**Advanced Programming- Python Final Project**
**Authors: Yiwei Han, Yuvik Umapathy, Jeffrey Dean**

# An Investigation on different methods to analyze DJIA stock prices and returns

## 1. Executive Summary

Prediction of the stock market is a long-time attractive topic to researchers from different fields. In particular, numerous studies have been conducted to predict the movement of the stock market using machine learning algorithms and models. It is difficult to give the accurate prediction of the stock prices. But there is some instructive advice we can give the investors by analyzing the price of each stock. In this project, based on the analysis of the history of DJIA, we use a few codes to leverage the specific timing which works as a signal indicating the time of the upwards and downwards of the price. And based on the implementation of our model, it can be easy for people trying to reach out to the daily report on managing the sharpe ratio and volatility of each one of the stocks.

## 2. Introduction

The Dow Jones is one of the most popular and widely-recognized stock market indices. It measures the daily stock market movements of 30 U.S. publicly-traded companies listed on the NASDAQ or the New York Stock Exchange (NYSE). The 30 publicly-owned companies are considered leaders in the United States economy.

As more and more money is being invested in the stock market, investors get nervous and anxious of the future trends of the stock prices in the markets. The primary area of concern is to determine the appropriate time to buy, hold or sell. Unfortunately, stock market prediction is not an easy task because stock market indices are dynamic, complicated, and chaotic in nature. Even though it is difficult to predict the market and the price of the stocks precisely, researchers are trying to find various tools for giving investors advice on investing. Thus, the purpose of the project is to construct tools to help people point out the times to buy or sell any stock in a way that manages to make profit.

## 3. Data

The data prepared for this study was taken from the same stocks, but in different ways as to compliment the type of analysis each member of the group was doing. As a base, all 30 stocks from the Dow Jones Industrial Average were taken.

### 3.1: Data gathering/cleaning methods for Part 1

For the first part of the analysis, the stocks from the Dow Jones Industrial Average were taken from Wikipedia. This was done as a way to quickly consolidate the names as well as the ticker symbols of these stocks. By obtaining specific indicators, new data could be quickly merged into the preexisting dataframe to reduce the computational cost of the procedure.

```
Dow_Jones
```

| | Company | Exchange | Symbol | Industry | Date added | Notes | Index weighting |
|---|---|---|---|---|---|---|---|
| 0 | 3M Company | NYSE | MMM | Conglomerate | 1976-08-09 | As Minnesota Mining and Manufacturing | 3.84% |
| 1 | American Express | NYSE | AXP | Financial services | 1982-08-30 | NaN | 2.88% |
| 2 | Amgen | NASDAQ | AMGN | Pharmaceutical industry | 2020-08-31 | NaN | 4.87% |
| 3 | Apple Inc. | NASDAQ | AAPL | Information technology | 2015-03-19 | NaN | 2.57% |
| 4 | Boeing | NYSE | BA | Aerospace and defense | 1987-03-12 | NaN | 4.92% |
| 5 | Caterpillar Inc. | NYSE | CAT | Construction and Mining | 1991-05-06 | NaN | 4.54% |
| 6 | Chevron Corporation | NYSE | CVX | Petroleum industry | 2008-02-19 | Also 1930-07-18 to 1999-11-01 | 2.03% |
| 7 | Cisco Systems | NASDAQ | CSCO | Information technology | 2009-06-08 | NaN | 1.00% |
| 8 | The Coca-Cola Company | NYSE | KO | Food industry | 1987-03-12 | Also 1932-05-26 to 1935-11-20 | 1.04% |
| 9 | Dow Inc. | NYSE | DOW | Chemical industry | 2019-04-02 | NaN | 1.25% |
| 10 | Goldman Sachs | NYSE | GS | Financial services | 2013-09-20 | NaN | 6.54% |
| 11 | The Home Depot | NYSE | HD | Retailing | 1999-11-01 | NaN | 6.24% |
| 12 | Honeywell | NYSE | HON | Conglomerate | 2020-08-31 | NaN | 4.47% |
| 13 | IBM | NYSE | IBM | Information technology | 1979-06-29 | Also 1932-05-26 to 1939-03-04 | 2.59% |
| 14 | Intel | NASDAQ | INTC | Information technology | 1999-11-01 | NaN | 1.25% |
| 15 | Johnson & Johnson | NYSE | JNJ | Pharmaceutical industry | 1997-03-17 | NaN | 3.12% |
| 16 | JPMorgan Chase | NYSE | JPM | Financial services | 1991-05-06 | NaN | 2.95% |
| 17 | McDonald's | NYSE | MCD | Food industry | 1985-10-30 | NaN | 4.49% |
| 18 | Merck & Co. | NYSE | MRK | Pharmaceutical industry | 1979-06-29 | NaN | 1.49% |
| 19 | Microsoft | NASDAQ | MSFT | Information technology | 1999-11-01 | NaN | 4.98% |
| 20 | Nike | NYSE | NKE | Apparel | 2013-09-20 | NaN | 2.58% |
| 21 | Procter & Gamble | NYSE | PG | Fast-moving consumer goods | 1932-05-26 | NaN | 2.64% |
| 22 | Salesforce | NYSE | CRM | Information technology | 2020-08-31 | NaN | 4.45% |

