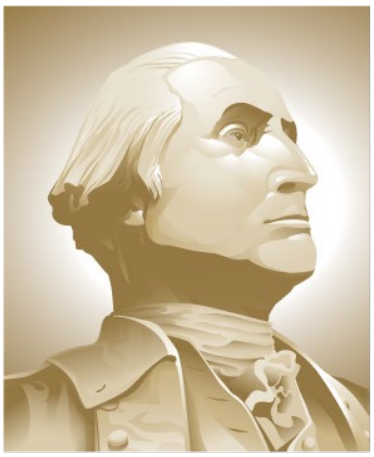


THE GEORGE WASHINGTON UNIVERSITY

WASHINGTON, DC

Time Series Analysis and Modeling

Final Term Project

Ujjwal Oli

Table of Contents

ABSTRACT **3**

INTRODUCTION **3**

DESCRIPTION OF DATASET 5
STATIONARITY CHECK 7
TIME SERIES DECOMPOSITION 11
HOLT-WINTER METHOD 13
FEATURE SELECTION 15
MULTIPLE LINEAR REGRESSION 16
BASE MODELS 24
b. Naïve method *25*
c. Drift method *26*
d. SES method *28*
ARMA AND ARIMA AND SARIMA MODEL ORDER DETERMINATION 29
FINAL MODEL SELECTION 38
FORECAST FUNCTION FOR THE BEST MODEL 39
H-STEP PREDICTION 39

CONCLUSION **40**

APPENDIX **40**

REFERENCES **65**

Abstract

This report is detail explanation of the real time series data analysis project. The project uses time series analysis technique to examine time series data. The main purpose of this paper is to implement all the techniques and time series analysis method to come up with the model that represents the data best. This paper experiments different linear model covered throughout the semester and finally selects a model. To accomplish the purpose, this report will be using python libraries and different python packages and comparing the results of different models. In addition with the python packages, the own code developed will also be used to accomplish the goal. The dataset that this paper will be using is from Yahoo's Amazon Stock Price from 2000 to 2021. Develop a linear model representation of the data using Python.

Introduction

Time series data is a set of quantitative information collected over time in a chronological manner. Time series data has huge significance as it can be applied to forecast or predict the future based on the current and previous pieces of information we have. Though forecasting and predicting sounds fascinating, it is not that simple. To end up with insightful thought, the simple procedures to start is with cleaning data and verifying data. This project initiates with import and implementation of data. Once the data is understood, and once we assure that we have fair amount of clean data we move to the further steps. Real time data entails many variability so it is a must to set that each data points are independent to each other. Thus before forecasting the future, each point in time-series model have to be independent of one another. In regard to it, one way of validating if each data point is independent is via indicating if the dataset is stationary.

To check the stationarity of the data, first we can plot the data and observe it. After that in this paper, we find the mean and variances of each data points and plot them to see its pattern. And we perform statistical test also called as Augmented Dickey-Fuller test (ADF test). It is one of the most commonly used statistical test when it comes to analyzing the stationary of a series. This test is hypothesis testing where we either reject the hypothesis to support that the dataframe is stationary or do not reject the hypothesis to support that the dataframe is not stationary. Along with it, we also plot the ACF/PACF plot to validate the result of stationary.

Time series data can exhibit a variety of patterns and it is useful in forecasting to split a time series into several components. Time series pattern can have trend, can be seasonal and cyclic. In this report we will detrend the data and adjust the seasonality by python packages. Being able to detrend and adjust the seasonality of original data also helps us to calculate the strength of trend and seasonality. For strongly trended data, the seasonally adjusted data ($yt - St = Tt + Rt$) should have much more variation than the remainder component. For strongly seasonal data, the detrended data ($yt - Tt = St + Rt$) should have much more variation than the remainder component. The strength is ranged from 0 to 1. If the value is closer to 1, then we can tell it has

higher strength; similarly if the value is closer to 0, we can tell that the dataset has lower strength or is weak.

Above many models, Holt Winter Method is one technique to forecast the model and predict the behavior of a sequence of values over time—a time series. Holt-Winters is one of the most popular forecasting techniques for time series. It addresses both trend and seasonality part. It requires different parameters to perform calculation like seasonality periods and so on. In this report, we will be implementing this model to aid us to predict the future values.

Similarly, another technique is the regression model. In the regression model, we have a collection of observations (xi,t and yt). X represents the predictors or regressors or independent variable and Y represents the dependent variable or regressand or forecast variable. We want to predict the Y using the data we have so we need to generate the model that predicts Y precisely. While features X and Y are known, the coefficients $\beta 0, \beta 1, \ldots \beta k$ are unknowns which needs to be estimated. To measure the efficacy of our features in the model we use coefficient of determination (R-squared) and adjusted R-squared.

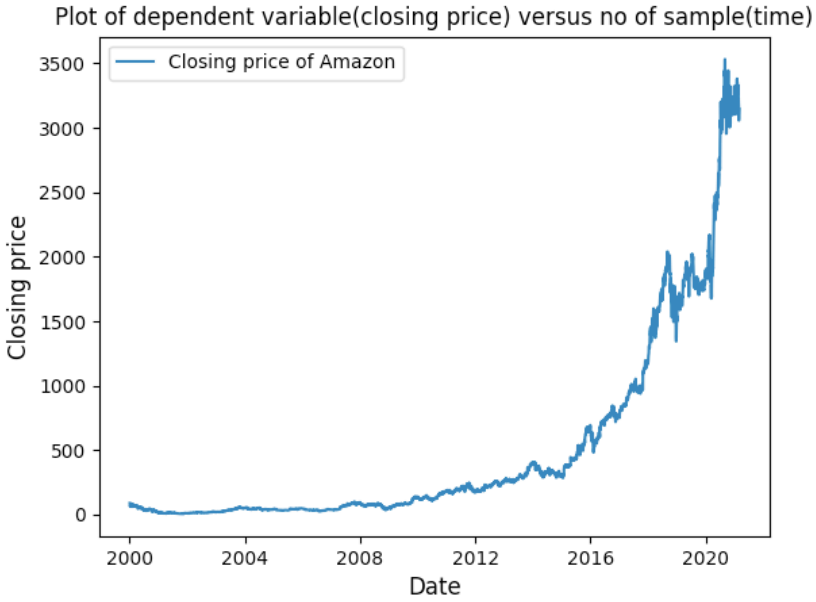
Likewise we also have other base models like average, naïve, drift and simple exponential smoothing method. Average method assumes that all observation are of equal importance and gives equal weights when generating forecasts. Generally, all the historic data are averaged to calculate the future values. In same way, naïve method forecasts the future values based on the last observation of historic data. It emphasizes on last observation. Generally, this method is useful for highly seasonal data. Similarly, drift method is the variation on the naïve to allow the forecast to increase or decrease over time, where the amount of change over time is set to be the average change seen in historical data. Finally, Simple Exponential Smoothing method is calculated using weighted averages where the weights decreases as observation come from further in past; meaning SES method weights are associated with older observations. These all methods have their own usability

Moreover, other way of finding the appropriate model is to implement the auto-regressive model, moving average method and combination of both. Autoregressive moving average (ARMA(na, nb)) models are the combination of AR(na) and MA(nb) models. ARMA models play a key role in the modeling of time series. ARMA models provide the most effective linear model of stationary time series, for real dataset we need to generate the model and to come up with the model, we need to find the order. One way of determining order for our model is by constructing GPAC table and examining it. This is exactly what we will be working in this project. A partial correlation is a conditional correlation between two variables, while excluding the effect of one or more independent variables. The generalized partial autocorrelation is used to estimate the order of ARMA model when na is not equal to 0 and nb is not equal to 0. Like ARMA, ARIMA is a forecasting for univariate time series data and it supports both an AR and MA elements. ARIMA includes the differencing part with AR and MA, but does not support seasonality. In this report, we will attempt to implement ARMA, ARIMA for different patterns and come up with the order with help of LM algorithm that represent the dataset most. The LM algorithm combines two minimization methods: the gradient decent method and the Gauss-Newton method. Thus, with help of estimated parameters we will develop ARMA and ARIMA model and compare the results of it.

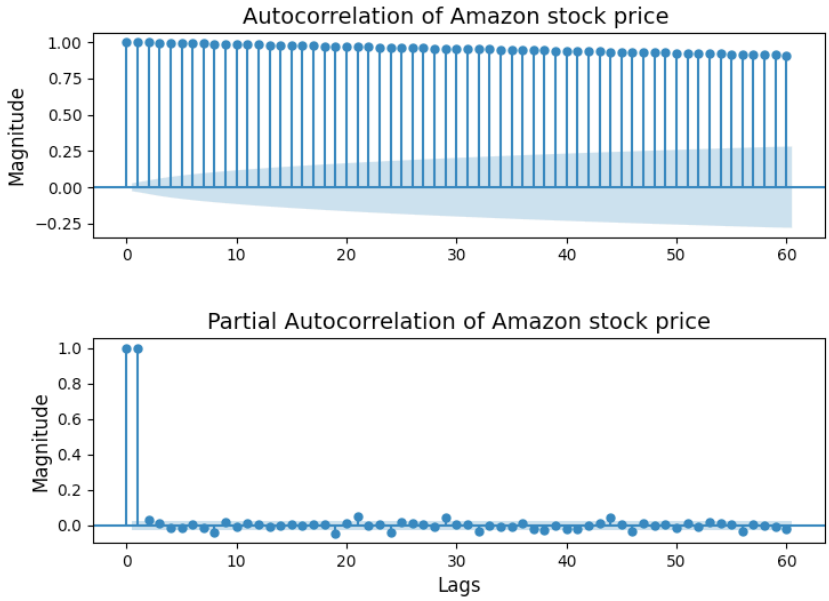
Thus, the major component for this report will be using different methods and compare the results of the methods to develop the best model.

Description of dataset

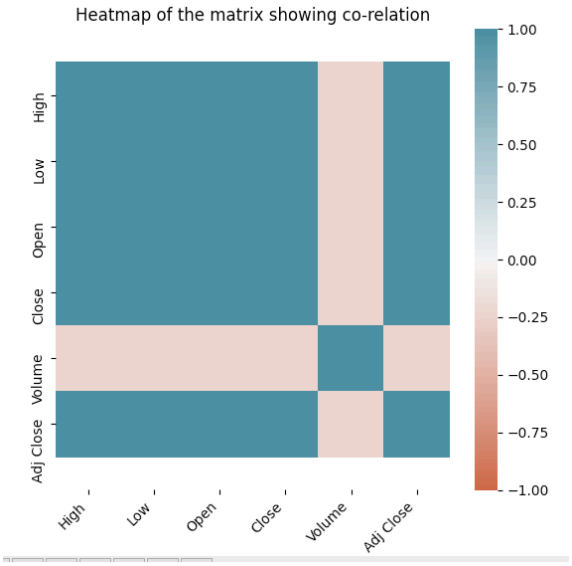
For this project, I did time series analysis for Amazon stock price. The data was taken from yahoo starting from '2000-01-01' to '2021-03-01'. In total there were 5323 rows and 6 columns. The features for the data were Open, High, Low, Adj Close, Volume and Close. Since we wanted to forecast the Amazon stock price the Close column was chosen to be as a predictor. Being stock price data, my dependent variable was Close column and my independent variables were Open, High , Low, Adj Close and Volume column. To start with the project, we wanted to see the nature of the plot of dependent variable and have some feeling about it. Thus, the plot of dependent variable versus the time, ACF/PACF of dependent variable, correlation matrix was plotted.



From the plot we can see that we have data from 2000 to 2021. We move from the left side of the plot to right side and see there is increase in stock price of Amazon. The Closing price seems to be more trended over the time. Now let's see the ACF/PACF plot of the Closing price (dependent variable)



The ACF/PACF plot gives us information about auto-correlation values and partial auto-correlation values for 60 lags. And from the ACF plot, we can see that there is nominal decay even until the 60 lags which means our dataset is not stationary. Now let's look the co-relation matrix



```
The pearson correlation coefficeint is :
                High       Low      Open     Close    Volume  Adj Close
High        1.000000  0.999864  0.999920  0.999906 -0.229940   0.999906
Low         0.999864  1.000000  0.999890  0.999913 -0.233535   0.999913
Open        0.999920  0.999890  1.000000  0.999810 -0.231582   0.999810
Close       0.999906  0.999913  0.999810  1.000000 -0.231757   1.000000
Volume     -0.229940 -0.233535 -0.231582 -0.231757  1.000000  -0.231757
Adj Close   0.999906  0.999913  0.999810  1.000000 -0.231757   1.000000
```

From the heatmap, we can see that we have almost all features co-related with other each other except the Volume column which is one of the independent variable. The heatmap co-relation matrix shows that it is ranged from -1 to 1.The darker color and lighter color shows high co-relation values. The darker greenish type of color shows that we have high positive co-relation and darker brownish type of color shows that we have high negative co-relation.
After analyzing the co-relation matrix, the data was analyzed for pre-processing, cleaning and double-checking for the missing values. Our data did not had any missing or nan values and no pre-processing had to be done. The check of nan values is shown in below figure.

```
High          0
Low           0
Open          0
Close         0
Volume        0
Adj Close     0
dtype: int64
```

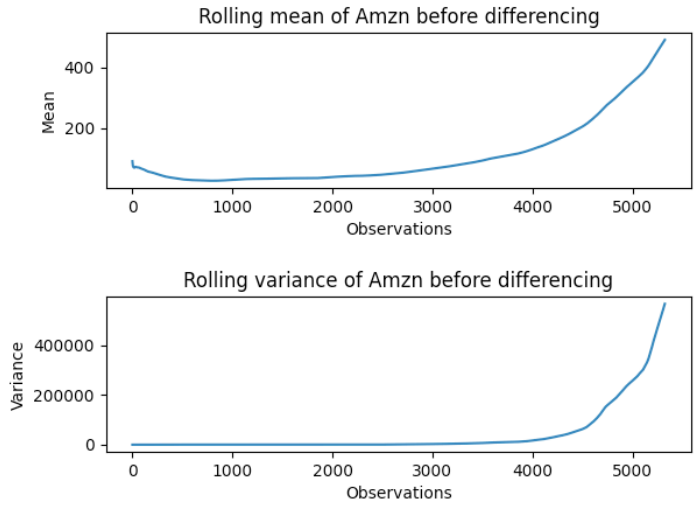
After that the dataset was splitted into training and testing set with regular rule of thumb which is 80-20 ratio. Of out total dataset size, 80% data was splitted for training and 20% data was splitted for testing size. Our training size had 4258 rows and remaining was kept for testing set.

```
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.2, shuffle=False)
X_train_copy = X_train.copy()
```

After the split of the data, we proceeded to stationarity check because we had seen our ACF plot which was not decaying.

Stationarity check

For the stationarity check, first of all the mean and variance of dependent variable was plotted. If the dataset is stationary the mean and variance would be constant over the time, but if the dataset is not stationary the mean and variance would have increase or decrease trend over the time.



So we can see that the rolling mean and the rolling variance is not stationary as they are not constant over the time. Let's confirm the plot result with ADF test.

For our ADF test,

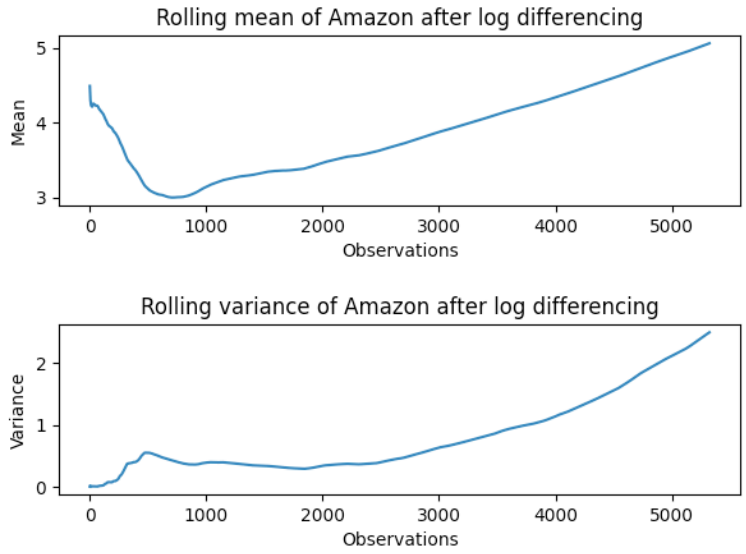
Null hypothesis: the time-series has a unit root, meaning it is non-stationary.
Alternative hypothesis: the time-series do not have a unit root, meaning it is stationary.

If the p-value from the test is below than 0.05(for our lab), we reject the null hypothesis claiming dataset to be stationary. However, if the p-value is above than 0.05, we fail to reject the null hypothesis, claiming the dataset to be non-stationary.

```
ADF Statistic: 3.282038
P-value: 1.000000
Critical values:
    1%: -3.432
    5%: -2.862
    10%: -2.567
```

As seen in ADF test, the p-value is 1.00 which is significantly greater than 0.05. Even the ADF statistic is positive so we can confirm that the dataset is not stationary. We have already seen the ACF/PACF plot where the ACF was not decaying so now lets make our dataset stationary. To make our dataset stationary we have the increasing variance so I used log transformation

technique followed by regular differencing. The following is the plot of rolling mean and rolling variance of log transformed dataset.

