DATS 6313

Final Term Project Report

Predicting Ethereum Close Price

Date:05/04/2022

Professor: Reza Jafari

Subject: Time Series Analysis and Modelling

Sagar Tripathi(G30209731)

The George Washington University

Table of Contents

Table of Contents

Table of figures and tables

Abstract:

In this research, we attempt to accurately anticipate the Ethereum price by considering a variety of factors that influence its value. We want to understand and uncover Hourly trends in the Ethereum market in the first phase of our inquiry, as well as acquire insight into the best characteristics surrounding Ethereum price. Our data collection includes hourly records of several variables related to the Ethereum price during a five-year period. Using the information, we have; we will anticipate the sign of the hourly closing price change with the greatest precision feasible in the second phase of our inquiry.

Introduction:

What exactly is Ethereum?

Ethereum is a decentralized, open source blockchain that allows users to create smart contracts. The platform's native cryptocurrency is Ether (ETH or). It is the second-largest cryptocurrency by market capitalization after Bitcoin. The Ethereum blockchain is the most widely utilized. Vitalik Buterin, a programmer, proposed Ethereum in 2013. The Data set used in this project is ETH_1H.csv contains all the 1-hour historical data from 2020-04-16 00:00:00 - 2016-05-09 13:00:00, this dataset contains 7 columns including Unix Timestamp, dates, opening, high, low, closing, and volume.

To complete the project's objectives, we must first clean the data, As the data didn't have any null values or missing values, we still have performed data cleaning to ensure that there should be no null or missing values which includes replacing missing values with the column's mean. We then use ACF plots, ADF, and KPSS tests to determine whether the target variable is stationary, and if it is not, we apply log transformation and differentiation to make it stationary. To model and predict test values, we use base models such as the average, naïve, drift, SES, holt linear, and Holt winter approaches. These baseline models serve as a benchmark against which the ARMA/ ARIMA models can be measured in order to determine how much better they are. For feature reduction, we also employ the OLS regression approach. For the ARMA/ ARIMA models, we use the GPAC table to find the na, nb values. The ARMA/ARIMA models are subjected to diagnostic tests. in order to see if the model is accurate enough to forecast. The best model is then chosen based on the mean, variance of residuals, and prediction error, and the forecast values are plotted alongside the test values to illustrate the difference

Theory:

The functions and theories that we developed during our classes are listed below.

1. **Pearson correlation coefficient:**
   [Source: https://www.sciencedirect.com/topics/computer-science/pearson-correlation]
   The Pearson correlation method is the most widely used approach for numerical variables; it assigns a number between 0 and 1, with 0 representing no correlation, 1 representing total positive correlation, and 1 representing total negative correlation. This means that a correlation value of 0.7 between two variables indicates that there is a significant and positive association between them. A positive correlation indicates that if variable A rises, so will variable B, whereas a negative correlation indicates that if A rises, so will B.

Formula

$$r = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}}$$

2. **Auto-Correlation:**
   [Source: https://www.investopedia.com/terms/a/autocorrelation.asp]
   The degree of similarity between a particular time series and a lagged version of itself over subsequent time intervals is represented mathematically by autocorrelation. Autocorrelation is similar to correlation between two different time series in that it uses the same time series twice: once in its original form and again with one or more time periods added.

$$\hat{R}_y(\tau) = \frac{\sum_{t=\tau+1}^{T} (y_t - \overline{y})(y_{t-\tau} - \overline{y})}{\sum_{t=1}^{T} (y_t - \overline{y})^2}$$

3. **Partial Auto Correlation:**
   [source: DATS 6313 time series analysis and Modeling lecture notes]
   partial correlation is a conditional correlation. It is the correlation between two variables under the assumption that we know and consider the values of some other set of variables. For instance, consider a regression context in which *y* is the response variable and x1, x2, and x3 are predictor variables. The partial correlation between *y* and x3 is the correlation between the variables determined taking into account how both y and x3 are related to x1 and x2. Formally partial correlation is described as:

$$\frac{\text{Covariance}(y, x_3 | x_1, x_2)}{\sqrt{\text{Variance}(y | x_1, x_2)\,\text{Variance}(x_3 | x_1, x_2)}}$$

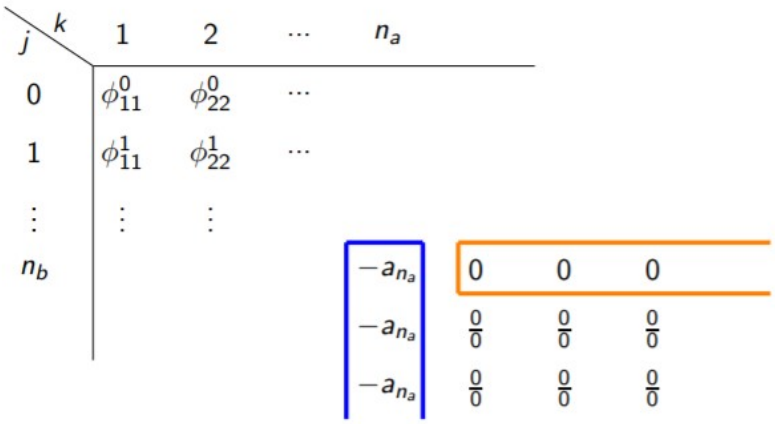
To calculate PAC we derive a set of linear equations which can be written as below:

$$\underbrace{\begin{pmatrix} \hat{R}_y(0) & \hat{R}_y(1) & \dots & \hat{R}_y(n_a-1) \\ \hat{R}_y(1) & \hat{R}_y(0) & \dots & \hat{R}_y(na-2) \\ \vdots & \vdots & \vdots & \vdots \\ \hat{R}_y(n_a-1) & \hat{R}_y(n_a-2) & \dots & \hat{R}_y(0) \end{pmatrix}}_{\hat{X}} \underbrace{\begin{pmatrix} \hat{a}_1 \\ \hat{a}_2 \\ \vdots \\ \hat{a}_{n_a} \end{pmatrix}}_{\hat{a}} = \underbrace{\begin{pmatrix} \hat{R}_y(1) \\ \hat{R}_y(2) \\ \vdots \\ \hat{R}_y(n_a) \end{pmatrix}}_{\hat{Y}}$$

4. **Generalized partial autocorrelation (GPAC):**
   [source: DATS 6313 time series analysis and Modeling lecture notes]
   The GPAC is used to estimate the order of ARMA model when na and nb is not equal to 0. The matrix representation of GPAC table is given below:

$$\phi_{kk}^{j} = \frac{\begin{vmatrix} \hat{R}_y(j) & \hat{R}_y(j-1) & \dots & \hat{R}_y(j+1) \\ \hat{R}_y(j+1) & \hat{R}_y(j) & \dots & \hat{R}_y(j+2) \\ \vdots & \vdots & \vdots & \vdots \\ \hat{R}_y(j+k-1) & \hat{R}_y(j+k-2) & \dots & \hat{R}_y(j+k) \end{vmatrix}}{\begin{vmatrix} \hat{R}_y(j) & \hat{R}_y(j-1) & \dots & \hat{R}_y(j-k+1) \\ \hat{R}_y(j+1) & \hat{R}_y(j) & \dots & \hat{R}_y(j-k+2) \\ \vdots & \vdots & \vdots & \vdots \\ \hat{R}_y(j+k-1) & \hat{R}_y(j+k-2) & \dots & \hat{R}_y(j) \end{vmatrix}}$$

The above table is used as a recipe in the python function to construct GPAC table. To estimate the orders, we should look for a pattern highlighted in blue and orange in the table below:



5. **Average method**:

   [source: DATS 6313 time series analysis and Modeling lecture notes]

   The forecast of all future values is equal to the average of the historical data. The average method treats all the observations with equal importance. It gives equal weights while forecasting.

$$\hat{y}_{T+h|T} = \frac{y_1 + y_2 + \ldots + y_T}{T}$$

6. **Naive method:**

   [source: DATS 6313 time series analysis and Modeling lecture notes]

   In the naive method the future value is equal to the last value. While forecast all the values are equal to the last value of the train data.

   Predicted value = Last Actual value

7. **Drift method:**

   [source: DATS 6313 time series analysis and Modeling lecture notes]

   In drift method the forecast values is equal to the last value plus the average change seen in the data. The forecast values graphs is just extrapolating the line connecting 1st and the last value of the train dataset.

$$\hat{y}_{T+h|T} = y_T + \frac{h}{T-1}\sum_{t=2}^{T}(y_t - y_{t-1}) = y_T + h(\frac{y_T - y_1}{T-1})$$

8. **Simple exponential smoothing (SES) method:**
   [source: DATS 6313 time series analysis and Modeling lecture notes]
   SES method is calculated using weighted averages, where least weight value is given the oldest observation from the current data.

$$\hat{y}_{T+1|T} = \sum_{j=0}^{T-1} \alpha(1-\alpha)^j y_{T-j} + (1-\alpha)^T \ell_0$$

9. **Holt Linear method:**
   [source: DATS 6313 time series analysis and Modeling lecture notes]
   This method involves a forecast equation and two smoothing equations(one for level and one for trend)

$$\begin{aligned} (\textit{Forecast equation}) \quad & \hat{y}_{t+h|t} = \ell_t + hb_t \\ (\textit{Level equation}) \quad & \ell_t = \alpha y_t + (1-\alpha)(\ell_{t-1} + b_{t-1}) \\ (\textit{Trend equation}) \quad & b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)b_{t-1} \end{aligned}$$

   Here, lt is Level, and bt is trend

10. **Holt Winter method:**
    [source: DATS 6313 time series analysis and Modeling lecture notes]
    The Holt winter method uses three smoothing equations for the forecast equation so that it takes in level, trend, and seasonality of the data into account while calculating the forecast values.

$$\hat{y}_{t+h|t} = \ell_t + hb_t + s_{t+h-m(k+1)}$$
$$\ell_t = \alpha y_t + (1-\alpha)(\ell_{t-1} + b_{t-1})$$
$$b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)b_{t-1}$$
$$s_t = \gamma(y_t - \ell_{t-1} - b_{t-1}) + (1-\gamma)s_{t-m}$$

here, lt is level, bt is trend and st is seasonality.

11. Time Series Decomposition:

[source: DATS 6313 time series analysis and Modeling lecture notes]

Time series data often exhibit a variety of patterns, and it is useful to split a time series into these patterns for forecasting such that all underlying patterns are captured. Following are the common time series patterns:

Trend: pattern exists when there is a long-term increase or decrease in the data

Cyclic: pattern exists when data exhibits rise and fall that are not of fixed period

Seasonal: pattern exists when a series is influenced by seasonal factors such as a quarter of the year, seasonal holiday period, month or even day of the week.

If we assume an additive decomposition, then time series can be written as: $y=T+S+R$ .Where y is original time series, T is the trend-cycle component, S is the seasonal component and R is the remaining component.

Alternatively, multiplicative decomposition would be written as $y=T*S*R$.

12. Moving averages:

[source: DATS 6313 time series analysis and Modeling lecture notes]

The classical method of time series decomposition originated in the year 1920 and was widely used until the year 1950. As it still forms the basis of many time series decomposition methods, it is important to understand how it works. The first step in a classical decomposition is moving average to estimate the trend-cycle.

Moving average smoothing:

A moving average of order m can be written as

$$\widehat{T}_t = \frac{1}{m}\sum_{j=-k}^{k} y_{t+j}, \quad where\ m = 2k+1$$

That is the estimate of the trend-cycle at time t is obtained by averaging values of the time series within k periods of t. The average eliminates some of the random variations in the data, leaving a smooth trend-cycle component. This is also called m-MA meaning a moving average of order m for example, suppose one is calculating a moving average of order 5, to estimate trend-cycle. There are 20 data points. Then the first value of 5-MA will be an average of the first 5 data points centered on the corresponding year. There will be no values for either the first two years or the last two years because of a lack of observations on either side. The order of the moving average determines the smoothness of the trend-cycle estimate. Larger order means a smoother curve. The effect of changing orders is shown in the questions section to make moving average symmetric even in case of even moving average, moving average of moving average is used. For example, to make 4-Ma symmetric apply 2 X 4MA This is also called a centered moving average of 4. An even order MA should be followed by an even order MA to make it symmetric. Similarly, an odd order MA should be followed by an odd order MA. Moreover, a 2x m-MA is equivalent to a weighted moving average of order m +1 where all observations take the weight 1/m expect for the first and last terms which take weights 1/ (2m). Choices for the order of the MA will result in trend-cycle estimates being contaminated by the seasonality in the data. For example, to estimate the trend-cycle of monthly data use 2 x 12-MA and to estimate the trend-cycle of daily data with a weekly seasonality use 7-MA

13. STL decomposition:

[source: DATS 6313 time series analysis and Modeling lecture notes]

STL is an acronym for "Seasonal and Trend decomposition using Loess" Loess is a method for estimating nonlinear relationships. The STL method was developed by Cleveland, Cleveland, McRae, & Terpenning (1990)

Advantages:

a.) STL will handle any type of seasonality, not only monthly and quarterly data.

b.) The seasonal component is allowed to change over time and the rate of change can be

controlled by the user

c.) The smoothness of the trend-cycle can also be controlled by the user

d.) It can be robust to outliers so that occasional unusual observations will not affect the estimates of the trend-cycle and seasonal component. They will however affect the remainder component

14. Autoregressive models:

[source: DATS 6313 time series analysis and Modeling lecture notes]

Autoregressive models are remarkably flexible at handling a wide range of different time series patterns. For an AR(1) model:

- When $a_1 = 0$, y_t is equivalent to white noise;
- When $a_1 = 1$ and c=0, y_t is equivalent to a random walk;
- When $a_1 = 1$ and $c \neq 0$, y_t is equivalent to a random walk with drift;
- When $a_1 < 1$, y_t tends to oscillate around the mean

Normally AR model is restricted to stationary data, hence there are some constraints on the values of the parameters:

- For an AR(1) model $|a_1| < 1$
- For an AR(2) model $|a_2| < 1$

As AR models are multiple linear regression so AR coefficients can be approximated using LSE. The coefficients of the AR model can be estimated using previous observations as:

$$\boxed{\hat{\boldsymbol{a}} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{Y}}$$

Where n_a is the order of AR model and the T = #samples – na -1

$$X = \begin{pmatrix} -y(n_a-1) & -y(n_a-2) & \cdots & -y(0) \\ -y(n_a) & -y(n_a-1) & \cdots & -y(1) \\ \vdots & \vdots & \vdots & \vdots \\ -y(n_a+T-1) & -y(n_a+T-2) & \cdots & -y(T) \end{pmatrix}_{(T+1)\times na}$$

$$\boldsymbol{Y} = \begin{pmatrix} y(n_a) \\ y(n_a+1) \\ \vdots \\ y(n_a+T) \end{pmatrix}_{(T+1)\times 1} \qquad \hat{\boldsymbol{a}} = \begin{pmatrix} \hat{a}_1 \\ \hat{a}_2 \\ \vdots \\ \hat{a}_{n_a} \end{pmatrix}_{(n_a)\times 1}$$

Mean of AR models:

Let an AR(na) defined as:

$$y(t) + a_1 y(t-1) + a_2 y(t-2) + \ldots + a_{n_a} y(t-n_a) = \epsilon(t)$$

where $\epsilon(t) \sim (\mu_\epsilon, \sigma_\epsilon^2)$

To find mean, take expectation from both sides as follows:

$$E[y(t)] + a_1 E[y(t-1)] + a_2 E[y(t-2)] + \ldots + a_{n_a} E[y(t-n_a)] = E[\epsilon(t)]$$

As data is stationary [y(t)] = E[y(t+1)] = … = E[y(t-na)] = mean y and E[∈(t)] = mean of ∈ hence

$$\mu_y = \frac{\mu_\epsilon}{1 + \sum_{i=1}^{n_a} a_i}$$

Moving average models:

In MA models, post forecast errors are regressed for prediction. Note that the moving average model is used for forecasting future values, while moving average smoothing is used for estimating the trend-cycle of past values.

It is possible to write any stationary AR(p) model as an MA(infinity) model using repeated substitution. We can demonstrate this for an AR(1) model as:

$$\begin{aligned} y_t &= \phi_1 y_{t-1} + \varepsilon_t \\ &= \phi_1(\phi_1 y_{t-2} + \varepsilon_{t-1}) + \varepsilon_t \\ &= \phi_1^2 y_{t-2} + \phi_1 \varepsilon_{t-1} + \varepsilon_t \\ &= \phi_1^3 y_{t-3} + \phi_1^2 \varepsilon_{t-2} + \phi_1 \varepsilon_{t-1} + \varepsilon_t \end{aligned}$$

Provided -1 < Φ < 1, the value of Φk will get smaller as k gets larger. So eventually we obtain

$$y_t = \varepsilon_t + \phi_1\varepsilon_{t-1} + \phi_1^2\varepsilon_{t-2} + \phi_1^3\varepsilon_{t-3} + \cdots,$$

an MA(infinity) process.

Just like AR models, mean of MA models is given by:

$$\mu_y = \mu_\epsilon(1 + \sum_{i=1}^{n_b} b_i)$$

15. ARIMA:

A generalization of an autoregressive moving average (ARMA) model is an autoregressive integrated moving average (ARIMA) model. Both of these models are used to fit time series data in order to better understand or anticipate future points in the series (forecasting). When data exhibit signs of non-stationarity in the sense of mean, an initial differencing step, which corresponds to the "integrated" element of the model, can be applied one or more times to eliminate the non-stationarity of the mean function, ARIMA models are used.

ARIMA in its broadest sense:

$$\left(1+a_1q^{-1}+\ldots+a_{n_a}q^{-n_a}\right)\left(1-q^{-1}\right)^d y(t) = \left(1+b_1q^{-1}+\ldots+b_{n_b}q^{-n_b}\right)\epsilon(t)$$

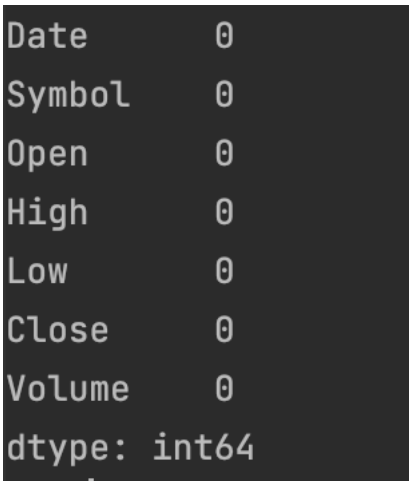
Description of the Dataset:

The Data set used in this project is ETH_1H.csv contains all the 1-hour historical data from 2020-04-16 00:00:00 - 2016-05-09 13:00:00, this dataset contains 7 columns including Unix Timestamp, dates, opening, high, low, closing, and volume and 34497 rows/Observations. We are going to analyze and do the modelling on the Closing price of Ethereum.

Pre-processing dataset & EDA (exploratory data analysis):

The ETH_1H.csv Data has no null values and no missing values.
The target variable is Close.

Missing value check: Figure:1 null values1



We can see that there are zero nan values in all the columns. Hence there is no need of filling the nan values with mean, median, mode or simply with 0.

EDA (Exploratory Data Analysis):

Open, Close, High, low: Price in USD vs time, we can see that between 2017 to 2018 there is a huge raise in open, close and high low in hourly basis of ETH price in USD.

Figure:2 Open, Close, High, low vs time

From the above graph we can see that there is trend and high Seasonality due the peaks in both dependent and independent variable.

Volume: It is the amount of Ethereum sold per hour. We can see in the below graph that volume purchase was high in 2017.

