance Optimization is to increase portfolio return while decreasing portfolio volatility. As a result, the Sharpe ratio will be maximized. Mean Variance Optimization or Modern Portfolio theory is a form of technical analysis that attempts to optimally maximize an investors portfolio return while minimizing it's risk. As an investor, could you be making more money? What is the associated risk? How much risk is too much? Mean Variance Optimization attempts to answer these questions based on historical investment data. Using Excel and the programming language Python, I apply the concept of mean variance optimization to a simulated stock portfolio in order to better understand this theory. In this paper, I use the words stock and asset interchangeably.

### 1.1 Opportunity cost and Technical analysis

Opportunity cost is "the loss of potential gain from other alternatives when one alternative is chosen" (Oxford English Dictionary). In an investment portfolio, the loss of choosing one stock over another is opportunity cost. In order to quantify the potential loss or gain between assets, technical analysis is used to examine the value of stocks. Technical analysis is the evaluation of an assets past trends through a study of past market data employed to evaluate possible return and risk for a given stock. Technical analysis assumes stock market price properly represents its value. The efficient frontier model allows investors to shape their portfolio according to their personal risk tolerance and investment objectives.By using Mean Variance Optimization, investors are able to make more informed decisions about how to allocate their capital in order to maximize their returns (money made) while minimizing their exposure to unnecessary risk.

### 1.2 Theory: MVO Graph

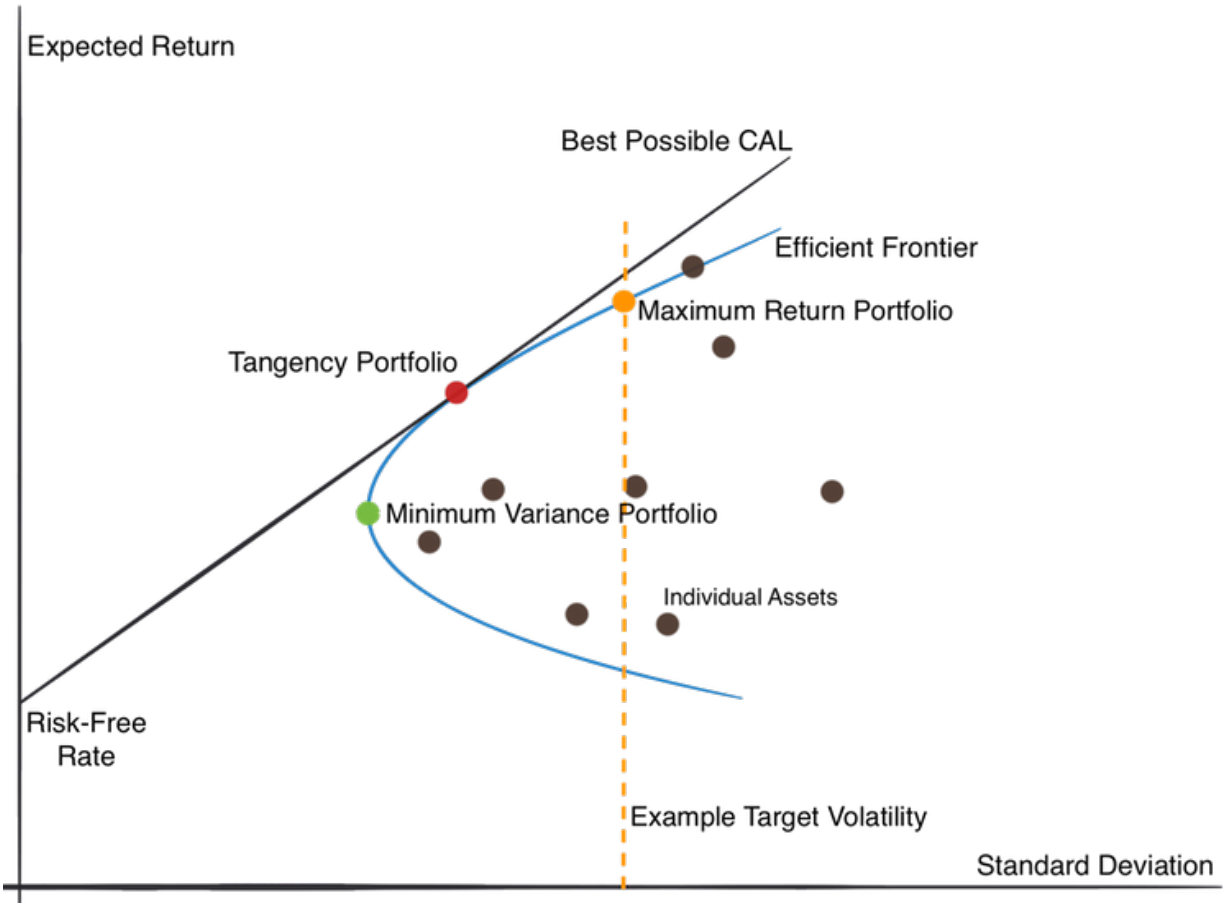



Figure 1: MVO Theory

- **Blue Curve**: Efficient Frontier - All portfolios that assume the least amount of risk for a given portfolio percent return. On a horizontal line, this is the leftmost point.
- **Gray Points**: Non-Optimal Portfolios - These portfolios have a return that can be reproduced with decreased amount of risk. On a horizontal line this is any point that is not the efficient frontier (leftmost point).
- **Green Point**: Minimum Variance Portfolio - This portfolio assumes the absolute least amount of possible expected risk and returns the highest for this risk.
- **Red Point**: Tangency Portfolio (highest Sharpe ratio) - This portfolio is the optimal portfolio, optimally balancing return and risk. This portfolio maximizes the Sharpe ratio of the portfolio.
- **Yellow Point**: Maximum Return Portfolio - This portfolio has the highest possible return and assumes the least risk given this.

### 1.3 Methodology

1. Get historical stock data
2. Calculate % change
3. Calculate Average return per asset
4. Calculate Average variance
5. Calculate Covariance Matrix
6. Not Necessary: Correlation Matrix
7. Portfolio Generation: Weights, Return, St. Deviation, Sharpe Ratio, Efficient Frontier
   (a) Generate data for equally weighted portfolio
   (b) Generate data for ‘n‘ number random weighted portfolios
   (c) Generate data for optimal tangency, minimum risk, and maximum return portfolios
8. Plotting
9. Analyze new optimized portfolio

## Section 2: Mean Variance Optimization in Excel

### 2.1 Getting Stock Historical Close Price Data

The STOCKHISTORY function retrieves historical data about a financial instrument such as a stock ticker and loads it as an array, which will spil beyond just the current cell. The stock history function takes arguments for the stock ticker, start date, end date, interval of data, headers, and property. In order to reverse the date range of the array, I encapsulate the Stockhistory formula with a sort instructed to reverse the data by date range. In doing so, start date to end date becomes end date to start date.

`=STOCKHISTORY(ticker,startDate, endDate, interval, headers, property)`

`=SORT(array), index, order)` to become `=SORT(STOCKHISTORY())` Passing all required arguments, headers = ”show” is used for the first column so that an index of dates is displayed while headers = ”none” is used for the remaining tickers so as not to repeat the display of same dates.