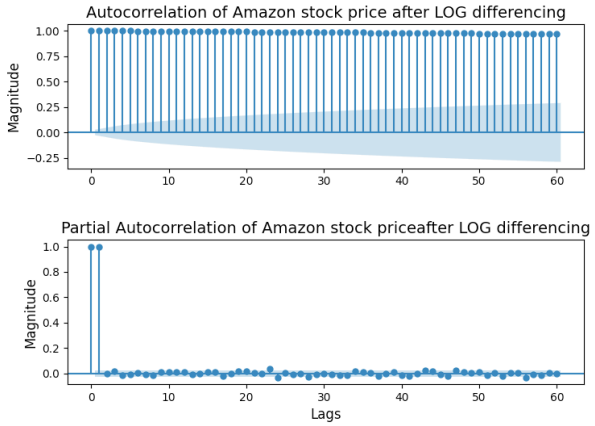


The plot of log differencing of mean and variance still is not constant over the time. Let's see the ADF test of log transformed dataset.

```
ADF Statistic: 0.523761
P-value: 0.985578
Critical values:
    1%: -3.432
    5%: -2.862
    10%: -2.567
```

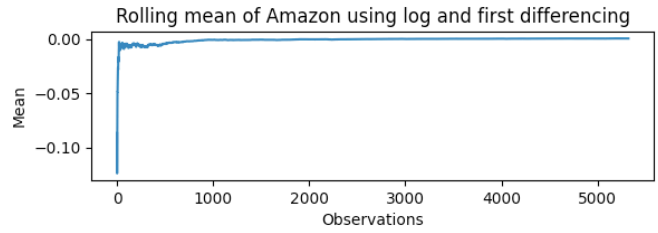
.

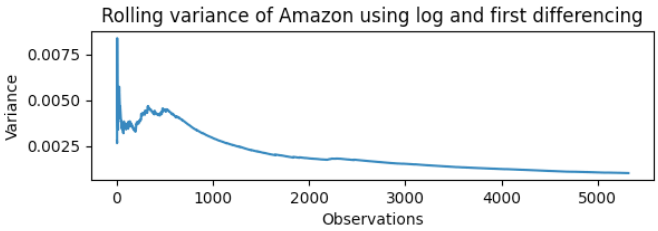
The p-value of the log differenced dataset still is greater than 0.05 which means we still do not have stationary dataset. We can even see the ACF/PACF plot of log differenced dataset.



There is not much of a change in lags for log differenced dataset with comparison of original dataset.

Then the log differencing was followed by regular differencing so let's see the result by plotting rolling mean-variance, ADF test and ACF/PACF plot.

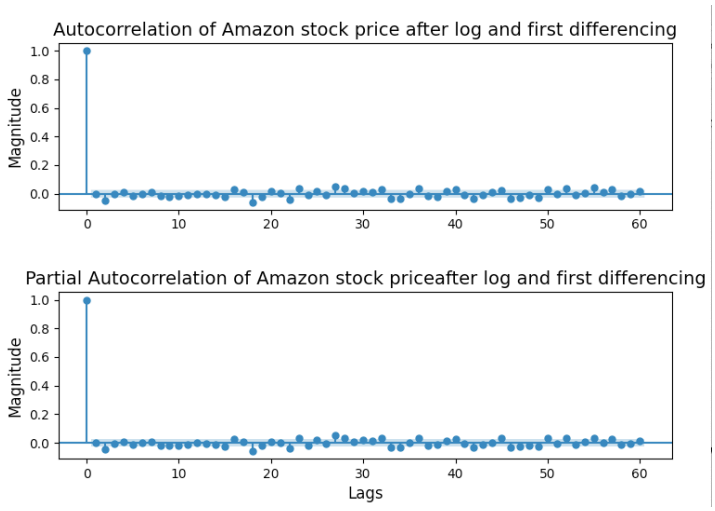




Now the rolling mean and variance looks constant over time. Lets see the ADF results and plot the ACF/PACF plot.

```
AFTER LOG differening used first differencing
ADF Statistic: -12.305611
P-value: 0.000000
Critical values:
    1%: -3.432
    5%: -2.862
    10%: -2.567
```

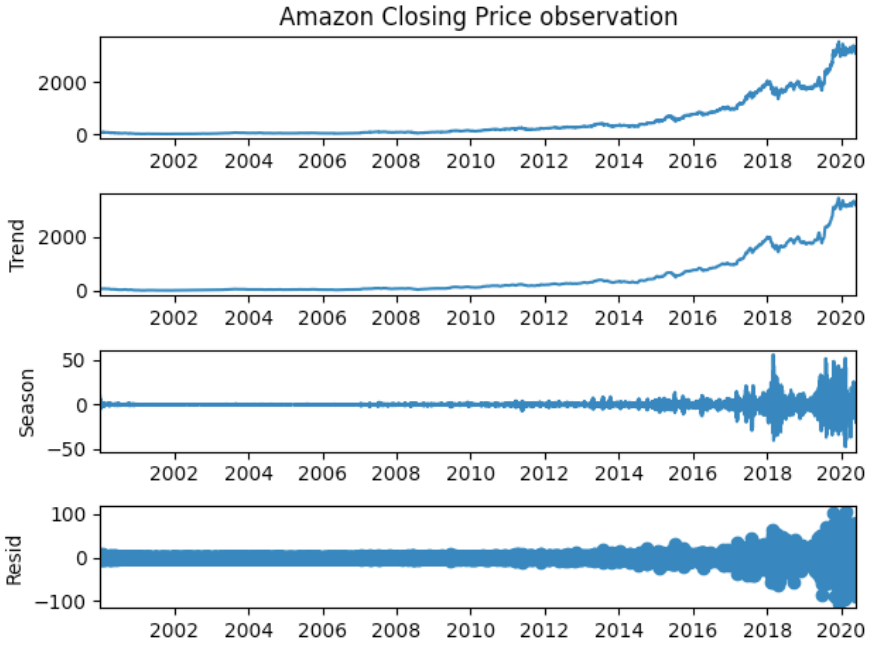
After log differencing and first differencing, the p-value came out to be zero which is lower than 0.05 supporting the result of rolling mean and variance.

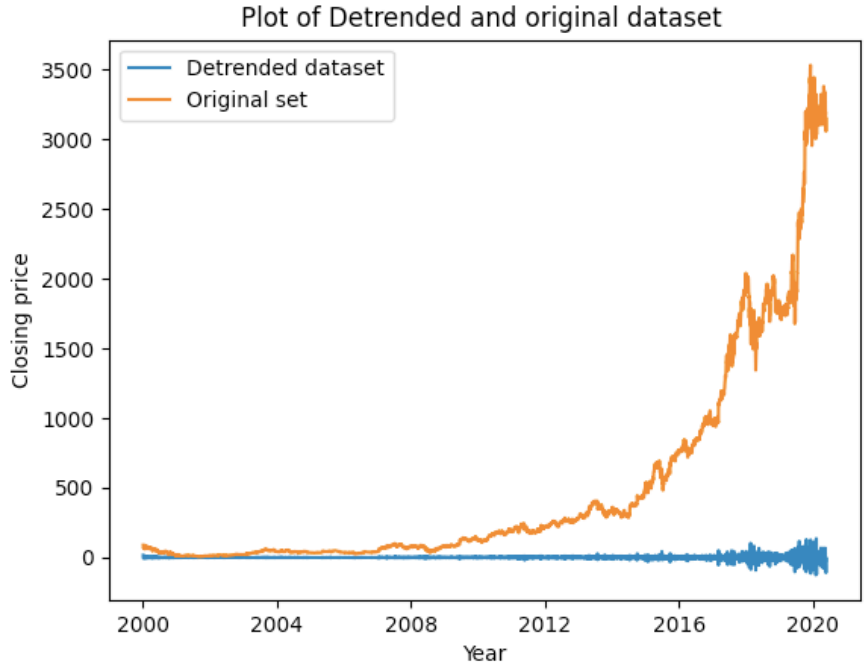


Then we confirmed the ADF test result and plot of mean and variance with ACF/PACF plot, In ACF/PACF plot we can only see one spike at lag zero and there are no any other spikes. This shows that our dataset is stationary.

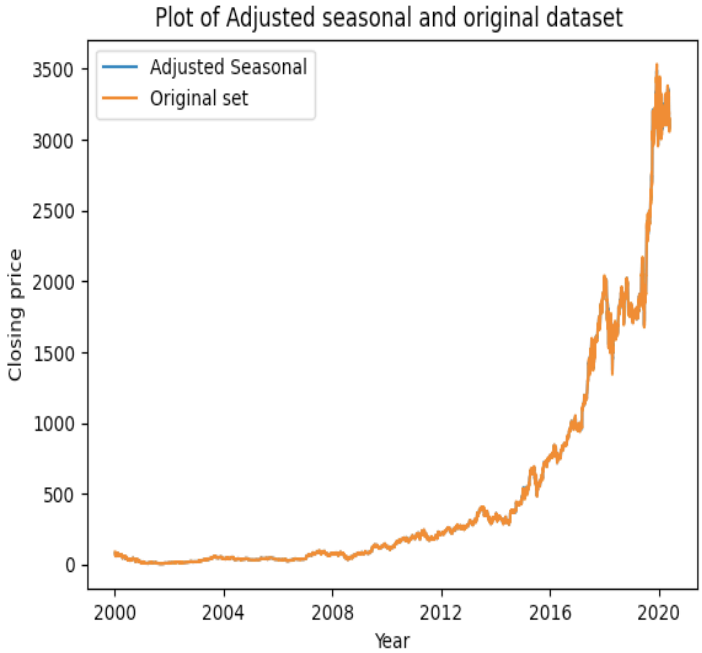
Time Series Decomposition

After stationarity analysis, lets approximate the trend and seasonality and plot the detrended and the seasonally adjusted data set. To do this, this paper implements the STL decomposition technique. The time series is composed of trend, seasonality and residuals so we can plot the each component for our dataset.

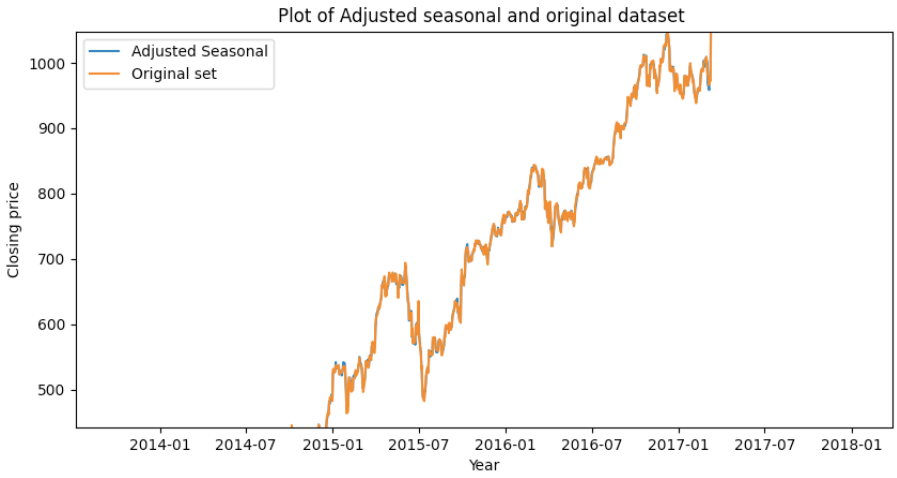




Using STL decomposition technique, we approximated the detrended dataset and plotted it. This plot makes sense as the detrended dataset is removing just a trend from the original dataset.



The adjusted seasonal and the original dataset is completely overlapped which we can see more clearly in below figure.



This figure is zoomed-in figure of (above) adjusted seasonal plot. The figure shows that the original dataset and adjusted seasonal plots are overlapped. The overlapped happened because adjusted seasonal calculation just takes the seasonality away not the trend. So as we can see there isn't high seasonality in the original dataset which is catching up the trend and making it overlap.

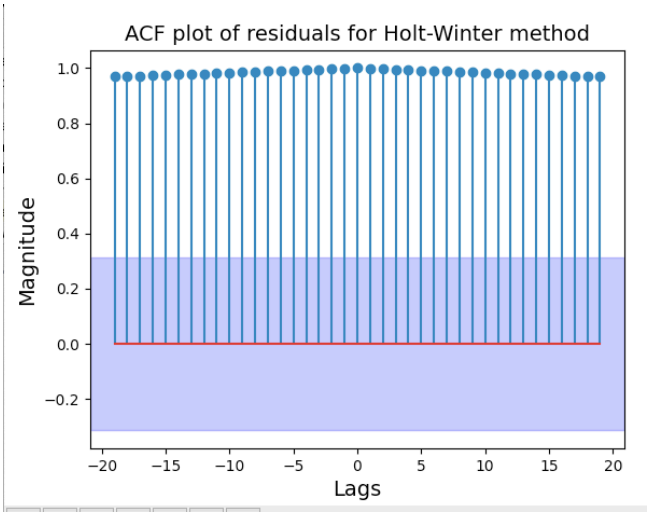
After the plot of detrended and seasonality, we can check the strength of trend and seasonality to determine if our dataset is highly trended or seasonal.

```
The strength of trend for this data set is  0.9998056380613058

The strength of seasonality for this data set is 0.3185882358133769
```

From the result we can see that our dataset is highly trended with value of 0.999 while strength of seasonality is 0.318 which is not that high. Thus we can also say that our data set is more trended than being seasonal.

Holt-Winter Method

After analysis of trend and seasonality, this paper goes over the Holt Winter method to find the best fit using the train dataset and make a prediction using the test set. Holt- Winter method, in general captures both trend and seasonality. For Holt-Winter method, python packages are used and then residuals are calculated. Let's plot the ACF of residuals of Holt-Winter method

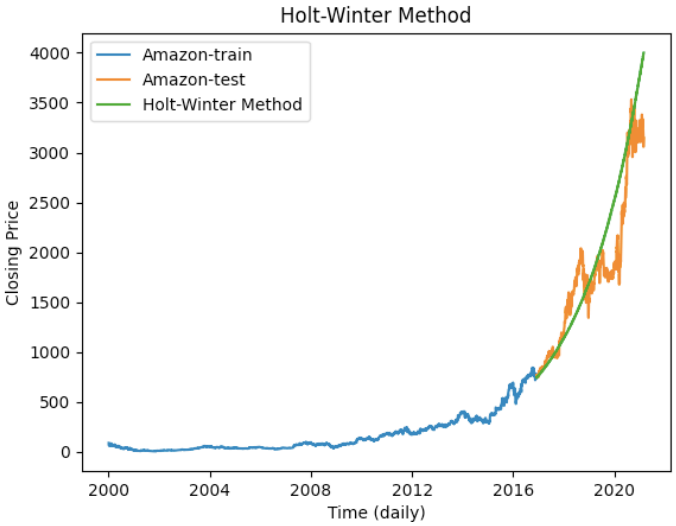


The ACF of residuals of Holt-Winter method is not white. Now let's look the other results of Holt-Winter method.

```
The Q value of residual using HWM is   73994.14812947212
The Mean of residual of HLWM is -92651.54026785398
The variance of residual of HLWM is 20417240939.642185
Mean square error for Holt Winter method is of testing set is  130608.4356451353
The Mean of forecast of HLWM is -129.80402311662888
The variance of forecast of HLWM is 113759.35122787298
```

The MSE of training set looks very large along with the Q value. However MSE of training set is lot lower than the MSE of training set, we have to compare that with the other methods.

Now let's look over the predictions and test- prediction results.



This figure explains the predictions calculated by Holt winter method. The blue line represents the training set, the yellow line represents the training set and the green line represents the test predictions calculated by Holt-Winter method. We can see that Holt-Winter method prediction for test set is catching up the trend to actual test-data.

Feature Selection

After Holt Winter Method, this paper implemented the multiple linear regression method, but to do that we need to check if all the features that we have are significantly important. To implement the multiple regression model, we implemented OLS package. To have the feature that we only need is basic principle on how we select the feature. So to perform feature selection, first of all we performed the collinearity test by calculating the condition number and singular values. Generally for singular values, we look if any values are closer to zero. If any numbers are closer to zero indicates that we have to remove the features. Similarly, condition number explains if the collinearity exists between the features. We remove the features which are co-related because predictions made on co-related features are unreliable.

From the condition number, if the condition number < 100, then it represents the weak degree of co-linearity(DOC); if the 100< condition number < 1000, then there is Moderate to Strong DOC and if the condition number > 1000 then it represents the severe DOC.

```
SingularValues = [3.34136640e+17 8.96300190e+08 2.92577213e+04 1.54107976e+04
 3.22413473e+03 9.56898577e+02]

The condition number including constant in original data= 18686548.581222363
The condition number without constant (original data) = 10180501.964410346
```

Based on the above results, we can see that there are some numbers closer to zero which indicates that they have to be removed. From this we can say that most probably we have to remove 3 to 4 features because the value of singular values are closer to 0 (in this case the values are not more than 5). Similarly, the condition number greater than 1000 represents there is severe degree of collinearity. Thus, based on the results we implemented OLS calculation and get the summary for regression.