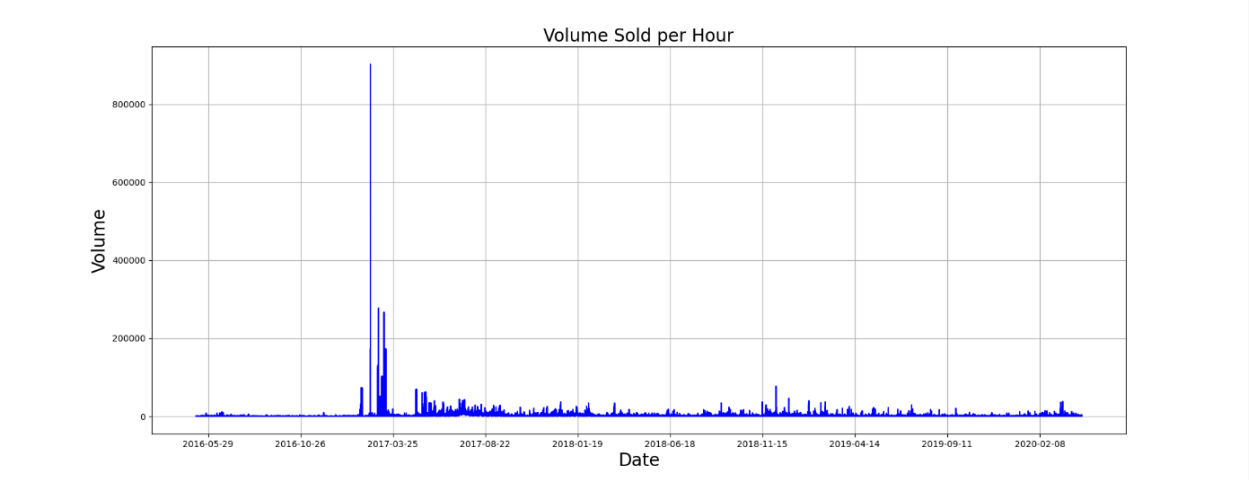


Figure:3 Volume vs time

Market Capitalization: The total dollar market value of a company's outstanding shares of stock is referred to as market capitalization. It is computed by multiplying the entire number of a

company's outstanding shares by the current market price of one share, which is commonly referred to as "market cap." We can observe that the market cap was highest in 2018.
Market Capitalization = Open price * Volume

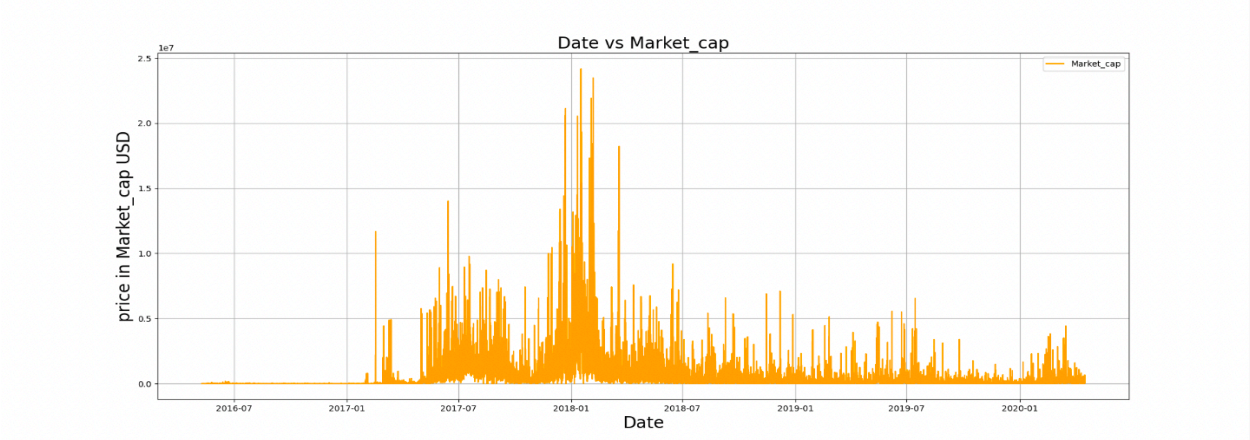


Figure:4 Market cap vs time

Volatility: In order to know the volatility of the stock, we find daily percentage change in the closing price of the stock. We can observe that volatility is max 0.3 and min is 0.2.

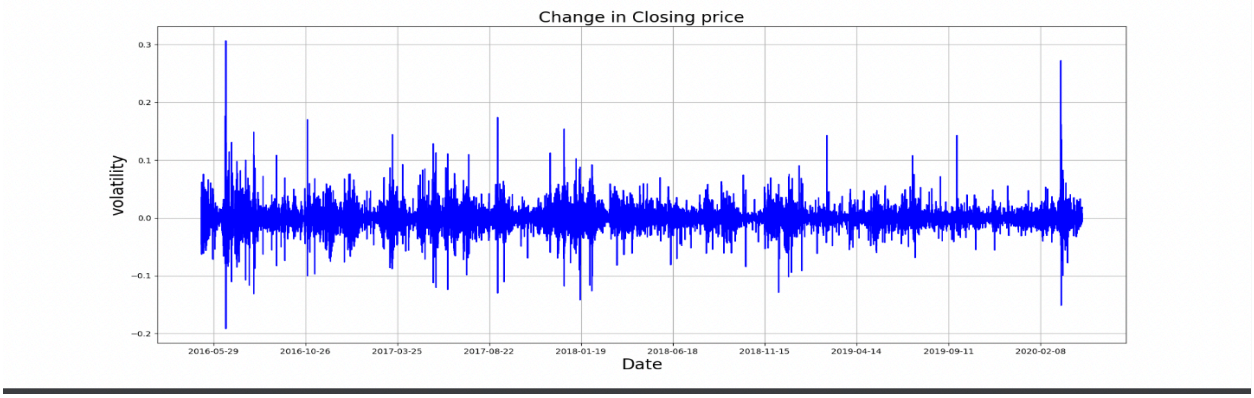


Figure:5 Volatility vs time

Cumulative return: A cumulative return on an investment is the total amount gained or lost by the investment throughout time, regardless of the length of time involved. We can see that there are high cumulative return for who purchased ETH before 2018-03-25.

Figure:6 Cumulative Return vs time

Plot of the dependent variable versus time:

Close: It is the closing price of ETH. From the below graph we can see that there is trend and high Seasonality due the peaks in both dependent and independent variable. Also, we can observe that there the close value started accelerating in 2017, reached to its peak in 2018-01and then started decomposing after 2018-01.

Figure:7 Close (Dependent variable) vs time

Rolling mean and rolling Variance of dependent variable:

we can observe from the below graph that the trend is negative, and dataset is not stationary as the rolling mean and rolling variance are varying with respect to time.

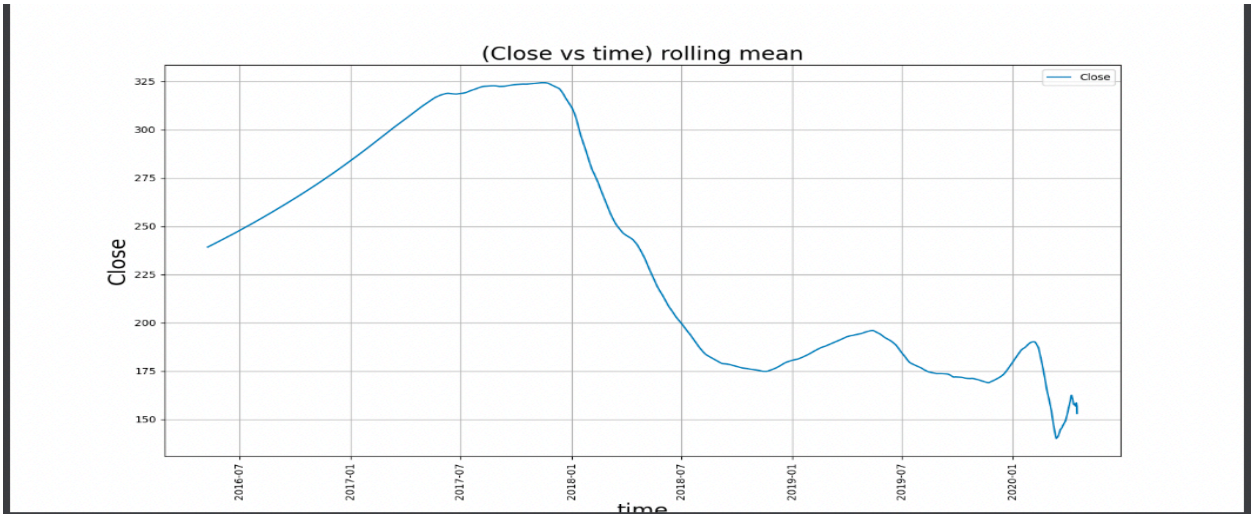


Figure:8 (Close vs time) rolling mean

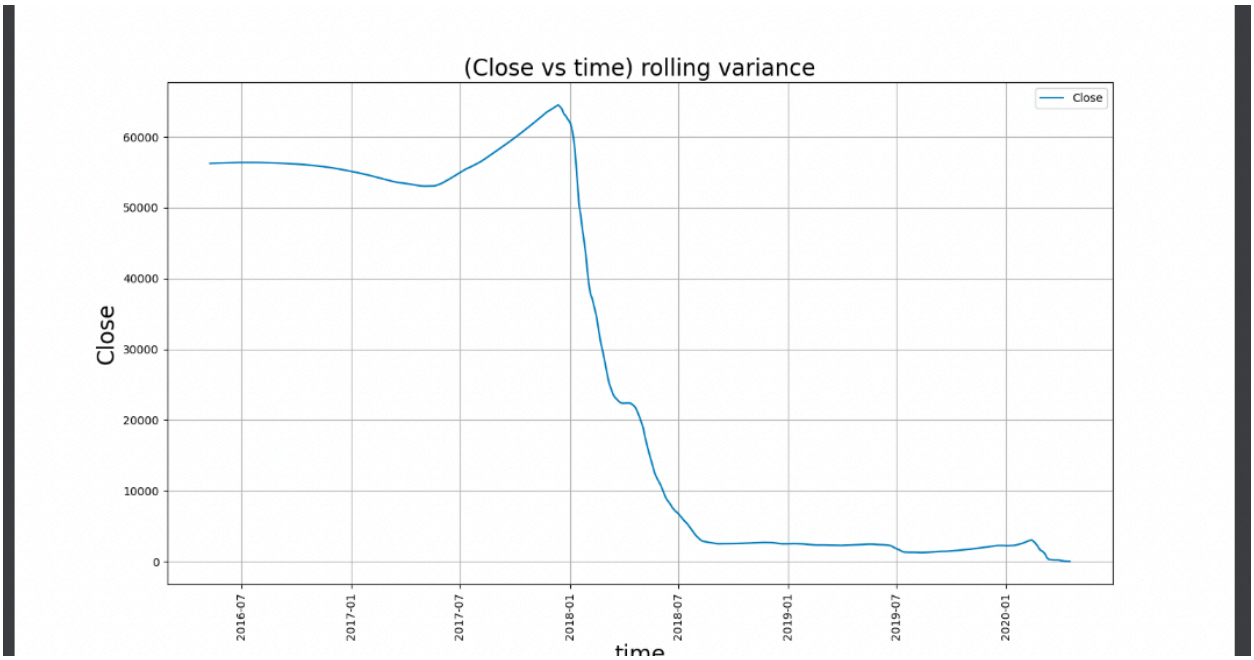


Figure:9 (Close vs time) rolling Variance

ACF/PACF of the dependent variable:

ACF (Auto correlation function): The plot of ACF shown below in that We can observe that the dependent variable does not tail-off to reach 0 value. So, we can say that the variable is non-stationary.

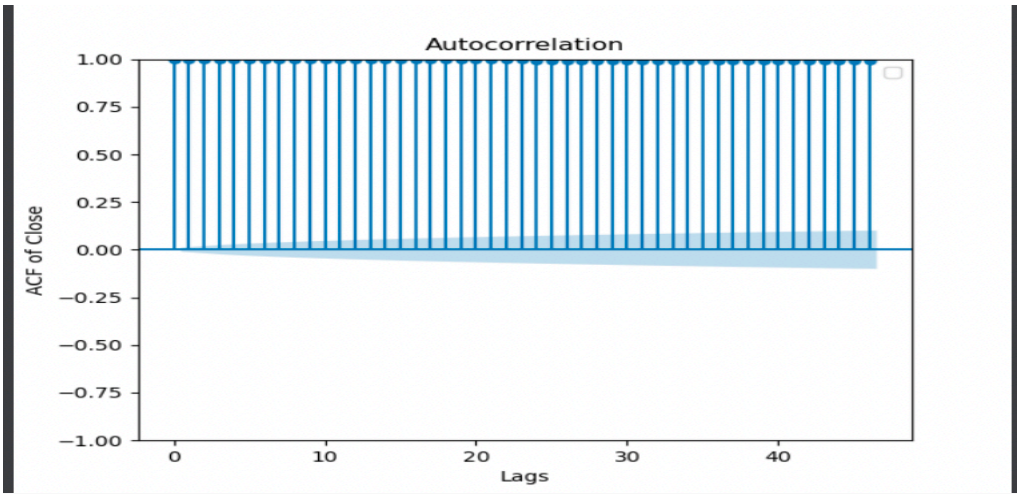


Figure:10 (ACF of Close) Auto correlation plot

PACF (Partial Auto correlation function): The plot of PACF is shown below, We can observe that the dependent variable cuts-off to reach in significant value for lag = 1. So, we can say that the nb value we can select as nb = 1.

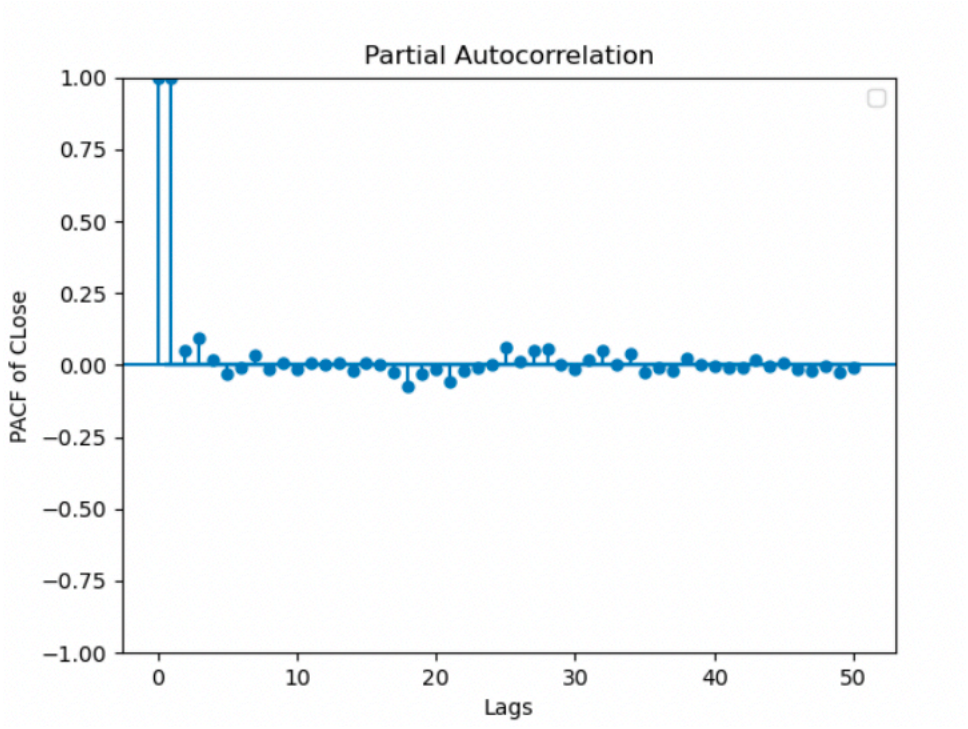


Figure:11 (PACF of Close) Partial Auto correlation plot

Heatmap of Pearson's correlation coefficient:

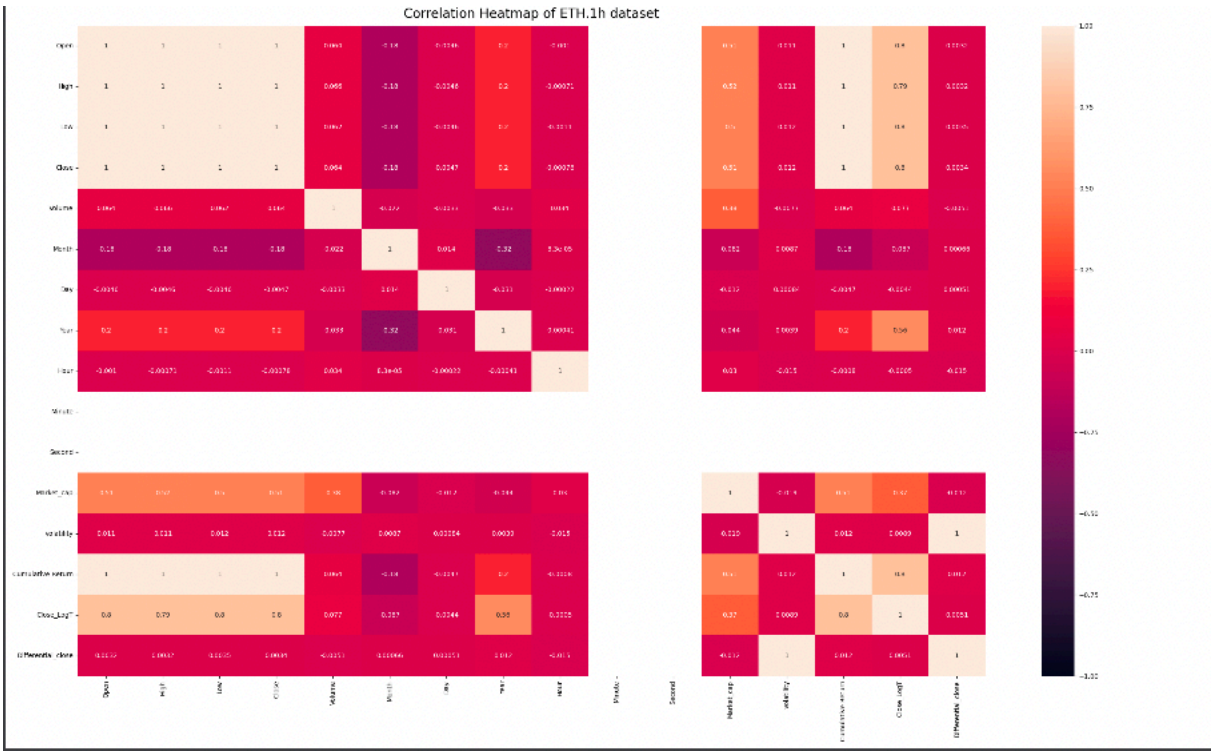


Figure:12 Heatmap plot of data

The main feature such as Open, high, and low have high correlation with the dependent variable which is close and Volume on the other has less correlation with the dependent variable.

Stationarity check:

ADF test of dependent variable: The Augmented Dickey-Fuller test can be used with serial correlation. The ADF test can handle more complex models than the Dickey-Fuller test, and it is also more powerful. All unit root tests should be used with caution as they have a relatively high type 1 error rate. The hypotheses for the test:

- The null hypothesis for this test is that there is a unit root (stationarity)
- The basic alternate is that the time series is stationary (or trend-stationary)

```
ADF test of Close Price
ADF Statistic: -1.629823
p-value: 0.467584
Critical Values:
    1%: -3.431
    5%: -2.862
    10%: -2.567
```

Figure:13 ADF test of Close

From the ADF test results we can say that the variable is stationary since the p-value is greater than 0.05.

Kpss test of dependent variable: KPSS test is a statistical test to check for stationarity of a series around a deterministic trend. Like ADF test, the KPSS test is also commonly used to analyze the stationarity of a series. However, it has couple of key differences compared to the ADF test in function and in practical usage**.**

```
KPSS test of Close Price
Results of KPSS Test:
Test Statistic            4.942598
p-value                   0.010000
Lags Used               112.000000
Critical Value (10%)      0.347000
Critical Value (5%)       0.463000
Critical Value (2.5%)     0.574000
Critical Value (1%)       0.739000
```

Figure:14 Kpss test of Close

From the KPSS test results we can say that the variable is non - stationary since the p-value is less than 0.05

After performing log transform and 1st order differencing, we were able to make the dependent variable stationary

ADF test of transformed dependent variable:

From the ADF test results we can say that the variable is stationary, since the p-value is less than 0.05

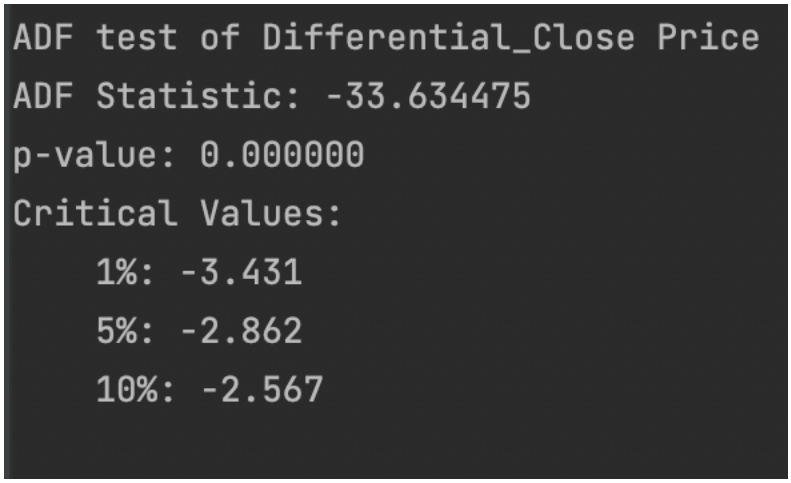


Figure:15 ADF test of Transformed close

Kpss test of transformed dependent variable:

From the KPSS test results we can say that the variable is stationary, since the p-value is greater than 0.05

```
KPSS test of Differential_Close Price
Results of KPSS Test:
Test Statistic           0.555551
p-value                  0.029155
Lags Used                2.000000
Critical Value (10%)     0.347000
Critical Value (5%)      0.463000
Critical Value (2.5%)    0.574000
Critical Value (1%)      0.739000
```

Figure:16 kpss test of Transformed close

Time Series Decomposition:

We performed time series decomposition on the raw data, and we got the below mentioned results from which we can conclude that there is a very high trend in the raw data since the strength of the trend is very high at around ~ 0.99 and there is low seasonality in the raw data since the strength of the seasonality is low at around ~0.24.