**Dow Jones data that was taken from Wikipedia**

To collect each stock's Closing Price, Yahoo Finance was used. A function was created to cycle through each stock's ticker symbol and use it to find the Closing Price of the stock within the last 100 days. The Adjusted Closing Price information was stored as a new column in the Dow Jones dataframe.

```
MMM = pd.read_html('https://finance.yahoo.com/quote/MMM/history?p=MMM')
z = MMM[0]
z
```

| | Date | Open | High | Low | Close* | Adj Close** | Volume |
|---|---|---|---|---|---|---|---|
| 0 | May 04, 2021 | 198.49 | 199.56 | 197.54 | 199.38 | 199.38 | 1879000 |
| 1 | May 03, 2021 | 197.99 | 199.47 | 196.66 | 198.61 | 198.61 | 1779800 |
| 2 | Apr 30, 2021 | 197.66 | 198.99 | 196.13 | 197.14 | 197.14 | 2506700 |
| 3 | Apr 29, 2021 | 196.81 | 199.41 | 196.27 | 199.05 | 199.05 | 1887000 |
| 4 | Apr 28, 2021 | 195.00 | 197.31 | 194.81 | 195.93 | 195.93 | 2516900 |
| ... | ... | ... | ... | ... | ... | ... | ... |
| 96 | Dec 15, 2020 | 174.39 | 175.06 | 172.55 | 174.68 | 173.25 | 2270600 |
| 97 | Dec 14, 2020 | 175.67 | 176.20 | 172.99 | 173.08 | 171.67 | 3700100 |
| 98 | Dec 11, 2020 | 172.30 | 174.65 | 172.17 | 174.02 | 172.60 | 1875700 |
| 99 | Dec 10, 2020 | 174.87 | 175.40 | 172.69 | 173.49 | 172.07 | 2228800 |
| 100 | *Close price adjusted for splits.**Adjusted cl... | *Close price adjusted for splits.**Adjusted cl... | *Close price adjusted for splits.**Adjusted cl... | *Close price adjusted for splits.**Adjusted cl... | *Close price adjusted for splits.**Adjusted cl... | *Close price adjusted for splits.**Adjusted cl... | *Close price adjusted for splits.**Adjusted cl... |

**Sample volume data for a stock (Ticker: MMM)**

As can be seen from the above snapshot, there are specific issues that needed to be addressed. The most glaring issue was the presence of character data in some of the rows. Specifically, row index 100 had phrases that were unnecessary for this study. Therefore, that row was removed completely (also another row midway through the column that accounted for a dividend split was also deleted). Another issue that was harder to notice but extremely important was the order the data was taken in. Since this data is live, the most recent day's data will always be the first row. Therefore, this would have to be taken into account when plotting the Closing Prices and thus the column was reversed entirely.

```
Dow_Jones
```

