THE GEORGE WASHINGTON UNIVERSITY

WASHINGTON, DC

DATS 6450_15: Time Series Analysis and Modeling

Instructor: Dr. Reza Jafari

Final Project Report

Harish Ram HR

5 May, 2021

Table of Contents

1. Abstract

Using time series modeling concepts and techniques, this project will focus on analyzing a multivariate time-dependent dataset in an attempt to create a predictive model that can forecast future values. The modeling techniques that will be used in this project will compare the accuracy, error and performance of several models. Each model will be assessed by looking at important statistical measures and tests of significance to evaluate the reliability and accuracy. After, the best model will be chosen and its predictive accuracy will be tested.

2. Introduction

Stock data has always been near impossible to predict with great accuracy because human error and decision making is a significant part of its fluctuation. However, by analyzing historical data, a model can be created that uses the behavior of that data to make future predictions. The data that is going to be used in this project is Google stock prices from 2006 to 2018 that was web-scrapped from Google finance. In order to create the best predictive model, pre-processing and cleaning the data is necessary to perform meaningful exploratory data analysis. Observing the trend of the data over a 12-year period is important when studying the trend and behavior of the data. Only after completely understanding the data can the models be created and evaluated. The statistical measures of the error of each model will help determine its precision. Using those measures will allow for proper model comparison and successful model selection.

3. Description of the Dataset

3.1 About the Data

Figure 3.1.1

Column Name	Data Type	Description
Date	String	In format: yy-mm-dd
Open	float	Price of the stock at market open ($USD)
High	float	Highest price reached in the day ($USD)
Low	float	Lowest price reached in the day ($USD)

Close	float	Price of the stock at market close ($USD)
Volume	int	Number of shares traded
Name	String	The stock's ticker name

Figure 3.1.1 shows the data dictionary of the Google stock dataset that will be used throughout this project. The date column is in year-month-day format and represents the date that the stock price was collected. The dataset contains almost every day from 2006-01-03 to 2017-12-29. The Close column is going to be the target variable throughout this project because it is the final price at which the stock is traded by the end of regular market hours (Chen & Mansa, 2020).

3.2 Correlation Matrix

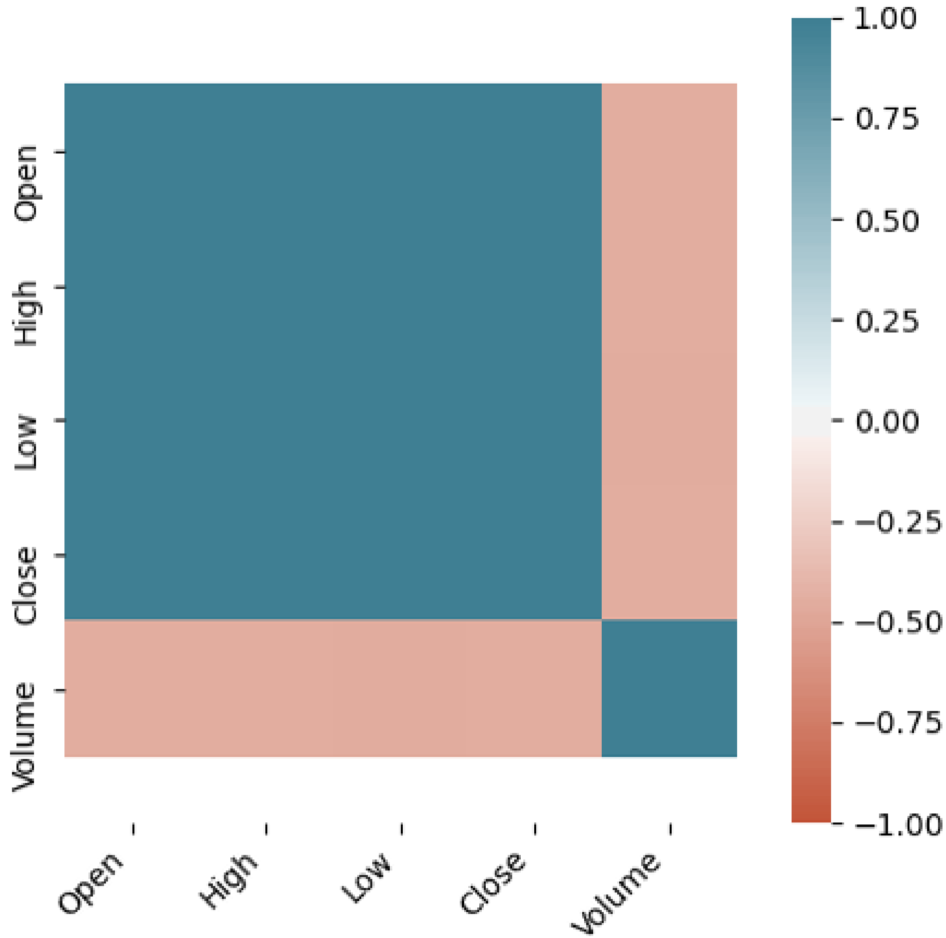


Figure 3.2.1

The correlation matrix above displays the correlation between all the variables in the dataset. The variables, Open, High, Low and Close show a dark shade of blue at the intersection indicating the relationship between those variables have a very high correlation coefficient. However, it is apparent the correlation between volume and the rest of the variables has a very low correlation coefficient given the color is pink at those intersections.

4. Time Series Analysis

4.1 Stationarity Analysis

In order to properly utilize time series data, it is imperative that the mean and variance of the Close price do not change over time. If these statistical components fluctuate across all the samples, it becomes difficult for further modeling of the target variable. There are several ways to observe the stationarity of data that involve plotting the data, performing an Augmented Dickey-Fuller test and plotting the respective autocorrelation coefficients.

Figure 4.1.1

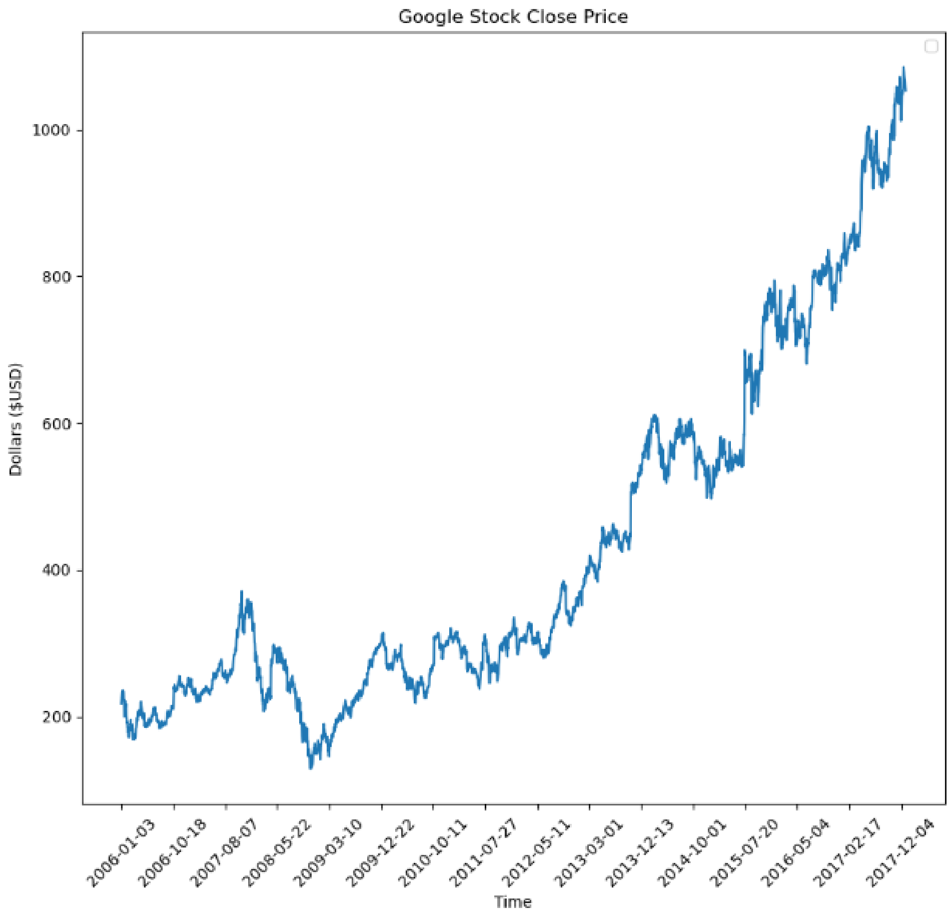


Figure 4.1.1 is a plot of the Google stock close price for the full time-period of the dataset. There is an overall, increasing trend across all the samples with several fluctuations indicating that the mean and variance of the data are not constant over time, but rather change very frequently with the data.

Figure 4.1.2

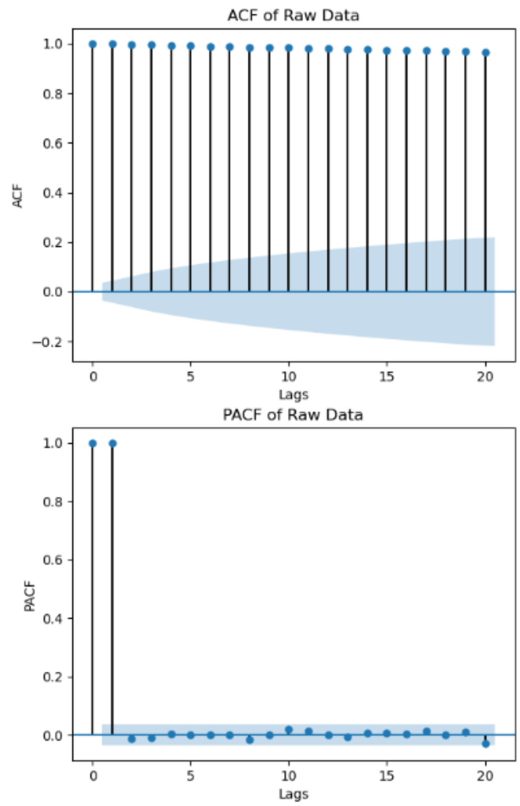


Figure 4.1.2 is an ACF plot with 20 lags of the target variable, close price. The stems in this plot are all very similar height after the first lag indicating that the autocorrelation coefficient between the present and lagged value is very high throughout all 20 lags. This ACF plot reinforces the observations made about the basic plot of the data. Since all the stems in each stem plot exceed the gray significance area, this implies that the current level of the stock prices is significantly autocorrelated with its lagged values. If that is the case then the data clearly has a time-dependent structure and is not stationary.

Figure 4.1.3

```
ADF Statistic: 1.322424
p-value: 0.996732
Critical Values:
    1%: -3.433
    5%: -2.863
    10%: -2.567
```

Figure 4.1.3 displays the ouput of the Augmented Dickey-Fuller or ADF test. The ADF-statistic is much larger than any of the critical values and the p-value is greater than 0.05; thus, the null hypothesis must be rejected indicating that the data has a clear time-dependent structure. This reinforces the observations made in figure 4.1.1 because the data increases with time.

Figure 4.1.4

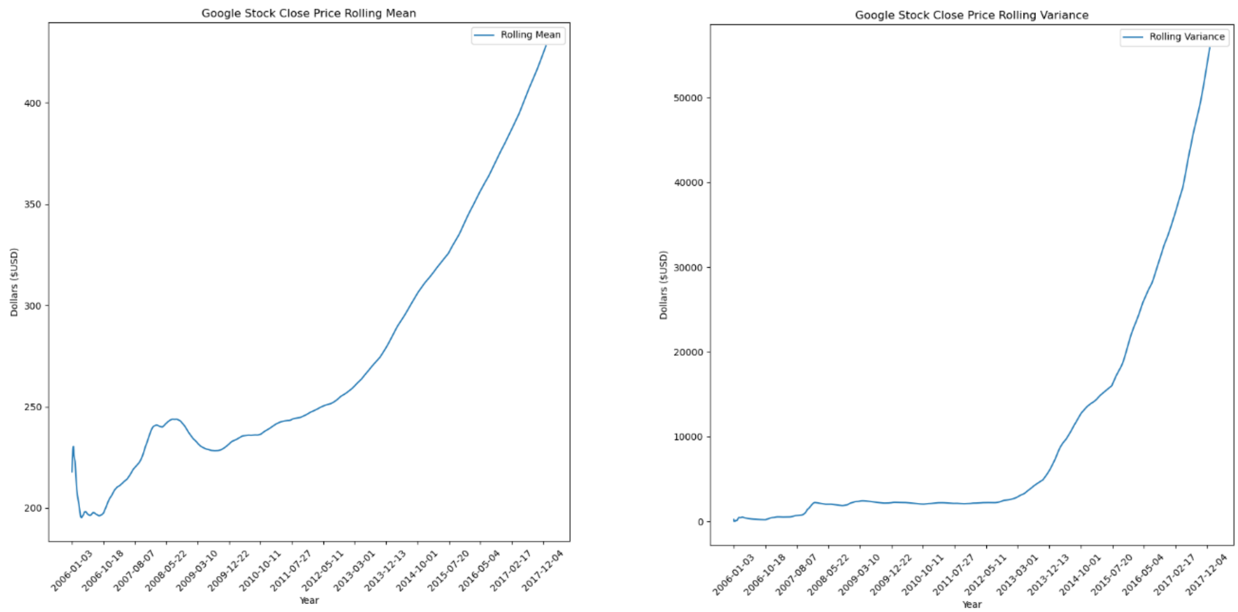
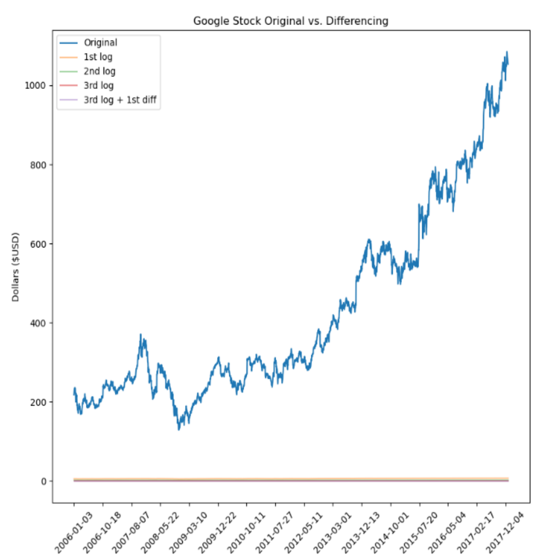


Figure 4.1.4 shows the rolling mean and variance of the data. Both lines are increasing over time indicating that the mean and variance have a time-dependent structure. By plotting the data, the ACF and PACF as well as the rolling mean and variance and observing the output of the ADF test, the data has been demonstrated as non-stationary. In order to make the data stationary, it is necessary to perform differencing or logarithmic transformation.

Figure 4.1.5




```
ADF Statistic: -0.187582
p-value: 0.939986
Critical Values:
    1%: -3.433
    5%: -2.862
    10%: -2.567
None
ADF Statistic: -0.514585
p-value: 0.889139
Critical Values:
    1%: -3.433
    5%: -2.863
    10%: -2.567
None
ADF Statistic: -0.669881
p-value: 0.854398
Critical Values:
    1%: -3.433
    5%: -2.863
    10%: -2.567
None
ADF Statistic: -10.350164
p-value: 0.000000
Critical Values:
    1%: -3.433
    5%: -2.863
    10%: -2.567
None
```

Figure 4.1.5 displays the results of applying 3 logarithmic transformations and an additional first differencing to the dataset. The plot of the original data is in blue while the transformed data is in the other colors. Although difficult to observe while all 4 lines are plotted in the same graph, it is clear that the transformed data expresses behavior that is independent of time as it almost looks like a straight line. The corresponding 4 ADF tests are shown to the right of the plot and after performing all the transformations, the ADF-statistic is finally less than the critical values and the p-value is less than 0.05 indicating that the transformed data does not have a unit root and lacks a time-dependent structure.

Figure 4.1.6

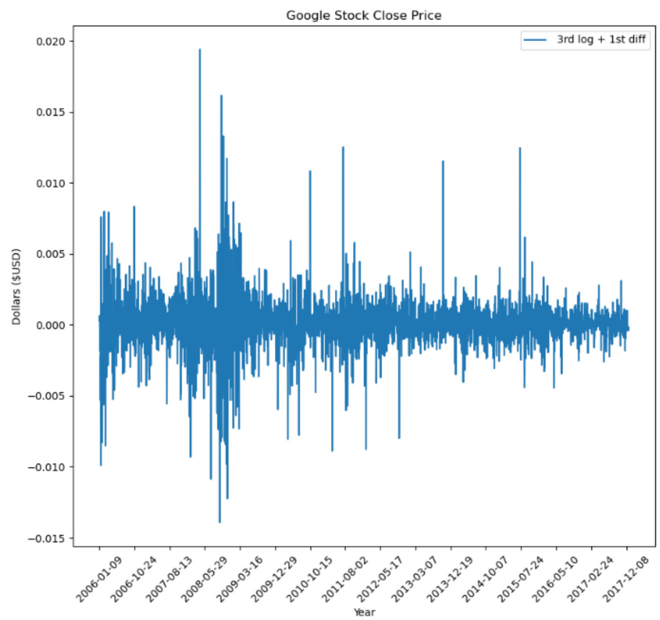
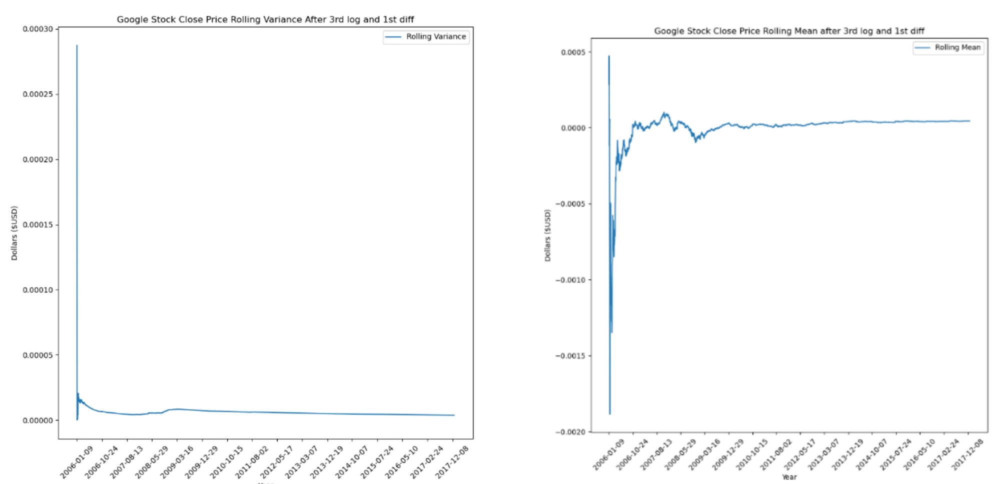


Figure 4.1.6 shows a closer view of the data after applying 3 logarithmic transformations followed by first differencing. The mean is constant throughout all the samples and the variance is seems stable as well.

Figure 4.1.7



The corresponding rolling mean and variance are shown in Figure 4.1.7. The behavior of these curves are significantly different than those representing the raw data. Both the rolling mean and variance flatten out over time indicating a lack of time-dependence and stationarity.

4.2 Time Series Decomposition

Time series decomposition allows for the separation of the trend and seasonality behaviors of the data. By subtracting the separated trend and seasonality from the original data, the resulting plot of the data will be adjusted for seasonality and trend. In order to calculate the trend and seasonality of the data, STL (Seasonal and Trend decomposition using Loess) must be performed on the non-stationary data.

Figure 4.2.1

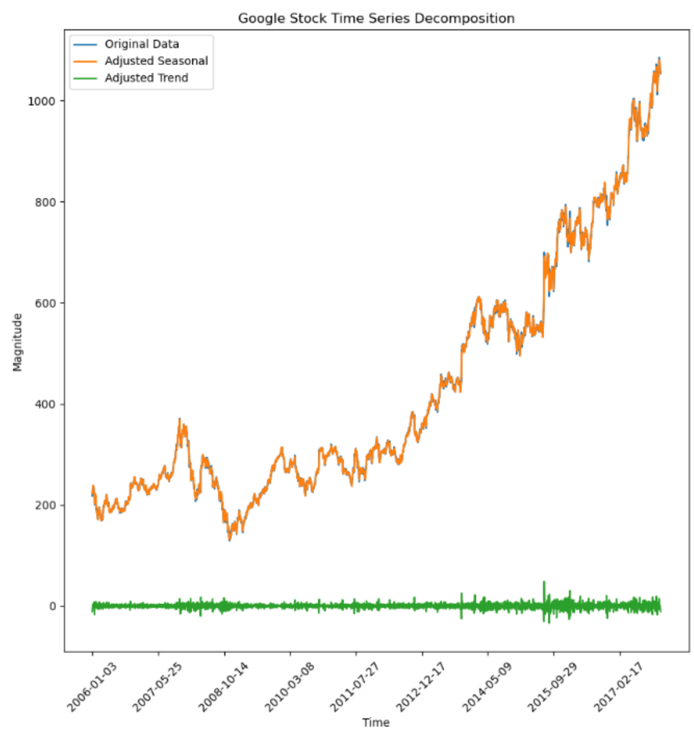


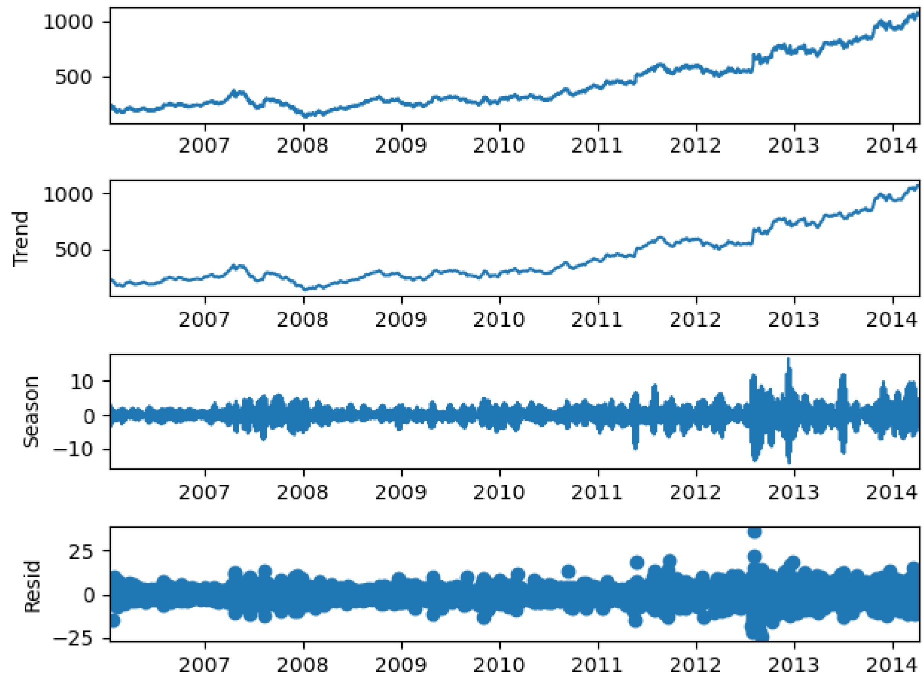
Figure 4.2.1 shows the original data plotted against the data that is adjusted for seasonality in orange and adjusted for trend in green. The orange line is almost exactly the same as the raw data because there was very weak seasonality in the raw data to begin with. The green line is almost linear throughout all the samples. This reinforces the idea that the raw data had a very strong trend; therefore, when using STL decomposition to extract the trend and subtract it from the raw data, the resulting line is stationary.

Figure 4.2.2

```
The strength of the trend for this data set is:  0.9997154404252081
The strength of the seasonality for this data set is:  0.36960409413665085
```

Figure 4.2.2 displays the calculated strength of trend and seasonality using STL decomposition. The strength of the trend is incredibly high – almost 1 – which reinforces our observations of the trend from plotting the raw data in the beginning. The strength of seasonality is very weak which explains why the adjusted seasonality plot is very similar to the raw data.

Figure 4.2.3



The plot of the full STL decomposition is shown in figure 4.2.3. The raw data is shown in the first graph, followed by the extracted trend, seasonality and residuals. The trend is clearly increasing while the seasonality stays constant for the most part with slight fluctuations towards the end. The residuals seem to not express any trends or irregular behavior.

5. Model Creation

From the exploratory data analysis, we have observed that the raw data has an increasing trend and very little seasonality. After performing several steps of stationarity analysis, the data is now stationary meaning that it does not have a time-dependent structure. To continue on with working towards creating a predictive model, it is necessary to use and test several base models to determine which one works best. Then, multiple linear regression will be used on the dataset to incorporate all the variables in the dataset. Finally, an Autoregressive Moving Average (ARMA) model will be generated by estimating the parameters using GPAC and the Levenburg Marquadt algorithm.

5.1 Base Models

In order to efficiently and properly generate base models, it is necessary to split 80% of the data into a training and 20% into a test set. The model will be created using the values from the training set and tested using the test set. If the predicted values from the model are similar to the values of the test set, then the model can be considered accurate. All of the following models are going to be trained with the training set and forecasted with the test set. The statistical components of each model will be discussed at the end of this section.

5.1.1 Average Method

The Average method of creating a model involves representing the entire training set with its average. This model will then use that average to make predictions of the test set.

Figure 5.1.1.1

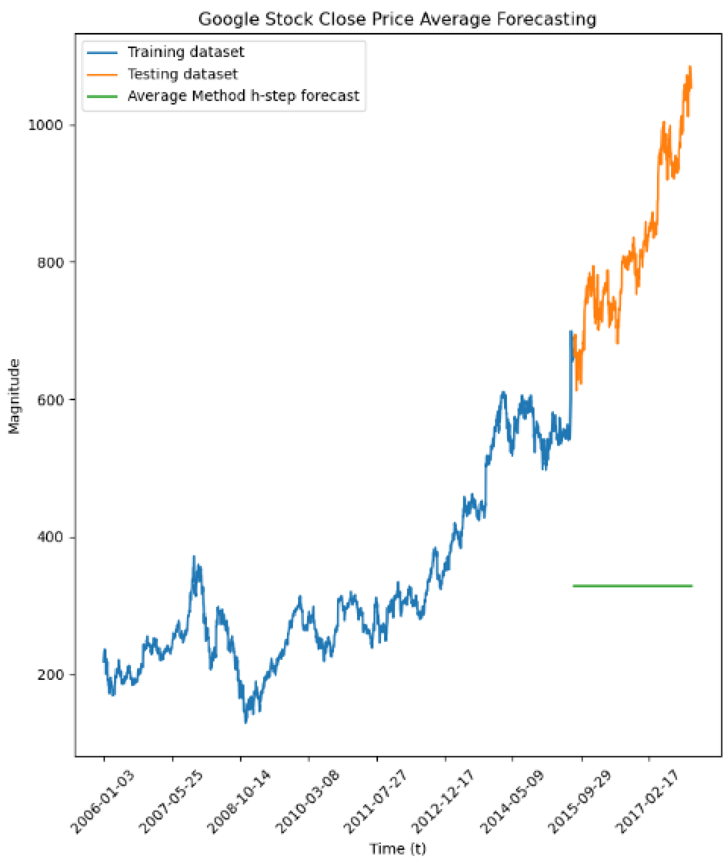
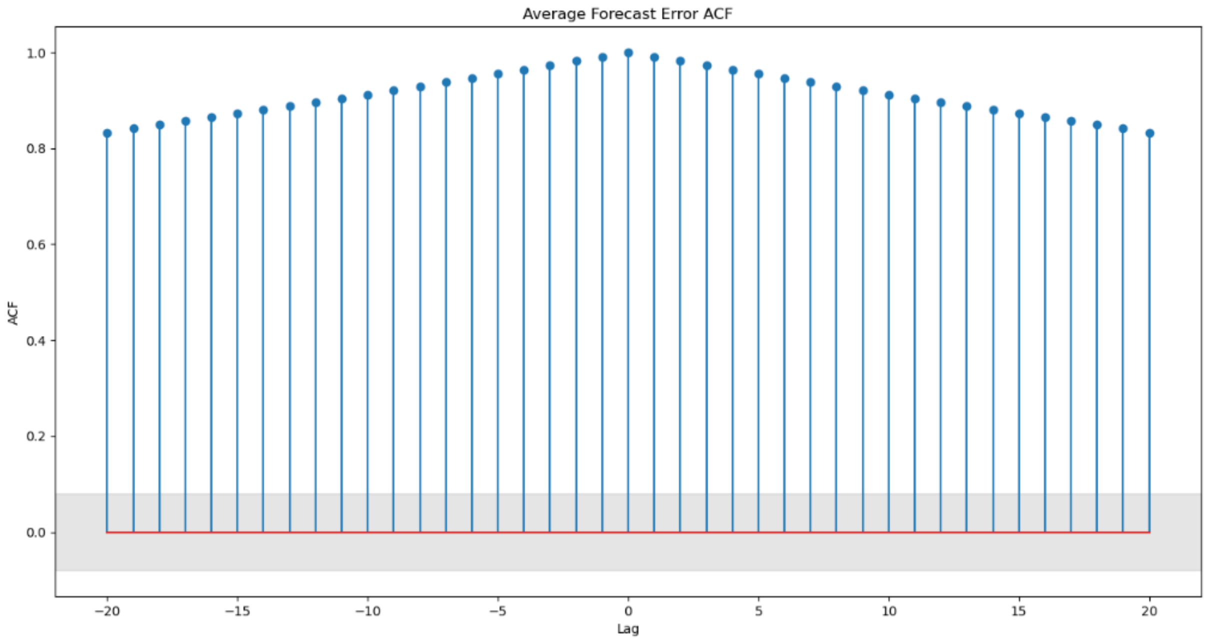


Figure 5.1.1.1 displays the plot of the training and test set with the h-step forecast generated by the Average model in green. The green line spans the entire duration of the test set. Since the forecast is the average of the training set, it makes sense that the green line is positioned around where the average of all the values in the training set would be. However, compared to the actual values in the test set, the Average model does poorly in predicting future values.

