DATS 6313
Final Term Project Report
Apple Stock price Predicting

Aravind Chinnari
Professor : Reza Jafari
The George Washington University

Table of Contents

Abstract:

This project aims to forecast the opening price of Apple Inc. stock through time series analysis using historical stock market data spanning over four decades. The dataset comprises daily stock prices, encompassing essential metrics like Open, High, Low, Close, Adjusted Close, Volume, and corresponding dates.

The primary focus is on predicting the "Open" price, crucial for understanding potential price trends, facilitating informed trading strategies, and aiding in risk management. The proposed analysis involves rigorous exploration of the dataset through exploratory data analysis (EDA), pre-processing, and model selection phases.

Through EDA, patterns, correlations, and inherent characteristics within the data will be explored, offering insights into the stock's historical performance. Pre-processing steps will address missing values, feature engineering, and potential normalization or scaling requirements for model training. Model selection will involve the exploration and evaluation of various time series forecasting techniques to identify the most accurate predictor of Apple's stock opening prices.

The project's significance lies in its potential to provide a predictive model that assists investors, analysts, and stakeholders in making informed decisions based on future Apple Inc. stock price trends. The comprehensive report and presentation will encapsulate the analysis, methodologies employed, and the efficacy of the forecasting models, providing valuable insights into the dynamics of stock price prediction through time series analysis.

Introduction:

This project aims to employ advanced time series forecasting techniques to predict the opening price of Apple Inc. stock using historical market data spanning more than four decades. To achieve this goal, several critical steps are undertaken, starting with comprehensive data cleaning procedures. Missing values within the dataset are imputed using the mean of their respective columns, ensuring a complete and consistent dataset for analysis.

An essential aspect of this analysis involves assessing the stationarity of the target variable. This evaluation is conducted through rigorous statistical tests such as Autocorrelation Function (ACF) plots, Augmented Dickey-Fuller (ADF), and Kwiatkowski-Phillips-Schmidt-Shin (KPSS) tests. If the target variable is found to be non-stationary, differential transformations are applied to induce stationarity, ensuring the suitability of the data for time series modeling.

To establish baseline performance metrics, fundamental forecasting methods like the average method, naive method, drift method, Simple Exponential Smoothing (SES) method, and Holt-Winters methods are employed. These serve as benchmarks to gauge the effectiveness of more complex models like ARMA/ARIMA.

Feature reduction and initial predictions are also conducted using Ordinary Least Squares (OLS) regression, providing insights into feature importance and potential linear relationships within the data.

The identification of optimal ARMA/ARIMA model parameters is facilitated through the Generalized Partial Autocorrelation (GPAC) table, aiding in the determination of suitable lag orders (na, nb).

Diagnostic tests are rigorously performed on the ARMA/ARIMA models to ensure their adequacy for accurate forecasting. Criteria such as residual mean, variance, and forecast error are employed to select the most fitting model. The selected model is then utilized to generate forecasts, which are subsequently compared with test values to evaluate performance.

By systematically undertaking these steps and leveraging diverse forecasting methodologies, this project seeks to identify the most effective model for predicting Apple Inc. stock prices, offering valuable insights into future price trends for informed decision-making in financial markets.

[source: Time series analysis lecture notes]

The following are the theories behind the methods we used in this project.

Autocorrelation:

Autocorrelation, a fundamental concept in time series analysis, assesses the relationship between a series of observations at different time intervals. Essentially, it quantifies the similarity or correlation between a data point and its lagged counterparts within the same series.

Symbolized as Tk, autocorrelation measures the association between a particular observation at time 't' (yt) and its value at a prior time point, 't - k' (yt-k). The estimated autocorrelation function, denoted as Ry(t), is particularly pertinent for stationary time series datasets.

This analysis provides critical insights into the pattern and dependency structure inherent within the time series.

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

Average method:

The average method in time series forecasting simplistically predicts future values by assigning them the same value as the average of historical data. It assumes a constant trend, making forecasts solely based on the mean of past observations. While straightforward, its simplicity might limit accuracy, especially when the data exhibits complex patterns or fluctuations.

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

The average method treats all the observations with equal importance. It gives equal weights while forecasting.

Naive method:

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.

$$\hat{y}_{T+h|T} = y_T$$

for $h = 1, 2, ...$

Drift method:

The drift method in time series forecasting incorporates a trend by considering the average change over time. It assumes a linear progression, accounting for a gradual shift or slope in the series. This method adds a trend component to predictions beyond simply using the most recent value, providing a more nuanced forecast by considering the overall direction of the data.

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

(SES) Simple exponential smoothing method:

SES Simple exponential smoothing computes weighted averages, where the weights diminish exponentially as observations move further into the past. Older observations receive the smallest weights in this method, emphasizing recent data while gradually decreasing the impact of historical data on the forecasts.

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

Holt Winter method:

The Holt-Winters seasonal method, an extension of Holt's method by Holt in 1975 and Winter in 1960, accommodates seasonal patterns in time series forecasting. It encompasses a forecast equation and three smoothing equations–level (ℓt), trend (bt), and seasonal (st). the forecast yt+h|t at time 't' plus 'h' periods ahead is composed of these components, accounting for both trend and seasonality. The equations update the level, trend, and seasonal components iteratively, considering weighted combinations of the observed value, previous level and trend, and deviations from the level and trend to estimate future values while incorporating seasonal patterns with a seasonal parameter 'm'.

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

Where, lt - level, bt - trend, st - seasonality

ARMA model:

Analysis of an adaptive time-series autoregressive moving-average (ARMA) model for short-term load forecasting.

The ARMA model, derived from Autoregressive (AR) and Moving Average (MA) models, characterizes a stationary stochastic process by combining these two components into a unified framework.

The Autoregressive (AR) component in the ARMA model involves regressing the variable against its own past values, capturing the relationship between the variable and its lagged observations. This component reflects how the variable's current value depends linearly on its previous values, emphasizing temporal patterns and dependencies within the time series.

On the other hand, the Moving Average (MA) component models the error term as a linear combination of the error terms occurring simultaneously and at different points in time in the past. It focuses on explaining the current value of the variable based on the error terms from previous time points, capturing short-term irregularities or shocks in the data.

The ARMA model, combining these AR and MA components, offers a powerful tool to analyze and forecast time series data by accounting for both the auto-correlation of the series and the moving average of past errors. Its ability to capture both short-term dynamics and long-term trends makes it a versatile and widely used model in time series analysis and forecasting.

General form of ARMA:

$$y(t) + a_1 y(t-1) + a_2 y(t-2) + \ldots + a_{n_a} y(t-n_a) = \epsilon(t) + b_1 \epsilon(t-1) + b_2 \epsilon(t-2) + \ldots + b_{n_b} \epsilon(t-n_b)$$

ARIMA model:

Interrupted time series analysis using autoregressive integrated moving average (ARIMA) models: a guide for evaluating large-scale health interventions.

The Autoregressive Integrated Moving Average (ARIMA) model, an extension of the Autoregressive Moving Average (ARMA) model, serves as a comprehensive tool for time series analysis and forecasting. When confronted with non-stationary data in terms of the mean, ARIMA models step in to address this by incorporating differencing–hence the "integrated" component. The "integrated" part of the ARIMA model pertains to applying differencing one or more times to the original time series data. This differencing operation transforms the data to achieve stationarity in the mean function, essential for effectively modeling and understanding the underlying patterns within the series. By iteratively differencing the series, ARIMA handles non-stationary mean behavior, thereby enabling the utilization of autoregressive and moving average components for modeling and forecasting purposes.

Structure of ARIMA:

$$(1 + a_1 q^{-1} + \ldots + a_{n_a} q^{-n_a})(1 - q^{-1})^d y(t) = (1 + b_1 q^{-1} + \ldots + b_{n_b} q^{-n_b}) \epsilon(t)$$

Detail Description of the Dataset:

The dataset comprises 10,468 daily observations detailing Apple stock prices from the expansive period spanning 1980 to 2022. This extensive dataset encapsulates the daily fluctuations and trends within the stock's pricing behavior over more than four decades.

a. Pre-processing dataset:

The columns of this dataset are:

'Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume'

The Target variable in this data set is `'open'` which refers to open price of daily for apple stock.

Basic description of the Dataset:

	Date	Open	High	Low	Close
count	10468	10468.000000	10468.000000	10468.000000	10468.000000
unique	10468	NaN	NaN	NaN	NaN
top	1980-12-12	NaN	NaN	NaN	NaN
freq	1	NaN	NaN	NaN	NaN
mean	NaN	14.757987	14.921491	14.594484	14.763533
std	NaN	31.914174	32.289158	31.543959	31.929489
min	NaN	0.049665	0.049665	0.049107	0.049107
25%	NaN	0.283482	0.289286	0.276786	0.283482
50%	NaN	0.474107	0.482768	0.465960	0.475446
75%	NaN	14.953303	15.057143	14.692589	14.901964
max	NaN	182.630005	182.940002	179.119995	182.009995

	Adj Close	Volume
count	10468.000000	1.046800e+04
unique	NaN	NaN
top	NaN	NaN
freq	NaN	NaN
mean	14.130431	3.308489e+08
std	31.637275	3.388418e+08
min	0.038329	0.000000e+00
25%	0.235462	1.237768e+08
50%	0.392373	2.181592e+08
75%	12.835269	4.105794e+08
max	181.511703	7.421641e+09

Figure:1 description of data set

The missing values of the data set is given below:

```
Finding NaN values
 Date           0
Open           0
High           0
Low            0
Close          0
Adj Close      0
Volume         0
dtype: int64
```

Figure:2 null values1

As per the above attached snippet , I have zero missing values in each and every column of my dataset.