| | Company | Exchange | Symbol | Industry | Date added | Notes | Index weighting | close |
|---|---|---|---|---|---|---|---|---|
| 0 | 3M Company | NYSE | MMM | Conglomerate | 1976-08-09 | As Minnesota Mining and Manufacturing | 3.84% | [[172.07, 172.60, 171.67, 173.25, 173.98, 175.... |
| 1 | American Express | NYSE | AXP | Financial services | 1982-08-30 | NaN | 2.88% | [[119.45, 116.60, 118.33, 118.23, 118.21, 116.... |
| 2 | Amgen | NASDAQ | AMGN | Pharmaceutical industry | 2020-08-31 | NaN | 4.87% | [[226.48, 225.72, 227.77, 228.83, 226.75, 229.... |
| 3 | Apple Inc. | NASDAQ | AAPL | Information technology | 2015-03-19 | NaN | 2.57% | [[123.06, 122.23, 121.60, 127.69, 127.62, 128.... |
| 4 | Boeing | NYSE | BA | Aerospace and defense | 1987-03-12 | NaN | 4.92% | [[232.06, 234.43, 230.33, 228.62, 229.50, 225.... |
| 5 | Caterpillar Inc. | NYSE | CAT | Construction and Mining | 1991-05-06 | NaN | 4.54% | [[177.54, 176.31, 179.96, 177.75, 177.42, 179.... |
| 6 | Chevron Corporation | NYSE | CVX | Petroleum industry | 2008-02-19 | Also 1930-07-18 to 1999-11-01 | 2.03% | [[92.05, 91.16, 88.19, 88.12, 87.45, 87.18, 85... |
| 7 | Cisco Systems | NASDAQ | CSCO | Information technology | 2009-06-08 | NaN | 1.00% | [[43.65, 43.57, 44.04, 44.04, 44.14, 44.75, 44... |
| 8 | The Coca-Cola Company | NYSE | KO | Food industry | 1987-03-12 | Also 1932-05-26 to 1935-11-20 | 1.04% | [[52.61, 52.91, 52.83, 53.40, 52.62, 52.83, 53... |
| 9 | Dow Inc. | NYSE | DOW | Chemical industry | 2019-04-02 | NaN | 1.25% | [[53.83, 53.41, 52.48, 53.54, 53.24, 53.24, 55... |
| 10 | Goldman Sachs | NYSE | GS | Financial services | 2013-09-20 | NaN | 6.54% | [[243.44, 239.05, 236.86, 241.48, 242.82, 243.... |
| 11 | The Home Depot | NYSE | HD | Retailing | 1999-11-01 | NaN | 6.24% | [[263.16, 262.90, 263.93, 266.31, 267.91, 272.... |
| 12 | Honeywell | NYSE | HON | Conglomerate | 2020-08-31 | NaN | 4.47% | [[211.04, 213.68, 210.92, 213.23, 209.38, 211.... |
| 13 | IBM | NYSE | IBM | Information technology | 1979-06-29 | Also 1932-05-26 to 1939-03-04 | 2.59% | [[123.31, 122.63, 121.90, 124.27, 123.89, 123.... |

**Updated Dow Jones dataframe with ordered and cleaned Adj. Close column (close)**

Finally, another small function was made to return the Adj. Close Price column as one list. As can be seen in the "close" column, the entries are bundled in a double array. This quality of life function made performing subsequent analyses easier.

```
apple = AdjClose(Dow_Jones, 'AAPL')
apple
```

```
[123.06,
 122.23,
 121.6,
 127.69,
 127.62,
 128.51,
 126.47,
 128.04,
 131.68,
 130.76,
 131.77,
 136.49,
 134.67,
 133.52,
 132.49,
 129.22,
 130.81,
 126.41,
 130.72,
 131.85,
 128.79,
 128.61,
 130.69,
 128.72,
 126.95,
 127.64,
 131.83,
 136.67,
 138.86,
 142.71,
 142.95,
 141.85,
 136.89,
 131.76,
```


**Adj. Closing Price of Apple ("AAPL") was outputted as one list and then plotted (via matplot.lib)**

### 3.2: Data gathering/cleaning methods for Part 2

The data for part 2 is similar to the methodology found in part 1. The process is explained in the body of the code below, but essentially it is retrieved and cleaned from both Wikipedia to have a list of Dow 30 tickers that are most current, and data from Yahoo Finance to pair the information with the tickers themselves. The Yahoo Finance data is for all thirty tickers and is similar to the exercise from the course in a prior lecture. I made sure that within Python you could see when the information is extracted and when its completed as a response output text box. This process cycles through Yahoo Finance grabbing information corresponding to my ticker list in a loop process. From there the only cleaning in this portion was to support data frames with the right formatting and information to use later. Python Pandas was useful for this task to simplify how I wanted the cleaned data frames to look. The corresponding code samples are shown below to support these methods.