Multiple Linear Regression

OLS Regression Results

Dep. Variable:	Close	R-squared:	1.000
Model:	OLS	Adj. R-squared:	1.000
Method:	Least Squares	F-statistic:	5.342e+26
Date:	Wed, 05 May 2021	Prob (F-statistic):	0.00
Time:	18:40:17	Log-Likelihood:	88386.
No. Observations:	4258	AIC:	-1.768e+05
Df Residuals:	4252	BIC:	-1.767e+05
Df Model:	5		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	1.593e-13	7.56e-12	0.021	0.983	-1.47e-11	1.5e-11
Open	5.579e-14	2.21e-12	0.025	0.980	-4.29e-12	4.4e-12
High	-1.235e-14	2.6e-12	-0.005	0.996	-5.11e-12	5.08e-12
Low	-2.787e-14	2.29e-12	-0.012	0.990	-4.52e-12	4.46e-12
Volume	2.64e-17	7.21e-19	36.616	0.000	2.5e-17	2.78e-17
Adj Close	1.0000	2.35e-12	4.26e+11	0.000	1.000	1.000

Omnibus:	4537.022	Durbin-Watson:	0.315
Prob(Omnibus):	0.000	Jarque-Bera (JB):	475152.877
Skew:	-5.203	Prob(JB):	0.00
Kurtosis:	53.694	Cond. No.	1.87e+07

From here we can see that we have 5 independent features with their co-efficient. Even though the adjusted R-squared is 1.00, they all have the higher p-values greater than 0.05. So now we perform the backward step-regression and remove the feature that has the highest p-value which is column “High”

Removing High

OLS Regression Results

Dep. Variable:	Close	R-squared:	1.000
Model:	OLS	Adj. R-squared:	1.000
Method:	Least Squares	F-statistic:	1.556e+28
Date:	Wed, 05 May 2021	Prob (F-statistic):	0.00
Time:	18:40:17	Log-Likelihood:	95088.
No. Observations:	4258	AIC:	-1.902e+05
Df Residuals:	4253	BIC:	-1.901e+05
Df Model:	4		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-6.273e-14	1.57e-12	-0.040	0.968	-3.13e-12	3.01e-12
Open	3.746e-14	2.73e-13	0.137	0.891	-4.97e-13	5.72e-13
Low	-3.039e-15	4.33e-13	-0.007	0.994	-8.52e-13	8.46e-13
Volume	-5.487e-18	1.47e-19	-37.324	0.000	-5.77e-18	-5.2e-18
Adj Close	1.0000	3.19e-13	3.13e+12	0.000	1.000	1.000

Omnibus:	4512.486	Durbin-Watson:	0.317
Prob(Omnibus):	0.000	Jarque-Bera (JB):	464246.460
Skew:	5.159	Prob(JB):	0.00
Kurtosis:	53.102	Cond. No.	1.87e+07

Even after removing high, we have features that have higher p-values so, now we remove feature “Constant”.

```
Removing const
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Close   R-squared (uncentered):                   1.000
Model:                            OLS   Adj. R-squared (uncentered):              1.000
Method:                 Least Squares   F-statistic:                          3.523e+27
Date:                Wed, 05 May 2021   Prob (F-statistic):                        0.00
Time:                        18:40:17   Log-Likelihood:                          90701.
No. Observations:                4258   AIC:                                 -1.814e+05
Df Residuals:                    4254   BIC:                                 -1.814e+05
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Open        6.911e-14   7.64e-13      0.090      0.928   -1.43e-12    1.57e-12
Low        -6.775e-14   1.21e-12     -0.056      0.955   -2.45e-12    2.31e-12
Volume     -1.535e-17   2.69e-19    -57.008      0.000   -1.59e-17   -1.48e-17
Adj Close      1.0000   8.94e-13   1.12e+12      0.000       1.000       1.000
==============================================================================
Omnibus:                     4533.780   Durbin-Watson:                   0.316
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           473753.709
Skew:                           5.197   Prob(JB):                         0.00
Kurtosis:                      53.619   Cond. No.                     6.33e+06
==============================================================================
```

Even after removing the feature Constant, we still have feature that have higher p-value, so we can remove feature “Low” too.

Removing low

OLS Regression Results

Dep. Variable:	Close	R-squared (uncentered):	1.000
Model:	OLS	Adj. R-squared (uncentered):	1.000
Method:	Least Squares	F-statistic:	9.588e+32
Date:	Wed, 05 May 2021	Prob (F-statistic):	0.00
Time:	18:40:17	Log-Likelihood:	1.1673e+05
No. Observations:	4258	AIC:	-2.335e+05
Df Residuals:	4255	BIC:	-2.334e+05
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Open	1.971e-15	1.26e-15	1.567	0.117	-4.95e-16	4.44e-15
Volume	2.104e-21	5.6e-22	3.755	0.000	1.01e-21	3.2e-21
Adj Close	1.0000	1.26e-15	7.95e+14	0.000	1.000	1.000

Omnibus:	1369.609	Durbin-Watson:	0.005
Prob(Omnibus):	0.000	Jarque-Bera (JB):	3544.433
Skew:	-1.765	Prob(JB):	0.00
Kurtosis:	5.740	Cond. No.	3.42e+06

After removing low column we still have Open column with p-value higher than 0.05. So remove “Open feature” too.

Removing Open

Removing Open

OLS Regression Results

Dep. Variable:	Close	R-squared (uncentered):	1.000
Model:	OLS	Adj. R-squared (uncentered):	1.000
Method:	Least Squares	F-statistic:	4.169e+34
Date:	Wed, 05 May 2021	Prob (F-statistic):	0.00
Time:	18:40:17	Log-Likelihood:	1.2390e+05
No. Observations:	4258	AIC:	-2.478e+05
Df Residuals:	4256	BIC:	-2.478e+05
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Volume	1.635e-21	1.04e-22	15.712	0.000	1.43e-21	1.84e-21
Adj Close	1.0000	3.73e-18	2.68e+17	0.000	1.000	1.000

Omnibus:	2160.937	Durbin-Watson:	0.048
Prob(Omnibus):	0.000	Jarque-Bera (JB):	14252.129
Skew:	-2.376	Prob(JB):	0.00
Kurtosis:	10.600	Cond. No.	3.86e+04

Now we have all the p values less than 0.05 and the adjusted R squared is still 1. Here the co-linearity still exists. So in that case if we need to remove the feature, we have to remove the feature that has greatest standard error. In this case, it would be adj close. So I tried to remove the “Adj Close” feature. The figure below shows the result after I remove the Adj close feature.

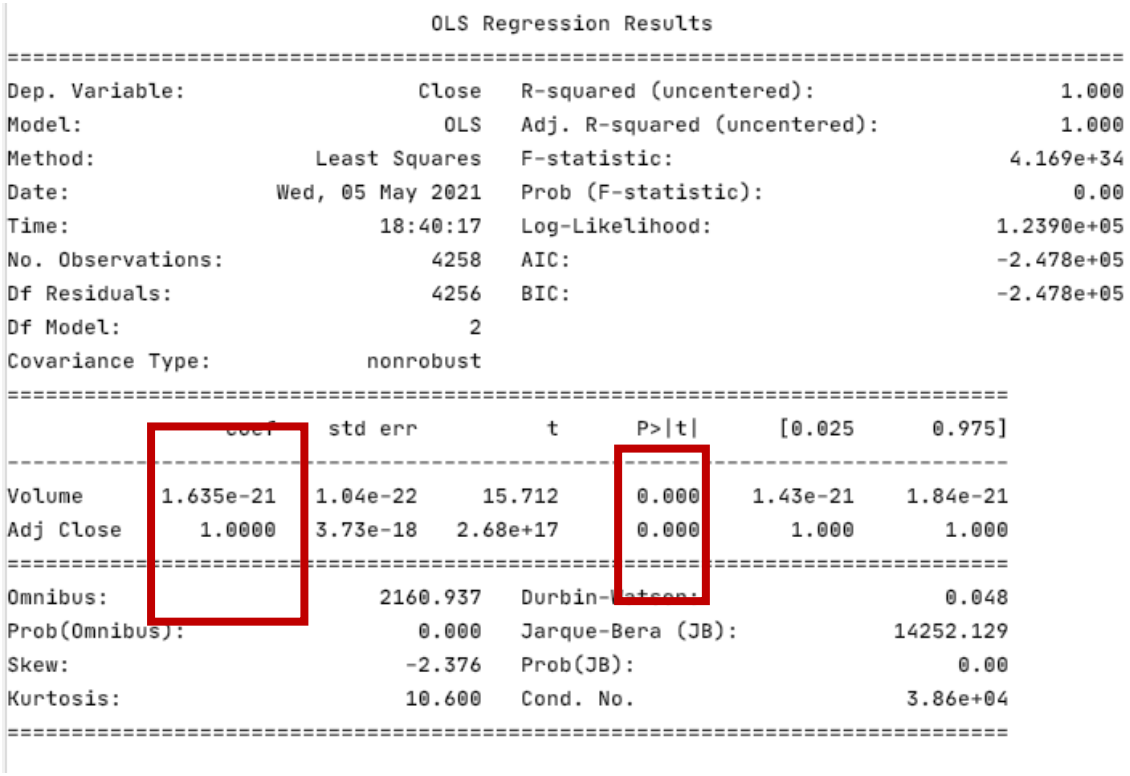
Removing Adj Close

OLS Regression Results

Dep. Variable:	Close	R-squared (uncentered):	0.138
Model:	OLS	Adj. R-squared (uncentered):	0.138
Method:	Least Squares	F-statistic:	680.2
Date:	Sat, 01 May 2021	Prob (F-statistic):	3.04e-139
Time:	14:13:26	Log-Likelihood:	-29187.
No. Observations:	4258	AIC:	5.838e+04
Df Residuals:	4257	BIC:	5.838e+04

From here we see that the adjusted R-squared is dropped quickly as soon as we drop “Adjusted-Close” feature so it is better to not to remove the feature. However, in this case we might still have co-linearity.

Thus our final model regression summary is:



OLS Regression Results

Dep. Variable:	Close	R-squared (uncentered):	1.000
Model:	OLS	Adj. R-squared (uncentered):	1.000
Method:	Least Squares	F-statistic:	4.169e+34
Date:	Wed, 05 May 2021	Prob (F-statistic):	0.00
Time:	18:40:17	Log-Likelihood:	1.2390e+05
No. Observations:	4258	AIC:	-2.478e+05
Df Residuals:	4256	BIC:	-2.478e+05
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Volume	1.635e-21	1.04e-22	15.712	0.000	1.43e-21	1.84e-21
Adj Close	1.0000	3.73e-18	2.68e+17	0.000	1.000	1.000

Omnibus:	2160.937	Durbin-Watson:	0.048
Prob(Omnibus):	0.000	Jarque-Bera (JB):	14252.129
Skew:	-2.376	Prob(JB):	0.00
Kurtosis:	10.600	Cond. No.	3.86e+04

So finally, have two features: Volume and Adj Close. Even though there is some collinearity, our adjusted r squared is 1. So let's perform T-test and F-test.

T-test

Null hypothesis: H0 :βi = 0 (The coefficient and intercept are equal to zero)

Alternative-hypothesis: Ha :βi ≠ 0 (The coefficient and intercept are not equal to zero)

```
T-test P values :  Volume       4.031563e-54
Adj Close    0.000000e+00
dtype: float64
```

Here the p-value of t-test for all of the features is less than 0.05. Thus we reject null hypothesis and support that the features of the final model are significant. In other words, we have enough evidence to support the claim that the features should not be removed.

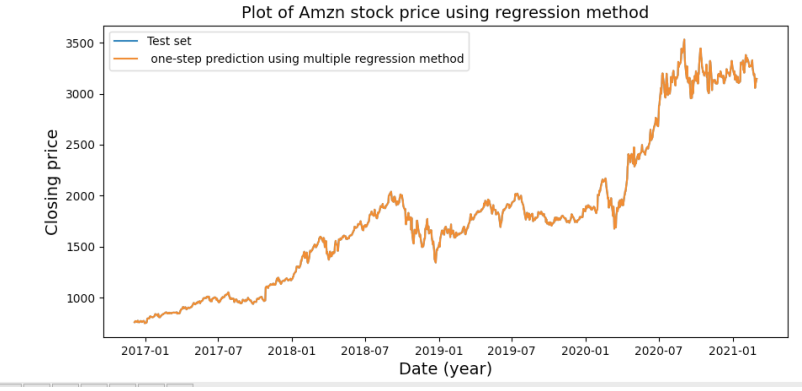
F-test:

Null hypothesis: The fit of the intercept-only model and your model are equal.

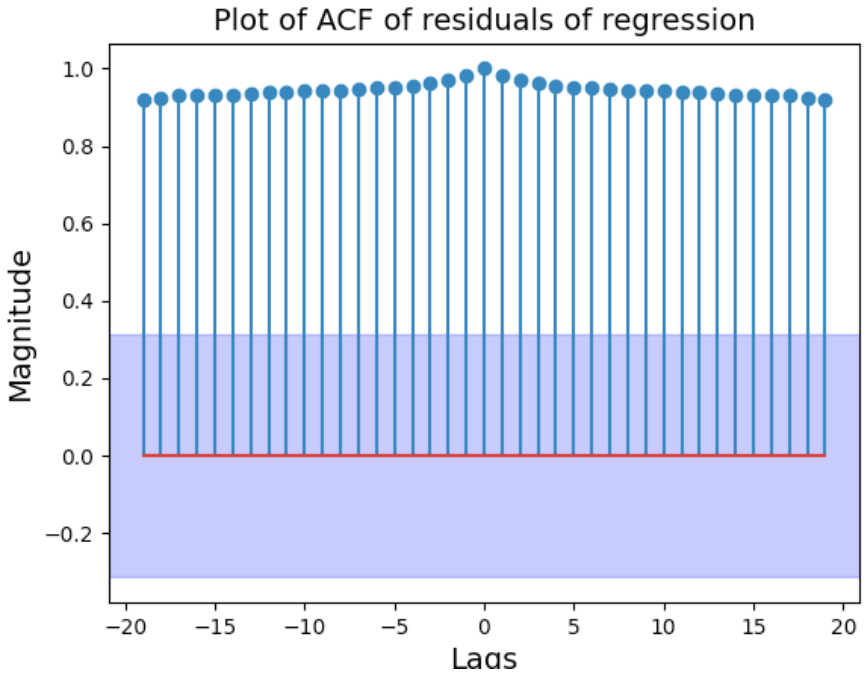
Alternative hypothesis: The fit of the intercept-only model is significantly reduced compared to your model.

```
F-statistic:  4.168728858769324e+34
Probability of observing value at least as high as F-statistic  0.0
```

Here from the F-test, probability of F-statistic is significantly lower than 0.05, so we reject the null hypothesis and support the claim that the model provides a better fit than intercept-only model.



The above plot is predictions compared with real test set. It shows that the plot of test predictions and original test set are overlapped indicating that it had real well performance on test set.



However, the ACF for the residuals for multiple regression model is not white. Let's see other results.

```
Mean square error of residuals for multiple regression is  3.1163016322155246e-27
RMSE for training set using multiple regression is :5.582384465634309e-14
The Q value of residual of regression is 58414.644974374176
The mean of residuals is -4.2633189917155826e-14
The variance of residual is 1.2987127497032477e-27

Mean square error for testing set multiple regression is  2.392343224069518e-25
RMSE for testing set using multiple regression is :4.891158578567575e-13
The mean of forecast of multiple regression is -4.401228468790825e-13
The variance of foreacast of multiple regression is 4.5526202062003473e-26

 The ratio of resid vs forecast is 0.02852670969422164
```

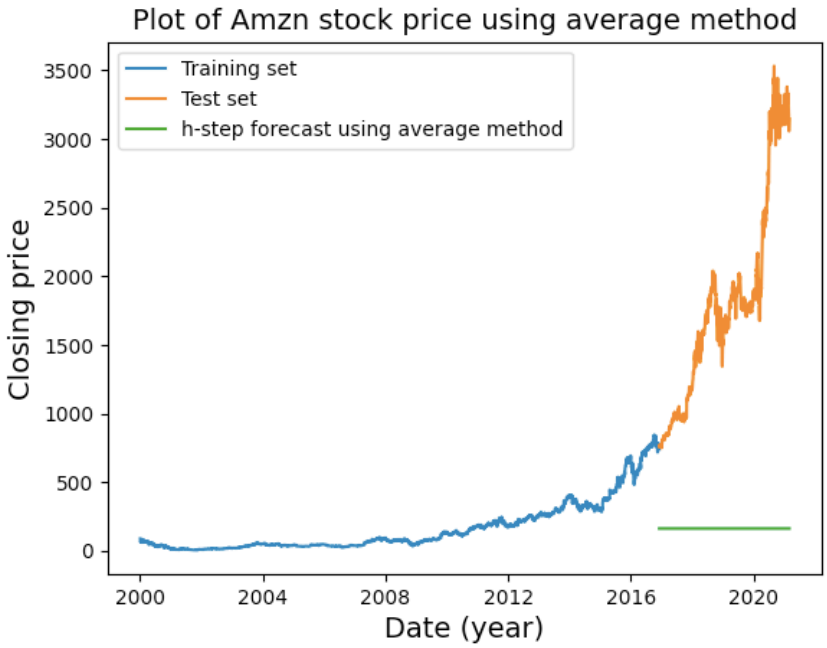
The MSE of residuals of multiple regression is low but the Q value is high.