Figure 5.1.1.2 shows an ACF plot of the Average Model Forecast Error. The stems decaying slightly over time, but overall, they are very similar to one another in magnitude indicating a correlation between present and lagged values. This makes sense with the figure 5.1.1.1 because the h-step forecast is horizontal and linear. Since this plot does not express the same behavior as white noise, it is a good indication that the Average model is not a statistically significant model.

Figure 5.1.1.2



5.1.2 Naïve Model

The naïve method involves representing the test set with the most recent value of the training set.

Figure 5.1.2.1

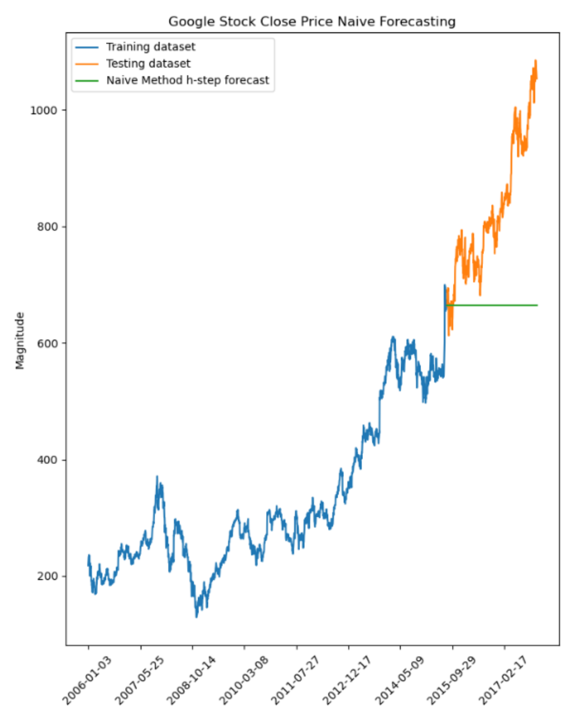


Figure 5.1.2.1 shows the training and test set plotted along with the forecast using the Naïve Method in the green line. The green line is positioned at the last value of the training set is on the y-axis and extends the entire length of the test set. This forecast is not accurate as it does not predict any specific value or trend in the test set.

Figure 5.1.2.2

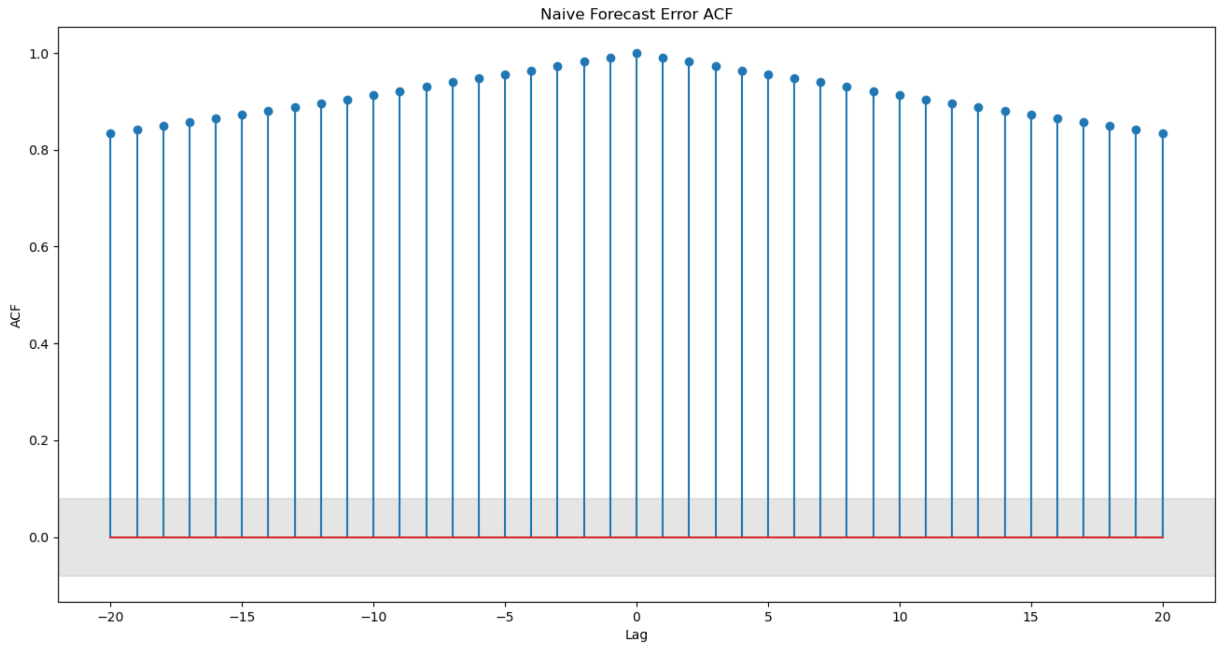
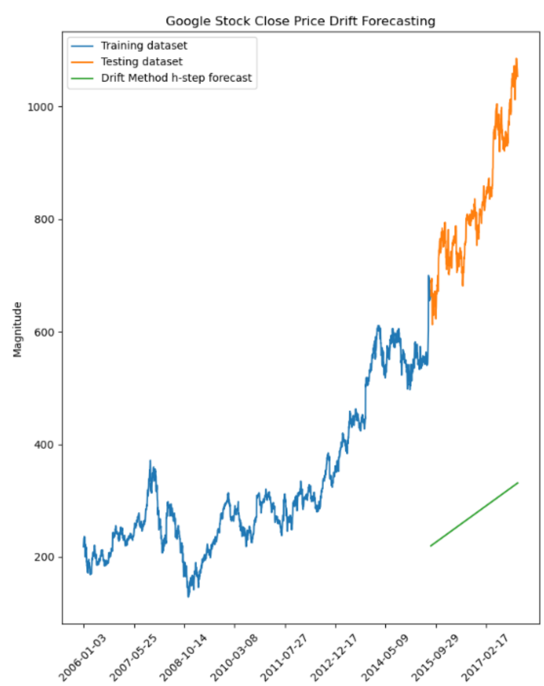


Figure 5.1.2.2 is the ACF plot of the Naïve model's Forecast Error. The autocorrelation coefficients are very similar to one another even as the number of lags increase which indicates that the present values are correlated to the past values of the error. Since this plot does not resemble the ACF of white noise, it is an indication that the Naïve method is not a good predictor of the data.

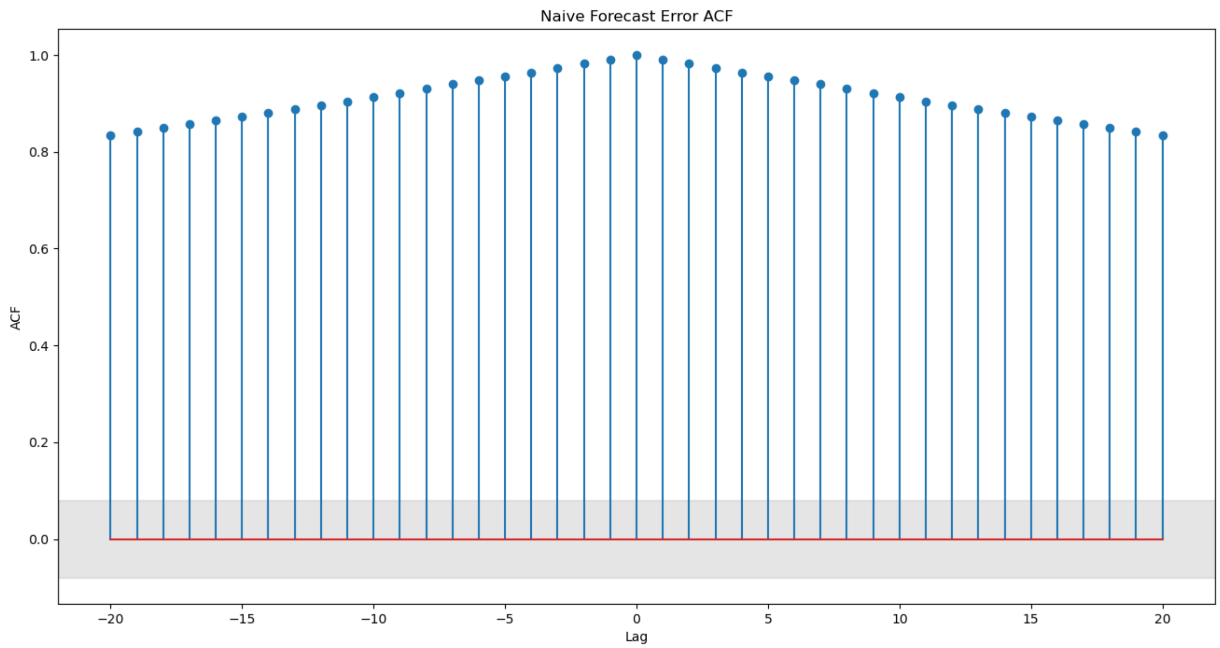
Figure 5.1.3.1

5.1.3 Drift Method



The drift method calculates the average change over time in the training data based on each step in the testing data. Figure 5.1.3.1 shows the training and test set of the raw data along with the h-step forecast using the Drift method in green. The forecast line is far away from the actual test set indicating the Drift method is not a great, predictive model for this dataset.

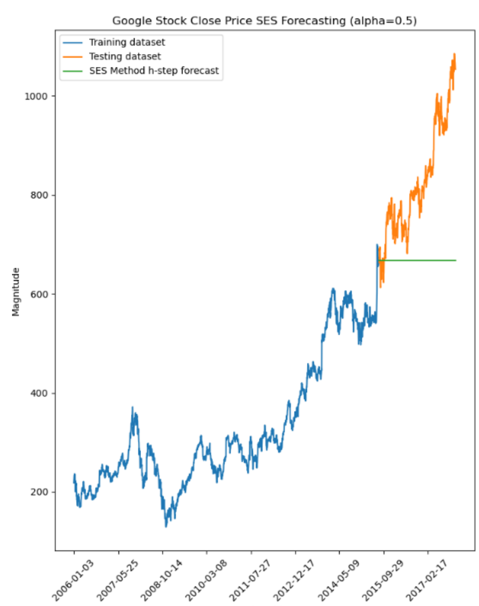
Figure 5.1.3.2



The ACF plot of the Naïve model's forecast error is shown in figure 5.1.3.2. Similar to the Average and Naïve method model, this plot shows clear autocorrelation between present and past values. Since this plot does not resemble the ACF of white noise, it is an indication that the Drift method is not a good predictor of the data.

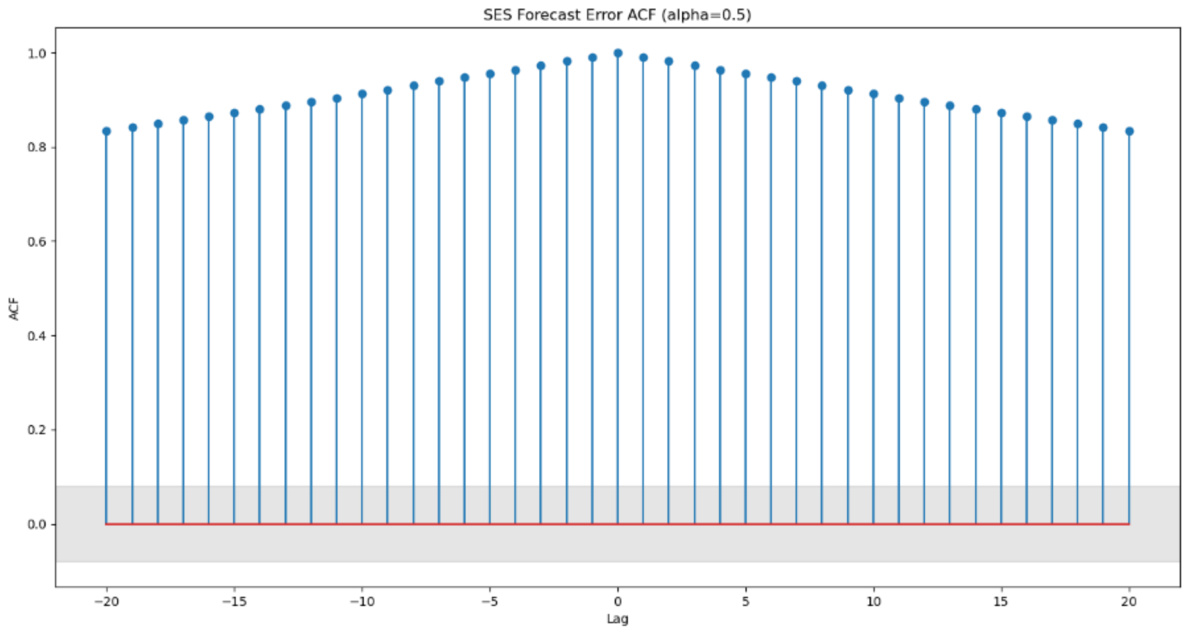
5.1.4 SES Method

Figure 5.1.4.1



The SES method of forecasting uses weighted averages. This means that older data has a different weight than newer data depending on the alpha value. It is important to test out different alpha values because having different weights on different time periods of data could elicit more accurate results. Figure 5.1.4.1 shows the forecast using the SES method with an alpha of 0.5 which makes the weight of new and older values the same. The forecast line in green is linear and does not model the test set accurately.

Figure 5.1.4.2



The ACF plot in figure 5.1.4.2 depicts the forecast error for the SES method using an alpha of 0.5. Similar to the previous base models, the autocorrelation between present and past values is very strong indicating that the residuals have an undesired trend. Also, the ACF plot does not show signs of mimicking the pattern of white noise which indicates that the forecast error does not indicate a good predictive model.

Figure 5.1.4.3

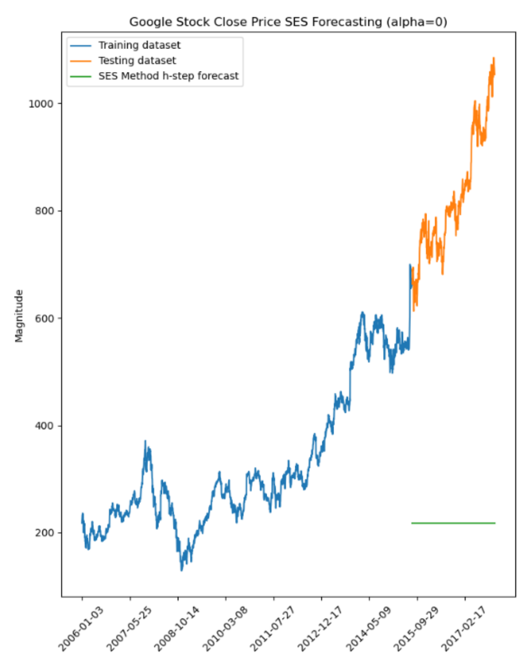
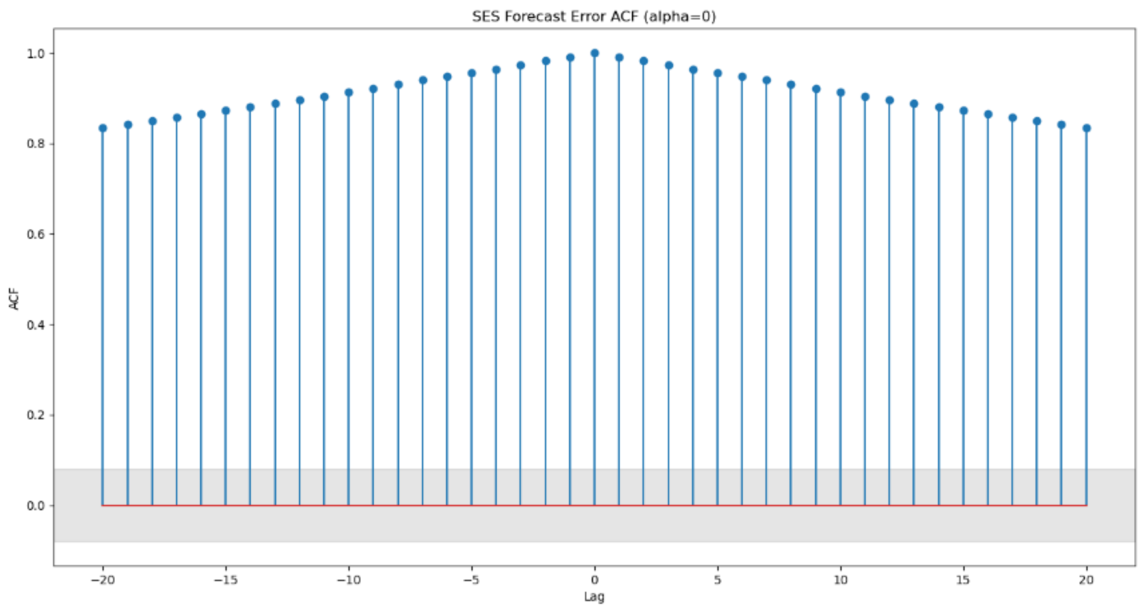


Figure 5.1.4.3 shows the forecast using the SES method, but with an alpha of 0. This means that 100% of the weight is placed on the older values while the more recent values have a 0 weight and are not considered at all in the forecast. The forecast line reinforces this calculation as it is very low on the y-axis. All in all, it still does not perform predictions very well. In comparison to the alpha value of 0.5, the forecast in this plot is further off from the test set.

Figure 5.1.4.4



The ACF plot above shows almost the same autocorrelation coefficient pattern as the ACF using the SES model with an alpha of 0.5. Since all the stems seem very auto correlated to one another, this is a clear depiction of a poor model.

Figure 5.1.4.5

Figure 5.1.4.5 shows the training set and testing set along with the forecast line in green. This forecast uses the SES method with an alpha of 0.99 which means that 99% of the weight will be placed on newer values while only 1% of the weight will be placed on older values. The forecast line in green resonates with the alpha value as it is positioned near the most recent value in the training set. Since the forecast in this plot demonstrates a linear line while the actual test set has many more fluctuations, the SES model cannot be considered successful at predicting stock data.

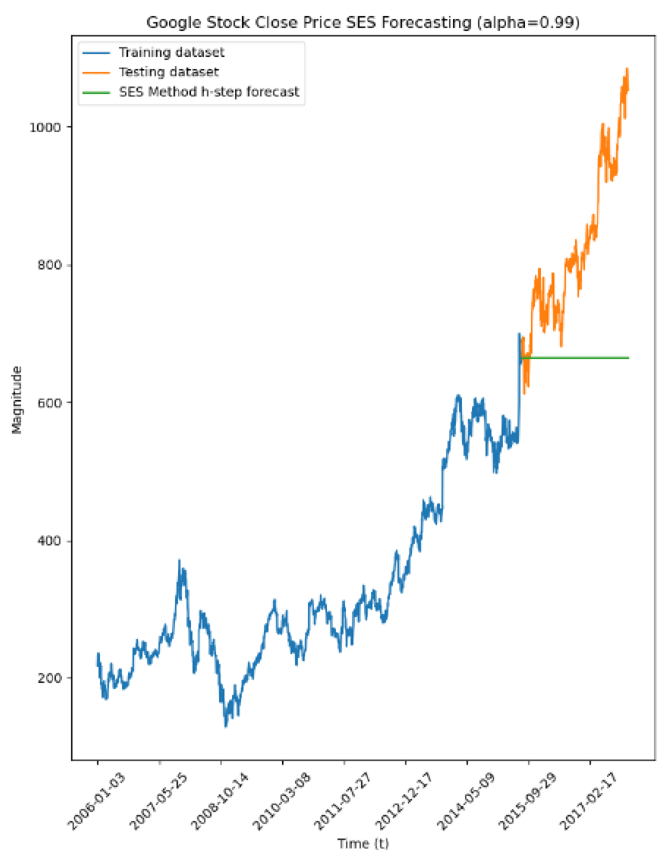
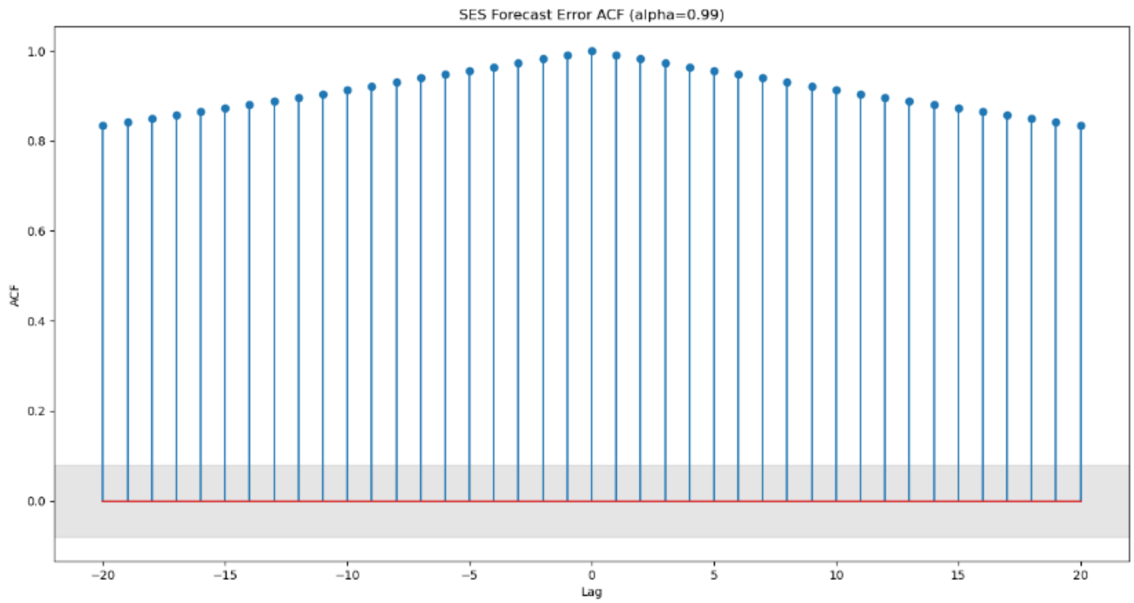


Figure 5.1.4.6



The figure above shows the ACF of the SES method's forecast error with an alpha of 0.99. This plot is almost the same as the previous two plots which indicates that regardless of the alpha value, the forecast using the SES method is very poor. In this case as well as previous models, the stems have very similar autocorrelation coefficients which means that present values and lagged values are highly correlated. This makes sense since the forecast line is linear; therefore, the error has a trend. Also, this ACF plot does not contain characteristics of white noise in which most stems besides at lag 0 would be in the insignificant, gray region.

5.2 Holt-Winters

The holt-winters method of modeling takes into account the seasonality and the trend of the data. Since the variance of the dataset is increasing across all the samples, the trend will be set to additive as opposed to multiplicative. From the observations made in the time series decomposition section, seasonality is very weak, so the seasonal parameter is set to none. The damp parameter will be tested with true and false to determine which option produces the best prediction.

Figure 5.2.1

The plot to the right shows the training set in blue and test set in orange. The green line indicates the forecast made using the holt-winters model with the parameters defined above but with the damp parameter set to false. The damp parameter dampens the trend to a flat line. In comparison to the previous models, the forecast line is moving in the direction in which the test set trend is moving which means that is more successful than the base models. However, since the values are not predicted with precision, holt-winters is still a poor model to reflect the data.

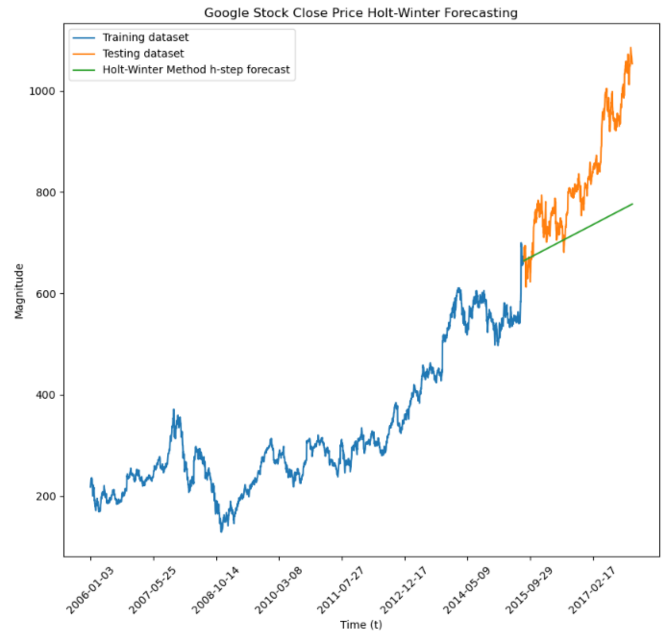


Figure 5.2.2

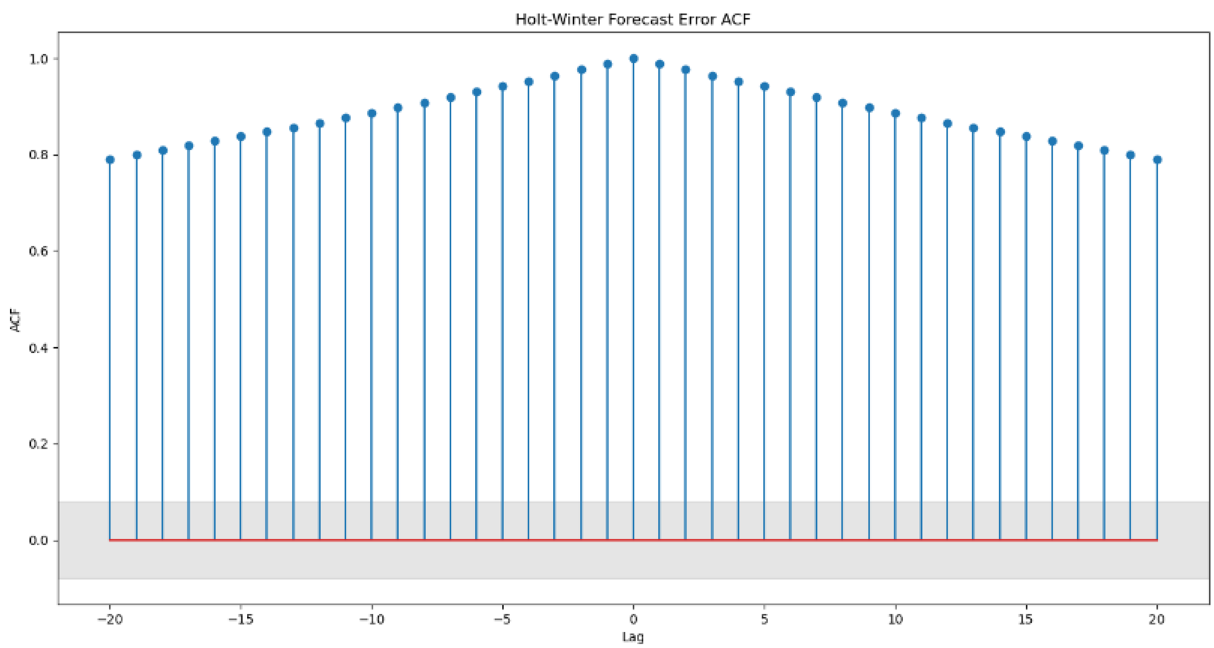


Figure 5.2.2 shows the ACF plot of the holt-winter's model forecast error. In comparison to the ACF plot of the base models, the stems in this plot seem to decay relatively faster. This makes sense since the forecast line is increasing. Therefore, the last value in the forecast line will be less correlated with a lagged value at lag 20 than lag 2. However, this is still a poor reflection of the model as the forecast error ACF should have characteristics of white noise.

Figure 5.2.3

The forecast plot for the holt-winters model is shown to the right with the forecast line in green. In this plot, the "damp" parameter is set to true which means that the forecast line in figure 5.2.1 was simply damped to become a linear line. It is clear that this is a worse depiction of the testing set as it does not reflect any fluctuations or variance in the data.

