Capital Asset Pricing Model

Shikhar Bharat Jain
Data Analytics Engineering
George Mason University
sjain26@gmu.edu

Priyank Nilesh Gandhi
Data Analytics Engineering
George Mason University
pgandhi4@gmu.edu

Beenil Sunil Jain
Data Analytics Engineering
George Mason University
bjain2@gmu.edu

***Abstract*—This document explains the working of Capital Asset Pricing Model. The research selects twelve stocks over four industries and creates a portfolio to manage. We have used python to construct CAPM and time series model. To technically make the portfolio strong we use Sharpe Ratio. The project uses multivariate time series as the machine learning model.**

Keywords—Sharpe Ratio, CAPM, multivariate time series.

I. INTRODUCTION

The wealth management sector has experienced a substantial expansion in recent years as people and businesses search for ways to increase their profits. However, they typically need more knowledge or time to choose the best investing options. As a result, portfolio managers—dedicated people who administer this function on behalf of investors for a fee—were created. Portfolio managers act on behalf of vested investors by making investment choices and carrying out other pertinent activities. These experts seek to ascertain the investor's objectives and offer a portfolio that fully meets them. They work along with a group of researchers and analysts. They are responsible for picking the best investment strategy, selecting suitable investments, and properly distributing the assets. Investment portfolio management is creating and maintaining a collection of assets, such as stocks, bonds, and cash, that satisfy an investor's long-term financial objectives and risk tolerance. To outperform the performance of the overall market, active portfolio management involves systematically purchasing and selling stocks and other assets. The portfolio manager must use his resources to provide the investor with the optimal option. The expected rate of return on an asset or investment is determined using a capital asset pricing model (CAPM), a type of financial model. The Financial Model is accomplished via CAPM employing the correlation or sensitivity of the asset to the market, the projected rates of return on the market, and the risk-free asset (beta). The following CAPM drawbacks exist: Use a linear understanding of risk and return and erroneous assumptions. Due to its simplicity and convenience in evaluating investment possibilities, the CAPM method is still extensively employed despite its flaws. For instance, it is combined with Modern Portfolio Theory (MPT) to comprehend portfolio risk and projected return. For evaluating risk-adjusted performance, the Sharpe ratio divides a portfolio's excess return by a measure of volatility. Excess returns exceed industry benchmarks or risk-free returns. To calculate the betas, we have used simple linear regression model. Forecasts or past returns may be used in calculations to calculate *beta* for CAPM

II. OBJECTIVE

One of our essential jobs as a team is to authenticate the results obtained. In this task, we will leverage machine learning models to check the future prices of the stocks in the portfolio. Depending on the result, we can continue with the same investment, ensuring good profit. Otherwise, we as a team can deviate the investment into other groups of stocks if the model performance could be better. The objective of the current study is to use multivariate time series analysis to determine the primary goal of our job, to minimize the risk while increasing return, which would determine whether the investor meets their financial objectives.

III. DATA

The dataset was imported using python library pandas and converted into a data-frame. This dataset was generated by the data collected from Yahoo finance[1]. For this dataset, we have chosen a total of twelve stocks. All these stocks are from four different sectors: Technology, Real estate, Finance and Health care. The dataset contains three stocks from each sector. From the technology sector, we have selected NVDA (Nvidia), IBM (IBM) and GOOGL (Google). There is AMT (American Tower Corp), SPG (Simon Property Group), and WPC (WP Carey INC) from the real estate sectors. BAC (Bank of America Corp), GS (Goldman Sachs) and MS (Morgan Stanley) are the stocks selected from the Finance industry, and lastly, from Health Care, we selected ABT (Abbott Laboratories), JNJ (Johnson & Johnson) and PFE (Pfizer). S&P index was also imported for the same period. The data collected for this study was collected over a period of 10 years, from October 31, 2012, to October 31, 2022.

```
NVDA.head()
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	10/31/2012	3.0250	3.0500	2.9875	2.9950	2.749370	34711200
1	11/1/2012	3.0100	3.1400	3.0075	3.1375	2.880185	47322000
2	11/2/2012	3.1700	3.1750	3.1025	3.1225	2.866414	25670000
3	11/5/2012	3.1150	3.2675	3.1150	3.2550	2.988048	44484000
4	11/6/2012	3.2625	3.2625	3.1975	3.2525	2.985752	35080400

```
IBM.head()
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	10/31/2012	186.233276	187.772461	185.114716	185.975143	125.305962	6330706
1	11/1/2012	186.118546	189.187378	185.994263	188.479919	126.993668	3931705
2	11/2/2012	188.843216	189.292542	184.789673	184.923523	124.597412	4456065
3	11/5/2012	183.900574	186.395798	183.565964	185.602295	125.054703	2862065
4	11/6/2012	186.673035	188.097519	186.118546	186.491394	125.653831	3431926

Fig. 1: Importing and displaying all the datasets

A. *Data Preparation for Analysis*

For analysis only three columns are of our importance, Closing Price, Date, and Volume. We have taken the subset (refer figure 2) of the main dataset and merged it into a new dataset (refer figure 3) with all the required details of the twelve stocks.

```
NVDA_subset=NVDA[['Date','Close_NVDA','Volume_NVDA']]
GOOGL_subset=GOOGL[['Date','Close_GOOGL','Volume_GOOGL']]
IBM_subset=IBM[['Date','Close_IBM','Volume_IBM']]
AMT_subset=AMT[['Date','Close_AMT','Volume_AMT']]
SPG_subset=SPG[['Date','Close_SPG','Volume_SPG']]
WPC_subset=WPC[['Date','Close_WPC','Volume_WPC']]
ABT_subset=ABT[['Date','Close_ABT','Volume_ABT']]
JNJ_subset=JNJ[['Date','Close_JNJ','Volume_JNJ']]
PFE_subset=PFE[['Date','Close_PFE','Volume_PFE']]
BAC_subset=BAC[['Date','Close_BAC','Volume_BAC']]
MS_subset=MS[['Date','Close_MS','Volume_MS']]
GS_subset=GS[['Date','Close_GS','Volume_GS']]
snp_index_subset=snp_index[['Date','Close_snp_index','Volume_snp_index']]
```

Fig. 2: Preparing data for Analysis

```
final_data=final.set_index('Date')
final_data.head()
```

Date	Close_NVDA	Volume_NVDA	Close_GOOGL	Volume_GOOGL	Close_IBM	Volume_IBM	Close_AMT	Volume_AMT	Close_SPG	Volume_SPG	...	Clo
10/31/2012	2.9950	34711200	17.024525	61418520	185.975143	6330706	75.290001	2992600	143.189087	1392211	...	23
11/1/2012	3.1375	47322000	17.206957	81921996	188.479919	3931705	74.580002	2418000	143.508942	1025157	...	23
11/2/2012	3.1225	25670000	17.215216	92883024	184.923523	4456065	74.470001	2650500	146.519287	1908404	...	23
11/5/2012	3.2550	44484000	17.091091	65370564	185.602295	2862065	73.989998	2295900	145.559738	1072780	...	23
11/6/2012	3.2525	35080400	17.060061	63248688	186.491394	3431926	73.769997	2115100	146.274689	902274	...	23

5 rows × 26 columns

Fig. 3: Final dataset with date as index column

The final dataset is created with all the stock's volume, closing price and the date has been used as an index. This makes our dataset as time series dataset. After this we check if our dataset has any null values.

```
In [24]: final.isnull().sum()
Out[24]: Date                0
         Close_NVDA          0
         Volume_NVDA         0
         Close_GOOGL         0
         Volume_GOOGL        0
         Close_IBM           0
         Volume_IBM          0
         Close_AMT           0
         Volume_AMT          0
         Close_SPG           0
         Volume_SPG          0
         Close_WPC           0
         Volume_WPC          0
         Close_ABT           0
         Volume_ABT          0
         Close_JNJ           0
         Volume_JNJ          0
         Close_PFE           0
         Volume_PFE          0
         Close_BAC           0
         Volume_BAC          0
         Close_MS            0
         Volume_MS           0
         Close_GS            0
         Volume_GS           0
         Close_snp_index     0
         Volume_snp_index    0
         dtype: int64
```

Fig. 4: Output for null values

B. *Data Preparation for Model Fitting*

To calculate the CAPM and further model fitting, we need to calculate **Beta** of every stock. The figure (refer figure 5) below displays beta for each stock and the how to interpret these values. Beta is a way of *measuring a stock's volatility* compared with the *overall market's volatility*. We have used linear regression model to calculate beta with closing price of S&P index as a feature.

	stock	beta
0	Close_NVDA	1.588282
1	Close_GOOGL	1.115866
2	Close_IBM	0.860469
3	Close_AMT	0.797999
4	Close_SPG	1.081307
5	Close_WPC	0.805243
6	Close_ABT	0.904360
7	Close_JNJ	0.613033
8	Close_PFE	0.667715
9	Close_BAC	1.279226
10	Close_MS	1.370356
11	Close_GS	1.218485

```
# The value of beta is greater than 1 indicates that the stock is volatile
# The value of beta is equal to 1 indicates that the stock is goes with market trend
# The value of beta is less than 1 indicates that the stock is less volatile
```

Fig. 5: Beta value for each stock

After checking for Beta values, we need to make sure we are taking care of all the outliers. To recognize the outliers, we use the box plot method. We dedicate 25% to first quantile and 75% to the second quantile. We replace the outliers detected by median and repeat the procedure till we have minimum outliers.