AIC	BIC	RMSE	R-squared	Adjusted R-squared	Q value	Mean of residual	Variance Residual
-2.4e+05	-2.4e+05	5.588e-14	1.00	1.00	58400.92	-4.26e-14	1.2987e-27

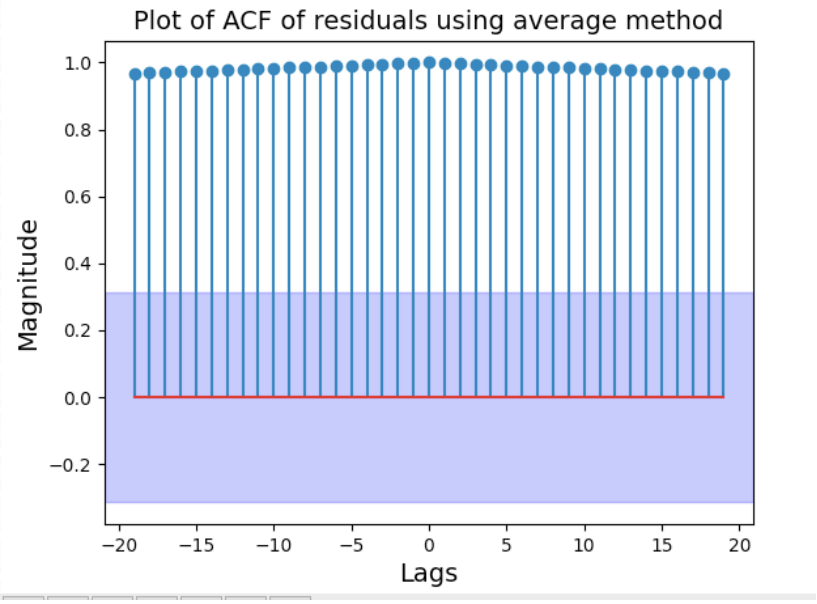
Base Models

For the base model, we start with the Average method.

a. Average method



Average model averages out all the previous observations, so in above plot the predictions are made based on the average of training set observations.



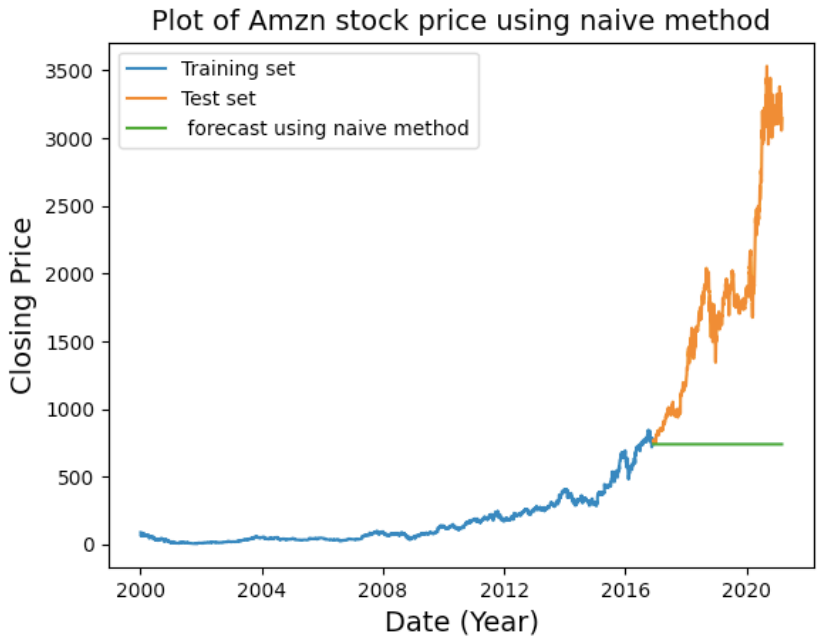
The ACF plot of residuals of average method shows that the residuals are not white.

```
MSE of prediction error (training set) using average method is : 34359.09580776401
The Q value of residual using average method is 73854.4557210594
The mean of residuals using average method is 105.17170388620056
The variaince of residual using average method is 23298.008509437353
```

```
MSE of forecast (testing set) using average method is : 3234368.512421253
Mean of forecast error is: 1643.8298708277925
Variance of forecast error is: 532191.8681955364
```

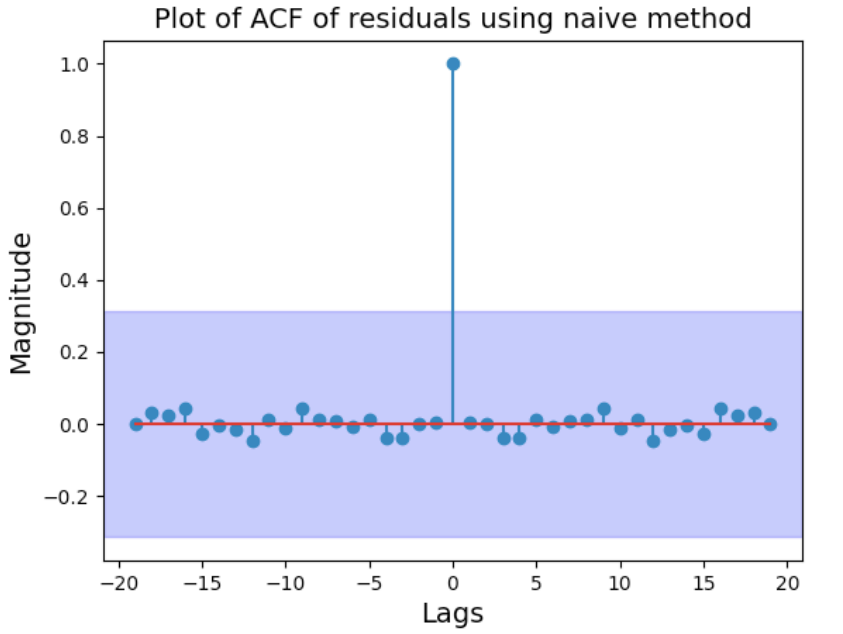
The result from the average method is shown above; the Q value of the residuals seems to be very high. Now let's see for other base models.

b. Naïve method



Naive method emphasizes or gives more weight for the latest observation, so we can see from the plot that the predictions are made out from the latest observation of training set.

Let's see the other results of Naive method



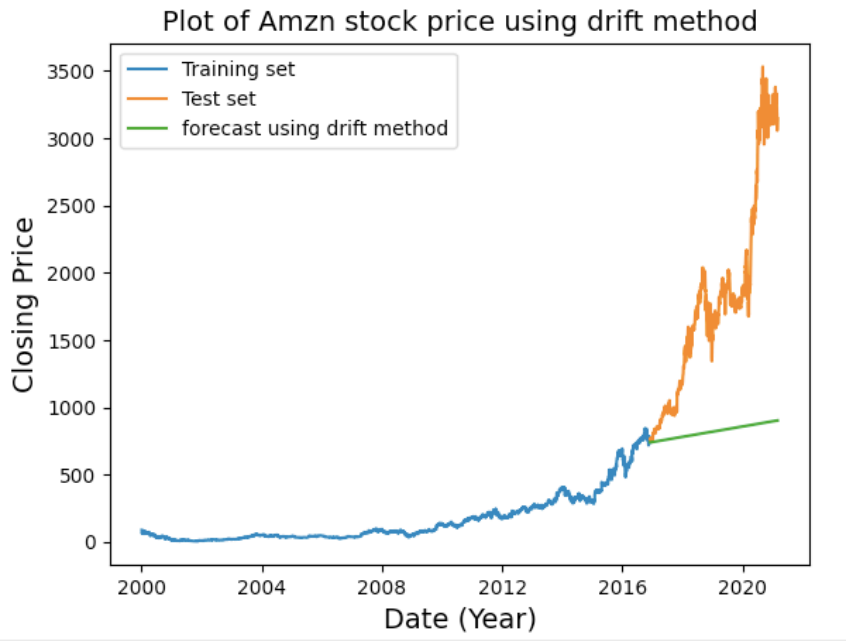
The ACF plot of the residuals seems to be white. If ACF of the residuals are white, this can indicate that the model is good. Let's look the other results

```
MSE of prediction error (training set) using naive method is : 23.259027367700547
The Q value of residual using naive method is 51.6833850712938
The mean of residuals using naive method is 0.1529163793411954
The variance of residual using naive method is 23.235643948629722

MSE of prediction error (testing set) using naive method is : 1670489.4759726904
Mean of forecast error using naive method is: 1066.9103091530956
Variance of forecast error using naive method is: 532191.8681955362

 The ratio of resid vs forecast of naive method is 4.3660276184627006e-05
```

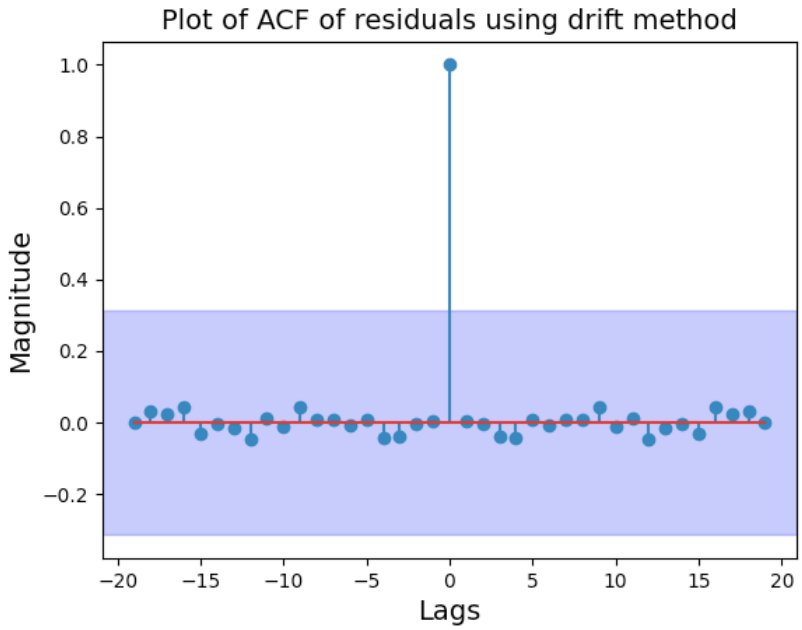
The MSE of prediction for training set is low with low Q values but MSE for the predicted values of training set is very high. Even the variance of forecast error seems to be high.

Now let's see the other base model:

c. Drift method



The drift method is based on previous observations but makes extrapolated predictions for future observations. This is exactly observed in the plot.

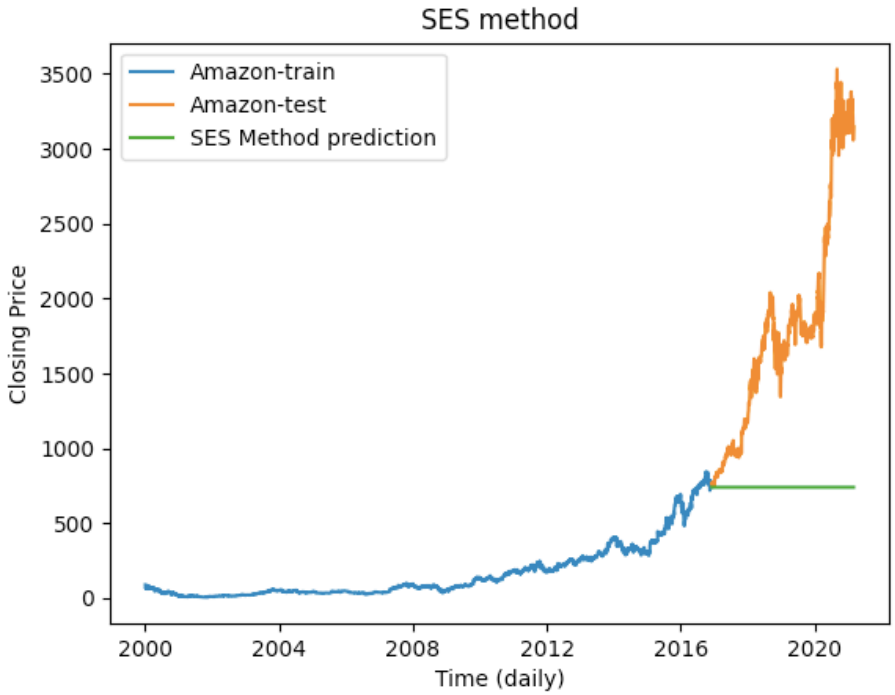


The ACF of residuals of drift method looks white. Like naïve method, this also indicates that the model is good. Let's see the other results.

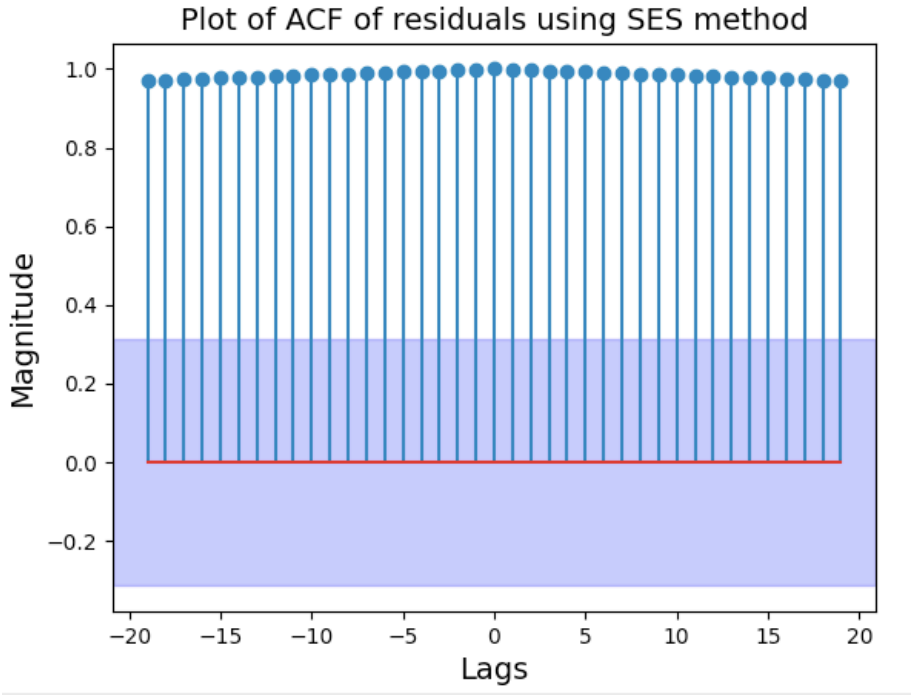
```
MSE of prediction error (training set) using drift method is : 23.275022222924193
The Q value of residual using drift method is 52.00512894643638
The mean of residuals using drift method is 0.1937422107143655
The variance of residual using drift method is 23.237486178711706

MSE of prediction error of testing set using drift method is : 1442115.1436984742
Mean of forecast error using drift method is: 985.4058789642384
Variance of forecast error using drift method is: 471090.3974011908

 The ratio of resid vs forecast of drift method is 4.9327021537486696e-05
```

From the result, we can observe that the MSE and Q value for the residuals are low. However, the MSE of forecast set and variance of forecast error is high.

d. SES method



Simple exponential smoothing is calculated using weighted averages where the weights decreases exponentially as observations come from further in the past, the smallest weights are associated with the oldest observations. It kind of a looks like naïve and average.



The ACF of residuals of SES method do not look white. Let's see other results

```
Mean square error for (training set) simple exponential smoothing is  367177.9423485008
The Q value of residual using SES method is 74158.5207647678
Mean of residual using SES method is: -576.9195617240197
Variance of residual using SES method is: 34341.761648665924
Mean square error for (testing set) simple exponential smoothing is  1670489.4758674442
Mean of forecast error using SES method is: 1066.9103091037728
Variance of forecast error using SES method is: 532191.8681955362
```

The MSE of training set and Q value is very high compared to naïve and drift method.