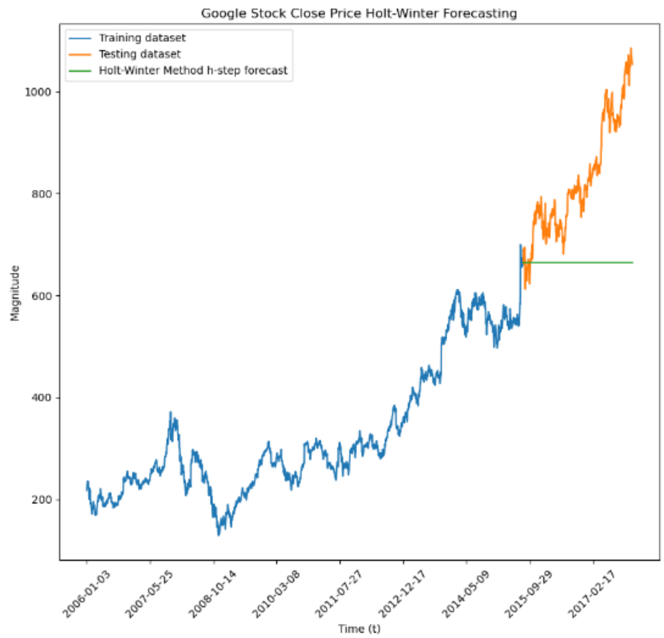


Figure 5.2.4

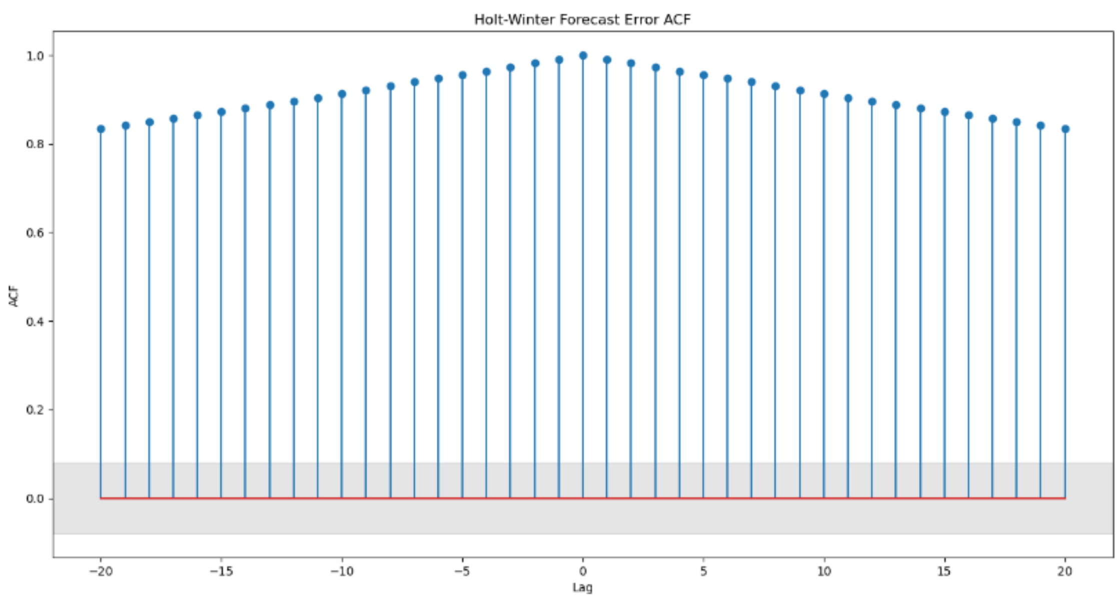


Figure 5.2.4 shows the ACF plot of the forecast error of the holt-winters model with damp set to true. Although very slightly, the stems seem to decay at a relatively slower rate as when the damp was set to false. However, since this ACF plot does not match the characteristics of an ACF plot of white noise, this is an indication this model poorly predicts the data.

5.3 Feature Selection

Feature selection involves running OLS (Ordinary Least Squares) Regression on all the variables in the dataset in an effort to determine which predictor variables are the best at predicting the target variable which in this case is Close price. The process of feature selection requires the removal of any statistically insignificant predictors. Once all the unimportant features are removed, then multiple linear regression can be performed on the data.

Figure 5.3.1

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Close   R-squared:                       1.000
Model:                            OLS   Adj. R-squared:                  1.000
Method:                 Least Squares   F-statistic:                 2.591e+06
Date:                Tue, 04 May 2021   Prob (F-statistic):               0.00
Time:                        16:58:04   Log-Likelihood:                -5065.6
No. Observations:                2415   AIC:                         1.014e+04
Df Residuals:                    2410   BIC:                         1.017e+04
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
one           -0.3672      0.152     -2.416      0.016      -0.665      -0.069
Open          -0.5393      0.016    -34.314      0.000      -0.570      -0.508
High           0.7541      0.017     45.535      0.000       0.722       0.787
Low            0.7864      0.015     53.514      0.000       0.758       0.815
Volume      4.268e-08   1.61e-08      2.658      0.008    1.12e-08    7.42e-08
==============================================================================
Omnibus:                      219.501   Durbin-Watson:                   2.121
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1292.759
Skew:                           0.188   Prob(JB):                    1.91e-281
Kurtosis:                       6.565   Cond. No.                     1.94e+07
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.94e+07. This might indicate that there are
strong multicollinearity or other numerical problems.
```

The OLS Regression model summary is shown above. In order to determine the performance of each predictor, a t-test must be performed on each variable by looking at the p-value and comparing it to 0.05. The confidence interval must also be looked at to check if there is a 0 in the interval which indicates statistical insignificance. Finally, after removing the feature the R-squared, adjusted R-squared, AIC, BIC and condition number should be checked for any changes. In figure 5.3.1, there is a constant that was inputted into the data; however, since all the features seem significant through the t-test and confidence interval, it is necessary to remove it as it is not a meaningful feature and may influence other values. Also, the condition number is incredibly high and the goal of feature selection is to get it as low as possible.

Figure 5.3.2

```
Singular Values of Original [6.33556262e+16 5.52022655e+08 1.25644198e+04 9.22182023e+03]
The condition number of original =  2621104.108949624
Calculated Coefficients using LSE: [-3.67222714e-01 -5.39257799e-01  7.54067887e-01  7.86421613e-01
  4.26827367e-08]
```

Figure 5.3.2 shows the calculated Singular Values and Condition number as well as the coefficients of each feature to the target variable using LSE. Since there is an absence of 0 values in the Singular Values, the features seem to not be correlated to one another. However, that is not true because the correlation matrix proved otherwise. The condition number is incredibly high which means that there is a multi-collinearity problem in the dataset. This means that the target variables is highly correlated to multiple variables.

Figure 5.3.3

```
                            OLS Regression Results
=======================================================================================
Dep. Variable:                  Close   R-squared (uncentered):                   1.000
Model:                            OLS   Adj. R-squared (uncentered):              1.000
Method:                 Least Squares   F-statistic:                          1.925e+07
Date:                Tue, 04 May 2021   Prob (F-statistic):                        0.00
Time:                        16:58:04   Log-Likelihood:                         -5068.5
No. Observations:                2415   AIC:                                  1.014e+04
Df Residuals:                    2411   BIC:                                  1.017e+04
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Open          -0.5372      0.016    -34.198      0.000      -0.568      -0.506
High           0.7559      0.017     45.644      0.000       0.723       0.788
Low            0.7817      0.015     53.613      0.000       0.753       0.810
Volume      1.777e-08   1.23e-08      1.442      0.149    -6.4e-09    4.19e-08
==============================================================================
Omnibus:                      230.703   Durbin-Watson:                   2.119
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1273.455
Skew:                           0.266   Prob(JB):                    2.97e-277
Kurtosis:                       6.518   Cond. No.                     2.62e+06
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.62e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Figure 5.3.3 shows the OLS Regression model summary after the constant was removed as one of the features. The AIC, BIC, R-squared and Adjusted R-squared remain unchanged; however, the condition number decreased about almost 1000. Furthermore, the Volume variable now does not pass the t-test as the p-value is greater than 0.05 and there is a 0 in the confidence interval. This indicates that Volume is statistically insignificant and should be removed. As seen in the correlation matrix, Volume had a very week correlation with the target variable.

Figure 5.3.4

```
                                 OLS Regression Results
=======================================================================================
Dep. Variable:                  Close   R-squared (uncentered):                   1.000
Model:                            OLS   Adj. R-squared (uncentered):              1.000
Method:                 Least Squares   F-statistic:                          2.565e+07
Date:                Tue, 04 May 2021   Prob (F-statistic):                        0.00
Time:                        16:58:05   Log-Likelihood:                         -5069.5
No. Observations:                2415   AIC:                                  1.015e+04
Df Residuals:                    2412   BIC:                                  1.016e+04
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Open          -0.5379      0.016    -34.254      0.000      -0.569      -0.507
High           0.7675      0.014     52.998      0.000       0.739       0.796
Low            0.7708      0.012     61.836      0.000       0.746       0.795
==============================================================================
Omnibus:                      220.633   Durbin-Watson:                   2.119
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1253.738
Skew:                           0.215   Prob(JB):                    5.68e-273
Kurtosis:                       6.504   Cond. No.                         299.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Figure 5.3.4 shows the OLS regression model summary after Volume was removed from the features list. All the p-values for each variable are below 0.05 indicating that each feature passes the t-test and there is an absence of 0 in the corresponding confidence intervals which means that Open, High and Low are statistically significant features. The R-squared and adjusted R-squared value are both still at 1.0 which means that 100% of the variation in the target variable can be explained by these features. The AIC increased by 0.001 since the first iteration while the BIC decreased by 0.001. These values should be minimized. The Prob(F-statistic) is below 0.05 which means this model passes the F-test demonstrating that it has better performance than an intercept only model. Thus, the strength of the relationship between the features and dependent variable in our model is statistically significant. Finally, the initial condition number was 1.94E7 while the final condition number was reduced to 299 which shows significance.

5.4 Multiple Linear Regression

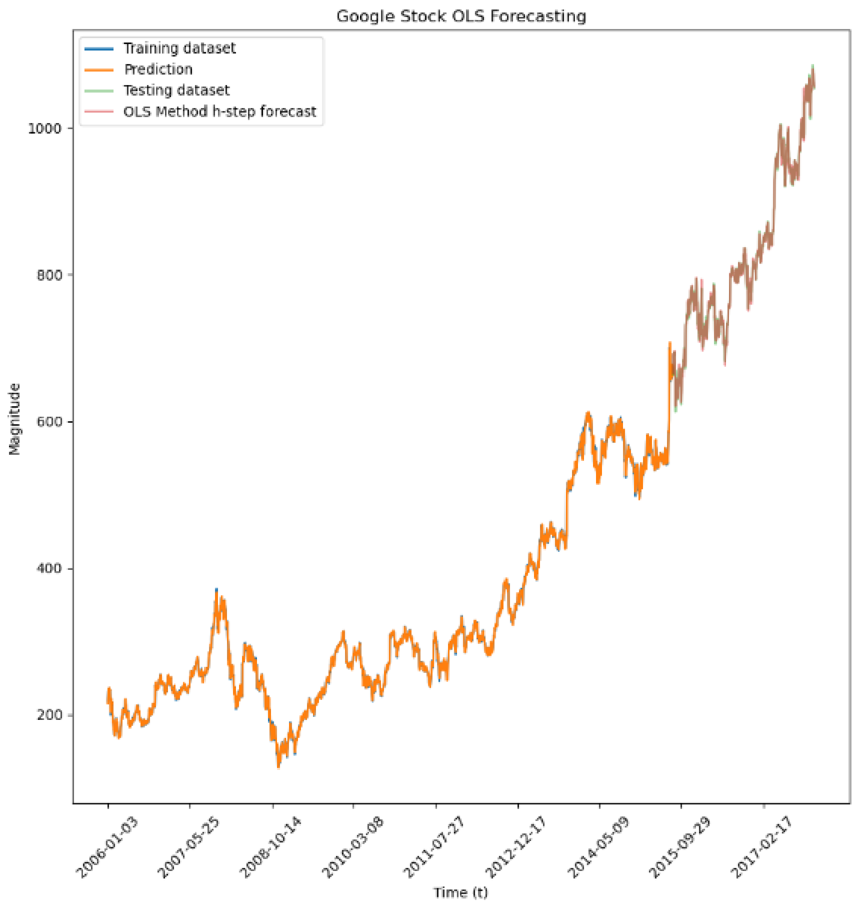
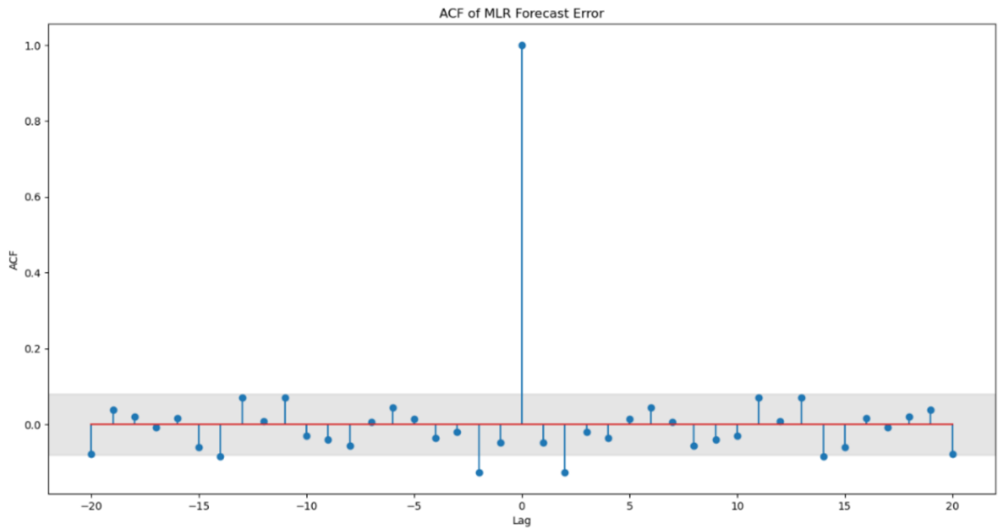


Figure 5.4.1

Figure 5.4.1 shows the training set and testing set of the raw data with the prediction and forecast calculated using Multiple Linear Regression. The orange and brown lines signifying the prediction and forecast almost completely match the original data behind. This shows that the model predicts the real values with extreme accuracy. Multiple Linear Regression uses the other features in the dataset as predictors for the target variable. Since the other features were highly correlated to the target variable, the resulting prediction and forecast are very accurate.

Figure 5.4.1



In figure 5.4.1, the ACF plot of the Forecast Error from the Multiple Linear Regression model shows a distinct resemblance to the ACF of white noise. This means that the forecast error has no trend or correlated behavior. Almost all the stems are inside the insignificant region indicating that the present values have an insignificant correlation to the lagged values. Thus, multiple linear regression is a great model that accurately reflects the data properly.

5.5 Generalized Partial Autocorrelation

Figure 5.5.1

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	-0.0210507	-0.00357612	-0.0207024	0.00786397	0.0086275	-0.0362376	-0.00145672	0.069901	-0.0269053	-1.26006e-05	-0.0177638	0.0560873	-0.00379617	-0.0285336	-0.00915505
1	0.148755	0.118286	-0.0220602	0.030575	0.0416556	-0.0365839	-1.74032	0.069343	-0.026938	37.93	-0.0178035	0.0548888	-0.425366	-0.0273166	0.0178817
2	6.56304	1.06367	-0.0328129	-0.0562235	-0.0611418	-0.0176437	-0.000302994	0.0639618	-0.145652	-0.13117	-0.279626	0.0519405	-0.0339204	-0.039193	-0.150167
3	-0.425325	-0.554477	0.855538	0.0450769	-0.0211033	-0.0171001	-3.72334	0.0638928	0.108136	0.180403	0.0786874	0.0421316	-0.0891487	-0.0956978	-0.0134292
4	0.962816	-1.55284	1.64647	0.766729	-0.0450223	0.124469	-0.117705	0.0691855	-0.0775934	0.0683671	-0.0450871	0.0735747	-0.0155119	-0.113734	0.643914
5	-4.30094	-3.39096	0.977948	-0.781063	-2.44756	-0.228006	-0.0468558	0.0801207	0.027107	0.0365879	0.130402	0.080669	-0.461101	-0.0971822	-0.197543
6	0.0094007	-1.92269	-0.121701	-6.96741	-6.33035	1.19635	-0.281531	0.0776151	-0.118816	-0.00831998	0.0423054	0.00879854	0.0272901	0.0392206	-0.0323482
7	-204.964	-1.92295	123.651	-7.8432	-4.24148	-8.10975	-1.93342	0.0641372	-0.148074	-0.749859	0.0434743	-0.108107	0.0138967	0.0708127	-0.015508
8	-0.400851	-0.156817	-0.398117	0.562043	0.706918	-0.243873	-0.354917	-0.710989	-0.0680955	0.0152848	0.0949991	-0.0338534	0.284254	0.0683612	0.177925
9	-0.0104064	0.777851	-0.620216	1.09866	0.933127	-1.24174	0.0439403	-0.300174	-0.154293	0.447145	0.0885437	0.514028	0.125862	0.0427493	-0.0423249
10	-72.8172	0.728396	-1.38891	0.595722	0.465289	-1.37685	-9.48911	-0.313225	0.946871	-0.416114	-0.0878496	0.163154	0.0285772	0.0894362	-0.0404778
11	-2.8323	-8.0995	-1.77612	1.09232	1.34911	-1.02794	0.371083	0.933889	-0.0355156	-0.802046	-1.62741	0.173658	-0.366101	0.0753111	-0.0214404
12	-0.0838334	-0.572136	-0.387419	-0.736329	-0.231818	-0.90452	3.05922	0.990077	-21.9165	-0.868883	-0.127538	0.394523	0.127273	0.0588503	-0.890329
13	6.53884	-0.519378	0.462791	-0.532137	3.1349	-0.998655	0.938627	-3.08431	2.2663	-1.13696	-2.60469	0.342195	-0.118573	0.479701	-0.291743
14	0.264046	0.0178834	-0.847803	0.223913	0.67298	-1.3053	-1.33609	-0.246097	0.558948	0.464917	0.296515	0.222486	2.55559	0.995397	-0.203493

The GPAC (Generalized Partial Autocorrelation) table is shown above with the highlighted patterns in red. There is a column of constants at both k = 8 and k =12 while j = 0 is filled entirely with 0 values. The first pattern is at k = 8 and j = 0 indicating an AR order of 8 and MA order of 0. The second pattern is at k = 12 and j = 0 indicating an AR order of 12 and MA order of 0. These patterns estimate a possible ARMA(8,0) and an ARMA(12,0).

5.6 Levenburg Marquadt Algorithm Diagnostic Analysis

Figure 5.6.1

```
                              ARMA Model Results
==============================================================================
Dep. Variable:                  Close   No. Observations:                 3019
Model:                     ARMA(8, 0)   Log Likelihood              -10126.203
Method:                       css-mle   S.D. of innovations              6.903
Date:                Tue, 04 May 2021   AIC                          20270.406
Time:                        16:58:07   BIC                          20324.520
Sample:                             0   HQIC                         20289.864

================================================================================
                   coef    std err          z      P>|z|      [0.025      0.975]
--------------------------------------------------------------------------------
ar.L1.Close      1.0286      1e-05   1.03e+05      0.000       1.029       1.029
ar.L2.Close     -0.0145      0.019     -0.778      0.437      -0.051       0.022
ar.L3.Close     -0.0479      0.028     -1.721      0.085      -0.102       0.007
ar.L4.Close      0.0166      0.027      0.618      0.537      -0.036       0.069
ar.L5.Close      0.0079      0.026      0.304      0.761      -0.043       0.059
ar.L6.Close     -0.0041      0.026     -0.159      0.874      -0.054       0.046
ar.L7.Close      0.0238      0.025      0.968      0.333      -0.024       0.072
ar.L8.Close     -0.0105      0.018     -0.583      0.560      -0.046       0.025
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            1.0000           -0.0000j            1.0000           -0.0000
AR.2           -1.6161           -0.7871j            1.7976           -0.4279
AR.3           -1.6161           +0.7871j            1.7976            0.4279
AR.4           -0.2879           -1.8733j            1.8953           -0.2743
AR.5           -0.2879           +1.8733j            1.8953            0.2743
AR.6            1.3062           -1.2748j            1.8252           -0.1231
AR.7            1.3062           +1.2748j            1.8252            0.1231
AR.8            2.4692           -0.0000j            2.4692           -0.0000
-----------------------------------------------------------------------------
```

The LM algorithm model summary for the first pattern of ARMA(8,0) is shown above. The parameter estimated coefficients are shown on the left of the figure 5.6.1 and do not resemble the coefficients seen in the GPAC table. The standard deviation shown to the right of the coefficients and all values are below 0.05 while almost all the p-values except for the p-value for AR(1) are above 0.05 indicating the t-test was fauked. There is a 0 in the confidence interval of every coefficient besides AR(1) in which the interval is the same value. This shows that the model is biased. Since the MA order is 0, there are no zeros/pole cancellations or simplifications. The Q value that was calculated from the error of model.predict() from the LM algorithm is less than the chi-critical value, so we can reject the null hypothesis and conclude that the residuals and forecast error are white.

Figure 5.6.2

```
Dep. Variable:                      y   No. Observations:                 3015
Model:                    ARMA(12, 0)   Log Likelihood               14604.702
Method:                       css-mle   S.D. of innovations              0.002
Date:                Tue, 04 May 2021   AIC                         -29183.404
Time:                        21:01:16   BIC                         -29105.256
Sample:                             0   HQIC                        -29155.302