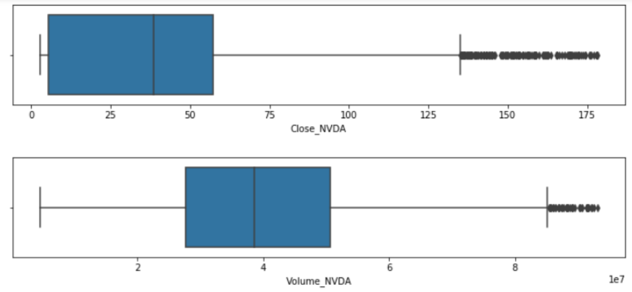


Fig. 6 Outliers detection using box plot

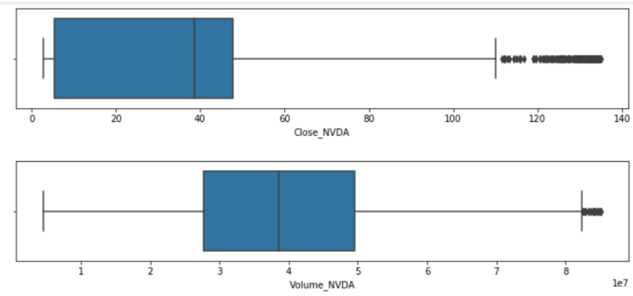


Fig. 7 Box plot after outlier removal

IV. Data Analysis

Investment has basic concept of risk to reward ratio, so we can analyse the risky stocks and then consider investing in them. Risk associated with investing in the stocks in portfolio was calculated in order to get an insight of the most and the least risky stocks. For calculating daily returns we calculate the percentage change between two days. After calculating percentage change, we take the standard deviation and that gives us the riskiness of a stock (refer fig. 8).

Fig. 8: Bar plot of risk in the stock market.

Figure 8 gives the idea that NVDA is the riskiest stock to invest. But we also need to know is the risk to reward ratio good enough for us to invest in this stock. We continue to make further visualizations to determine the average returns of all the twelve stocks.

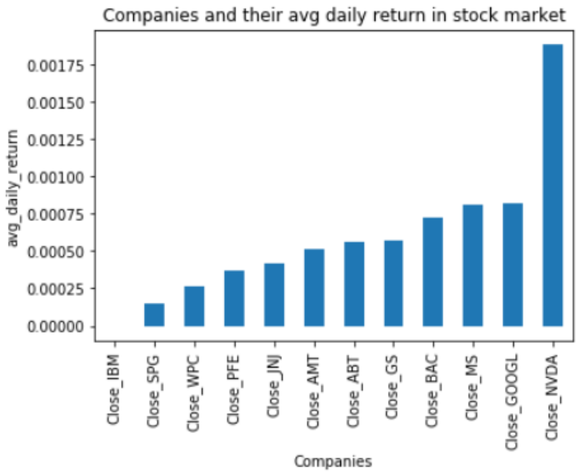


Fig. 9 Average Daily return

According to figure 9 NVDA has the highest average daily return and IBM has the lowest. This still suggests that NVDA is the riskiest stock currently.

This research assumes that we are investing equally in all the twelve stocks, and we call this equality of investment as weights. We calculate weight by dividing **1/12 = 0.083** and then assigning this weight equally to all the stocks. This indicates we are investing equal amount of money in each stock. The next visualization plots the total daily return if invested equally.

```python
daily_return.daily_return_stocks.plot(color="red")
plt.title("Total daily return from the all stocks having equal weights")
plt.ylabel("daily total return")
plt.xticks(rotation=90)
plt.show()
```

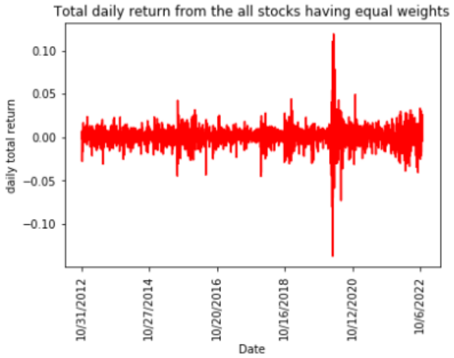


Fig.10: Total daily returns with equal weights.

We further analyse the covariance of each stock to understand the relation between them.

```python
cov_stocks = daily_return.iloc[:,:-1]
cov_stocks = (cov_stocks.cov())*252

cov_stocks
```

	Close_NVDA	Close_GOOGL	Close_IBM	Close_AMT	Close_SPG	Close_WPC	Close_ABT	Close_JNJ	Close_PFE	Close_BAC	Close_MS	Close_GS
ose_NVDA	0.190917	0.063104	0.034310	0.033331	0.034048	0.024821	0.041411	0.019574	0.021724	0.049999	0.059285	0.051790
e_GOOGL	0.063104	0.071230	0.025828	0.024160	0.029424	0.021770	0.029528	0.018043	0.019840	0.036471	0.040943	0.036459
Close_IBM	0.034310	0.025828	0.055279	0.019835	0.031685	0.023366	0.023917	0.018691	0.020071	0.037221	0.038894	0.034044
lose_AMT	0.033331	0.024160	0.019835	0.056950	0.029110	0.032002	0.026612	0.019254	0.018996	0.023162	0.026806	0.022756
lose_SPG	0.034048	0.029424	0.031685	0.029110	0.127723	0.048005	0.017829	0.015013	0.019321	0.052826	0.050015	0.049881
lose_WPC	0.024821	0.021770	0.023366	0.032002	0.048005	0.064176	0.020496	0.016278	0.016414	0.031329	0.033466	0.028218
lose_ABT	0.041411	0.029528	0.023917	0.026612	0.017829	0.020496	0.054101	0.023355	0.024461	0.030474	0.034772	0.028816
Close_JNJ	0.019574	0.018043	0.018691	0.019254	0.015013	0.016278	0.023355	0.031254	0.021051	0.020926	0.023341	0.019938
lose_PFE	0.021724	0.019840	0.020071	0.018996	0.019321	0.016414	0.024461	0.021051	0.048260	0.025094	0.026884	0.022289
lose_BAC	0.049999	0.036471	0.037221	0.023162	0.052826	0.031329	0.030474	0.020926	0.025094	0.095708	0.080551	0.071396
Close_MS	0.059285	0.040943	0.038894	0.026806	0.050015	0.033466	0.034772	0.023341	0.026884	0.080551	0.097833	0.076652
Close_GS	0.051790	0.036459	0.034044	0.022756	0.049881	0.028218	0.028816	0.019938	0.022289	0.071396	0.076652	0.078784

Fig. 11: Covariance matrix of stocks

The covariance matrix in fig. 11 tells us how the two given stocks in the portfolio are related to one another. If the covariance is negative, it means that the two stocks are going in opposite directions, while when it is positive, it means the stocks go in the same direction. Here, none of the covariances seem to be negative, so, this suggests us that all the stocks are going in same direction. Followed by this, the variance of the daily returns of the complete portfolio is calculated. *The variance helps in the computation of risk* associated with the investment in the portfolio.

```
In [45]: variance
Out[45]: 0.035769734344243326

In [46]: all_stocks_risk = np.sqrt(variance)
         all_stocks_risk
Out[46]: 0.18912888289270713
```

Fig. 12: Risk calculation of stock.

Risk is calculated by taking the square root of variance. The variance is calculated by multiplying the weights with covariance for 252 trading days. Further we assume that the risk-free rate is zero and that helps us calculate **Sharpe Ratio.**

```python
# Assuming that the risk free rate is zero
Sharpe_Ratio = daily_return['daily_return_stocks'].mean() /daily_return['daily_return_stocks'].std()
Sharpe_Ratio
```
```
0.0495103385289565
```
```python
Annual_Sharpe_Ratio = (252**0.5)*Sharpe_Ratio
Annual_Sharpe_Ratio
```
```
0.7859522584454303
```

Fig.13 Calculation of Sharpe and Annual Sharpe Ratio

As we can see from the dataset all the values are not within one range hence analysing them becomes quite tedious. Hence, we have normalized every stock, so they fall in one range and hence it becomes easy to create any visualization.

$$\textbf{Normalized Price} = \frac{\textbf{Current Closing Price}}{\textbf{Initial Price of Stock}}$$

```python
for i in stocks:
    stocks_normalized[i]=stocks_normalized[i]/final_data[i][0]
```
```python
stocks_normalized.head()
```

Date	Close_NVDA	Close_GOOGL	Close_IBM	Close_AMT	Close_SPG	Close_WPC	Close_ABT	Close_JNJ
10/31/2012	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
11/1/2012	1.047579	1.010716	1.013468	0.990570	1.002234	0.942596	0.999237	1.009602
11/2/2012	1.042571	1.011201	0.994345	0.989109	1.023257	0.933090	0.991756	1.001130
11/5/2012	1.086811	1.003910	0.997995	0.982733	1.016556	0.913528	0.992977	0.999576
11/6/2012	1.085977	1.002087	1.002776	0.979811	1.021549	0.910238	0.989618	1.002683

Fig 14 Normalized prices of all the stocks

As we have normalized the prices, we can use line plot to determine which stocks in individual sectors are performing better than S&P 500 index.

Fig. 15 Health sector stocks

In the figure above (refer to figure 15) we can see stock ABT (blue) is performing well above S&P 500 index (red). Comparatively, other stocks are performing well below the index and hence are considered to be less volatile.

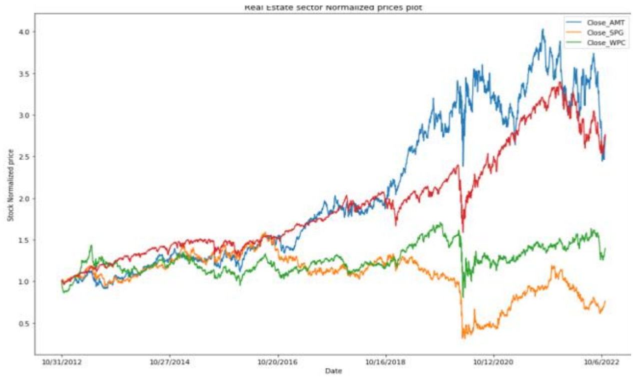


Fig 16 Real estate sector

In this plot (refer fig 16) S&P index and AMT are almost performing on equal levels and hence the volatile level of the stock is near to 1.

Fig 17 Finance Sector

Stocks of finance sector (refer figure 17) are really performing better than S&P index and hence can be considered volatile.

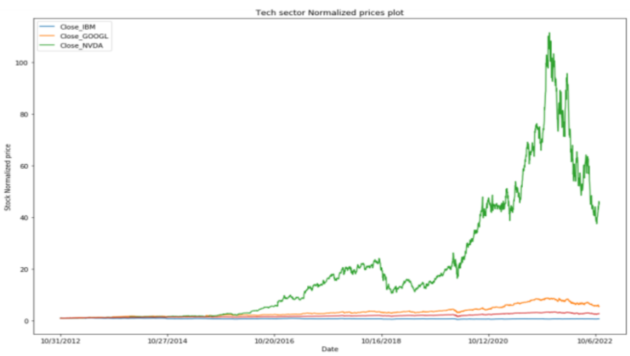


Fig. 18: Technology Sector

As we can clearly see (refer fig 18), only one stock NVDA is giving outrageous returns and is highly volatile. NVDA stock is way above the index and hence its beta is greater one.