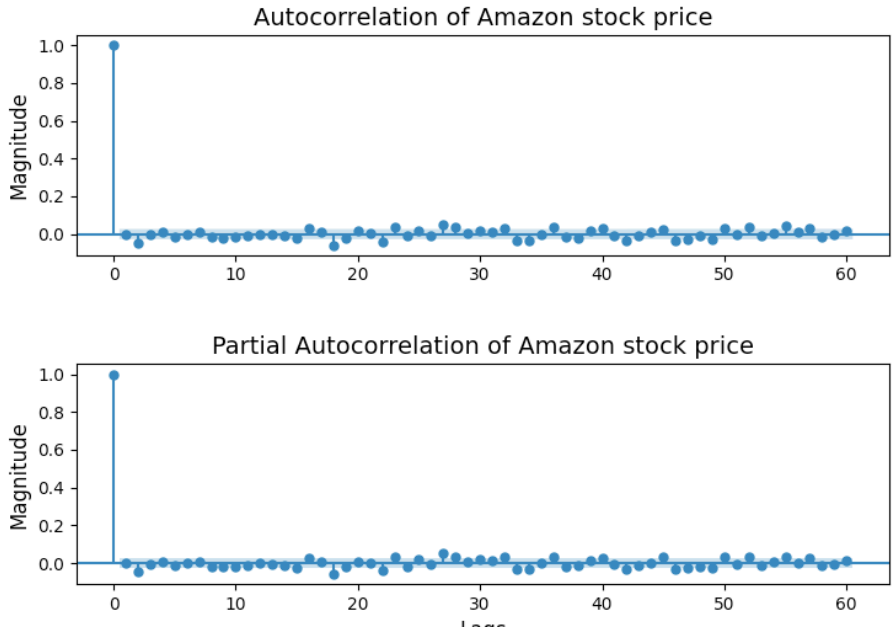
ARMA and ARIMA and SARIMA model order determination

After having multiple regression, holt-winter and other base models, now let's look into our ARMA and ARIMA model.

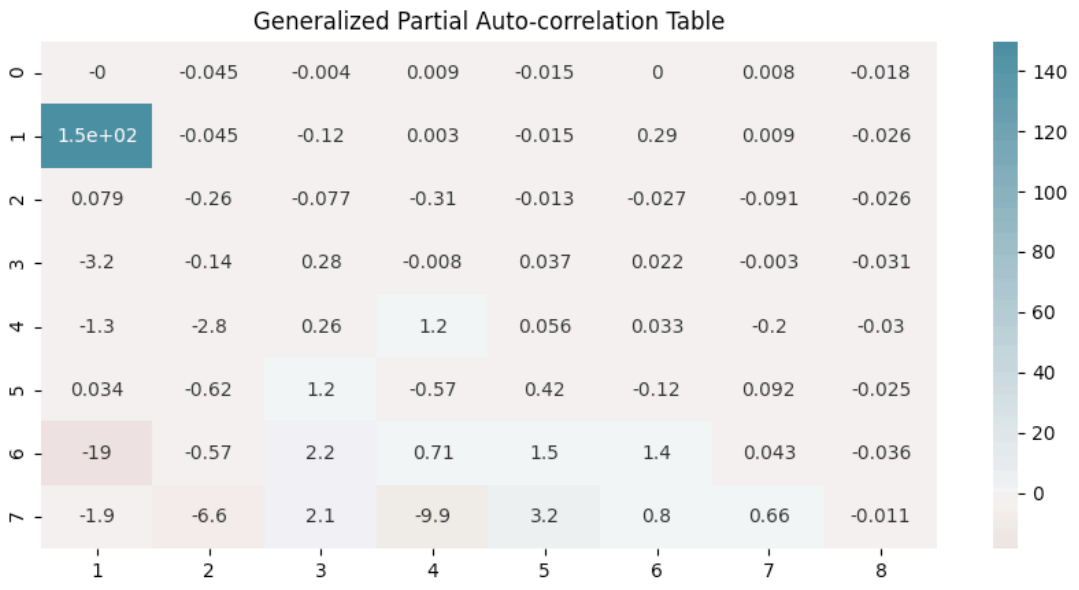
To develop ARMA and ARIMA model means to know the co-efficient and order of coefficients for our dataset. However all the process initiates with the process of making dataset stationary. The ADF test, rolling mean and variance and ACF/PACF plot should validate the data to be stationary. Once the data is stationary, we use the stationary dataset to estimate the order. We feed the stationary data into General Partial Autocorrelation function to get the AR and MA orders. Once we have different sets of estimated orders, we use that information to estimate the

coefficients. Our estimated coefficients will be validated by residual diagnostics, then only will be forwarded to make forecast.

Let's revise our stationary ACF/PACF plot.

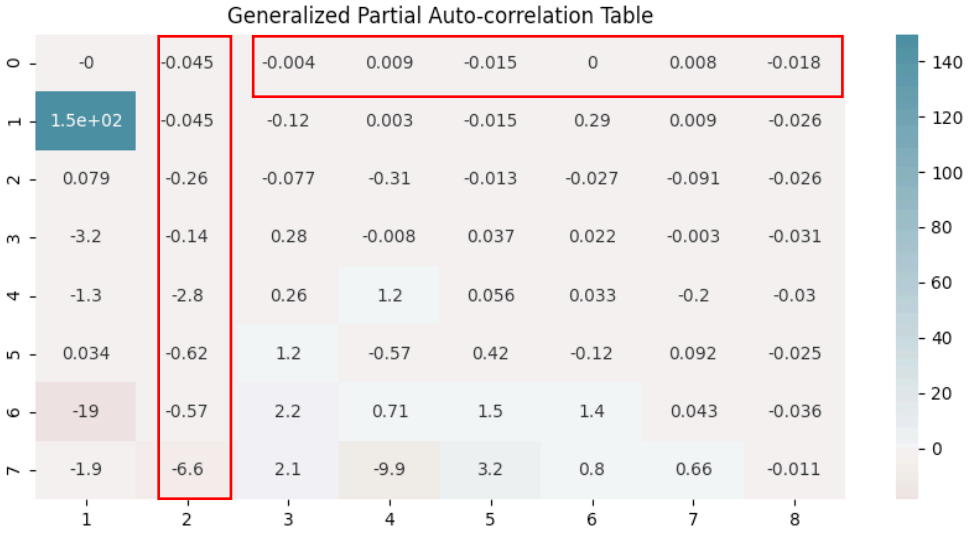


Many of the times, our orders can be estimated from ACF/PACF plot. Generally, if there is decay in ACF plot and cut off in PACF plot then it refers that we will be Auto-regressive(AR) orders. In contrast to it, if there is cut off in ACF plot and decay in PACF plot, it refers that we will have Moving Average (MA) orders. In our case, the ACF/PACF plot do not seem to have the pattern revealed so let's plot the GPAC table. In GPAC table, we look for the columns that have pattern of constant number and we look for the row that have values closer to zero.

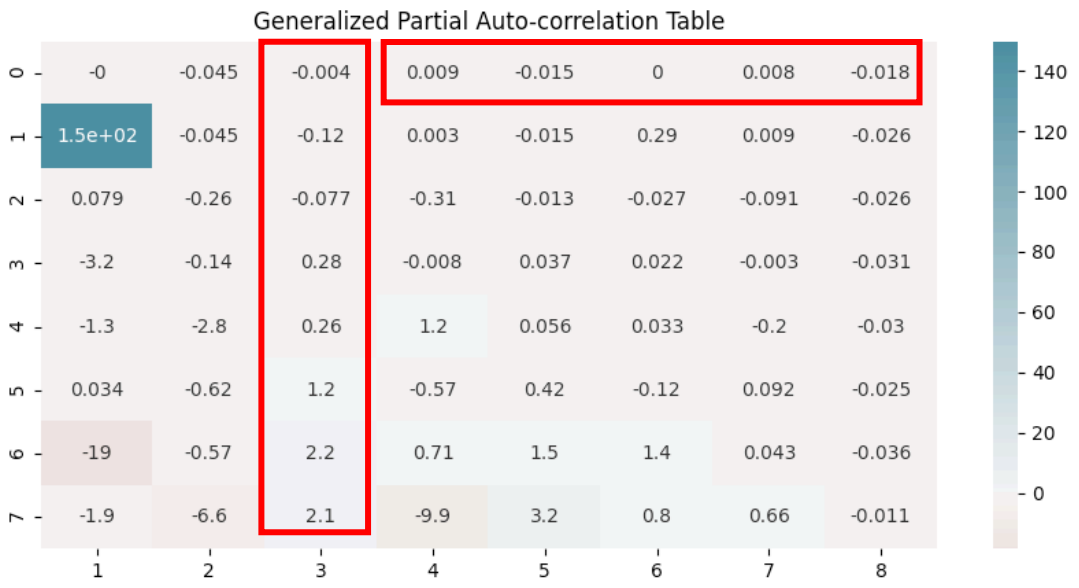


We do not have the clear pattern here to get AR and MA orders, so let's dive into it and highlight the order that seems to be right.

Possible patterns : ARMA(2,0)



Possible Pattern: ARMA(3,0)



Now once we have our potential orders, we use Levenberg Marquardt algorithm to estimate our parameters.

```
For our estimated ARMA (2,0):
The estimated parameters >>>  [-0.94214316 -0.05887971]

 The estimated co-variance matrix is [[ 0.00018755 -0.00018772]
 [-0.00018772  0.00018797]]

 The estimated variance of error is 332.4465440000013
The standard deviation for a1 is 0.013695007280738294
The standard deviation for a2 is 0.01371028318396496

 The confidence interval for parameters are:
-0.9695331789638233 < a1 < -0.9147531498408701
-0.08630028131971126 < a2 < -0.03145914858385143
```

Here we only have roots of denominators we do not have zero/pole cancellation. Also the confidence interval do not include zero so the estimated parameters are statistically significant.

========== Writing the model==================
ARMA (2,0)

---- y(t) – 0.9421436 * y(t-1) - 0.05887971* y(t - 2) = e(t)
-----y(t) = 0.9421436 * y(t-1) + 0.05887971* y(t - 2) + e(t)

y_hat_(t) = 0.9421436 *** y(t-1)** + 0.05887971*** y(t - 2) + e(t)**

$y\_hat_{(t)}(1)$ = 0.9421436 *** y(t)** + 0.05887971*** y(t - 1)**

After this we do one-step prediction

Let's see the Q value.

```
The Q value is   5246.321652277973
The chi-critical is 4473.566707528634

The Q value is not less than 4473.566707528634 (chi-critical) so, the residual is not white
```

```
Mean of residual with ARMA(2,0) is -0.15692198217023456
The variance of residual with ARMA(2,0) is 0.11710247534189062
```

From the above results after estimating the parameters we made one-step prediction and calculated the residual.

Also the mean of residual is -0.16 which means it is close to zero so it means that the derived model is unbiased estimator. When the Q-value is greater than chi-critical, the residual is not white. The residual is not white, so let's try with another potential order.

Thus lets try another ARMA(3,0) pattern

```
For our estimated ARMA (3,0):
The estimated parameters >>>  [-9.42118813e-01 -5.84963746e-02 -4.08243678e-04]

 The estimated co-variance matrix is [[ 1.88260167e-04 -1.77184776e-04 -1.12611022e-05]
 [-1.77184776e-04  3.54464912e-04 -1.77271519e-04]
 [-1.12611022e-05 -1.77271519e-04  1.88787799e-04]]

 The estimated variance of error is 332.50897877507066
The standard deviation for a1 is 0.013720793236619734
The standard deviation for a2 is 0.018827238572974997
The standard deviation for a3 is 0.013740007239366756

 The confidence interval for parameters are:
-0.9695603993167882 < a1 < -0.9146772263703092
-0.09615085178740305 < a2 < -0.020841897495503067
-0.027888258156760125 < a3 < 0.0270717708007069

The roots of the numerators are []
The roots of dinominators are [ 1.00096618 -0.05082236 -0.008025  ]
```

Here we only have roots of denominators, so we do not have zero/pole cancellation. Also last confidence interval includes zero so one of the estimated parameters is not statistically significant. Thus, we remove the estimated parameter that is not statistically significant and see the Q value.

```
The estimated parameters are -0.9421188128435487 and -0.05849637464145306
So lets do one-step prediction accordingly.
```

========== Writing the model==================
ARMA (2,0)

---- y(t) – 0.9421188128435487 * y(t-1) - 0.05849637464* y(t - 2) = e(t)
-----y(t) = 0.9421188128435487 * y(t-1) + 0.05849637464* y(t - 2) + e(t)

y_hat_(t) = 0.9421188128435487 * y(t-1) + 0.05849637464* y(t - 2) + e(t)

$y\_hat_{(t)}(1) = 0.9421188128435487 * y(t) + 0.05849637464* y(t - 1)$

After this we do one-step prediction

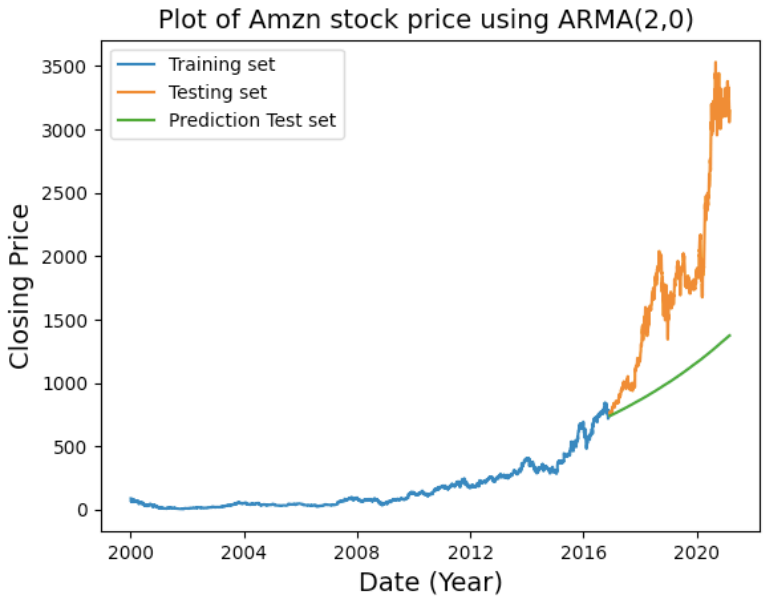
```
The Q value is   858.2739048417094
The chi-critical is 4473.566707528634

 The Q value is less than 4473.566707528634 (chi-critical) so, the residual is white
MSE of residual for ARMA(2,0) is 0.10355628340805502

Mean of residual with ARMA(2,0) is -0.09036343878801825
The variance of residual with ARMA(2,0) is 0.09539073233845911

MSE of forecast for ARMA(2,0) is 1176592.1112770392
The mean of trainings set error is 781.6973090618258
The variance of training set error is 565541.4282825392

The ratio of variance of residual to variance of forecast is 1.6867152001250525e-07
```

After removing one estimated parameter, we did one-step prediction and calculated the Q value.

The Q value is less than chi-critical and the residuals are white which is sign of good model. The mean of residual is -0.09 (close to zero), so we can say that the derived model is an unbiased estimator.

Then we compared its performance with the test set by calculating the ration of variance of residual to variance of forecast which is very low and kind of away from 1.

```
MSE of residual for ARMA(2,0) is 0.10355628340805502

Mean of residual with ARMA(2,0) is -0.09036343878801825
The variance of residual with ARMA(2,0) is 0.09539073233845911

MSE of forecast for ARMA(2,0) is 1176592.1112770392
The mean of trainings set error is 781.6973090618258
The variance of training set error is 565541.4282825392

The ratio of variance of residual to variance of forecast is 1.6867152001250525e-07
```



With ARMA(2,0) our Q value is 858. 072, so let's try ARIMA model and see if we can have model that will yield us a less Q value than 858.072.

ARIMA(2,0) is our potential order, so let's see our ARIMA(2,0) model results

```
For our ARIMA(2,0)
The estimated parameters >>>  [-0.00014876  0.04469821]

 The estimated co-variance matrix is [[ 1.87604404e-04 -2.01844630e-08]
 [-2.01844630e-08  1.87609056e-04]]

 The estimated variance of error is 0.0010237275761098231
The standard deviation for a1 is 0.01369687569837044
The standard deviation for a2 is 0.013697045524060035

 The confidence interval for parameters are:
-0.027542509979648824 < a1 < 0.027244992813832938
0.017304116980104114 < a2 < 0.07209229907634425

The roots of the numerators are []
The roots of dinominators are [7.43792915e-05+0.21141949j 7.43792915e-05-0.21141949j]
```

Here we only have roots of denominators we do not have zero/pole cancellation. Also first confidence interval includes zero so one of the estimated parameters is not statistically significant. Thus, we remove the estimated parameter that is not statistically significant and develop our model

```
The estimated parameter after removing the statistically not significant parameter is 0.04469821
```

========== Writing the model==========
ARIMA (1,1,0)
---- $(1 + 0.04469821\ q^{-1})\ (1- q^{-1})\ y(t) = e(t)$
---- $y(t) - q^{-1} * y(t) + 0.04469821\ q^{-1} * y(t) - 0.04469821\ q^{-2}\ y(t) = e(t)$
-----$y(t) = 0.95530179\ q^{-1}\ y(t) - 0.04469821\ q^{-2}\ y(t) + e(t)$

$y\_hat\_(t) = 0.95530179\ y(t-1) - 0.04469821\ y(t-2) + e(t)$

thet = [0.95530179 , - 0.04469821]

Now let's perform one-step prediction and calculate the Q value.

```
MSE of resid of ARIMA(1,1,0) is 487.31403071067797

The Q value is   74170.58909377547
The chi-critical is 4474.591921592612

The Q value is not less than 4474.591921592612 (chi-critical) so, the residual is not white
```

ARIMA(1,1,0) with estimated parameter of 0.044692821 has lower MSE compared to other models, but the Q values is high. The Q value is 74170.

```
Mean of residual with ARIMA(1,1,0) is 14.60143284767495
The variance of residual with ARIMA(1,1,0) is 274.1121895055171
MSE of prediction for testing error is 3776394.5835568015

The mean of error of testing ARIMA is1800.5185285464388
The variance of forecast error is 534527.6119177697

The variance of forecast error is 534527.6119177697
The ratio of variance of residual to variance of forecast is 0.000512812029526523
```

Also, the mean of residual of ARIMA(1,1,0) is 14.60 ; this is not close to zero. Thus the derived model is biased estimator. Now let's compare all the models we have with us and find the best model that represent our dataset.