==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1.y       -0.0173      0.018     -0.950      0.342      -0.053       0.018
ar.L2.y       -0.0009      0.018     -0.050      0.960      -0.037       0.035
ar.L3.y       -0.0194      0.018     -1.066      0.287      -0.055       0.016
ar.L4.y        0.0042      0.018      0.233      0.816      -0.031       0.040
ar.L5.y        0.0095      0.018      0.522      0.602      -0.026       0.045
ar.L6.y       -0.0339      0.018     -1.870      0.061      -0.069       0.002
ar.L7.y        0.0004      0.018      0.021      0.983      -0.035       0.036
ar.L8.y        0.0691      0.018      3.808      0.000       0.034       0.105
ar.L9.y       -0.0252      0.018     -1.379      0.168      -0.061       0.011
ar.L10.y       0.0005      0.018      0.029      0.977      -0.035       0.036
ar.L11.y      -0.0168      0.018     -0.918      0.359      -0.053       0.019
ar.L12.y       0.0576      0.018      3.141      0.002       0.022       0.093
                                    Roots
==============================================================================
                  Real          Imaginary           Modulus         Frequency
------------------------------------------------------------------------------
AR.1           -1.1995           -0.0000j            1.1995           -0.5000
AR.2           -1.0413           -0.6759j            1.2414           -0.4084
AR.3           -1.0413           +0.6759j            1.2414            0.4084
AR.4           -0.6844           -1.1382j            1.3282           -0.3362
AR.5           -0.6844           +1.1382j            1.3282            0.3362
AR.6            0.0040           -1.2102j            1.2102           -0.2495
AR.7            0.0040           +1.2102j            1.2102            0.2495
AR.8            0.6960           -1.0795j            1.2844           -0.1589
AR.9            0.6960           +1.0795j            1.2844            0.1589
AR.10           1.2839           -0.0000j            1.2839           -0.0000
AR.11           1.1297           -0.6643j            1.3105           -0.0846
AR.12           1.1297           +0.6643j            1.3105            0.0846
```

The LM algorithm model summary for the first pattern of ARMA(12,0) is shown above. The coefficient shown in the GPAC at k = 12 reflects the coefficient of AR(12) in the model summary. The entire column of standard errors are the same value, while the p-value for every coefficient besides AR(8) and AR(12) is above 0.05 indicating statistical insignificance. There is a 0 in a every confidence interval besides at AR(12) indicating that the model is biased. Since the MA order is 0, there are no zeros/pole cancellations or simplifications. The Q value that was calculated from the error of model.predict() from the LM algorithm is less than the chi-critical value, so we can reject the null hypothesis and conclude that the residuals and forecast error are white.

5.7 ARMA

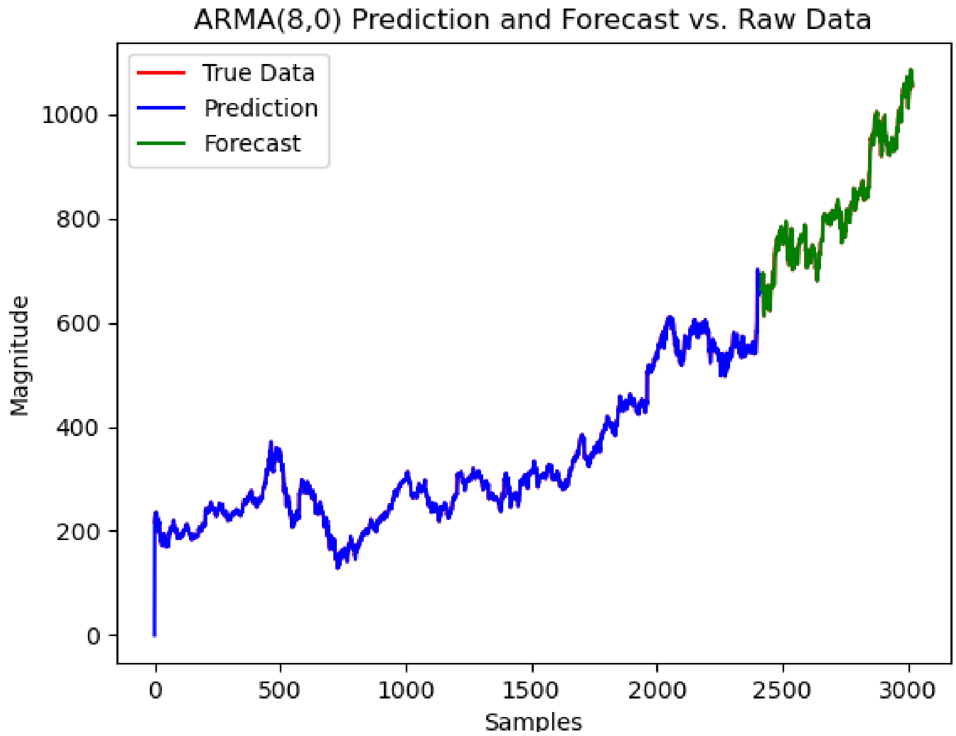


Figure 5.7.1

The raw data with the prediction and forecast of ARMA(8,0) is shown in in figure 5.7.1. The prediction and forecast values accurately predict the values of the test set. The fluctuations in both the prediction and forecast reflect the true data.

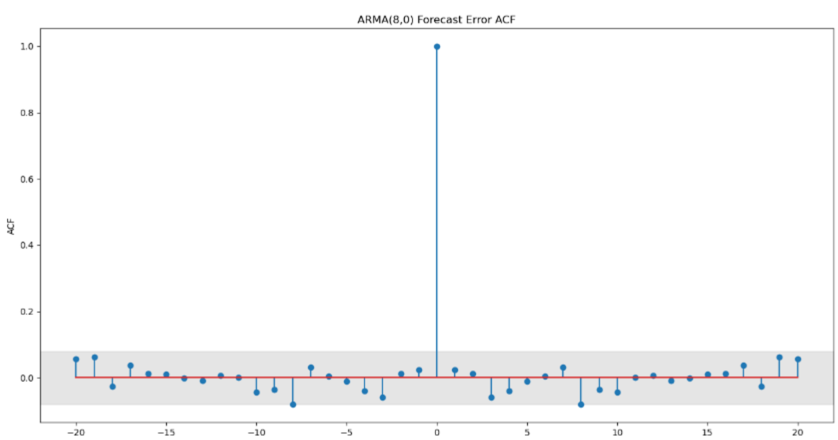


Figure 5.7.2

The ACF plot of the forecast error of ARMA(8,0) is shown in figure 5.7.2. This ACF plot resembles an ACF plot of white noise very well indicating that the model successfully predicted the actual values with great accuracy and little error.

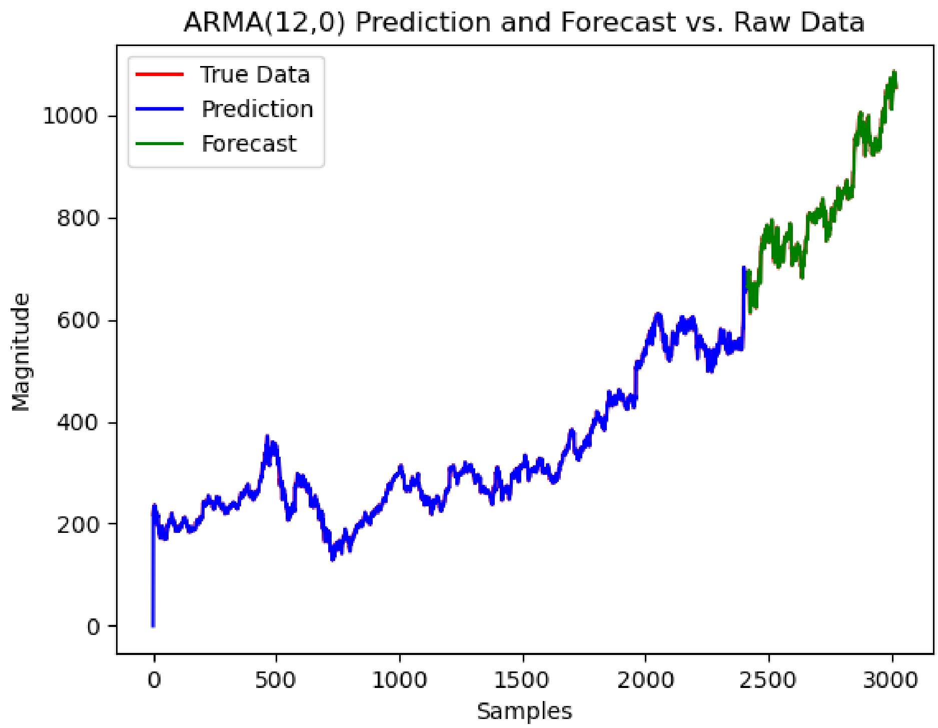


Figure 5.7.3

In figure 5.7.3, the plot of the prediction and forecast function of ARMA(12,0) is compared to the true data. The predictions and forecast reflect the behavior of the original data so well that there is no red line that can be seen. Overall, this model reflects the data very well.

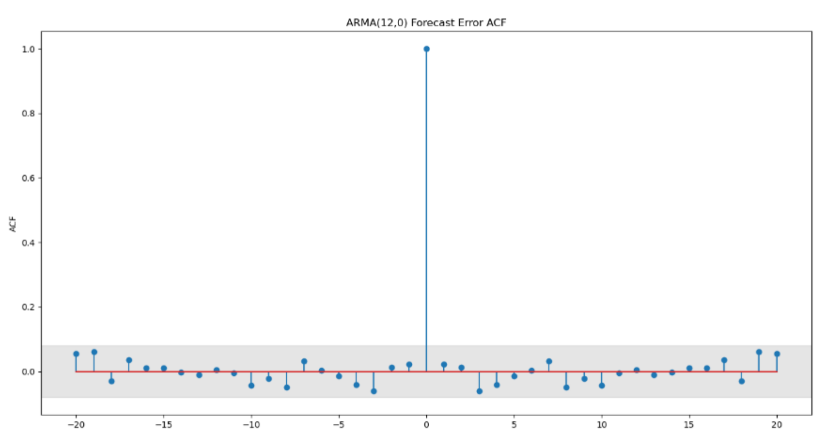


Figure 5.7.4

The ACF plot in figure 5.7.4 reflects the forecast error of ARMA(12,0). This plot resembles the ACF of white noise which indicates that the forecast error has little irregular behavior. Almost all the stems are within the insignificant region indicating that there is little autocorrelation between present and past values. Overall, ARMA(12,0) does a very good job at reflecting the true data.

6. Model Selection

In this section, all of the statistical components of each model will be compared with one another to determine which model has the highest accuracy and is the most statistically significant. Once, the selection of the best model is made, the h-step forecast will be generated and observed.

6.1 Model Comparison

Figure 6.1.1

	Forecast Error Mean	.Forecast Error Variance	Forecast Error MSE	Forecast Error Q
Average	498.982424	10008.7	261616	10008.7
Naive	163.215149	10039.3	39334.9	10039.3
Drift	552.336508	9495.21	312048	9495.21
Simple Exponential Smoothing	160.976080	10039.3	38609	10039.3
Holt-Linear	107.255725	196378714125]	[18476.234350928127]	[9495.196378714125]
Holt-Winters	107.435975	724491994602]	[18506.871457321355]	[9471.724491994602]
Multiple Linear Regression	0.188576	172.995	13.3775	172.995
ARMA(8,0)	0.658937	1.92675	96.1645	1.92675
ARMA(12,0)	0.687060	1.88148	95.7308	1.88148

The forecast error mean, variance, MSE and Q value are shown in figure 6.1.1. In order to choose the best model, it is necessary to compare the statistical components of each model to one another. All of the base models, Holt-Linear and Holt-Winters have incredibly high forecast error mean, variance, MSE and Q values thus the predicted values are very far away from the actual values indicating the model is very poor at reflecting the raw data. Multiple Linear Regression, ARMA(8,0) and ARMA(12,0) are the best models used in this project; however, due to the biased nature of the confidence intervals and the failed t-test in both ARMA models, they are not statistically significant. Thus, the Multiple Linear Regression model is the best model that reflects the data.

6.2 Forecast Function

The forecast function was developed using OLS Regression model.predict(). This function uses the patterns from the training set to produce the best model with regards to the test set. The forecast function uses the coefficients of feature in the model to make calculations about future values.

Figure 6.2.1

	coef
Open	-0.5379
High	0.7675
Low	0.7708

6.3 H-step Ahead Predictions

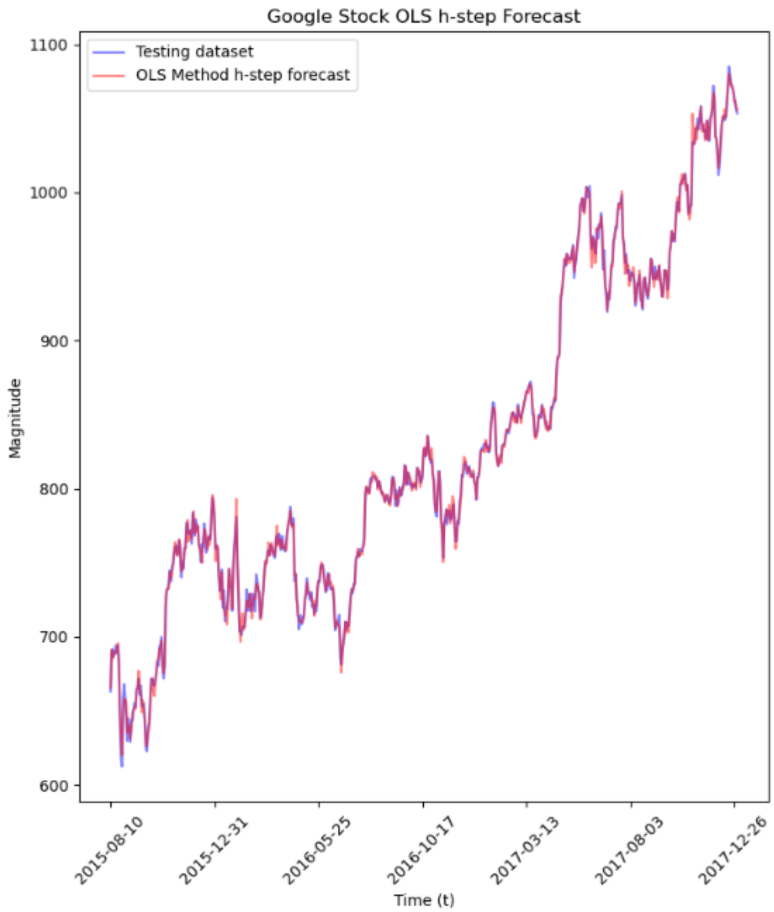


Figure 6.3.1

The h-step ahead predictions using Multiple Linear Regression is shown above compared to the raw Google stock close price data. The h-step ahead prediction accurately predicts the test size with very little error. The MSE was around 14 which is why there are some areas of the curve in which the original data can be seen. Overall, this was the best model at predicting the data because the ACF of the residuals and forecast error reflected behavior of white noise.

7. Summary and Conclusion

Throughout this project, 9 models were tested with different parameters tested for several. The best model out of each one that was tested that was both statistically significant in its residuals and made the most accurate predictions was the multiple linear regression model. Multiple Linear Regression uses other features in the dataset beside the target variable to make predictions of the target variable. Once the model is fit with the training data, the coefficients that were calculated are used to predict the values in the test set. Although stock data is very difficult to accurately predict, this project focused on making predictions using the patterns and behavior from historical data. In that aspect, multiple linear regression was the most successful of any model.

To continue this study, it would be important to test more techniques and models that would decrease the forecast error variance as this was the highest value out of all the multiple linear regression statistical components. It would also be interesting to determine if the other variables, Open, High and Low had a similar trend or similar accuracy rate using multiple linear regression. Furthermore, I would like to use Google sales and profit data to observe any patterns between that and stock prices.

8. References:

Chen, J. (2020, September 16). *How the Valuation Process Works*. Investopedia. https://www.investopedia.com/terms/v/valuation.asp.

9. Appendix

```
import pandas as pd
import matplotlib.pyplot as plt
import statistics
import numpy as np
from statsmodels.tsa.stattools import adfuller
import statsmodels.api as sm
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
import math
from tabulate import tabulate
import seaborn as sns
from statsmodels.tsa.seasonal import STL
from sklearn.model_selection import train_test_split
import statsmodels.tsa.holtwinters as ets
from numpy import linalg as LA
from scipy.stats import chi2


#Loading in the data
df = pd.read_csv('../Final_Project/google_stock.csv')

#Defining the dependent variable
date = df['Date']
stock = df['Close']

#Checking for missing data or NaN
print(df.isnull().values.any())
print(stock.isnull().values.any())

#Correlation Matrix
corr = df.corr()
ax = sns.heatmap(corr, vmin=-1, vmax=1, center=0,
cmap=sns.diverging_palette(20, 220, n=200),square=True)
bottom,top = ax.get_ylim()
ax.set_ylim(bottom+0.5, top-0.5)
ax.set_xticklabels(ax.get_xticklabels(), rotation=45,
horizontalalignment='right')
plt.show()

#Preliminary visualization of raw data
fig, ax = plt.subplots(figsize = (10,9))
plt.plot(date,stock)
plt.xticks(rotation=45)
```

```python
ax.set_xticks(ax.get_xticks()[::200])
plt.xlabel("Time")
plt.ylabel("Dollars ($USD)")
plt.title("Google Stock Close Price")
plt.show()

#ADF-test
def ADF_Cal(x):
    result = adfuller(x)
    print("ADF Statistic: %f" %result[0])
    print('p-value: %f' % result[1])
    print('Critical Values:')
    for key, value in result[4].items():
       print('\t%s: %.3f' % (key, value))


stock_acf = ADF_Cal(stock)
print(stock_acf)

#Calculating ACF
def calc_autocorrelation_coef(y,t_lag,title):
    mean = sum(y)/len(y)
    denominator = 0

    for value in y:
        denominator = denominator + (value - mean)**2

    numerator = 0
    numerator_list = []
    for x in range(0, t_lag+1):
        for i in range(x,len(y)):
            numerator = numerator + (y[i] - mean)*(y[i-x]-mean)
        numerator_list.append(numerator)
        numerator = 0

    r_list = [x/denominator for x in numerator_list]

    x = np.linspace(-t_lag, t_lag, t_lag*2+1)
    # Plotting ACF of Residuals for AR(1)
    fig = plt.figure(figsize=(16, 8))
    r_list_r_yy = r_list[::-1]
    r_list_Ry = r_list_r_yy[:-1] + r_list
    r_list_Ry_np = np.array(r_list_Ry)
    plt.stem(x, r_list_Ry_np, use_line_collection=True)
    plt.title(title)
    plt.ylabel('ACF')
    plt.xlabel('Lag')
    m_pred = 1.96 / math.sqrt(len(y))
    plt.axhspan(-m_pred, m_pred, alpha=.1, color='black')
    plt.show()

    return r_list

#Plotting ACF and PACF using statsmodel
acf = sm.tsa.stattools.acf(stock, nlags=20)
pacf = sm.tsa.stattools.pacf(stock, nlags=20)
```

```
fig = plt.figure(figsize=(6, 10))
fig.tight_layout(pad=4)
plt.subplot(2, 1, 1)
plot_acf(stock, ax=plt.gca(), lags=20, title="ACF of Raw Data")
plt.xlabel("Lags")
plt.ylabel("ACF")
plt.subplot(2, 1, 2)
plot_pacf(stock, ax=plt.gca(), lags=20, title="PACF of Raw Data")
plt.xlabel("Lags")
plt.ylabel("PACF")
plt.show()

#Plotting ACF
raw_data_acf = calc_autocorrelation_coef(stock,20,'ACF of Raw Data')

#Calculating Rolling Mean and Variance of Raw Data
def cal_rolling_var_mean(df,dependent_variable):
    sales_mean = 0
    sales_variance = 0

    mean_sales_time = []
    var_sales_time = []

    for i in range(1, len(df)+1):
        rows = df.head(i)

        seq_sales = rows[dependent_variable].values[-1]
        seq_sales_var = rows[dependent_variable].values

        sales_mean = sales_mean + seq_sales
        mean_sales_time.append(sales_mean / i)

        if i == 1:
            var_sales_time.append(seq_sales_var[-1])
        elif i > 1:
            var_sales_time.append(statistics.variance(seq_sales_var))

    return (mean_sales_time,var_sales_time)

var_sales_time = cal_rolling_var_mean(df,'Close')[1]
mean_sales_time = cal_rolling_var_mean(df,'Close')[0]

#Close Price Rolling Variance
fig, ax = plt.subplots(figsize = (10,10))
ax.plot(date,var_sales_time, label = "Rolling Variance")
plt.xticks(rotation=45)
ax.set_xticks(ax.get_xticks()[::200])
plt.xlabel("Year")
plt.ylabel("Dollars ($USD)")
plt.legend()
plt.title("Google Stock Close Price Rolling Variance")
plt.show()

#Close Price Rolling Mean
fig, ax = plt.subplots(figsize = (10,10))
ax.plot(date,mean_sales_time, label = "Rolling Mean")
plt.xticks(rotation=45)
```

```
ax.set_xticks(ax.get_xticks()[::200])
plt.xlabel("Year")
plt.ylabel("Dollars ($USD)")
plt.legend()
plt.title("Google Stock Close Price Rolling Mean")
plt.show()

#Logarithmic Transformation and Differencing
stock_log1 = np.log(stock)
stock_log1 = stock_log1[1:]

print(ADF_Cal(stock_log1))

stock_log2 = np.log(stock_log1)
stock_log2 = stock_log2[1:]

print(ADF_Cal(stock_log2))

stock_log3 = np.log(stock_log2)
stock_log3 = stock_log3[1:]

print(ADF_Cal(stock_log3))

stock_diff1_after_log3 = stock_log3.diff()
stock_diff1_after_log3 = stock_diff1_after_log3[1:]
print(ADF_Cal(stock_diff1_after_log3))


#Calculating Rolling Mean and Variance AFter 3rd log + 1st diff
diff_air_pass_mean = 0
diff_air_pass_variance = 0

diff_mean_air_pass_time = []
diff_var_air_pass_time = []

for i in range(1, len(stock_diff1_after_log3)+1):
    rows = stock_diff1_after_log3.head(i)
    seq_air_mean = rows.values[-1]
    seq_air_var = rows.values

    diff_air_pass_mean = diff_air_pass_mean + seq_air_mean
    diff_mean_air_pass_time.append(diff_air_pass_mean / i)

    if i == 1:
        diff_var_air_pass_time.append(seq_air_var[-1])
    elif i > 1:
        diff_var_air_pass_time.append(statistics.variance(seq_air_var))


#Close Price Rolling Variance (3rd log + 1st diff)
fig, ax = plt.subplots(figsize = (10,10))
ax.plot(date[4:],diff_var_air_pass_time, label = "Rolling Variance")
plt.xticks(rotation=45)
ax.set_xticks(ax.get_xticks()[::200])
plt.xlabel("Year")
plt.ylabel("Dollars ($USD)")
plt.legend()
```

```
plt.title("Google Stock Close Price Rolling Variance After 3rd log and 1st
diff")
plt.show()

#Close Price Rolling Mean (3rd log + 1st diff)
fig, ax = plt.subplots(figsize = (10,10))
ax.plot(date[4:],diff_mean_air_pass_time, label = "Rolling Mean")
plt.xticks(rotation=45)
ax.set_xticks(ax.get_xticks()[::200])
plt.xlabel("Year")
plt.ylabel("Dollars ($USD)")
plt.legend()
plt.title("Google Stock Close Price Rolling Mean after 3rd log and 1st diff")
plt.show()

#Defining New Data
stationary_data = stock_diff1_after_log3
new_data = stationary_data.tolist()

#Plotting Stationary Data
fig, ax = plt.subplots(figsize = (10,9))
#ax.plot(date,stock, label = "Original")
plt.plot(date[4:],stationary_data, label = " 3rd log + 1st diff")
plt.xticks(rotation=45)
ax.set_xticks(ax.get_xticks()[::200])
plt.xlabel("Year")
plt.ylabel("Dollars ($USD)")
plt.legend()
plt.title("Google Stock Close Price")
plt.show()

#Close Price Differencing
fig, ax = plt.subplots(figsize = (10,9))
ax.plot(date,stock, label = "Original")
plt.plot(date[1:], stock_log1, label="1st log", alpha = 0.5)
plt.plot(date[2:], stock_log2, label="2nd log", alpha = 0.5)
plt.plot(date[3:], stock_log3, label="3rd log", alpha = 0.5)
plt.plot(date[4:], stock_diff1_after_log3, label="3rd log + 1st diff", alpha
= 0.5)
plt.xticks(rotation=45)
ax.set_xticks(ax.get_xticks()[::200])
plt.xlabel("Year")
plt.ylabel("Dollars ($USD)")
plt.legend()
plt.title("Google Stock Original vs. Differencing")
plt.show()

#Plotting ACF and PACF using statsmodel
acf = sm.tsa.stattools.acf(stock, nlags=20)
pacf = sm.tsa.stattools.pacf(stock, nlags=20)

fig = plt.figure(figsize=(6, 10))
fig.tight_layout(pad=4)
plt.subplot(2, 1, 1)
plot_acf(new_data, ax=plt.gca(), lags=20, title="ACF of Stationary Data")
plt.xlabel("Lags")
plt.ylabel("ACF")
```

```python
plt.subplot(2, 1, 2)
plot_pacf(new_data, ax=plt.gca(), lags=20, title="PACF of Stationary Data")
plt.xlabel("Lags")
plt.ylabel("PACF")
plt.show()


df_stationary = pd.DataFrame(new_data,columns={'stationary_stock'})

#Time Series Decomposition======================
"""decomp_data = pd.Series(np.array(df_stationary['stationary_stock']),
                 index = pd.date_range('2006-01-09',
                 periods=len(df_stationary['stationary_stock']),
                 freq = 'm',
                 name='google_stock'))"""
decomp_data = pd.Series(np.array(df['Close']),
                 index = pd.date_range('2006-01-09',
                 periods=len(df['Close']),
                 freq = 'd',
                 name='google_stock'))
STL = STL(decomp_data)
res = STL.fit() #optimization

T = res.trend
S = res.seasonal
R = res.resid

#Plotting decomposition
fig = res.plot()
plt.show()


adjusted_seasonal = decomp_data.values - S
adjusted_trend = decomp_data.values -T


fig, ax  = plt.subplots(figsize=(10,10))
plt.plot(date,decomp_data.values, label = 'Original Data')
plt.plot(date,adjusted_seasonal, label = 'Adjusted Seasonal')
plt.plot(date,adjusted_trend, label = 'Adjusted Trend')
ax.set_xticks(ax.get_xticks()[::350])
plt.xticks(rotation=45)
plt.title('Google Stock Time Series Decomposition')
plt.ylabel("Magnitude")
plt.xlabel("Time")
plt.legend()
plt.show()


#Strength and Seasonality
#Calculate Strength of Trend
R = np.array(R)
T = np.array(T)
S = np.array(S)

F_t = np.max([0, (1 - np.var(R)/np.var(T+R))])
print("The strength of the trend for this data set is: ",F_t)
```

```
#Calculate Strength of /Seasonality
F_s = max([0, (1 - np.var(R)/np.var(S+R))])
print("The strength of the seasonality for this data set is: ",F_s)



#===============================================

#Holt-Winters Method:==========================
#Splitting the Data
#Calculate Q Function
def calc_q(autocorrelation_list, num_samples):
    acf_sq =[x**2 for x in autocorrelation_list[1:]]
    q = sum(acf_sq)*num_samples
    return q

X_train, X_test, y_train, y_test = train_test_split(date,stock, shuffle =
False,test_size=0.20)

#Calculate Error
def calc_error(y,predicted_value):
    err = []
    for i,j in zip(y,predicted_value):
        err.append(i-j)
    return err[1:]

#Error Squared
def calc_error_squared(error):
    error_squared = [x**2 for x in error]
    return error_squared

#MSE
def calc_mse(error_squared):
    mse = sum(error_squared)/len(error_squared)
    return mse

#Chi-square test
def chi_square(Q,y_train_length,row_length,columns_length):
    DOF = y_train_length - row_length - columns_length
    alfa = 0.01
    chi_critical = chi2.ppf(1-alfa,DOF)

    if Q < chi_critical:
        print("Residual is white")
    else:
        print("The Residual is NOT white")


def holt_winter(y_train,y_test):
    aapl_holttw = ets.ExponentialSmoothing(y_train.values,
trend='additive',damped=False,seasonal=None,
                                          seasonal_periods=4).fit()
    aapl_holtfw = aapl_holttw.forecast(steps=len(y_test))
    aapl_holtfw = pd.DataFrame(aapl_holtfw).set_index(y_test.index)
```

```
    aapl_forecast_error_holtw = calc_error(y_test,aapl_holtfw.values)
    forecast_error_squared = calc_error_squared(aapl_forecast_error_holtw)
    forecast_error_mse = calc_mse(forecast_error_squared)
    #aapl_holtw_forecast_error_mse = np.square(np.subtract(y_test,
aapl_forecast_error_holtw)).mean()

    return aapl_holtfw,forecast_error_mse, aapl_forecast_error_holtw

stock_hw_forecast,stock_hw_forecast_error_mse, stock_hw_forecast_error =
holt_winter(y_train,y_test)
print('Holt-Winters Forecast Error MSE:',stock_hw_forecast_error_mse)
hw_forecast_error_acf =
calc_autocorrelation_coef(stock_hw_forecast_error,20,'Holt-Winter Forecast
Error ACF')
hw_forecast_error_Q =
calc_q(hw_forecast_error_acf,len(stock_hw_forecast_error))
print('Holt-Winters Forecast Error Q:',hw_forecast_error_Q)
hw_forecast_error_variance = np.var(stock_hw_forecast_error)
print('Holt-Winters Forecast Error Variance:',hw_forecast_error_variance)
hw_forecast_error_mean = np.mean(stock_hw_forecast_error)
print('Holt-Winters Forecast Error Mean:',hw_forecast_error_mean)


#Plotting Holt-Winter Forecast
fig, ax  = plt.subplots(figsize = (10,10))
plt.plot(X_train,y_train, label = 'Training dataset')
plt.plot(X_test,y_test, label = 'Testing dataset')
plt.plot(X_test,stock_hw_forecast, label = 'Holt-Winter Method h-step
forecast')
ax.set_xticks(ax.get_xticks()[::450])
plt.xticks(rotation=45)
plt.xlabel("Time (t)")
plt.ylabel("Magnitude")
plt.legend(loc = 'upper left')
plt.title("Google Stock Holt-Winter Forecasting")
plt.show()



#================================================


#Feature Selection==============

X = df[['Open','High','Low','Volume']]
Y = df['Close']

X_train1, X_test1, Y_train1, Y_test1  = train_test_split(X,Y,shuffle =
False,test_size=0.20)

#Calculating Singular Values and Condition Number
X = X_train1.values
H = np.matmul(X.T,X)
_,d,_ = np.linalg.svd(H)
print('Singular Values of Original',d)
print('The condition number of original = ',LA.cond(X))
```

```python
#Addding constant to matrix
X_train1['one'] = 1
X_train1 = X_train1[ ['one'] + [ column for column in X_train1.columns if
column!= 'one']]
#print(X_train)

#Calculating Coefficients using LSE
H = np.matmul(X_train1.values.T,X_train1.values)
inv = np.linalg.inv(H)
beta = np.matmul(np.matmul(inv,X_train1.values.T),Y_train1.values)
print("Calculated Coefficients using LSE:",beta)

#Calculating Coefficients using OLS
model = sm.OLS(Y_train1,X_train1).fit()
print(model.summary())#Coefficients are identical

#Removing Predictors - one constant
X_train1.drop(['one'],axis=1,inplace=True)

model = sm.OLS(Y_train1,X_train1).fit()
X = X_train1.values
H = np.matmul(X.T,X)
_,d,_ = np.linalg.svd(H)
print('Singular Values of First Iteration',d)
print('The Condition Number of First Iteration = ',LA.cond(X))
print(model.summary())

#Removing Volume
X_train1.drop(['Volume'],axis=1,inplace=True)#pvalue > 0.05
X_test1.drop(['Volume'],axis=1,inplace=True)

model = sm.OLS(Y_train1,X_train1).fit()
X = X_train1.values
H = np.matmul(X.T,X)
_,d,_ = np.linalg.svd(H)
print('Singular Values of Final Iteration',d)
print('The Condition Number of Final Iteration = ',LA.cond(X))
print(model.summary())
#============================


#Multiple Linear Regression==================

forecast = model.predict(X_test1)
prediction = model.predict(X_train1)

#Plotting Multiple Linear Regression Forecast vs. Test Set
fig, ax = plt.subplots(figsize = (10,10))
ax.plot(X_train,Y_train1, label = 'Training dataset')
ax.plot(X_train,prediction, label='Prediction')
ax.plot(X_test,Y_test1,alpha=0.5, label = 'Testing dataset')
ax.plot(X_test,forecast, alpha=0.5,label = 'OLS Method h-step forecast')
plt.xlabel("Time (t)")
plt.ylabel("Magnitude")
ax.set_xticks(ax.get_xticks()[::350])
plt.legend(loc = 'upper left')
plt.xticks(rotation=45)
```

```
plt.title("Google Stock OLS 1-step Prediction and h-step Forecast")
plt.show()

#Plotting Multiple Linear Regression Forecast vs. Test Set
fig, ax = plt.subplots(figsize = (8,9))
ax.plot(X_test,Y_test1,'b',alpha=0.5, label = 'Testing dataset')
ax.plot(X_test,forecast,'r', alpha=0.5,label = 'OLS Method h-step forecast')
plt.xlabel("Time (t)")
plt.ylabel("Magnitude")
ax.set_xticks(ax.get_xticks()[::100])
plt.legend(loc = 'upper left')
plt.xticks(rotation=45)
plt.title("Google Stock OLS h-step Forecast")
plt.show()

#Comparing Performance

#Calculating Prediction Error
def calc_error(y,predicted_value):
    err = []

    for i,j in zip(y,predicted_value):
        err.append(i-j)
    return err
mlr_prediction_error = calc_error(Y_train1,prediction)
mlr_forecast_error = calc_error(Y_test1,forecast)

#Calculating MSE of Forecast/Prediction Error Multiple Linear Regression
mlr_forecast_error_squared = calc_error_squared(mlr_forecast_error)
mlr_forecast_error_mse = calc_mse(mlr_forecast_error_squared)
print("MLR Forecast Error MSE:",mlr_forecast_error_mse)

#Plotting ACF of Forecast Error
forecast_error_acf_r_y = calc_autocorrelation_coef(mlr_forecast_error,20,"ACF
of MLR Forecast Error")

#Calculating MLR Forecast Error Q
stock_mlr_q = calc_q(forecast_error_acf_r_y,len(Y))
print("MLR Forecast Error Q", stock_mlr_q)

#Calculating MLR Mean and Variance of Forecast Error
mlr_forecast_error_mean = np.mean(mlr_forecast_error)
mlr_forecast_error_variance = np.var(mlr_forecast_error)
print("MLR Forecast Error Variance:",mlr_forecast_error_variance)
print("MLR Forecast Error Mean:",mlr_forecast_error_mean)

#Forecast Estimated Variance
forecast_error_squared_sum = sum([x**2 for x in mlr_forecast_error])
fore_variance = math.sqrt(forecast_error_squared_sum *
(1/(len(mlr_forecast_error) - 7 - 1)))
print('Multiple Linear Regression Forecast Error Estimated
Variance:',fore_variance)

#Prediction Error Analysis
mlr_prediction_error_squared = calc_error_squared(mlr_prediction_error)
mlr_prediction_error_mse = calc_mse(mlr_prediction_error_squared)
print("MLR Prediction Error MSE:",mlr_prediction_error_mse)
```

```python
#Plotting ACF of Prediction Error
prediction_error_acf_r_y =
calc_autocorrelation_coef(mlr_prediction_error,20,"ACF of Prediction Error")

#Calculating MLR Prediction Error Q
mlr_prediction_error_Q = calc_q(prediction_error_acf_r_y,len(Y))
print("MLR Prediction Error Q", stock_mlr_q)

mlr_prediction_error_mean = np.mean(mlr_prediction_error)
mlr_prediction_error_variance = np.var(mlr_prediction_error)
print("MLR Prediction Error Variance:",mlr_forecast_error_variance)
print("MLR Prediction Error Mean:",mlr_forecast_error_mean)

#F-test and T-test analysis
print(model.summary())