Excel Stock History Formula with headers (dates):

`=SORT(STOCKHISTORY(C2,DATE(2012,1,1), DATE(2022,3,1), 2, 0, 0,1),1,-1)`

interval = month, headers = show, property = ’close’

Excel Stock History Formula with no headers (no dates):

`=SORT(STOCKHISTORY(D2,DATE(2012,1,1), DATE(2022,3,1), 2, 0,1),1,-1)`

interval = month, headers = none, property = ’close’

Applying these formulas in excel would be organized as followed:

| **hist with date** | spill | **hist** | **hist** | **hist** |
|---|---|---|---|---|
| spill | spill | spill | spill | spill |

Example of returned historical close price data:

Table 1: Combined Excel Historical Close Price

| DATE | NKE | MSFT | DIS | COST |
|---|---|---|---|---|
| **3/1/22** | $134.56 | $336.32 | $189.04 | $575.85 |
| **2/1/22** | $136.55 | $331.62 | $186.02 | $567.70 |
| **1/1/22** | $148.07 | $330.59 | $184.52 | $539.38 |
| **...** | ... | ... | ... | ... |
| **3/1/12** | $27.11 | $27.45 | $43.11 | $80.37 |
| **2/1/12** | $26.98 | $26.71 | $41.99 | $80.06 |
| **1/1/12** | $26.00 | $26.62 | $38.90 | $76.54 |

### 2.2 Excel Historical Percent Change

See Appendix A: Formulas for percent change

Excel Formula: `=(C3-C4)/C3`

This formula is used on each cell to generate the table below.

Table 2: Excel Historical Percent Change

| NKE | MSFT | DIS | COST |
|---|---|---|---|
| -1.4789% | 1.3975% | 1.5975% | 1.4153% |
| -8.4365% | 0.3106% | 0.8064% | 4.9886% |
| -12.5616% | 5.9318% | 1.7451% | 3.7321% |
| ... | ... | ... | ... |
| 3.0661% | 1.2590% | 1.5304% | 2.0299% |
| 0.4795% | 2.6969% | 2.5980% | 0.3821% |
| 3.6416% | 0.3546% | 7.3589% | 4.4038% |

By calculating percent change on a month to month basis, we are able to view the amount in % that this stock changes on a monthly basis. A positive % is an increase in stock price from month to month while a negative % displays a decrease in stock price from month to month.

### 2.3 Average Return and Variance

**Average Return** See Appendix A: Formulas for mean return

Excel Formula: `=AVERAGE(H3:H124)` The average * 12 is taken for each stock's percent change column

Average monthly return is the average of the % change column for each asset because that is the % change in stock price which is equivalent to the investment return in %.

**Average Variance** See Appendix A: Formulas for mean variance

Excel Formula: `=VAR(H3:H124)` variance for each percent change column

Average monthly variance is the variance of the % change column for each asset which is a measure of dispersion for each asset.

**Yearly Return and Variance**: Multiply * 12

Yearly Return = Monthly Return * 12 and Yearly Variance = Monthly Variance * 12 because my historical close price interval is monthly over a period of 10 years while I want the yearly mean instead of monthly.

To gain access to a yearly statistic, we multiply the monthly return or variance by 12 because there are 12 months in a year. If I was analyzing weekly data, I would multiply the return or variance by 52 to account for the 52 weeks in a year. If I were analyzing daily data attempting to gain monthly return or variance, I must multiply the daily mean by a fixed time factor of 30.437 for the average 30.437 days in a month.

Table 3: Excel Return and Variance

| | NKE | MSFT | DIS | COST |
|---|---|---|---|---|
| **Avg Monthly Ret** | 1.139% | 2.040% | 1.274% | 1.627% |
| **Monthly Variance** | 0.406% | 0.035% | 0.026% | 0.027% |
| **Avg Annual Ret** | 13.665% | 24.474% | 15.287% | 19.519% |
| **Annual Variance** | 4.870% | 0.422% | 0.318% | 0.325% |

**Reformat Annual Variance** See Appendix A: Formulas for Matrix Transposition

For later ease of use, we reformat the annual variance to vertical form by transposing the horizontal average annual return row into a vertical column as so.

$$\begin{bmatrix} E(r_1) & \dots & E(r_n) \end{bmatrix} \Rightarrow \begin{bmatrix} E(r_1) \\ \vdots \\ E(r_n) \end{bmatrix}$$

### 2.4 Covariance Matrix

See Appendix A: Formulas for covariance and yearly covariance

yrCov Excel Formula: `=COVAR(H3:H124,H3:H124)*12`

$$\text{Covar. Mat.} = \begin{bmatrix} yrCov(A,A) & yrCov(A,B) & \dots & yrCov(A,Z) \\ yrCov(B,A) & yrCov(B,B) & \dots & yrCov(B,Z) \\ \vdots & \vdots & \ddots & \vdots \\ yrCov(Z,A) & yrCov(Z,B) & & yrCov(Z,Z) \end{bmatrix}$$

where input $A$, $B$, $Z$, represent each asset from $A$ to $Z$ as their % change columns. While variance

Table 4: Excel Covariance Matrix

| Covariance Matrix | NKE | MSFT | DIS | COST |
|---|---|---|---|---|
| **NKE** | 0.04830511 | 0.000329414 | 0.000100843 | 0.000608578 |
| **MSFT** | 0.000329414 | 0.004181378 | -0.000365195 | 9.64809E-06 |
| **DIS** | 0.000100843 | -0.000365195 | 0.003151954 | 0.000377709 |
| **COST** | 0.000608578 | 9.64809E-06 | 0.000377709 | 0.003223492 |

reveals the variance of a single variable, covariance is the variance between two variables. Because of the nature of the covariance Matrix, it will always be a symmetric square matrix where the blue diagonal is simply the variance because it represents the covariance between an item and itself.

### 2.5 Correlation Matrix

The correlation matrix is not necessary for Mean Variance Optimization and the calculation of the optimal tangency portfolio; however, it can be useful to see the correlation between the performance of different stocks in order to better understand market correlation between assets.

Correlation is the statistical measure that expresses the extent to which two variables are linearly related. The correlation coefficient is measured on a scale from 1 to -1.

See Appendix A: Formulas for correlation

## 2.6 Portfolio Generation

### 2.6.1 Asset Weight Rule

See Appendix A: Formulas for Asset Weight Constraint

Each asset weight must be $\geq 0\%$ and $\leq 100\%$ of the portfolio while the sum of the asset weights must be = 100% of the portfolio asset weights.

### 2.6.2 Expected Portfolio Return

See Appendix A: Formulas for Expected Portfolio Return

Excel Formula: `=MMULT(TRANSPOSE(N13:N16),N3:N6)`

### 2.6.3 Standard Deviation (Risk)

See Appendix A: Formulas for Standard Deviation

$$\sigma = \sqrt{\text{Weights Transposed} * \text{Covariance Matrix} * \text{Weights}}$$

Excel Formula: `=SQRT(MMULT(MMULT(TRANSPOSE(N13:N16),Q3:T6),N13:N16))`

### 2.6.4 Sharpe Ratio

Excel Formula: `=(N19-N8)/N20`

## 2.7 Equal and Optimal Portfolios

Optimal Asset Weights are solved for by ”Excel Solver” tool set to maximize Sharpe Ratio by changing weights of portfolio with the constraint of sum(weights) must = 1 as per the Asset Weight Rules (See Appendix A: Formulas Asset Weight Rules).