The last test before we begun model fitting was to check if the values, we are using are stationary or non-stationary/seasonal values. We use KPSS test for stationarity. When a time series has a root of 1, the Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test determines whether it is stationary around its mean, linearly trending, or nonstationary. A series whose statistical characteristics, such as mean and variance, remain constant across time is referred to as a stationary time series.

```
-------------------------------------

Close_IBM
KPSS Statistic: 6.407540
Critical Values @ 0.05: 0.46
p-value: 0.010000

Series is Non-Stationary

-------------------------------------

Volume_IBM
KPSS Statistic: 0.605140
Critical Values @ 0.05: 0.46
p-value: 0.022169

Series is Non-Stationary
```

Fig 19 Finance Sector

The assumption that the data are stationary serves as the test's null hypothesis.

The possibility that the data are not stationary is an option for this test. (Refer fig 19)

V. Model Fitting

1.) CAPM (Capital Asset Pricing Model):

In the data preparation process for model fitting, Betas for all the stocks were calculated. These calculated betas were used during the CAPM model fitting. In order to construct the CAPM model, it is necessary to take into account the beta of a security, the risk-free rate, and the expected return on the market minus the risk-free rate.[2]

It is important to note that the CAPM can be easily stress-tested to provide a range of possible outcomes so that the required rates of return can be determined in a confident manner.[6]

Capital returns on each stock are calculated in percentages. In order to do that, we needed to consider certain factors. The risk-free rate here was considered to be 0.75, and in order to calculate market returns we used S&P500 index over the period of 10 years.[5]

```
cap_ret=[]
for i in col:
    ER[i] = rf+(capm[i]*(rm-rf))
    cap_ret.append(ER[i])
```

Fig.20: Calculating Capital returns

Fig. 20 shows the calculation of expected returns (ER) based on the betas, risk-free interest(rf), and market returns(rm).

ER= rf + [beta (rm - rf)].

```
In [85]: capm_df
Out[85]:
```

	stock	returns in %
0	Close_NVDA	18.023953
1	Close_GOOGL	12.886019
2	Close_IBM	10.108346
3	Close_AMT	9.428938
4	Close_SPG	12.510160
5	Close_WPC	9.507721
6	Close_ABT	10.585700
7	Close_JNJ	7.417267
8	Close_PFE	8.011978
9	Close_BAC	14.662698
10	Close_MS	15.653811
11	Close_GS	14.002085

Fig. 21: Stocks and their returns (%)

Fig. 21 shows stocks and their corresponding expected returns obtained using CAPM. NVDA has the highest expected return on investment over the span of 10 years, while JNJ has the least expected return percentages.

CAPM return for the whole portfolio containing 12 stocks from 4 different sectors was also calculated, considering the allotment of capital with equal weight in each of the 12 stocks.

```
portfolio_capm_ret = sum(list(ER.values())*weights)

portfolio_capm_ret

11.895129777348798
```

Fig 22: Expected return of the Portfolio.

Fig. 22 shows the expected return on capital investment in this portfolio containing the 12 stocks is 11.89%, which is approximately **12%**.

To validate the results from the CAPM, we have used one more model named VAR (Vector Auto Regression)

2.) VAR (Vector Auto Regression) Model:

It is one of the most commonly used multivariate timeseries forecasting techniques. In VAR, every attribute is a linear function of previous values of its own and also of the other attributes.[3]

Using VAR here is appropriate and helpful because it is capable of understanding and using the association among many other variables. The use of VAR is helpful for explaining the dynamic behaviour of the data and provides a better forecast as well.

After verifying the non-stationarity of timeseries data using KPSS test, we move forward and use differencing to make the data stationary. (Refer fig 19)

```
Close_IBM
KPSS Statistic: 0.040082
Critical Values @ 0.05: 0.46
p-value: 0.100000

Series is Stationary

--------------------------------------

Volume_IBM
KPSS Statistic: 0.018219
Critical Values @ 0.05: 0.46
p-value: 0.100000

Series is Stationary

--------------------------------------
```

Fig. 23: Converting series to stationary

We then create train-test split with 70% data as train data and rest of the data as test/validation data set. This procedure is performed for all the 12 stocks individually.

Firstly, building a VAR model for the NVDA stock. In order to determine which of several possible models best fits the data, the AIC is used to compare these different models and determine which one is the best fit.

Method of selecting the right order of the VAR model involves fitting increasing orders of the VAR model iteratively and picking the order that gives a model that has the least AIC value as a result.

Using AIC, it is possible to produce weights that can be directly used for model-averaging predictions or to generate parameters with a consistent interpretation across different models that can be used directly.

VAR Order Selection (* highlights the minimums)

	AIC	BIC	FPE	HQIC
0	39.05	39.05	9.063e+16	39.05
1	32.34	32.36	1.105e+14	32.34
2	32.29	32.32*	1.057e+14	32.30
3	32.28	32.33	1.048e+14	32.30
4	32.27	32.33	1.037e+14	32.29
5	32.27	32.34	1.032e+14	32.29
6	32.26	32.34	1.025e+14	32.29*
7	32.26	32.36	1.028e+14	32.30
8	32.26	32.37	1.027e+14	32.30
9	32.25	32.37	1.017e+14	32.30
10	32.25	32.39	1.018e+14	32.30
11	32.26	32.40	1.019e+14	32.31
12	32.25	32.41	1.017e+14	32.31
13	32.25	32.42	1.014e+14	32.31
14	32.25	32.43	1.014e+14	32.32
15	32.25	32.45	1.014e+14	32.32
16	32.25	32.46	1.018e+14	32.33
17	32.25	32.48	1.017e+14	32.34
18	32.25	32.48	1.013e+14	32.34
19	32.25*	32.50	1.012e+14*	32.34
20	32.25	32.51	1.014e+14	32.35
21	32.25	32.53	1.018e+14	32.36
22	32.25	32.54	1.016e+14	32.36
23	32.25	32.55	1.018e+14	32.36
24	32.26	32.57	1.020e+14	32.37
25	32.26	32.58	1.019e+14	32.38
26	32.26	32.59	1.020e+14	32.38
27	32.26	32.61	1.022e+14	32.39

Fig.24: Lag orders

Fig. 24 shows the lag orders, and at lag order 19, the '*' highlights the minimums of all metrics of evaluation and suggests fitting VAR model with maxlag=19.

```
In [113]: # Providing maxlags as 19 (minimum AIC vaue)
          res = model.fit(maxlags =19, ic = 'aic')

In [114]: res.summary()

Out[114]:   Summary of Regression Results
          ==================================
          Model:                         VAR
          Method:                        OLS
          Date:           Sun, 11, Dec, 2022
          Time:                     03:13:53
          --------------------------------------------------------------------
          No. of Equations:         2.00000    BIC:                    32.4795
          Nobs:                     1743.00    HQIC:                   32.3254
          Log likelihood:          -32961.3    FPE:                9.98874e+13
          AIC:                      32.2351    Det(Omega_mle):     9.55631e+13
          --------------------------------------------------------------------
          Results for equation Close_NVDA
          ==================================================================================
                               coefficient       std. error           t-stat            prob
          ----------------------------------------------------------------------------------
          const                   0.008341         0.084101            0.099           0.921
          L1.Close_NVDA           0.963221         0.024270           39.687           0.000
          L1.Volume_NVDA          0.000000         0.000000            0.279           0.780
```

Fig. 25: Summary of VAR model with max lag=19

Fig. 25 shows that the AIC selected was 32.2351, and BIC= 32.4795.

Now, to evaluate the model, validation data was used, which was the remaining 30% of the total data.

```
In [117]: # Model Evaluation
          from sklearn.metrics import mean_squared_error

          eval_results = pd.DataFrame(columns=['Column', 'RMSE', 'MAPE'])
          tempResults = pd.DataFrame(columns=['Column', 'RMSE', 'MAPE'])

          for col in NVDA_ts.columns:
              rmse = np.sqrt(mean_squared_error(test_NVDA_ts[col], forecast[col][:])).round(2)
              mape = np.round(np.mean(np.abs(test_NVDA_ts[col]-forecast[col][:])/test_NVDA_ts[col])*100,2)

              tempResults = pd.DataFrame({'Column':[col], 'RMSE': [rmse],'MAPE': [mape] })
              eval_results = pd.concat([eval_results, tempResults])

          eval_results
```

Out[117]:

	Column	RMSE	MAPE
0	Close_NVDA	37.79	40.25
0	Volume_NVDA	17075957.01	45.11

Fig. 26: VAR Model Evaluation for NVDA stock

Fig. 26 shows he evaluation metrics, such as RMSE and MAPE for NVDA stock.

MAPE is the Mean Absolute Percentage Error, which is one of the most used Key Performance Indicators (KPI) for measuring the accuracy of forecast.

Lower the MAPE, better the forecasting.

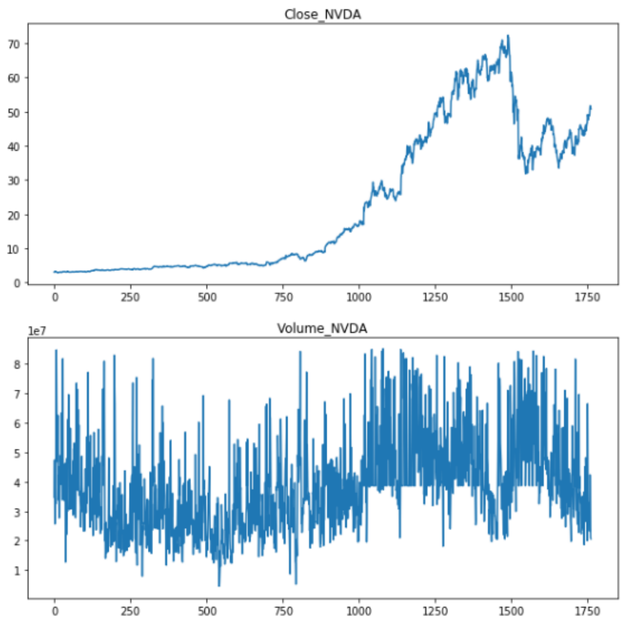


Fig. 27: Closing price and Volume graphs

Fig. 27 shows that the NVDA stock had a dramatic rise in the closing price after 1000 days.