| | Company | Exchange | Symbol | Industry |
|---|---|---|---|---|
| 0 | 3M Company | NYSE | MMM | Conglomerate |
| 1 | American Express | NYSE | AXP | Financial services |
| 2 | Amgen | NASDAQ | AMGN | Pharmaceutical industry |
| 3 | Apple Inc. | NASDAQ | AAPL | Information technology |
| 4 | Boeing | NYSE | BA | Aerospace and defense |
| 5 | Caterpillar Inc. | NYSE | CAT | Construction and Mining |
| 6 | Chevron Corporation | NYSE | CVX | Petroleum industry |
| 7 | Cisco Systems | NASDAQ | CSCO | Information technology |
| 8 | The Coca-Cola Company | NYSE | KO | Food industry |
| 9 | Dow Inc. | NYSE | DOW | Chemical industry |
| 10 | Goldman Sachs | NYSE | GS | Financial services |
| 11 | The Home Depot | NYSE | HD | Retailing |
| 12 | Honeywell | NYSE | HON | Conglomerate |
| 13 | IBM | NYSE | IBM | Information technology |
| 14 | Intel | NASDAQ | INTC | Information technology |
| 15 | Johnson & Johnson | NYSE | JNJ | Pharmaceutical industry |
| 16 | JPMorgan Chase | NYSE | JPM | Financial services |
| 17 | McDonald's | NYSE | MCD | Food industry |
| 18 | Merck & Co. | NYSE | MRK | Pharmaceutical industry |
| 19 | Microsoft | NASDAQ | MSFT | Information technology |
| 20 | Nike | NYSE | NKE | Apparel |
| 21 | Procter & Gamble | NYSE | PG | Fast-moving consumer goods |
| 22 | Salesforce | NYSE | CRM | Information technology |
| 23 | The Travelers Companies | NYSE | TRV | Financial services |
| 24 | UnitedHealth Group | NYSE | UNH | Managed health care |
| 25 | Verizon | NYSE | VZ | Telecommunication |
| 26 | Visa Inc. | NYSE | V | Financial services |
| 27 | Walgreens Boots Alliance | NASDAQ | WBA | Retailing |
| 28 | Walmart | NYSE | WMT | Retailing |
| 29 | The Walt Disney Company | NYSE | DIS | Broadcasting and entertainment |

```
''' Downloading Historical Stock Data for all DJI Stocks '''
today = DT.date.today()
start = today - DT.timedelta(days=7)
end = today
dji_data = {}

for stock in tickers:
    print(f'Downloading stock data for {stock}')
    dji_data[stock] = web.get_data_yahoo(stock, start, end, interval = 'd')
    print(f'{stock} downloaded...')

''' Calculating the weekly returns of all DJI Stocks '''
dji_returns = pd.DataFrame()

for stock in tickers:
    print(f'Calculating the weekly return for {stock}')
    dji_returns[stock] = dji_data[stock]['Adj Close'].pct_change().dropna()
```

```
Downloading stock data for MMM
MMM downloaded...
Downloading stock data for AXP
AXP downloaded...
Downloading stock data for AMGN
AMGN downloaded...
Downloading stock data for AAPL
AAPL downloaded...
Downloading stock data for BA
BA downloaded...
Downloading stock data for CAT
CAT downloaded...
Downloading stock data for CVX
CVX downloaded...
Downloading stock data for CSCO
CSCO downloaded...
Downloading stock data for KO
KO downloaded...
Downloading stock data for DOW
DOW downloaded...
Downloading stock data for GS
GS downloaded...
Downloading stock data for HD
HD downloaded...
```

```
dji_volatility = dji_returns.std()
dji_volatility = pd.DataFrame(dji_volatility)
dji_volatility.columns = ['Volatility']
dji_means = dji_returns.mean(axis = 0)
dji_means = pd.DataFrame(dji_means)
dji_means.columns = ['Mean_Returns']
```

## 4. Analysis

### 4.1: Results/Discussion of Part 1

The first metric that we wanted to gather from the stocks in the Dow Jones Industrial Average was the potential of creating a model that could correctly identify when to buy/sell a stock within the last 100 days. Without calculating any specific metrics, could a rudimentary model be created with sufficient predictive accuracy?

The method taken in this approach involved the use of rolling averages. A 5 day and 10 day rolling average was taken (1,2 market weeks respectively) for the 100 day Adjusted Closing Prices of each of the 30 stocks. If the 5 day rolling average was greater than the first 10 day rolling average, a buy signal would be registered. Then, if the 5 day rolling average ever became less than the 10 day rolling average, a sell signal would be called. The mathematical reasoning behind this revolved around the sensitivity of these buy and sell signals. If the 5 day average is greater than the 10 day average, then the stock has seen some recent growth so it would be a good time to buy. Similarly, if the 5 day rolling average drops below the 10 day rolling average,

then the stock has seen some recent decline in the past few days. Thus, it may be time to sell it. Some interesting results arose from this method.