Table 5: Equal and Optimal Portfolios

| Portfolio | Eq. Weight of assets | Opt. Weight of assets |
|---|---|---|
| NKE | 25.000% | 1.089% |
| MSFT | 25.000% | 35.950% |
| DIS | 25.000% | 21.521% |
| COST | 25.000% | 41.439% |
| Sum | 1 | 1 |
| | | |
| Expected Return | 18.236% | 20.326% |
| Standard Deviation | 6.174% | 3.561% |
| Sharpe Ratio | 2.630 | 5.147 |

### 2.8 Generating 16,000 Random Trials

See Appendix A: Formulas for calculating random trial weights. Excel Formula: `=RANDARRAY(4,1)`

Excel Formula: `=O52/$O$56`

Table 6: Excel Randomly Weighted Portfolio

| Rand Weight Portfolio | Weights of equities (randomized) | randarray |
|---|---|---|
| NKE | 35.226% | 67.910% |
| MSFT | 13.727% | 26.462% |
| DIS | 20.701% | 39.908% |
| COST | 30.346% | 58.502% |
| Sum | 1 | 192.782% |
| | | |
| Expected Return | 17.261% | |
| Standard Deviation | 8.191% | |
| Sharpe Ratio | 2.107 | |

The Table 7: Excel Randomly Weighted Portfolio is applied to 16,0000 times returning 16,000 independent and random risk and and return values. These values are kept to plot on a scatter plot, creating the efficient frontier.

| 16K Trials | Risk | Return |
|---|---|---|
| **1** | 5.07% | 16.72% |
| **2** | 8.63% | 16.71% |
| **3** | 4.55% | 18.06% |
| **...** | ... | ... |
| **15988** | 8.69% | 17.27% |
| **15999** | 6.39% | 17.55% |
| **16000** | 11.01% | 16.21% |

## 2.9 Plotting

In order to plot the efficient frontier,

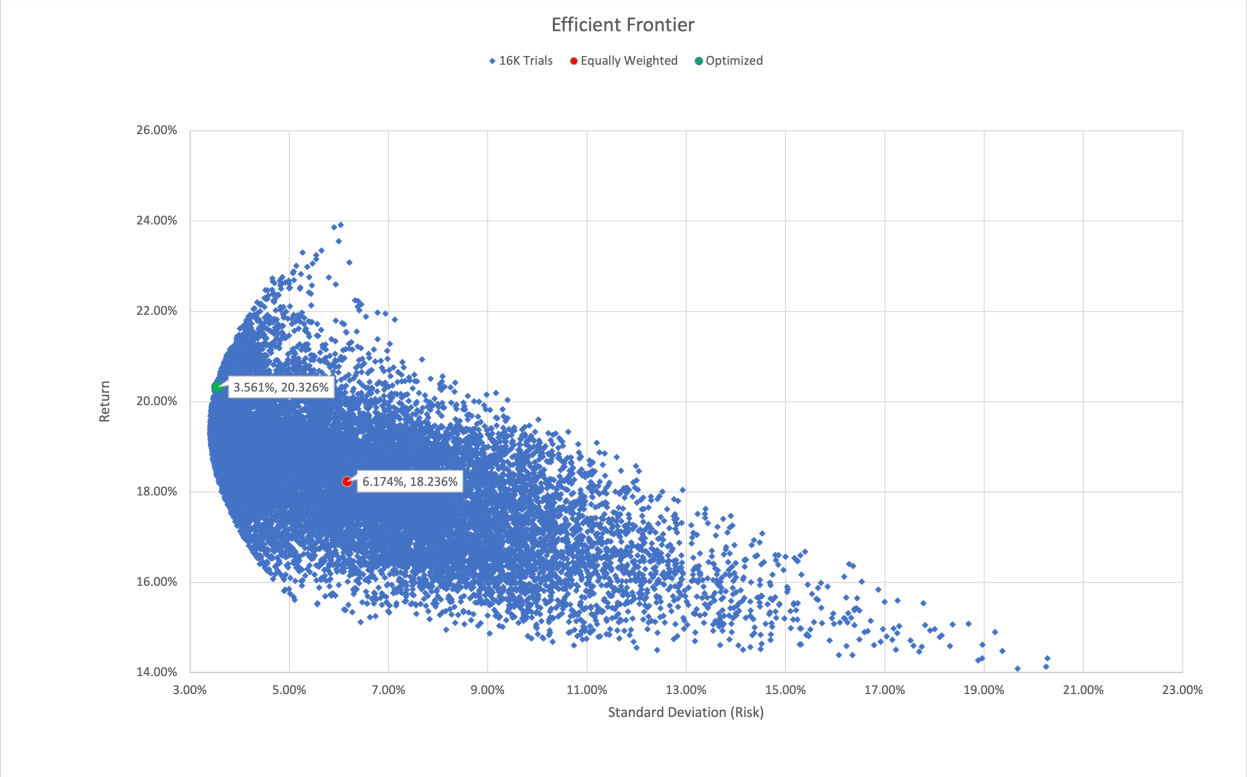



Figure 2: Excel Efficient Frontier Scatter Plot

### 2.9.1 Excel Plot Analysis

Figure 2, the Excel Efficient Frontier Scatter Plot, displays the Standard Deviation of the portfolio (Risk) along the x-axis and the Expected Yearly Return along the y-axis. There are 16,000 blue


(a) Equally Weighted Bar Chart

(b) Optimal Weighted Bar Chart

(c) Equally Weighted Pie Chart

(d) Optimal Weighted Pie Chart

Figure 3: Excel Chart Comparison of Equal and Optimal Portfolios.

points, each representing a portfolio with random weight of assets. By plotting the risk and return for each possible portfolio, the plot of the efficient frontier is formed.

The red point represents the risk and return for a portfolio where each asset is equally weighted. In figure 2.a, the bar chart, the standard deviation (or risk) for an equally weighted portfolio is anticipated to be 6.174% annually and the annual return is expected to be 18.236%. The four bars and pie graph, figure 2.c, display the equal weights for a portfolio containing 4 assets. Since there are four assets in the portfolio, each asset will make up 25% of the portfolio.

The green point represents the risk and return for a portfolio where each asset is weighted optimally according to the theory of Mean Variance Optimization. In figure 2.b, the bar chart, the standard deviation (or risk) for an optimally weighted portfolio is anticipated to be 3.561% annually and hte annual return is expected to be 20.326%. The four bars and pie graph, figure 2.d, display the optimal weights for a portfolio containing these specific 4 assets. Of all possible portfolios, the optimal weighted portfolio maximizes the Sharpe ratio.

By changing the weights from the equally weighted to optimally weighted portfolio, the annual anticipated risk decreases by an approximate 2.613% and the annual expected return increases by an approximate 2.090%. By following MVO and changing the asset weight allocation of a portfolio, we optimize the risk and return for our investment portfolio. In this case it is about 2% increase in return with about 2% decrease in risk while in other portfolios this margin of change may be greater or lesser.

Table 7: Excel Comparing Portfolios

| | Equally Weighted | Optimally Weighted | Difference |
|---|---|---|---|
| **Risk** | 6.174% | 3.561% | -2.613% |
| **Return** | 18.236% | 20.326% | +2.090% |

With the Sharpe ratio maximized, the return was increased and risk decreased; both great outcomes for an investment portfolio.

## Section 3: Mean Variance Optimization in Python

First, all required packages must be downloaded and installed within our working python environment. Each code snippet then becomes a 'code' section in a Jupyter local jupyter notebook. See Appendix C: Installing and Importing Python Packages

### 3.1 Defining constant variables

These defined variables are human set values that give the program necessary information to run. Here I define the risk free rate, the number of trial portfolios to generate, the tickers to use for MVO, and a start and end date. The larger the number for trial_n, the higher the probability of improved accuracy because there are more random portfolios generated for the selection of the Max Sharpe Portfolio.

```
risk_free_rate = 0.03
trial_n = 100000
tickers = ['NKE', 'MSFT', 'DIS', 'COST']
start = '2012-1-1'
end = '2022-1-1'
```

currently the risk free rate is $\approx 3\%$ or $0.03$ (Jan 2022) See Appendix A: Formulas for risk free rate formula

### 3.2 Downloading Historical Close Data for Stocks

```
df_close = yf.download(tickers, start, end, interval='1mo')['Close']
# output [**********100%**********]  4 of 4 completed
df_close.to_csv('./data/df_close.csv')
```

Using yfinance, I download the historical data for our stock ticker list from start date to end date on the interval of 1 month and save the data to a .csv file called "df_close.csv" in the folder called "data" within the current program directory. I download the data, rather than just storing it to a variable so that, later, I can implement a read function that first checks for stored data and does not repeatedly download the same data from yfinance.