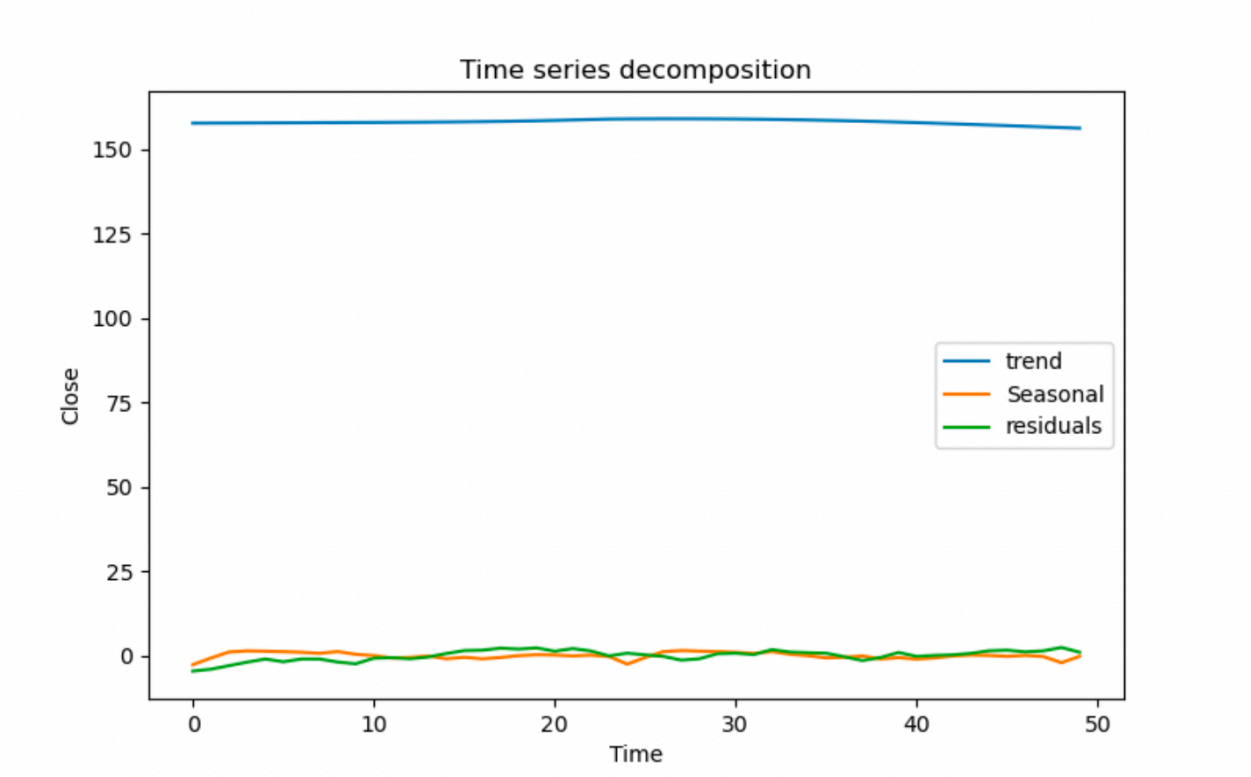


Figure:16a Time Series Decomposition

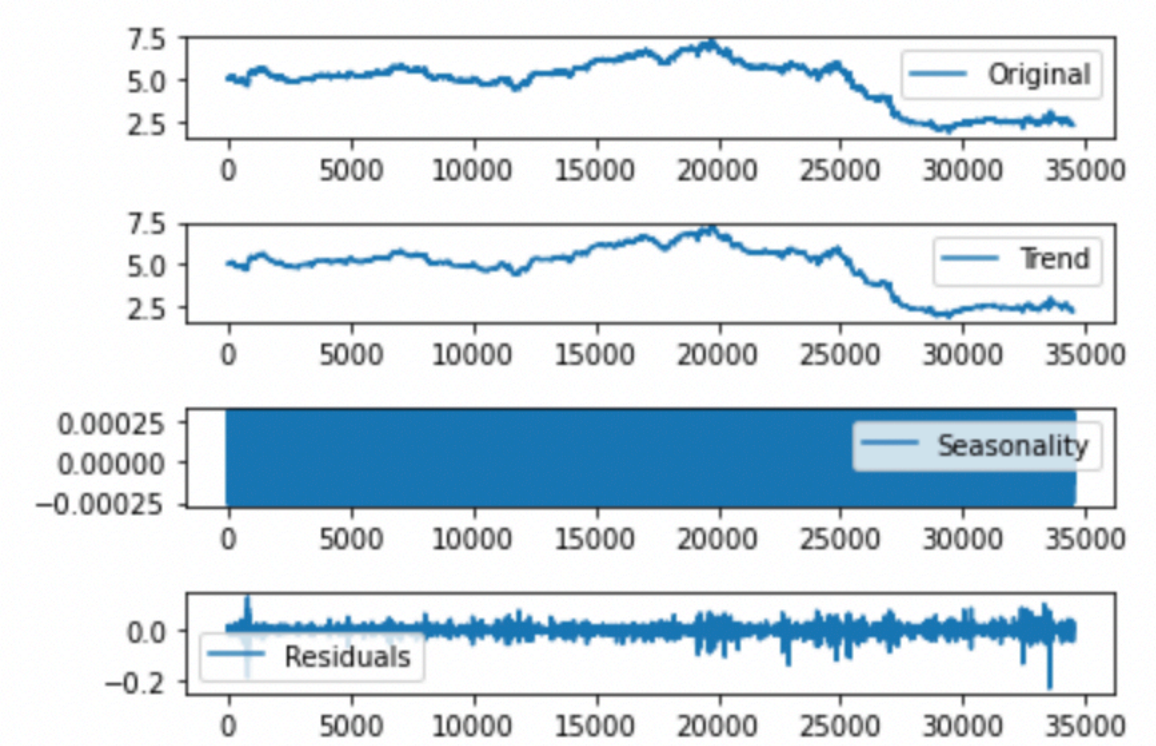


Figure:16b Time Series Decomposition

```
The strength of trend for this data set is 0.9993839070439491
The strength of seasonality for this data set is 0.24739763219669086
```

Figure:16c Time Series Decomposition Strength

Figure:16d Time Series Decomposition Seasonal adjusted

Feature selection:

Initially we have started with Open, High, Low, Market_cap and Volume as volume pvalue is was higher hence we dropped it and then we again ran the OLS model and found Markept_cap has the next high pvalue than we have dropped Market_cap.

We perform feature selection of the dataset using the OLS regression. The results of the OLS regression are shown below:

```
                                 OLS Regression Results                                
=======================================================================================
Dep. Variable:                  Close   R-squared (uncentered):                   1.000
Model:                            OLS   Adj. R-squared (uncentered):              1.000
Method:                 Least Squares   F-statistic:                          1.885e+08
Date:                Wed, 04 May 2022   Prob (F-statistic):                        0.00
Time:                        16:08:46   Log-Likelihood:                         -62890.
No. Observations:               27597   AIC:                                  1.258e+05
Df Residuals:                   27594   BIC:                                  1.258e+05
Df Model:                           3                                         
Covariance Type:            nonrobust                                         
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Open          -0.4475      0.005    -96.486      0.000      -0.457      -0.438
High           0.8099      0.004    223.169      0.000       0.803       0.817
Low            0.6369      0.003    200.698      0.000       0.631       0.643
==============================================================================
Omnibus:                    14997.773   Durbin-Watson:                   2.044
Prob(Omnibus):                  0.000   Jarque-Bera (JB):         30013810.791
Skew:                           0.994   Prob(JB):                         0.00
Kurtosis:                     164.548   Cond. No.                         235.
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
>>>
```

Figure:17a Feature Selection via OLS

Since none of the variables (Open, high, and low) has a high p-value we don't need to drop any variable anymore. We can perform predictions with this.

Estimated Prediction Error:

```
estimated variance of prediction error : 2.363154390281497
estimated variance of forecast error : 1.320530624261502
```

Figure:17b Feature Selection estimated variance

We can observe that the OLS model is very good at predicting the ETH model. The variance of Prediction error and forecast error is low

Split the dataset into train and test values:

We may divide the dataset into train and test datasets by using the following code line. Because this is a time series dataset, shuffle=False is used.

```
y=df['Close']
yt,yf=train_test_split(y,shuffle=False,test_size=0.3)
```

Base Models:

Holt winter method:

We can observe that the Holt winter method did not give good predictions as it is not following the test results.

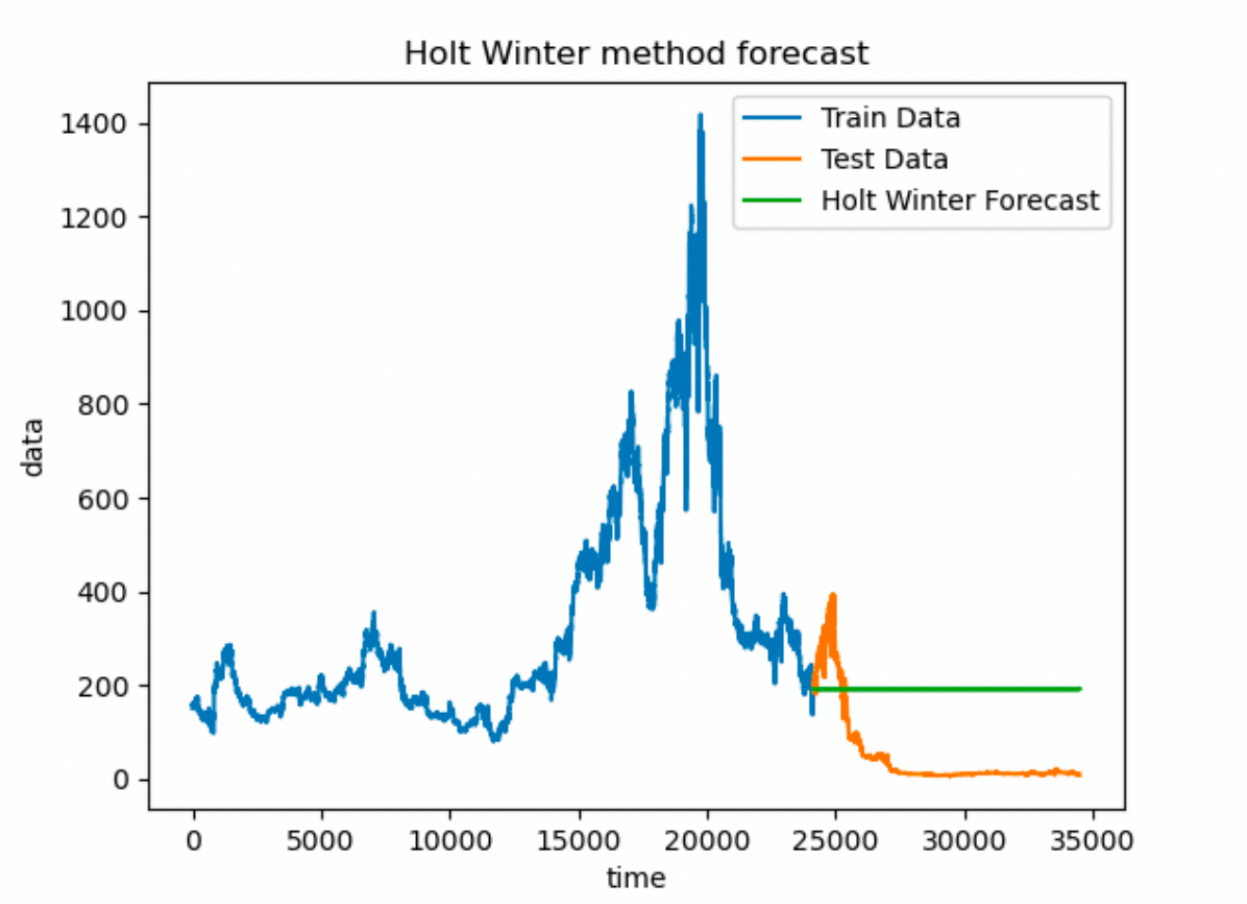


Figure:18a Holt winter method Forecast

Average model: From the graph we can see that the forecast of the average method for the length of the test dataset is the average of the train dataset.

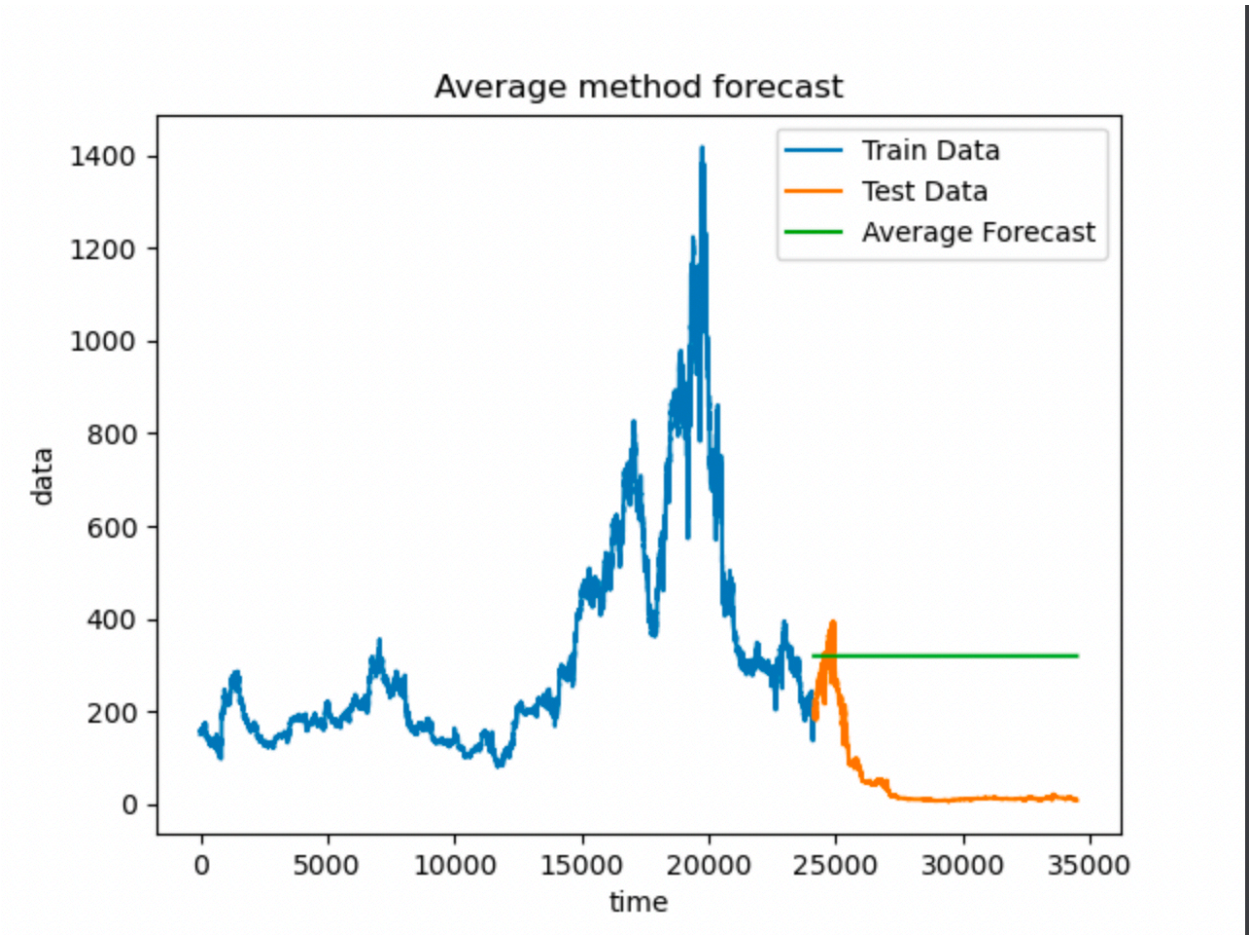


Figure:18b Average method Forecast

Naive model:

The graph we can see that the forecast of the naive method for the length of the test dataset is the last value of the train dataset.

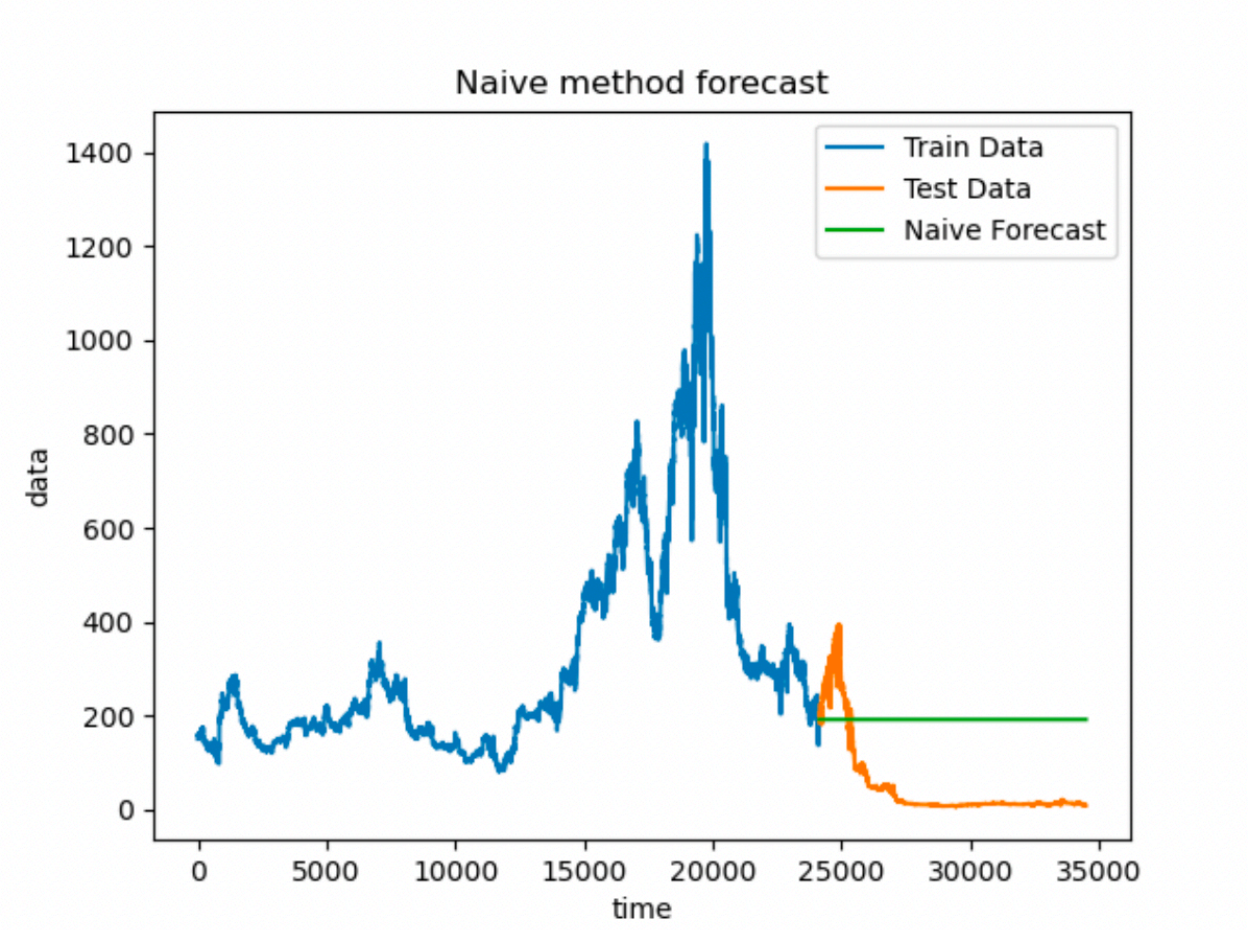


Figure:18c naive method Forecast

Drift method:

The graph we can see that the forecast of the drift method for the length of the test dataset is the extrapolating the line that connects the 1st and last value of the train dataset.

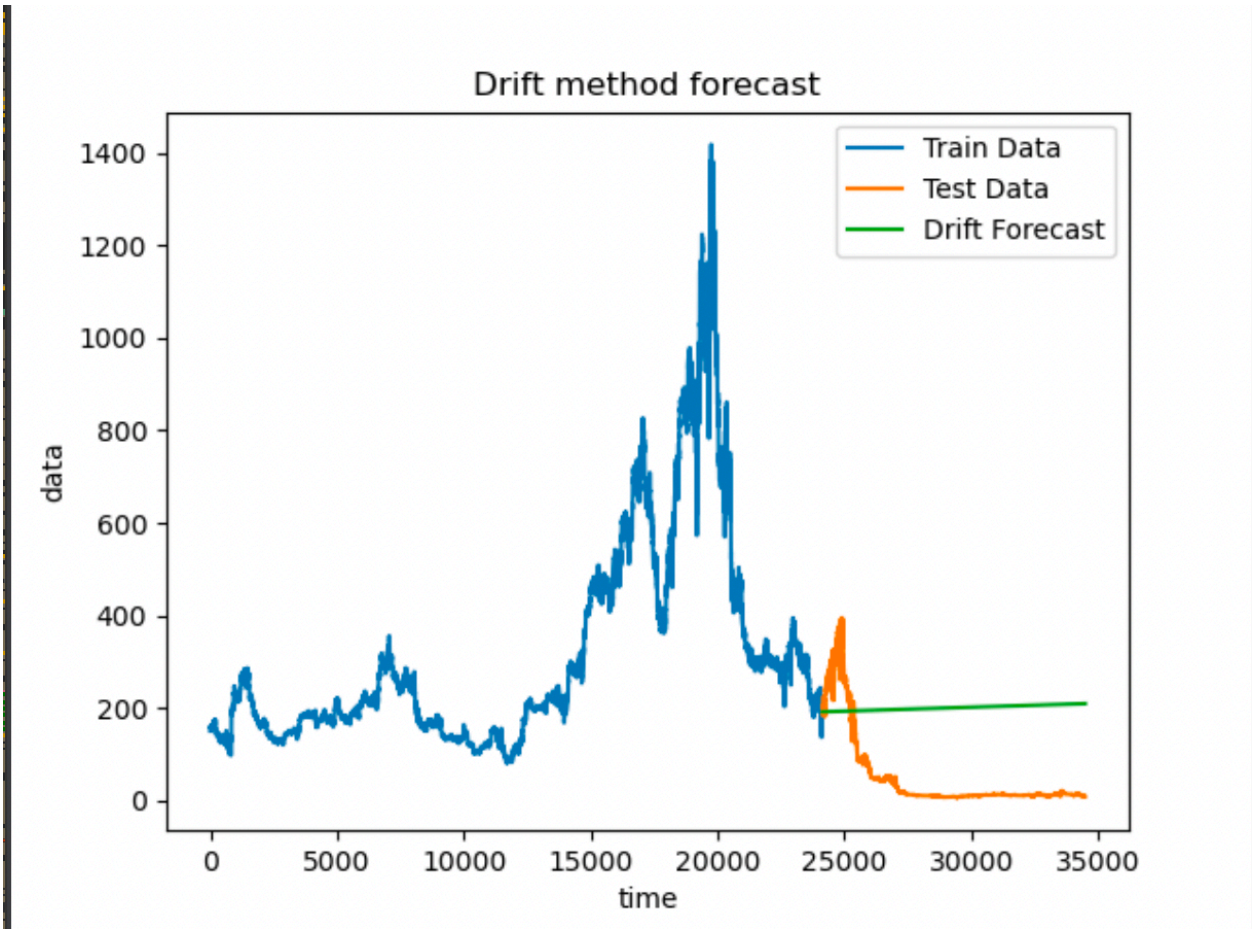


Figure:18d Drift method Forecast

Simple Exponential Smoothing (SES) method:

The graph we can see that the forecast of the SES method for the length of the test dataset follows the SES equation. The results are similar to the Holt winter model forecast.