**Examples of graphs produced by this method, orange = 5 day rolling average, green = 10 day rolling average, blue = actual Adj. Closing Price of stock, buy signals in green, sell signals in red**

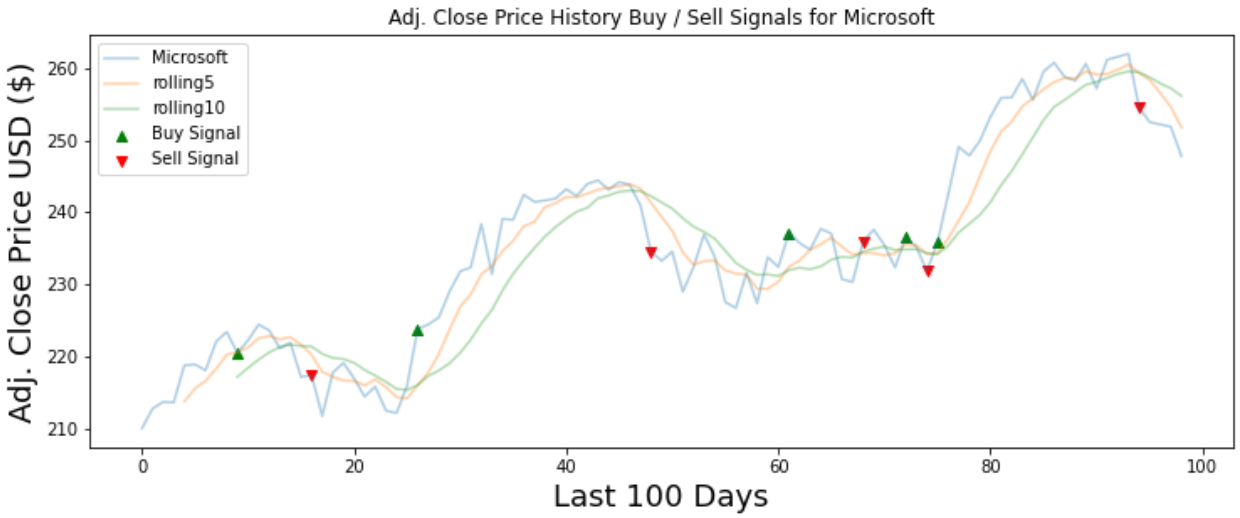
**Ex 1: Boeing**


As shown in the plot of Boeing, this method of prediction seems to be pretty accurate. The model correctly predicted to sell at day 10 (when the 10 day rolling average starts) and after an iteration of buy/sell at day 30, the model correctly predicted to buy the stock at day 40. If these predictions were followed, a substantial profit would be made. One important thing to note is the signal color that is indicated at day 70. Even though the closing price of Boeing Stock had been falling for the past 10 days before the signal was called, the algorithm took some time to make the decision. This occurred because of the overall volatility between days 60-70. Because of the sharp increase in closing price from days 60-65, it took some time for the 5 day-average to become lower than the 10 day average. As a general rule of thumb, this way of predicting seems to be better at identifying longer trends of growth or decline. However, as will be shown in future plots, it is nearly impossible to predict sudden drops or rises in Closing Price (almost for no apparent reason) at least within the past 100 days.

**Ex 2: Apple**


This visualization of Apple shows more of the true nature of this method of prediction. Just by looking from left to right, one can see that this model fails to make logical sense. Selling at day 18 is extremely puzzling, especially because the stock price rose for the next 20 days afterward. Again, the model was able to correctly identify to sell at day 45 and buy at day 75. Compared to the last graph, the accuracy of prediction is lower. However, if one bought X shares in Apple and followed this model, he/she would at least break even. This statement holds true because of the autocorrelations of stock price data. Since stock price data is highly autoregressive (hence time series analysis is required), it is more likely than not to see longer trends of decline and growth. Thus, this model can probably be concluded as sufficient in predicting times to buy or sell a stock.

**Ex 3: Microsoft**




The plot of Microsoft stock price displays a positive, cyclical trend. Again, there are large stretches where there is constant growth. The algorithm is able to identify these stretches and convert them into some profit. Even though it wasn't able to sell the stock before it decreased, the model sold the stock 1-2 days after, which was a very good sign. Following the model's prediction of buy/sell signals for this stock, a profit can definitely be made.

Overall, this model does an above-average job at predicting what times to buy/sell a certain stock. Certainly, this algorithm should not be blindly trusted if the stock price fluctuates positively and negatively frequently during a period of time (would not use this model for cryptocurrency). But, this model works quite well because of the inherent nature of time series data (autocorrelations). As long as it can be shown that the time series is not random white noise, the algorithm will rarely lose money in the long run. In future iterations of this study, machine learning techniques could be introduced to try and more accurately predict buy/sell signals. Also, specific stocks can be chosen and a larger time horizon can be used to predict the long-term success of this algorithm. Finally, the algorithm can be used on stocks that are not in the Dow Jones in order to prove whether this method works on all types of stocks.