Next, the data must be checked to see if it is returned correctly. Printing the shape Attribute returns (120,4). The returned dataframe should be 120 rows because it returns a monthly interval for a period of 10 years. 12 mo * 10 yrs = 120 rows. Should be 4 colmns because I have formatted a dataframe to just the 'Close' column for 4 different tickers. Therefore I know that the returned data set is at least the right size. Next, the functions head() and tail() are used to check the top and bottom rows of data, showing us that the order of dates and returned data looks correct. The returned close price data appears to be valid and what I requested using the yfinance download call.

See Appendix D: Dataframe Helper Attributes and Functions for python implementation.

**Checking for Null Values**

Sometimes, even API's like yahoo finance return data sets with missing values. In this case, there are no (0) null values in the close price dataframe. However, if there were any null values, this table would display the count of null values for each column in the data and then fill these values using the dataframe forward and backward fill functions. In this case, since there are no missing values, the if statement would be skipped and the fillna function would not be performed on the dataframe.

```
print(df_close.isnull().sum())
if df_close.isnull().values.any():
    df_close.fillna(method='ffill', inplace = True) #front fills
    df_close.fillna(method='bfill', inplace = True) #back fills
df_close.isnull().sum()
```

Table 8: Python Null Values

| | 0 |
|---|---|
| COST | 0 |
| DIS | 0 |
| MSFT | 0 |
| NKE | 0 |

**Plotting Historical Close Price**

When plotting line graphs, I call the plot_line function. See Appendix E: Python Plotting for plotting line graphs.

```
plot_line(df_close, 'Date', 'Stock Price', 'Close Price')
```

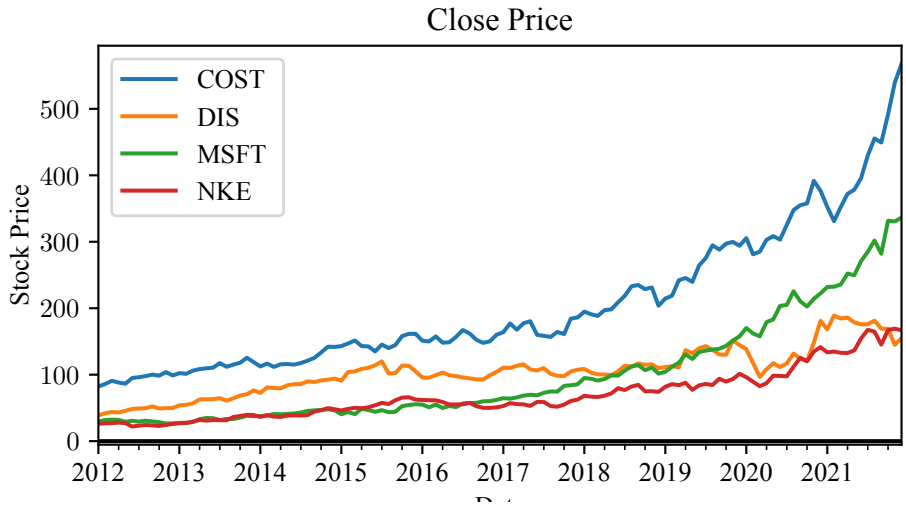



Figure 4: `Matplotlib` Historical Close Price


Figure 5: `Plotly` Historical Close Price PNG

Plotly graphs are interactive Ib-based graphs built on js and html. Meanwhile, Latex PDF's and PDF's in general are static. As a result, Plotly Graphs must be inserted by converting the Plotly graph to a static form such as JPG, PNG, PDF, etc (in this case PNG), forcing them to lose their

functionality and interactability. These plotly pdf static image representations are not accurate depictions of the graphs. To view these graphs, see this program's Jupyter Notebook file, "mvo.ipynb" at github.com/Terraform05/MVO

**I will be omitting the majority of the remaining Plotly Graphs for conciseness**

### 3.3 Historical Percent Change

See Appendix A: Formulas for percent change. Historical Percent Change display the intervals returns, or in this case monthly return, for each stock over the time period, which in this case is 10 years. Historical Percent Change allow us to see the percent change that the stock price increases or decreases by on a monthly basis. Because the percent change function calculates percent change of the row compared to the previous row, there will be row 1 past the index extremity that will try to be accessed and dividing by this non-existent row will return a NaN value filled row using this function. To combat this, I remove the NaN value row which is the first row using '.iloc[1:]'.

```
df_pct_change = df_close.pct_change().iloc[1:]
#.iloc[1:] gets rid of NaN value row because of indexing
df_pct_change.head()
```

Table 9: Python Historical Percent Change

| Date | COST | DIS | MSFT | NKE |
|---|---|---|---|---|
| 2012-02-01 00:00:00-05:00 | 0.046068 | 0.079434 | 0.074839 | 0.037792 |
| 2012-03-01 00:00:00-05:00 | 0.055078 | 0.042629 | 0.016383 | 0.004818 |
| 2012-04-01 00:00:00-04:00 | -0.028855 | -0.015304 | -0.007439 | 0.031630 |
| 2012-05-01 00:00:00-04:00 | -0.020299 | 0.060311 | -0.088382 | -0.032985 |
| 2012-06-01 00:00:00-04:00 | 0.099664 | 0.061037 | 0.047962 | -0.188575 |

**Describe Percent Change**

Describe can be used again to make sure that the dataframe is 119 rows, length-1. Additional intriguing information about the monthly percent change can be found. See Appendix D: Dataframe Helper Attributes and Functions for the describe function.

### Plotting Historical Percent Change

```
plot_line(df_pct_change, 'Date', 'Percent', 'Percent Change (monthly return)')
```

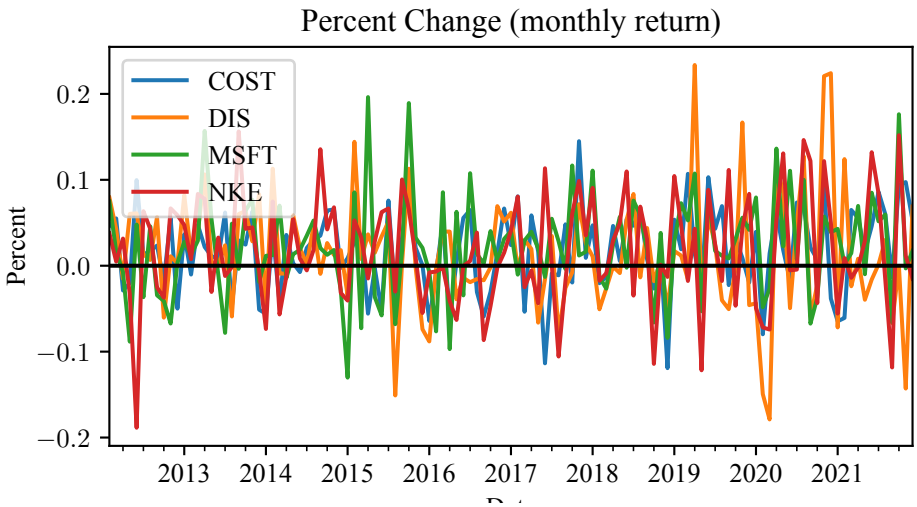



Figure 6: `Matplotlib` Percent Change (monthly return)

Here I call the plot_line function created earlier, using the percent change dataframe and titles instead to create a monthly return plot.