#ARMA, ARIMA and SARIMA==================================

#GPAC FUNCTION
def cal_gpac(j1,k1,data):
    y = data
    ry = calc_autocorrelation_coef(y,50,"ACF")
    print(len(ry))
    numerator = []
    denominator = []

    phi_k1 = []
    phi_k2 = []
    phi_k3 = []
    phi_k4 = []
    phi_k5 = []
    phi_k6 = []
    phi_k7 = []
    phi_k8 = []
    phi_k9 = []
    phi_k10 = []
    phi_k11 = []
    phi_k12 = []
    phi_k13 = []
    phi_k14 = []
    phi_k15 = []


    for k in range(1,k1+1):
        for j in range(0,j1):
            if k==1:
                numerator.append(ry[j+k])
                denominator.append(ry[j])
                phi_k1.append(numerator[j]/denominator[j])

            elif k==2:
                numerator_array2 = np.zeros((k, k))
                denominator_array2 = np.zeros((k, k))
                numerator_array2[0] = (ry[j], ry[j + 1])
                numerator_array2[1] = (ry[j + 1], ry[j + 2])
                if j == 0:
```

```
                    denominator_array2[0] = (ry[j],ry[j+1])
                    denominator_array2[1] = (ry[j+1],ry[j])
                else:
                    denominator_array2[0] = (ry[j],ry[j-1])
                    denominator_array2[1] = (ry[j+1],ry[j])

phi_k2.append(np.linalg.det(numerator_array2)/np.linalg.det(denominator_array
2))

            elif k==3:
                numerator_array3 = np.zeros((k, k))
                denominator_array3 = np.zeros((k, k))

                if j == 0:
                    numerator_array3[0] = (ry[j], ry[j+1], ry[j + 1])
                    numerator_array3[1] = (ry[j + 1], ry[j], ry[j + 2])
                    numerator_array3[2] = (ry[j + 2], ry[j + 1], ry[j + 3])
                    denominator_array3[0] = (ry[j], ry[j + 1], ry[j+2])
                    denominator_array3[1] = (ry[j + 1], ry[j], ry[j+1])
                    denominator_array3[2] = (ry[j+2], ry[j+1], ry[j])
                    phi_k3.append(np.linalg.det(numerator_array3) /
np.linalg.det(denominator_array3))

                elif j==1:
                    numerator_array3[0] = (ry[j], ry[j - 1], ry[j + 1])
                    numerator_array3[1] = (ry[j + 1], ry[j], ry[j + 2])
                    numerator_array3[2] = (ry[j + 2], ry[j + 1], ry[j + 3])

                    denominator_array3[0] = (ry[j], ry[j - 1], ry[j])
                    denominator_array3[1] = (ry[j + 1], ry[j], ry[j - 1])
                    denominator_array3[2] = (ry[j + 2], ry[j + 1], ry[j])
                    phi_k3.append(np.linalg.det(numerator_array3) /
np.linalg.det(denominator_array3))

                elif j > 1:
                    numerator_array3[0] = (ry[j], ry[j - 1], ry[j + 1])
                    numerator_array3[1] = (ry[j + 1], ry[j], ry[j + 2])
                    numerator_array3[2] = (ry[j + 2], ry[j + 1], ry[j + 3])

                    denominator_array3[0] = (ry[j], ry[j - 1], ry[j-2])
                    denominator_array3[1] = (ry[j + 1], ry[j], ry[j-1])
                    denominator_array3[2] = (ry[j+2], ry[j+1], ry[j])

phi_k3.append(np.linalg.det(numerator_array3)/np.linalg.det(denominator_array
3))

            elif k == 4:
                numerator_array4 = np.zeros((k, k))
                denominator_array4 = np.zeros((k, k))


                if j == 0:
                    numerator_array4[0] = (ry[j],     ry[j - 1 + 2], ry[j - 2
+ 4], ry[j + 1])
                    numerator_array4[1] = (ry[j + 1], ry[j],     ry[j - 1 +
2], ry[j + 2])
                    numerator_array4[2] = (ry[j + 2], ry[j + 1], ry[j],
```

```
ry[j + 3])
                    numerator_array4[3] = (ry[j + 3], ry[j + 2], ry[j + 1],
ry[j + 4])

                    denominator_array4[0] = (ry[j],     ry[j - 1 + 2], ry[j -
2 + 4], ry[j-k+1+6])
                    denominator_array4[1] = (ry[j + 1], ry[j],     ry[j - 1 +
2], ry[j-k+2+4])
                    denominator_array4[2] = (ry[j + 2], ry[j + 1], ry[j],
ry[j-k+3+2])
                    denominator_array4[3] = (ry[j + 3], ry[j + 2], ry[j + 1],
ry[j-k+4])

                    phi_k4.append(np.linalg.det(numerator_array4) /
np.linalg.det(denominator_array4))

                elif j == 1:
                    numerator_array4[0] = (ry[j], ry[j - 1], ry[j - 2 + 2],
ry[j + 1])
                    numerator_array4[1] = (ry[j + 1], ry[j], ry[j - 1], ry[j
+ 2])
                    numerator_array4[2] = (ry[j + 2], ry[j + 1], ry[j], ry[j
+ 3])
                    numerator_array4[3] = (ry[j + 3], ry[j + 2], ry[j + 1],
ry[j + 4])

                    denominator_array4[0] = (ry[j], ry[j - 1], ry[j - 2 + 2],
ry[j - k + 1 + 4])
                    denominator_array4[1] = (ry[j + 1], ry[j], ry[j - 1],
ry[j - k + 2 + 2])
                    denominator_array4[2] = (ry[j + 2], ry[j + 1], ry[j],
ry[j - k + 3])
                    denominator_array4[3] = (ry[j + 3], ry[j + 2], ry[j + 1],
ry[j - k + 4])
                    phi_k4.append(np.linalg.det(numerator_array4) /
np.linalg.det(denominator_array4))
                elif j == 2:
                    numerator_array4[0] = (ry[j], ry[j - 1], ry[j - 2], ry[j
+ 1])
                    numerator_array4[1] = (ry[j + 1], ry[j], ry[j - 1], ry[j
+ 2])
                    numerator_array4[2] = (ry[j + 2], ry[j + 1], ry[j], ry[j
+ 3])
                    numerator_array4[3] = (ry[j + 3], ry[j + 2], ry[j + 1],
ry[j + 4])

                    denominator_array4[0] = (ry[j], ry[j - 1], ry[j - 2],
ry[j - k + 1 + 2])
                    denominator_array4[1] = (ry[j + 1], ry[j], ry[j - 1],
ry[j - k + 2])
                    denominator_array4[2] = (ry[j + 2], ry[j + 1], ry[j],
ry[j - k + 3])
                    denominator_array4[3] = (ry[j + 3], ry[j + 2], ry[j + 1],
ry[j - k + 4])
                    phi_k4.append(np.linalg.det(numerator_array4) /
np.linalg.det(denominator_array4))
                elif j > 1:
```

```
                    numerator_array4[0] = (ry[j],   ry[j-1], ry[j-2], ry[j+1])
                    numerator_array4[1] = (ry[j+1], ry[j],   ry[j-1], ry[j+2])
                    numerator_array4[2] = (ry[j+2], ry[j+1], ry[j], ry[j+3])
                    numerator_array4[3] = (ry[j+3], ry[j+2], ry[j+1], ry[j+4])

                    denominator_array4[0] = (ry[j],   ry[j-1], ry[j-2], ry[j-k+1])
                    denominator_array4[1] = (ry[j+1], ry[j],   ry[j-1], ry[j-k+2])
                    denominator_array4[2] = (ry[j+2], ry[j+1], ry[j],   ry[j-k+3])
                    denominator_array4[3] = (ry[j+3], ry[j+2], ry[j+1], ry[j-k+4])
                    phi_k4.append(np.linalg.det(numerator_array4) / np.linalg.det(denominator_array4))

            elif k == 5:
                numerator_array5 = np.zeros((k, k))
                denominator_array5 = np.zeros((k, k))

                if j == 0:
                    numerator_array5[0] = (ry[j], ry[j - 1 + 2], ry[j - 2 + 4], ry[j - 3 + 6], ry[j + 1])
                    numerator_array5[1] = (ry[j + 1], ry[j], ry[j - 1 + 2], ry[j - 2 + 4], ry[j + 2])
                    numerator_array5[2] = (ry[j + 2], ry[j + 1], ry[j], ry[j - 1 + 2], ry[j + 3])
                    numerator_array5[3] = (ry[j + 3], ry[j + 2], ry[j + 1], ry[j], ry[j + 4])
                    numerator_array5[4] = (ry[j + 4], ry[j + 3], ry[j + 2], ry[j + 1], ry[j + 5])

                    denominator_array5[0] = (ry[j], ry[j - 1 + 2], ry[j - 2 + 4], ry[j - 3 + 6], ry[j - k + 1 + 8])
                    denominator_array5[1] = (ry[j + 1], ry[j], ry[j - 1 + 2], ry[j - 2 + 4], ry[j - k + 2 + 6])
                    denominator_array5[2] = (ry[j + 2], ry[j + 1], ry[j], ry[j - 1 + 2], ry[j - k + 3 + 4])
                    denominator_array5[3] = (ry[j + 3], ry[j + 2], ry[j + 1], ry[j], ry[j - k + 4 + 2])
                    denominator_array5[4] = (ry[j + 4], ry[j + 3], ry[j + 2], ry[j + 1], ry[j - k + 5])

                    phi_k5.append(np.linalg.det(numerator_array5) / np.linalg.det(denominator_array5))

                elif j == 1:
                    numerator_array5[0] = (ry[j], ry[j - 1], ry[j - 2 + 2], ry[j - 3 + 4], ry[j + 1])
                    numerator_array5[1] = (ry[j + 1], ry[j], ry[j - 1], ry[j - 2 +2], ry[j + 2])
                    numerator_array5[2] = (ry[j + 2], ry[j + 1], ry[j], ry[j - 1], ry[j + 3])
```

```
                    numerator_array5[3] = (ry[j + 3], ry[j + 2], ry[j + 1],
ry[j], ry[j + 4])
                    numerator_array5[4] = (ry[j + 4], ry[j + 3], ry[j + 2],
ry[j + 1], ry[j + 5])

                    denominator_array5[0] = (ry[j], ry[j - 1], ry[j - 2 + 2],
ry[j - 3 + 4], ry[j - k + 1 + 6])
                    denominator_array5[1] = (ry[j + 1], ry[j], ry[j - 1],
ry[j - 2+2], ry[j - k + 2 + 4])
                    denominator_array5[2] = (ry[j + 2], ry[j + 1], ry[j],
ry[j - 1], ry[j - k + 3 + 2])
                    denominator_array5[3] = (ry[j + 3], ry[j + 2], ry[j + 1],
ry[j], ry[j - k + 4])
                    denominator_array5[4] = (ry[j + 4], ry[j + 3], ry[j + 2],
ry[j + 1], ry[j - k + 5])

                    phi_k5.append(np.linalg.det(numerator_array5) /
np.linalg.det(denominator_array5))

                elif j == 2:
                    numerator_array5[0] = (ry[j], ry[j - 1], ry[j - 2], ry[j
- 3 + 2], ry[j + 1])
                    numerator_array5[1] = (ry[j + 1], ry[j], ry[j - 1], ry[j
- 2], ry[j + 2])
                    numerator_array5[2] = (ry[j + 2], ry[j + 1], ry[j], ry[j
- 1], ry[j + 3])
                    numerator_array5[3] = (ry[j + 3], ry[j + 2], ry[j + 1],
ry[j], ry[j + 4])
                    numerator_array5[4] = (ry[j + 4], ry[j + 3], ry[j + 2],
ry[j + 1], ry[j + 5])

                    denominator_array5[0] = (ry[j], ry[j - 1], ry[j - 2],
ry[j - 3 + 2], ry[j - k + 1 + 4])
                    denominator_array5[1] = (ry[j + 1], ry[j], ry[j - 1],
ry[j - 2], ry[j - k + 2 + 2])
                    denominator_array5[2] = (ry[j + 2], ry[j + 1], ry[j],
ry[j - 1], ry[j - k + 3])
                    denominator_array5[3] = (ry[j + 3], ry[j + 2], ry[j + 1],
ry[j], ry[j - k + 4])
                    denominator_array5[4] = (ry[j + 4], ry[j + 3], ry[j + 2],
ry[j + 1], ry[j - k + 5])

                    phi_k5.append(np.linalg.det(numerator_array5) /
np.linalg.det(denominator_array5))

                elif j == 3:
                    numerator_array5[0] = (ry[j], ry[j - 1], ry[j - 2], ry[j
- 3], ry[j + 1])
                    numerator_array5[1] = (ry[j + 1], ry[j], ry[j - 1], ry[j
- 2], ry[j + 2])
                    numerator_array5[2] = (ry[j + 2], ry[j + 1], ry[j], ry[j
- 1], ry[j + 3])
                    numerator_array5[3] = (ry[j + 3], ry[j + 2], ry[j + 1],
ry[j], ry[j + 4])
                    numerator_array5[4] = (ry[j + 4], ry[j + 3], ry[j + 2],
ry[j + 1], ry[j + 5])
```

```
                    denominator_array5[0] = (ry[j], ry[j - 1], ry[j - 2],
ry[j - 3], ry[j - k + 1 + 2])
                    denominator_array5[1] = (ry[j + 1], ry[j], ry[j - 1],
ry[j - 2], ry[j - k + 2])
                    denominator_array5[2] = (ry[j + 2], ry[j + 1], ry[j],
ry[j - 1], ry[j - k + 3])
                    denominator_array5[3] = (ry[j + 3], ry[j + 2], ry[j + 1],
ry[j], ry[j - k + 4])
                    denominator_array5[4] = (ry[j + 4], ry[j + 3], ry[j + 2],
ry[j + 1], ry[j - k + 5])
                    phi_k5.append(np.linalg.det(numerator_array5) /
np.linalg.det(denominator_array5))

                elif j > 3:
                    numerator_array5[0] = (ry[j], ry[j - 1], ry[j - 2], ry[j-
3], ry[j + 1])
                    numerator_array5[1] = (ry[j + 1], ry[j], ry[j - 1], ry[j-
2], ry[j + 2])
                    numerator_array5[2] = (ry[j + 2], ry[j + 1], ry[j], ry[j-
1], ry[j + 3])
                    numerator_array5[3] = (ry[j + 3], ry[j + 2], ry[j + 1],
ry[j], ry[j + 4])
                    numerator_array5[4] = (ry[j + 4], ry[j + 3], ry[j + 2],
ry[j+1], ry[j + 5])

                    denominator_array5[0] = (ry[j], ry[j - 1], ry[j - 2],
ry[j-3], ry[j - k + 1])
                    denominator_array5[1] = (ry[j + 1], ry[j], ry[j - 1],
ry[j-2], ry[j - k + 2])
                    denominator_array5[2] = (ry[j + 2], ry[j + 1], ry[j],
ry[j-1], ry[j - k + 3])
                    denominator_array5[3] = (ry[j + 3], ry[j + 2], ry[j + 1],
ry[j], ry[j - k + 4])
                    denominator_array5[4] = (ry[j + 4], ry[j + 3], ry[j + 2],
ry[j+1], ry[j - k + 5])

                    phi_k5.append(np.linalg.det(numerator_array5) /
np.linalg.det(denominator_array5))

            elif k == 6:
                numerator_array6 = np.zeros((k, k))
                denominator_array6 = np.zeros((k, k))

                if j == 0:
                    numerator_array6[0] = (ry[j],     ry[j - 1 + 2], ry[j - 2
+ 4], ry[j - 3 + 6], ry[j - 4 + 8], ry[j + 1])
                    numerator_array6[1] = (ry[j + 1], ry[j],         ry[j - 1
+ 2], ry[j - 2 + 4], ry[j - 3 + 6], ry[j + 2])
                    numerator_array6[2] = (ry[j + 2], ry[j + 1],     ry[j],
ry[j - 1 + 2], ry[j - 2 + 4], ry[j + 3])
                    numerator_array6[3] = (ry[j + 3], ry[j + 2],     ry[j +
1],     ry[j], ry[j - 1 + 2], ry[j + 4])
                    numerator_array6[4] = (ry[j + 4], ry[j + 3],     ry[j +
2],     ry[j + 1], ry[j], ry[j + 5])
                    numerator_array6[5] = (ry[j + 5], ry[j + 4],     ry[j +
3],     ry[j + 2], ry[j + 1], ry[j + 6])
```

```
                    denominator_array6[0] = (ry[j], ry[j - 1 + 2], ry[j - 2 +
4], ry[j - 3 + 6], ry[j - 4 + 8], ry[j - k + 1 + 10])
                    denominator_array6[1] = (ry[j + 1], ry[j], ry[j - 1 + 2],
ry[j - 2 + 4], ry[j - 3 + 6], ry[j - k + 2 + 8])
                    denominator_array6[2] = (ry[j + 2], ry[j + 1], ry[j],
ry[j - 1 + 2], ry[j - 2 + 4], ry[j - k + 3 + 6])
                    denominator_array6[3] = (ry[j + 3], ry[j + 2], ry[j + 1],
ry[j], ry[j - 1 + 2], ry[j - k + 4 + 4])
                    denominator_array6[4] = (ry[j + 4], ry[j + 3], ry[j + 2],
ry[j + 1], ry[j], ry[j - k + 5 + 2])
                    denominator_array6[5] = (ry[j + 5], ry[j + 4], ry[j + 3],
ry[j + 2], ry[j + 1], ry[j - k + 6])


phi_k6.append(np.linalg.det(numerator_array6)/np.linalg.det(denominator_array
6))

                if j == 1:
                    numerator_array6[0] = (ry[j], ry[j - 1], ry[j - 2 + 2],
ry[j - 3 + 4],ry[j-4 + 6], ry[j + 1])
                    numerator_array6[1] = (ry[j + 1], ry[j], ry[j - 1], ry[j
- 2 + 2],ry[j-3+4], ry[j + 2])
                    numerator_array6[2] = (ry[j + 2], ry[j + 1], ry[j], ry[j
- 1],ry[j-2 +2], ry[j + 3])
                    numerator_array6[3] = (ry[j + 3], ry[j + 2], ry[j + 1],
ry[j],ry[j-1], ry[j + 4])
                    numerator_array6[4] = (ry[j + 4], ry[j + 3], ry[j + 2],
ry[j + 1],ry[j], ry[j + 5])
                    numerator_array6[5] = (ry[j + 5], ry[j + 4], ry[j + 3],
ry[j + 2],ry[j+1], ry[j + 6])

                    denominator_array6[0] = (ry[j], ry[j - 1], ry[j - 2 + 2],
ry[j - 3 + 4],ry[j-4 + 6], ry[j - k + 1 + 8])
                    denominator_array6[1] = (ry[j + 1], ry[j], ry[j - 1],
ry[j - 2+2],ry[j-3+4], ry[j - k + 2 + 6])
                    denominator_array6[2] = (ry[j + 2], ry[j + 1], ry[j],
ry[j - 1],ry[j-2+2], ry[j - k + 3 + 4])
                    denominator_array6[3] = (ry[j + 3], ry[j + 2], ry[j + 1],
ry[j],ry[j-1], ry[j - k + 4 + 2])
                    denominator_array6[4] = (ry[j + 4], ry[j + 3], ry[j + 2],
ry[j + 1],ry[j], ry[j - k + 5])
                    denominator_array6[5] = (ry[j + 5], ry[j + 4], ry[j + 3],
ry[j + 2],ry[j+1], ry[j - k + 6])


phi_k6.append(np.linalg.det(numerator_array6)/np.linalg.det(denominator_array
6))

                if j==2:
                    numerator_array6[0] = (ry[j], ry[j - 1], ry[j - 2], ry[j
- 3 + 2],ry[j-4 +4], ry[j + 1])
                    numerator_array6[1] = (ry[j + 1], ry[j], ry[j - 1], ry[j
- 2],ry[j-3 + 2], ry[j + 2])
                    numerator_array6[2] = (ry[j + 2], ry[j + 1], ry[j], ry[j
- 1],ry[j-2], ry[j + 3])
                    numerator_array6[3] = (ry[j + 3], ry[j + 2], ry[j + 1],
ry[j],ry[j-1], ry[j + 4])
```

```
                    numerator_array6[4] = (ry[j + 4], ry[j + 3], ry[j + 2],
ry[j + 1],ry[j], ry[j + 5])
                    numerator_array6[5] = (ry[j + 5], ry[j + 4], ry[j + 3],
ry[j + 2],ry[j+1], ry[j + 6])

                    denominator_array6[0] = (ry[j], ry[j - 1], ry[j - 2],
ry[j - 3 + 2],ry[j-4 + 4], ry[j - k + 1 + 6])
                    denominator_array6[1] = (ry[j + 1], ry[j], ry[j - 1],
ry[j - 2],ry[j-3+2], ry[j - k + 2 + 4])
                    denominator_array6[2] = (ry[j + 2], ry[j + 1], ry[j],
ry[j - 1],ry[j-2], ry[j - k + 3 + 2])
                    denominator_array6[3] = (ry[j + 3], ry[j + 2], ry[j + 1],
ry[j],ry[j-1], ry[j - k + 4])
                    denominator_array6[4] = (ry[j + 4], ry[j + 3], ry[j + 2],
ry[j + 1],ry[j], ry[j - k + 5])
                    denominator_array6[5] = (ry[j + 5], ry[j + 4], ry[j + 3],
ry[j + 2],ry[j+1], ry[j - k + 6])


phi_k6.append(np.linalg.det(numerator_array6)/np.linalg.det(denominator_array
6))

                if j==3:
                    numerator_array6[0] = (ry[j], ry[j - 1], ry[j - 2], ry[j
- 3],ry[j-4 + 2], ry[j + 1])
                    numerator_array6[1] = (ry[j + 1], ry[j], ry[j - 1], ry[j
- 2],ry[j-3], ry[j + 2])
                    numerator_array6[2] = (ry[j + 2], ry[j + 1], ry[j], ry[j
- 1],ry[j-2], ry[j + 3])
                    numerator_array6[3] = (ry[j + 3], ry[j + 2], ry[j + 1],
ry[j],ry[j-1], ry[j + 4])
                    numerator_array6[4] = (ry[j + 4], ry[j + 3], ry[j + 2],
ry[j + 1],ry[j], ry[j + 5])
                    numerator_array6[5] = (ry[j + 5], ry[j + 4], ry[j + 3],
ry[j + 2],ry[j+1], ry[j + 6])

                    denominator_array6[0] = (ry[j], ry[j - 1], ry[j - 2],
ry[j - 3],ry[j-4 + 2], ry[j - k + 1 + 4])
                    denominator_array6[1] = (ry[j + 1], ry[j], ry[j - 1],
ry[j - 2],ry[j-3], ry[j - k + 2 + 2])
                    denominator_array6[2] = (ry[j + 2], ry[j + 1], ry[j],
ry[j - 1],ry[j-2], ry[j - k + 3])
                    denominator_array6[3] = (ry[j + 3], ry[j + 2], ry[j + 1],
ry[j],ry[j-1], ry[j - k + 4])
                    denominator_array6[4] = (ry[j + 4], ry[j + 3], ry[j + 2],
ry[j + 1],ry[j], ry[j - k + 5])
                    denominator_array6[5] = (ry[j + 5], ry[j + 4], ry[j + 3],
ry[j + 2],ry[j+1], ry[j - k + 6])


phi_k6.append(np.linalg.det(numerator_array6)/np.linalg.det(denominator_array
6))

                if j==4:
                    numerator_array6[0] = (ry[j], ry[j - 1], ry[j - 2], ry[j
- 3],ry[j-4], ry[j + 1])
                    numerator_array6[1] = (ry[j + 1], ry[j], ry[j - 1], ry[j
```

```
- 2],ry[j-3], ry[j + 2])
                    numerator_array6[2] = (ry[j + 2], ry[j + 1], ry[j], ry[j
- 1],ry[j-2], ry[j + 3])
                    numerator_array6[3] = (ry[j + 3], ry[j + 2], ry[j + 1],
ry[j],ry[j-1], ry[j + 4])
                    numerator_array6[4] = (ry[j + 4], ry[j + 3], ry[j + 2],
ry[j + 1],ry[j], ry[j + 5])
                    numerator_array6[5] = (ry[j + 5], ry[j + 4], ry[j + 3],
ry[j + 2],ry[j+1], ry[j + 6])

                    denominator_array6[0] = (ry[j], ry[j - 1], ry[j - 2],
ry[j - 3],ry[j-4], ry[j - k + 1 + 2])
                    denominator_array6[1] = (ry[j + 1], ry[j], ry[j - 1],
ry[j - 2],ry[j-3], ry[j - k + 2])
                    denominator_array6[2] = (ry[j + 2], ry[j + 1], ry[j],
ry[j - 1],ry[j-2], ry[j - k + 3])
                    denominator_array6[3] = (ry[j + 3], ry[j + 2], ry[j + 1],
ry[j],ry[j-1], ry[j - k + 4])
                    denominator_array6[4] = (ry[j + 4], ry[j + 3], ry[j + 2],
ry[j + 1],ry[j], ry[j - k + 5])
                    denominator_array6[5] = (ry[j + 5], ry[j + 4], ry[j + 3],
ry[j + 2],ry[j+1], ry[j - k + 6])


phi_k6.append(np.linalg.det(numerator_array6)/np.linalg.det(denominator_array
6))

                elif j > 4:
                    numerator_array6[0] = (ry[j], ry[j - 1], ry[j - 2], ry[j
- 3],ry[j-4], ry[j + 1])
                    numerator_array6[1] = (ry[j + 1], ry[j], ry[j - 1], ry[j
- 2],ry[j-3], ry[j + 2])
                    numerator_array6[2] = (ry[j + 2], ry[j + 1], ry[j], ry[j
- 1],ry[j-2], ry[j + 3])
                    numerator_array6[3] = (ry[j + 3], ry[j + 2], ry[j + 1],
ry[j],ry[j-1], ry[j + 4])
                    numerator_array6[4] = (ry[j + 4], ry[j + 3], ry[j + 2],
ry[j + 1],ry[j], ry[j + 5])
                    numerator_array6[5] = (ry[j + 5], ry[j + 4], ry[j + 3],
ry[j + 2],ry[j+1], ry[j + 6])

                    denominator_array6[0] = (ry[j], ry[j - 1], ry[j - 2],
ry[j - 3],ry[j-4], ry[j - k + 1])
                    denominator_array6[1] = (ry[j + 1], ry[j], ry[j - 1],
ry[j - 2],ry[j-3], ry[j - k + 2])
                    denominator_array6[2] = (ry[j + 2], ry[j + 1], ry[j],
ry[j - 1],ry[j-2], ry[j - k + 3])
                    denominator_array6[3] = (ry[j + 3], ry[j + 2], ry[j + 1],
ry[j],ry[j-1], ry[j - k + 4])
                    denominator_array6[4] = (ry[j + 4], ry[j + 3], ry[j + 2],
ry[j + 1],ry[j], ry[j - k + 5])
                    denominator_array6[5] = (ry[j + 5], ry[j + 4], ry[j + 3],
ry[j + 2],ry[j+1], ry[j - k + 6])

                    phi_k6.append(np.linalg.det(numerator_array6) /
np.linalg.det(denominator_array6))
```