We have generated the Datetime variable from the date & time1980-12-12 till 2022-06-17 and added it to the data frame since the given date time column is in string format and was converted to datatime format.

b. The plot of dependent variable versus the time:

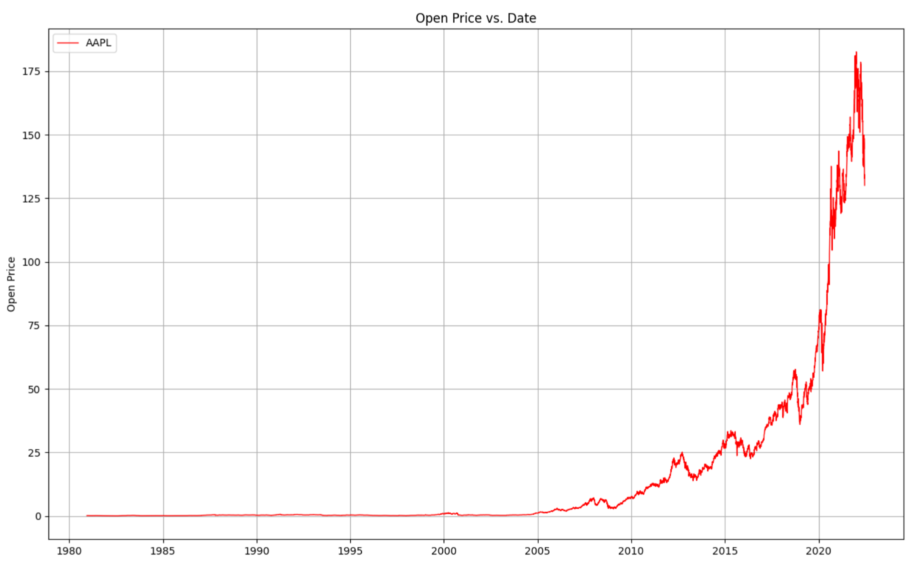


Figure:3 Plot of Open price vs Time

We can observe from the above plot the apple stock price is almost similar from the year 1980 to 2000 after 2000 the stock price of apple is increasing drastically till now 2022, but there are some price drops in the middle of some years but again the price are increased .

C.Sampling:

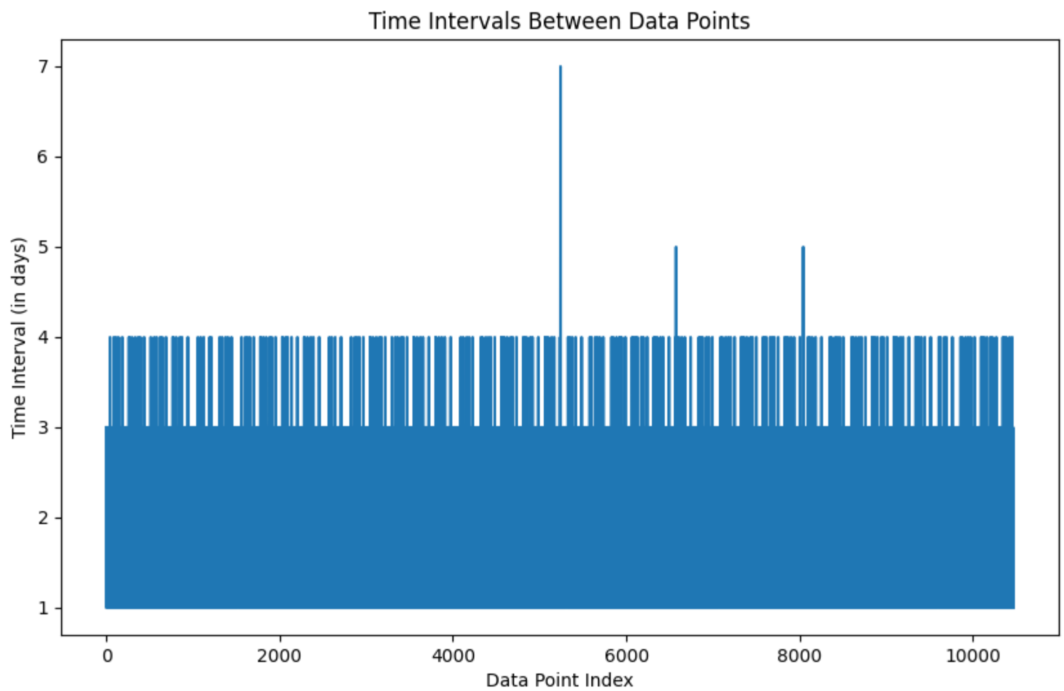


Figure:4 data sampling plot

```
data['Date'] = pd.to_datetime(data['Date'])
time_intervals = data['Date'].diff().dt.days
is_equally_sampled = time_intervals.nunique() == 1
fig, ax = plt.subplots(figsize=(10, 6))

if is_equally_sampled:
    # If data is equally sampled, plot as a bar chart
    ax.bar(data.index, time_intervals, width=0.5)
    ax.set_ylabel("Time Interval (in days)")
    ax.set_title("Time Intervals Between Equally Sampled Data Points")
else:
    # If data is not equally sampled, plot as a line chart
    ax.plot(time_intervals)
    ax.set_ylabel("Time Interval (in days)")
    ax.set_title("Time Intervals Between Data Points")
    ax.set_xlabel("Data Point Index")

plt.show()
```

We can see from above fig that data is equally sampled.

D.ACF/PACF of the dependent variable:

The plot of ACF of the dependent variable (open price) is shown below:

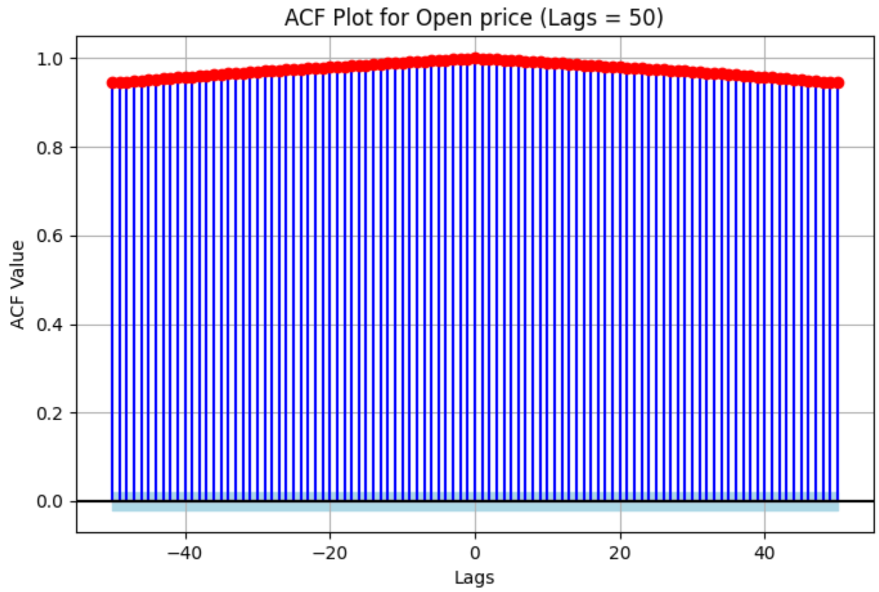


Figure:5 ACF plot of ‘OPEN PRICE

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.