**4.2: Results/Discussion of Part 2**

For the second portion of our work we developed a method of gathering, cleaning and sorting data from the Dow 30 to be made into useful dataframes. From these data frames built, we set out to find measures of interest, specifically the return and risk parameters of stock tickers and then graphed such data into a useful diagram for a user, particularly an investor with a

portfolio. This data further is used in an email setup to send out to a user with both the return, risk data and the graph. The key of the section is reporting data and specifically, mix and match factors of interest and create quick and direct ways to automate how to receive and use the data on a daily, weekly or longer basis. This will be further explained at the end of the section.

```
''' Downloading Historical Stock Data for all DJI Stocks '''
today = DT.date.today()
start = today - DT.timedelta(days=7)
end = today
dji_data = {}

for stock in tickers:
    print(f'Downloading stock data for {stock}')
    dji_data[stock] = web.get_data_yahoo(stock, start, end, interval = 'd')
    print(f'{stock} downloaded...')

''' Calculating the weekly returns of all DJI Stocks '''
dji_returns = pd.DataFrame()

for stock in tickers:
    print(f'Calculating the weekly return for {stock}')
    dji_returns[stock] = dji_data[stock]['Adj Close'].pct_change().dropna()
```

The above code is useful to support gathering code for our data on Yahoo Finance, which is the most robust source of free api data. Dow 30 is commonly known by investors with highly visibly tickers that demonstrate this tool well.

```
dji_volatility = dji_returns.std()
dji_volatility = pd.DataFrame(dji_volatility)
dji_volatility.columns = ['Volatility']
dji_means = dji_returns.mean(axis = 0)
dji_means = pd.DataFrame(dji_means)
dji_means.columns = ['Mean_Returns']
```

```
dji_series_means = dji_returns.mean(axis = 0)
dji_series_volatility = dji_returns.std()
max_dji_series_means = dji_series_means.idxmax()
max_dji_series_volatility = dji_series_volatility.idxmax()
```

The above sets of code develop data frames or series of the data, depending on what we need to use. The key of the data is that we can adjust the scope so that the returns are compiled weekly, monthly or even longer and then processed based on goals. The information conveys the ticker with the max of either return or volatility or the number return or volatility of that ticker. We can look at minimums or averages as well easily by substituting a different function of use. Below are the sets of output returns, volatility, and sharpe ratio as good metrics for the tool to look at, as a user.

## Dow Jones Mean Returns Data

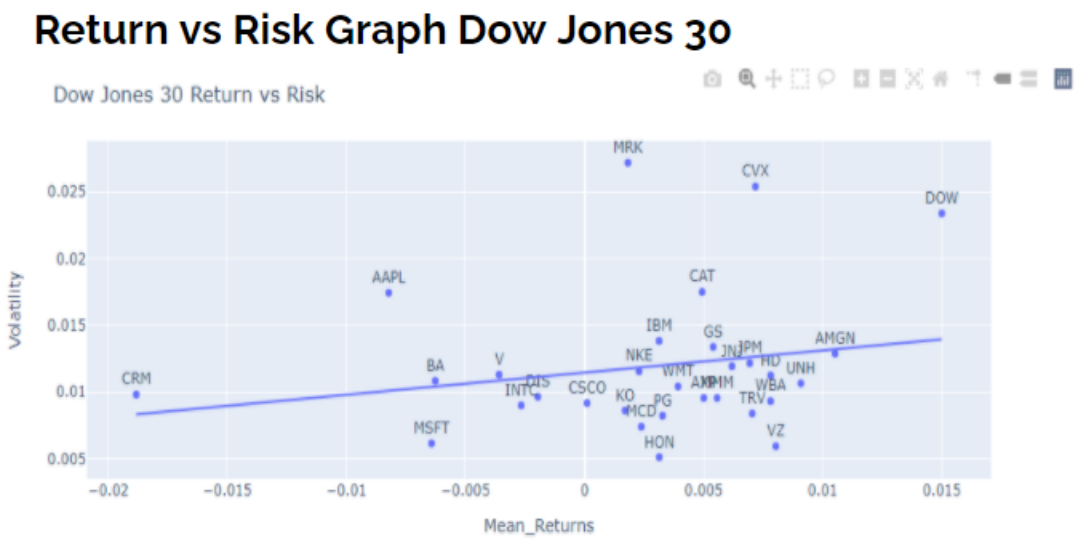
| | Mean_Returns |
|---|---|
| MMM | 0.005569 |
| AXP | 0.005008 |
| AMGN | 0.010524 |
| AAPL | -0.008219 |
| BA | -0.006266 |
| CAT | 0.004940 |
| CVX | 0.007178 |
| CSCO | 0.000112 |
| KO | 0.001703 |
| DOW | 0.015002 |
| GS | 0.005403 |
| HD | 0.007817 |
| HON | 0.003141 |
| IBM | 0.003135 |
| INTC | -0.002655 |
| JNJ | 0.006192 |
| JPM | 0.006940 |
| MCD | 0.002386 |
| MRK | 0.001826 |
| MSFT | -0.006423 |
| NKE | 0.002292 |
| PG | 0.003278 |
| CRM | -0.018820 |
| TRV | 0.007043 |
| UNH | 0.009085 |
| VZ | 0.008036 |
| V | -0.003582 |
| WBA | 0.007813 |
| WMT | 0.003929 |
| DIS | -0.001956 |