Now we will check how good this model is in forecasting the trend, by using the same train dataset, but forecasting only the last 1000 data points(days) based on the past days.

```
res.plot_forecast(1000)
```

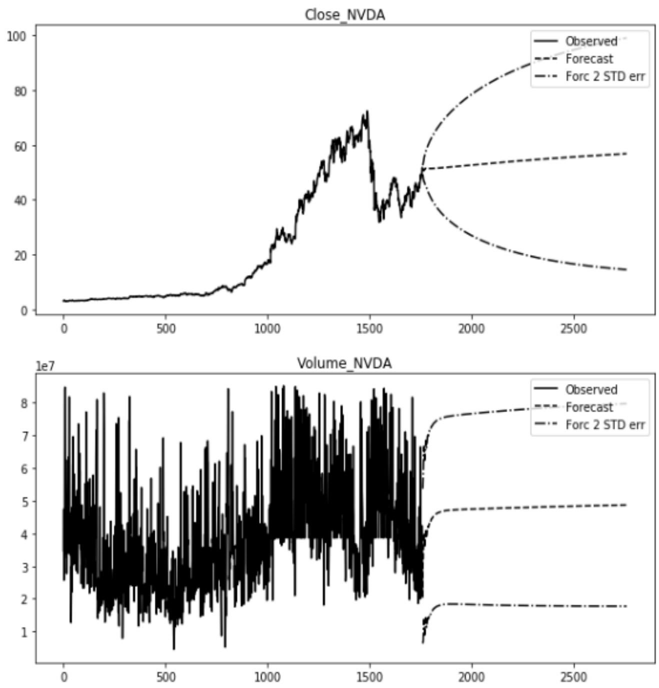


Fig. 28: Forecasting trend of NVDA stock

Fig. 28 depicts that the stock closing price and volume are being forecasted for the last 1000 days based on the previous days.

This forecasting shows an error of 2 standard deviations.

Likewise, the same procedure is followed for forecasting the prices and volumes of the remaining 11 stocks in the portfolio, using the VAR model, by selecting the appropriate lag orders for model fitting, considering the AIC and BIC values.

Stock	**Selected AIC**	**RMSE**	**MAPE**
NVDA	32.2351	37.79	40.25
GOOGL	31.1600	19.13	25.81
IBM	30.5579	19.61	14.41
AMT	27.2226	63.12	19.7
SPG	27.2423	34.06	21.35
WPC	24.7663	9.91	10.77
ABT	27.7577	15.06	10.92
JNJ	28.9495	33.51	18.5
PFE	29.2491	3.98	6.6
BAC	31.5050	8.54	20.7
GS	29.0957	38.39	8.2
MS	28.7677	6.15	7.15

Table 1: AIC and Accuracy Metrics

VI. References:

1.) https://finance.yahoo.com/

2.) Capital Asset Pricing Model (CAPM), CFI Team, November 24, 2022. https://corporatefinanceinstitute.com/resources/valuation/what-is-capm-formula/

3.) Vector Autoregression (VAR) – Comprehensive Guide with Examples in Python, Selva Prabhakaran, July7, 2019. https://www.machinelearningplus.com/time-series/vector-autoregression-examples-python/

4.) Does the Capital Asset Pricing Model Work?, David W. Mullins, Jr.,

5.) The capital asset pricing model: A critical literature review, https://www.researchgate.net/publication/307611046_The_capital_asset_pricing_model_A_critical_literature_review

6.) Capital Asset Pricing Model (CAPM) and Assumptions Explained, WILL KENTON, October 24, 2022

APPENDIX

Importing libraries and data sets.

In [1]:
```
# Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
# reading the technology industry data

NVDA=pd.read_csv('C:\\Users\\shikh\\OneDrive\\Desktop\\technology_companies\\NVDA.csv')
IBM=pd.read_csv('C:\\Users\\shikh\\OneDrive\\Desktop\\technology_companies\\IBM.csv')
GOOGL=pd.read_csv('C:\\Users\\shikh\\OneDrive\\Desktop\\technology_companies\\GOOGL.csv')
```

In [3]:
```
NVDA.head()
```

Out[3]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	10/31/2012	3.0250	3.0500	2.9875	2.9950	2.749370	34711200
1	11/1/2012	3.0100	3.1400	3.0075	3.1375	2.880185	47322000
2	11/2/2012	3.1700	3.1750	3.1025	3.1225	2.866414	25670000
3	11/5/2012	3.1150	3.2675	3.1150	3.2550	2.988048	44484000
4	11/6/2012	3.2625	3.2625	3.1975	3.2525	2.985752	35080400

In [4]:
```
IBM.head()
```

Out[4]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	10/31/2012	186.233276	187.772461	185.114716	185.975143	125.305962	6330706
1	11/1/2012	186.118546	189.187378	185.994263	188.479919	126.993668	3931705
2	11/2/2012	188.843216	189.292542	184.789673	184.923523	124.597412	4456065
3	11/5/2012	183.900574	186.395798	183.565964	185.602295	125.054703	2862065
4	11/6/2012	186.673035	188.097519	186.118546	186.491394	125.653831	3431926

In [5]:
```
GOOGL.head()
```

Out[5]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	10/31/2012	17.013514	17.042042	16.891891	17.024525	17.024525	61418520
1	11/1/2012	17.004505	17.289789	16.984985	17.206957	17.206957	81921996
2	11/2/2012	17.387136	17.406157	17.201450	17.215216	17.215216	92883024
3	11/5/2012	17.129629	17.188688	16.905907	17.091091	17.091091	65370564
4	11/6/2012	17.154154	17.179680	16.955706	17.060061	17.060061	63248688

Reading data from 4 different industries:

In [6]:
```
# reading the real estate industry data

AMT=pd.read_csv('C:\\Users\\shikh\\OneDrive\\Desktop\\realestate_companies\\AMT.csv')
SPG=pd.read_csv('C:\\Users\\shikh\\OneDrive\\Desktop\\realestate_companies\\SPG.csv')
WPC=pd.read_csv('C:\\Users\\shikh\\OneDrive\\Desktop\\realestate_companies\\WPC.csv')
```

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

Out[7]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	10/31/2012	72.889999	75.410004	72.139999	75.290001	62.339504	2992600
1	11/1/2012	75.169998	75.540001	73.230003	74.580002	61.751621	2418000
2	11/2/2012	75.040001	75.550003	74.339996	74.470001	61.660568	2650500
3	11/5/2012	73.769997	74.400002	73.230003	73.989998	61.263119	2295900
4	11/6/2012	74.019997	74.500000	73.379997	73.769997	61.080948	2115100

In [8]:
```
SPG.head()
```

Out[8]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	10/31/2012	142.652863	143.555969	141.495773	143.189087	93.011658	1392211
1	11/1/2012	143.142044	144.242706	141.909683	143.508942	93.219429	1025157
2	11/2/2012	145.296326	147.158981	145.296326	146.519287	95.174889	1908404
3	11/5/2012	145.268112	146.434616	144.421448	145.559738	94.551590	1072780
4	11/6/2012	145.813736	146.641586	145.239883	146.274689	95.015991	902274

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

Out[9]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	10/31/2012	48.599998	54.700001	47.400002	54.700001	30.570089	2466700
1	11/1/2012	53.500000	53.630001	51.099998	51.560001	28.815247	735500
2	11/2/2012	50.770000	51.740002	49.990002	51.040001	28.524635	410800
3	11/5/2012	50.500000	50.900002	49.669998	49.970001	27.926647	246800
4	11/6/2012	49.730000	50.099998	49.360001	49.790001	27.826050	312000

In [10]:
```
# reading the Health care industry data

ABT=pd.read_csv('C:\\Users\\shikh\\OneDrive\\Desktop\\healthcare_companies\\ABT.csv')
JNJ=pd.read_csv('C:\\Users\\shikh\\OneDrive\\Desktop\\healthcare_companies\\JNJ.csv')
PFE=pd.read_csv('C:\\Users\\shikh\\OneDrive\\Desktop\\healthcare_companies\\PFE.csv')
```

In [11]:
```
ABT.head()
```

Out[11]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	10/31/2012	31.575638	31.714781	31.287758	31.426901	26.005512	14188203
1	11/1/2012	31.551647	31.801144	31.374123	31.402910	25.985664	14941851
2	11/2/2012	31.609224	31.647608	31.139021	31.167809	25.791117	13617132
3	11/5/2012	31.081444	31.268566	31.043060	31.206192	25.822878	9482493
4	11/6/2012	31.220587	31.330940	31.076647	31.100636	25.735529	11469571

In [12]:
```
JNJ.head()
```

Out[12]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	10/31/2012	71.110001	71.250000	70.480003	70.820000	53.470665	9950600
1	11/1/2012	71.099998	71.900002	70.830002	71.500000	53.984085	11226000
2	11/2/2012	71.699997	71.699997	70.830002	70.900002	53.531078	7946700
3	11/5/2012	70.860001	71.000000	70.470001	70.790001	53.448017	6874500
4	11/6/2012	71.000000	71.620003	70.889999	71.010002	53.614117	7927500

In [13]:
```
PFE.head()
```

Out[13]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	10/31/2012	24.335863	24.430740	23.586338	23.595825	16.349766	40352495
1	11/1/2012	23.548388	23.548388	23.140417	23.292219	16.139393	57376598
2	11/2/2012	23.444023	23.595825	23.292219	23.292219	16.139393	32793734
3	11/5/2012	23.168880	23.453510	23.168880	23.320683	16.159119	21531428
4	11/6/2012	23.349146	23.681213	23.311195	23.444023	16.244583	31404246

In [14]:
```
# reading the Finance industry data

BAC=pd.read_csv('C:\\Users\\shikh\\OneDrive\\Desktop\\finance_companies\\BAC.csv')
GS=pd.read_csv('C:\\Users\\shikh\\OneDrive\\Desktop\\finance_companies\\GS.csv')
MS=pd.read_csv('C:\\Users\\shikh\\OneDrive\\Desktop\\finance_companies\\MS.csv')
```

In [15]:
```
BAC.head()
```

Out[15]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	10/31/2012	9.20	9.35	9.15	9.32	7.957659	94925500
1	11/1/2012	9.34	9.75	9.27	9.74	8.316269	205700000
2	11/2/2012	9.87	9.97	9.77	9.85	8.410188	220993300
3	11/5/2012	9.83	9.93	9.62	9.75	8.324805	121104400
4	11/6/2012	9.83	9.97	9.75	9.94	8.487031	132533500

In [16]:
```
GS.head()
```

Out[16]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	10/31/2012	119.730003	122.599998	119.660004	122.389999	103.646263	3679700
1	11/1/2012	122.820000	124.879997	122.360001	124.849998	105.729538	3336200
2	11/2/2012	125.349998	125.889999	123.059998	123.250000	104.374565	3188800
3	11/5/2012	123.190002	124.459999	122.110001	124.080002	105.077469	2568700
4	11/6/2012	124.330002	126.730003	124.300003	126.250000	106.915115	3986000

In [17]:
```
MS.head()
```

Out[17]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	10/31/2012	17.190001	17.600000	17.129999	17.379999	14.000978	26046300
1	11/1/2012	17.410000	17.610001	17.320000	17.610001	14.186262	25089900
2	11/2/2012	17.760000	17.840000	17.520000	17.780001	14.323213	19752700
3	11/5/2012	17.650000	17.770000	17.410000	17.750000	14.299043	16147600
4	11/6/2012	17.840000	18.240000	17.830000	18.190001	14.653499	15345900

Final Dataset after removing unwanted columns and merging different data frames and snp index column as well.