```
            elif k == 7:
                numerator_array7 = np.zeros((k, k))
                denominator_array7 = np.zeros((k, k))

                if j == 0:
                    numerator_array7[0] = (ry[j], ry[j - 1 + 2], ry[j - 2 +
4], ry[j - 3 + 6], ry[j - 4 + 8], ry[j-5+10], ry[j + 1])
                    numerator_array7[1] = (ry[j + 1], ry[j], ry[j - 1+2],
ry[j - 2+4], ry[j - 3+6],ry[j-4+8], ry[j + 2])
                    numerator_array7[2] = (ry[j + 2], ry[j + 1], ry[j], ry[j
- 1+2], ry[j - 2+4], ry[j-3+6],ry[j + 3])
                    numerator_array7[3] = (ry[j + 3], ry[j + 2], ry[j + 1],
ry[j], ry[j - 1+2],ry[j-2+4], ry[j + 4])
                    numerator_array7[4] = (ry[j + 4], ry[j + 3], ry[j + 2],
ry[j + 1], ry[j],ry[j-1+2], ry[j + 5])
                    numerator_array7[5] = (ry[j + 5], ry[j + 4], ry[j + 3],
ry[j + 2], ry[j + 1],ry[j], ry[j + 6])
                    numerator_array7[6] = (ry[j + 6], ry[j + 5], ry[j + 4],
ry[j + 3], ry[j + 2],ry[j+1], ry[j + 7])


                    denominator_array7[0] = (ry[j], ry[j - 1+2], ry[j - 2+4],
ry[j - 3+6], ry[j - 4+8], ry[j-5+10], ry[j - k + 1+12])
                    denominator_array7[1] = (ry[j + 1], ry[j], ry[j - 1+2],
ry[j - 2+4], ry[j - 3+6], ry[j-4+8],ry[j - k + 2+10])
                    denominator_array7[2] = (ry[j + 2], ry[j + 1], ry[j],
ry[j - 1+2], ry[j - 2+4], ry[j-3+6],ry[j - k + 3+8])
                    denominator_array7[3] = (ry[j + 3], ry[j + 2], ry[j + 1],
ry[j], ry[j - 1+2],ry[j-2+4], ry[j - k + 4+6])
                    denominator_array7[4] = (ry[j + 4], ry[j + 3], ry[j + 2],
ry[j + 1], ry[j],ry[j-1+2], ry[j - k + 5+4])
                    denominator_array7[5] = (ry[j + 5], ry[j + 4], ry[j + 3],
ry[j + 2], ry[j + 1],ry[j], ry[j - k + 6+2])
                    denominator_array7[6] = (ry[j + 6], ry[j + 5], ry[j + 4],
ry[j + 3], ry[j + 2], ry[j+1], ry[j - k + 7])

                    phi_k7.append(np.linalg.det(numerator_array7) /
np.linalg.det(denominator_array7))

                elif j == 1:
                    numerator_array7[0] = (ry[j], ry[j - 1], ry[j - 2+2],
ry[j - 3+4], ry[j - 4+6], ry[j-5+8], ry[j + 1])
                    numerator_array7[1] = (ry[j + 1], ry[j], ry[j - 1], ry[j
- 2+2], ry[j - 3+4],ry[j-4+6], ry[j + 2])
                    numerator_array7[2] = (ry[j + 2], ry[j + 1], ry[j], ry[j
- 1], ry[j - 2+2], ry[j-3+4],ry[j + 3])
                    numerator_array7[3] = (ry[j + 3], ry[j + 2], ry[j + 1],
ry[j], ry[j - 1],ry[j-2+2], ry[j + 4])
                    numerator_array7[4] = (ry[j + 4], ry[j + 3], ry[j + 2],
ry[j + 1], ry[j],ry[j-1], ry[j + 5])
                    numerator_array7[5] = (ry[j + 5], ry[j + 4], ry[j + 3],
ry[j + 2], ry[j + 1],ry[j], ry[j + 6])
                    numerator_array7[6] = (ry[j + 6], ry[j + 5], ry[j + 4],
ry[j + 3], ry[j + 2],ry[j+1], ry[j + 7])


                    denominator_array7[0] = (ry[j], ry[j - 1], ry[j - 2+2],
```

```
ry[j - 3+4], ry[j - 4+6], ry[j-5+8], ry[j - k + 1+10])
                    denominator_array7[1] = (ry[j + 1], ry[j], ry[j - 1],
ry[j - 2+2], ry[j - 3+4], ry[j-4+6],ry[j - k + 2+8])
                    denominator_array7[2] = (ry[j + 2], ry[j + 1], ry[j],
ry[j - 1], ry[j - 2+2], ry[j-3+4],ry[j - k + 3+6])
                    denominator_array7[3] = (ry[j + 3], ry[j + 2], ry[j + 1],
ry[j], ry[j - 1],ry[j-2+2], ry[j - k + 4+4])
                    denominator_array7[4] = (ry[j + 4], ry[j + 3], ry[j + 2],
ry[j + 1], ry[j],ry[j-1], ry[j - k + 5+2])
                    denominator_array7[5] = (ry[j + 5], ry[j + 4], ry[j + 3],
ry[j + 2], ry[j + 1],ry[j], ry[j - k + 6])
                    denominator_array7[6] = (ry[j + 6], ry[j + 5], ry[j + 4],
ry[j + 3], ry[j + 2], ry[j+1], ry[j - k + 7])

                    phi_k7.append(np.linalg.det(numerator_array7) /
np.linalg.det(denominator_array7))

                elif j==2:
                    numerator_array7[0] = (ry[j], ry[j - 1], ry[j - 2], ry[j
- 3+2], ry[j - 4+4], ry[j-5+6], ry[j + 1])
                    numerator_array7[1] = (ry[j + 1], ry[j], ry[j - 1], ry[j
- 2], ry[j - 3+2],ry[j-4+4], ry[j + 2])
                    numerator_array7[2] = (ry[j + 2], ry[j + 1], ry[j], ry[j
- 1], ry[j - 2], ry[j-3+2],ry[j + 3])
                    numerator_array7[3] = (ry[j + 3], ry[j + 2], ry[j + 1],
ry[j], ry[j - 1],ry[j-2], ry[j + 4])
                    numerator_array7[4] = (ry[j + 4], ry[j + 3], ry[j + 2],
ry[j + 1], ry[j],ry[j-1], ry[j + 5])
                    numerator_array7[5] = (ry[j + 5], ry[j + 4], ry[j + 3],
ry[j + 2], ry[j + 1],ry[j], ry[j + 6])
                    numerator_array7[6] = (ry[j + 6], ry[j + 5], ry[j + 4],
ry[j + 3], ry[j + 2],ry[j+1], ry[j + 7])


                    denominator_array7[0] = (ry[j], ry[j - 1], ry[j - 2],
ry[j - 3+2], ry[j - 4+4], ry[j-5+6], ry[j - k + 1 + 8])
                    denominator_array7[1] = (ry[j + 1], ry[j], ry[j - 1],
ry[j - 2], ry[j - 3+2], ry[j-4+4],ry[j - k + 2 + 6])
                    denominator_array7[2] = (ry[j + 2], ry[j + 1], ry[j],
ry[j - 1], ry[j - 2], ry[j-3+2],ry[j - k + 3 + 4])
                    denominator_array7[3] = (ry[j + 3], ry[j + 2], ry[j + 1],
ry[j], ry[j - 1],ry[j-2], ry[j - k + 4 + 2])
                    denominator_array7[4] = (ry[j + 4], ry[j + 3], ry[j + 2],
ry[j + 1], ry[j],ry[j-1], ry[j - k + 5])
                    denominator_array7[5] = (ry[j + 5], ry[j + 4], ry[j + 3],
ry[j + 2], ry[j + 1],ry[j], ry[j - k + 6])
                    denominator_array7[6] = (ry[j + 6], ry[j + 5], ry[j + 4],
ry[j + 3], ry[j + 2], ry[j+1], ry[j - k + 7])

                    phi_k7.append(np.linalg.det(numerator_array7) /
np.linalg.det(denominator_array7))

                elif j==3:
                    numerator_array7[0] = (ry[j], ry[j - 1], ry[j - 2], ry[j
- 3], ry[j - 4+2], ry[j-5+4], ry[j + 1])
                    numerator_array7[1] = (ry[j + 1], ry[j], ry[j - 1], ry[j
- 2], ry[j - 3],ry[j-4+2], ry[j + 2])
```

```
                    numerator_array7[2] = (ry[j + 2], ry[j + 1], ry[j], ry[j
- 1], ry[j - 2], ry[j-3],ry[j + 3])
                    numerator_array7[3] = (ry[j + 3], ry[j + 2], ry[j + 1],
ry[j], ry[j - 1],ry[j-2], ry[j + 4])
                    numerator_array7[4] = (ry[j + 4], ry[j + 3], ry[j + 2],
ry[j + 1], ry[j],ry[j-1], ry[j + 5])
                    numerator_array7[5] = (ry[j + 5], ry[j + 4], ry[j + 3],
ry[j + 2], ry[j + 1],ry[j], ry[j + 6])
                    numerator_array7[6] = (ry[j + 6], ry[j + 5], ry[j + 4],
ry[j + 3], ry[j + 2],ry[j+1], ry[j + 7])


                    denominator_array7[0] = (ry[j], ry[j - 1], ry[j - 2],
ry[j - 3], ry[j - 4+2], ry[j-5+4], ry[j - k + 1 + 6])
                    denominator_array7[1] = (ry[j + 1], ry[j], ry[j - 1],
ry[j - 2], ry[j - 3], ry[j-4+2],ry[j - k + 2 + 4])
                    denominator_array7[2] = (ry[j + 2], ry[j + 1], ry[j],
ry[j - 1], ry[j - 2], ry[j-3],ry[j - k + 3 + 2])
                    denominator_array7[3] = (ry[j + 3], ry[j + 2], ry[j + 1],
ry[j], ry[j - 1],ry[j-2], ry[j - k + 4])
                    denominator_array7[4] = (ry[j + 4], ry[j + 3], ry[j + 2],
ry[j + 1], ry[j],ry[j-1], ry[j - k + 5])
                    denominator_array7[5] = (ry[j + 5], ry[j + 4], ry[j + 3],
ry[j + 2], ry[j + 1],ry[j], ry[j - k + 6])
                    denominator_array7[6] = (ry[j + 6], ry[j + 5], ry[j + 4],
ry[j + 3], ry[j + 2], ry[j+1], ry[j - k + 7])
                    phi_k7.append(np.linalg.det(numerator_array7) /
np.linalg.det(denominator_array7))

                elif j==4:
                    numerator_array7[0] = (ry[j], ry[j - 1], ry[j - 2], ry[j
- 3], ry[j - 4], ry[j-5+2], ry[j + 1])
                    numerator_array7[1] = (ry[j + 1], ry[j], ry[j - 1], ry[j
- 2], ry[j - 3],ry[j-4], ry[j + 2])
                    numerator_array7[2] = (ry[j + 2], ry[j + 1], ry[j], ry[j
- 1], ry[j - 2], ry[j-3],ry[j + 3])
                    numerator_array7[3] = (ry[j + 3], ry[j + 2], ry[j + 1],
ry[j], ry[j - 1],ry[j-2], ry[j + 4])
                    numerator_array7[4] = (ry[j + 4], ry[j + 3], ry[j + 2],
ry[j + 1], ry[j],ry[j-1], ry[j + 5])
                    numerator_array7[5] = (ry[j + 5], ry[j + 4], ry[j + 3],
ry[j + 2], ry[j + 1],ry[j], ry[j + 6])
                    numerator_array7[6] = (ry[j + 6], ry[j + 5], ry[j + 4],
ry[j + 3], ry[j + 2],ry[j+1], ry[j + 7])


                    denominator_array7[0] = (ry[j], ry[j - 1], ry[j - 2],
ry[j - 3], ry[j - 4], ry[j-5+2], ry[j - k + 1 + 4])
                    denominator_array7[1] = (ry[j + 1], ry[j], ry[j - 1],
ry[j - 2], ry[j - 3], ry[j-4],ry[j - k + 2 + 2])
                    denominator_array7[2] = (ry[j + 2], ry[j + 1], ry[j],
ry[j - 1], ry[j - 2], ry[j-3],ry[j - k + 3])
                    denominator_array7[3] = (ry[j + 3], ry[j + 2], ry[j + 1],
ry[j], ry[j - 1],ry[j-2], ry[j - k + 4])
                    denominator_array7[4] = (ry[j + 4], ry[j + 3], ry[j + 2],
ry[j + 1], ry[j],ry[j-1], ry[j - k + 5])
                    denominator_array7[5] = (ry[j + 5], ry[j + 4], ry[j + 3],
```

```
ry[j + 2], ry[j + 1],ry[j], ry[j - k + 6])
                    denominator_array7[6] = (ry[j + 6], ry[j + 5], ry[j + 4],
ry[j + 3], ry[j + 2], ry[j+1], ry[j - k + 7])
                    phi_k7.append(np.linalg.det(numerator_array7) /
np.linalg.det(denominator_array7))
                elif j ==5:
                    numerator_array7[0] = (ry[j], ry[j - 1], ry[j - 2], ry[j
- 3], ry[j - 4], ry[j-5], ry[j + 1])
                    numerator_array7[1] = (ry[j + 1], ry[j], ry[j - 1], ry[j
- 2], ry[j - 3],ry[j-4], ry[j + 2])
                    numerator_array7[2] = (ry[j + 2], ry[j + 1], ry[j], ry[j
- 1], ry[j - 2], ry[j-3],ry[j + 3])
                    numerator_array7[3] = (ry[j + 3], ry[j + 2], ry[j + 1],
ry[j], ry[j - 1],ry[j-2], ry[j + 4])
                    numerator_array7[4] = (ry[j + 4], ry[j + 3], ry[j + 2],
ry[j + 1], ry[j],ry[j-1], ry[j + 5])
                    numerator_array7[5] = (ry[j + 5], ry[j + 4], ry[j + 3],
ry[j + 2], ry[j + 1],ry[j], ry[j + 6])
                    numerator_array7[6] = (ry[j + 6], ry[j + 5], ry[j + 4],
ry[j + 3], ry[j + 2],ry[j+1], ry[j + 7])


                    denominator_array7[0] = (ry[j], ry[j - 1], ry[j - 2],
ry[j - 3], ry[j - 4], ry[j-5], ry[j - k + 1 + 2])
                    denominator_array7[1] = (ry[j + 1], ry[j], ry[j - 1],
ry[j - 2], ry[j - 3], ry[j-4],ry[j - k + 2])
                    denominator_array7[2] = (ry[j + 2], ry[j + 1], ry[j],
ry[j - 1], ry[j - 2], ry[j-3],ry[j - k + 3])
                    denominator_array7[3] = (ry[j + 3], ry[j + 2], ry[j + 1],
ry[j], ry[j - 1],ry[j-2], ry[j - k + 4])
                    denominator_array7[4] = (ry[j + 4], ry[j + 3], ry[j + 2],
ry[j + 1], ry[j],ry[j-1], ry[j - k + 5])
                    denominator_array7[5] = (ry[j + 5], ry[j + 4], ry[j + 3],
ry[j + 2], ry[j + 1],ry[j], ry[j - k + 6])
                    denominator_array7[6] = (ry[j + 6], ry[j + 5], ry[j + 4],
ry[j + 3], ry[j + 2], ry[j+1], ry[j - k + 7])
                    phi_k7.append(np.linalg.det(numerator_array7) /
np.linalg.det(denominator_array7))

                elif j > 5:
                    numerator_array7[0] = (ry[j], ry[j - 1], ry[j - 2], ry[j
- 3], ry[j - 4], ry[j-5], ry[j + 1])
                    numerator_array7[1] = (ry[j + 1], ry[j], ry[j - 1], ry[j
- 2], ry[j - 3],ry[j-4], ry[j + 2])
                    numerator_array7[2] = (ry[j + 2], ry[j + 1], ry[j], ry[j
- 1], ry[j - 2], ry[j-3],ry[j + 3])
                    numerator_array7[3] = (ry[j + 3], ry[j + 2], ry[j + 1],
ry[j], ry[j - 1],ry[j-2], ry[j + 4])
                    numerator_array7[4] = (ry[j + 4], ry[j + 3], ry[j + 2],
ry[j + 1], ry[j],ry[j-1], ry[j + 5])
                    numerator_array7[5] = (ry[j + 5], ry[j + 4], ry[j + 3],
ry[j + 2], ry[j + 1],ry[j], ry[j + 6])
                    numerator_array7[6] = (ry[j + 6], ry[j + 5], ry[j + 4],
ry[j + 3], ry[j + 2],ry[j+1], ry[j + 7])

                    denominator_array7[0] = (ry[j], ry[j - 1], ry[j - 2],
ry[j - 3], ry[j - 4], ry[j-5], ry[j - k + 1])
```

```
                    denominator_array7[1] = (ry[j + 1], ry[j], ry[j - 1], ry[j - 2], ry[j - 3], ry[j-4],ry[j - k + 2])
10.                     denominator_array7[2] = (ry[j + 2], ry[j + 1], ry[j], ry[j - 1], ry[j - 2], ry[j-3],ry[j - k + 3])
                    denominator_array7[3] = (ry[j + 3], ry[j + 2], ry[j + 1], ry[j], ry[j - 1],ry[j-2], ry[j - k + 4])
                    denominator_array7[4] = (ry[j + 4], ry[j + 3], ry[j + 2], ry[j + 1], ry[j],ry[j-1], ry[j - k + 5])
                    denominator_array7[5] = (ry[j + 5], ry[j + 4], ry[j + 3], ry[j + 2], ry[j + 1],ry[j], ry[j - k + 6])
                    denominator_array7[6] = (ry[j + 6], ry[j + 5], ry[j + 4], ry[j + 3], ry[j + 2], ry[j+1], ry[j - k + 7])

                    phi_k7.append(np.linalg.det(numerator_array7) / np.linalg.det(denominator_array7))

            elif k==8:
                numerator_array8 = np.zeros((k, k))
                denominator_array8 = np.zeros((k, k))

                if j==0:
                    numerator_array8[0] = (ry[j], ry[j - 1 + 2], ry[j - 2+4], ry[j - 3+6], ry[j - 4+8], ry[j - 5+10], ry[j-6+12], ry[j + 1])
                    numerator_array8[1] = (ry[j + 1], ry[j], ry[j - 1+2], ry[j - 2+4], ry[j - 3+6], ry[j - 4+8],ry[j-5+10], ry[j + 2])
                    numerator_array8[2] = (ry[j + 2], ry[j + 1], ry[j], ry[j - 1+2], ry[j - 2+4], ry[j - 3+6],ry[j-4+8], ry[j + 3])
                    numerator_array8[3] = (ry[j + 3], ry[j + 2], ry[j + 1], ry[j], ry[j - 1+2], ry[j - 2+4],ry[j-3+6], ry[j + 4])
                    numerator_array8[4] = (ry[j + 4], ry[j + 3], ry[j + 2], ry[j + 1], ry[j], ry[j - 1+2],ry[j-2+4], ry[j + 5])
                    numerator_array8[5] = (ry[j + 5], ry[j + 4], ry[j + 3], ry[j + 2], ry[j + 1], ry[j],ry[j-1+2], ry[j + 6])
                    numerator_array8[6] = (ry[j + 6], ry[j + 5], ry[j + 4], ry[j + 3], ry[j + 2], ry[j + 1],ry[j], ry[j + 7])
                    numerator_array8[7] = (ry[j + 7], ry[j + 6], ry[j + 5], ry[j + 4], ry[j + 3], ry[j + 2],ry[j+1], ry[j + 8])

                    denominator_array8[0] = (ry[j], ry[j - 1+2], ry[j - 2+4], ry[j - 3+6], ry[j - 4+8], ry[j - 5+10], ry[j-6+12], ry[j - k + 1 +14])
                    denominator_array8[1] = (ry[j + 1], ry[j], ry[j - 1+2], ry[j - 2+4], ry[j - 3+6], ry[j - 4+8],ry[j-5+10], ry[j - k + 2+12])
                    denominator_array8[2] = (ry[j + 2], ry[j + 1], ry[j], ry[j - 1+2], ry[j - 2+4], ry[j - 3+6],ry[j-4+8], ry[j - k + 3+10])
                    denominator_array8[3] = (ry[j + 3], ry[j + 2], ry[j + 1], ry[j], ry[j - 1+2], ry[j - 2+4],ry[j-3+6], ry[j - k + 4+8])
                    denominator_array8[4] = (ry[j + 4], ry[j + 3], ry[j + 2], ry[j + 1], ry[j], ry[j - 1+2],ry[j-2+4], ry[j - k + 5+6])
                    denominator_array8[5] = (ry[j + 5], ry[j + 4], ry[j + 3], ry[j + 2], ry[j + 1], ry[j],ry[j-1+2], ry[j - k + 6+4])
                    denominator_array8[6] = (ry[j + 6], ry[j + 5], ry[j + 4], ry[j + 3], ry[j + 2], ry[j + 1],ry[j], ry[j - k + 7+2])
                    denominator_array8[7] = (ry[j + 7], ry[j + 6], ry[j + 5], ry[j + 4], ry[j + 3], ry[j + 2],ry[j+1], ry[j - k + 8])
```

```
                    phi_k8.append(np.linalg.det(numerator_array8) /
np.linalg.det(denominator_array8))

                if j==1:
                    numerator_array8[0] = (ry[j], ry[j - 1], ry[j -
2+2], ry[j - 3+4], ry[j - 4+6], ry[j - 5+8], ry[j-6+10], ry[j + 1])
                    numerator_array8[1] = (ry[j + 1], ry[j], ry[j - 1],
ry[j - 2+2], ry[j - 3+4], ry[j - 4+6],ry[j-5+8], ry[j + 2])
                    numerator_array8[2] = (ry[j + 2], ry[j + 1], ry[j],
ry[j - 1], ry[j - 2+2], ry[j - 3+4],ry[j-4+6], ry[j + 3])
                    numerator_array8[3] = (ry[j + 3], ry[j + 2], ry[j +
1], ry[j], ry[j - 1], ry[j - 2+2],ry[j-3+4], ry[j + 4])
                    numerator_array8[4] = (ry[j + 4], ry[j + 3], ry[j +
2], ry[j + 1], ry[j], ry[j - 1],ry[j-2+2], ry[j + 5])
                    numerator_array8[5] = (ry[j + 5], ry[j + 4], ry[j +
3], ry[j + 2], ry[j + 1], ry[j],ry[j-1], ry[j + 6])
                    numerator_array8[6] = (ry[j + 6], ry[j + 5], ry[j +
4], ry[j + 3], ry[j + 2], ry[j + 1],ry[j], ry[j + 7])
                    numerator_array8[7] = (ry[j + 7], ry[j + 6], ry[j +
5], ry[j + 4], ry[j + 3], ry[j + 2],ry[j+1], ry[j + 8])

                    denominator_array8[0] = (ry[j], ry[j - 1], ry[j -
2+2], ry[j - 3+4], ry[j - 4+6], ry[j - 5+8], ry[j-6+10], ry[j - k +
1+12])
                    denominator_array8[1] = (ry[j + 1], ry[j], ry[j -
1], ry[j - 2+2], ry[j - 3+4], ry[j - 4+6],ry[j-5+8], ry[j - k + 2+10])
                    denominator_array8[2] = (ry[j + 2], ry[j + 1],
ry[j], ry[j - 1], ry[j - 2+2], ry[j - 3+4],ry[j-4+6], ry[j - k + 3+8])
                    denominator_array8[3] = (ry[j + 3], ry[j + 2], ry[j
+ 1], ry[j], ry[j - 1], ry[j - 2+2],ry[j-3+4], ry[j - k + 4+6])
                    denominator_array8[4] = (ry[j + 4], ry[j + 3], ry[j
+ 2], ry[j + 1], ry[j], ry[j - 1],ry[j-2+2], ry[j - k + 5+4])
                    denominator_array8[5] = (ry[j + 5], ry[j + 4], ry[j
+ 3], ry[j + 2], ry[j + 1], ry[j],ry[j-1], ry[j - k + 6+2])
                    denominator_array8[6] = (ry[j + 6], ry[j + 5], ry[j
+ 4], ry[j + 3], ry[j + 2], ry[j + 1],ry[j], ry[j - k + 7])
                    denominator_array8[7] = (ry[j + 7], ry[j + 6], ry[j
+ 5], ry[j + 4], ry[j + 3], ry[j + 2],ry[j+1], ry[j - k + 8])
                    phi_k8.append(np.linalg.det(numerator_array8) /
np.linalg.det(denominator_array8))

                elif j==2:
                    numerator_array8[0] = (ry[j], ry[j - 1], ry[j - 2],
ry[j - 3+2], ry[j - 4+4], ry[j - 5+6], ry[j-6+8], ry[j + 1])
                    numerator_array8[1] = (ry[j + 1], ry[j], ry[j - 1],
ry[j - 2], ry[j - 3+2], ry[j - 4+4],ry[j-5+6], ry[j + 2])
                    numerator_array8[2] = (ry[j + 2], ry[j + 1], ry[j],
ry[j - 1], ry[j - 2], ry[j - 3],ry[j-4+4], ry[j + 3])
                    numerator_array8[3] = (ry[j + 3], ry[j + 2], ry[j +
1], ry[j], ry[j - 1], ry[j - 2],ry[j-3], ry[j + 4])
                    numerator_array8[4] = (ry[j + 4], ry[j + 3], ry[j +
2], ry[j + 1], ry[j], ry[j - 1],ry[j-2], ry[j + 5])
                    numerator_array8[5] = (ry[j + 5], ry[j + 4], ry[j +
3], ry[j + 2], ry[j + 1], ry[j],ry[j-1], ry[j + 6])
                    numerator_array8[6] = (ry[j + 6], ry[j + 5], ry[j +
4], ry[j + 3], ry[j + 2], ry[j + 1],ry[j], ry[j + 7])
                    numerator_array8[7] = (ry[j + 7], ry[j + 6], ry[j +
```

```
5], ry[j + 4], ry[j + 3], ry[j + 2],ry[j+1], ry[j + 8])

                    denominator_array8[0] = (ry[j], ry[j - 1], ry[j -
2], ry[j - 3], ry[j - 4+4], ry[j - 5+6], ry[j-6+8], ry[j - k + 1+10])
                    denominator_array8[1] = (ry[j + 1], ry[j], ry[j -
1], ry[j - 2], ry[j - 3], ry[j - 4+4],ry[j-5+6], ry[j - k + 2+8])
                    denominator_array8[2] = (ry[j + 2], ry[j + 1],
ry[j], ry[j - 1], ry[j - 2], ry[j - 3],ry[j-4+4], ry[j - k + 3+6])
                    denominator_array8[3] = (ry[j + 3], ry[j + 2], ry[j
+ 1], ry[j], ry[j - 1], ry[j - 2],ry[j-3], ry[j - k + 4+4])
                    denominator_array8[4] = (ry[j + 4], ry[j + 3], ry[j
+ 2], ry[j + 1], ry[j], ry[j - 1],ry[j-2], ry[j - k + 5+2])
                    denominator_array8[5] = (ry[j + 5], ry[j + 4], ry[j
+ 3], ry[j + 2], ry[j + 1], ry[j],ry[j-1], ry[j - k + 6])
                    denominator_array8[6] = (ry[j + 6], ry[j + 5], ry[j
+ 4], ry[j + 3], ry[j + 2], ry[j + 1],ry[j], ry[j - k + 7])
                    denominator_array8[7] = (ry[j + 7], ry[j + 6], ry[j
+ 5], ry[j + 4], ry[j + 3], ry[j + 2],ry[j+1], ry[j - k + 8])
                    phi_k8.append(np.linalg.det(numerator_array8) /
np.linalg.det(denominator_array8))

                elif j==3:
                    numerator_array8[0] = (ry[j], ry[j - 1], ry[j - 2],
ry[j - 3], ry[j - 4+2], ry[j - 5+4], ry[j-6+6], ry[j + 1])
                    numerator_array8[1] = (ry[j + 1], ry[j], ry[j - 1],
ry[j - 2], ry[j - 3], ry[j - 4+2],ry[j-5+4], ry[j + 2])
                    numerator_array8[2] = (ry[j + 2], ry[j + 1], ry[j],
ry[j - 1], ry[j - 2], ry[j - 3],ry[j-4+2], ry[j + 3])
                    numerator_array8[3] = (ry[j + 3], ry[j + 2], ry[j +
1], ry[j], ry[j - 1], ry[j - 2],ry[j-3], ry[j + 4])
                    numerator_array8[4] = (ry[j + 4], ry[j + 3], ry[j +
2], ry[j + 1], ry[j], ry[j - 1],ry[j-2], ry[j + 5])
                    numerator_array8[5] = (ry[j + 5], ry[j + 4], ry[j +
3], ry[j + 2], ry[j + 1], ry[j],ry[j-1], ry[j + 6])
                    numerator_array8[6] = (ry[j + 6], ry[j + 5], ry[j +
4], ry[j + 3], ry[j + 2], ry[j + 1],ry[j], ry[j + 7])
                    numerator_array8[7] = (ry[j + 7], ry[j + 6], ry[j +
5], ry[j + 4], ry[j + 3], ry[j + 2],ry[j+1], ry[j + 8])

                    denominator_array8[0] = (ry[j], ry[j - 1], ry[j -
2], ry[j - 3], ry[j - 4+2], ry[j - 5+4], ry[j-6+6], ry[j - k + 1+8])
                    denominator_array8[1] = (ry[j + 1], ry[j], ry[j -
1], ry[j - 2], ry[j - 3], ry[j - 4+2],ry[j-5+4], ry[j - k + 2+6])
                    denominator_array8[2] = (ry[j + 2], ry[j + 1],
ry[j], ry[j - 1], ry[j - 2], ry[j - 3],ry[j-4+2], ry[j - k + 3+4])
                    denominator_array8[3] = (ry[j + 3], ry[j + 2], ry[j
+ 1], ry[j], ry[j - 1], ry[j - 2],ry[j-3], ry[j - k + 4+2])
                    denominator_array8[4] = (ry[j + 4], ry[j + 3], ry[j
+ 2], ry[j + 1], ry[j], ry[j - 1],ry[j-2], ry[j - k + 5])
                    denominator_array8[5] = (ry[j + 5], ry[j + 4], ry[j
+ 3], ry[j + 2], ry[j + 1], ry[j],ry[j-1], ry[j - k + 6])
                    denominator_array8[6] = (ry[j + 6], ry[j + 5], ry[j
+ 4], ry[j + 3], ry[j + 2], ry[j + 1],ry[j], ry[j - k + 7])
                    denominator_array8[7] = (ry[j + 7], ry[j + 6], ry[j
+ 5], ry[j + 4], ry[j + 3], ry[j + 2],ry[j+1], ry[j - k + 8])
                    phi_k8.append(np.linalg.det(numerator_array8) /
np.linalg.det(denominator_array8))
```