## Dow Jones Volatility Data

| | Volatility |
|---|---|
| MMM | 0.009551 |
| AXP | 0.009553 |
| AMGN | 0.012868 |
| AAPL | 0.017430 |
| BA | 0.010838 |
| CAT | 0.017493 |
| CVX | 0.025383 |
| CSCO | 0.009173 |
| KO | 0.008611 |
| DOW | 0.023373 |
| GS | 0.013379 |
| HD | 0.011245 |
| HON | 0.005121 |
| IBM | 0.013829 |
| INTC | 0.008999 |
| JNJ | 0.011929 |
| JPM | 0.012161 |
| MCD | 0.007394 |
| MRK | 0.027171 |
| MSFT | 0.006152 |
| NKE | 0.011550 |
| PG | 0.008228 |
| CRM | 0.009809 |
| TRV | 0.008386 |
| UNH | 0.010650 |
| VZ | 0.005935 |
| V | 0.011299 |
| WBA | 0.009314 |
| WMT | 0.010410 |
| DIS | 0.009642 |

## Dow Jones Mean Sharpe Ratio Data

| | |
|---|---|
| MMM | 0.583046 |
| AXP | 0.524260 |
| AMGN | 0.817887 |
| AAPL | -0.471558 |
| BA | -0.578165 |
| CAT | 0.282406 |
| CVX | 0.282789 |
| CSCO | 0.012207 |
| KO | 0.197816 |
| DOW | 0.641841 |
| GS | 0.403854 |
| HD | 0.695125 |
| HON | 0.613312 |
| IBM | 0.226674 |
| INTC | -0.295015 |
| JNJ | 0.519049 |
| JPM | 0.570671 |
| MCD | 0.322659 |
| MRK | 0.067221 |
| MSFT | -1.044026 |
| NKE | 0.198458 |
| PG | 0.398444 |
| CRM | -1.918732 |
| TRV | 0.839819 |
| UNH | 0.853024 |
| VZ | 1.353885 |
| V | -0.317031 |
| WBA | 0.838868 |
| WMT | 0.377433 |
| DIS | -0.202813 |




After we found our data on risk, return, and sharpe ratio data we can build a graph of interest on the data and apply the function in Python Plotly. The above plotly graph shows return vs risk for our developed data and the below is the risk vs return in a png file. The data is useful to see the performance quickly and easily once the program is run. If we want to know where there was high volatility or high return. Most data will cluster around a low return and low volatility as these blue chip tickers oscillate very slowly.

The next section supports code work on the email functionality built to help a user see data on stocks as soon as they open their emails. If an investor or fund manager wants to view return results or other metrics, it is very convenient. The analysis built here fits with the code pulled earlier.