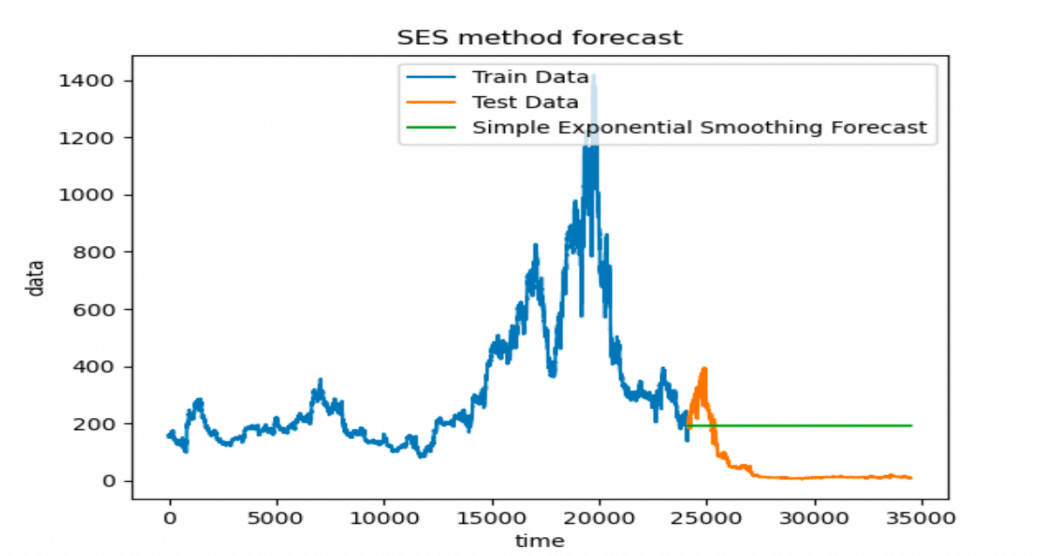


Figure:18e SES method Forecast

Holt Linear method: We can observe that the Holt winter method gave good predictions.

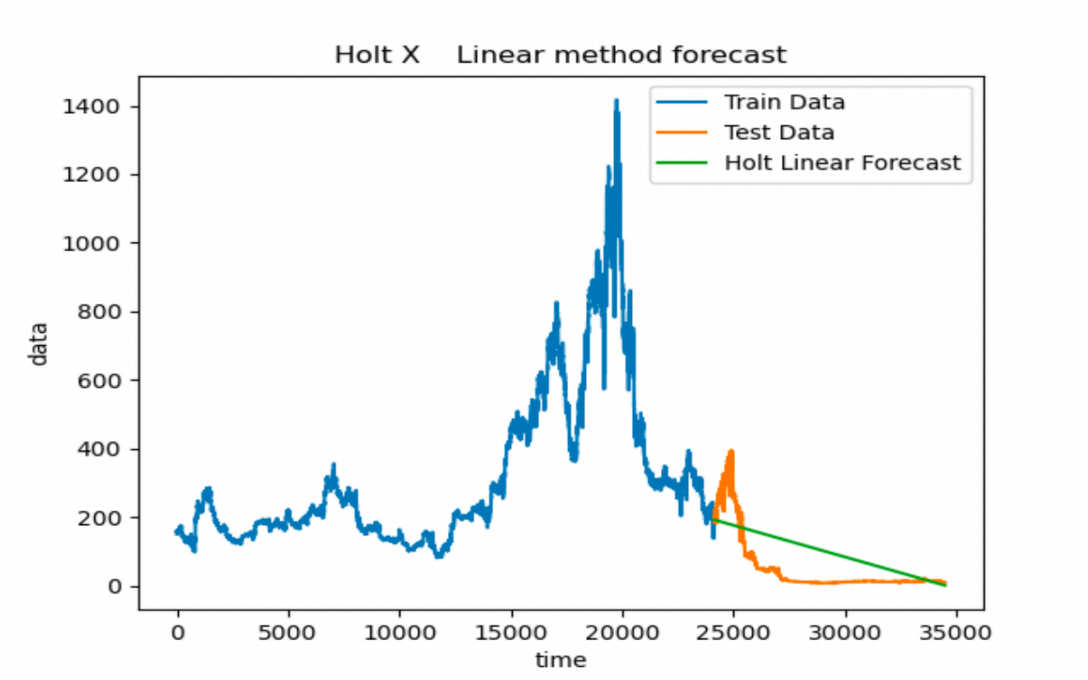


Figure:18f Holt linear method Forecast

ARMA model:

We perform the GPAC table of the ACF values of the Transformed data. The following is the GPAC table that we obtain from it.

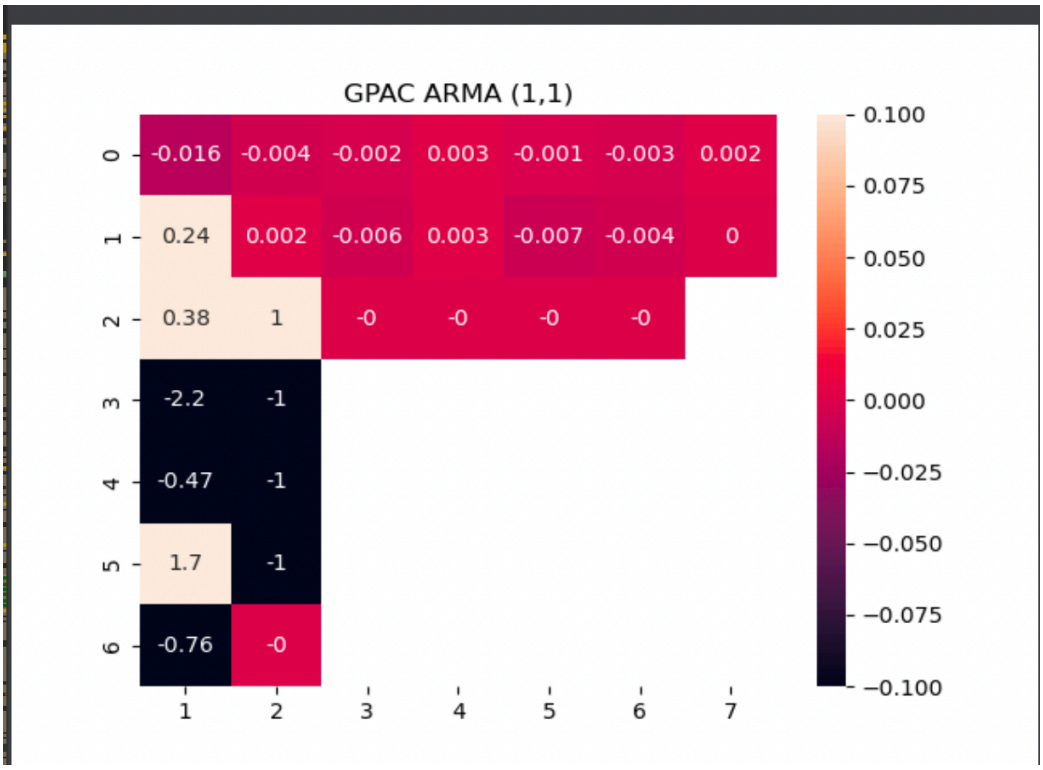


Figure:19 GPAC

From the above GPAC table we can observe that the 2st column is the column of constants, and its 1st row is the row of zeros, since the value of the 1st row is very close to zero.
So, we can select the number of AR coefficients, na=2 and the number of MA coefficients, nb=1 for the ARMA and ARIMA models.

In this project we performed the 2 ARMA models, each for ARMA(1,1) and ARMA (2,2).

```
========================= ARMA (1,1) model ================================
                              SARIMAX Results
==========================================================================
Dep. Variable:     Differential_close   No. Observations:               27597
Model:                 ARIMA(1, 0, 1)   Log Likelihood              55194.550
Date:                Wed, 04 May 2022   AIC                       -110381.099
Time:                        20:46:40   BIC                       -110348.198
Sample:                             0   HQIC                      -110370.500
                                - 27597
Covariance Type:                  opg
==========================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
--------------------------------------------------------------------------
const         7.8e-05      0.000      0.162      0.872      -0.001       0.001
ar.L1          0.7542      0.163      4.627      0.000       0.435       1.074
ma.L1         -0.7665      0.162     -4.737      0.000      -1.084      -0.449
sigma2         0.0011   2.98e-07   3597.383      0.000       0.001       0.001
==========================================================================
Ljung-Box (L1) (Q):                   1.94   Jarque-Bera (JB):      454824853813.66
Prob(Q):                              0.16   Prob(JB):                         0.00
Heteroskedasticity (H):               0.09   Skew:                           129.94
Prob(H) (two-sided):                  0.00   Kurtosis:                     19889.56
==========================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
>>>
```

The AR coefficient is 7.79 and MA coefficient is 0.75.

The AIC and BIC values are low so we can say that this is not good model.
Diagnostic test:

The ACF plot of residual error is shown below:

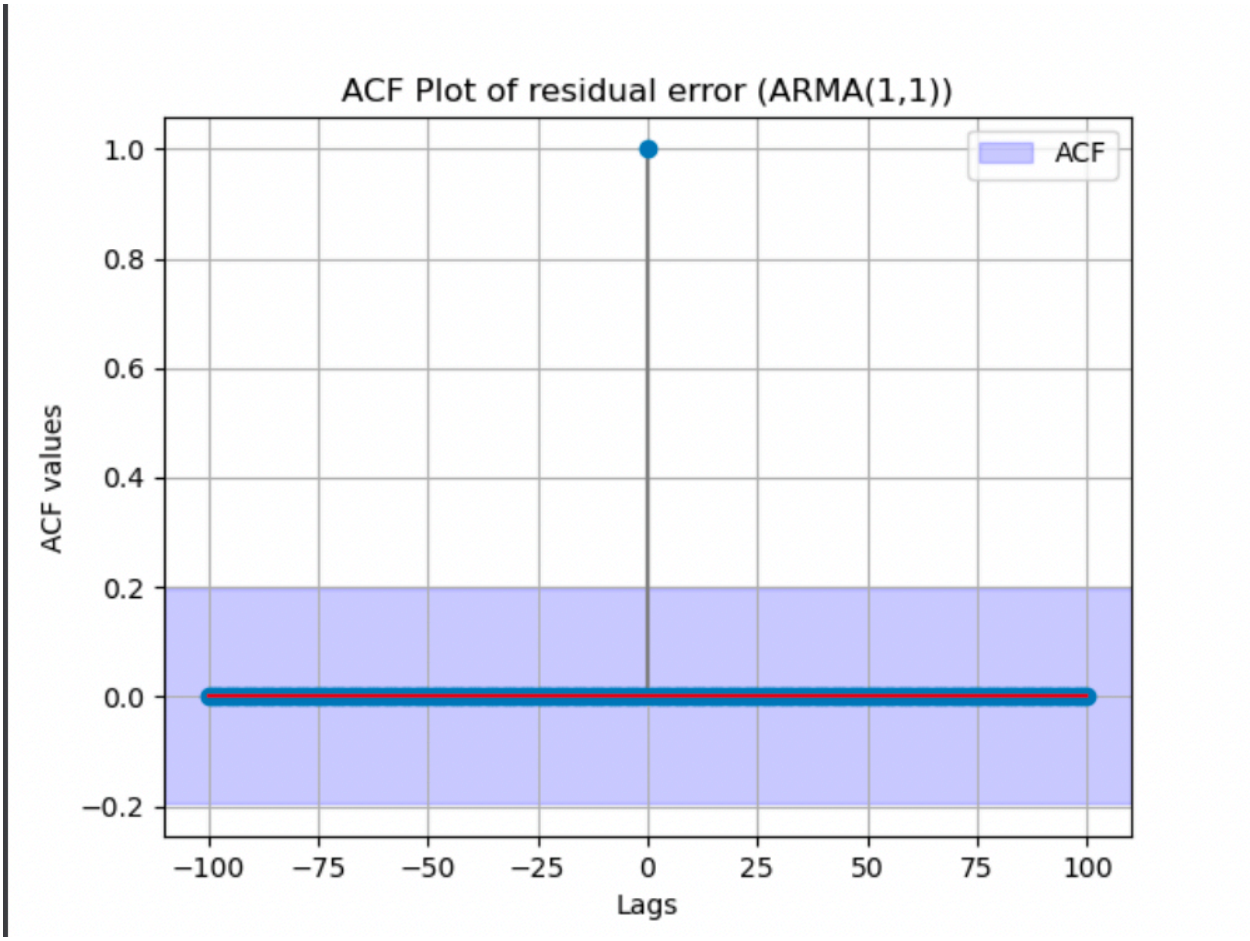


Figure:20a ACF residual Error (ARMA (1,1))

confidence intervals of estimated parameters: 0 1

const -0.000867 0.001023

ar.L1 0.434724 1.073694

ma.L1 -1.083625 -0.449376

sigma2 0.001072 0.001073

zero/cancellation:

zeros : [-0.7542091216727449]

poles : [-7.79992318299006e-05]

Chi Squared test results

The residuals is white, chi squared value :2.5749919487075363e-09

variance of residual error : 0.7554447231482881

variance of forecast error : 0.0006132498037137332

The variance of residual and forecast are relatively small so this is a good model for forecasting Close.

The 1-step prediction of 1st 100 samples plot is shown below:

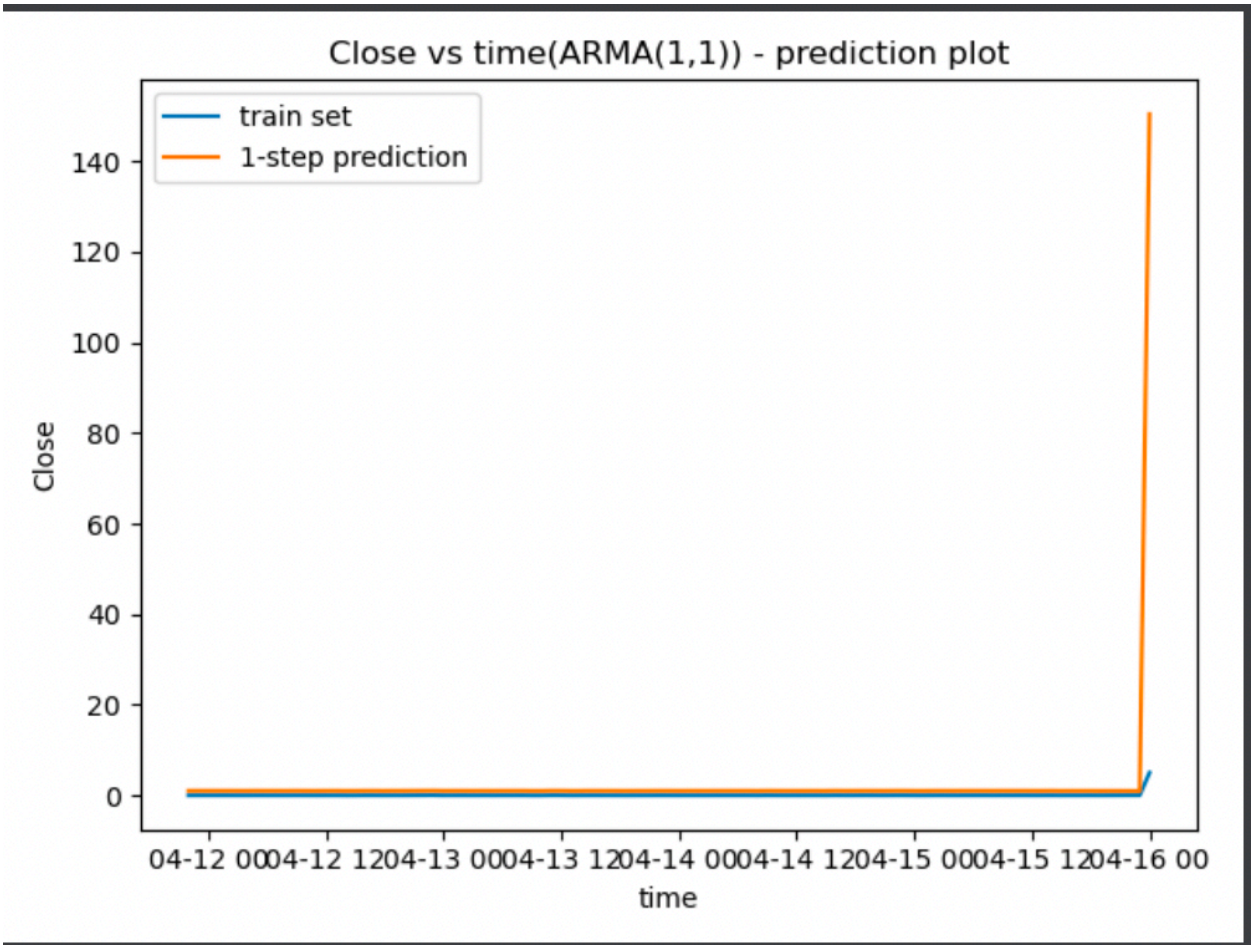


Figure:21 Close vs Time(ARMA (1,1)) prediction plot

The h-step forecast plot is shown below:

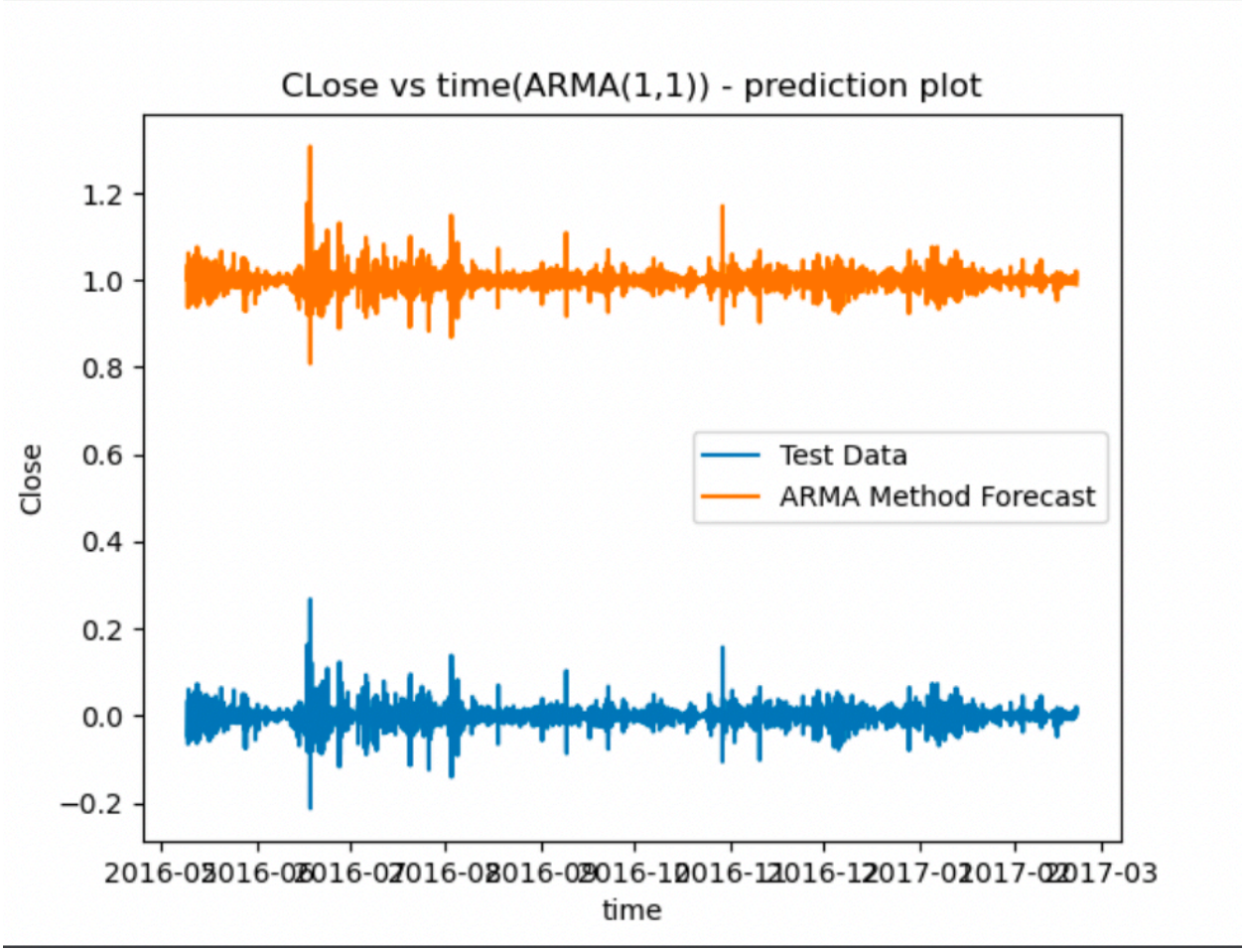


Figure:22 Close vs Time(ARMA (1,1))h step prediction plot

From above 1-step prediction and h-step forecast plots we can see that the model is fitting the predictions very well. So we can say that this is a good model to predict Close values.