Final Model Selection

Metrics	Regression	Avg	Naïve	Drift	SES	HWM	ARMA(2,0)	ARIMA(1,1,0)
MSE residual	3.11e-27	34359.09	23.25	23.27	367177.94	290015488	0.10355	487.3140
Q value	58414.644	73854.10	51.68	52.00	74158.104	73994.148	858.27	74170.169
Mean Residual	-3.339e-14	105.17	105.17	0.193	-576.919	-92651.54	-0.0903	14.60
Var Residual	1.57e-27	23298.008	23298.01	23.23	34341.76	20417240939	0.0953	274.1121
MSE Test	2.39e-25	3234368.51	1670489.47	1442115	1670489.47	130608.435	1176592.11	3776934.58
Mean Test	-4.40e-13	1643.829	1066.910	985.40	1066.91	-129.8040	781.69	1800.51
Var Test	4.55e-26	532191.86	532191.86	471090.39	532191.868	113,759.35	565541.428	534,527
Ratio (var residual) (var forecast)	0.0346	0.043777	4.366e-05	4.932e-05	0.064	179477.473	1.6867e-07	0.00051

Based on the previous comparison, Naïve, Drift, ARMA(2,0) model have white ACF residuals. Thus among them **Naïve and Drift have the lower Q value with** pretty low MSE of residuals.

So based on our Q value, Drift method is the best model that represent our dataset, I chose drift method over naïve because naïve method is based on last observation, but drift method extrapolates towards the future while predicting future.

Now let's develop the forecast function for our model and do h-step ahead predictions. The rule of thumb for doing h-step predictions is to divide the total number of rows by 50. Thus, in my case I will have around 100 step ahead predictions.

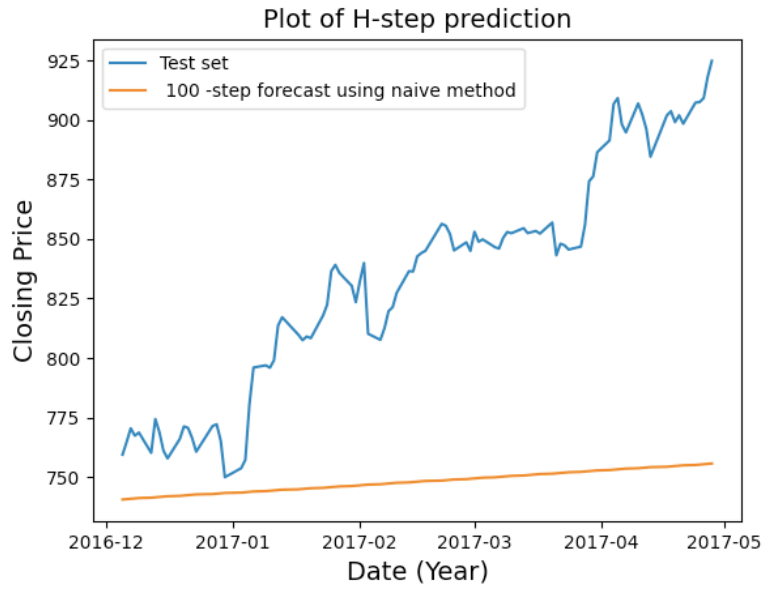
Best model: Drift method

Forecast function for the best model

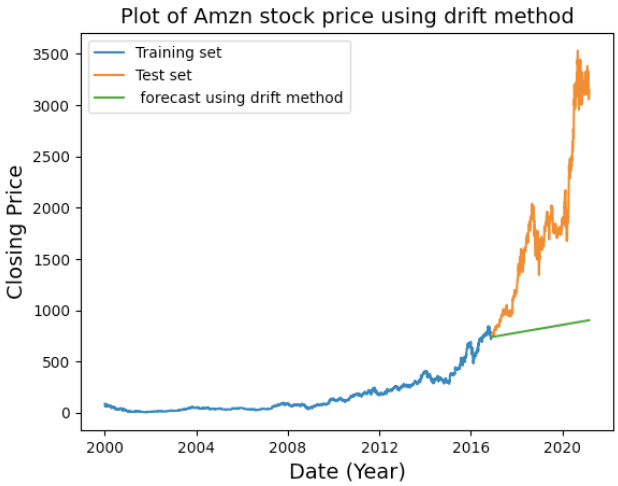
```
def forecast_function(Y_train, step):
  y_predict_test_set_drift= []
  for h in range(step):
    slope_val = (Y_train[-1] - Y_train[0] ) /( len(Y_train) - 1 )
    y_predict_each = Y_train[-1] + ((h +1) * slope_val)
    y_predict_test_set_drift.append(y_predict_each)

  return y_predict_test_set_drift
```

The lowest Q value means that it fit best for training set. So n. Let's plot the h-step prediction and analyze it.

H-step prediction



From the plot we can see that our h-step prediction is based on drift model for 100 steps ahead is like a slightly increasing trend. It is the nature of drift method to be extrapolatd towards the future based on the previous observations so based on it's nature, the h-step prediction looks fine. However, as the time goes (moving from left to right), the prediction seems to deviate a lot than the original test set. In other words, it might not perform that well.



Conclusion

In conclusion, this paper is a detail report for the project of explanation of time series analysis of Amazon stock price. This report includes calculation, explanation and comparison of Holt-winter method, Multiple Regression method, Base models (Average method, Drift method, Naïve method, SES method), ARMA method and ARIMA method. From all the comparison results, Drift was the best model according to the model was trained. However, the performance for testing set was not like the model predicted for training set. The benchmark model out trained the other models. Even though Drift was best fit for the training data, it did not predict precisely for testing set. Based on the results, drift method is also not too bad. However, from this project analysis we should keep in mind that in real world Amazon stock closing price is not just predicted on basis of its own past observations. In other words, our all the models (except regression model) that we came up with was univariate modeling. Thus I would say multivariate modeling with other financial indicator variables might help to boost the performance.