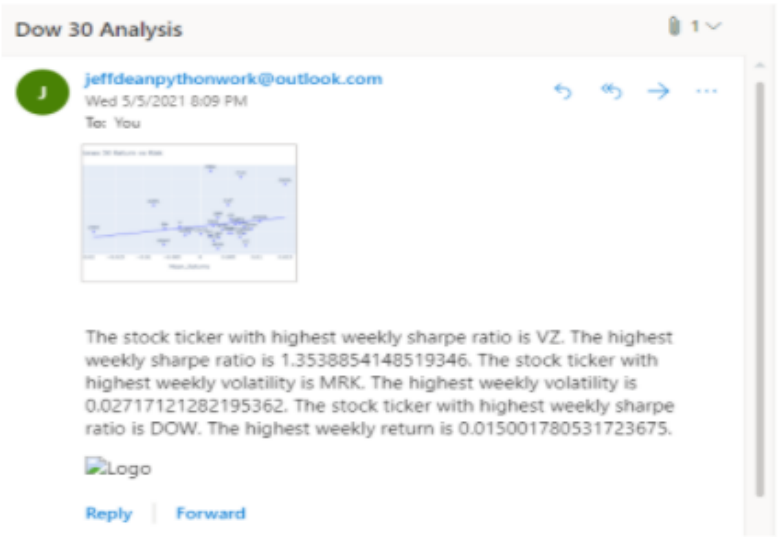
# Email Details and Code Part 1

```
def send_email(email_recipient,
               email_subject):

    email_sender = 'jeffdeanpythonwork@outlook.com'

    msg = MIMEMultipart()
    msg['From'] = email_sender
    msg['To'] = email_recipient
    msg['Subject'] = email_subject

    text = "The stock ticker with highest weekly sharpe ratio is " + str(max_dji_sharpe_series_ticker)  + ". " \
       "The highest weekly sharpe ratio is " + str(dji_largest_sharpe_series) + ". " \
       "The stock ticker with highest weekly volatility is " + str(max_dji_series_volatility) + ". " \
       "The highest weekly volatility is " + str(dji_largest_fluctuation_series) + ". " \
       "The stock ticker with highest weekly sharpe ratio is " + str(max_dji_series_means) + ". " \
       "The highest weekly return is " + str(dji_largest_gain_series) + ". "

    html = """\
    <html>
      <head></head>
        <body>
           <p>
           """ + text + """\
           </p>
           <img src="cid:image1" alt="Logo"><br>
        </body>
    </html>
    """
```

# Email Details and Code Part 2

```
    part2 = MIMEText(html, 'html')

    msg.attach(part2)

    image_file_name = "risk_return_graph.png"
    with open(image_file_name, 'rb') as file_in:
        attachment_data = file_in.read()

    attachment = MIMEImage(attachment_data)
    attachment.add_header(
        'Content-Disposition', 'attachment', filename=image_file_name
    )
    msg.attach(attachment)

    try:
        server = smtplib.SMTP('smtp.office365.com', 587)
        server.ehlo()
        server.starttls()
        server.login('jeffdeanpythonwork@outlook.com', 'jrd1995kpd')
        text = msg.as_string()
        server.sendmail(email_sender, email_recipient, text)
        print('email sent')
        server.quit()
    except:
        print("SMPT server connection error")
```

# Email Output




The code above and the output support that the code works and it will be rerun and generated each time the code is run. The graph is included in the body as a png file, with the body of the data updating and output as a string. We can adapt the email function to put other metrics in the body of the email, making the automation tool very robust. Finally below we will look at additional functions of interest that are built if we only need one item or other functions based on pulled data. An example email has been sent to the class email: "rutgersadvancedpython01@outlook.com" with the most recent weekly Dow 30 data.

# Other Functions Built- Volatility, Sharpe Ratio

```
def annual_volatility(dji_means):
    ''' Calculate the Annualized Volatility of a Trading Strategy '''
    volatility = dji_means.std() * np.sqrt(52)

    return annual_volatility

def monthly_volatility(dji_means):
    ''' Calculate the Monthly Volatility of a Trading Strategy '''
    volatility = dji_means.std() * np.sqrt(4)

    return monthly_volatility

def volatility(dji_means):
    ''' Calculate the Daily Volatility of a Trading Strategy '''
    volatility = dji_means.std()

    return volatility
```

```
def sharpe_ratio(dji_means, risk_free):
    ''' Calculate the Sharpe Ratio of a Portfolio '''
    sharpe = (CAGR(dji_means) - risk_free) / volatility(dji_means)

    return sharpe
```

The functions above are example functions that are good if we quickly need to see important outputs and more can be built if needed. We can look beyond these functions if we want information on alpha, beta, volume bought, volume sold and more.

## 5. Conclusion

In conclusion, we leveraged programs to pull data, clean it, and visualize it to interpret work on financial data. The recommendation tool can be given parameters and support if an investor should buy or sell a position including timing the investment. Models can be adapted to support new tools and methods to support sophisticated financial strategies, including the use of machine learning. The tools can be leveraged to find return and risk on investment portfolios over a time horizon and automate email functions for an investor.

The two ways we believe an investor or portfolio manager can utilize tools for their portfolio are through analysis and reporting. The work here supports these ideas and in the future, one could add code to these ideas in addition to the examples presented earlier in the report. If derivatives or bonds were used, an investor could pull data on characteristics like the "Greeks" or pull data on bond yields. The options are limitless. To add to the work we already have, all we would need are the correct sources to verify our plans for investigation. From there on out, reporting automated work would only need to be slightly amended. Python is very robust in its use of packages and we're certain that additional packages would be able to add to this project further with other types of analysis.