In [21]:
```
NVDA_subset=NVDA[['Date','Close_NVDA','Volume_NVDA']]
GOOGL_subset=GOOGL[['Date','Close_GOOGL','Volume_GOOGL']]
IBM_subset=IBM[['Date','Close_IBM','Volume_IBM']]
AMT_subset=AMT[['Date','Close_AMT','Volume_AMT']]
SPG_subset=SPG[['Date','Close_SPG','Volume_SPG']]
WPC_subset=WPC[['Date','Close_WPC','Volume_WPC']]
ABT_subset=ABT[['Date','Close_ABT','Volume_ABT']]
JNJ_subset=JNJ[['Date','Close_JNJ','Volume_JNJ']]
PFE_subset=PFE[['Date','Close_PFE','Volume_PFE']]
BAC_subset=BAC[['Date','Close_BAC','Volume_BAC']]
MS_subset=MS[['Date','Close_MS','Volume_MS']]
GS_subset=GS[['Date','Close_GS','Volume_GS']]
snp_index_subset=snp_index[['Date','Close_snp_index','Volume_snp_index']]
```

In [22]:
```
# merging different dataframe into one dataframe
final=pd.merge(NVDA_subset,GOOGL_subset,on='Date')
final=pd.merge(final,IBM_subset,on='Date')
final=pd.merge(final,AMT_subset,on='Date')
final=pd.merge(final,SPG_subset,on='Date')
final=pd.merge(final,WPC_subset,on='Date')
final=pd.merge(final,ABT_subset,on='Date')
final=pd.merge(final,JNJ_subset,on='Date')
final=pd.merge(final,PFE_subset,on='Date')
final=pd.merge(final,BAC_subset,on='Date')
final=pd.merge(final,MS_subset,on='Date')
final=pd.merge(final,GS_subset,on='Date')
final=pd.merge(final,snp_index_subset,on='Date')
```

In [23]:
```
final.head()
```

Out[23]:

	Date	Close_NVDA	Volume_NVDA	Close_GOOGL	Volume_GOOGL	Close_IBM	Volume_IBM	Close_AMT	Volume_AMT	Close_SPG	...	Close_PFE	Vo
0	10/31/2012	2.9950	34711200	17.024525	61418520	185.975143	6330706	75.290001	2992600	143.189087	...	23.595825	
1	11/1/2012	3.1375	47322000	17.206957	81921996	188.479919	3931705	74.580002	2418000	143.508942	...	23.292219	
2	11/2/2012	3.1225	25670000	17.215216	92883024	184.923523	4456065	74.470001	2650500	146.519287	...	23.292219	
3	11/5/2012	3.2550	44484000	17.091091	65370564	185.602295	2862065	73.989998	2295900	145.559738	...	23.320683	
4	11/6/2012	3.2525	35080400	17.060061	63248688	186.491394	3431926	73.769997	2115100	146.274689	...	23.444023	

5 rows × 27 columns

Average returns calculations:

```
In [31]: # For each company average daily return
         avg_daily_ret=daily_return.mean().sort_values()
         avg_daily_ret

Out[31]: Close_IBM     -0.000007
         Close_SPG      0.000147
         Close_WPC      0.000261
         Close_PFE      0.000365
         Close_JNJ      0.000419
         Close_AMT      0.000515
         Close_ABT      0.000563
         Close_GS       0.000568
         Close_BAC      0.000727
         Close_MS       0.000811
         Close_GOOGL    0.000822
         Close_NVDA     0.001890
         dtype: float64
```

```
In [32]: avg_daily_ret.plot(kind="bar")
         plt.xlabel("Companies")
         plt.ylabel("avg_daily_return")
         plt.title("Companies and their avg daily return in stock mar
         plt.show()
```

Betas Calculations:

```
In [49]: for i in stocks:
             print(i)
             print('maximum value is ${}'.format(final[i].max()))
             print('minimum value is ${}'.format(final[i].min()))
             print('-------------------------')

Close_NVDA
maximum value is $333.76001
minimum value is $2.845
-------------------------
Close_GOOGL
maximum value is $149.838501
minimum value is $16.195695999999998
-------------------------
Close_IBM
maximum value is $206.309753
minimum value is $90.602295
-------------------------
Close_AMT
maximum value is $303.619995
minimum value is $68.360001
-------------------------
Close_SPG
maximum value is $227.600006
minimum value is $44.009997999999996
-------------------------
Close_WPC
maximum value is $93.449997
minimum value is $43.860001000000004
-------------------------
Close_ABT
maximum value is $141.46000700000002
minimum value is $30.169825
-------------------------
Close_JNJ
maximum value is $186.009995
minimum value is $68.809998
-------------------------
Close_PFE
maximum value is $61.25
minimum value is $22.447819
-------------------------
Close_BAC
maximum value is $49.380001
minimum value is $8.99
-------------------------
Close_MS
maximum value is $108.730003
minimum value is $16.09
-------------------------
Close_GS
maximum value is $423.85000599999995
minimum value is $114.239998
-------------------------
```

CAPM model:

```
In [81]: ER = {}
         rf = 0.75
         rm = ret['Close_snp_index'].mean()*252*100

In [82]: cap_ret=[]
         for i in col:
             ER[i] = rf+(capm[i]*(rm-rf))
             cap_ret.append(ER[i])

In [83]: cap_ret

Out[83]: [18.023952888809323,
          12.88601905133368,
          10.108346098453387,
          9.428938481621167,
          12.51016840907292,
          9.507721334758084,
          10.585699782305404,
          7.417266553975563,
          8.011977959273093,
          14.662698291405869,
          15.653810531452562,
          14.002085416444087]

In [84]: capm_df = pd.DataFrame(
             {'stock': col,
              'returns in %': cap_ret
             })

In [85]: capm_df

Out[85]:
```

	stock	returns in %
0	Close_NVDA	18.023953
1	Close_GOOGL	12.886019
2	Close_IBM	10.108346
3	Close_AMT	9.428938
4	Close_SPG	12.510168
5	Close_WPC	9.507721
6	Close_ABT	10.585700
7	Close_JNJ	7.417267
8	Close_PFE	8.011978
9	Close_BAC	14.662698
10	Close_MS	15.653811
11	Close_GS	14.002085

```
In [86]: portfolio_capm_ret = sum(list(ER.values())*weights)

In [87]: portfolio_capm_ret

Out[87]: 11.895129777348798

In [88]: # Expected return on the stocks using CAPM is around 12% having equal weights
```

Data Differencing to make series Stationary:

```
In [98]: data_diff = final_data.diff().dropna()

In [99]: from statsmodels.tsa.stattools import kpss

         for i in data_diff.columns:
             kpss_test = kpss(data_diff[i])

             print(i)
             print('KPSS Statistic: %f' % kpss_test[0])
             print('Critical Values @ 0.05: %.2f' % kpss_test[3]['5%'])
             print('p-value: %f' % kpss_test[1])

             if kpss_test[1] <= 0.05:
                 print("\nSeries is Non Stationary")
             else:
                 print("\nSeries is Stationary")

             print('\n-------------------------------------\n')

Close_NVDA
KPSS Statistic: 0.046360
Critical Values @ 0.05: 0.46
p-value: 0.100000

Series is Stationary

-------------------------------------

Volume_NVDA
KPSS Statistic: 0.007366
Critical Values @ 0.05: 0.46
p-value: 0.100000

Series is Stationary

-------------------------------------

Close_GOOGL
```

VAR Model Evaluations and outputs for 12 stocks:

1.) NVDA:

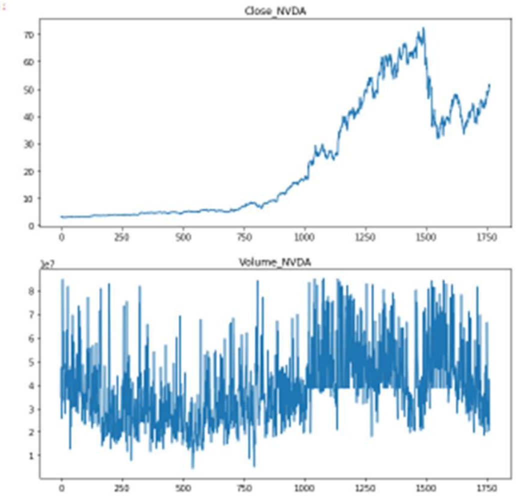
In [119]:
```
# Model Evaluation
from sklearn.metrics import mean_squared_error

eval_results = pd.DataFrame(columns=['Column', 'RMSE', 'MAPE'])
tempResults = pd.DataFrame(columns=['Column', 'RMSE', 'MAPE'])

for col in NVDA_ts.columns:
    rmse = np.sqrt(mean_squared_error(test_NVDA_ts[col], forecast[col][:])).round(2)
    mape = np.round(np.mean(np.abs(test_NVDA_ts[col]-forecast[col][:])/test_NVDA_ts[cc

    tempResults = pd.DataFrame({'Column':[col], 'RMSE': [rmse],'MAPE': [mape] })
    eval_results = pd.concat([eval_results, tempResults])

eval_results
```