Appendix

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

```
register_matplotlib_converters()


Amzn_df = web.DataReader('AMZN', data_source='yahoo',start ='2000-01-01', end='2021-03-01')
len(Amzn_df)
depend_var = Amzn_df['Close']

print(len(Amzn_df))

#5.a. Plot of the dependent variable versus time.
plt.plot(depend_var, label='Closing price of Amazon')
plt.legend()
plt.title('Plot of dependent variable(closing price) versus no of sample(time)')
plt.xlabel('Date',fontsize=12)
plt.ylabel("Closing price",fontsize=12)
plt.show()

#5.b. ACF/PACF of the dependent variable.
def plot_acf_pacf(y, title):
   lags=60
   acf =sm.tsa.stattools.acf(y, nlags=lags)
   pacf =sm.tsa.stattools.acf(y, nlags=lags)

   fig = plt.figure(figsize = (8, 6))
   plt.subplot(211)
   plot_acf(y,ax=plt.gca(), lags =lags)
   plt.ylabel('Magnitude', fontsize=12)
   plt.title('Autocorrelation of Amazon stock price ' +title, fontsize=14)
   plt.subplot(212)
   plot_pacf(y, ax=plt.gca(), lags=lags)
   plt.xlabel('Lags', fontsize=12)
   plt.ylabel('Magnitude', fontsize=12)
   plt.title('Partial Autocorrelation of Amazon stock price' +title, fontsize=14)
   fig.tight_layout(pad = 3)
   plt.show()

plot_acf_pacf(Amzn_df['Close'], title ='')


#5.c. Correlation Matrix with seaborn heatmap and Pearson's correlation coefficient.

corr_mx =  Amzn_df.corr()
ax = sns.heatmap(corr_mx, 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')
ax.set_title("Heatmap of the matrix showing co-relation")
plt.show()

#5.d. Preprocessing procedures (if applicable): Clean the dataset (no missing data or NAN)
#checking if the dataset has any nan values

print(Amzn_df.isna().sum())
```

```
#using pearsons correlation
pearsoncorr = Amzn_df.corr(method='pearson')
print("The pearson correlation coefficeint is : \n",pearsoncorr)


#5.e.Split the dataset into train set (80%) and test set (20%).
Amzn_df_train, Amzn_df_test= train_test_split(Amzn_df, test_size = 0.2, shuffle=False)

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

X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.2, shuffle=False)

#6. Stationarity check


def plot_rolling_mean_var(x, title1, title2):

    mean_list_rolling_mean= []
    var_list_rolling_var= []

    for i in range(len(x)):
        try:
            each_row = x.head(i)
            mean_each_val = each_row.mean()
            var_each_row = each_row.var()
        except:
            mean_each_val = np.mean(x[:i + 1])
            var_each_row = np.var(x[:i + 1])

        mean_list_rolling_mean.append(mean_each_val)
        var_list_rolling_var.append(var_each_row)

    fig = plt.figure()
    ax1 = fig.add_subplot(2, 1, 1)
    ax1.plot(mean_list_rolling_mean)
    ax1.set_title(title1)
    ax1.set_xlabel("Observations")
    ax1.set_ylabel("Mean")

    ax2 = fig.add_subplot(2, 1, 2)
    ax2.plot(var_list_rolling_var)
    ax2.set_title(title2)
    ax2.set_xlabel("Observations")
    ax2.set_ylabel("Variance")

    fig.tight_layout(pad =2)
    plt.show()

plot_rolling_mean_var(Amzn_df['Close'], 'Rolling mean of Amzn before differencing', 'Rolling variance of
Amzn before differencing')
```

```
#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))

ADF_Cal(Amzn_df['Close']) #caliing ADF function


#using differencing technique

log_transformed_data = np.log(Amzn_df['Close'])
plot_rolling_mean_var(log_transformed_data, 'Rolling mean of Amazon after log differencing', 'Rolling
variance of Amazon after log differencing')
ADF_Cal(log_transformed_data)
plot_acf_pacf(log_transformed_data, title ='after LOG differencing')


print("AFTER LOG differening used first differencing")

#Amzn_diff1 = Amzn_df['Close'].diff()
Amzn_diff1 = log_transformed_data.diff()[1:]

ADF_Cal(Amzn_diff1)

plot_rolling_mean_var(Amzn_diff1, 'Rolling mean of Amazon using log and first differencing', 'Rolling
variance of Amazon using log and first differencing')

plot_acf_pacf(Amzn_diff1, title='after log and first differencing')


#7.Time Series Decomposition

#using orginal y value data series for time series decomposition
df = pd.Series(np.array(Amzn_df['Close']), index = pd.date_range(start = '2000-01-03', periods
=len(Amzn_df['Close']), freq='b'), name= 'Amazon Closing Price observation')

STL = STL(df)
res = STL.fit()

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

#print(S)
#print(T)

fig = res.plot()
plt.show()

detrended = df -T  #detrended
```

```
adjusted_seasonal = df -S  #adjusted seasonal dataset


plt.figure()
#plt.plot(adjusted_seasonal, label='Adjusted Seasonal') #plots first 100 sample
plt.plot(detrended, label='Detrended dataset')
plt.plot(df, label ='Original set')
plt.legend()
plt.ylabel("Closing price")
plt.xlabel("Year")
plt.title("Plot of Detrended and original dataset ")
plt.show()

plt.figure()
plt.plot(adjusted_seasonal, label='Adjusted Seasonal')
#plt.plot(detrended, label='Detrended dataset')
plt.plot(df, label ='Original set')
plt.legend()
plt.ylabel("Closing price")
plt.xlabel("Year")
plt.title("Plot of Adjusted seasonal and original dataset ")
plt.show()



#----------------
#strength of Trend
#-------------
R= np.array(R)
T= np.array(T)
S= np.array(S)

#ratio_strength_trend = np.mac([0,1 - R.var()/adjusted_seasonal.var()])
ratio_strength_trend =np.max([0,1 - (R.var())/((T+R).var())])
print("\n\nThe strength of trend for this data set is ", ratio_strength_trend)


#----------------
#strength of seasonality
#-------------

ratio_strength_seasonlaity =np.max([0,1 -(R.var())/(S+R).var()])
print("\nThe strength of seasonality for this data set is", ratio_strength_seasonlaity)

# 8. Using the Holt-Winters method try to find the best fit

#print(Amzn_df_train)
#print(Amzn_df_test)

train_HLWM =
ets.ExponentialSmoothing(Amzn_df_train['Close'],trend='multiplicative',damped_trend=False,seasonal='mu
ltiplicative', seasonal_periods =12).fit()
HLWM_prediction_train = train_HLWM.forecast(steps=len(Amzn_df_train['Close']))
test_HLWM = train_HLWM.forecast(steps=len(Amzn_df_test['Close']))
test_predict_HLWM = pd.DataFrame(test_HLWM).set_index(Amzn_df_test['Close'].index)
```

```
resid_HLWM = np.subtract(Y_train.values,np.array(HLWM_prediction_train))
forecast_error_HLWM = np.subtract(Y_test.values,np.array(test_HLWM))

MSE_HLWM = np.square(resid_HLWM).mean()
print("Mean square error for (training set) HLWM is ", MSE_HLWM)

#print(test_predict_HLWM)


def auto_corelation(x, bar_x, n):
    list_ry =[]
    for i in range(n):
        total_numerator = 0
        total_dinom = 0
        a = i
        for t in range(len(x)):
            if a < len(x):
                each_sum = (x[a] - bar_x) * (x[t] -bar_x)
                total_numerator = each_sum + total_numerator
                each_dinom = (x[t] - bar_x) ** 2
                total_dinom = each_dinom + total_dinom
                a = a +1
            else:
                each_dinom = (x[t] - bar_x) ** 2
                total_dinom = each_dinom + total_dinom
        value = total_numerator / total_dinom
        list_ry.append(value)

    inverse_generated_list_ry = list_ry[::-1]
    Generated_list_Ry = inverse_generated_list_ry[:-1] + list_ry
    Generated_list_Ry_np = np.array(Generated_list_Ry)
    #return list_ry, Generated_list_Ry_np

    return Generated_list_Ry_np


# ================== Auto-correlation function=============

def calc_auto_corelation(x, bar_x, n, title):
    list_ry = []
    for i in range(n):
        total_numerator = 0
        total_dinom = 0
        a = i
        for t in range(len(x)):
            if a < len(x):
                each_sum = (x[a] - bar_x) * (x[t] - bar_x)
                total_numerator = each_sum + total_numerator
                each_dinom = (x[t] - bar_x) ** 2
                total_dinom = each_dinom + total_dinom
                a = a + 1
            else:
                each_dinom = (x[t] - bar_x) ** 2
                total_dinom = each_dinom + total_dinom
```

```
        value = total_numerator / total_dinom
        list_ry.append(value)

    inverse_generated_list_ry = list_ry[::-1]
    Generated_list_Ry = inverse_generated_list_ry[:-1] + list_ry
    Generated_list_Ry_np = np.array(Generated_list_Ry)

    x_np = np.linspace(-19, 19, 39)
    n = 1.96/(np.sqrt(len(Generated_list_Ry_np)))
    plt.axhspan(-n,n, alpha = 0.25, color= 'blue')
    plt.stem(x_np, Generated_list_Ry_np)
    plt.xlabel("Lags", fontsize=14)
    plt.ylabel("Magnitude", fontsize=14)
    plt.title(title, fontsize=14)
    plt.show()

    return list_ry, Generated_list_Ry_np



list_acf, acf_resid = calc_auto_corelation(resid_HLWM, np.mean(resid_HLWM), 20, title ='ACF plot of
residuals for Holt-Winter method')


Q  = (len(resid_HLWM)) *np.sum(np.square(acf_resid)[21:])

print("\nThe Q value of residual using HWM is  ",Q)

print(f"The Mean of residual of HLWM is {np.mean(resid_HLWM)}")
print(f"The variance of residual of HLWM is {np.var(resid_HLWM)}")

MSE = np.square(np.subtract(Amzn_df_test['Close'].values,np.ndarray.flatten(test_HLWM.values))).mean()
print("Mean square error for Holt Winter method is of testing set is ", MSE)


print(f"The Mean of forecast of HLWM is {np.mean(forecast_error_HLWM)}")
print(f"The variance of forecast of HLWM is {np.var(forecast_error_HLWM)}")

print(f"\n The ratio of resid vs forecast is {(np.var(resid_HLWM)) / (  np.var(forecast_error_HLWM)  )}")

plt.plot(Amzn_df_train['Close'],label= "Amazon-train")
plt.plot(Amzn_df_test['Close'],label= "Amazon-test")
plt.plot(test_predict_HLWM,label= "Holt-Winter Method-test")
plt.legend(loc='upper left')
plt.title('Holt-Winter Method')
plt.xlabel('Time (daily)')
plt.ylabel('Closing Price')
plt.show()

#============
# Feature selection
#=========================
X= Amzn_df[['Open','High','Low','Volume','Adj Close']]
Y =Amzn_df['Close']
```

```
#X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.2, shuffle=False)
X_train_copy = X_train.copy()
X_test_copy = X_test

X_val = np.c_[np.ones((4258,1)),X_train.values]
X_val_cond = np.c_[X_train.values]
H_vector = np.matmul(X_val.T, X_val)
s, d, v = np.linalg.svd(H_vector)
print(f"SingularValues = {d}")
print(f"\nThe condition number including constant in original data= {LA.cond(X_val)}")
print(f"The condition number without constant (original data) = {LA.cond(X_val_cond)}")


X_train=sm.add_constant(X_train)
model = sm.OLS(Y_train, X_train).fit()
X_test = sm.add_constant(X_test)
predictions =model.predict(X_test)
print('')
print(model.summary())

#Using backstep regression

###--------Removing high

print("\n\n\nRemoving High")
X_train.drop(['High'], axis =1, inplace =True)
model = sm.OLS(Y_train, X_train).fit()
X_test.drop(['High'], axis =1, inplace =True)
predictions1 =model.predict(X_test)
print(model.summary())

###--------Removing const

print("\n\n\nRemoving const")
X_train.drop(['const'], axis =1, inplace =True)
model = sm.OLS(Y_train, X_train).fit()
X_test.drop(['const'], axis =1, inplace =True)
predictions2 =model.predict(X_test)
print(model.summary())

###--------Removing low

print("\n\n\nRemoving low")
X_train.drop(['Low'], axis =1, inplace =True)
model = sm.OLS(Y_train, X_train).fit()
X_test.drop(['Low'], axis =1, inplace =True)
predictions3 =model.predict(X_test)
print(model.summary())

###--------Removing Open

print("\n\n\nRemoving Open")
X_train.drop(['Open'], axis =1, inplace =True)
model = sm.OLS(Y_train, X_train).fit()
X_test.drop(['Open'], axis =1, inplace =True)
```

```
predictions4 =model.predict(X_test)
print(model.summary())

###--------Removing Adj Close will decrease adjusted r squared

# prediction 4 is the final prediction for x test based on my multiple regression model
#so I will name it as prediction_test

prediction_test = predictions4


#ftest --- Multiple regression
f_test = model.fvalue
print('\nF-statistic: ', f_test)

print("Probability of observing value at least as high as F-statistic ",model.f_pvalue)

#t-test
t_Test =model.pvalues
print("\nT-test P values : ", t_Test)

#now I nedd to predict based on train set so....

model = sm.OLS(Y_train, X_train).fit()
prediction_train =model.predict(X_train)
training_residual = np.subtract(Y_train,prediction_train)

MSE = np.square(training_residual).mean()
print("Mean square error of residuals for multiple regression is ", MSE)
print(f"RMSE for training set using multiple regression is :{np.sqrt(MSE)} ")



def calc_Q_value(x):
   #calc_autocorr,calc_autocorr_np = auto_corelation(x, statistics.mean(x), n = 5)
   Q_calc_autocorr = []
   for i in x:
      i = i ** 2
      Q_calc_autocorr.append(i)
   Q_value = len(x) * sum(Q_calc_autocorr[21:])
   return Q_value



#calling auto-corelation function
list_acf_residauls, np_acf_calc_residuals = calc_auto_corelation(training_residual, np.mean(training_residual),
n=20, title='Plot of ACF of residuals of regression')


#calling function to calculate Q values

Q  = (len(training_residual)) *(np.sum(np.square(np_acf_calc_residuals)[21:]))

print(f"The Q value of residual of regression is {Q}")
```

```
print(f"The mean of residuals is {np.mean(training_residual)}")
print(f"The variance of residual is {np.var(training_residual)}")



testing_error_regression= np.subtract(Y_test, prediction_test)
MSE = np.square(testing_error_regression).mean()
print("\nMean square error for testing set multiple regression is ", MSE)
print(f"RMSE for testing set using multiple regression is :{np.sqrt(MSE)} ")


print(f"The mean of forecast of multiple regression is {np.mean(testing_error_regression)}")
print(f"The variance of foreacast of multiple regression is {np.var(testing_error_regression)}")

print(f"\n The ratio of resid vs forecast is {np.var(training_residual)/np.var(testing_error_regression)}")


#plt.plot(Y_train, label = 'Training set')
plt.plot(Y_test, label = 'Test set')
plt.plot(prediction_test, label = ' one-step prediction using multiple regression method')
plt.xlabel("Date (year)", fontsize= 14)
plt.ylabel("Closing price", fontsize =14)
plt.title("Plot of Amzn stock price using regression method", fontsize =14)
plt.legend()
plt.show()

#14. Base models

#Averge method

y_predict_train_set = []
value = 0
for i in range(len(Y_train)):
  if i != 0:
    value = value + Y_train[i - 1]
    t_value  = i
    y_each_predict = value / i
    y_predict_train_set.append(y_each_predict)
  else:
    continue

#print(y_predict_train_set)



y_predict_test_set= []
for i in range(len(Y_test)):
  y_predict_each = sum(Y_train) / len(Y_train)
  y_predict_test_set.append(y_predict_each)

y_preidction_average= pd.DataFrame(y_predict_test_set).set_index(Y_test.index)

#print(y_predict_test_set)

plt.plot(Y_train, label = 'Training set')
```

```
plt.plot(Y_test, label = 'Test set')
plt.plot(y_preidction_average, label = 'forecast using average method')
plt.xlabel("Date (year)", fontsize= 14)
plt.ylabel("Closing price", fontsize =14)
plt.title("Plot of Amzn stock price using average method", fontsize =14)
plt.legend()
plt.show()

#now lets find out the MSE of our prediction error --on training set using average method

error_train_set_avg = np.subtract(Y_train[1:], y_predict_train_set)
#print(error_train_set_avg)

def calc_MSE(x):
   MSE = np.square(np.array(x)).mean()
   return MSE


MSE_train_set =calc_MSE(error_train_set_avg)
#MSE_train_set = np.sum((var_square_train_set_array) ** 2 )/ len(t_train_set)
print(f"\nMSE of prediction error (training set) using average method is : {MSE_train_set}")

#calling auto-corelation function
list_acf_residauls_average, np_acf_calc_residuals_average = calc_auto_corelation(error_train_set_avg,
np.mean(error_train_set_avg), n=20, title='Plot of ACF of residuals using average method')

#calling function to calculate Q values

Q_residual  = (len(error_train_set_avg)) *(np.sum(np.square(np_acf_calc_residuals_average)[21:]))

print(f"The Q value of residual using average method is {Q_residual}")



print(f"The mean of residuals using average method is {np.mean(error_train_set_avg)}")
print(f"The variaince of residual using average method is {np.var(error_train_set_avg)}")

error_test_set_avg = np.subtract(Y_test, y_predict_test_set)
MSE_train_set =calc_MSE(error_test_set_avg)
print(f"\nMSE of forecast (testing set) using average method is : {MSE_train_set}")
print(f"Mean of forecast error is: {np.mean(np.array(error_test_set_avg))}")
print(f"Variance of forecast error is: {np.var(np.array(error_test_set_avg))}")


print(f"\n The ratio of resid vs forecast of average method is {( np.var(error_train_set_avg) ) / (
np.var(np.array(error_test_set_avg))  )}")



#Naive method

print("****  Naive Method   ****")
y_predict_train_set_naive = []
value = 0
for i in range(len(Y_train[1:])):
```

```
    y_predict_train_set_naive.append(Y_train[i])

#print(y_predict_train_set_naive)

y_predict_test_set_naive= [Y_train[-1] for i in Y_test]

y_prediction_naive_test= pd.DataFrame(y_predict_test_set_naive).set_index(Y_test.index)

#print(y_predict_test_set_naive)

plt.plot(Y_train, label = 'Training set')
plt.plot(Y_test, label = 'Test set')
plt.plot(y_prediction_naive_test, label = ' forecast using naive method')
plt.xlabel("Date (Year)", fontsize= 14)
plt.ylabel("Closing Price", fontsize =14)
plt.title("Plot of Amzn stock price using naive method", fontsize =14)
plt.legend()
plt.show()

error_train_set_naive = np.subtract(Y_train[1:], y_predict_train_set_naive)

error_test_set_naive = np.subtract(Y_test, y_predict_test_set_naive)

MSE_train_set_naive =calc_MSE(error_train_set_naive)
print(f"\nMSE of prediction error (training set) using naive method is : {MSE_train_set_naive}")



#calling auto-corelation function
list_acf_residauls_naive, np_acf_calc_residuals_naive = calc_auto_corelation(error_train_set_naive,
np.mean(error_train_set_naive), n=20, title='Plot of ACF of residuals using naive method')

#calling function to calculate Q values

Q_residual  = (len(error_train_set_naive)) *(np.sum(np.square(np_acf_calc_residuals_naive)[21:]))
print(f"The Q value of residual using naive method is {Q_residual}")



print(f"The mean of residuals using naive method is {np.mean(error_train_set_naive)}")
print(f"The variance of residual using naive method is {np.var(error_train_set_naive)}")



MSE_test_set_naive =calc_MSE(error_test_set_naive)
print(f"\nMSE of prediction error (testing set) using naive method is : {MSE_test_set_naive}")

print(f"Mean of forecast error using naive method is: {np.mean(np.array(error_test_set_naive))}")
print(f"Variance of forecast error using naive method is: {np.var(np.array(error_test_set_naive))}")


print(f"\n The ratio of resid vs forecast of naive method is {( np.var(error_train_set_naive) ) / (
np.var(np.array(error_test_set_naive)) )}")
```

```
#Drift method

print("***** Drift Method   ********")

y_predict_train_set_drift = []
value = 0
for i in range(len(Y_train)):
   if i > 1:
      slope_val = (Y_train[i - 1] - Y_train[0]) / (i-1)
      y_each_predict = (slope_val * i) + Y_train[0]
      y_predict_train_set_drift.append(y_each_predict)
   else:
      continue

y_predict_test_set_drift= []
for h in range(len(Y_test)):
   slope_val = (Y_train[-1] - Y_train[0] ) /( len(Y_train) - 1 )
   y_predict_each = Y_train[-1] + ((h +1) * slope_val)
   y_predict_test_set_drift.append(y_predict_each)

#print(y_predict_test_set_drift)

y_preidction_drift= pd.DataFrame(y_predict_test_set_drift).set_index(Y_test.index)

plt.plot(Y_train, label = 'Training set')
plt.plot(Y_test, label = 'Test set')
plt.plot(y_preidction_drift, label = 'forecast using drift method')
plt.xlabel("Date (Year)", fontsize= 14)
plt.ylabel("Closing Price", fontsize =14)
plt.title("Plot of Amzn stock price using drift method", fontsize =14)
plt.legend()
plt.show()

error_train_set_drift = np.subtract(Y_train[2:], y_predict_train_set_drift)

error_test_set_drift = np.subtract(Y_test, y_predict_test_set_drift)

MSE_train_set_drift =calc_MSE(error_train_set_drift)
print(f"\nMSE of prediction error (training set) using drift method is : {MSE_train_set_drift}")



#calling auto-corelation function
list_acf_residauls_drift, np_acf_calc_residuals_drift = calc_auto_corelation(error_train_set_drift,
np.mean(error_train_set_drift), n=20, title='Plot of ACF of residuals using drift method')

#calling function to calculate Q values

Q_residual  = (len(error_train_set_drift)) *(np.sum(np.square(np_acf_calc_residuals_drift)[21:]))

print(f"The Q value of residual using drift method is {Q_residual}")
```

```
print(f"The mean of residuals using drift method is {np.mean(error_train_set_drift)}")