## 3.4 Average Return and Variance

**Average Return** See Appendix A: Formulas for mean return

```
mean_return = df_pct_change.mean() * 12
mean_return
```

Table 10: Python Mean Percent Change

| | 0 |
|---|---|
| COST | 0.210163 |
| DIS | 0.166885 |
| MSFT | 0.267666 |
| NKE | 0.212343 |

This returns the average monthly percent change and I need the yearly return instead. To get the yearly return, I multiply the monthly return by 12 for the 12 months in a year.

**Average Variance** See Appendix A: Formulas for mean variance

```
var_risk = df_pct_change.var() * 12
var_risk
```

Table 11: Python Mean Variance

| | 0 |
|---|---|
| COST | 0.027995 |
| DIS | 0.055262 |
| MSFT | 0.041114 |
| NKE | 0.047685 |

Variance is a numerical value quantifying how far values of a data set are from their mean. Yearly variance for each stock column is the monthly variance * 12.

### 3.5 Covariance Matrix

See Appendix A: Formulas for covariance and yearly covariance

```
cov_mat = df_pct_change.cov() * 12
cov_mat
```

Table 12: Python Covariance Matrix

| | COST | DIS | MSFT | NKE |
|---|---|---|---|---|
| COST | 0.027995 | 0.010468 | 0.010946 | 0.011377 |
| DIS | 0.010468 | 0.055262 | 0.016682 | 0.019448 |
| MSFT | 0.010946 | 0.016682 | 0.041114 | 0.012376 |
| NKE | 0.011377 | 0.019448 | 0.012376 | 0.047685 |

While variance reveals the variance of a single variable, covariance is the variance between two variables.

### 3.6 Correlation Matrix

The correlation matrix is not necessary for Mean Variance Optimization and the calculation of the optimal tangency portfolio; however, it can be useful to see the correlation between the performance of different stocks in order to better understand market correlation between assets.

Correlation is the statistical measure that expresses the extent to which two variables are linearly related. The correlation coefficient is measured on a scale from 1 to -1.

See Appendix A: Formulas for correlation

```
corr_mat = df_pct_change.corr()
corr_mat
```

returns

Table 13: Python Correlation Matrix

| | COST | DIS | MSFT | NKE |
|---|---|---|---|---|
| COST | 1.000000 | 0.266133 | 0.322651 | 0.311372 |
| DIS | 0.266133 | 1.000000 | 0.349985 | 0.378852 |
| MSFT | 0.322651 | 0.349985 | 1.000000 | 0.279516 |
| NKE | 0.311372 | 0.378852 | 0.279516 | 1.000000 |

**Plotting Correlation Matrix** When plotting Matrices, I call the plot_matrix function. See Appendix E: Python Plotting for plotting Matrices.

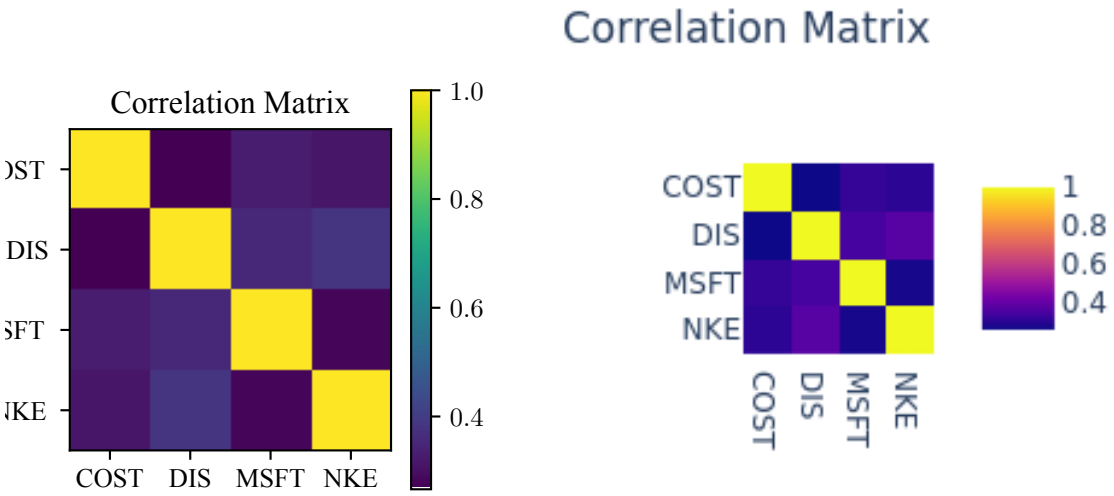
### 3.7 Equally Weighted Portfolio Generation

#### Equal Asset Weight

See Appendix A: Formulas for Asset Weight Constraint

Since all the weights will be equal to each other and the sum of the weights must be equal to 1, the weight will be 1 divided by the number of assets in the portfolio. Then, a nested for loop is used to create the list of equal weights for the portfolio.

```
weight = 1/len(tickers)
eq_weights = [weight for i in tickers]
```




(a) `Matplotlib` Correlation Matrix (b) `Plotly` Correlation Matrix

Figure 7: Python Correlation Matrix

```
eq_weights
# output [0.25, 0.25, 0.25, 0.25]
```

**Expected Portfolio Return**

See Appendix A: Formulas for Expected Portfolio Return

The eq_weights list is a 1x4 matrix and the mean_return list is a 4x1 matrix, so @, the python matrix multiplication binary operator, can be used to multiply the matrices and returns the double value of the 1x1 resultant matrix.

```
eq_return = eq_weights @ (mean_return)
eq_return
# output 0.21426419041415923
```

**Standard Deviation (Risk)**

See Appendix A: Formulas for Standard Deviation

$$\sigma = \sqrt{\text{Weights Transposed} * \text{Covariance Matrix} * \text{Weights}}$$

**Equal Portfolio Risk**

The standard deviation formula is applied using the lists created prior to output the standard deviation or risk of the portfolio.

```
eq_risk = ((eq_weights @ cov_mat) @ eq_weights)**(1/2)
eq_risk
# output 0.14462260526809836
```

**Sharpe Ratio**

See Appendix A: Formulas for Sharpe Ratio The Sharpe Ratio formula is applied using the lists created prior to output the standard deviation or risk of the portfolio.

```
eq_sharpe = (eq_return - risk_free_rate)/eq_risk
eq_sharpe
# output = 1.2741036580870198
```

Once the optimal portfolio is created, the statistics of the equally weighted portfolio will be compared with that of the optimal portfolio. The Sharpe ratio of the optimal portfolio should be greater than that of this equally weighted portfolio, increasing return while decreasing its risk.