The plot of PACF is shown below:

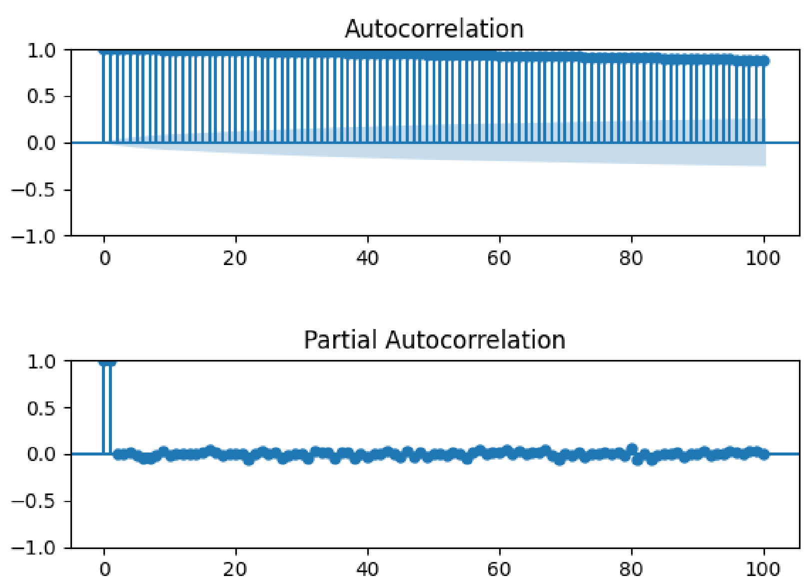


Figure:6 Autocorrelation and partial correlation

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.

c. Heatmap of pearson's correlation coefficient:

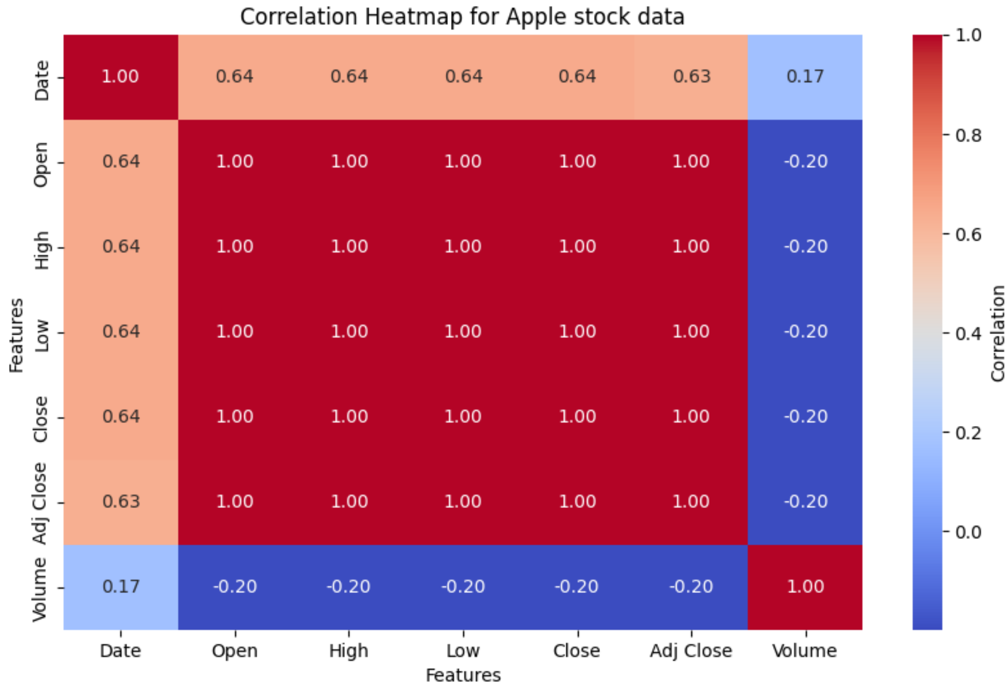


Figure: 7 Heatmap

correlation between different features of Apple stock data, namely Open, High, Low, Close, Adj Close, Volume, and Date. The correlation is calculated using the Pearson correlation coefficient, which ranges from -1 to 1. A correlation of 1 indicates a perfect positive correlation, meaning that the two features move in perfect sync. A correlation of -1 indicates a perfect negative correlation, meaning that the two features move in opposite directions. A correlation of 0 indicates no correlation.

As you can see from the heatmap, all of the features are highly correlated with each other, with the exception of Volume. This is not surprising, as the other features are all measures of the price of Apple stock, while Volume is a measure of the number of shares traded.

The highest correlation is between Open and High (0.64), followed by Open and Close (0.63), High and Close (0.63), and Low and Close (0.63). This means that the opening price, closing price, high price, and low price of Apple stock all tend to move in the same direction.

The lowest correlation is between Volume and all of the other features (-0.2 to -0.4). This means that there is a weak negative correlation between Volume and the price of Apple stock. In other words, when Volume is high, the price of Apple stock tends to be lower, and vice versa.

Split the dataset into train and test values:

By using the following code line we can split the dataset into train and test datasets.

Notice that shuffle=False since this is a time series dataset.

```
X_train shape: (8372, 5), y_train shape: (8372,)
X_test shape: (2094, 5), y_test shape: (2094,)
date_train_test shape: (8372,), y_test shape: (2094,)
```

Figure:8 splitting data code

Stationarity check:

ADF test:

The results of the ADF and KPSS tests on the raw data are shown below:

```
ADF Test for Open:
ADF Statistic: 1.5529526297497769
p-value: 0.9977093353551088
Critical Values:
   1%: -3.4309770043919983
   5%: -2.8618170999074475
   10%: -2.5669174951368516
Result: Open is likely non-stationary (p-value > 0.05)

KPSS Test for Open:
KPSS Statistic: 7.667608118279883
p-value: 0.01
Lags Used: 60
Critical Values:
   10%: 0.347
   5%: 0.463
   2.5%: 0.574
   1%: 0.739
Result: Open is likely non-stationary (p-value < 0.05)
```

Figure:9 ADF and KPSS statistic

ADF test:

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

KPSS test:

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

Rolling Mean and Rolling Variance:

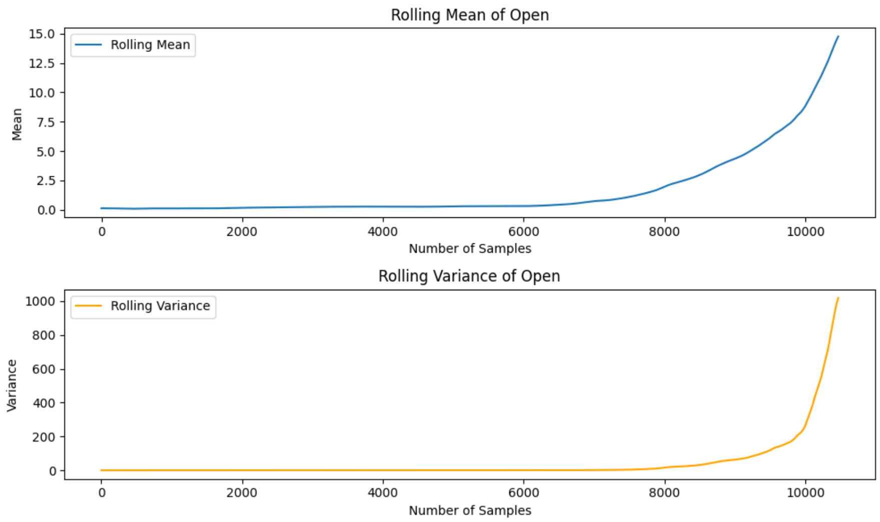


Figure:10 rolling mean vs time

From the rolling mean graph we can see that the trend is decreasing strongly and the values are not constant. So we can say that the variable is non-stationary.

From the rolling variance graph we can see that the value is increasing and the values are not constant. So we can say that the variable is non-stationary.

In order to make our target variable stationary we can use differencing of order 1. This is a commonly used method to make the variable stationary.

First order differencing:

ADF test on 1st order data:

The results of the ADF tests and KPSS test of the target column after doing first order differencing is shown below:

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

KPSS test on 1st order data:

From the analysis of KPSS test we can say that the data set is still non stationary after doing first order differencing, we can see in the below fig like still the p value of KPSS test is less than 0.05

```
[10468 rows x 8 columns]
ADF Test for Open_1st_Difference:
ADF Statistic: -14.53883757101667
p-value: 5.185602233472823e-27
Critical Values:
   1%: -3.4309773051299146
   5%: -2.861817232802289
   10%: -2.566917565876739
Result: Open_1st_Difference is likely stationary (p-value <= 0.05)

KPSS Test for Open_1st_Difference:
KPSS Statistic: 0.7380826301904274
p-value: 0.01008339725541569
Lags Used: 4
Critical Values:
   10%: 0.347
   5%: 0.463
   2.5%: 0.574
   1%: 0.739
Result: Open_1st_Difference is likely non-stationary (p-value < 0.05)
```

Figure:11 ADF test and KPSS test on 1st order data

Rolling mean and variance graphs of 1st order data:

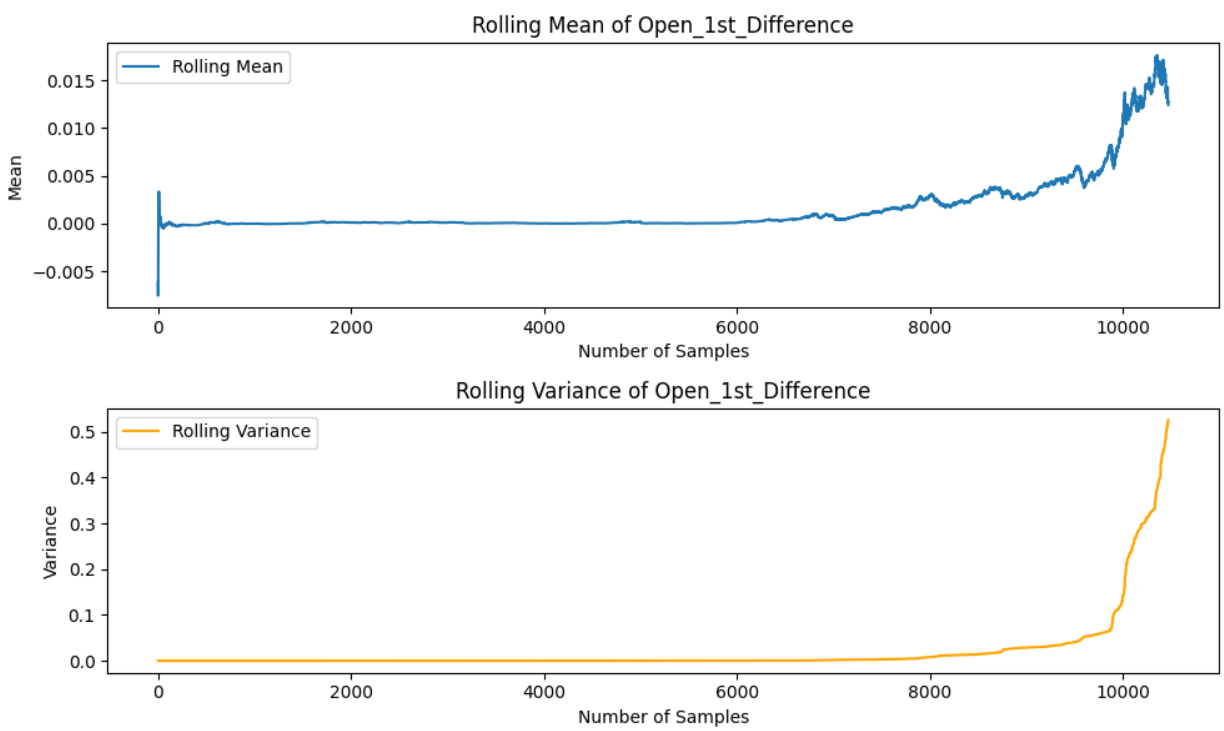


Figure:12 rolling mean and rolling variance-1st order differencing

From rolling mean plot from the above fig , we see that after first order differencing we can see that some the data became constant , but still we have some trend present in the dataset .

From the rolling variance graph we can see that the value is increasing and the values are not constant. So we can say that the variable is non-stationary.

Second order differencing:

ADF test on 2nd order data:

The results of the ADF tests and KPSS test of the target column after doing second order differencing is shown below:

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

KPSS test on 2nd order data:

From the analysis of KPSS test we can say that the data set is stationary after doing second order differencing, we can see in the below fig like still the p value of KPSS test is grater than 0.05

```
ADF Test for Open_2nd_Difference:
ADF Statistic: -28.386662662058296
p-value: 0.0
Critical Values:
   1%: -3.4309773051299146
   5%: -2.861817232802289
   10%: -2.566917565876739
Result: Open_2nd_Difference is likely stationary (p-value <= 0.05)

KPSS Test for Open_2nd_Difference:
KPSS Statistic: 0.03375836795724416
p-value: 0.1
Lags Used: 205
Critical Values:
   10%: 0.347
   5%: 0.463
   2.5%: 0.574
   1%: 0.739
Result: Open_2nd_Difference is likely stationary (p-value >= 0.05)
```

Figure:13 ADF and KPSS test on second order differencing

Rolling Mean and Rolling variance of 2nd order differencing :

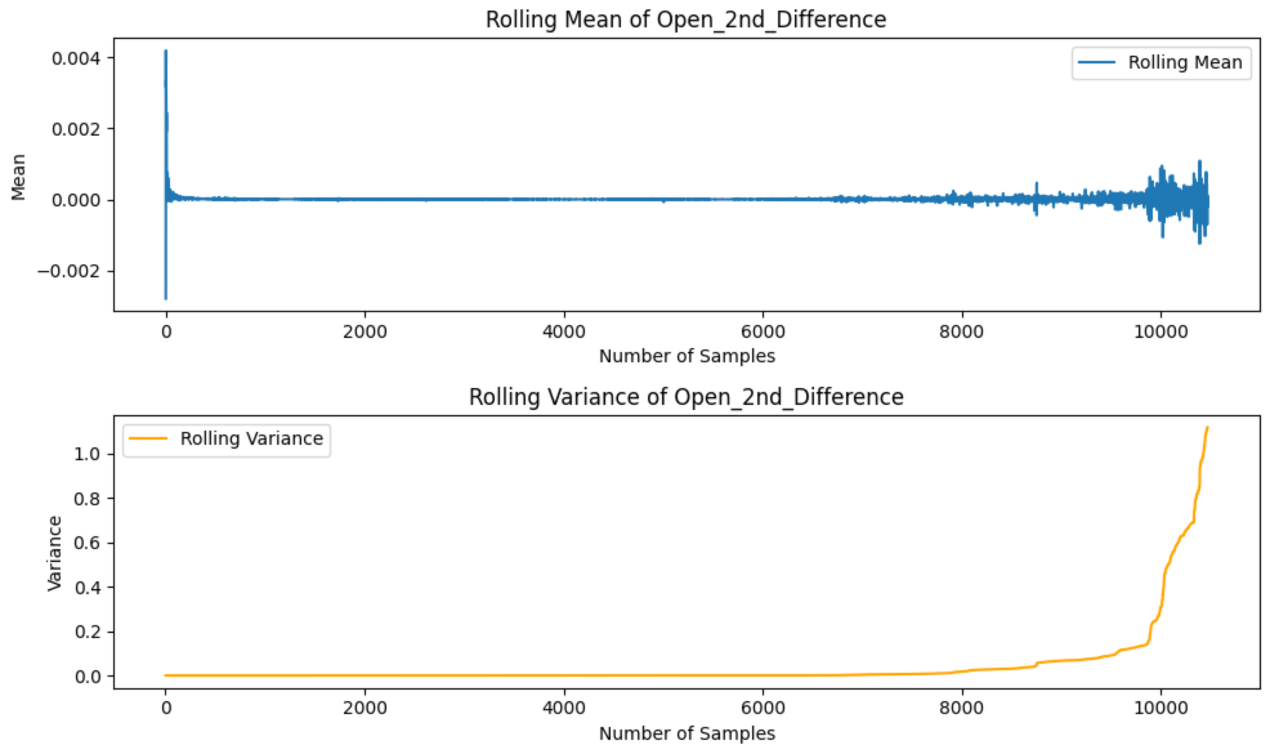


Figure:14Rolling mean variance plot of 2nd order differencing

From above fig , we can say that the values in the rolling mean are now constant with no trend on the dataset now , we can say the data is stationary.
From the rolling variance graph we can see that the values are constant and there is no major trend. So we can say that the variable is stationary.

After second order differencing plot for target variable:

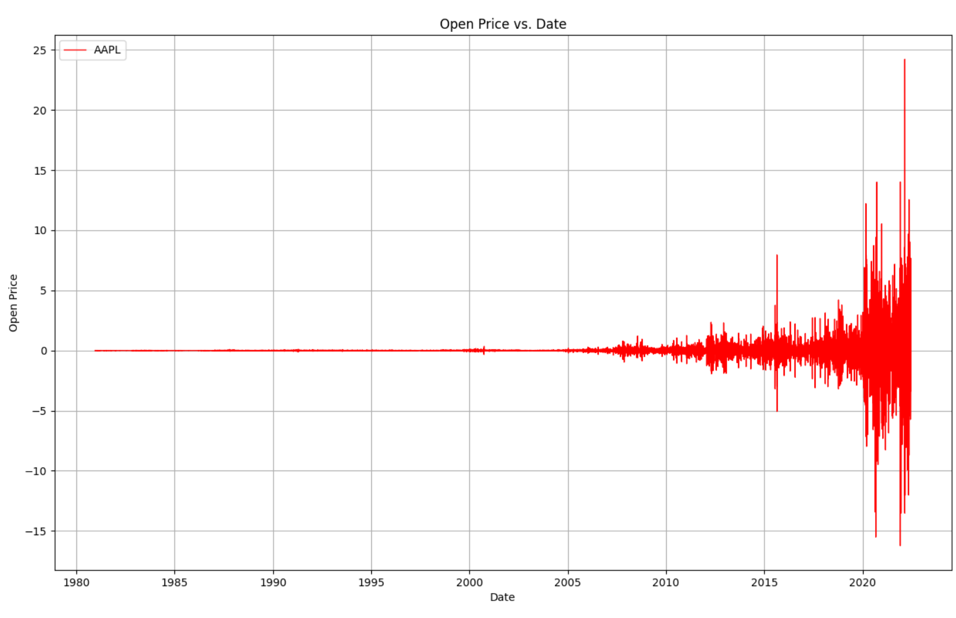


Figure:15 plot for dependent variable after making stationary.

Time Series Decomposition:

We performed time series decomposition on the raw data and we get these results:

```
 Strength of trend for the raw data is 99.976%
Strength of seasonality for the raw data is 30.018%
```

Figure:16 Time series Decomposition

We can observe 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 ~0.30.