Out[119]:

	Column	RMSE	MAPE
0	Close_NVDA	37.79	40.25
0	Volume_NVDA	17075957.01	45.11

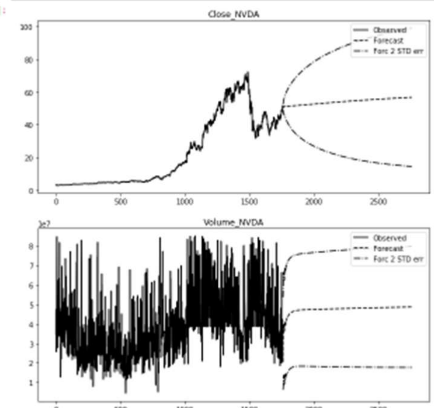
In [120]:
```
res.plot()
```

Out[120]:



In [121]:
```
res.plot_forecast(1000)
```

Out[121]:



2.) GOOGL

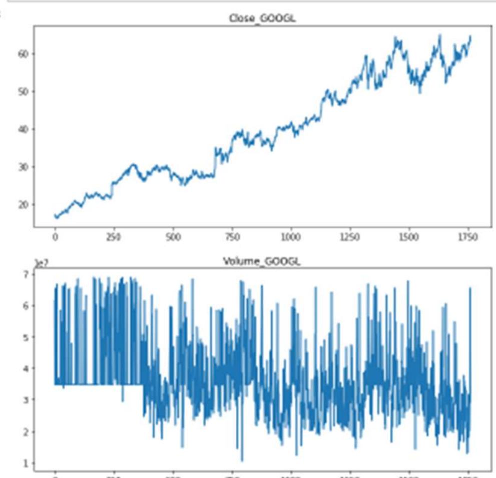
In [129]:
```
# Model Evaluation
from sklearn.metrics import mean_squared_error

eval_results = pd.DataFrame(columns=['Column', 'RMSE', 'MAPE'])
tempResults = pd.DataFrame(columns=['Column', 'RMSE', 'MAPE'])

for col in GOOGL_ts.columns:
    rmse = np.sqrt(mean_squared_error(test_GOOGL_ts[col], forecast[col][:])).round(2)
    mape = np.round(np.mean(np.abs(test_GOOGL_ts[col]-forecast[col][:])/test_GOOGL_ts[col])*100,2)

    tempResults = pd.DataFrame({'Column':[col], 'RMSE': [rmse],'MAPE': [mape] })
    eval_results = pd.concat([eval_results, tempResults])

eval_results
```

Out[129]:

	Column	RMSE	MAPE
0	Close_GOOGL	19.13	25.81
0	Volume_GOOGL	10871048.44	25.28

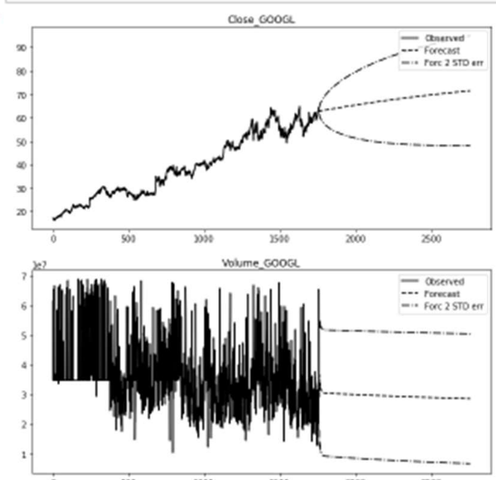
In [130]:
```
res.plot()
```

Out[130]:



In [131]:
```
res.plot_forecast(1000)
```

Out[131]:



3.) IBM

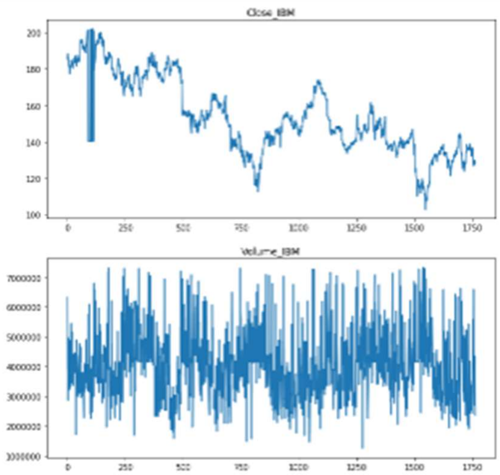
In [139]:
```
# Model Evaluation
from sklearn.metrics import mean_squared_error

eval_results = pd.DataFrame(columns=['Column', 'RMSE', 'MAPE'])
tempResults = pd.DataFrame(columns=['Column', 'RMSE', 'MAPE'])

for col in IBM_ts.columns:
    rmse = np.sqrt(mean_squared_error(test_IBM_ts[col], forecast[col][:])).round(2)
    mape = np.round(np.mean(np.abs(test_IBM_ts[col]-forecast[col][:])/test_IBM_ts[col])*100,2)

    tempResults = pd.DataFrame({'Column':[col], 'RMSE': [rmse],'MAPE': [mape] })
    eval_results = pd.concat([eval_results, tempResults])

eval_results
```

Out[139]:

	Column	RMSE	MAPE
0	Close_IBM	19.61	14.41
0	Volume_IBM	1186930.42	19.89

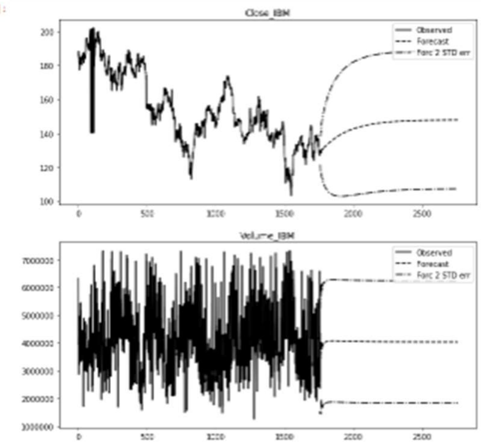
In [140]:
```
res.plot()
```

Out[140]:



In [141]:
```
res.plot_forecast(1000)
```

Out[141]:

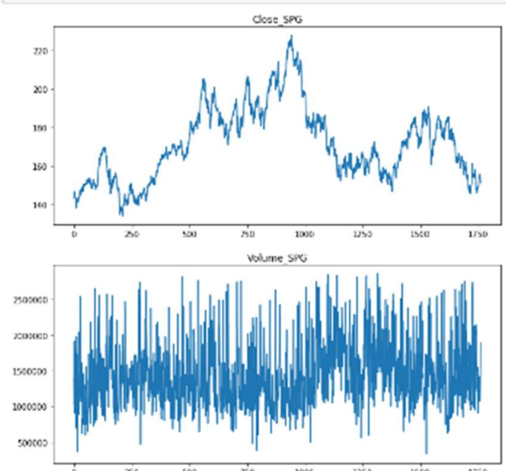


4.) AMT

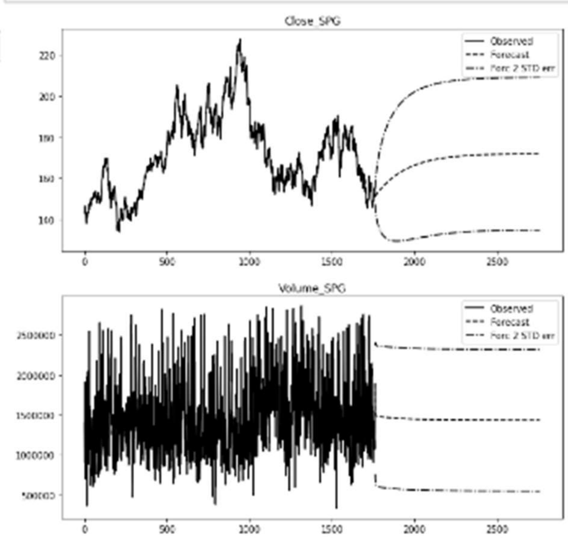
```
# Model Evaluation
from sklearn.metrics import mean_squared_error

eval_results = pd.DataFrame(columns=['Column', 'RMSE', 'MAPE'])
tempResults = pd.DataFrame(columns=['Column', 'RMSE', 'MAPE'])

for col in SPG_ts.columns:
    rmse = np.sqrt(mean_squared_error(test_SPG_ts[col], forecast[col][:])).round(2)
    mape = np.round(np.mean(np.abs(test_SPG_ts[col]-forecast[col][:])/test_SPG_ts[col])*100,2)

    tempResults = pd.DataFrame({'Column':[col], 'RMSE': [rmse],'MAPE': [mape] })
    eval_results = pd.concat([eval_results, tempResults])

eval_results
```

	Column	RMSE	MAPE
0	Close_SPG	34.06	21.35
0	Volume_SPG	522062.28	18.27

```
res.plot()
```




```
res.plot_forecast(1000)
```