### 3.8 Randomly weighted Portfolio Generation

t_weight is a list of random numbers, with a list length equal to the number of assets in the portfolio.Then the weights are all divided by the sum of the random numbers to generate a random weight where all are equal to one for each stock in our portfolio. With these weights, we apply the portfolio return formula, portfolio risk formula, and shape ratio formula to this portfolio. This process is done, trial_n times, or 100,000 times to generate return, risk, and Sharpe for 100,000 randomly generated portfolios.

```
efficient_frontier_data = []
for trial in tqdm(range(trial_n)):
    t_weight = np.random.random(len(tickers, ))
    t_weights = t_weight/sum(t_weight)
    t_return = t_weights @ (mean_return)
    t_risk = ((t_weights @ cov_mat) @ t_weights)**(1/2)
    t_sharpe = (t_return - risk_free_rate)/t_risk
    efficient_frontier_data.append([t_weights, t_risk, t_return, t_sharpe])

efficient_frontier_data = pd.DataFrame(efficient_frontier_data, columns=['weights', 'Risk', '
    Return', 'Sharpe'])
efficient_frontier_data

# tqdm elapsed for-loop time
# 100%[||||||||||||] 100000/100000 [00:11<00:00, 8463.76it/s]
```

For each randomly generated portfolio, weights, return, risk, and sharpe are recorded so that they can be searched through for a max sharpe, optimal portfolio. Additionally, the non-optimal, but minimal risk, portfolio and non-optimal, but maximum return, portfolio can be found by filtering return and risk, rather than Sharpe.

Table 14: Python Randomly Weighted Portfolios

| | weights | Risk | Return | Sharpe |
|---|---|---|---|---|
| 0 | [0.10444119899962606, 0.14225468491340973, 0.3... | 0.154951 | 0.223343 | 1.247774 |
| 1 | [0.30958524600219867, 0.26589169957978853, 0.1... | 0.144312 | 0.208568 | 1.237378 |
| 2 | [0.09235399837268944, 0.002890823487057892, 0.... | 0.167824 | 0.227946 | 1.179485 |
| 3 | [0.05031144080854111, 0.3606072389018749, 0.05... | 0.175373 | 0.199126 | 0.964377 |
| 4 | [0.1978628961229039, 0.23699117139451767, 0.29... | 0.146759 | 0.217322 | 1.276388 |
| ... | ... | ... | ... | ... |
| 99995 | [0.3895478511869071, 0.14657799671244848, 0.22... | 0.138843 | 0.217116 | 1.347676 |
| 99996 | [0.003860538459429772, 0.36439329393335324, 0.... | 0.167874 | 0.208146 | 1.061188 |
| 99997 | [0.33000520563029684, 0.3290010474791345, 0.31... | 0.147717 | 0.213860 | 1.244675 |
| 99998 | [0.4257732373184667, 0.25385051163218997, 0.00... | 0.147887 | 0.200315 | 1.151658 |
| 99999 | [0.40113471427018366, 0.14945449118690274, 0.2... | 0.138568 | 0.220947 | 1.378004 |

### 3.9 Optimal Portfolio Selection

#### Optimal Portfolio (Max Sharpe Ratio)

See Appendix A: Formulas for Sharpe Ratio By finding the row of data for the portfolio with the maximum Sharpe in the randomly generated trials column 'Sharpe', the weights, return, risk, and Sharpe of the optimal portfolio are found.

```
i_max_sharpe = efficient_frontier_data['Sharpe'].idxmax()
max_sharpe = efficient_frontier_data.iloc[i_max_sharpe, :]
max_sharpe
```

| | Weights | Risk | Return | Sharpe |
|---|---|---|---|---|
| 21264 | [0.4096023854828801, 0.002275271479076909, 0.4... | 0.142449 | 0.233813 | 1.430783 |

While the Minimum Risk and Maximum Return portfolios are not optimal, they are interesting figures using different weight allocation and can be found using the same searching methods as for the optimally weighted portfolio. See Appendix F: Minimum Risk and Maximum Return Portfolios for these additional portfolios.

### 3.10 Plotting Efficient Frontier

The coloring of the Efficient frontier in Figure 8 is according to the Sharpe ratio of the portfolio. The yellow coloration is a higher Sharpe ratio. The green point represents the portfolio with the highest Sharpe ratio while the blue point represents the portfolio with the minimum risk and the red point represents the portfolio with the maximum return. The optimal portfolio is the green point under yellow coloration. While an equally weighted portfolio has a Sharpe ratio of 1.274, the optimally weighted portfolio has a Sharpe ratio of 1.431. Similarly, the Sharpe ratio of the optimal portfolio in green is higher than the Sharpe ratio of all other points.

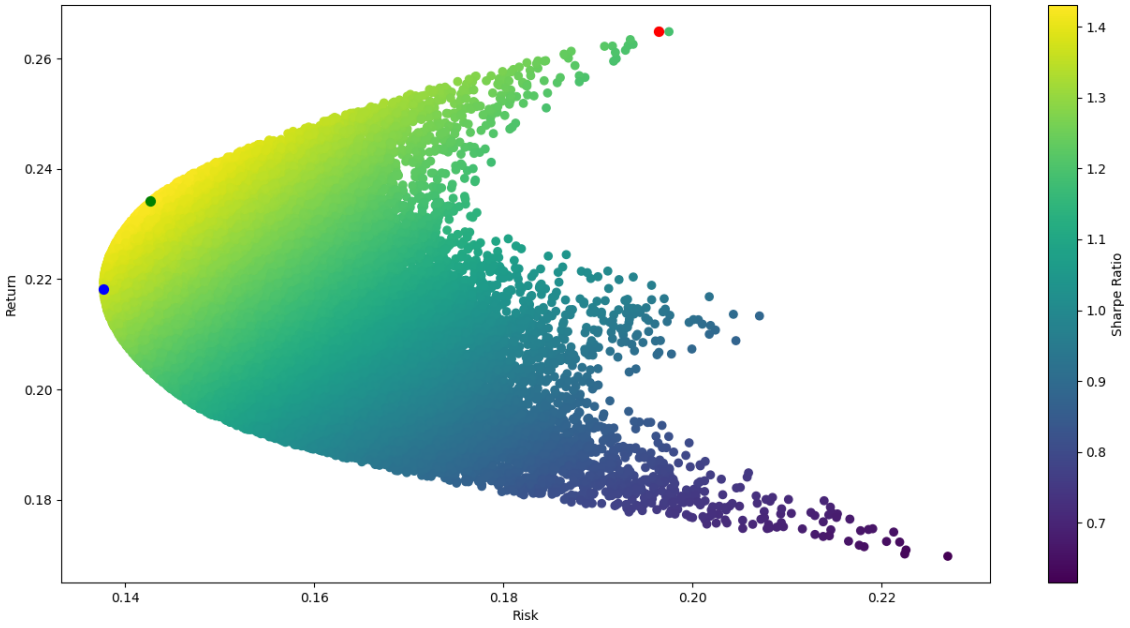



Figure 8: `Matplotlib` Efficient Frontier

### 3.11 Optimal, Equal, Min Risk, and Max Return Portfolios

Table 15: Python Comparing Portfolios

| Portfolio | Optimal | Equal | Min Risk | Max Ret |
|---|---|---|---|---|
| COST | 40.960% | 25% | 46.686% | 1.968% |
| DIS | 0.228% | 25% | 12.728% | 0.129% |
| MSFT | 40.609% | 25% | 24.300% | 95.443% |
| NKE | 18.203% | 25% | 16.286% | 2.460% |
| Sum | 1 | 1 | 1 | 1 |
| | | | | |
| Expected Return | 23.381% | 21.426% | 21.898% | 26.504% |
| Standard Deviation | 14.245% | 14.462% | 13.771% | 19.6313% |
| Sharpe Ratio | 1.431 | 1.274 | 1.372 | 1.197 |

While the efficient frontier in Figure8 represents the data and these points, this table shows specifics about the individual portfolios. Starting with the maximum return portfolio, we can see that Microsoft has the highest return over the time frame studied. However, with that high return, also comes a higher risk. This portfolio has one of the highest risk percentages out of all trial port-

folios. The relationship of these portfolios are demonstrated by the difference in Sharpe Ratios for each portfolio. When comparing the Minimum Risk and Equally weighted portfolios, the minimum risk has a greater return and lower risk, thus having a higher Sharpe ratio than the equally weighted portfolio. When comparing the optimal portfolio to the others, the optimal portfolio has a return that's higher than the both equal and minimum risk while lower than maximum return. This can be visualized by the y values in the efficient frontier Figure8. In the same comparison, the optimal portfolio has a risk that's higher than the Minimum risk portfolio, but lower than the Equally weighted portfolio and maximum return portfolio. This can be visualized by the x values in the efficient frontier Figure8. This ratio of between return and risk is greatest for the Optimal Portfolio. By simply changing the weight of an asset's bearing on a portfolio, meaning diversification of current assets to specific values, the performance of the portfolio will increase.