Holt winter method:

We performed holt winter modelling on the raw data and we get these results:

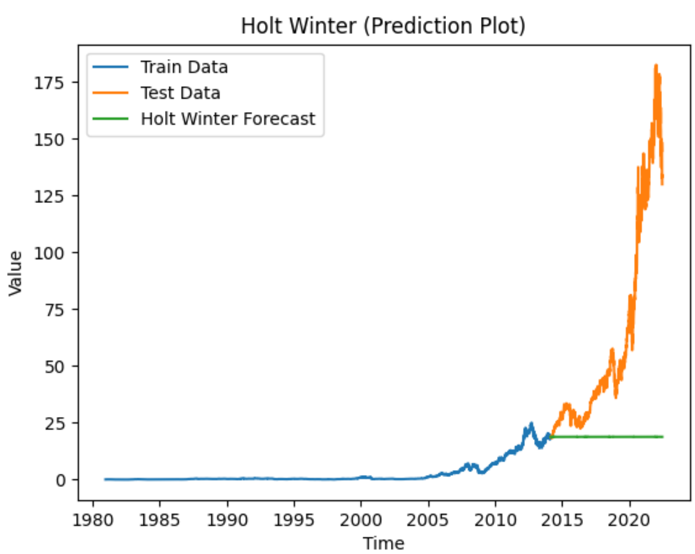


Figure:17 AQI vs time prediction plot

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

Feature selection:

```
Shape of H is (6, 6)
Singular Values [2.34759347e+21 6.26155853e+07 6.95098927e+03 1.46758627e+03
 8.90531261e+02 1.97131936e+02]
The condition number is 1891051078.5970602
unknown coefficients : [ 1.01520676e-01  1.71186157e-01 -2.62194491e-01 -1.13615866e-02
  1.23078466e-12]
```

condition number of 1891051078.5970602 implies that the matrix used in a computation is ill-conditioned. In numerical analysis, a high condition number indicates that the matrix is nearly singular or very sensitive to changes, leading to potential numerical instability or loss of precision in computations.

mentioned (1891051078.5970602) often indicates a high degree of collinearity among variables in a dataset, specifically when dealing with linear regression or matrix operations

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

```
===================================OLS MODEL==================================
                                 OLS Regression Results
=======================================================================================
Dep. Variable:     Open_2nd_Difference   R-squared (uncentered):                   0.006
Model:                             OLS   Adj. R-squared (uncentered):              0.005
Method:                  Least Squares   F-statistic:                              9.958
Date:                 Mon, 11 Dec 2023   Prob (F-statistic):                    1.63e-09
Time:                         12:37:46   Log-Likelihood:                          2991.1
No. Observations:                 8372   AIC:                                     -5972.
Df Residuals:                     8367   BIC:                                     -5937.
Df Model:                            5
Covariance Type:             nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
High           0.1015      0.036      2.838      0.005       0.031       0.172
Low            0.1712      0.035      4.885      0.000       0.102       0.240
Close         -0.2622      0.047     -5.634      0.000      -0.353      -0.171
Adj Close     -0.0114      0.031     -0.368      0.713      -0.072       0.049
Volume      1.231e-12   4.57e-12      0.269      0.788   -7.73e-12    1.02e-11
==============================================================================
Omnibus:                     2655.807   Durbin-Watson:                   3.035
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           528350.953
Skew:                           0.210   Prob(JB):                         0.00
Kurtosis:                      41.916   Cond. No.                     1.54e+10
==============================================================================
```

Figure:18 Feature selection 1 using OLS

Here we can observe that volume has a p-value of 0.788, which is high so we can remove this variable and run the regression again.

```
=========================== Volume dropped ==================================
                                 OLS Regression Results
=======================================================================================
Dep. Variable:     Open_2nd_Difference   R-squared (uncentered):                   0.006
Model:                             OLS   Adj. R-squared (uncentered):              0.005
Method:                  Least Squares   F-statistic:                              12.43
Date:                 Mon, 11 Dec 2023   Prob (F-statistic):                    4.42e-10
Time:                         12:37:46   Log-Likelihood:                          2991.1
No. Observations:                 8372   AIC:                                     -5974.
Df Residuals:                     8368   BIC:                                     -5946.
Df Model:                            4
Covariance Type:             nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
High           0.1048      0.034      3.120      0.002       0.039       0.171
Low            0.1680      0.033      5.095      0.000       0.103       0.233
Close         -0.2609      0.046     -5.636      0.000      -0.352      -0.170
Adj Close     -0.0130      0.030     -0.429      0.668      -0.072       0.046
==============================================================================
Omnibus:                     2650.859   Durbin-Watson:                   3.035
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           528067.849
Skew:                           0.201   Prob(JB):                         0.00
Kurtosis:                      41.906   Cond. No.                         330.
==============================================================================
```

Figure:19 Feature selection 2

Here we can observe that Adj Close has a p-value of 0.668, which is high so we can remove this variable and run the regression again.

```
========================== Adj Close dropped =================================
                            OLS Regression Results
=======================================================================================
Dep. Variable:     Open_2nd_Difference   R-squared (uncentered):                   0.006
Model:                             OLS   Adj. R-squared (uncentered):              0.006
Method:                  Least Squares   F-statistic:                              16.51
Date:                 Mon, 11 Dec 2023   Prob (F-statistic):                    1.07e-10
Time:                         12:51:59   Log-Likelihood:                          2991.0
No. Observations:                 8372   AIC:                                     -5976.
Df Residuals:                     8369   BIC:                                     -5955.
Df Model:                            3
Covariance Type:             nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
High           0.1060      0.033      3.166      0.002       0.040       0.172
Low            0.1660      0.033      5.085      0.000       0.102       0.230
Close         -0.2714      0.039     -6.896      0.000      -0.349      -0.194
==============================================================================
Omnibus:                     2650.657   Durbin-Watson:                   3.035
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           527941.792
Skew:                           0.201   Prob(JB):                         0.00
Kurtosis:                      41.901   Cond. No.                         262.
==============================================================================
```

Figure:20 final result after removing unwanted features

Since none of the variables has a high p-value we don't need to drop any variable any more. We can perform predictions with this.

The predictions of the OLS model is:

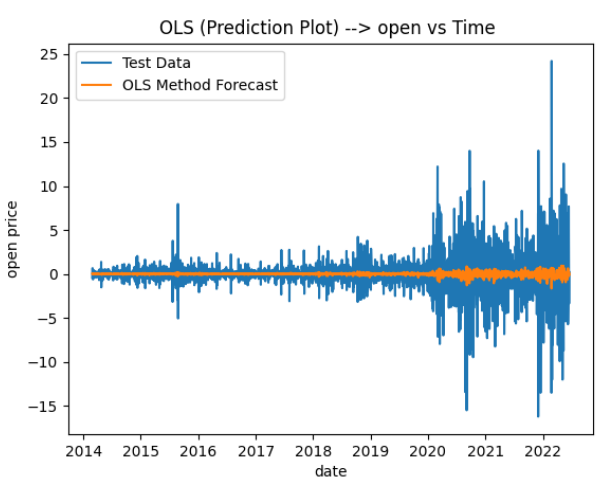


Figure:21 test data and prediction plot of Ols

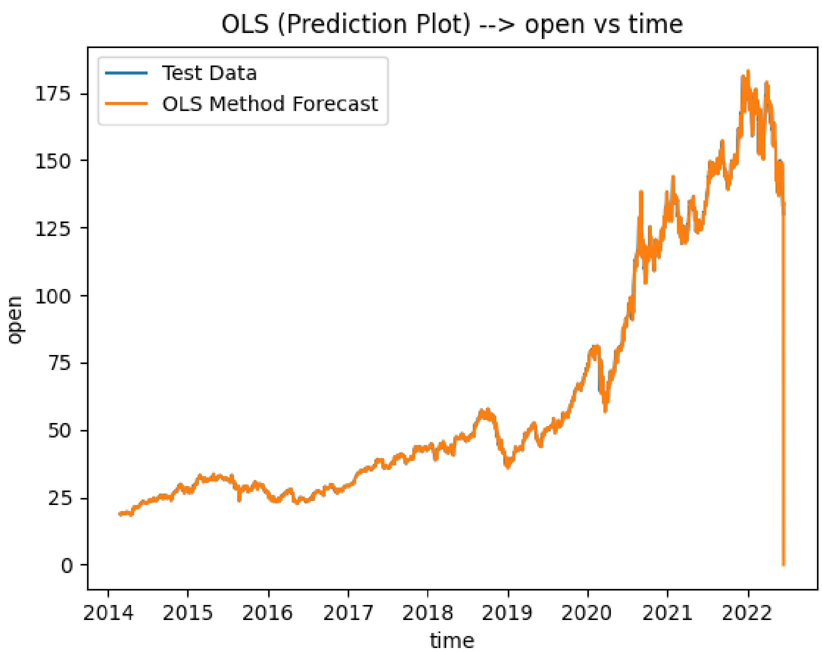


Figure:22 open price vs time

```
Mean of residual error (OLS) method : 0.005051862384056998
variance of residual error (OLS) method : 5.589153199638759
```

Figure:23 OLS method

We can observe that the OLS model is very good at predicting the open price model. The mean of residual error is low.