From the confidence intervals we can see that only for one coefficient the interval is crossing 0, which is not good for the model.

The 1-step prediction of 1st 100 samples plot is shown below:

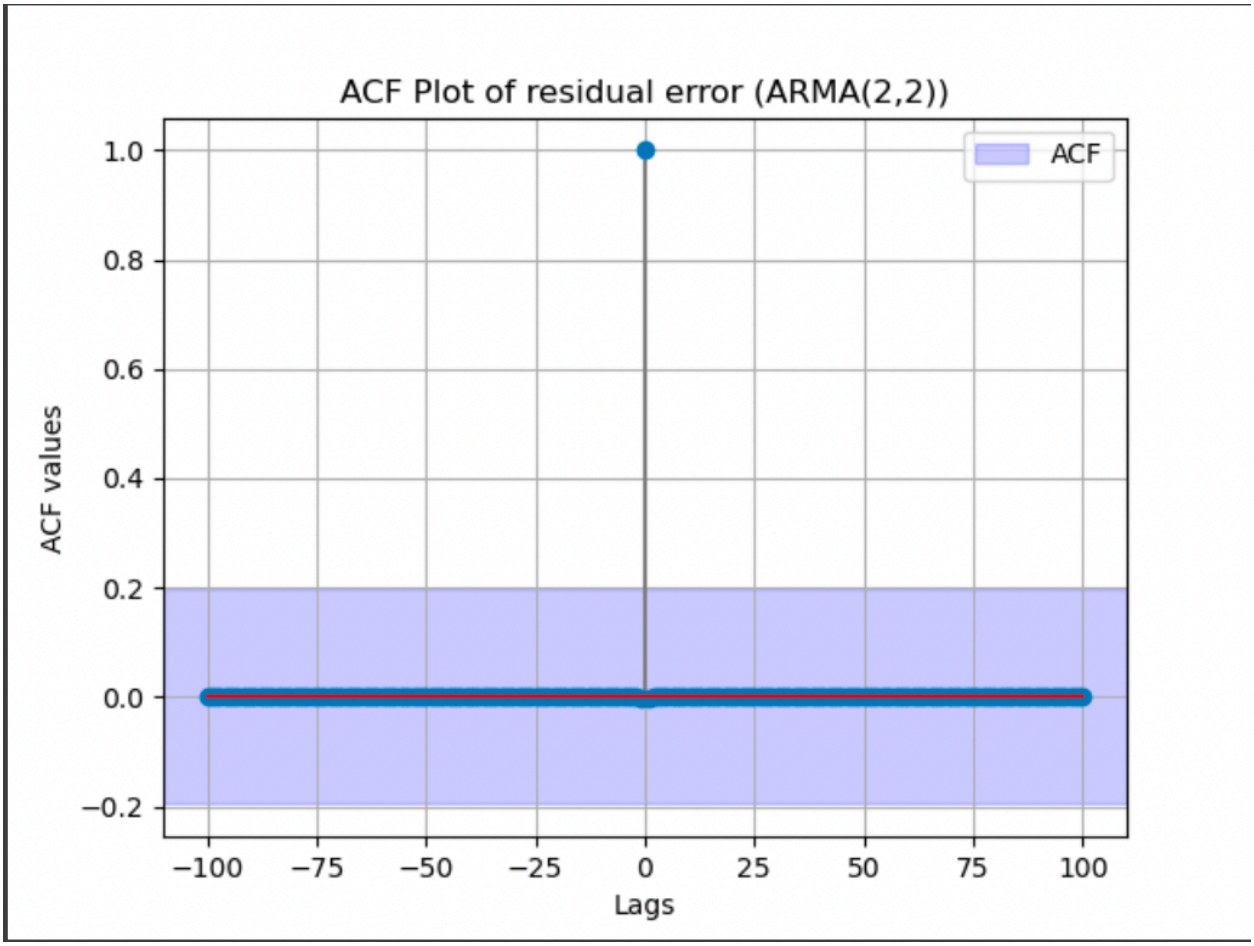


Figure:23 ACF Residual Error(ARMA (2,2))

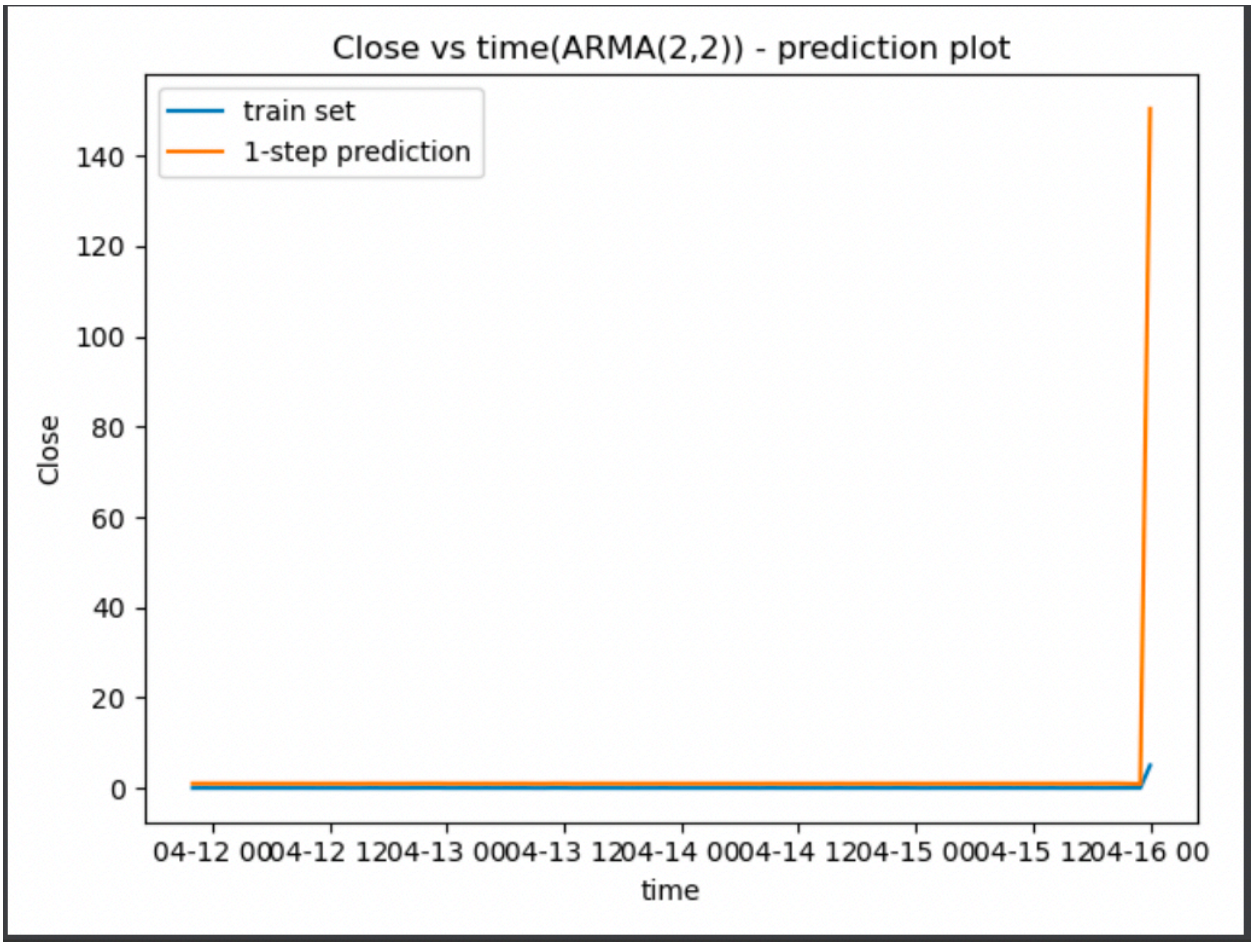


Figure:24 Close vs Time(ARMA (2,2)) prediction plot

From above 1-step prediction and h-step forecast plots we can see that the model is fitting the predictions very well. So, we can say that this is a good model to predict Close values.

```
========================= ARMA (2,2) model ================================
                              SARIMAX Results
==========================================================================
Dep. Variable:     Differential_close   No. Observations:               27597
Model:                 ARIMA(2, 0, 2)   Log Likelihood              55195.719
Date:                Wed, 04 May 2022   AIC                       -110379.438
Time:                        20:46:49   BIC                       -110330.085
Sample:                             0   HQIC                      -110363.540
                                - 27597
Covariance Type:                  opg
==========================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
--------------------------------------------------------------------------
const       8.196e-05      0.000      0.166      0.868      -0.001       0.001
ar.L1          0.7904      0.885      0.893      0.372      -0.945       2.526
ar.L2         -0.4010      0.305     -1.317      0.188      -0.998       0.196
ma.L1         -0.8155      0.885     -0.921      0.357      -2.551       0.920
ma.L2          0.4092      0.321      1.275      0.202      -0.220       1.038
sigma2         0.0011   2.93e-07   3659.128      0.000       0.001       0.001
==========================================================================
Ljung-Box (L1) (Q):                  12.37   Jarque-Bera (JB):      454133234170.19
Prob(Q):                              0.00   Prob(JB):                         0.00
Heteroskedasticity (H):               0.09   Skew:                           129.87
Prob(H) (two-sided):                  0.00   Kurtosis:                     19874.43
>>>
```

The AR Coefficient : a1 is:, 8.195856264119555e-05

The AR Coefficient : a2 is:, 0.7904231422672403

The MA Coefficient : b1 is:, -0.40103370405770855

The MA Coefficient : b2 is:, -0.8155494920475628

confidence intervals of estimated parameters: 0 1

const -0.000886 0.001050

ar.L1 -0.944678 2.525525

ar.L2 -0.997960 0.195892

ma.L1 -2.550916 0.919817

ma.L2 -0.219679 1.038017

sigma2 0.001072 0.001073

zero/cancellation:

zeros : [0.40103370405770855, 0.8155494920475628]

poles : [-8.195856264119555e-05, -0.7904231422672403]

Chi Squared test results

The residuals is white, chi squared value :5.779291945732471e-10

variance of residual error : 0.755451362874238

variance of forecast error : 3.9080038750518717

We can observe that the variance of the residual errors is slightly less than the ARMA (1,1). So we can say that the ARMA (2,2) model is better than the ARMA (1,1) model.

ARIMA model:

ARIMA(3,1,18) and ARIMA(2,2,10) results mentioned below.

ARIMA(3,1,18)

AR model(3,1,0)

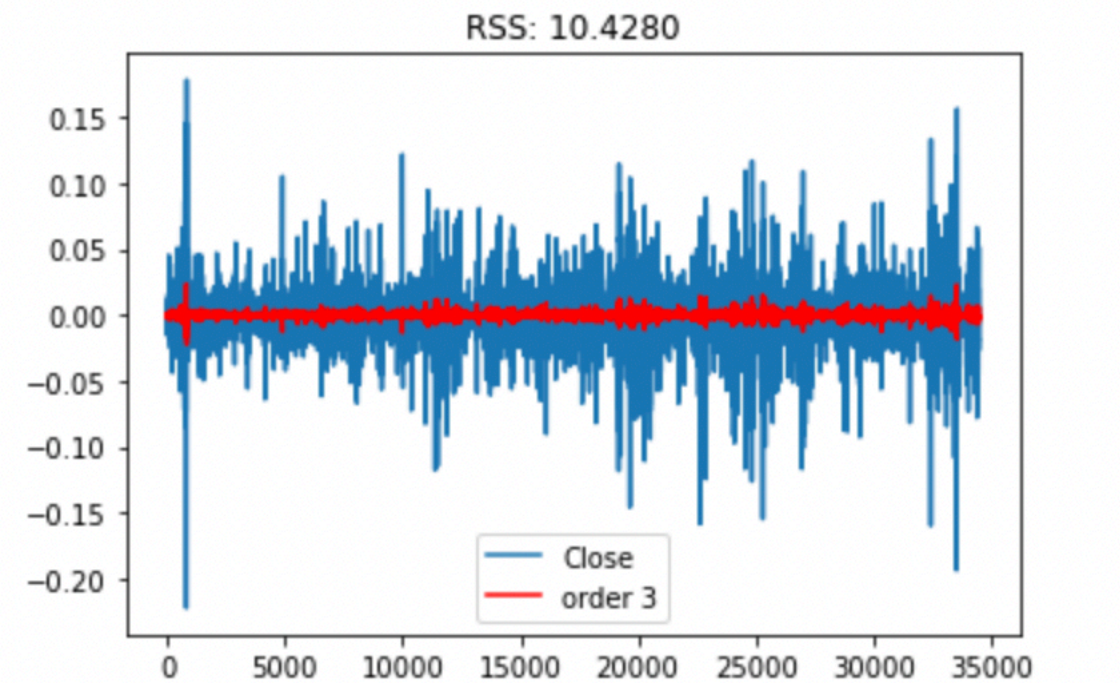


Figure:25 Close vs Time(ARIMA (3,1,0)) prediction plot

MA model(0,1,18)

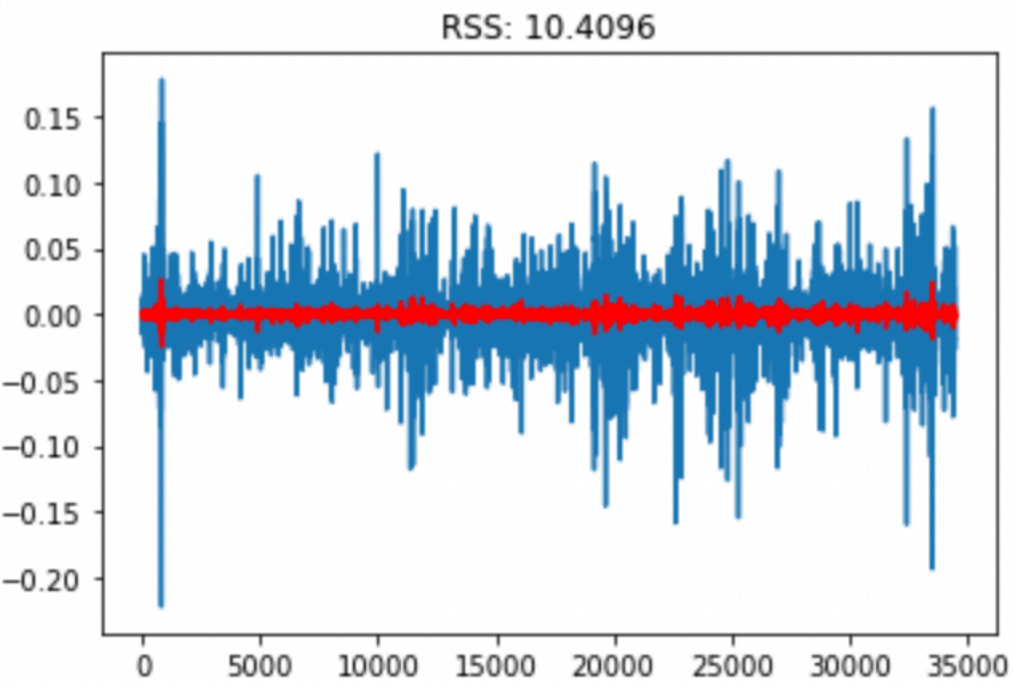


Figure:26 Close vs Time(ARIMA (0,1,18)) prediction plot

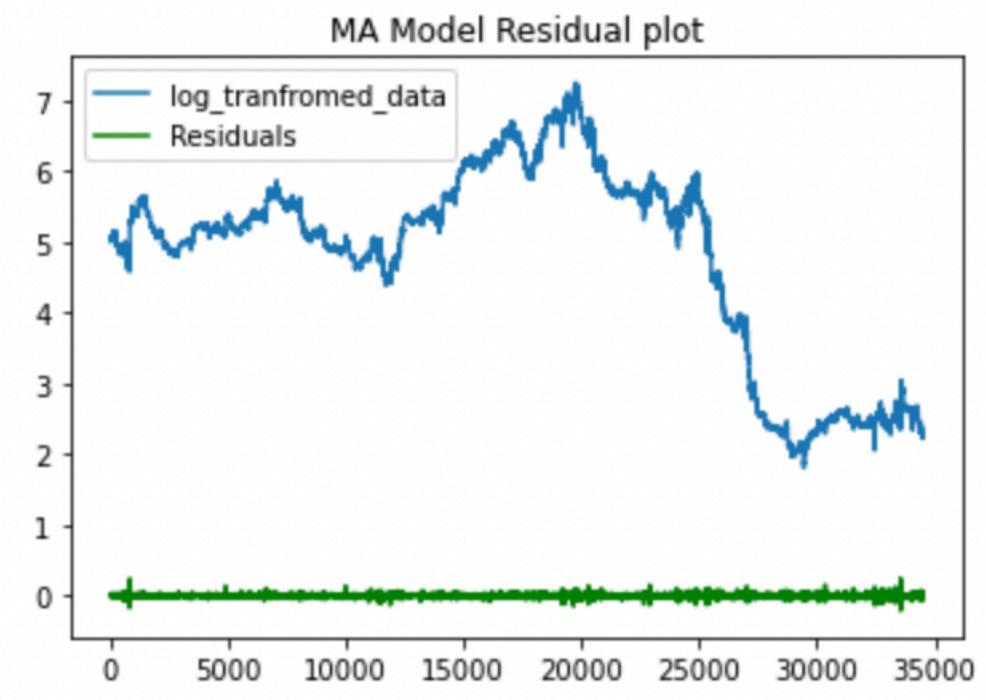


Figure:27 Residual Plot(ARIMA (0,1,18)) prediction plot

#ARIMA Combined model(3,1,18)