The percent difference improvement in return between an equally weighted portfolio and an optimally weighted portfolio was **8.726%** The percent difference decrease in risk between an equally weighted portfolio and an optimally weighted portfolio was **1.512%**

For this investment portfolio of 4 stocks, the application of Modern Portfolio Theory and Mean Variance Optimization increased yearly return while decreasing the yearly associated risk. This is the goal of Mean Variance Optimization, demonstrated here in both Excel and Python.

For full code and graphs, look at the "MVO.ipynb" notebook in my public github repository found at: https://github.com/Terraform05/MVO

**Appendix A: Formulas**

$$\%Change = \frac{x_1 - x_2}{x_1}$$

where $x_1$ = initial value and $x_2$ = final value

---

$$Mean = \frac{\sum_{i=1}^{n}(x_i)}{n}$$

$$Variance = s^2 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}$$

where $s^2$ = sample variance, $x_i = i^{th}$ element, $\bar{x}$ = sample mean, and $n$ = sample size.

---

$$\text{Transpose Matrix} = \left[\mathbf{A}^{\mathrm{T}}\right]_{ij} = [\mathbf{A}]_{ji}\,.$$

where if $[\mathbf{A}]$ is an m × n matrix, then $[\mathbf{A}]^T$ is an n × m matrix.

---

$$Cov(x,y) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{n-1}$$

$$yrCov(x,y) = Cov(x,y) * 12$$

where $cov(x,y)$ = covariance between sample x and y, $x_i = i^{th}$ element of x, $y_i = i^{th}$ element of y, $\bar{x}$ = mean of x, $\bar{y}$ = mean of y, and $n$ = sample size

---

$$\text{Asset Weight} = w_i \forall w (1 \leq w \geq 0)$$

$$\text{where } \sum_{i=1}^{n} w_i = 1$$

---

$$\text{portfolio expected return} = r = \sum_{i=1}^{n} w_i E(r_i) = \begin{bmatrix} w_1 & \dots & w_n \end{bmatrix} \begin{bmatrix} E(r_1) \\ \vdots \\ E(r_n) \end{bmatrix} = w^T R$$

where $r$=portfolio expected return, $n$=number of assets, $w$=weight of asset, $E(r)$=asset expected return, and $w^T$=weights transposed.

---

$$\sigma = \sqrt{w^T S w}$$

where $\sigma$=standard deviation, $w^T$=weights transposed, $S$=Yearly Covariance Matrix, and $w$=weights.

$$\text{Sharpe Ratio} = \frac{R_p - R_f}{\sigma_p}$$

where $R_p$=portfolio return, $R_f$=risk free rate, and $\sigma$=standard deviation.

$$R_{\text{new}} = \text{random} P(x) \in \mathbb{R} | \{0 \leq x \leq 1\}$$

$$R_{sum} = \sum_{i=1}^{n} R_{new}$$

$$W = \sum_{i=1}^{n} w_i = R_{new} / R_{sum}$$

$$\text{Risk Free Rate of Return} = (\frac{1 + \text{Government Bond Rate}}{1 + \text{Inflation Rate}}) - 1$$

**Appendix B: Correlation Matrix**

$$Corr(x, y) = r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

where $r$ = correlation coefficient between sample x and y, $x_i = i^{th}$ element of x, $y_i = i^{th}$ element of y, $\bar{x}$ = mean of x, $\bar{y}$ = mean of y, and $n$ = sample size

Corr Excel Formula: `CORREL(H3:H124,H3:H124)`

It is not necessary to multiply the correlation matrix by an interval factor because the returned correlation coefficient is always measured on a scale from 1 to -1.

$$\text{Corr. Mat.} = \begin{bmatrix} Corr(A,A) & Corr(A,B) & \dots & Corr(A,Z) \\ Corr(B,A) & Corr(B,B) & \dots & Corr(B,Z) \\ \vdots & \vdots & \ddots & \vdots \\ Corr(Z,A) & Corr(Z,B) & & Corr(Z,Z) \end{bmatrix}$$

Correlation is the statistical measure that expresses the extent to which two variables are linearly

Table 16: Excel Correlation Matrix

| Correlation Matrix | NKE | MSFT | DIS | COST |
|---|---|---|---|---|
| **NKE** | 1 | 0.023178503 | 0.00817255 | 0.048770382 |
| **MSFT** | 0.023178503 | 1 | -0.100594656 | 0.002627957 |
| **DIS** | 0.00817255 | -0.100594656 | 1 | 0.118496265 |
| **COST** | 0.048770382 | 0.002627957 | 0.118496265 | 1 |

related. The correlation coefficient is measured on a scale from 1 to -1.

Click to return to section 1.5: Correlation Matrix

### Appendix C: Installing and Importing Python Packages

These packages are installed in a ipynb notebook within a conda environment using the ‘conda python 3.10.9‘ python interpreter. In order to use ipynb notebooks, first `while{$a || $b}`

Within an ipynb notebook, ‘%‘ is used to tell the notebook to install these packages within the current environment. Alternatively, these packages can also be installed manually using ‘pip install ’packagenamehere‘.

```
#install relevant libraries
%pip install tqdm
%pip install numpy
%pip install pandas
%pip install yfinance
%pip install matplotlib
%pip install plotly
```

Then packages are imported as required and Plotly is set to work in notebook mode for combatablility with ipynb notebooks

```
from tqdm import tqdm
import numpy as np
import pandas as pd
import yfinance as yf
import matplotlib.pyplot as plt
%matplotlib inline
import plotly.express as px
from plotly.offline import init_notebook_mode
init_notebook_mode(connected=True)
```

- tqdm: used to create a smart progress bar for loops. 60ns overhead per iteration
- numpy: used for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.
- pandas: used for managing data structures and operations for manipulating numerical tables and time series.
- yfinance: threaded and Pythonic way to download market data from Yahoo!®finance.
- matplotlib: comprehensive library for creating static, animated, and interactive visualizations in Python
- plotly: high-level Python web-based interactive visualization library: Wrapper for Plotly.py simplifying Plotly syntax.
- plotly.offile/notebook_mode: used to enable inline Plotly graphing for jupyter ipynb notebooks.

## Appendix D: Dataframe Helper Attributes and Functions

### .shape Attribute

Printing the shape Attribute returns (120,4). The returned dataframe should be 120 rows because it returns a monthly interval for a period of 10 years. 12 mo * 10 yrs = 120 rows. Should be 4 colmns because I have formatted a dataframe to just the 'Close' column for 4 different tickers. Therefore I know that the returned data set is at least the right size. Next, the functions head() and tail() are used to check the top and bottom rows of data, showing us that the order of dates and returned data looks correct. The returned close price data appears to be valid and what I requested using the yfinance download call.

```
df_close.shape
# output (120, 4)
```

### head() function

```
df_close.head()
```

returns the first 5 rows of the dataframe.