Base models:

Average model:

The resultant graph of the average model is shown below:

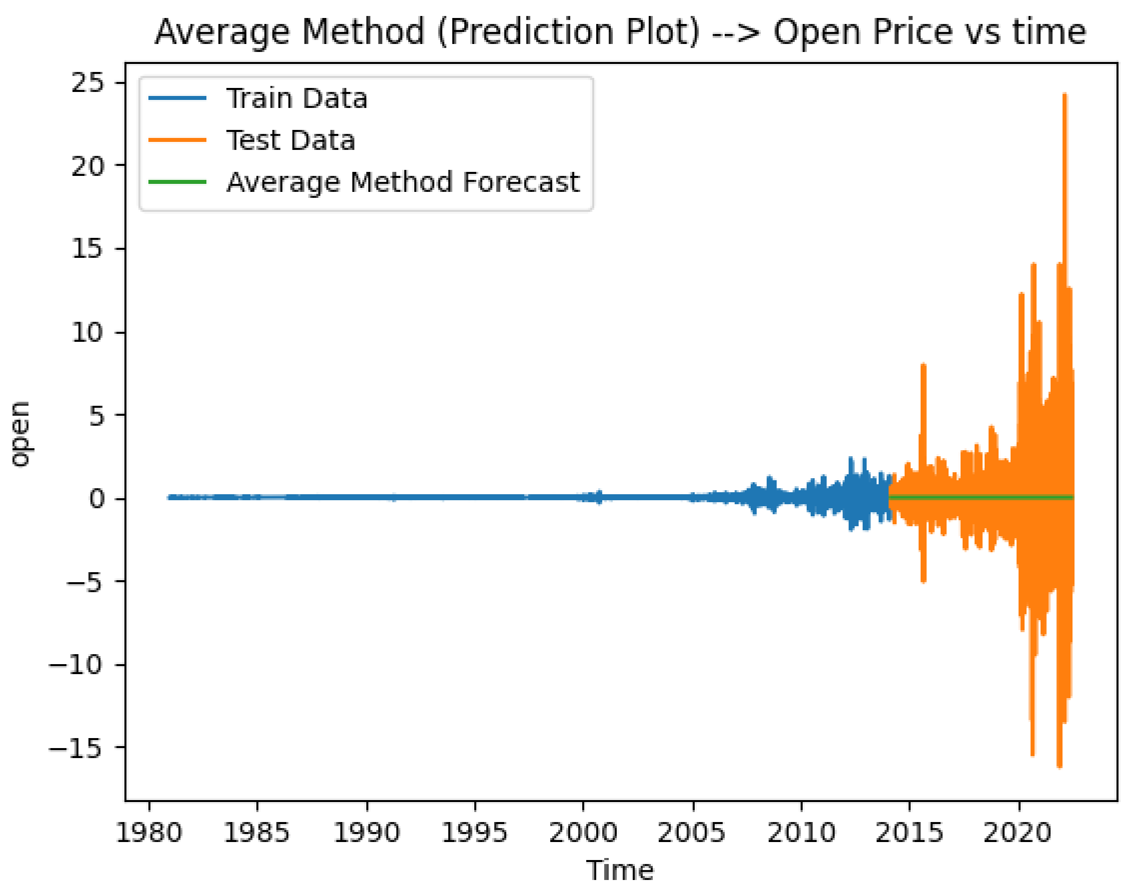


Figure:24 open price vs time(Average)

The graph demonstrates that the average method's forecast, covering the length of the test dataset, mirrors the average value derived from the train dataset.

Naive model:

The resultant graph of the naive model is shown below:

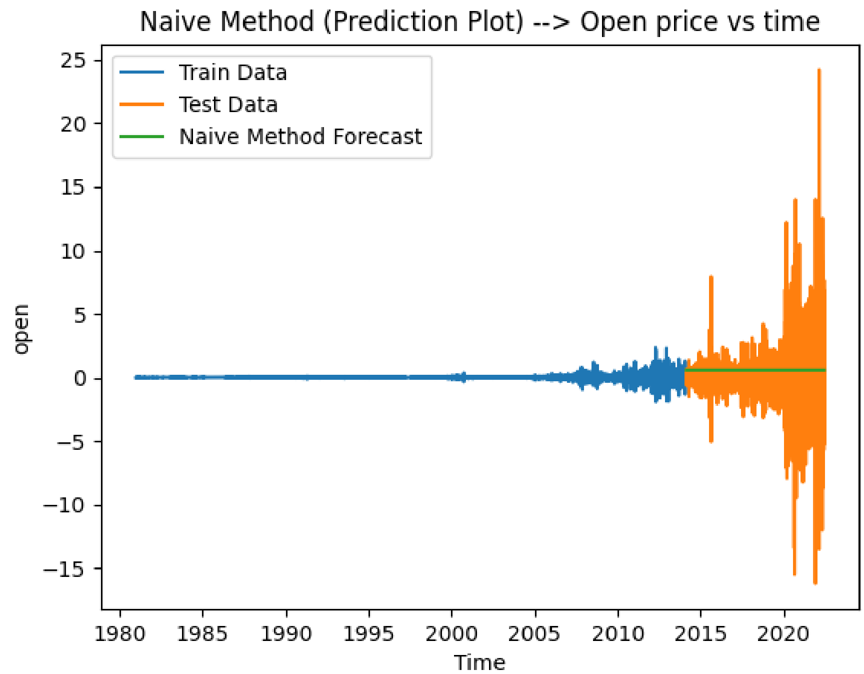


Figure:25 open price vs time(Naive)

The graph illustrates that the naive method's forecast for the duration of the test dataset corresponds to the final value observed in the train dataset.

Drift method:

The resultant graph of the drift model is shown below:

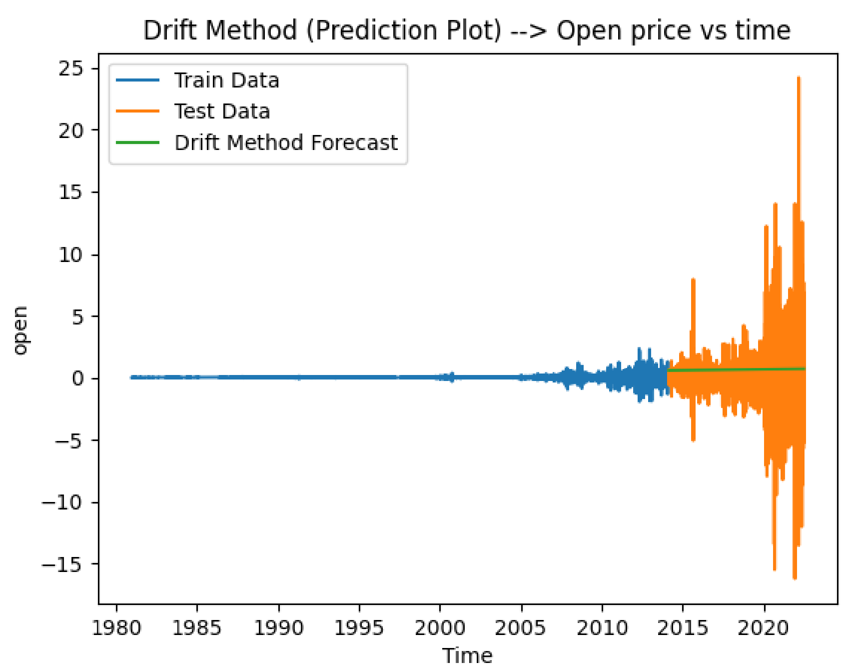


Figure:26 open price vs time(Drift)

The graph demonstrates that the drift method's forecast over the test dataset length extrapolates from the line connecting the initial and final values observed in the train dataset.

Simple Exponential Smoothing (SES) method:

The resultant graph of the SES model is shown below:

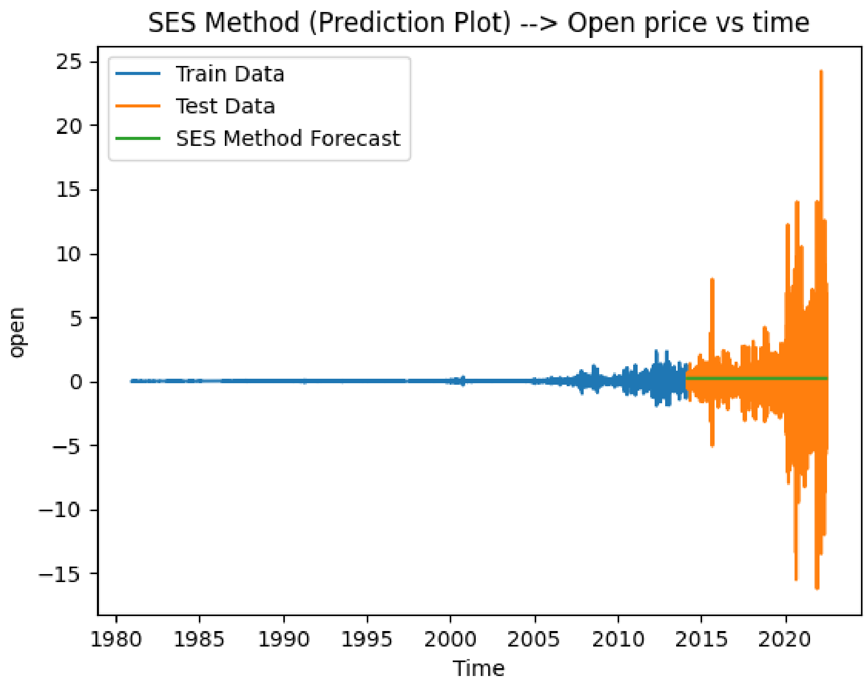


Figure: 27 open price vs time(ses)

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.