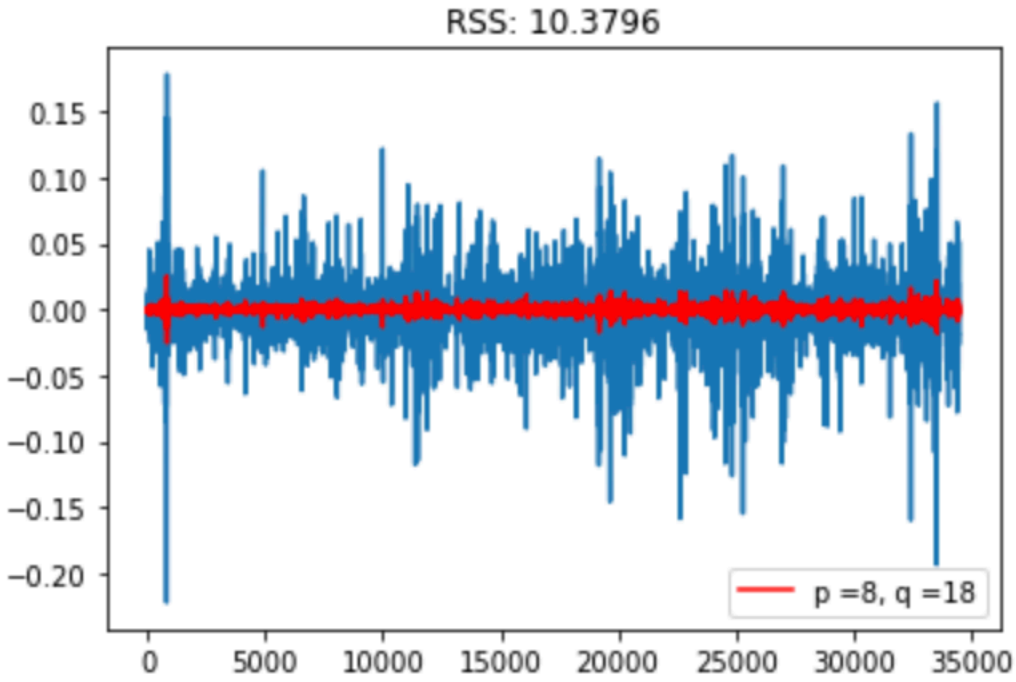


Figure:28 Close vs Time(ARMA (2,2)) prediction plot

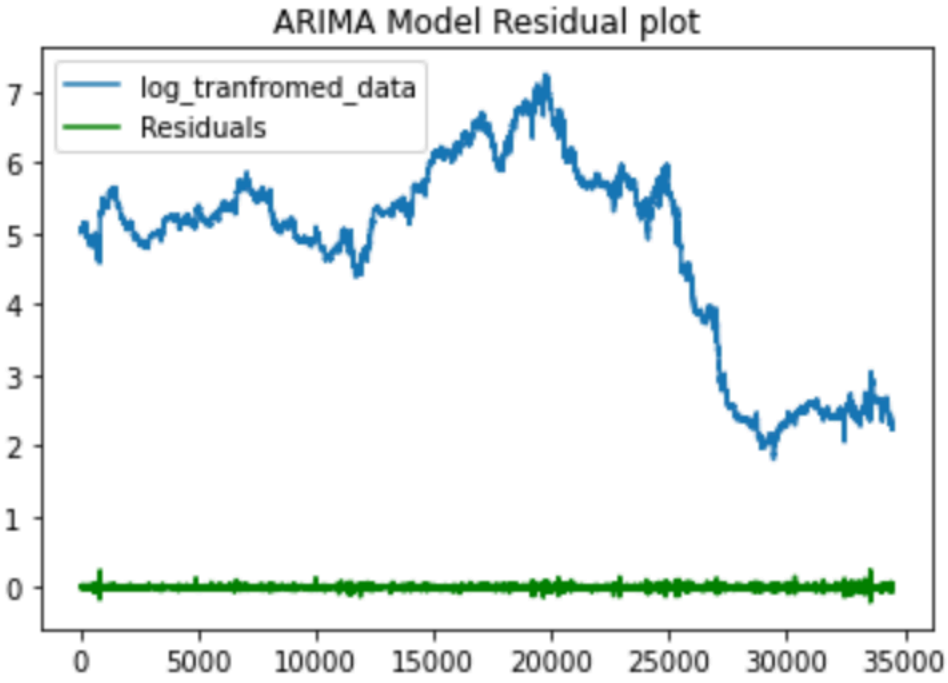


Figure:29 Close vs Time(ARMA (2,2)) prediction plot

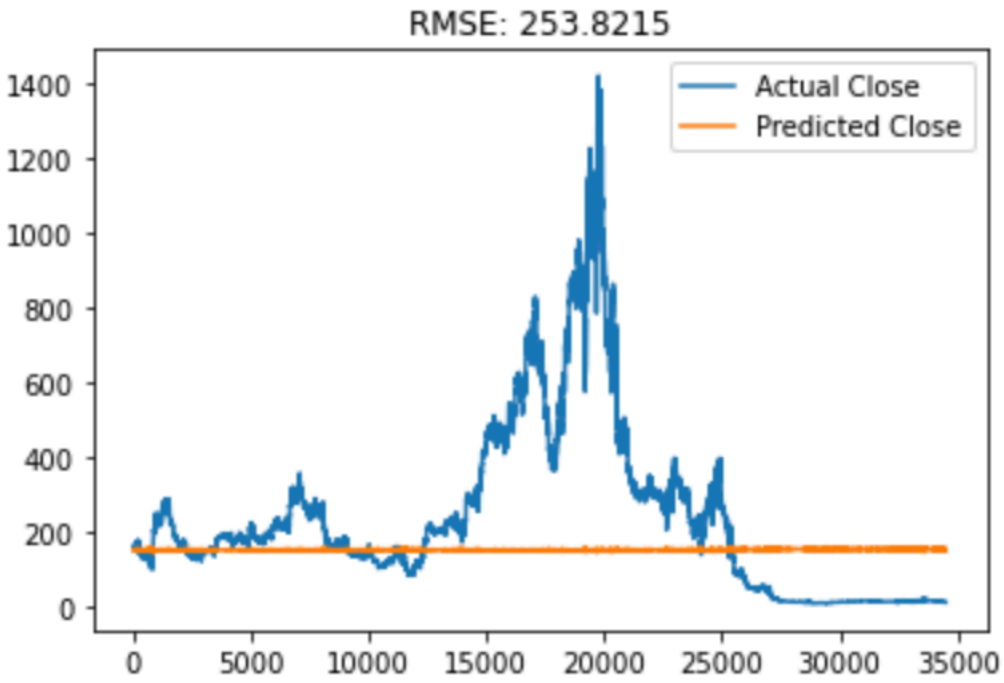


Figure:30 Actual vs Predicted(ARMA (3,1,18))

```
                               SARIMAX Results
==============================================================================
Dep. Variable:             Close_LogT   No. Observations:                34497
Model:                ARIMA(0, 1, 18)   Log Likelihood              100061.418
Date:                Wed, 04 May 2022   AIC                        -200084.837
Time:                        20:47:51   BIC                        -199924.313
Sample:                             0   HQIC                       -200033.671
                                - 34497
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ma.L1         -0.1026      0.002    -49.719      0.000      -0.107      -0.099
ma.L2         -0.0387      0.003    -15.111      0.000      -0.044      -0.034
ma.L3         -0.0163      0.002     -6.522      0.000      -0.021      -0.011
ma.L4          0.0103      0.003      3.860      0.000       0.005       0.015
ma.L5         -0.0033      0.003     -1.162      0.245      -0.009       0.002
ma.L6         -0.0162      0.003     -5.713      0.000      -0.022      -0.011
ma.L7          0.0125      0.003      4.532      0.000       0.007       0.018
ma.L8          0.0011      0.003      0.398      0.690      -0.004       0.006
ma.L9          0.0186      0.003      6.272      0.000       0.013       0.024
ma.L10         0.0082      0.003      2.791      0.005       0.002       0.014
ma.L11         0.0070      0.003      2.374      0.018       0.001       0.013
ma.L12         0.0060      0.003      1.974      0.048    4.39e-05       0.012
ma.L13         0.0249      0.003      8.851      0.000       0.019       0.030
>>>
```

```
ma.L13         0.0249      0.003      8.851      0.000       0.019       0.030
ma.L14        -0.0017      0.003     -0.527      0.598      -0.008       0.005
ma.L15         0.0176      0.003      5.888      0.000       0.012       0.023
ma.L16         0.0211      0.003      7.839      0.000       0.016       0.026
ma.L17         0.0201      0.003      6.944      0.000       0.014       0.026
ma.L18         0.0135      0.003      4.391      0.000       0.007       0.020
sigma2         0.0002   4.12e-07    429.464      0.000       0.000       0.000
===================================================================================
Ljung-Box (L1) (Q):                   0.21   Jarque-Bera (JB):            896792.56
Prob(Q):                              0.64   Prob(JB):                         0.00
Heteroskedasticity (H):               2.42   Skew:                             0.13
Prob(H) (two-sided):                  0.00   Kurtosis:                        27.98
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
count    34497.000000
mean         0.000064
std          0.030093
min         -0.214481
25%         -0.004277
50%         -0.000014
75%          0.004070
max          5.013232
dtype: float64
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:             Close_LogT   No. Observations:                34497
Model:                ARIMA(3, 1, 18)   Log Likelihood              100062.379
Date:                Wed, 04 May 2022   AIC                        -200080.758
Time:                        20:48:14   BIC                        -199894.889
Sample:                             0   HQIC                       -200021.513
                                - 34497
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.3118      2.694      0.116      0.908      -4.968       5.592
ar.L2          0.0281      2.079      0.014      0.989      -4.047       4.104
ar.L3          0.1257      0.508      0.247      0.805      -0.870       1.122
ma.L1         -0.4144      2.694     -0.154      0.878      -5.694       4.866
ma.L2         -0.0348      2.344     -0.015      0.988      -4.629       4.559
ma.L3         -0.1271      0.609     -0.209      0.835      -1.321       1.067
ma.L4          0.0293      0.027      1.069      0.285      -0.024       0.083
ma.L5         -0.0012      0.053     -0.024      0.981      -0.105       0.103
ma.L6         -0.0134      0.035     -0.380      0.704      -0.083       0.056
ma.L7          0.0164      0.036      0.456      0.649      -0.054       0.087
ma.L8         -0.0020      0.066     -0.030      0.976      -0.131       0.127
ma.L9          0.0200      0.030      0.676      0.499      -0.038       0.078
ma.L10         0.0007      0.051      0.014      0.989      -0.099       0.100
>>>
```

```
ma.L12          0.0012      0.008      0.155      0.877      -0.015       0.017
ma.L13          0.0218      0.005      3.977      0.000       0.011       0.033
ma.L14         -0.0103      0.056     -0.182      0.856      -0.121       0.100
ma.L15          0.0169      0.054      0.313      0.754      -0.089       0.122
ma.L16          0.0126      0.057      0.222      0.825      -0.099       0.124
ma.L17          0.0133      0.025      0.537      0.591      -0.035       0.062
ma.L18          0.0046      0.018      0.258      0.797      -0.030       0.039
sigma2          0.0002   4.24e-07    417.860      0.000       0.000       0.000
===================================================================================
Ljung-Box (L1) (Q):                   0.21   Jarque-Bera (JB):            894372.90
Prob(Q):                              0.64   Prob(JB):                         0.00
Heteroskedasticity (H):               2.42   Skew:                             0.13
Prob(H) (two-sided):                  0.00   Kurtosis:                        27.94
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
count    34497.000000
mean         0.000066
std          0.030093
min         -0.213923
25%         -0.004289
50%         -0.000016
75%          0.004073
max          5.013232
>>>
```

Predictions:

```
24147    189.646659
24148    171.383834
24149    172.547281
24150    192.868977
24151    189.489673
            ...
34492    189.140901
34493    198.697523
34494    190.168154
34495    183.650718
34496    164.796579
Length: 10350, dtype: float64
Test MSE: 24814.967
```

ARIMA(2,2,10)

```
                               SARIMAX Results
==============================================================================
Dep. Variable:             Close_LogT   No. Observations:                34497
Model:                ARIMA(0, 2, 10)   Log Likelihood               99986.394
Date:                Wed, 04 May 2022   AIC                        -199950.788
Time:                        20:48:53   BIC                        -199857.853
Sample:                             0   HQIC                       -199921.165
                                - 34497
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ma.L1         -1.1133      0.002   -571.382      0.000      -1.117      -1.109
ma.L2          0.0899      0.003     28.024      0.000       0.084       0.096
ma.L3          0.0052      0.003      1.492      0.136      -0.002       0.012
ma.L4          0.0268      0.004      6.988      0.000       0.019       0.034
ma.L5         -0.0042      0.004     -1.067      0.286      -0.012       0.004
ma.L6         -0.0257      0.004     -6.256      0.000      -0.034      -0.018
ma.L7          0.0313      0.004      7.686      0.000       0.023       0.039
ma.L8         -0.0143      0.004     -3.473      0.001      -0.022      -0.006
ma.L9          0.0143      0.004      3.583      0.000       0.006       0.022
ma.L10        -0.0079      0.003     -2.701      0.007      -0.014      -0.002
sigma2         0.0002   3.98e-07    445.613      0.000       0.000       0.000
===================================================================================
Ljung-Box (L1) (Q):                   8.39   Jarque-Bera (JB):            888181.84
>>>
```

```
ma.L8         -0.0143      0.004     -3.473      0.001      -0.022      -0.006
ma.L9          0.0143      0.004      3.583      0.000       0.006       0.022
ma.L10        -0.0079      0.003     -2.701      0.007      -0.014      -0.002
sigma2         0.0002   3.98e-07    445.613      0.000       0.000       0.000
===================================================================================
Ljung-Box (L1) (Q):                   8.39   Jarque-Bera (JB):            888181.86
Prob(Q):                              0.00   Prob(JB):                         0.00
Heteroskedasticity (H):               2.41   Skew:                             0.13
Prob(H) (two-sided):                  0.00   Kurtosis:                        27.86
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
count    34497.000000
mean         0.000058
std          0.032955
min         -2.489802
25%         -0.004253
50%          0.000044
75%          0.004140
max          5.013232
dtype: float64
                              SARIMAX Results
>>>
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:             Close_LogT   No. Observations:                34497
Model:                ARIMA(2, 2, 10)   Log Likelihood               99994.677
Date:                Wed, 04 May 2022   AIC                        -199963.353
Time:                        20:49:39   BIC                        -199853.522
Sample:                             0   HQIC                       -199928.345
                                - 34497
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -1.4921      0.187     -7.995      0.000      -1.858      -1.126
ar.L2         -0.7907      0.188     -4.203      0.000      -1.159      -0.422
ma.L1          0.3935      0.187      2.105      0.035       0.027       0.760
ma.L2         -0.7886      0.115     -6.872      0.000      -1.014      -0.564
ma.L3         -0.7525      0.198     -3.807      0.000      -1.140      -0.365
ma.L4          0.1211      0.016      7.633      0.000       0.090       0.152
ma.L5          0.0458      0.012      3.932      0.000       0.023       0.069
ma.L6         -0.0207      0.005     -4.130      0.000      -0.030      -0.011
ma.L7         -0.0023      0.008     -0.305      0.761      -0.017       0.013
ma.L8          0.0190      0.007      2.713      0.007       0.005       0.033
ma.L9         -0.0021      0.005     -0.394      0.694      -0.012       0.008
ma.L10        -0.0095      0.006     -1.716      0.086      -0.020       0.001
sigma2         0.0002   4.03e-07    441.528      0.000       0.000       0.000
```

```
ma.L6         -0.0207      0.005     -4.130      0.000      -0.030      -0.011
ma.L7         -0.0023      0.008     -0.305      0.761      -0.017       0.013
ma.L8          0.0190      0.007      2.713      0.007       0.005       0.033
ma.L9         -0.0021      0.005     -0.394      0.694      -0.012       0.008
ma.L10        -0.0095      0.006     -1.716      0.086      -0.020       0.001
sigma2         0.0002   4.03e-07    441.528      0.000       0.000       0.000
===================================================================================
Ljung-Box (L1) (Q):                   0.02   Jarque-Bera (JB):            904313.94
Prob(Q):                              0.89   Prob(JB):                         0.00
Heteroskedasticity (H):               2.42   Skew:                             0.14
Prob(H) (two-sided):                  0.00   Kurtosis:                        28.08
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
count    34497.000000
mean         0.000057
std          0.032953
min         -2.489802
25%         -0.004240
50%          0.000065
75%          0.004105
max          5.013232
```

```
24147    189.115204
24148    170.903557
24149    172.063744
24150    192.328491
24151    188.958658
             ...
34492    188.610863
34493    198.140704
34494    189.635237
34495    183.136066
34496    164.334763
Length: 10350, dtype: float64
Test MSE: 24679.431
```

Summary and conclusion:

We began by cleaning the data to remove missing values and doing a stationarity test to see whether the raw data was stationary. We discovered that the original data was not stationary, so we used a log transform followed by 1st order differencing, then checked for stationarity once again. 1st order differenced data is stationary, according to our findings.

To examine how they perform for the provided data, we developed basic base models such as average, naive, drift, and simple exponential smoothing, as well as Holt winter models. We noticed that none of the base models performed particularly well. Then we displayed the GPAC table and discovered that the auto regressive and moving average orders were na=1 and nb=1, respectively. We also discovered the second pair, na,nb=2,2. We found that ARIMA(3,1,18) is the best model for predicting the Close value of ethereum.

Appendix:

```
import numpy as np
import pandas as pd
import os
from subprocess import check_output
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
from sklearn.model_selection import train_test_split
from statsmodels.tsa.stattools import acf
import statsmodels.api as sm
import statsmodels.tsa.holtwinters as ets
from statsmodels.graphics.tsaplots import plot_pacf,plot_acf
from pandas.plotting import lag_plot
import datetime
from statsmodels.tsa.arima_model import ARIMA
from sklearn.metrics import mean_squared_error
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.seasonal import STL
from helper import  ADF_Cal, kpss_test, Cal_rolling_mean_var, calculate_MA,
plot_detrended, LSE,estimated_variance,calculate_gpac,inverse_diff,plot_gpac
```

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

#loading data
df = pd.read_csv('ETH_1H.csv', header=0, parse_dates=[0], index_col=0,
squeeze=True)
#data pre-processing
print(df.index)
ID=pd.Series(range(0,len(df.Close)))
print(df.columns)
df.set_index(ID,inplace=True)
print(df.index)

print(df.isnull().sum().sort_values(ascending=True))
# no null value found

print(df.nunique(dropna=False))
#converting the Date to day, month, year.
df['Date'] = pd.to_datetime(df['Date'])
df['Month']=df['Date'].dt.month
df['Day']=df['Date'].dt.day
df['Year']=df['Date'].dt.year
df['Hour']=df['Date'].dt.hour
df['Minute']=df['Date'].dt.minute
df['Second']=df['Date'].dt.second
# Data Analysis and Visualization
# Market Capitalization:
# The total dollar market value of a company's outstanding shares of stock is
# referred to as market capitalization. It is computed by multiplying the
entire
# number of a company's outstanding shares by the current market price of one
share,
# which is commonly referred to as "market cap."
df['Market_cap']= df['Open']*df['Volume']

print(df.iloc[df['Market_cap'].argmax()])

# Volatility : In order to KNOW the volatility of the stock,
# we find daily percentage change in the closing price of the stock

df['volatility'] = (df['Close']/df['Close'].shift(1)) - 1

# ploting Month vs High, low, Open, close, volume

def plot_Date_Vs_features(dataset):
        for i in dataset:
            if i == 'Open' or i == 'High' or i == 'Low' or i == 'Close' or i
== 'Market_cap':
                plt.figure(figsize=(20,8))
                plt.plot(dataset['Date'],dataset[i], label=i, color=
'orange')
                plt.title(f'Date vs {i}',fontsize='20')
                plt.xlabel('Date',fontsize='20')
                plt.ylabel(f'price in {i} USD',fontsize='20')
                plt.grid()
                plt.legend()
                plt.show()
```

```
#Plot volume
fig, ax = plt.subplots(figsize=(20,8))
ax.plot(df['Date'], df['Volume'], color='blue')
ax.xaxis.set_major_locator(plt.MaxNLocator(15))
ax.set_xlabel('Date', fontsize='20')
ax.set_ylabel('Volume', fontsize='20')
plt.title('Volume Sold per Hour',fontsize='20')
plt.grid()
plt.show()
#plot volatility
#In order to know the volatility of the stock, we find the daily percentage
change
# in the closing price of the stock.
fig, ax = plt.subplots(figsize=(20,8))
ax.plot(df['Date'], df['volatility'], color='blue')
ax.xaxis.set_major_locator(plt.MaxNLocator(15))
ax.set_xlabel('Date', fontsize='20')
ax.set_ylabel('volatility', fontsize='20')
plt.title('Change in Closing price',fontsize='20')
plt.grid()
plt.show()
# histogram of Volatility
df['volatility'].hist(bins=100, color='blue');

#Cumulative return
# A cumulative return on an investment is the total
# amount gained or lost by the investment throughout time,
# regardless of the length of time involved.

df['Cumulative Return'] = (1 + df['volatility']).cumprod()
fig, ax = plt.subplots(figsize=(20,8))
ax.plot(df['Date'], df['Cumulative Return'], color='red')
ax.xaxis.set_major_locator(plt.MaxNLocator(15))
ax.set_xlabel('Date', fontsize='20')
ax.set_ylabel('Cumulative Return', fontsize='20')
plt.title('Cumulative Return', fontsize='20')
plt.grid()
plt.show()


x=plot_Date_Vs_features(df)
print(x)

# OPEN, CLOSE, HIGH, LOW OF ETH
fig, ax = plt.subplots(figsize=(20,8))
ax.plot(df['Date'], df['Open'], color='Red', label='Open')
ax.plot(df['Date'], df['Close'], color='Green',label='Close')
ax.plot(df['Date'], df['High'], color='Blue', label='High')
ax.plot(df['Date'], df['Low'], color='Yellow', label='Low')
ax.xaxis.set_major_locator(plt.MaxNLocator(15))
ax.set_xlabel('Date', fontsize='20')
ax.set_ylabel('Price in USD', fontsize='20')
plt.title('ETH Prices', fontsize='20')
plt.grid()
plt.legend()
plt.show()
```

```
#Rolling mean and rolling variance
Cal_rolling_mean_var(df.Date,df.Open,'Open')
Cal_rolling_mean_var(df.Date,df.Open,'High')
Cal_rolling_mean_var(df.Date,df.Open,'Low')
Cal_rolling_mean_var(df.Date,df.Open,'Close')
Cal_rolling_mean_var(df.Date,df.Open,'Volume')


# check Dependent variable is stationary or not  Via ADF and Kpss test

print('\n')
print('ADF test of Close Price')
ADF_Cal(df['Close'])
print('\n')
print('KPSS test of Close Price')
kpss_test(df['Close'])
print('\n')

# As p value is greater than 0.05 and ADF statistics are greater than
critical values( 1%, 5% and 10%) we can say that the Close price is non-
stationary
# As p value is smaller than 0.05 and Test statistics are greater than
critical values( 1%, 2.5%, 5% and 10%) we can say that the Close price is
non-stationary

# Log Transformation and 1st order differencing to make the Close price
stationary

df["Close_LogT"]= np.log(df['Close'])
def differencing(dataset):
    diff_close=[]
    for i in range(0,len(dataset)):
        if i==0:
            diff_close.append(df.Close_LogT[0])
        else:
            value= dataset[i]-dataset[i-1]
            diff_close.append(value)
    return diff_close

df["Differential_close"]=differencing(df.Close_LogT)
print('\n')
print('ADF test of Differential_Close Price')
ADF_Cal(df["Differential_close"])
print('\n')
print('KPSS test of Differential_Close Price')
kpss_test(df["Differential_close"])
print('\n')

#differencing made the dependent variable Stationary as per ADF and Kpss test

#ACF and PACF plots before and After differencing
# Before
plot_acf(df['Close'])
plt.xlabel('Lags')
plt.ylabel('ACF of Close')
plt.legend()
plt.show()
```

```
plot_pacf(df['Close'], lags=50)
plt.xlabel('Lags')
plt.ylabel('PACF of CLose')
plt.legend()
plt.show()

#After

plot_acf(df["Differential_close"],label ='After Differencing')
plt.xlabel('Lags')
plt.ylabel('ACF of CLose')
plt.legend()
plt.show()
plot_pacf(df["Differential_close"], lags=50,label ='After Differencing')
plt.xlabel('Lags')
plt.ylabel('PACF of CLose')
plt.legend()
plt.show()