```
                elif j==4:
                    numerator_array8[0] = (ry[j], ry[j - 1], ry[j - 2], 
ry[j - 3], ry[j - 4], ry[j - 5+2], ry[j-6+4], ry[j + 1])
                    numerator_array8[1] = (ry[j + 1], ry[j], ry[j - 1], 
ry[j - 2], ry[j - 3], ry[j - 4],ry[j-5+2], ry[j + 2])
                    numerator_array8[2] = (ry[j + 2], ry[j + 1], ry[j], 
ry[j - 1], ry[j - 2], ry[j - 3],ry[j-4], ry[j + 3])
                    numerator_array8[3] = (ry[j + 3], ry[j + 2], ry[j + 
1], ry[j], ry[j - 1], ry[j - 2],ry[j-3], ry[j + 4])
                    numerator_array8[4] = (ry[j + 4], ry[j + 3], ry[j + 
2], ry[j + 1], ry[j], ry[j - 1],ry[j-2], ry[j + 5])
                    numerator_array8[5] = (ry[j + 5], ry[j + 4], ry[j + 
3], ry[j + 2], ry[j + 1], ry[j],ry[j-1], ry[j + 6])
                    numerator_array8[6] = (ry[j + 6], ry[j + 5], ry[j + 
4], ry[j + 3], ry[j + 2], ry[j + 1],ry[j], ry[j + 7])
                    numerator_array8[7] = (ry[j + 7], ry[j + 6], ry[j + 
5], ry[j + 4], ry[j + 3], ry[j + 2],ry[j+1], ry[j + 8])

                    denominator_array8[0] = (ry[j], ry[j - 1], ry[j - 
2], ry[j - 3], ry[j - 4], ry[j - 5+2], ry[j-6+4], ry[j - k + 1+6])
                    denominator_array8[1] = (ry[j + 1], ry[j], ry[j - 
1], ry[j - 2], ry[j - 3], ry[j - 4],ry[j-5+2], ry[j - k + 2+4])
                    denominator_array8[2] = (ry[j + 2], ry[j + 1], 
ry[j], ry[j - 1], ry[j - 2], ry[j - 3],ry[j-4], ry[j - k + 3+2])
                    denominator_array8[3] = (ry[j + 3], ry[j + 2], ry[j 
+ 1], ry[j], ry[j - 1], ry[j - 2],ry[j-3], ry[j - k + 4])
                    denominator_array8[4] = (ry[j + 4], ry[j + 3], ry[j 
+ 2], ry[j + 1], ry[j], ry[j - 1],ry[j-2], ry[j - k + 5])
                    denominator_array8[5] = (ry[j + 5], ry[j + 4], ry[j 
+ 3], ry[j + 2], ry[j + 1], ry[j],ry[j-1], ry[j - k + 6])
                    denominator_array8[6] = (ry[j + 6], ry[j + 5], ry[j 
+ 4], ry[j + 3], ry[j + 2], ry[j + 1],ry[j], ry[j - k + 7])
                    denominator_array8[7] = (ry[j + 7], ry[j + 6], ry[j 
+ 5], ry[j + 4], ry[j + 3], ry[j + 2],ry[j+1], ry[j - k + 8])
                    phi_k8.append(np.linalg.det(numerator_array8) / 
np.linalg.det(denominator_array8))

                elif j==5:
                    numerator_array8[0] = (ry[j], ry[j - 1], ry[j - 2], 
ry[j - 3], ry[j - 4], ry[j - 5], ry[j-6+2], ry[j + 1])
                    numerator_array8[1] = (ry[j + 1], ry[j], ry[j - 1], 
ry[j - 2], ry[j - 3], ry[j - 4],ry[j-5], ry[j + 2])
                    numerator_array8[2] = (ry[j + 2], ry[j + 1], ry[j], 
ry[j - 1], ry[j - 2], ry[j - 3],ry[j-4], ry[j + 3])
                    numerator_array8[3] = (ry[j + 3], ry[j + 2], ry[j + 
1], ry[j], ry[j - 1], ry[j - 2],ry[j-3], ry[j + 4])
                    numerator_array8[4] = (ry[j + 4], ry[j + 3], ry[j + 
2], ry[j + 1], ry[j], ry[j - 1],ry[j-2], ry[j + 5])
                    numerator_array8[5] = (ry[j + 5], ry[j + 4], ry[j + 
3], ry[j + 2], ry[j + 1], ry[j],ry[j-1], ry[j + 6])
                    numerator_array8[6] = (ry[j + 6], ry[j + 5], ry[j + 
4], ry[j + 3], ry[j + 2], ry[j + 1],ry[j], ry[j + 7])
                    numerator_array8[7] = (ry[j + 7], ry[j + 6], ry[j + 
5], ry[j + 4], ry[j + 3], ry[j + 2],ry[j+1], ry[j + 8])

                    denominator_array8[0] = (ry[j], ry[j - 1], ry[j - 
```

```
2], ry[j - 3], ry[j - 4], ry[j - 5], ry[j-6+2], ry[j - k + 1+4])
                    denominator_array8[1] = (ry[j + 1], ry[j], ry[j -
1], ry[j - 2], ry[j - 3], ry[j - 4],ry[j-5], ry[j - k + 2+2])
                    denominator_array8[2] = (ry[j + 2], ry[j + 1],
ry[j], ry[j - 1], ry[j - 2], ry[j - 3],ry[j-4], ry[j - k + 3])
                    denominator_array8[3] = (ry[j + 3], ry[j + 2], ry[j
+ 1], ry[j], ry[j - 1], ry[j - 2],ry[j-3], ry[j - k + 4])
                    denominator_array8[4] = (ry[j + 4], ry[j + 3], ry[j
+ 2], ry[j + 1], ry[j], ry[j - 1],ry[j-2], ry[j - k + 5])
                    denominator_array8[5] = (ry[j + 5], ry[j + 4], ry[j
+ 3], ry[j + 2], ry[j + 1], ry[j],ry[j-1], ry[j - k + 6])
                    denominator_array8[6] = (ry[j + 6], ry[j + 5], ry[j
+ 4], ry[j + 3], ry[j + 2], ry[j + 1],ry[j], ry[j - k + 7])
                    denominator_array8[7] = (ry[j + 7], ry[j + 6], ry[j
+ 5], ry[j + 4], ry[j + 3], ry[j + 2],ry[j+1], ry[j - k + 8])
                    phi_k8.append(np.linalg.det(numerator_array8) /
np.linalg.det(denominator_array8))

                elif j==6:
                    numerator_array8[0] = (ry[j], ry[j - 1], ry[j - 2],
ry[j - 3], ry[j - 4], ry[j - 5], ry[j-6], ry[j + 1])
                    numerator_array8[1] = (ry[j + 1], ry[j], ry[j - 1],
ry[j - 2], ry[j - 3], ry[j - 4],ry[j-5], ry[j + 2])
                    numerator_array8[2] = (ry[j + 2], ry[j + 1], ry[j],
ry[j - 1], ry[j - 2], ry[j - 3],ry[j-4], ry[j + 3])
                    numerator_array8[3] = (ry[j + 3], ry[j + 2], ry[j +
1], ry[j], ry[j - 1], ry[j - 2],ry[j-3], ry[j + 4])
                    numerator_array8[4] = (ry[j + 4], ry[j + 3], ry[j +
2], ry[j + 1], ry[j], ry[j - 1],ry[j-2], ry[j + 5])
                    numerator_array8[5] = (ry[j + 5], ry[j + 4], ry[j +
3], ry[j + 2], ry[j + 1], ry[j],ry[j-1], ry[j + 6])
                    numerator_array8[6] = (ry[j + 6], ry[j + 5], ry[j +
4], ry[j + 3], ry[j + 2], ry[j + 1],ry[j], ry[j + 7])
                    numerator_array8[7] = (ry[j + 7], ry[j + 6], ry[j +
5], ry[j + 4], ry[j + 3], ry[j + 2],ry[j+1], ry[j + 8])

                    denominator_array8[0] = (ry[j], ry[j - 1], ry[j -
2], ry[j - 3], ry[j - 4], ry[j - 5], ry[j-6], ry[j - k + 1 +2])
                    denominator_array8[1] = (ry[j + 1], ry[j], ry[j -
1], ry[j - 2], ry[j - 3], ry[j - 4],ry[j-5], ry[j - k + 2])
                    denominator_array8[2] = (ry[j + 2], ry[j + 1],
ry[j], ry[j - 1], ry[j - 2], ry[j - 3],ry[j-4], ry[j - k + 3])
                    denominator_array8[3] = (ry[j + 3], ry[j + 2], ry[j
+ 1], ry[j], ry[j - 1], ry[j - 2],ry[j-3], ry[j - k + 4])
                    denominator_array8[4] = (ry[j + 4], ry[j + 3], ry[j
+ 2], ry[j + 1], ry[j], ry[j - 1],ry[j-2], ry[j - k + 5])
                    denominator_array8[5] = (ry[j + 5], ry[j + 4], ry[j
+ 3], ry[j + 2], ry[j + 1], ry[j],ry[j-1], ry[j - k + 6])
                    denominator_array8[6] = (ry[j + 6], ry[j + 5], ry[j
+ 4], ry[j + 3], ry[j + 2], ry[j + 1],ry[j], ry[j - k + 7])
                    denominator_array8[7] = (ry[j + 7], ry[j + 6], ry[j
+ 5], ry[j + 4], ry[j + 3], ry[j + 2],ry[j+1], ry[j - k + 8])
                    phi_k8.append(np.linalg.det(numerator_array8) /
np.linalg.det(denominator_array8))


                elif j>6:
```

```
                    numerator_array8[0] = (ry[j], ry[j - 1], ry[j - 2],
ry[j - 3], ry[j - 4], ry[j - 5], ry[j-6], ry[j + 1])
                    numerator_array8[1] = (ry[j + 1], ry[j], ry[j - 1],
ry[j - 2], ry[j - 3], ry[j - 4],ry[j-5], ry[j + 2])
                    numerator_array8[2] = (ry[j + 2], ry[j + 1], ry[j],
ry[j - 1], ry[j - 2], ry[j - 3],ry[j-4], ry[j + 3])
                    numerator_array8[3] = (ry[j + 3], ry[j + 2], ry[j +
1], ry[j], ry[j - 1], ry[j - 2],ry[j-3], ry[j + 4])
                    numerator_array8[4] = (ry[j + 4], ry[j + 3], ry[j +
2], ry[j + 1], ry[j], ry[j - 1],ry[j-2], ry[j + 5])
                    numerator_array8[5] = (ry[j + 5], ry[j + 4], ry[j +
3], ry[j + 2], ry[j + 1], ry[j],ry[j-1], ry[j + 6])
                    numerator_array8[6] = (ry[j + 6], ry[j + 5], ry[j +
4], ry[j + 3], ry[j + 2], ry[j + 1],ry[j], ry[j + 7])
                    numerator_array8[7] = (ry[j + 7], ry[j + 6], ry[j +
5], ry[j + 4], ry[j + 3], ry[j + 2],ry[j+1], ry[j + 8])

                    denominator_array8[0] = (ry[j], ry[j - 1], ry[j -
2], ry[j - 3], ry[j - 4], ry[j - 5], ry[j-6], ry[j - k + 1])
                    denominator_array8[1] = (ry[j + 1], ry[j], ry[j -
1], ry[j - 2], ry[j - 3], ry[j - 4],ry[j-5], ry[j - k + 2])
                    denominator_array8[2] = (ry[j + 2], ry[j + 1],
ry[j], ry[j - 1], ry[j - 2], ry[j - 3],ry[j-4], ry[j - k + 3])
                    denominator_array8[3] = (ry[j + 3], ry[j + 2], ry[j
+ 1], ry[j], ry[j - 1], ry[j - 2],ry[j-3], ry[j - k + 4])
                    denominator_array8[4] = (ry[j + 4], ry[j + 3], ry[j
+ 2], ry[j + 1], ry[j], ry[j - 1],ry[j-2], ry[j - k + 5])
                    denominator_array8[5] = (ry[j + 5], ry[j + 4], ry[j
+ 3], ry[j + 2], ry[j + 1], ry[j],ry[j-1], ry[j - k + 6])
                    denominator_array8[6] = (ry[j + 6], ry[j + 5], ry[j
+ 4], ry[j + 3], ry[j + 2], ry[j + 1],ry[j], ry[j - k + 7])
                    denominator_array8[7] = (ry[j + 7], ry[j + 6], ry[j
+ 5], ry[j + 4], ry[j + 3], ry[j + 2],ry[j+1], ry[j - k + 8])
                    phi_k8.append(np.linalg.det(numerator_array8) /
np.linalg.det(denominator_array8))
            elif k ==9:
                numerator_array9 = np.zeros((k, k))
                denominator_array9 = np.zeros((k, k))
                numerator_array9[0] = (ry[j], ry[abs(j - 1)], ry[abs(j
- 2)], ry[abs(j - 3)], ry[abs(j - 4)], ry[abs(j - 5)], ry[abs(j-6)],
ry[abs(j -7)], ry[j + 1])
                numerator_array9[1] = (ry[j + 1], ry[j], ry[abs(j -
1)], ry[abs(j - 2)], ry[abs(j - 3)], ry[abs(j - 4)],ry[abs(j-5)],
ry[abs(j -6)],ry[j + 2])
                numerator_array9[2] = (ry[j + 2], ry[j + 1], ry[j],
ry[abs(j - 1)], ry[abs(j - 2)], ry[abs(j - 3)],ry[abs(j-4)], ry[abs(j -
5)],ry[j + 3])
                numerator_array9[3] = (ry[j + 3], ry[j + 2], ry[j + 1],
ry[j], ry[abs(j - 1)], ry[abs(j - 2)],ry[abs(j-3)], ry[abs(j -4)],ry[j
+ 4])
                numerator_array9[4] = (ry[j + 4], ry[j + 3], ry[j + 2],
ry[j + 1], ry[j], ry[abs(j - 1)],ry[abs(j-2)], ry[abs(j -3)],ry[j + 5])
                numerator_array9[5] = (ry[j + 5], ry[j + 4], ry[j + 3],
ry[j + 2], ry[j + 1], ry[j],ry[abs(j-1)], ry[abs(j -2)],ry[j + 6])
                numerator_array9[6] = (ry[j + 6], ry[j + 5], ry[j + 4],
ry[j + 3], ry[j + 2], ry[j + 1],ry[j], ry[abs(j -1)],ry[j + 7])
                numerator_array9[7] = (ry[j + 7], ry[j + 6], ry[j + 5],
```

```
ry[j + 4], ry[j + 3], ry[j + 2],ry[j+1], ry[j],ry[j + 8])
                numerator_array9[8] = (ry[j + 8], ry[j + 7], ry[j + 6],
ry[j + 5], ry[j + 4], ry[j + 3],ry[j+2], ry[j+1],ry[j + 9])


                for i in range(0,k):
                    denominator_array9[i][0:-1] =
numerator_array9[i][0:-1]
                    denominator_array9[i][-1] = ry[abs(j - k + i + 1)]

                phi_k9.append(np.linalg.det(numerator_array9) /
np.linalg.det(denominator_array9))

            elif k ==10:
                numerator_array10 = np.zeros((k, k))
                denominator_array10 = np.zeros((k, k))
                numerator_array10[0] = (ry[j], ry[abs(j - 1)], ry[abs(j
- 2)], ry[abs(j - 3)], ry[abs(j - 4)], ry[abs(j - 5)], ry[abs(j-6)],
ry[abs(j -7)],ry[abs(j -8)], ry[j + 1])
                numerator_array10[1] = (ry[j + 1], ry[j], ry[abs(j -
1)], ry[abs(j - 2)], ry[abs(j - 3)], ry[abs(j - 4)],ry[abs(j-5)],
ry[abs(j -6)],ry[abs(j -7)],ry[j + 2])
                numerator_array10[2] = (ry[j + 2], ry[j + 1], ry[j],
ry[abs(j - 1)], ry[abs(j - 2)], ry[abs(j - 3)],ry[abs(j-4)], ry[abs(j -
5)],ry[abs(j -6)],ry[j + 3])
                numerator_array10[3] = (ry[j + 3], ry[j + 2], ry[j +
1], ry[j], ry[abs(j - 1)], ry[abs(j - 2)],ry[abs(j-3)], ry[abs(j -
4)],ry[abs(j -5)],ry[j + 4])
                numerator_array10[4] = (ry[j + 4], ry[j + 3], ry[j +
2], ry[j + 1], ry[j], ry[abs(j - 1)],ry[abs(j-2)], ry[abs(j -
3)],ry[abs(j -4)],ry[j + 5])
                numerator_array10[5] = (ry[j + 5], ry[j + 4], ry[j +
3], ry[j + 2], ry[j + 1], ry[j],ry[abs(j-1)], ry[abs(j -2)],ry[abs(j -
3)],ry[j + 6])
                numerator_array10[6] = (ry[j + 6], ry[j + 5], ry[j +
4], ry[j + 3], ry[j + 2], ry[j + 1],ry[j], ry[abs(j -1)],ry[abs(j -
2)],ry[j + 7])
                numerator_array10[7] = (ry[j + 7], ry[j + 6], ry[j +
5], ry[j + 4], ry[j + 3], ry[j + 2],ry[j+1], ry[j],ry[abs(j -1)],ry[j +
8])
                numerator_array10[8] = (ry[j + 8], ry[j + 7], ry[j +
6], ry[j + 5], ry[j + 4], ry[j + 3],ry[j+2], ry[j+1],ry[abs(j)],ry[j +
9])
                numerator_array10[9] = (ry[j + 9], ry[j + 8], ry[j +
7], ry[j + 6], ry[j + 5], ry[j + 4], ry[j + 3], ry[j + 2], ry[j+1],ry[j
+ 10])

                for i in range(0,k):
                    denominator_array10[i][0:-1] =
numerator_array10[i][0:-1]
                    denominator_array10[i][-1] = ry[abs(j - k + i + 1)]

                phi_k10.append(np.linalg.det(numerator_array10) /
np.linalg.det(denominator_array10))

            elif k ==11:
                numerator_array11 = np.zeros((k, k))
```

```
                denominator_array11 = np.zeros((k, k))
                numerator_array11[0] = (ry[j], ry[abs(j - 1)], ry[abs(j
- 2)], ry[abs(j - 3)], ry[abs(j - 4)], ry[abs(j - 5)], ry[abs(j-6)],
ry[abs(j -7)],ry[abs(j -8)],ry[abs(j -9)],  ry[j + 1])
                numerator_array11[1] = (ry[j + 1], ry[j], ry[abs(j -
1)], ry[abs(j - 2)], ry[abs(j - 3)], ry[abs(j - 4)],ry[abs(j-5)],
ry[abs(j -6)],ry[abs(j -7)],ry[abs(j -8)],ry[j + 2])
                numerator_array11[2] = (ry[j + 2], ry[j + 1], ry[j],
ry[abs(j - 1)], ry[abs(j - 2)], ry[abs(j - 3)],ry[abs(j-4)], ry[abs(j -
5)],ry[abs(j -6)],ry[abs(j -7)],ry[j + 3])
                numerator_array11[3] = (ry[j + 3], ry[j + 2], ry[j +
1], ry[j], ry[abs(j - 1)], ry[abs(j - 2)],ry[abs(j-3)], ry[abs(j -
4)],ry[abs(j -5)],ry[abs(j -6)],ry[j + 4])
                numerator_array11[4] = (ry[j + 4], ry[j + 3], ry[j +
2], ry[j + 1], ry[j], ry[abs(j - 1)],ry[abs(j-2)], ry[abs(j -
3)],ry[abs(j -4)],ry[abs(j -5)],ry[j + 5])
                numerator_array11[5] = (ry[j + 5], ry[j + 4], ry[j +
3], ry[j + 2], ry[j + 1], ry[j],ry[abs(j-1)], ry[abs(j -2)],ry[abs(j -
3)],ry[abs(j -4)],ry[j + 6])
                numerator_array11[6] = (ry[j + 6], ry[j + 5], ry[j +
4], ry[j + 3], ry[j + 2], ry[j + 1],ry[j], ry[abs(j -1)],ry[abs(j -
2)],ry[abs(j -3)],ry[j + 7])
                numerator_array11[7] = (ry[j + 7], ry[j + 6], ry[j +
5], ry[j + 4], ry[j + 3], ry[j + 2],ry[j+1], ry[j],ry[abs(j -
1)],ry[abs(j -2)],ry[j + 8])
                numerator_array11[8] = (ry[j + 8], ry[j + 7], ry[j +
6], ry[j + 5], ry[j + 4], ry[j + 3],ry[j+2],
ry[j+1],ry[abs(j)],ry[abs(j -1)],ry[j + 9])
                numerator_array11[9] = (ry[j + 9], ry[j + 8], ry[j +
7], ry[j + 6], ry[j + 5], ry[j + 4], ry[j + 3], ry[j + 2],
ry[j+1],ry[j],ry[j + 10])
                numerator_array11[10] = (ry[j + 10], ry[j + 9], ry[j +
8], ry[j + 7], ry[j + 6], ry[j + 5], ry[j + 4], ry[j + 3],
ry[j+2],ry[j+1],ry[j + 11])

                for i in range(0,k):
                    denominator_array11[i][0:-1] =
numerator_array11[i][0:-1]
                    denominator_array11[i][-1] = ry[abs(j - k + i + 1)]

                phi_k11.append(np.linalg.det(numerator_array11) /
np.linalg.det(denominator_array11))

            elif k ==12:
                numerator_array12 = np.zeros((k, k))
                denominator_array12 = np.zeros((k, k))
                numerator_array12[0] = (ry[j], ry[abs(j - 1)], ry[abs(j
- 2)], ry[abs(j - 3)], ry[abs(j - 4)], ry[abs(j - 5)], ry[abs(j-6)],
ry[abs(j -7)],ry[abs(j -8)],ry[abs(j -9)],ry[abs(j-10)],  ry[j + 1])
                numerator_array12[1] = (ry[j + 1], ry[j], ry[abs(j -
1)], ry[abs(j - 2)], ry[abs(j - 3)], ry[abs(j - 4)],ry[abs(j-5)],
ry[abs(j -6)],ry[abs(j -7)],ry[abs(j -8)],ry[abs(j-9)],ry[j + 2])
                numerator_array12[2] = (ry[j + 2], ry[j + 1], ry[j],
ry[abs(j - 1)], ry[abs(j - 2)], ry[abs(j - 3)],ry[abs(j-4)], ry[abs(j -
5)],ry[abs(j -6)],ry[abs(j -7)],ry[abs(j-8)],ry[j + 3])
                numerator_array12[3] = (ry[j + 3], ry[j + 2], ry[j +
1], ry[j], ry[abs(j - 1)], ry[abs(j - 2)],ry[abs(j-3)], ry[abs(j -
```

```
4)],ry[abs(j -5)],ry[abs(j -6)],ry[abs(j-7)],ry[j + 4])
                numerator_array12[4] = (ry[j + 4], ry[j + 3], ry[j +
2], ry[j + 1], ry[j], ry[abs(j - 1)],ry[abs(j-2)], ry[abs(j -
3)],ry[abs(j -4)],ry[abs(j -5)],ry[abs(j-6)],ry[j + 5])
                numerator_array12[5] = (ry[j + 5], ry[j + 4], ry[j +
3], ry[j + 2], ry[j + 1], ry[j],ry[abs(j-1)], ry[abs(j -2)],ry[abs(j -
3)],ry[abs(j -4)],ry[abs(j-5)],ry[j + 6])
                numerator_array12[6] = (ry[j + 6], ry[j + 5], ry[j +
4], ry[j + 3], ry[j + 2], ry[j + 1],ry[j], ry[abs(j -1)],ry[abs(j -
2)],ry[abs(j -3)],ry[abs(j-4)],ry[j + 7])
                numerator_array12[7] = (ry[j + 7], ry[j + 6], ry[j +
5], ry[j + 4], ry[j + 3], ry[j + 2],ry[j+1], ry[j],ry[abs(j -
1)],ry[abs(j -2)],ry[abs(j-3)],ry[j + 8])
                numerator_array12[8] = (ry[j + 8], ry[j + 7], ry[j +
6], ry[j + 5], ry[j + 4], ry[j + 3],ry[j+2],
ry[j+1],ry[abs(j)],ry[abs(j -1)],ry[abs(j-2)],ry[j + 9])
                numerator_array12[9] = (ry[j + 9], ry[j + 8], ry[j +
7], ry[j + 6], ry[j + 5], ry[j + 4], ry[j + 3], ry[j + 2],
ry[j+1],ry[j],ry[abs(j-1)],ry[j + 10])
                numerator_array12[10] = (ry[j + 10], ry[j + 9], ry[j +
8], ry[j + 7], ry[j + 6], ry[j + 5], ry[j + 4], ry[j + 3],
ry[j+2],ry[j+1],ry[abs(j)],ry[j + 11])
                numerator_array12[11] = (ry[j + 11], ry[j + 10], ry[j +
9], ry[j + 8], ry[j + 7], ry[j + 6], ry[j + 5], ry[j + 4],
ry[j+3],ry[j+2],ry[abs(j+1)],ry[j + 12])

                for i in range(0,k):
                    denominator_array12[i][0:-1] =
numerator_array12[i][0:-1]
                    denominator_array12[i][-1] = ry[abs(j - k + i + 1)]

                phi_k12.append(np.linalg.det(numerator_array12) /
np.linalg.det(denominator_array12))

            elif k ==13:
                numerator_array13 = np.zeros((k, k))
                denominator_array13 = np.zeros((k, k))
                numerator_array13[0] = (ry[j], ry[abs(j - 1)], ry[abs(j
- 2)], ry[abs(j - 3)], ry[abs(j - 4)], ry[abs(j - 5)], ry[abs(j-6)],
ry[abs(j -7)],ry[abs(j -8)],ry[abs(j -9)],ry[abs(j-10)],ry[abs(j-11)],
ry[j + 1])
                numerator_array13[1] = (ry[j + 1], ry[j], ry[abs(j -
1)], ry[abs(j - 2)], ry[abs(j - 3)], ry[abs(j - 4)],ry[abs(j-5)],
ry[abs(j -6)],ry[abs(j -7)],ry[abs(j -8)],ry[abs(j-9)],ry[abs(j-
10)],ry[j + 2])
                numerator_array13[2] = (ry[j + 2], ry[j + 1], ry[j],
ry[abs(j - 1)], ry[abs(j - 2)], ry[abs(j - 3)],ry[abs(j-4)], ry[abs(j -
5)],ry[abs(j -6)],ry[abs(j -7)],ry[abs(j-8)],ry[abs(j-9)],ry[j + 3])
                numerator_array13[3] = (ry[j + 3], ry[j + 2], ry[j +
1], ry[j], ry[abs(j - 1)], ry[abs(j - 2)],ry[abs(j-3)], ry[abs(j -
4)],ry[abs(j -5)],ry[abs(j -6)],ry[abs(j-7)],ry[abs(j-8)],ry[j + 4])
                numerator_array13[4] = (ry[j + 4], ry[j + 3], ry[j +
2], ry[j + 1], ry[j], ry[abs(j - 1)],ry[abs(j-2)], ry[abs(j -
3)],ry[abs(j -4)],ry[abs(j -5)],ry[abs(j-6)],ry[abs(j-7)],ry[j + 5])
                numerator_array13[5] = (ry[j + 5], ry[j + 4], ry[j +
3], ry[j + 2], ry[j + 1], ry[j],ry[abs(j-1)], ry[abs(j -2)],ry[abs(j -
3)],ry[abs(j -4)],ry[abs(j-5)],ry[abs(j-6)],ry[j + 6])
```