### tail() function

```
df_close.tail()
```

returns the last 5 rows of the dataframe.

Table 17: head() function

| Date | COST | DIS | MSFT | NKE |
|---|---|---|---|---|
| 2012-01-01 00:00:00-05:00 | 82.269997 | 38.900002 | 29.530001 | 25.997499 |
| 2012-02-01 00:00:00-05:00 | 86.059998 | 41.990002 | 31.740000 | 26.980000 |
| 2012-03-01 00:00:00-05:00 | 90.800003 | 43.779999 | 32.259998 | 27.110001 |
| 2012-04-01 00:00:00-04:00 | 88.180000 | 43.110001 | 32.020000 | 27.967501 |
| 2012-05-01 00:00:00-04:00 | 86.389999 | 45.709999 | 29.190001 | 27.045000 |

Table 18: tail() function

| Date | COST | DIS | MSFT | NKE |
|---|---|---|---|---|
| 2021-08-01 00:00:00-04:00 | 455.489990 | 181.300003 | 301.880005 | 164.740005 |
| 2021-09-01 00:00:00-04:00 | 449.350006 | 169.169998 | 281.920013 | 145.229996 |
| 2021-10-01 00:00:00-04:00 | 491.540009 | 169.070007 | 331.619995 | 167.289993 |
| 2021-11-01 00:00:00-04:00 | 539.380005 | 144.899994 | 330.589996 | 169.240005 |
| 2021-12-01 00:00:00-05:00 | 567.700012 | 154.889999 | 336.320007 | 166.669998 |

**describe() function**

The describe function returns descriptive statistics for a given data set.

- Count is the number of rows. 120, should be the same as shape.
- Mean is the average stock price over the given time period, 10 years.
- Std is the standard deviation is the square root of variance measuring the dispersion of the data set relative to its mean.
- For this program, 25, 50, and 70 don't really matter, but these are the % 'percentile' stock prices when organized from min price to max price over the 10 year period.
- Max is equal to 100% or 100th percentile and is the highest stock price for each stock over this 10 year period.

```
1 df_close.describe()
```

returned dataframe for the describe function is displayed below.

Table 19: describe function

| | COST | DIS | MSFT | NKE |
|---|---|---|---|---|
| count | 120.000000 | 120.000000 | 120.000000 | 120.000000 |
| mean | 203.006667 | 104.673166 | 98.866833 | 68.593083 |
| std | 105.720335 | 35.325643 | 79.270263 | 37.206607 |
| min | 82.269997 | 38.900002 | 26.620001 | 21.945000 |
| 25% | 120.565001 | 85.844997 | 40.977501 | 39.308750 |
| 50% | 161.250000 | 104.514999 | 63.059999 | 58.289999 |
| 75% | 267.102509 | 117.030003 | 134.537506 | 84.972502 |
| max | 567.700012 | 189.039993 | 336.320007 | 169.240005 |

## Appendix E: Python Plotting

### Plotting Line Graph

```
def plot_line(df, xlabel, ylabel, title):
    # matplotlib
    ax = df.plot(title = title, figsize=(16, 8), ax = None)
    ax.set_xlabel(xlabel)
    ax.set_ylabel(ylabel)
    ax.axhline(y=0, color='black')
    ax.legend(loc='upper left')
    plt.show()

    # plotly
    fig = px.line(df, title=title)
    fig.update_layout(xaxis_title=xlabel, yaxis_title=ylabel, showlegend=True, template='seaborn'
    )
    fig.show()
```

Here I define a generic function for plotting our line graphs. Each time I want to plot a line graph, I just call this function which plots the line graph for the entered argument data in both Matplotlib and Plotly. As seen by the plot_line from the plot_line function, both of these graphs use the same close_price dataframe, thereby plotting the same data.

### Plotting Matrices

```
def plot_matrix(df, title):
    # matplotlib
    fig, ax = plt.subplots()
    im = ax.imshow(df)
    plt.title(title)
    plt.colorbar(im)
    column_names = list(df.columns.values)
    plt.xticks(np.arange(len(column_names)), column_names)
    plt.yticks(np.arange(len(column_names)), column_names)
    plt.show()

    # plotly
    fig = px.imshow(df, title=title)
    fig.show()
```

Here I define a generic function for plotting arrays of data, or in this case, our square correlation matrices. Similar to the line graph function I created above, each time I want to plot an ”imshow” matrix graph, I call this function which plots the data in both Matplotlib and Plotly.

### Plotting Efficient Frontier

This function adds a marker to plotly graphs so that I can make certain points stand out.

```
from plotly import graph_objects as go
def addMarker(fig, x, y, color):
    fig.add_trace(
        go.Scatter(
            x=[x],
            y=[y],
            mode="markers",
            marker_symbol='star',
            marker_size=15,
            marker_color=color
        )
    )
```

Here I define a function that plots the randomized points on a scatter plot and marks the important points, plotting them in Matplotlib and Plotly. The efficient frontier is formed by the curve of the random portfolio and the points are generated using portfolio selection.

```
def plot_efficient_frontier():
    #matplotlib
    plt.figure(figsize=(16,8))
    plt.scatter(efficient_frontier_data['Risk'], efficient_frontier_data['Return'], c=
    efficient_frontier_data['Sharpe'], cmap='viridis')
    plt.colorbar(label='Sharpe Ratio')
    plt.xlabel('Risk')
    plt.ylabel('Return')
    plt.scatter(max_sharpe['Risk'], max_sharpe['Return'],c='green', s=50)
    plt.scatter(min_risk['Risk'], min_risk['Return'],c='blue', s=50)
    plt.scatter(max_return['Risk'], max_return['Return'],c='red', s=50)
    plt.show()

    #plotly
    fig = px.scatter(efficient_frontier_data, x='Risk', y='Return', title='Efficient Frontier',
    color='Risk')
    addMarker(fig, max_sharpe['Risk'], max_sharpe['Return'], 'Dark Green')
    addMarker(fig, min_risk['Risk'], min_risk['Return'], 'Cyan')
    addMarker(fig, max_return['Risk'], max_return['Return'], 'Red')
    fig.show()
```

## Appendix F: Minimum Risk and Maximum Return Portfolios

### Minimum Risk Portfolio

While the minimum risk portfolio is not optimal because it has a lesser Sharpe ratio than the optimal portfolio, it is an important figure and can be found by filtering through the random risk column for the minimum value.

```
i_min_risk = efficient_frontier_data['Risk'].idxmin()
min_risk = efficient_frontier_data.iloc[i_min_risk, :]
min_risk
```

Table 20: Minimum Risk Portfolio

| | Weights | Risk | Return | Sharpe |
|---|---|---|---|---|
| 73786 | [0.46686217593243035, 0.12728364757502117, 0.2... | 0.137713 | 0.218982 | 1.372289 |

### Maximum Return Portfolio

While the maximum return portfolio is not optimal because it has a lesser Sharpe ratio than the optimal portfolio, it is an important figure and can be found by filtering through the random return column for the maximum value.

```
i_max_return = efficient_frontier_data['Return'].idxmax()
max_return = efficient_frontier_data.iloc[i_max_return, :]
max_return
```

Table 21: Maximum Return Portfolio

| | Weights | Risk | Return | Sharpe |
|---|---|---|---|---|
| 67010 | [0.01967587492800101, 0.0012887928723923205, 0... | 0.196313 | 0.265044 | 1.197289 |

### Appendix G: github.com/Terraform05/MVO

Public Github Repo can be found at: https://github.com/Terraform05/MVO