#rolling mean and Rolling variance
#before
Cal_rolling_mean_var(df['Date'],df['Close'],'Close')
#After
Cal_rolling_mean_var(df.Date,df["Differential_close"],'Differential_close')

# #Moving average
MA3_df = calculate_MA(df['Close'], 3, 1, False)
detrended = df['Close'].to_numpy() / MA3_df

plot_detrended(df.Close[0:50], MA3_df[0:50], detrended[0:50], 'Date',
'Close',
               'Eth 3 Months Moving Average', '3-MA')

MA5_df = calculate_MA(df['Close'], 5, 1, False)
detrended = df['Close'].to_numpy() / MA5_df
plot_detrended(df.Close[0:50], MA5_df[0:50], detrended[0:50], 'Time',
'AirPassenger',
               'ETH 5 Months Moving Average', '5-MA')

MA7_df = calculate_MA(df['Close'], 7, 1, False)
detrended = df['Close'].to_numpy() / MA7_df
plot_detrended(df.Close[0:50], MA7_df[0:50], detrended[0:50], 'Time',
'AirPassenger',
               'ETH 7 Months Moving Average', '7-MA')

MA9_df = calculate_MA(df['Close'], 9, 1, False)
detrended = df['Close'].to_numpy() / MA9_df
plot_detrended(df.Close[0:50], MA9_df[0:50], detrended[0:50], 'Time',
'AirPassenger',
               'ETH 9 Months Moving Average', '9-MA')
# Using the function developed in the step 1 plot the estimated cycle-trend
versus the original dataset
# (plot only the first 50 samples) for 2x4-MA, 2x6-MA, 2x8-MA, and 2x10-MA.
Plot the detrended data on
# the same graph. Add an appropriate title, x-label, y-label, and legend to
the graph.
```

```
MA4_df = calculate_MA(df['Close'], 4, 2, False)
detrended = df['Close'].to_numpy() / MA4_df

plot_detrended(df.Close[0:50], MA4_df[0:50], detrended[0:50], 'Time',
'AirPassenger',
               'ETH 2 x 4 Months Moving Average', '2 x 4-MA')

MA6_df = calculate_MA(df['Close'], 6, 2, False)
detrended = df['Close'].to_numpy() / MA6_df
plot_detrended(df.Close[0:50], MA6_df[0:50], detrended[0:50], 'Time',
'AirPassenger',
               'ETH 2 x 6 Months Moving Average', '2 x 6-MA')

MA8_df = calculate_MA(df['Close'], 8, 2, False)
detrended = df['Close'].to_numpy() / MA8_df
plot_detrended(df.Close[0:50], MA8_df[0:50], detrended[0:50], 'Time',
'AirPassenger',
               'ETH 2 x 8 Months Moving Average', '2 x 8-MA')

MA10_df = calculate_MA(df['Close'], 10, 2, False)
detrended = df['Close'].to_numpy() / MA10_df
plot_detrended(df.Close[0:50], MA10_df[0:50], detrended[0:50], 'Time',
'AirPassenger',
               'ETH 2 x 10 Months Moving Average', '2 x 10-MA')

# Compare the ADF-test of the original dataset versus the detrended dataset
using the 3-MA
text = (df['Close'].values)
print(text)

print("----- MA3------------")
detrended = df['Close'].to_numpy() / MA3_df
text = ADF_Cal(detrended[1:-1])
print(text)

#  Apply the STL decomposition method to the dataset. Plot the trend,
seasonality, and reminder in one
# graph. Add an appropriate title, x-label, y-label, and legend to the graph.

plt.figure(figsize=[10, 10])
# STL = STL(data, seasonal=13)
res = STL(df['Close'], period=24)

res = res.fit()
fig = res.plot()

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

plt.figure(figsize=[8, 5])
plt.plot(T[:50], label='trend')
plt.plot(S[:50], label='Seasonal')
plt.plot(R[:50], label='residuals')
plt.title('Time series decomposition')
plt.xlabel('Time')
plt.ylabel('Close')
```

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

# Calculate the seasonally adjusted data and plot it versus the original
data.
# Add an appropriate title, xlabel, y-label, and legend to the graph

seasonal_adjust = df['Close'] - res.seasonal

plt.figure(figsize=[8, 5])
plt.plot_date(df.Date, df['Close'], ls='solid', c='green', label='original',
marker='')
plt.plot_date(df.Date, seasonal_adjust, ls='solid', c='red', label='Seasonal
Adjusted', marker='')
plt.xlabel('Time')
plt.ylabel('Close')
plt.title('Seasonal adjusted data for Close price')
plt.legend()
plt.show()

# Calculate the strength of trend using the following equation and display
# the following message on the console
F_t = max(0, (1 - (np.var(res.resid) / (np.var(np.array(res.trend +
res.resid))))))

print("The strength of trend for this data set is {}".format(F_t))

#  Calculate the strength of seasonality using the following equation and
display
# the following message on the console:

F_t = max(0, (1 - (np.var(res.resid) / (np.var(res.seasonal + res.resid)))))

print("The strength of seasonality for this data set is {}".format(F_t))

#Using the Holt-Winters method try to find the best fit using the train
dataset and make a prediction using the test set.

#spliting the data.
y=df['Close']
yt,yf=train_test_split(y,shuffle=False,test_size=0.3)
lags=2000

def MSE(orignal,predicted):
    error=orignal-predicted
    error2=error**2
    return np.mean(error2)

def var_error(orignal,predicted):
    error=orignal-predicted
    return np.var(error)

def rolling_cal1(series):
    dummy_mean=[]
    for i in range(len(series)):
        dummy_mean.append(np.mean( series.head(i) ))
    return dummy_mean
```

```
def naive_method(series):
    naive=[]
    for i in range(len(series)):
        if i==0:
            naive.append(np.nan)
        else:
            naive.append(series[i-1])
    return naive

def drift_onestep(series):
    drift=[]
    h=1
    for i in range(len(series)):
        if i<=1:
            drift.append(np.nan)
        else:
            drift.append(series[i-1]+h*((series[i-1]-series[0])/(i-1)))
    return drift


def auto_correlation_calculator(series, lags):
    y = np.array(series).copy()
    y_mean = np.mean(series)
    cor = []
    for lag in np.arange(0, lags + 1):
        if lag==0:
            cor.append(1)
        else:
            num1 = y[lag:] - y_mean
            num2 = y[:-lag] - y_mean
            num = sum(num1 * num2)
            den = sum((y - y_mean) ** 2)
            cor.append(num / den)
    return pd.Series(cor)

def Qvalue(series,lags):
    r=auto_correlation_calculator(series,lags)
    rk=r**2
    return len(series)*(np.sum(rk))

def correlation_coefficent_cal(x,y):
    x=np.array(x)
    y=np.array(y)
    numerator=(np.sum((x-np.mean(x))*(y-np.mean(y))))
    denominator= np.sqrt(np.sum((x-np.mean(x))**2)) * np.sqrt(np.sum((y-
np.mean(y))**2))
    r=numerator/denominator
    return r


averagef=[]
for i in range(len(yf)):
    averagef.append(np.mean(yt))


averaget=rolling_cal1(yt)
```

```
fig,ax=plt.subplots()
ax.plot(yt,label='Train Data')
ax.plot(yf,label='Test Data')
ax.plot(np.arange(len(yt),len(yt)+len(yf)),averagef,label='Average Forecast')
ax.set_title('Average method forecast')
ax.set_xlabel('time')
ax.set_ylabel('data')
ax.legend()
plt.show()

average_forecast_mse=MSE(yf,averagef)
print(f'MSE of average method :{average_forecast_mse}')
average_var_pred_error=var_error(yt,averaget)
print(f'variance of prediction error of average
method:{average_var_pred_error}')
average_var_forecast_error=var_error(yf,averagef)
print(f'variance of forecast error of average
method:{average_var_forecast_error}')


error_average=yf-averagef
acf_average = auto_correlation_calculator(error_average, lags)

plt.stem(np.arange(-lags, lags + 1), np.hstack(((acf_average[::-1])[:-
1],acf_average)), linefmt='grey', markerfmt='o')
# plt.stem(np.arange(1, lags + 1), acf_average, linefmt='grey',
markerfmt='o')
plt.title("ACF Plot of Average Forecast Errors")
plt.xlabel("Lags")
plt.ylabel("ACF")
plt.grid()
plt.legend(["ACF"], loc='lower right')
plt.show()

average_qvalue=Qvalue(averagef,lags)
print(f'qvalue of forecast of average method:{average_qvalue}')

average_corr=correlation_coefficent_cal(error_average,yf)
print(f'correlation coefficient of forecast error and test set of average
method:{average_corr}')

# naive method
print("----------------Naive method-----------------")
naivet=naive_method(yt)

naivef=[]
for i in range(len(yf)):
    naivef.append(yt[len(yt)-1])

fig,ax=plt.subplots()
ax.plot(yt,label='Train Data')
ax.plot(yf,label='Test Data')
ax.plot(np.arange(len(yt),len(yt)+len(yf)),naivef,label='Naive Forecast')
ax.set_title('Naive method forecast')
ax.set_xlabel('time')
```

```
ax.set_ylabel('data')
ax.legend()
plt.show()

naive_forecast_mse=MSE(yf,naivef)
print(f'MSE of naive method :{naive_forecast_mse}')
naive_var_pred_error=var_error(yt,naivet)
print(f'variance of prediction error of naive method:{naive_var_pred_error}')
naive_var_forecast_error=var_error(yf,naivef)
print(f'variance of forecast error of naive
method:{naive_var_forecast_error}')

error_naive=yf-naivef
acf_naive = auto_correlation_calculator(error_naive, lags)

plt.stem(np.arange(-lags, lags + 1), np.hstack(((acf_naive[::-1])[:-
1],acf_naive)), linefmt='grey', markerfmt='o')
# plt.stem(np.arange(1, lags + 1), acf_naive, linefmt='grey', markerfmt='o')
plt.title("ACF Plot of Naive Forecast Errors")
plt.xlabel("Lags")
plt.ylabel("ACF")
plt.grid()
plt.legend(["ACF"], loc='lower right')
plt.show()

naive_qvalue=Qvalue(naivef,lags)
print(f'qvalue of forecast of naive method:{naive_qvalue}')

naive_corr=correlation_coefficent_cal(error_naive,yf)
print(f'correlation coefficient of forecast error and test set of naive
method:{naive_corr}')


# drift method

print("---------------Drift method----------------")
drifft=drift_onestep(yt)


driftf=[]
h=len(yf)
for i in range(1,h+1):
    driftf.append(yt[len(yt)-1]+i*((yt[len(yt)-1]-yt[0])/(len(yt)-1)))

fig,ax=plt.subplots()
ax.plot(yt,label='Train Data')
ax.plot(yf,label='Test Data')
ax.plot(np.arange(len(yt),len(yt)+len(yf)),driftf,label='Drift Forecast')
ax.set_title('Drift method forecast')
ax.set_xlabel('time')
ax.set_ylabel('data')
ax.legend()
plt.show()

drift_forecast_mse=MSE(yf,driftf)
print(f'MSE of drift method :{drift_forecast_mse}')
drift_var_pred_error=var_error(yt[2:],drifft[2:])
```

```
print(f'variance of prediction error of drift method:{drift_var_pred_error}')
drift_var_forecast_error=var_error(yf,driftf)
print(f'variance of forecast error of drift
method:{drift_var_forecast_error}')

error_drift=yf-driftf
acf_drift = auto_correlation_calculator(error_drift, lags)

plt.stem(np.arange(-lags, lags + 1), np.hstack(((acf_drift[::-1])[:-
1],acf_drift)), linefmt='grey', markerfmt='o')
# plt.stem(np.arange(1, lags + 1), acf_drift, linefmt='grey', markerfmt='o')
plt.title("ACF Plot of Drift Forecast Errors")
plt.xlabel("Lags")
plt.ylabel("ACF")
plt.grid()
plt.legend(["ACF"], loc='lower right')
plt.show()


drift_qvalue=Qvalue(driftf,lags)
print(f'qvalue of forecast of drift method:{drift_qvalue}')

drift_corr=correlation_coefficent_cal(error_drift,yf)
print(f'correlation coefficient of forecast error and test set of drift
method:{drift_corr}')


# SES
print("---------------SES method----------------")
ses_holtt=ets.ExponentialSmoothing(yt,trend=None,damped_trend=False,seasonal=
None).fit()
ses_holtf=ses_holtt.forecast(steps=len(yf))
ses_holtf=pd.DataFrame(ses_holtf).set_index(yf.index)[0]

ses_prediction = ets.ExponentialSmoothing(yt, trend=None, damped_trend=False,
seasonal=None).fit(smoothing_level=0.5)
ses_prediction_model = ses_prediction.forecast(steps=len(yt))
ses_predictions = pd.DataFrame(ses_prediction_model).set_index(yt.index)[0]


fig,ax=plt.subplots()
ax.plot(yt,label='Train Data')
ax.plot(yf,label='Test Data')
ax.plot(ses_holtf,label='Simple Exponential Smoothing Forecast')
ax.set_title('SES method forecast')
ax.set_xlabel('time')
ax.set_ylabel('data')
ax.legend()
plt.show()

ses_forecast_mse=MSE(yf,ses_holtf.values)
print(f'MSE of SES method :{ses_forecast_mse}')
ses_var_pred_error=var_error(yt,ses_predictions.values)
print(f'variance of prediction error of SES method:{ses_var_pred_error}')
ses_var_forecast_error=var_error(yf,ses_holtf.values)
print(f'variance of forecast error of SES method:{ses_var_forecast_error}')
```

```
error_ses=yf-ses_holtf.values
acf_ses = auto_correlation_calculator(error_ses, lags)

plt.stem(np.arange(-lags, lags + 1), np.hstack(((acf_ses[::-1])[:-
1],acf_ses)), linefmt='grey', markerfmt='o')
# plt.stem(np.arange(1, lags + 1), acf_ses, linefmt='grey', markerfmt='o')
plt.title("ACF Plot of SES Forecast Errors")
plt.xlabel("Lags")
plt.ylabel("ACF")
plt.grid()
plt.legend(["ACF"], loc='lower right')
plt.show()


ses_qvalue=Qvalue(ses_holtf,lags)
print(f'qvalue of forecast of SES method:{ses_qvalue}')
ses_corr=correlation_coefficent_cal(error_ses,yf)
print(f'correlation coefficient of forecast error and test set of SES
method:{ses_corr}')

# Holt linear method

print("---------------Holt linear method----------------")
holtlineart=ets.ExponentialSmoothing(yt,trend='add',damped_trend=False,season
al=None).fit()
holtlinearf=holtlineart.forecast(steps=len(yf))
holtlinearf=pd.DataFrame(holtlinearf).set_index(yf.index)[0]

holt_linear_prediction = ets.ExponentialSmoothing(yt, trend='add',
damped_trend=True, seasonal=None).fit()
holt_linear_prediction_model = holt_linear_prediction.forecast(steps=len(yt))
holt_linear_predictions =
pd.DataFrame(holt_linear_prediction_model).set_index(yt.index)[0]

fig,ax=plt.subplots()
ax.plot(yt,label='Train Data')
ax.plot(yf,label='Test Data')
ax.plot(holtlinearf,label='Holt Linear Forecast')
ax.set_title('Holt X    Linear method forecast')
ax.set_xlabel('time')
ax.set_ylabel('data')
ax.legend()
plt.show()

holtlinear_forecast_mse=MSE(yf,holtlinearf.values)
print(f'MSE of holt linear method :{holtlinear_forecast_mse}')
holtlinear_var_pred_error=var_error(yt,holt_linear_predictions.values)
print(f'variance of prediction error of holt linear
method:{holtlinear_var_pred_error}')
holtlinear_var_forecast_error=var_error(yf,holtlinearf.values)
print(f'variance of forecast error of holt linear
method:{holtlinear_var_forecast_error}')

error_holtlinear=yf-holtlinearf.values
acf_holtlinear = auto_correlation_calculator(error_holtlinear, lags)

plt.stem(np.arange(-lags, lags + 1), np.hstack(((acf_holtlinear[::-1])[:-
```

```
1],acf_holtlinear)), linefmt='grey', markerfmt='o')
# plt.stem(np.arange(1, lags + 1), acf_holtlinear, linefmt='grey',
markerfmt='o')
plt.title("ACF Plot of Holt Linear Forecast Errors")
plt.xlabel("Lags")
plt.ylabel("ACF")
plt.grid()
plt.legend(["ACF"], loc='lower right')
plt.show()


holtlinear_qvalue=Qvalue(holtlinearf,lags)
print(f'qvalue of forecast of Holt linear method:{holtlinear_qvalue}')

holtlinear_corr=correlation_coefficent_cal(error_holtlinear,yf)
print(f'correlation coefficient of forecast error and test set of holt linear
method:{holtlinear_corr}')



# Holt Winter method

print("---------------Holt winter method----------------")

holt_wintert=ets.ExponentialSmoothing(yt,trend='mul',damped_trend=True,season
al='mul',seasonal_periods=24).fit()
holt_winterf=holt_wintert.forecast(steps=len(yf))
holt_winterf=pd.DataFrame(holt_winterf).set_index(yf.index)[0]

holt_winter_prediction = ets.ExponentialSmoothing(yt, trend='add',
damped_trend=True, seasonal='mul',seasonal_periods=24).fit()
holt_winter_prediction_model = holt_winter_prediction.forecast(steps=len(yt))
holt_winter_predictions =
pd.DataFrame(holt_winter_prediction_model).set_index(yt.index)[0]

fig,ax=plt.subplots()
ax.plot(yt,label='Train Data')
ax.plot(yf,label='Test Data')
ax.plot(holt_winterf,label='Holt Winter Forecast')
ax.set_title('Holt Winter method forecast')
ax.set_xlabel('time')
ax.set_ylabel('data')
ax.legend()
plt.show()

holtwinter_forecast_mse=MSE(yf,holt_winterf.values)
print(f'MSE of holt winter method :{holtwinter_forecast_mse}')
holtwinter_var_pred_error=var_error(yt,holt_winter_predictions.values)
print(f'variance of prediction error of holt winter
method:{holtwinter_var_pred_error}')
holtwinter_var_forecast_error=var_error(yf,holt_winterf.values)
print(f'variance of forecast error of holt winter
method:{holtwinter_var_forecast_error}')

error_holtwinter=yf-holt_winterf.values
acf_holtwinter = auto_correlation_calculator(error_holtwinter, lags)
```

```
plt.stem(np.arange(-lags, lags + 1), np.hstack(((acf_holtwinter[::-1])[:-
1],acf_holtwinter)), linefmt='grey', markerfmt='o')
# plt.stem(np.arange(1, lags + 1), acf_holtwinter, linefmt='grey',
markerfmt='o')
plt.title("ACF Plot of Holt Winter Forecast Errors")
plt.xlabel("Lags")
plt.ylabel("ACF")
plt.grid()
plt.legend(["ACF"], loc='lower right')
plt.show()


holtwinter_qvalue=Qvalue(holt_winterf,lags)
print(f'qvalue of forecast of Holt winter method:{holtwinter_qvalue}')

holtwinter_corr=correlation_coefficent_cal(error_holtwinter,yf)
print(f'correlation coefficient of forecast error and test set of holt winter
method:{holtwinter_corr}')

print("-------------RESULT ----------------------")
result_table = pd.DataFrame(
    columns=["Method", "Forecast_MSE", "Prediction_Var", "Forecast_Var",
"Qvalue", "Corr"])
result_table["Method"] = ["Average", "Naive", "Drift", "SES", "Holt Linear",
"Holt-Winter"]
result_table["Forecast_MSE"] = [average_forecast_mse,
                          naive_forecast_mse,
                          drift_forecast_mse,
                          ses_forecast_mse,
                          holtlinear_forecast_mse,
                          holtwinter_forecast_mse]

result_table["Prediction_Var"] = [average_var_pred_error,
                            naive_var_pred_error,
                            drift_var_pred_error,
                            ses_var_pred_error,
                            holtlinear_var_pred_error,
                            holtwinter_var_pred_error]

result_table["Forecast_Var"] = [average_var_forecast_error,
                          naive_var_forecast_error,
                          drift_var_forecast_error,
                          ses_var_forecast_error,
                          holtlinear_var_forecast_error,
                          holtwinter_var_forecast_error]
result_table["Qvalue"] = [average_qvalue,
                    naive_qvalue,
                    drift_qvalue,
                    ses_qvalue,
                    holtlinear_qvalue,
                    holtwinter_qvalue]

result_table["Corr"] = [average_corr,
                  naive_corr,
                  drift_corr,
                  ses_corr,
                  holtlinear_corr,
```

```
                  holtwinter_corr]
print(result_table)

#multiple linear regression and Feature selection:

print(df.head(5))
# split the dataset into test and train
df
columns_drop=df[['Symbol','Date','Close','Day','Month','Year','Hour','Minute'
,'Second','volatility','Close_LogT','Differential_close','Cumulative
Return']]
x= df.drop(columns_drop,axis=1)
print(x)
y=df['Close']
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,
random_state=100)

# 2: plotting the correlation
plt.figure(figsize=(26, 16))
heatmap = sns.heatmap(df.corr(), vmin=-1, vmax=1, annot=True)
heatmap.set_title('Correlation Heatmap of ETH.1h dataset',
fontdict={'fontsize':20}, pad=12)
plt.tight_layout()
plt.show()

# 3: Collinearity detection:
# a.) Perform SVD analysis on the original feature space
C=np.dot(x.T,x)
U,S,VH=np.linalg.svd(C)
print('all singular values will be: ',list(S))
# We can see that two singular values are close to 0, indicating that there
is co-linearity between two or more features.

# b.)Calculate the condition number and write down your observation if co-
linearity exists. Justify your answer.
condition_number_cal= np.linalg.cond(x)
print('Condition number: ', condition_number_cal)

# We can see that the condition number is larger than 1000, indicating that
the degree of co-linearity is severe.
# c.) # C.) If collinearity exist, how many features will you remove to avoid
the co-linearity
# The number of features to be deleted is determined by doing a
backward/forward selection process and evaluating the t-test p values,
# then eliminating features one at a time until the modified R square, AIC,
and BIC values show a significant change.

#4: use the x-train and y-train dataset and estimate the regression model
unknown coefficients using the Normal equation
print('Unknown coefficients check: ',list(LSE(x_train,y_train)))
# #question 5: statsmodels package and OLS function to find the unknown
coefficients
#
modeling = sm.OLS(y_train,x_train).fit()
print(modeling.summary())

# We may state that the unknown coefficient results determined using steps 4
```

```
and 5 are same
# because the unknown coefficient results are identical.
# We should remove Volume from the equation.


# 6 Feature selection: Using a backward stepwise regression, reduce the
feature space dimension.
# use of AIC, BIC and Adjusted R2 as a predictive accuracy for your analysis
#
# dropping Volume from dataset

x_train1=x_train.drop(['Volume'],axis=1)
model0=sm.OLS(y_train,x_train1).fit()
print(model0.summary())

# dropping Market_cap from dataset

x_train1=x_train.drop(['Volume','Market_cap'],axis=1)
model1=sm.OLS(y_train,x_train1).fit()
print(model1.summary())

# dropping Open from dataset

x_train1=x_train.drop(['Volume','Market_cap','Open'],axis=1)
model2=sm.OLS(y_train,x_train1).fit()
print(model2.summary())
# # Because the r-squared, AIC, and BIC values change a lot when we tweak
them, we can keep the stroke and call it a day.
#
#7: the reduced feature spaced and display the final model.
# # Final Model is without Volume the Market_cap
#
# print('<========> FINAL MODEL <==========>')
x_train1=x_train.drop(['Volume','Market_cap'],axis=1)
model1=sm.OLS(y_train,x_train1).fit()
print(model1.summary())
#
#8: a prediction for the length of test-set and plot the train, test, and
predicted values in one graph.
column_to_drop = x_test[['Volume','Market_cap']]
x_test1=x_test.drop(column_to_drop, axis= 1)
#prediction
y_prediction = model1.predict(x_test1)

plt.figure(figsize = (10, 10))
plt.plot(y_train,label = 'y_train')
plt.plot(y_test,label = 'y_test')
plt.plot(y_prediction,label = 'y_prediction')
plt.title('Prediction of ETH1 dataset',fontsize='20')
plt.xlabel('observation',)
plt.ylabel('Close price in USD',fontsize='20')
plt.grid()
plt.legend()
plt.show()

# 9 the prediction errors and plot the ACF of prediction errors.
y_predictor = model1.predict(x_train1)
```

```
y_prediction_error = y_train-y_predictor
lags=20
ACF1 = auto_correlation_calculator(y_prediction_error, lags)
x=np.arange(-lags, lags + 1)
y=np.hstack(((ACF1[::-1])[:-1],ACF1))
plt.stem(x,y, linefmt='grey', markerfmt='o')
plt.title("ACF Plot prediction Errors",fontsize='20')
plt.xlabel("Lags",fontsize='20')
plt.ylabel("ACF",fontsize='20')
plt.grid()
plt.legend(["ACF"], loc='upper right')
plt.show()

# # We can see that the auto correlation is insignificant after  1 delay.
#
#10  the forecast errors and plot the ACF of forecast errors.
forecast_error=y_test-y_prediction
lags=20
ACF2 = auto_correlation_calculator(forecast_error, lags)
x=np.arange(-lags, lags + 1)
y=np.hstack(((ACF2[::-1])[:-1],ACF2))
plt.stem(x,y , linefmt='grey', markerfmt='o')
plt.title("ACF Plot of forecast Errors",fontsize='20')
plt.xlabel("Lags",fontsize='20')
plt.ylabel("ACF",fontsize='20')
plt.grid()
plt.legend(["ACF"], loc='upper right')
plt.show()

# # We can see that the auto correlation is insignificant after only 1 lag.


# 11: the estimated variance of the prediction errors and the forecast
errors.
#prediction error
prediction_error=estimated_variance(y_prediction_error,len(x_train1),len(x_tr
ain1.columns))
print('estimated variance of prediction error :',prediction_error)

#forecast error
forecast_error_Check=estimated_variance(forecast_error,len(x_train1),len(x_tr
ain1.columns))
print('estimated variance of forecast error :',forecast_error_Check)

# We can see that the expected variance of prediction and forecast errors are
rather similar.

#12
r1 = np.identity(len(model1.params))
t_test1=model1.t_test(r1)
print('t test final model:', t_test1)
print('p-value of TTest:',t_test1.pvalue)

# because the p-value is less than 5%, reject the null hypothesis # because
the regression coefficients are not equal to 0

f_test=model1.f_test(r1)
```

```
print('f test of final model:',f_test)
print('p-value of Ftest:',f_test.pvalue)

# we can observe that p-value is smaller that 5 percent , thus we reject the
null hypothesis\s# and conclude that your model gives a better fit than the
intercept-only model
yt1,yf1 = train_test_split(df.Differential_close,shuffle=False,test_size=0.2)
dtrain,dtest = train_test_split(df.Date,shuffle=False,test_size=0.2)
#  ================================ ARMA Process
================================

print('========================= ARMA (1,1) model
================================')

acf_y2 = auto_correlation_calculator(df.Differential_close,100)
plot_gpac(calculate_gpac(acf_y2,7,7), "GPAC ARMA (1,1)")


na = 1
nb = 1

model = sm.tsa.ARIMA(yt1, order=(1,0,1)).fit()
print(model.summary())
for i in range(na):
    print(f'The AR Coefficient : a{i+1} is:, {model.params[i]}')

for i in range(nb):
    print(f'The MA Coefficient : b{i+1} is:, {model.params[i+na]}')

# ========== 1 step prediction ================
# def inverse_diff(y20,z_hat,interval=1):
#     y_new = np.zeros(len(y20))
#     for i in range(1,len(z_hat)):
#         y_new[i] = z_hat[i-interval] + y20[i-interval]
#     y_new = y_new[1:]
#     return y_new

def inverse_log_diff(y20,z_hat,interval=1):
    y_new = np.zeros(len(y20))
    for i in range(1,len(z_hat)):
        y_new[i] = np.exp(z_hat[i-interval] + y20[i-interval])
    y_new = y_new[1:]
    return y_new

model_hat = model.predict(start=0,end=len(yt1)-1)

# res_arma_error = np.array(yt) - np.array(model_hat)

y_hat =
inverse_log_diff(df.Differential_close[:len(yt1)].values,np.array(model_hat),
1)

res_arma_error = df.Differential_close[:len(yt1)-1] - y_hat

lags=100
```

```
plot_acf(res_arma_error,lags=lags)
plt.title('ACF of the residual error (ARMA(1,1))')
plt.legend()
plt.show()
plot_pacf(res_arma_error,lags=lags)
plt.title('PACF of the residual error (ARMA(1,1))')
plt.legend()
plt.show()
acf_res = acf(res_arma_error, nlags= lags)
plt.stem(np.arange(-lags, lags+1 ), np.hstack(((acf_res[::-1])[:-
1],acf_res)), linefmt='grey', markerfmt='o')
m = 1.96 / np.sqrt(100)
plt.axhspan(-m, m, alpha=.2, color='blue')
plt.title("ACF Plot of residual error (ARMA(1,1))")
plt.xlabel("Lags")
plt.ylabel("ACF values")
plt.grid()
plt.legend(["ACF"], loc='upper right')
plt.show()


plt.plot(df.Date[:99],df.Differential_close[:99], label = 'train set')
plt.plot(df.Date[:99],y_hat[:99], label = '1-step prediction')
plt.title('Close vs time(ARMA(1,1)) - prediction plot')
plt.xlabel('time')
plt.ylabel('Close')
plt.legend()
plt.tight_layout()
plt.show()

# diagnostic testing
from scipy.stats import chi2

print('confidence intervals of estimated parameters:',model.conf_int())

poles = []
for i in range(na):
    poles.append(-(model.params[i]))

print('zero/cancellation:')
zeros = []
for i in range(nb):
    zeros.append(-(model.params[i+na]))

print(f'zeros : {zeros}')
print(f'poles : {poles}')

Q = len(yt)*np.sum(np.square(acf_res[lags:]))

DOF = lags-na-nb

alfa = 0.01

chi_critical = chi2.ppf(1-alfa, DOF)

print('Chi Squared test results')
```

```
if Q<chi_critical:
    print(f'The residuals is white, chi squared value :{Q}')
else:
    print(f'The residual is NOT white, chi squared value :{Q}')

# ========== h step prediction ===============
forecast = model.forecast(steps=len(yf1))

y_arma_pred = pd.Series(forecast.iloc[0], index=yf1.index)

y_hat_fore =
inverse_log_diff(df.Differential_close[len(yt1):].values,np.array(y_arma_pred
),1)

# h step prediction

plt.plot(dtest[1:],df.Differential_close[(len(yt1)+1):].values.flatten(),
label='Test Data')
plt.plot(dtest[1:],y_hat_fore, label='ARMA Method Forecast')
plt.title('CLose vs time(ARMA(1,1)) - prediction plot')
plt.xlabel('time')
plt.ylabel('Close')
plt.legend()
plt.show()

res_arma_forecast = df.Differential_close[len(yt1)+1:] - y_hat_fore

print(f'variance of residual error : {np.var(res_arma_error)}')

print(f'variance of forecast error : {np.var(res_arma_forecast)}')

# ========================== 2nd ARMA model na=4, nb=4
==========================
print('========================== ARMA (2,2) model
================================')

na = 2
nb = 2

model1 = sm.tsa.ARIMA(yt1, order=(2,0,2)).fit()
print(model1.summary())

for i in range(na):
    print(f'The AR Coefficient : a{i+1} is:, {model1.params[i]}')

for i in range(nb):
    print(f'The MA Coefficient : b{i+1} is:, {model1.params[i+na]}')


#================ 1 step prediction ====================
model_hat1 = model1.predict(start=0,end=len(yt1)-1)

# res_arma_error = np.array(yt) - np.array(model_hat)

y_hat1 =
inverse_log_diff(df.Differential_close[:len(yt1)].values,np.array(model_hat1)
,1)
```

```
res_arma_error1 = df.Differential_close[:len(yt1)-1] - y_hat1

lags=100

plot_acf(res_arma_error,lags=lags)
plt.title('ACF of the residual error (ARMA(2,2))')
plt.legend()
plt.show()
plot_pacf(res_arma_error,lags=lags)
plt.title('PACF of the residual error (ARMA(2,2))')
plt.legend()
plt.show()

acf_res = acf(res_arma_error1, nlags= lags)
plt.stem(np.arange(-lags, lags+1 ), np.hstack(((acf_res[::-1])[:-
1],acf_res)), linefmt='grey', markerfmt='o')
m = 1.96 / np.sqrt(100)
plt.axhspan(-m, m, alpha=.2, color='blue')
plt.title("ACF Plot of residual error (ARMA(2,2))")
plt.xlabel("Lags")
plt.ylabel("ACF values")
plt.grid()
plt.legend(["ACF"], loc='upper right')
plt.show()


plt.plot(df.Date[:99],df.Differential_close[:99], label = 'train set')
plt.plot(df.Date[:99],y_hat1[:99], label = '1-step prediction')
plt.title('Close vs time(ARMA(2,2)) - prediction plot')
plt.xlabel('time')
plt.ylabel('Close')
plt.legend()
plt.tight_layout()
plt.show()

# diagnostic testing
from scipy.stats import chi2

print('confidence intervals of estimated parameters:',model1.conf_int())

poles = []
for i in range(na):
    poles.append(-(model1.params[i]))

print('zero/cancellation:')
zeros = []
for i in range(nb):
    zeros.append(-(model1.params[i+na]))

print(f'zeros : {zeros}')
print(f'poles : {poles}')

Q = len(yt1)*np.sum(np.square(acf_res[lags:]))

DOF = lags-na-nb
```

```
alfa = 0.01

chi_critical = chi2.ppf(1-alfa, DOF)

print('Chi Squared test results')

if Q<chi_critical:
    print(f'The residuals is white, chi squared value :{Q/100}')
else:
    print(f'The residual is NOT white, chi squared value :{Q/100}')

# ========== h step prediction ===============

forecast1 = model1.forecast(steps=len(yf1))

y_arma_pred1 = pd.Series(forecast1.iloc[0], index=yf1.index)

y_hat_fore1 = inverse_diff(y[len(yt1):].values,np.array(y_arma_pred1),1)

# h step prediction

plt.plot(dtest[1:],df.Differential_close[len(yt1)+1:].values.flatten(),
label='Test Data')
plt.plot(dtest[1:],y_hat_fore1, label='ARMA Method Forecast')
plt.title('Close vs time(ARMA(2,2)) - prediction plot')
plt.xlabel('time')
plt.ylabel('CLose')
plt.legend()
plt.show()

res_arma_forecast1 = df.Differential_close[len(yt1)+1:] - y_hat_fore1

print(f'variance of residual error : {np.var(res_arma_error1)}')

print(f'variance of forecast error : {np.var(res_arma_forecast1)}')
#===============ARIMA order
(3,1,18)=============================================================

df_train, df_test = train_test_split(df.Close, shuffle=False, test_size=0.3)

print('train shape :', df_train.shape)
print('validation shape :', df_test.shape)
Rolling_average = df["Close_LogT"].rolling(window = 7, center= False).mean()
log_Rolling_difference= df["Close_LogT"] - Rolling_average
log_Rolling_difference.dropna(axis=0,inplace=True)
from statsmodels.tsa.stattools import acf, pacf

#ACF and PACF plots:
lag_acf = acf(log_Rolling_difference, nlags=100)
lag_pacf = pacf(log_Rolling_difference, nlags=100, method='ols')
log_Rolling_difference = log_Rolling_difference.fillna(0)
model = sm.tsa.ARIMA(df["Close_LogT"], order=(3, 1, 0))
results_AR = model.fit()
plt.plot(log_Rolling_difference, label ='Close')
plt.plot(results_AR.fittedvalues, color='red', label = 'order 3')
RSS = results_AR.fittedvalues-log_Rolling_difference
RSS.dropna(inplace=True)
```

```
plt.title('RSS: %.4f'% sum(RSS**2))
plt.legend(loc = 'best')
#MA model
model = sm.tsa.ARIMA(df["Close_LogT"], order=(0, 1, 18))
results_MA = model.fit()
plt.plot(log_Rolling_difference)
plt.plot(results_MA.fittedvalues, color='red')
RSS = results_MA.fittedvalues-log_Rolling_difference
RSS.dropna(inplace=True)
plt.title('RSS: %.4f'% sum(RSS**2))
print(results_MA.summary())
plt.plot(df["Close_LogT"], label = 'log_tranfromed_data')
plt.plot(results_MA.resid, color ='green',label= 'Residuals')
plt.title('MA Model Residual plot')
plt.legend(loc = 'best')
results_MA.resid.plot(kind='kde')
plt.title('Density plot of the residual error values')
print(results_MA.resid.describe())

#ARIMA Combined model
model = sm.tsa.ARIMA(df["Close_LogT"], order=(3, 1, 18))
results_ARIMA = model.fit()
plt.plot(log_Rolling_difference)
plt.plot(results_ARIMA.fittedvalues, color='red', label = 'p =8, q =18')
RSS =results_ARIMA.fittedvalues-log_Rolling_difference
RSS.dropna(inplace=True)
plt.title('RSS: %.4f'% sum(RSS**2))
plt.legend(loc='best')
print(results_ARIMA.summary())
plt.plot(df["Close_LogT"], label = 'log_tranfromed_data')
plt.plot(results_ARIMA.resid, color ='green',label= 'Residuals')
plt.title('ARIMA Model Residual plot')
plt.legend(loc = 'best')

results_ARIMA.resid.plot(kind='kde')
plt.title('Density plot of the residual error values')
print(results_ARIMA.resid.describe())

train, test = train_test_split(df,shuffle=False,test_size=0.3)
Test_y=test.Close
test.drop(columns='Close', inplace=True)

predictions_ARIMA_diff = pd.Series(results_ARIMA.fittedvalues, copy=True)
print(predictions_ARIMA_diff.head())
predictions_ARIMA_diff_cumsum = predictions_ARIMA_diff.cumsum()
print(predictions_ARIMA_diff_cumsum.head())
predictions_ARIMA_log = pd.Series(df.Close_LogT.iloc[0],
index=df.Close_LogT.index)
predictions_ARIMA_log =
predictions_ARIMA_log.add(predictions_ARIMA_diff_cumsum,fill_value=0)
predictions_ARIMA_log.head()

predictions_ARIMA = np.exp(predictions_ARIMA_log)
plt.plot(df.Close,label='Actual Close')
plt.plot(predictions_ARIMA, label='Predicted Close')
plt.title('RMSE: %.4f'% np.sqrt(sum((predictions_ARIMA-
df.Close)**2)/len(df.Close)))
```