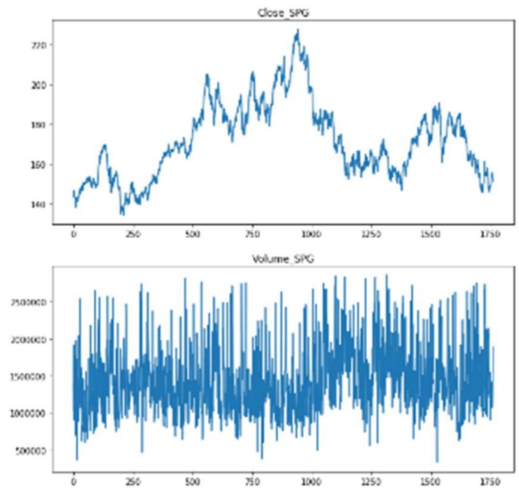


5.) SPG

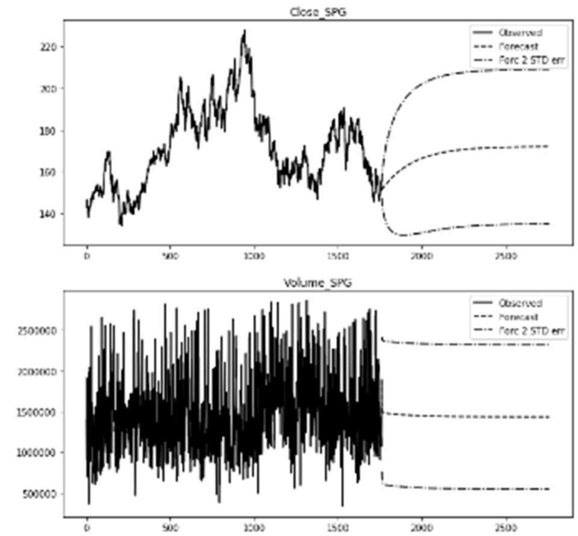
```python
# Model Evaluation
from sklearn.metrics import mean_squared_error

eval_results = pd.DataFrame(columns=['Column', 'RMSE', 'MAPE'])
tempResults = pd.DataFrame(columns=['Column', 'RMSE', 'MAPE'])

for col in SPG_ts.columns:
    rmse = np.sqrt(mean_squared_error(test_SPG_ts[col], forecast[col][:])).round(2)
    mape = np.round(np.mean(np.abs(test_SPG_ts[col]-forecast[col][:])/test_SPG_ts[col])*100,2)

    tempResults = pd.DataFrame({'Column':[col], 'RMSE': [rmse],'MAPE': [mape] })
    eval_results = pd.concat([eval_results, tempResults])

eval_results
```

	Column	RMSE	MAPE
0	Close_SPG	34.06	21.35
0	Volume_SPG	522062.28	18.27

```python
res.plot()
```




```python
res.plot_forecast(1000)
```

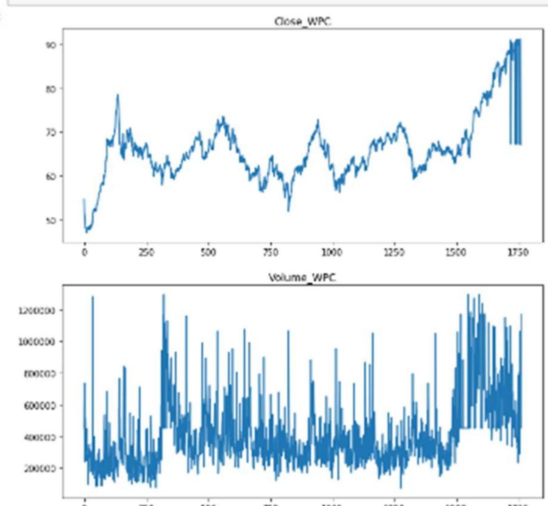


6.) WPC

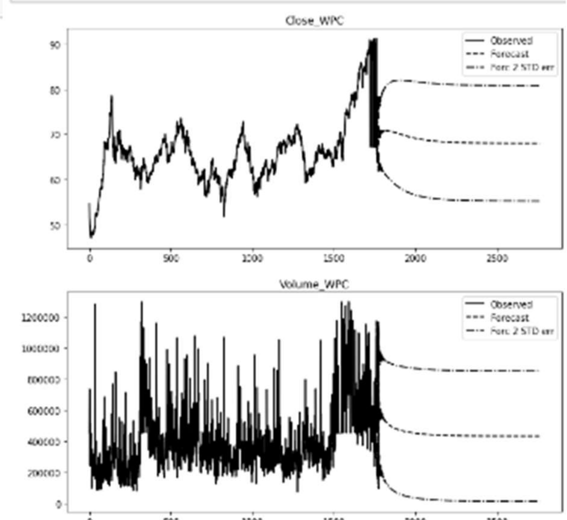
```python
# Model Evaluation
from sklearn.metrics import mean_squared_error

eval_results = pd.DataFrame(columns=['Column', 'RMSE', 'MAPE'])
tempResults = pd.DataFrame(columns=['Column', 'RMSE', 'MAPE'])

for col in WPC_ts.columns:
    rmse = np.sqrt(mean_squared_error(test_WPC_ts[col], forecast[col][:])).round(2)
    mape = np.round(np.mean(np.abs(test_WPC_ts[col]-forecast[col][:])/test_WPC_ts[col])*100,2)

    tempResults = pd.DataFrame({'Column':[col], 'RMSE': [rmse],'MAPE': [mape] })
    eval_results = pd.concat([eval_results, tempResults])

eval_results
```

	Column	RMSE	MAPE
0	Close_WPC	9.91	10.77
0	Volume_WPC	388151.21	36.28

```python
res.plot()
```




```python
res.plot_forecast(1000)
```

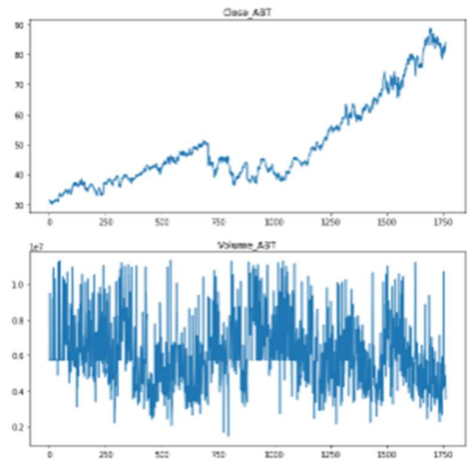


7.) ABT

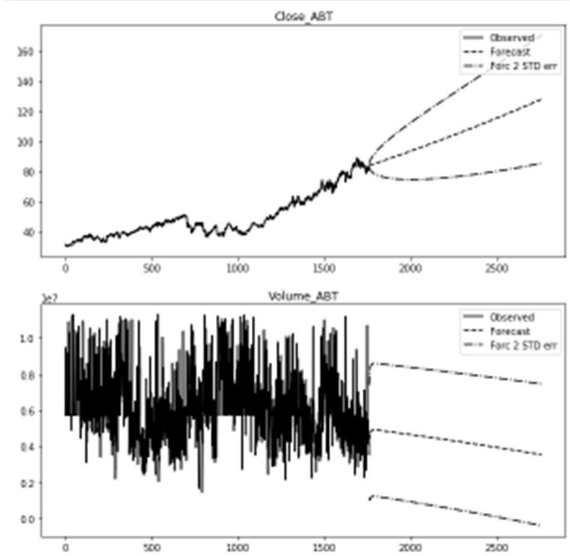
```python
# Model Evaluation
from sklearn.metrics import mean_squared_error

eval_results = pd.DataFrame(columns=['Column', 'RMSE', 'MAPE'])
tempResults = pd.DataFrame(columns=['Column', 'RMSE', 'MAPE'])

for col in ABT_ts.columns:
    rmse = np.sqrt(mean_squared_error(test_ABT_ts[col], forecast[col][:])).round(2)
    mape = np.round(np.mean(np.abs(test_ABT_ts[col]-forecast[col][:])/test_ABT_ts[col])*100,2)

    tempResults = pd.DataFrame({'Column':[col], 'RMSE': [rmse],'MAPE': [mape] })
    eval_results = pd.concat([eval_results, tempResults])

eval_results
```

	Column	RMSE	MAPE
0	Close_ABT	15.06	10.92
0	Volume_ABT	1998264.72	25.64

```python
res.plot()
```




```python
res.plot_forecast(1000)
```

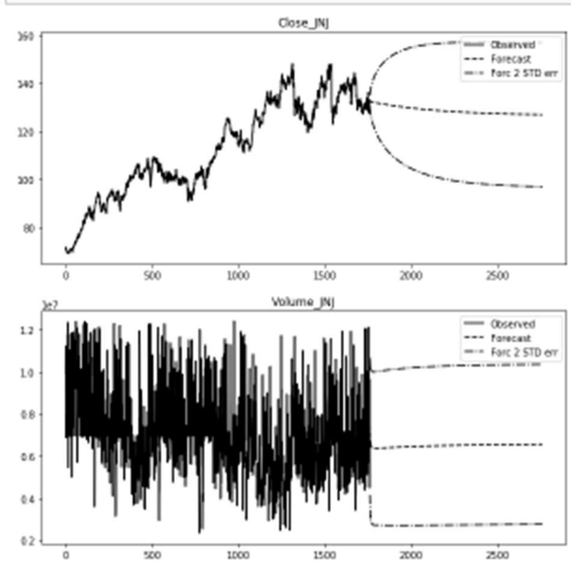


8.) JNJ

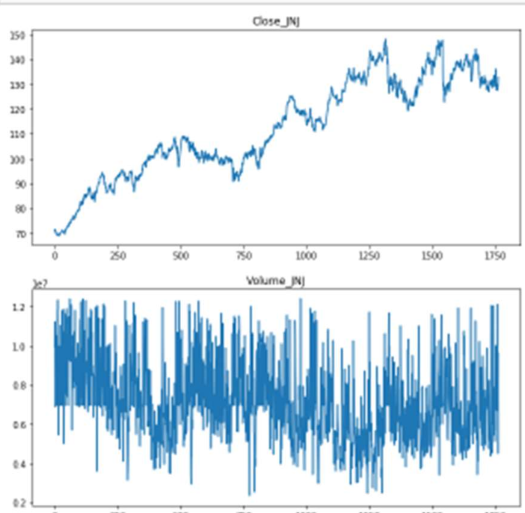
```python
# Model Evaluation
from sklearn.metrics import mean_squared_error

eval_results = pd.DataFrame(columns=['Column', 'RMSE', 'MAPE'])
tempResults = pd.DataFrame(columns=['Column', 'RMSE', 'MAPE'])

for col in JNJ_ts.columns:
    rmse = np.sqrt(mean_squared_error(test_JNJ_ts[col], forecast[col][:])).round(2)
    mape = np.round(np.mean(np.abs(test_JNJ_ts[col]-forecast[col][:])/test_JNJ_ts[col])*100,2)

    tempResults = pd.DataFrame({'Column':[col], 'RMSE': [rmse],'MAPE': [mape] })
    eval_results = pd.concat([eval_results, tempResults])

eval_results
```

	Column	RMSE	MAPE
0	Close_JNJ	33.51	18.50
0	Volume_JNJ	1901782.14	21.45

```python
res.plot_forecast(1000)
```




```python
res.plot()
```