print(f"The variance of residual using drift method is {np.var(error_train_set_drift)}")


MSE_test_set_drift =calc_MSE(error_test_set_drift)
print(f"\nMSE of prediction error of testing set using drift method is : {MSE_test_set_drift}")



print(f"Mean of forecast error using drift method is: {np.mean(np.array(error_test_set_drift))}")
print(f"Variance of forecast error using drift method is: {np.var(np.array(error_test_set_drift))}")


print(f"\n The ratio of resid vs forecast of drift method is {( np.var(error_train_set_drift) ) / (
np.var(np.array(error_test_set_drift)) )}")

#Simple and exponential smoothing

print("*******  SES Method *****")

SES = ets.ExponentialSmoothing(Y_train,trend=None,damped=False,seasonal=None).fit()
SES_predict_train= SES.forecast(steps=len(Y_train))
SES_predict_test= SES.forecast(steps=len(Y_test))
predict_test_SES = pd.DataFrame(SES_predict_test).set_index(Y_test.index)

resid_SES = np.subtract(Y_train.values,np.array(SES_predict_train))
forecast_error_Ses = np.subtract(Y_test.values,np.array(SES_predict_test))

MSE_SES = np.square(resid_SES).mean()
print("Mean square error for (training set) simple exponential smoothing is ", MSE_SES)

plt.plot(Y_train,label= "Amazon-train")
plt.plot(Y_test,label= "Amazon-test")
plt.plot(predict_test_SES,label= "SES Method prediction")
plt.legend(loc='upper left')
plt.title('SES method')
plt.xlabel('Time (daily)')
plt.ylabel('Closing Price')
plt.show()


#calling auto-corelation function
list_acf_residauls_SES, np_acf_calc_residuals_SES = calc_auto_corelation(resid_SES, np.mean(resid_SES), n=20,
title='Plot of ACF of residuals using SES method')

#calling function to calculate Q values

Q_residual  = (len(resid_SES)) *(np.sum(np.square(np_acf_calc_residuals_SES)[21:]))

print(f"The Q value of residual using SES method is {Q_residual}")


print(f"Mean of residual using SES method is: {np.mean(np.array(resid_SES))}")
print(f"Variance of residual using SES method is: {np.var(np.array(resid_SES))}")
```

```
MSE_SES = np.square(forecast_error_Ses).mean()
print("Mean square error for (testing set) simple exponential smoothing is ", MSE_SES)


print(f"Mean of forecast error using SES method is: {np.mean(np.array(forecast_error_Ses))}")
print(f"Variance of forecast error using SES method is: {np.var(np.array(forecast_error_Ses))}")

print(f"\n The ratio of resid vs forecast of SES method is {( np.var(np.array(resid_SES)) ) / (
np.var(np.array(forecast_error_Ses)) )}")



def auto_corelation(x, bar_x, n):
    list_ry =[]
    for i in range(n):
        total_numerator = 0
        total_dinom = 0
        a = i
        for t in range(len(x)):
            if a < len(x):
                each_sum = (x[a] - bar_x) * (x[t] -bar_x)
                total_numerator = each_sum + total_numerator
                each_dinom = (x[t] - bar_x) ** 2
                total_dinom = each_dinom + total_dinom
                a = a +1
            else:
                each_dinom = (x[t] - bar_x) ** 2
                total_dinom = each_dinom + total_dinom
        value = total_numerator / total_dinom
        list_ry.append(value)

    inverse_generated_list_ry = list_ry[::-1]
    Generated_list_Ry = inverse_generated_list_ry[:-1] + list_ry
    Generated_list_Ry_np = np.array(Generated_list_Ry)
    #return list_ry, Generated_list_Ry_np

    return Generated_list_Ry_np



AR_ACF_array = auto_corelation(Amzn_diff1, np.mean(Amzn_diff1), 20)


lags =20
Ry = AR_ACF_array

####====================
# ======= calculating GPAC table
##======================
def calc_Gpac(na_order, nb_order, Ry):
    x = int((len(Ry) - 1) / 2)
    df = pd.DataFrame(np.zeros((na_order, nb_order + 1)))
    df = df.drop(0, axis=1)
```

```
    for k in df:  # this for loop iterates over the column to calculate the value
        for j, row_val in df.iterrows():  # this iterates over the rows
            if k == 1:  # for first column
                dinom_val = Ry[x + j]  # Here Ry(0) = lags -1   = x
                numer_val = Ry[x + j + k]
            else:  # when our column is 2 or more than 2; when k > 2
                dinom_matrix = []
                for rows in range(k):  # this loop is for calculating the square matrix (iterating over the rows of matrix)
                    # print(rows)
                    row_list = []
                    for col in range(k):  # this loop is for calculating the square matrix (iterating over the columns of
matrix)
                        # print(col)
                        each = Ry[x - col + rows + j]
                        # print(each)
                        row_list.append(each)
                    dinom_matrix.append(np.array(row_list))

                # dinominator matrix and numerator matrix have same values except for the last column so:
                dinomator_matrix = np.array(dinom_matrix)
                numerator_matrix = np.array(dinom_matrix)

                # updating values for last column of numerator matrix

                last_col =k
                for r in range(k):
                    numerator_matrix[r][last_col - 1] = Ry[x + r + 1 + j]

                # calculating determinants
                numer_val = np.linalg.det(numerator_matrix)
                dinom_val = np.linalg.det(dinomator_matrix)

            df[k][j] = numer_val / dinom_val  # plugs the value in GPAC table

    print(df)

    import seaborn as sns
    sns.heatmap(df, cmap=sns.diverging_palette(20, 220, n=200), annot=True, center=0)
    plt.title('Generalized Partial Auto-correlation Table')
    plt.show()


na_order =8
nb_order =8
calc_Gpac(na_order, nb_order, Ry)

y =Amzn_df['Close']
#mean_y = y.mean()
#y = y - y.mean()

Y_train, Y_test = train_test_split(y, test_size = 0.2, shuffle=False)

def calc_theta(y,na,nb, theta):
    if na == 0:
        dinominator = [1]
```

```
    else:
        dinominator =np.append([1], theta[:na])

    if nb ==0:
        numerator = [1]
    else:
        numerator = np.append([1], theta[-nb:])

    diff = na -nb
    if diff > 0:
        numerator =np.append(numerator, np.zeros(diff))


    sys = (dinominator, numerator, 1)
    _,e = signal.dlsim(sys, y)
    theta =[]
    for i in e:
        theta.append(i[0])

    theta_e =np.array(theta)

    return theta_e

def step_1(y, na, nb, theta, delta):
    e = calc_theta(y,na,nb, theta)
    SSE = np.dot(e, e.T)

    X=[]

    n =na +nb

    for i in range(n):
        theta_new =theta.copy()
        theta_new[i] =theta_new[i] +delta
        new_e = calc_theta(y, na, nb, theta_new)
        x_i = np.subtract(e, new_e)/delta
        X.append(x_i)


    X =  np.transpose(X)
    A =  np.transpose(X).dot(X)
    g=  np.transpose(X).dot(e)

    return A,g, SSE

def step_2(theta, A, g, u, y):
    n = na +nb
    idt = np.identity(n)
    before_inv= A + (u * idt)
    AUI_inv = np.linalg.inv(before_inv)
    diff_theta= AUI_inv.dot(g)
    theta_new = theta +diff_theta

    new_e = calc_theta(y, na, nb, theta_new)
    SSE_new = new_e.dot(new_e.T)
```

```
    return SSE_new, theta_new, diff_theta

def calc_LMA(count, y, na, nb, theta, delta, u, u_max):
    i =0
    SSE_count=[]
    norm_theta=[]

    while i <count:
        A,g,SSE = step_1(y,na,nb,theta, delta)
        SSE_new, theta_new, diff_theta =step_2(theta, A, g,u, y)
        SSE_count.append(SSE_new)

        n =na+nb

        if SSE_new < SSE:
            norm_theta2  = np.linalg.norm(np.array(diff_theta),2)
            norm_theta.append(norm_theta2)

            if norm_theta2 < 0.001:
                theta = theta_new.copy()
                break

            else:
                theta =theta_new.copy()
                u = u/10

        while SSE_new >= SSE:
            u = u *10
            if u>u_max:
                print("Mue is high now and cannot go higher than that!!!")
                break
            SSE_new, theta_new, diff_theta = step_2(theta, A, g, u, y)
        theta  = theta_new
        i += 1

    variance_error = SSE_new / (len(y) - n)
    co_variance =variance_error * np.linalg.inv(A)
    print("The estimated parameters >>> ", theta)
    print(f"\n The estimated co-variance matrix is {co_variance}")
    print(f"\n The estimated variance of error is {variance_error}")


    for i in range(na):
        std_deviation = np.sqrt(co_variance[i][i])
        print(f'The standard deviation for a{i+1} is {std_deviation}")

    for j in range(na, n):
        std_deviation = np.sqrt(co_variance[j][j])
        print(f'The standard deviation for b{i + 1} is {std_deviation}")

    print(f"\n The confidence interval for parameters are: ")
    for i in range(na):
        interval = 2 * np.sqrt(co_variance[i][i])
        print(f'{(theta[i]- interval)} < a{i+1} < {(theta[i] + interval)}")
```

```
    for j in range(na,n):
        interval = 2 * np.sqrt(co_variance[j][j])
        print(f"{(theta[j]- interval)} < b{j -na + 1} < {(theta[j] + interval)}")

    #zero/pole

    num_root =[1]
    den_root= [1]

    for i in range(na):
        num_root.append(theta[i])
    for i in range(nb):
        den_root.append(theta[i + na])

    poles =np.roots(num_root)
    zeros = np.roots(den_root)

    print(f"\nThe roots of the numerators are {zeros}")
    print(f"The roots of dinominators are {poles}")

    #plt.plot(SSE_count)
    #plt.xlabel("Numbers of Iternations")
    #plt.ylabel("Sum square Error")
    #plt.title("Sum of square Error vs the iterations")
    #plt.show()

    return theta,co_variance


delta = 10**-6
na =2
nb =0
n = na +nb
theta = np.zeros(n)
u = 0.01
u_max = 1e10
count = 60

print("\nFor our estimated ARMA (2,0): ")

theta, cov = calc_LMA(count, Amzn_df['Close'], na,nb, theta, delta, u, u_max)
Y_train , Y_test = train_test_split(Amzn_df['Close'], test_size= 0.2, shuffle =False)

#y_prediction = one_step_pred(theta,na,nb,Y_train)
#y_prediction = one_step_pred(theta,na,nb,Y_train)    -0.9421188128435487
y_predict=[]
for i in range(len(Y_train)):
    if i ==0:
        predict = (-theta[0]) * Y_train[i]
    else:
        predict = ( - theta[0] * Y_train[i] ) + ( -theta[1] * Y_train[i-1])
    y_predict.append(predict)

resid = np.subtract(np.array(Y_train),np.array(y_predict))
acf_resid = auto_corelation(resid, np.mean(resid), 20)
```

```
Q  = (len(resid)) *(np.sum(np.square(acf_resid)[21:]))
DOF = len(resid) - na -nb
alfa =0.01
chi_critical = chi2.ppf(1-alfa, DOF)
print("\nThe Q value is  ",Q)

print(f"The chi-critical is {chi_critical}")


if Q <chi_critical:
   print(f"\n The Q value is less than {chi_critical} (chi-critical) so, the residual is white")
else:
   print(f"\nThe Q value is not less than {chi_critical} (chi-critical) so, the residual is not white")

print(resid[:5])

print(f"Mean of residual with ARMA(2,0) is {np.mean(resid)}")
print(f"The variance of residual with ARMA(2,0) is {np.var(resid)}")


y_predict = pd.DataFrame(y_predict).set_index(Y_train.index)

plt.plot(Y_train, label='Y_train')
plt.plot(y_predict, label ='Predicted values')
plt.xlabel('Number of observations')
plt.ylabel('y-values')
plt.title("One-step-ahead prediction")
plt.legend()
plt.show()


print("\nFor our estimated ARMA (3,0): ")
#delta = 10**-6
na =3
nb =0
n = na +nb
theta = np.zeros(n)
#u = 0.01
#u_max = 1e10
#count = 60


theta, cov = calc_LMA(count, Amzn_df['Close'], na,nb, theta, delta, u, u_max)
Y_train , Y_test = train_test_split(Amzn_df['Close'], test_size= 0.2, shuffle =False)


print(f"The estimated parameters are {theta[0]} and {theta[1]}")
na  = 2
print("So lets do one-step prediction accordingly.")
#y_prediction = one_step_pred(theta,na,nb,Y_train)
y_predict=[]
for i in range(len(Y_train)):
   if i ==0:
```

```
        predict = (-theta[0]) * Y_train[i]
    else:
        predict = ( - theta[0] * Y_train[i] ) + ( -theta[1] * Y_train[i-1])
    y_predict.append(predict)

resid = np.subtract(np.array(Y_train),np.array(y_predict))
acf_resid = auto_corelation(resid, np.mean(resid), 20)

print(resid[:5])


Q  = (len(resid)) *np.sum(np.square(acf_resid)[21:])
DOF = len(resid) - na -nb
alfa =0.01
chi_critical = chi2.ppf(1-alfa, DOF)
print("\nThe Q value is  ",Q)

print(f"The chi-critical is {chi_critical}")


if Q <chi_critical:
    print(f"\n The Q value is less than {chi_critical} (chi-critical) so, the residual is white")
else:
    print(f"\nThe Q value is not less than {chi_critical} (chi-critical) so, the residual is not white")

print(f"MSE of residual for ARMA(2,0) is {np.square(resid).mean()}")
print(f"\nMean of residual with ARMA(2,0) is {np.mean(resid)}")
print(f"The variance of residual with ARMA(2,0) is {np.var(resid)}")

y_predict = pd.DataFrame(y_predict).set_index(Y_train.index)

plt.plot(Y_train, label='Y_train')
plt.plot(y_predict, label ='Predicted values')
plt.xlabel('Number of observations')
plt.ylabel('y-values')
plt.title("One-step-ahead prediction")
plt.legend()
plt.show()


# prediction for test set
y_prediction_test=[]
for i in range(len(Y_test)):
    if i ==0:
        predict = (-theta[0] * Y_train[-1])+ (-theta[1] * Y_train[-2])
    elif i ==1:
        predict = ( -theta[0] * y_prediction_test[0] ) + (-theta[1] * Y_train[-1])
    elif i ==2:
        predict = ( -theta[0] * y_prediction_test[1] ) + (-theta[1] * y_prediction_test[0])
    else:
        predict = (-theta[0] * y_prediction_test[i - 1]) + (-theta[1] * y_prediction_test[i-2])
    y_prediction_test.append(predict)

y_prediction_test = pd.DataFrame(y_prediction_test).set_index(Y_test.index)
```

```
forecast_error = np.subtract(np.array(Y_test), np.array(y_prediction_test))

print(f"\nMSE of forecast for ARMA(2,0) is {np.square(forecast_error).mean()}")
print(f"The mean of trainings set error is {np.mean(forecast_error)}")
print(f"The variance of training set error is {np.var(forecast_error)}")

ratio = np.var(resid)/np.var(forecast_error)

print(f"\nThe ratio of variance of residual to variance of forecast is {ratio}")

plt.plot(Y_train, label='Training set')
plt.plot(Y_test, label = 'Testing set')
plt.plot(y_prediction_test, label ='Prediction Test set')
plt.xlabel("Date (Year)", fontsize= 14)
plt.ylabel("Closing Price", fontsize =14)
plt.title("Plot of Amzn stock price using ARMA(2,0) ", fontsize =14)
plt.legend()
plt.show()

#==================ARIMA MODEL ==================================

delta = 10**-6
na =2
nb =0
n = na +nb
theta = np.zeros(n)
u = 0.01
u_max = 1e10
count = 60

print(f"For our ARIMA(2,0)")

theta, cov = calc_LMA(count, Amzn_diff1, na,nb, theta, delta, u, u_max)



# this amazon diff is in different domain so we make ckomparison with the same domain so lets divid Amazon
diff1 to training and testing
thet = 0.04469821 # we only got one parameter from LM
na = 1
nb =0

print(f"The estimated parameter after removing the statistically not significant parameter is {thet}")

# ========== Writing the model==================
#  ARIMA (1,1,0)   ---- (1 + 0.04469821 q **-1) (1- q**-1 ) y(t)  = e(t)
#             ---- y(t) - q **-1 y(t) + 0.04469821 q **-1 y(t)  - 0.04469821 q **-2 y(t)  = e(t)
#                 y(t) = 0.95530179 q **-1 y(t) - 0.04469821 q **-2 y(t) + e(t)
#               y_hat_(t) = 0.95530179 y(t-1) - 0.04469821 y(t-2) + e(t)

Y_train, Y_test = train_test_split(Amzn_df['Close'], test_size=0.2, shuffle =False)

diff_Y_train, diff_Y_test = train_test_split(Amzn_diff1, test_size = 0.2, shuffle=False)

#y_prediction_diff = one_step_pred(thet ,na,nb,Y_train)
```

```
#======= One-step prediction ========================

y_predict_hat=[]
for i in range(len(Y_train)):
   if i ==0:
      predict = 0.95530179 * Y_train[i]
   else:
      predict = ( 0.95530179 * Y_train[i] ) - (0.04469821 * Y_train[i-1])
   y_predict_hat.append(predict)


# Residual calculation


def calc_MSE(x):
   MSE = np.square(np.array(x)).mean()
   return MSE

resid = np.subtract(Y_train,y_predict_hat)
#resid = np.subtract(diff_Y_train,y_predict_diff)\
MSE_resid = calc_MSE(resid)

print(f"MSE of resid of ARIMA(1,1,0) is {MSE_resid}")
acf_resid = auto_corelation(resid, np.mean(resid), 20)


Q  = (len(resid)) *np.sum(np.square(acf_resid)[21:])
DOF = len(resid)- na -nb
alfa =0.01
chi_critical = chi2.ppf(1-alfa, DOF)
print("\nThe Q value is  ",Q)


print(f"The chi-critical is {chi_critical}")


if Q <chi_critical:
   print(f"\n The Q value is less than {chi_critical} (chi-critical) so, the residual is white")
else:
   print(f"\nThe Q value is not less than {chi_critical} (chi-critical) so, the residual is not white")

print(f"Mean of residual with ARIMA(1,1,0) is {np.mean(resid)}")
print(f"The variance of residual with ARIMA(1,1,0) is {np.var(resid)}")


#========= One-step prediction plot ===================
#y_predict_diff = pd.DataFrame(y_predict_diff).set_index(diff_Y_train.index)
y_predict_hat = pd.DataFrame(y_predict_hat).set_index(Y_train.index)

plt.plot(Y_train, label='Original Y_train')
plt.plot(y_predict_hat, label ='Predicted values')
plt.xlabel('Time (Date-Month)')
plt.ylabel('Closing price ($)')
plt.title("One-step-ahead prediction for original plot using ")
```

```
plt.legend()
plt.show()

# Here back transformation is just the original dataset
#==========Back transformation

diff_transform=[]
for i in range(len(log_transformed_data)):
  if i ==0:
    val =log_transformed_data[i]
  else:
    val = Amzn_diff1[i-1] + log_transformed_data[i -1]

  diff_transform.append(val)

anti_log_transformation =[]
for k in diff_transform:
  anti_log_transformation.append(np.exp(k))

#  y_hat_(t) = 0.95530179 y(t-1) - 0.04469821 y(t-2) + e(t)
def forecast_func(thet, Y_train,step):

  y_prediction_test=[]
  for i in range(step):
    if i ==0:
      predict = (thet[0] * Y_train[-1])- (thet[1] * Y_train[-2])
    elif i ==1:
      predict = ( thet[0] * y_prediction_test[0] ) - (thet[1] * Y_train[-1])
    elif i ==2:
      predict = ( thet[0] * y_prediction_test[1] ) - (thet[1] * y_prediction_test[0])
    else:
      predict = (thet[0] * y_prediction_test[i - 1]) - (thet[1] * y_prediction_test[i-2])
    y_prediction_test.append(predict)

  return y_prediction_test

thet = [0.95530179 , 0.04469821]

y_prediction_test = forecast_func(thet, Y_train, step =len(Y_test))


y_prediction_test = pd.DataFrame(y_prediction_test).set_index(Y_test.index)

forecast_error = np.subtract(np.array(Y_test), np.array(y_prediction_test))

MSE_testing_error = calc_MSE(forecast_error)
print(f"MSE of prediction for testing error is {MSE_testing_error}")
print(f"\nThe mean of error of testing ARIMA is{np.mean(forecast_error)}")
print(f"The variance of forecast error is {np.var(forecast_error)}")
print(f"\nThe variance of forecast error is {np.var(forecast_error)}")

ratio = np.var(resid)/np.var(forecast_error)

print(f"The ratio of variance of residual to variance of forecast is {ratio}")
```

```
#=========================================
#========== Forecast function ============
#=====================================

print("****  Drift Method   ****")

def forecast_function(Y_train, step):
   y_predict_test_set_drift= []
   for h in range(step):
      slope_val = (Y_train[-1] - Y_train[0] ) /( len(Y_train) - 1 )
      y_predict_each = Y_train[-1] + ((h +1) * slope_val)
      y_predict_test_set_drift.append(y_predict_each)

   return y_predict_test_set_drift

step = len(Y_test)
y_predict_test_set_drift = forecast_function(Y_train, step)

y_predict_test_set_drift = pd.DataFrame(y_predict_test_set_drift).set_index(Y_test.index)



#print(y_predict_test_set_naive)

plt.plot(Y_train, label = 'Training set')
plt.plot(Y_test, label = 'Test set')
plt.plot(y_predict_test_set_drift, label = ' forecast using drift method')
plt.xlabel("Date (Year)", fontsize= 14)
plt.ylabel("Closing Price", fontsize =14)
plt.title("Plot of Amzn stock price using drift method", fontsize =14)
plt.legend()
plt.show()

#============ H-step prediction ======================================


step =100
y_predict_h_step_drift = forecast_function(Y_train, step)

y_predict_h_step_drift= pd.DataFrame(y_predict_h_step_drift).set_index(Y_test[:100].index)

plt.plot(Y_test[:100], label = 'Test set')
plt.plot(y_predict_h_step_drift, label = ' 100 -step forecast using naive method')
plt.xlabel("Date (Year)", fontsize= 14)
plt.ylabel("Closing Price", fontsize =14)
plt.title("Plot of H-step prediction", fontsize =14)
plt.legend()
plt.show()
```

References

George Washington University Logo Image - Image-Link
https://www.publichealth.columbia.edu/research/population-health-methods/time-event-data-analysis

Jafari, R. (2021). *Time Series Modeling & Analysis: DATS 6450 Lecture* [Powerpoint slides].