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

from sklearn.metrics import mean_squared_error
dates=test.index
forecast = pd.Series(results_ARIMA.forecast(steps=10350).iloc[0],dates)
forecast = 10**(forecast+test.Differential_close)
print(forecast)
error = mean_squared_error(Test_y, forecast)
print('Test MSE: %.3f' % error)


#============================Arima order(2,2,10)=============

#ARIMA Combined model
model = sm.tsa.ARIMA(df["Close_LogT"], order=(2, 2, 0))
results_AR = model.fit()
plt.plot(log_Rolling_difference, label ='Close')
plt.plot(results_AR.fittedvalues, color='red', label = 'order 3')
RSS = results_AR.fittedvalues-log_Rolling_difference
RSS.dropna(inplace=True)
plt.title('RSS: %.4f'% sum(RSS**2))
plt.legend(loc = 'best')
#MA model
model = sm.tsa.ARIMA(df["Close_LogT"], order=(0, 2, 10))
results_MA = model.fit()
plt.plot(log_Rolling_difference)
plt.plot(results_MA.fittedvalues, color='red')
RSS = results_MA.fittedvalues-log_Rolling_difference
RSS.dropna(inplace=True)
plt.title('RSS: %.4f'% sum(RSS**2))
print(results_MA.summary())
plt.plot(df["Close_LogT"], label = 'log_tranfromed_data')
plt.plot(results_MA.resid, color ='green',label= 'Residuals')
plt.title('MA Model Residual plot')
plt.legend(loc = 'best')
results_MA.resid.plot(kind='kde')
plt.title('Density plot of the residual error values')
print(results_MA.resid.describe())

#ARIMA Combined model
model = sm.tsa.ARIMA(df["Close_LogT"], order=(2, 2, 10))
results_ARIMA = model.fit()
plt.plot(log_Rolling_difference)
plt.plot(results_ARIMA.fittedvalues, color='red', label = 'p =8, q =18')
RSS =results_ARIMA.fittedvalues-log_Rolling_difference
RSS.dropna(inplace=True)
plt.title('RSS: %.4f'% sum(RSS**2))
plt.legend(loc='best')
print(results_ARIMA.summary())
plt.plot(df["Close_LogT"], label = 'log_tranfromed_data')
plt.plot(results_ARIMA.resid, color ='green',label= 'Residuals')
plt.title('ARIMA Model Residual plot')
plt.legend(loc = 'best')

results_ARIMA.resid.plot(kind='kde')
plt.title('Density plot of the residual error values')
```

```
print(results_ARIMA.resid.describe())

train, test = train_test_split(df,shuffle=False,test_size=0.3)
Test_y=test.Close
test.drop(columns='Close', inplace=True)

predictions_ARIMA_diff = pd.Series(results_ARIMA.fittedvalues, copy=True)
print(predictions_ARIMA_diff.head())
predictions_ARIMA_diff_cumsum = predictions_ARIMA_diff.cumsum()
print(predictions_ARIMA_diff_cumsum.head())
predictions_ARIMA_log = pd.Series(df.Close_LogT.iloc[0],
index=df.Close_LogT.index)
predictions_ARIMA_log =
predictions_ARIMA_log.add(predictions_ARIMA_diff_cumsum,fill_value=0)
predictions_ARIMA_log.head()

predictions_ARIMA = np.exp(predictions_ARIMA_log)
plt.plot(df.Close,label='Actual Close')
plt.plot(predictions_ARIMA, label='Predicted Close')
plt.title('RMSE: %.4f'% np.sqrt(sum((predictions_ARIMA-
df.Close)**2)/len(df.Close)))
plt.legend()
plt.show()

from sklearn.metrics import mean_squared_error
dates=test.index
forecast = pd.Series(results_ARIMA.forecast(steps=10350).iloc[0],dates)
forecast = 10**(forecast+test.Differential_close)
print(forecast)
error = mean_squared_error(Test_y, forecast)
print('Test MSE: %.3f' % error)
```

References:

1. Lectures and professor notes and Homework

2. Dataset from Kaggle : https://www.kaggle.com/prasoonkottarathil/ethereum-historical-dataset?select=ETH_1H.csv