```
                numerator_array13[6] = (ry[j + 6], ry[j + 5], ry[j + 
4], ry[j + 3], ry[j + 2], ry[j + 1],ry[j], ry[abs(j -1)],ry[abs(j -
2)],ry[abs(j -3)],ry[abs(j-4)],ry[abs(j-5)],ry[j + 7])
                numerator_array13[7] = (ry[j + 7], ry[j + 6], ry[j + 
5], ry[j + 4], ry[j + 3], ry[j + 2],ry[j+1], ry[j],ry[abs(j -
1)],ry[abs(j -2)],ry[abs(j-3)],ry[abs(j-4)],ry[j + 8])
                numerator_array13[8] = (ry[j + 8], ry[j + 7], ry[j + 
6], ry[j + 5], ry[j + 4], ry[j + 3],ry[j+2], 
ry[j+1],ry[abs(j)],ry[abs(j -1)],ry[abs(j-2)],ry[abs(j-3)],ry[j + 9])
                numerator_array13[9] = (ry[j + 9], ry[j + 8], ry[j + 
7], ry[j + 6], ry[j + 5], ry[j + 4], ry[j + 3], ry[j + 2], 
ry[j+1],ry[j],ry[abs(j-1)],ry[abs(j-2)],ry[j + 10])
                numerator_array13[10] = (ry[j + 10], ry[j + 9], ry[j + 
8], ry[j + 7], ry[j + 6], ry[j + 5], ry[j + 4], ry[j + 3], 
ry[j+2],ry[j+1],ry[abs(j)],ry[abs(j-1)],ry[j + 11])
                numerator_array13[11] = (ry[j + 11], ry[j + 10], ry[j + 
9], ry[j + 8], ry[j + 7], ry[j + 6], ry[j + 5], ry[j + 4], 
ry[j+3],ry[j+2],ry[abs(j+1)],ry[abs(j)],ry[j + 12])
                numerator_array13[12] = (ry[j + 12], ry[j + 11], ry[j + 
10], ry[j + 9], ry[j + 8], ry[j + 7], ry[j + 6], ry[j + 5], ry[j 
+4],ry[j + 3], ry[abs(j + 2)], ry[abs(j+1)],ry[j + 13])

                for i in range(0,k):
                    denominator_array13[i][0:-1] = 
numerator_array13[i][0:-1]
                    denominator_array13[i][-1] = ry[abs(j - k + i + 1)]

                phi_k13.append(np.linalg.det(numerator_array13) / 
np.linalg.det(denominator_array13))

            elif k ==14:
                numerator_array14 = np.zeros((k, k))
                denominator_array14 = np.zeros((k, k))
                numerator_array14[0] = (ry[j], ry[abs(j - 1)], ry[abs(j 
- 2)], ry[abs(j - 3)], ry[abs(j - 4)], ry[abs(j - 5)], ry[abs(j-6)], 
ry[abs(j -7)],ry[abs(j -8)],ry[abs(j -9)],ry[abs(j-10)],ry[abs(j-
11)],ry[abs(j-12)],ry[j + 1])
                numerator_array14[1] = (ry[j + 1], ry[j], ry[abs(j -
1)], ry[abs(j - 2)], ry[abs(j - 3)], ry[abs(j - 4)],ry[abs(j-5)], 
ry[abs(j -6)],ry[abs(j -7)],ry[abs(j -8)],ry[abs(j-9)],ry[abs(j-
10)],ry[abs(j-11)],ry[j + 2])
                numerator_array14[2] = (ry[j + 2], ry[j + 1], ry[j], 
ry[abs(j - 1)], ry[abs(j - 2)], ry[abs(j - 3)],ry[abs(j-4)], ry[abs(j -
5)],ry[abs(j -6)],ry[abs(j -7)],ry[abs(j-8)],ry[abs(j-9)],ry[abs(j-
10)],ry[j + 3])
                numerator_array14[3] = (ry[j + 3], ry[j + 2], ry[j + 
1], ry[j], ry[abs(j - 1)], ry[abs(j - 2)],ry[abs(j-3)], ry[abs(j -
4)],ry[abs(j -5)],ry[abs(j -6)],ry[abs(j-7)],ry[abs(j-8)],ry[abs(j-
9)],ry[j + 4])
                numerator_array14[4] = (ry[j + 4], ry[j + 3], ry[j + 
2], ry[j + 1], ry[j], ry[abs(j - 1)],ry[abs(j-2)], ry[abs(j -
3)],ry[abs(j -4)],ry[abs(j -5)],ry[abs(j-6)],ry[abs(j-7)],ry[abs(j-
8)],ry[j + 5])
                numerator_array14[5] = (ry[j + 5], ry[j + 4], ry[j + 
3], ry[j + 2], ry[j + 1], ry[j],ry[abs(j-1)], ry[abs(j -2)],ry[abs(j -
3)],ry[abs(j -4)],ry[abs(j-5)],ry[abs(j-6)],ry[abs(j-7)],ry[j + 6])
                numerator_array14[6] = (ry[j + 6], ry[j + 5], ry[j + 
```

```
4], ry[j + 3], ry[j + 2], ry[j + 1],ry[j], ry[abs(j -1)],ry[abs(j -
2)],ry[abs(j -3)],ry[abs(j-4)],ry[abs(j-5)],ry[abs(j-6)],ry[j + 7])
                numerator_array14[7] = (ry[j + 7], ry[j + 6], ry[j +
5], ry[j + 4], ry[j + 3], ry[j + 2],ry[j+1], ry[j],ry[abs(j -
1)],ry[abs(j -2)],ry[abs(j-3)],ry[abs(j-4)],ry[abs(j-5)],ry[j + 8])
                numerator_array14[8] = (ry[j + 8], ry[j + 7], ry[j +
6], ry[j + 5], ry[j + 4], ry[j + 3],ry[j+2],
ry[j+1],ry[abs(j)],ry[abs(j -1)],ry[abs(j-2)],ry[abs(j-3)],ry[abs(j-
4)],ry[j + 9])
                numerator_array14[9] = (ry[j + 9], ry[j + 8], ry[j +
7], ry[j + 6], ry[j + 5], ry[j + 4], ry[j + 3], ry[j + 2],
ry[j+1],ry[j],ry[abs(j-1)],ry[abs(j-2)],ry[abs(j-3)],ry[j + 10])
                numerator_array14[10] = (ry[j + 10], ry[j + 9], ry[j +
8], ry[j + 7], ry[j + 6], ry[j + 5], ry[j + 4], ry[j + 3],
ry[j+2],ry[j+1],ry[abs(j)],ry[abs(j-1)],ry[abs(j-2)],ry[j + 11])
                numerator_array14[11] = (ry[j + 11], ry[j + 10], ry[j +
9], ry[j + 8], ry[j + 7], ry[j + 6], ry[j + 5], ry[j + 4],
ry[j+3],ry[j+2],ry[abs(j+1)],ry[abs(j)],ry[abs(j-1)],ry[j + 12])
                numerator_array14[12] = (ry[j + 12], ry[j + 11], ry[j +
10], ry[j + 9], ry[j + 8], ry[j + 7], ry[j + 6], ry[j + 5], ry[j
+4],ry[j + 3], ry[abs(j + 2)], ry[abs(j+1)],ry[abs(j)],ry[j + 13])
                numerator_array14[13] = (ry[j + 13], ry[j + 12], ry[j +
11], ry[j + 10], ry[j + 9], ry[j + 8], ry[j + 7], ry[j + 6], ry[j
+5],ry[j + 4], ry[abs(j + 3)], ry[abs(j+2)],ry[abs(j+1)],ry[j + 14])

                for i in range(0,k):
                    denominator_array14[i][0:-1] =
numerator_array14[i][0:-1]
                    denominator_array14[i][-1] = ry[abs(j - k + i + 1)]

                phi_k14.append(np.linalg.det(numerator_array14) /
np.linalg.det(denominator_array14))

            elif k ==15:
                numerator_array15 = np.zeros((k, k))
                denominator_array15 = np.zeros((k, k))
                numerator_array15[0] = (ry[j], ry[abs(j - 1)], ry[abs(j
- 2)], ry[abs(j - 3)], ry[abs(j - 4)], ry[abs(j - 5)], ry[abs(j-6)],
ry[abs(j -7)],ry[abs(j -8)],ry[abs(j -9)],ry[abs(j-10)],ry[abs(j-
11)],ry[abs(j-12)],ry[abs(j-13)],ry[j + 1])
                numerator_array15[1] = (ry[j + 1], ry[j], ry[abs(j -
1)], ry[abs(j - 2)], ry[abs(j - 3)], ry[abs(j - 4)],ry[abs(j-5)],
ry[abs(j -6)],ry[abs(j -7)],ry[abs(j -8)],ry[abs(j-9)],ry[abs(j-
10)],ry[abs(j-11)],ry[abs(j-12)],ry[j + 2])
                numerator_array15[2] = (ry[j + 2], ry[j + 1], ry[j],
ry[abs(j - 1)], ry[abs(j - 2)], ry[abs(j - 3)],ry[abs(j-4)], ry[abs(j -
5)],ry[abs(j -6)],ry[abs(j -7)],ry[abs(j-8)],ry[abs(j-9)],ry[abs(j-
10)],ry[abs(j-11)],ry[j + 3])
                numerator_array15[3] = (ry[j + 3], ry[j + 2], ry[j +
1], ry[j], ry[abs(j - 1)], ry[abs(j - 2)],ry[abs(j-3)], ry[abs(j -
4)],ry[abs(j -5)],ry[abs(j -6)],ry[abs(j-7)],ry[abs(j-8)],ry[abs(j-
9)],ry[abs(j-10)],ry[j + 4])
                numerator_array15[4] = (ry[j + 4], ry[j + 3], ry[j +
2], ry[j + 1], ry[j], ry[abs(j - 1)],ry[abs(j-2)], ry[abs(j -
3)],ry[abs(j -4)],ry[abs(j -5)],ry[abs(j-6)],ry[abs(j-7)],ry[abs(j-
8)],ry[abs(j-9)],ry[j + 5])
                numerator_array15[5] = (ry[j + 5], ry[j + 4], ry[j +
```

```
3], ry[j + 2], ry[j + 1], ry[j],ry[abs(j-1)], ry[abs(j -2)],ry[abs(j -3)],ry[abs(j -4)],ry[abs(j-5)],ry[abs(j-6)],ry[abs(j-7)],ry[abs(j-8)],ry[j + 6])
                numerator_array15[6] = (ry[j + 6], ry[j + 5], ry[j + 4], ry[j + 3], ry[j + 2], ry[j + 1],ry[j], ry[abs(j -1)],ry[abs(j -2)],ry[abs(j -3)],ry[abs(j-4)],ry[abs(j-5)],ry[abs(j-6)],ry[abs(j-7)],ry[j + 7])
                numerator_array15[7] = (ry[j + 7], ry[j + 6], ry[j + 5], ry[j + 4], ry[j + 3], ry[j + 2],ry[j+1], ry[j],ry[abs(j -1)],ry[abs(j -2)],ry[abs(j-3)],ry[abs(j-4)],ry[abs(j-5)],ry[abs(j-6)],ry[j + 8])
                numerator_array15[8] = (ry[j + 8], ry[j + 7], ry[j + 6], ry[j + 5], ry[j + 4], ry[j + 3],ry[j+2], ry[j+1],ry[abs(j)],ry[abs(j -1)],ry[abs(j-2)],ry[abs(j-3)],ry[abs(j-4)],ry[abs(j-5)],ry[j + 9])
                numerator_array15[9] = (ry[j + 9], ry[j + 8], ry[j + 7], ry[j + 6], ry[j + 5], ry[j + 4], ry[j + 3], ry[j + 2], ry[j+1],ry[j],ry[abs(j-1)],ry[abs(j-2)],ry[abs(j-3)],ry[abs(j-4)],ry[j + 10])
                numerator_array15[10] = (ry[j + 10], ry[j + 9], ry[j + 8], ry[j + 7], ry[j + 6], ry[j + 5], ry[j + 4], ry[j + 3], ry[j+2],ry[j+1],ry[abs(j)],ry[abs(j-1)],ry[abs(j-2)],ry[abs(j-3)],ry[j + 11])
                numerator_array15[11] = (ry[j + 11], ry[j + 10], ry[j + 9], ry[j + 8], ry[j + 7], ry[j + 6], ry[j + 5], ry[j + 4], ry[j+3],ry[j+2],ry[abs(j+1)],ry[abs(j)],ry[abs(j-1)],ry[abs(j-2)],ry[j + 12])
                numerator_array15[12] = (ry[j + 12], ry[j + 11], ry[j + 10], ry[j + 9], ry[j + 8], ry[j + 7], ry[j + 6], ry[j + 5], ry[j +4],ry[j + 3], ry[abs(j + 2)], ry[abs(j+1)],ry[abs(j)],ry[abs(j-1)],ry[j + 13])
                numerator_array15[13] = (ry[j + 13], ry[j + 12], ry[j + 11], ry[j + 10], ry[j + 9], ry[j + 8], ry[j + 7], ry[j + 6], ry[j +5],ry[j + 4], ry[abs(j + 3)], ry[abs(j+2)],ry[abs(j+1)],ry[abs(j)],ry[j + 14])
                numerator_array15[14] = (ry[j + 14], ry[j + 13], ry[j + 12], ry[j + 11], ry[j + 10], ry[j + 9], ry[j + 8], ry[j + 7], ry[j +6],ry[j + 5], ry[abs(j + 4)], ry[abs(j+3)],ry[abs(j+2)],ry[abs(j+1)],ry[j + 15])

                for i in range(0,k):
                    denominator_array15[i][0:-1] = numerator_array15[i][0:-1]
                    denominator_array15[i][-1] = ry[abs(j - k + i + 1)]

                phi_k15.append(np.linalg.det(numerator_array15) / np.linalg.det(denominator_array15))


    phi = []

    phi.append(phi_k1)
    phi.append(phi_k2)
    phi.append(phi_k3)
    phi.append(phi_k4)
    phi.append(phi_k5)
    phi.append(phi_k6)
```

```
    phi.append(phi_k7)
    phi.append(phi_k8)
    phi.append(phi_k9)
    phi.append(phi_k10)
    phi.append(phi_k11)
    phi.append(phi_k12)
    phi.append(phi_k13)
    phi.append(phi_k14)
    phi.append(phi_k15)

    phi = np.array(phi[:k1]).T.tolist()
    header_row = np.arange(1,k1+1,1)
    phi.insert(0,header_row)
    gpac_table = tabulate(phi, headers='firstrow', showindex=True)
    print(gpac_table)
    return gpac_table

gpac = cal_gpac(15,15,new_data)

#1st set: k = 8, j = 0
#2nd set: k = 12, j = 0

#Plotting ACF and PACF using statsmodel
acf = sm.tsa.stattools.acf(new_data, nlags=20)
pacf = sm.tsa.stattools.pacf(new_data, nlags=20)

fig = plt.figure(figsize=(6, 7))
fig.tight_layout(pad=4)
plt.subplot(2, 1, 1)
plot_acf(new_data, ax=plt.gca(), lags=20, title="ACF of Stationary
Data")
plt.subplot(2, 1, 2)
plot_pacf(new_data, ax=plt.gca(), lags=20, title="PACF of Stationary
Data")
plt.show()

#Levenberg Marquadt algorithm=================

#ARMA Parameter Estimation


def LM(y,na,nb,lags):
    model = sm.tsa.ARMA(y,(na,nb)).fit(trend='nc',disp=0)
    for i in range(na):
        print("AR Coefficient a{}".format(i), "is:", model.params[i])
    for i in range(nb):
        print("MA Coefficient b{}".format(i), "is:",
model.params[i+na])
    print(model.summary())

    #Prediction
    y_train_set = y[:2415]
    y_test_set = y[2415:]
    model_hat = model.predict(start=0,end=len(y_train_set)-1)
    model_forecast = model.predict(start=2415, end = len(y)-1)

    #Residuals Testing and Chi-square test
```

```
    e = y_train_set - model_hat
    forecast_error = y_test_set - model_forecast
    re = calc_autocorrelation_coef(e,lags,f"ACF of Residuals for
ARMA({na},{nb})")
    Q = len(e)*np.sum(np.square(re[lags:]))
    fore_re =
calc_autocorrelation_coef(forecast_error.to_list(),lags,f"ARMA({na},{nb
}) Forecast Error ACF")
    fore_Q = len(forecast_error)*np.sum(np.square(fore_re[lags:]))

    DOF = lags - na - nb
    alfa = 0.01
    chi_critical = chi2.ppf(1-alfa,DOF)

    #Printing Prediction Error Statistical Measures
    print(f"ARMA({na},{nb}) Prediction Error
MSE:",calc_mse(calc_error_squared(e)))
    print(f"ARMA({na},{nb}) Prediction Error Q:", Q)
    print(f"ARMA({na},{nb}) Prediction Error Variance: ", np.var(e))
    print(f"ARMA({na},{nb}) Prediction Error Mean:",np.mean(e))

    #Printing Forecast Error Statistical Measures
    arma_forecast_error_mse =
calc_mse(calc_error_squared(forecast_error))
    print(f"ARMA({na},{nb}) Forecast Error
MSE:",arma_forecast_error_mse)
    print(f"ARMA({na},{nb}) Forecast Error Q:", fore_Q)
    arma_forecast_error_var = np.var(forecast_error)
    print(f"ARMA({na},{nb}) Forecast Error Variance: ",
arma_forecast_error_var)
    arma_forecast_error_mean = np.mean(forecast_error)
    print(f"ARMA({na},{nb}) Forecast Error
Mean:",arma_forecast_error_mean)

    if Q < chi_critical:
        print("Residual is white")
    else:
        print("The Residual is NOT white")

    if fore_Q < chi_critical:
        print("Forecast Error is white")
    else:
        print("Forecast Error is NOT white")

    lbvalue, pvalue = sm.stats.acorr_ljungbox(e, lags = [lags])
    print("lbvalue",lbvalue)
    print("pvalue",pvalue)

    #Plotting
    plt.figure()
    plt.plot(y,'r',label='True Data')
    plt.plot(model_hat,'b',label = "Prediction")
    plt.plot(model_forecast,'g',label='Forecast')
    plt.xlabel("Samples")
    plt.ylabel("Magnitude")
    plt.legend()
    plt.title(f"ARMA({na},{nb}) Prediction and Forecast vs. Raw Data")
```

```
    plt.show()

    #Plotting Hstep
    plt.figure()
    plt.plot(y_test_set,'r',label='True Data')
    #plt.plot(model_hat,'b',label = "Prediction")
    plt.plot(model_forecast,'g',label='Forecast')
    plt.xlabel("Samples")
    plt.ylabel("Magnitude")
    plt.legend()
    plt.title(f"ARMA({na},{nb}) Prediction and Forecast vs. Raw Data")
    plt.show()

    return
arma_forecast_error_mse,fore_Q,arma_forecast_error_var,arma_forecast_er
ror_mean

arma8_forecast_error_mse,arma8_forecast_error_Q,arma8_forecast_error_va
r,arma8_forecast_error_mean = LM(stock,8,0,20)
arma12_forecast_error_mse,arma12_forecast_error_Q,arma12_forecast_error
_var,arma12_forecast_error_mean = LM(stock,12,0,20)

#Diagnostic Analysis======================================




#Base-models=============================================

y_train_list = y_train.to_list()
y_test_list = y_test.to_list() #604

#Predict Average Forecast
def calc_average_forecast(training_data,testing_data):
    forecast =[]
    for i in range(1,len(testing_data)):
        forecast.append(sum(training_data)/(len(training_data)-1))
    return forecast

average_forecast = calc_average_forecast(y_train_list,y_test_list) #603
average_forecast_error = calc_error(y_test,average_forecast)
average_forecast_error_squared
=calc_error_squared(average_forecast_error)
average_forecast_error_mse =
calc_mse(average_forecast_error_squared)#261615.53316583813
print("Average Forecast Error MSE: ",average_forecast_error_mse)
average_forecast_error_acf =
calc_autocorrelation_coef(average_forecast_error,20,'Average Forecast
Error ACF')
average_forecast_error_Q =
calc_q(average_forecast_error_acf,len(average_forecast_error))
print("Average Forecast Error Q: ",average_forecast_error_Q)
average_forecast_error_variance = np.var(average_forecast_error)
print("Average Forecast Error Variance:
",average_forecast_error_variance)
average_forecast_error_mean = np.mean(average_forecast_error)
print("Average Forecast Error Mean: ",average_forecast_error_mean)
```

```
def calc_naive_forecast(training_data,test_data):
    forecast = []
    for i in range(0,len(test_data)):
        forecast.append(training_data[-1])
    return forecast

naive_forecast = calc_naive_forecast(y_train_list,y_test_list) #604
naive_forecast_error = calc_error(y_test,naive_forecast)
naive_forecast_error_squared =calc_error_squared(naive_forecast_error)
naive_forecast_error_mse =
calc_mse(naive_forecast_error_squared)#39334.89404387418
print("Naive Forecast Error MSE: ",naive_forecast_error_mse)
naive_forecast_error_acf =
calc_autocorrelation_coef(naive_forecast_error,20,'Naive Forecast Error
ACF')
naive_forecast_error_Q =
calc_q(naive_forecast_error_acf,len(naive_forecast_error))
print("Naive Forecast Error Q: ",naive_forecast_error_Q)
naive_forecast_error_variance = np.var(naive_forecast_error)
print("Naive Forecast Error Variance: ",naive_forecast_error_variance)
naive_forecast_error_mean = np.mean(naive_forecast_error)
print("Naive Forecast Error Mean: ",naive_forecast_error_mean)


#Calculate Drift Method Forecast
def calc_drift_forecast(training_set,test_set):
    forecast = []
    for i in range(0,len(test_set)):
        forecast.append(((training_set[-1] -
training_set[0])/(len(training_set)-1))*((i+10)-1) + training_set[0])
    return forecast

drift_forecast = calc_drift_forecast(y_train_list,y_test_list) #604
drift_forecast_error = calc_error(y_test,drift_forecast)
drift_forecast_error_squared =calc_error_squared(drift_forecast_error)
drift_forecast_error_mse =
calc_mse(drift_forecast_error_squared)#39334.89404387418
print("Drift Forecast Error MSE: ",drift_forecast_error_mse)
drift_forecast_error_acf =
calc_autocorrelation_coef(drift_forecast_error,20,'Drift Forecast Error
ACF')
drift_forecast_error_Q =
calc_q(drift_forecast_error_acf,len(drift_forecast_error))
print("Drift Forecast Error Q: ",drift_forecast_error_Q)
drift_forecast_error_variance = np.var(drift_forecast_error)
print("Drift Forecast Error Variance: ",drift_forecast_error_variance)
drift_forecast_error_mean = np.mean(drift_forecast_error)
print("Drift Forecast Error Mean: ",drift_forecast_error_mean)


#Calculating SES
def calc_ses_method_prediction(training_set,alpha):
    prediction = []
    for i in range(0,len(training_set)):
        if i == 0:
            prediction.append((alpha*training_set[i]) + (1-
```

```
alpha)*training_set[i])
        elif i>0:
            prediction.append((alpha * training_set[i]) + (1 - alpha) *
prediction[i-1])
    return prediction
alpha = 0.5

ses_forecast = [calc_ses_method_prediction(y_train_list,alpha)[-
1]]*len(y_test_list) #604
ses_forecast_error = calc_error(y_test,ses_forecast)
ses_forecast_error_squared =calc_error_squared(ses_forecast_error)
ses_forecast_error_mse =
calc_mse(ses_forecast_error_squared)#39334.89404387418
print("SES Forecast Error MSE: ",ses_forecast_error_mse)
ses_forecast_error_acf =
calc_autocorrelation_coef(ses_forecast_error,20,f'SES Forecast Error
ACF (alpha={alpha})')
ses_forecast_error_Q =
calc_q(ses_forecast_error_acf,len(ses_forecast_error))
print("SES Forecast Error Q: ",ses_forecast_error_Q)
ses_forecast_error_variance = np.var(ses_forecast_error)
print("SES Forecast Error Variance: ",ses_forecast_error_variance)
ses_forecast_error_mean = np.mean(ses_forecast_error)
print("SES Forecast Error Mean: ",ses_forecast_error_mean)


#Calculating Holt's Linear Method
def calc_holt_linear_forecast(y_train,y_test):
    aapl_holtt = ets.ExponentialSmoothing(y_train.values,
trend='additive',damped=False,seasonal=None).fit()
    aapl_holtf = aapl_holtt.forecast(steps=len(y_test))
    aapl_holtf = pd.DataFrame(aapl_holtf).set_index(y_test.index)
    #aapl_holt_mse =
np.square(np.subtract(y_test.values,np.ndarray.flatten(aapl_holtf.value
s))).mean()
    aapl_forecast_error_holt = calc_error(y_test,aapl_holtf.values)
    forecast_error_squared =
calc_error_squared(aapl_forecast_error_holt)
    forecast_error_mse = calc_mse(forecast_error_squared)

    return aapl_holtf, forecast_error_mse, aapl_forecast_error_holt

holt_linear_forecast, holt_linear_forecast_error_mse,
holtlinear_forecast_error = calc_holt_linear_forecast(y_train,y_test)
#604
print("Holt-Linear Forecast Error MSE:
",holt_linear_forecast_error_mse)
holt_linear_forecast_error_acf =
calc_autocorrelation_coef(holtlinear_forecast_error,20,'Holt-Linear
Forecast Error ACF')
holt_linear_forecast_error_Q =
calc_q(holt_linear_forecast_error_acf,len(holtlinear_forecast_error))
print("Holt-Linear Forecast Error Q: ",holt_linear_forecast_error_Q)
holt_linear_forecast_error_variance = np.var(holtlinear_forecast_error)
print("Holt-Linear Forecast Error Variance:
",holt_linear_forecast_error_variance)
holt_linear_forecast_error_mean = np.mean(holtlinear_forecast_error)
```

```
print("Holt-Linear Forecast Error Mean:
",holt_linear_forecast_error_mean)


#Plotting Base Models
fig = plt.figure(figsize=(16,10))
#fig.tight_layout(pad = 4)
ax1 = fig.add_subplot(3,2,1)
ax1.plot(X_train,y_train, label = 'Training dataset')
ax1.plot(X_test,y_test, label = 'Testing dataset')
ax1.plot(X_test[:-1],average_forecast, label = 'Average Method h-step
forecast')
ax1.set_xticks(ax.get_xticks()[::1])
plt.xticks(rotation=45)
ax1.set_xlabel("Time (t)")
ax1.set_ylabel("Magnitude")
ax1.legend(loc = 'upper left')
ax1.set_title("Google Stock Close Price Average Forecasting")

ax2 = fig.add_subplot(3,2,2)
ax2.plot(X_train,y_train, label = 'Training dataset')
ax2.plot(X_test,y_test, label = 'Testing dataset')
ax2.plot(X_test,naive_forecast, label = 'Naive Method h-step forecast')
ax2.set_xticks(ax.get_xticks()[::1])
plt.xticks(rotation=45)
ax2.set_xlabel("Time (t)")
ax2.set_ylabel("Magnitude")
ax2.legend(loc = 'upper left')
ax2.set_title("Google Stock Close Price Naive Forecasting")

ax3 = fig.add_subplot(3,2,3)
ax3.plot(X_train,y_train, label = 'Training dataset')
ax3.plot(X_test,y_test, label = 'Testing dataset')
ax3.plot(X_test,drift_forecast, label = 'Drift Method h-step forecast')
ax3.set_xticks(ax.get_xticks()[::1])
plt.xticks(rotation=45)
ax3.set_xlabel("Time (t)")
ax3.set_ylabel("Magnitude")
ax3.legend(loc = 'upper left')
ax3.set_title("Google Stock Close Price Drift Forecasting")

ax4 = fig.add_subplot(3,2,4)
ax4.plot(X_train,y_train, label = 'Training dataset')
ax4.plot(X_test,y_test, label = 'Testing dataset')
ax4.plot(X_test,ses_forecast, label = 'SES Method h-step forecast')
ax4.set_xticks(ax.get_xticks()[::1])
plt.xticks(rotation=45)
ax4.set_xlabel("Time (t)")
ax4.set_ylabel("Magnitude")
ax4.legend(loc = 'upper left')
ax4.set_title(f"Google Stock Close Price SES Forecasting
(alpha={alpha})")

ax5 = fig.add_subplot(3,2,5)
ax5.plot(X_train,y_train, label = 'Training dataset')
ax5.plot(X_test,y_test, label = 'Testing dataset')
ax5.plot(X_test,holt_linear_forecast, label = 'Holt-Linear Method h-
```

```
step forecast')
ax5.set_xticks(ax.get_xticks()[::1])
plt.xticks(rotation=45)
ax5.set_xlabel("Time (t)")
ax5.set_ylabel("Magnitude")
ax5.legend(loc = 'upper left')
ax5.set_title("Google Stock Close Price Holt-Linear Forecasting")

ax6 = fig.add_subplot(3,2,6)
#fig,ax = plt.subplots(figsize = (10,8))
ax6.plot(X_train,y_train, label = 'Training dataset')
ax6.plot(X_test,y_test, label = 'Testing dataset')
ax6.plot(X_test,stock_hw_forecast, label = 'Holt-Winter Method h-step
forecast')
ax6.set_xticks(ax.get_xticks()[::1])
plt.xticks(rotation=45)
ax6.set_xlabel("Time (t)")
ax6.set_ylabel("Magnitude")
ax6.legend(loc = 'upper left')
ax6.set_title("Google Stock Close Price Holt-Winter Forecasting")

fig.tight_layout(pad = 4)
plt.show()

#==========================================================

#Creating DataFrame of Each Model Statistical Measures

model_stats = pd.DataFrame({'Forecast Error Mean':
[average_forecast_error_mean,naive_forecast_error_mean,

drift_forecast_error_mean,ses_forecast_error_mean,

holt_linear_forecast_error_mean,hw_forecast_error_mean,

mlr_forecast_error_mean,arma8_forecast_error_mean,

arma12_forecast_error_mean],
                            'Forecast Error
Variance':[average_forecast_error_variance,naive_forecast_error_varianc
e,

drift_forecast_error_variance,ses_forecast_error_variance,

holt_linear_forecast_error_variance,hw_forecast_error_variance,

mlr_forecast_error_variance,arma8_forecast_error_var,

arma12_forecast_error_var],
                            'Forecast Error
MSE':[average_forecast_error_mse,naive_forecast_error_mse,

drift_forecast_error_mse,ses_forecast_error_mse,

holt_linear_forecast_error_mse,stock_hw_forecast_error_mse,

mlr_forecast_error_mse,arma8_forecast_error_mse,
```

```
arma12_forecast_error_mse],
                            'Forecast Error
Q':[average_forecast_error_Q,naive_forecast_error_Q,

drift_forecast_error_Q,ses_forecast_error_Q,

holt_linear_forecast_error_Q,hw_forecast_error_Q,

stock_mlr_q,arma8_forecast_error_Q,arma12_forecast_error_Q]},
                           index = ['Average','Naive','Drift','Simple
Exponential Smoothing','Holt-Linear',
                                    'Holt-Winters','Multiple Linear
Regression','ARMA(8,0)','ARMA(12,0)'])

print(model_stats)
```