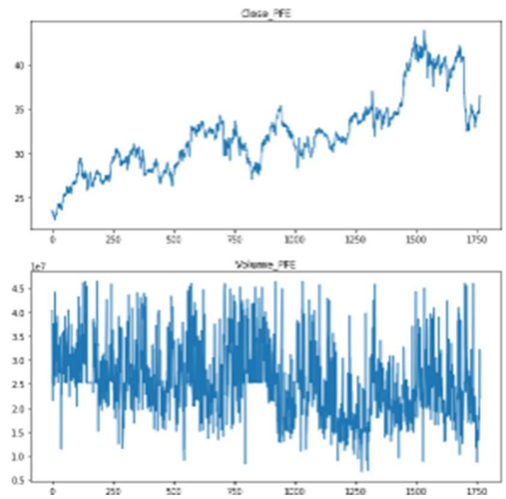


9.) PFE

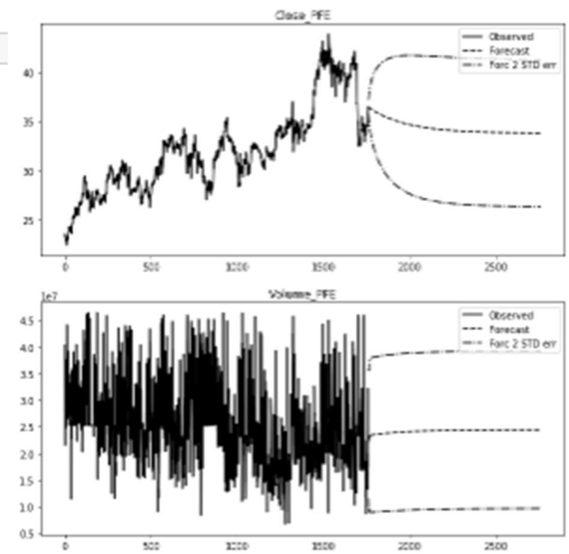
```
# Model Evaluation
from sklearn.metrics import mean_squared_error

eval_results = pd.DataFrame(columns=['Column', 'RMSE', 'MAPE'])
tempResults = pd.DataFrame(columns=['Column', 'RMSE', 'MAPE'])

for col in PFE_ts.columns:
    rmse = np.sqrt(mean_squared_error(test_PFE_ts[col], forecast[col][:])).round(2)
    mape = np.round(np.mean(np.abs(test_PFE_ts[col]-forecast[col][:])/test_PFE_ts[col])*100,2)

    tempResults = pd.DataFrame({'Column':[col], 'RMSE': [rmse],'MAPE': [mape] })
    eval_results = pd.concat([eval_results, tempResults])

eval_results
```

	Column	RMSE	MAPE
0	Close_PFE	3.98	6.61
0	Volume_PFE	7816084.84	24.75

```
res.plot()
```




```
res.plot_forecast(1000)
```

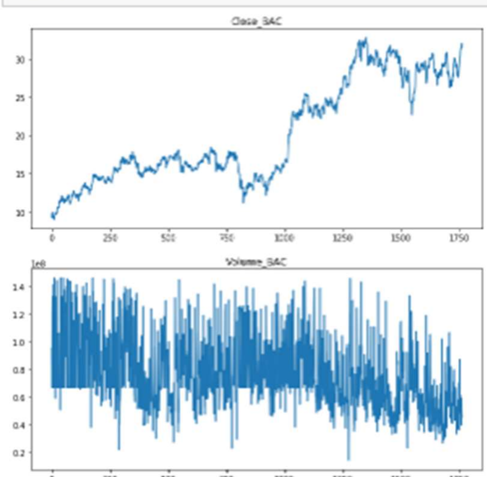


10.) BAC

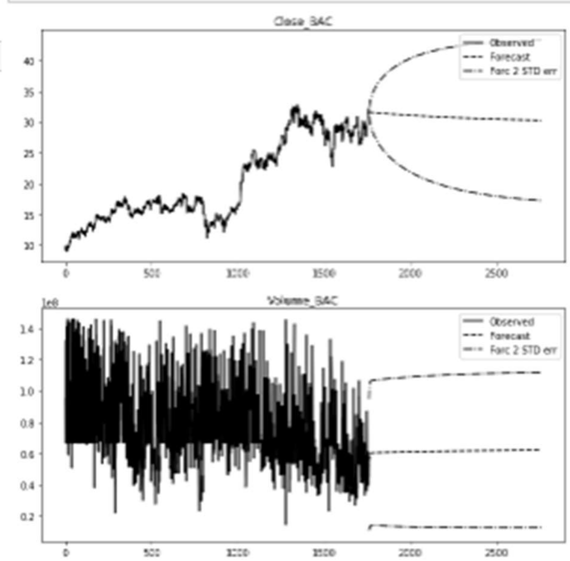
```
# Model Evaluation
from sklearn.metrics import mean_squared_error

eval_results = pd.DataFrame(columns=['Column', 'RMSE', 'MAPE'])
tempResults = pd.DataFrame(columns=['Column', 'RMSE', 'MAPE'])

for col in BAC_ts.columns:
    rmse = np.sqrt(mean_squared_error(test_BAC_ts[col], forecast[col][:])).round(2)
    mape = np.round(np.mean(np.abs(test_BAC_ts[col]-forecast[col][:])/test_BAC_ts[col])*100,2)

    tempResults = pd.DataFrame({'Column':[col], 'RMSE': [rmse],'MAPE': [mape] })
    eval_results = pd.concat([eval_results, tempResults])

eval_results
```

	Column	RMSE	MAPE
0	Close_BAC	8.54	20.67
0	Volume_BAC	21978318.95	39.24

```
res.plot()
```




```
res.plot_forecast(1000)
```

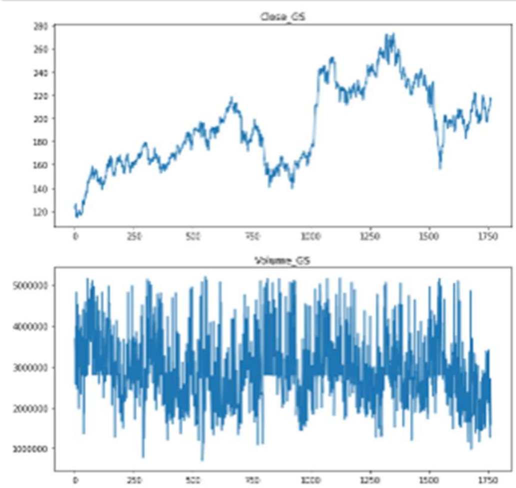


11.) GS

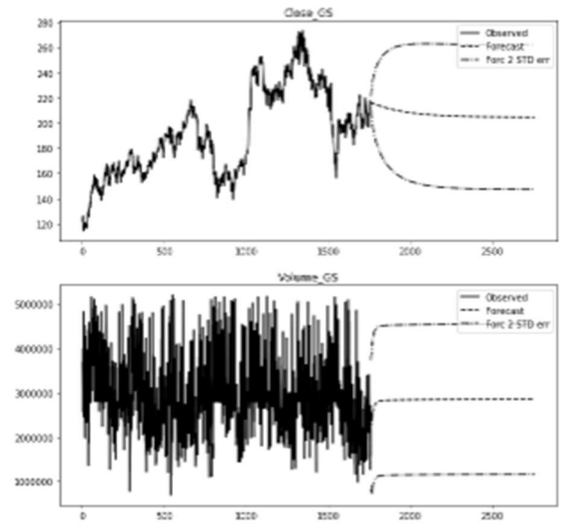
```python
# Model Evaluation
from sklearn.metrics import mean_squared_error

eval_results = pd.DataFrame(columns=['Column', 'RMSE', 'MAPE'])
tempResults = pd.DataFrame(columns=['Column', 'RMSE', 'MAPE'])

for col in GS_ts.columns:
    rmse = np.sqrt(mean_squared_error(test_GS_ts[col], forecast[col][:])).round(2)
    mape = np.round(np.mean(np.abs(test_GS_ts[col]-forecast[col][:])/test_GS_ts[col])*100,2)

    tempResults = pd.DataFrame({'Column':[col], 'RMSE': [rmse],'MAPE': [mape] })
    eval_results = pd.concat([eval_results, tempResults])

eval_results
```

	Column	RMSE	MAPE
0	Close_GS	38.39	8.26
0	Volume_GS	846092.00	29.89

```python
res.plot()
```




```python
res.plot_forecast(1000)
```



12.) MS

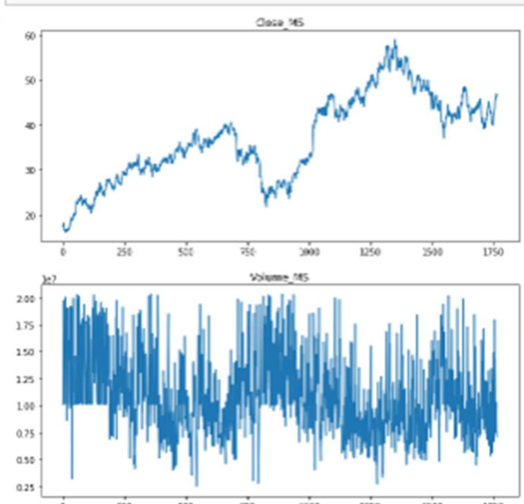
In [229]:
```python
# Model Evaluation
from sklearn.metrics import mean_squared_error

eval_results = pd.DataFrame(columns=['Column', 'RMSE', 'MAPE'])
tempResults = pd.DataFrame(columns=['Column', 'RMSE', 'MAPE'])

for col in MS_ts.columns:
    rmse = np.sqrt(mean_squared_error(test_MS_ts[col], forecast[col][:])).round(2)
    mape = np.round(np.mean(np.abs(test_MS_ts[col]-forecast[col][:])/test_MS_ts[col])*100,2)

    tempResults = pd.DataFrame({'Column':[col], 'RMSE': [rmse],'MAPE': [mape] })
    eval_results = pd.concat([eval_results, tempResults])

eval_results
```

Out[229]:

	Column	RMSE	MAPE
0	Close_MS	6.15	7.15
0	Volume_MS	3241873.70	29.02

In [230]:
```python
res.plot()
```

Out[230]:




```python
res.plot_forecast(1000)
```