ARMA model:

We perform the GPAC table of the ACF values of the 1st order differenced data. The following is the GPAC table that we obtain from it.

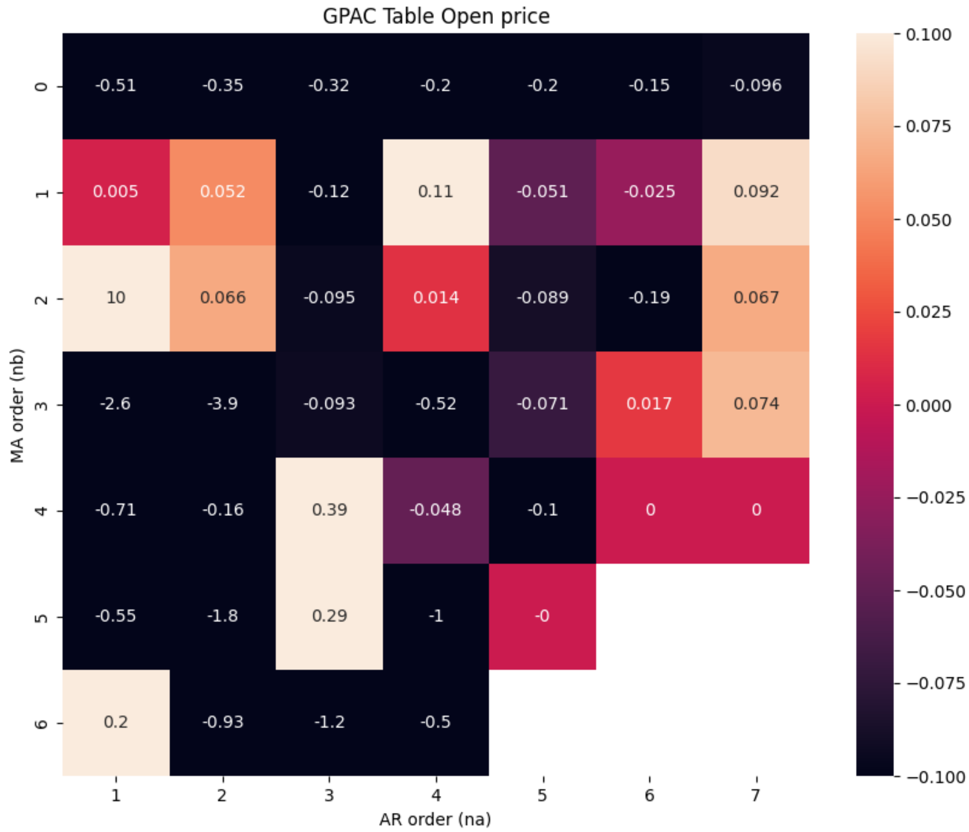
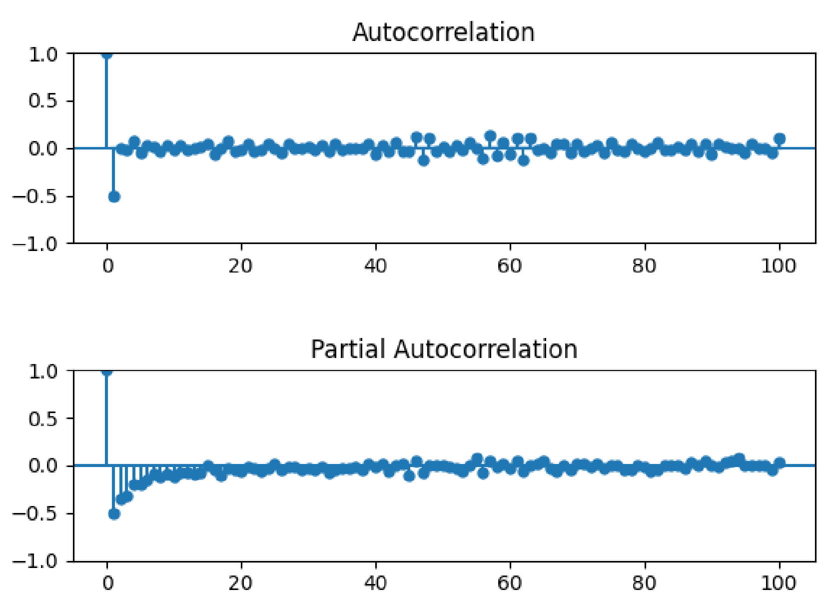


Figure:28 GPAC table

From the above GPAC table we can observe that there is no particular order , so I tried auto arima function to get the best model for my data set



AUTO ARIMA:

```
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(0,0,0)[0] intercept   : AIC=inf, Time=9.66 sec
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=30865.380, Time=0.16 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=27744.921, Time=0.34 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=inf, Time=4.89 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=30863.381, Time=0.06 sec
 ARIMA(2,0,0)(0,0,0)[0] intercept   : AIC=26367.846, Time=0.53 sec
 ARIMA(3,0,0)(0,0,0)[0] intercept   : AIC=25248.818, Time=0.69 sec
 ARIMA(4,0,0)(0,0,0)[0] intercept   : AIC=24837.791, Time=0.84 sec
 ARIMA(5,0,0)(0,0,0)[0] intercept   : AIC=24434.219, Time=1.02 sec
 ARIMA(5,0,1)(0,0,0)[0] intercept   : AIC=inf, Time=14.60 sec
 ARIMA(4,0,1)(0,0,0)[0] intercept   : AIC=inf, Time=14.34 sec
 ARIMA(5,0,0)(0,0,0)[0]             : AIC=24432.227, Time=0.50 sec
 ARIMA(4,0,0)(0,0,0)[0]             : AIC=24835.794, Time=0.38 sec
 ARIMA(5,0,1)(0,0,0)[0]             : AIC=inf, Time=3.46 sec
 ARIMA(4,0,1)(0,0,0)[0]             : AIC=inf, Time=2.33 sec

Best model:  ARIMA(5,0,0)(0,0,0)[0]
```

Figure:29 Auto Arima

From the Autoarima model we got the best model as na=5 and nb=0

In this project we performed the 2 ARMA models, each for ARMA(5,0)

ARMA (5,0) model:

```
                               SARIMAX Results
================================================================================
Dep. Variable:     Open_2nd_Difference   No. Observations:                 8372
Model:                  ARIMA(5, 0, 0)   Log Likelihood                5676.105
Date:                 Mon, 11 Dec 2023   AIC                         -11338.210
Time:                         13:33:30   BIC                         -11288.981
Sample:                              0   HQIC                        -11321.398
                                - 8372
Covariance Type:                   opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const      -1.571e-05      0.000     -0.043      0.966      -0.001       0.001
ar.L1         -0.9260      0.004   -251.708      0.000      -0.933      -0.919
ar.L2         -0.7225      0.005   -150.245      0.000      -0.732      -0.713
ar.L3         -0.5932      0.005   -121.443      0.000      -0.603      -0.584
ar.L4         -0.3698      0.005    -76.510      0.000      -0.379      -0.360
ar.L5         -0.1671      0.004    -47.406      0.000      -0.174      -0.160
sigma2         0.0151   5.03e-05    299.847      0.000       0.015       0.015
===================================================================================
Ljung-Box (L1) (Q):                   1.04   Jarque-Bera (JB):            803545.23
Prob(Q):                              0.31   Prob(JB):                         0.00
Heteroskedasticity (H):             562.84   Skew:                             0.70
Prob(H) (two-sided):                  0.00   Kurtosis:                        50.97
===================================================================================
```

Figure:30 summary of Arma(5,0)

Diagnostic test:

The ACF plot of residual error is shown below:

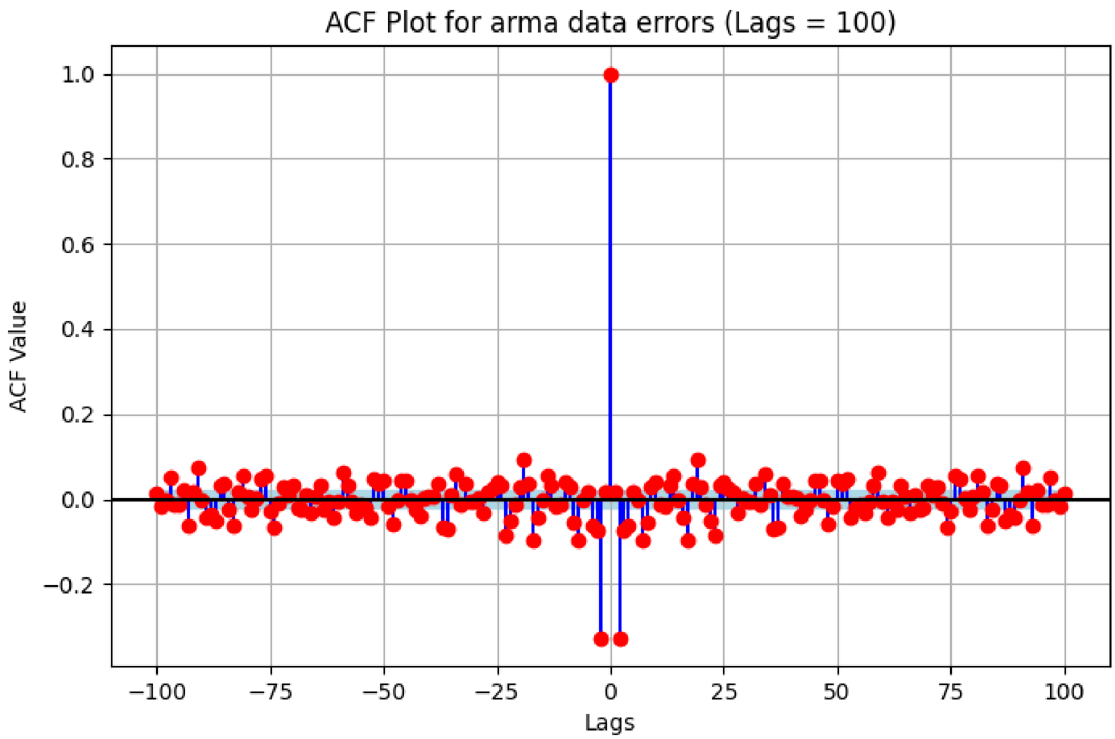


Figure:31 ACF plot of residual error

```
Chi Squared test results
The residuals is white,  and the chi_square values is :1.4993612958876334
```

From the above ACF plot we can observe that residual errors is white. We can confirm this by seeing the chi squared test.

```
parameters for confidence intervals :                 0         1
const   -0.000728  0.000696
ar.L1   -0.933187 -0.918766
ar.L2   -0.731915 -0.713065
ar.L3   -0.602735 -0.583589
ar.L4   -0.379228 -0.360284
ar.L5   -0.174028 -0.160209
sigma2  0.014986  0.015183
zero/cancellation:
zeros : []
poles : [1.571347531728229e-05, 0.9259766982985105, 0.722490402079021, 0.5931617118365285, 0.36975570056363294]
```

```
variance of residual error : 0.03652071736108011
variance of forecast error : 5.4712054126825045
```

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

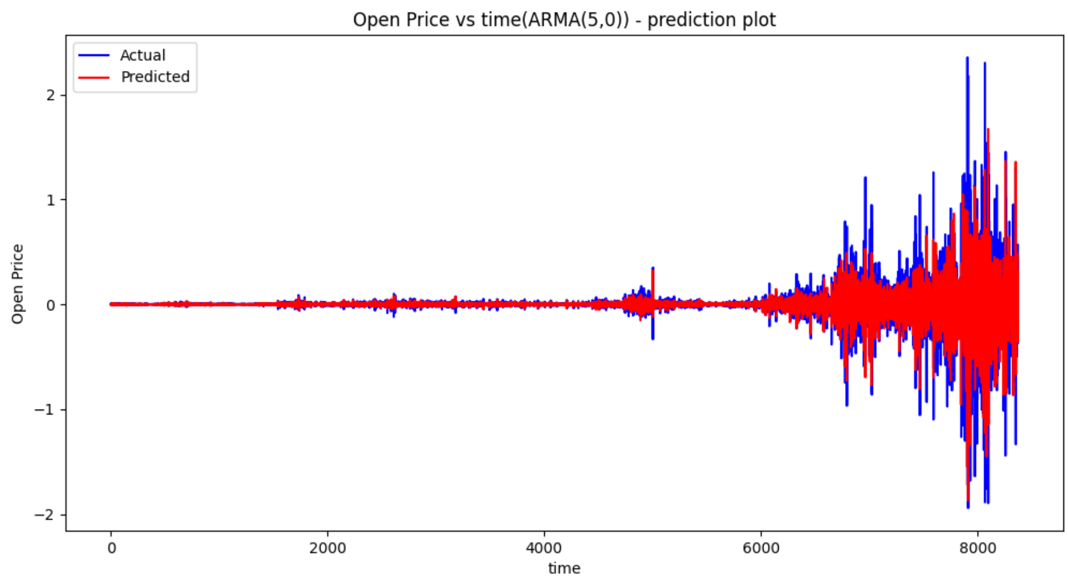


Figure:32 Actual data and prediction data plot

The 1-step prediction of 1st 2500 samples plot is shown below:

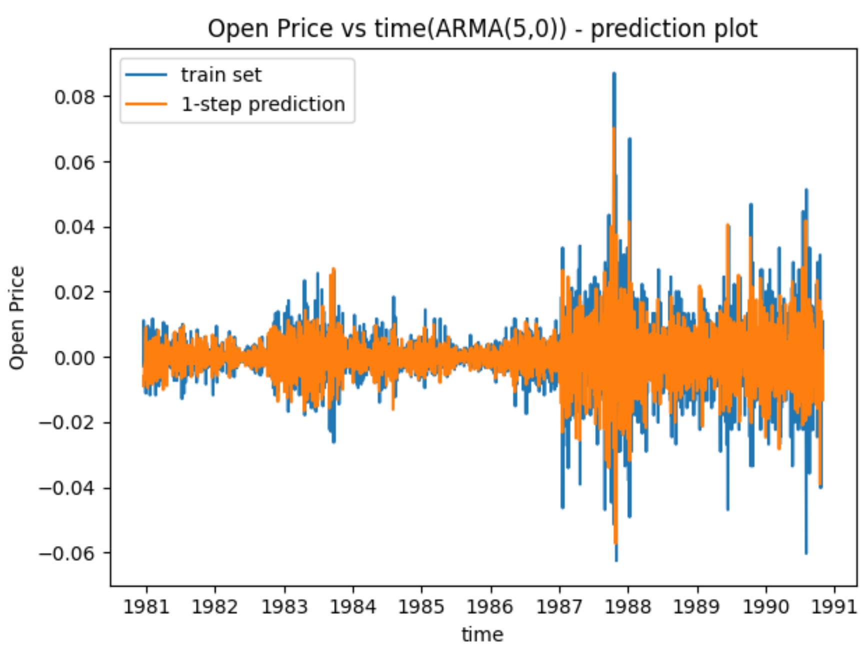


Figure33 plot of train and prediction set up to 2500 samples

The h-step forecast plot is shown below:

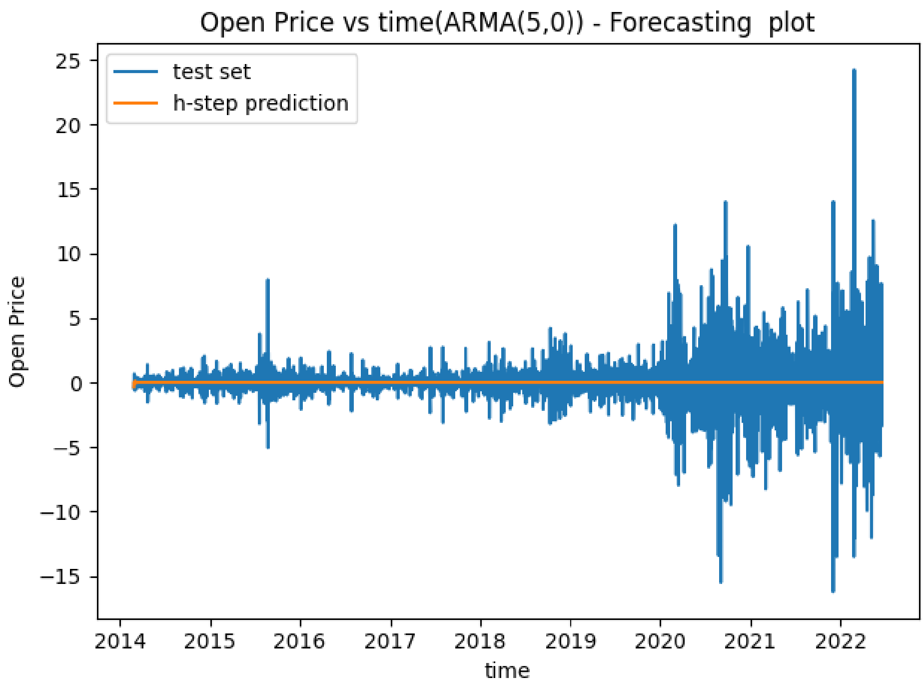


Figure:34 openprice vs time(ARMA(5,0))

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 open values.

ARIMA (5,2,0) model:

ARIMA Model Results

```
                               SARIMAX Results
==============================================================================
Dep. Variable:     Open_2nd_Difference   No. Observations:                 8372
Model:                  ARIMA(5, 2, 0)   Log Likelihood                1758.995
Date:                 Mon, 11 Dec 2023   AIC                          -3505.989
Time:                         14:06:58   BIC                          -3463.795
Sample:                              0   HQIC                         -3491.580
                              - 8372
Covariance Type:                   opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -1.9353      0.003   -749.293      0.000      -1.940      -1.930
ar.L2         -2.2190      0.005   -426.622      0.000      -2.229      -2.209
ar.L3         -1.9020      0.006   -303.026      0.000      -1.914      -1.890
ar.L4         -1.1858      0.005   -224.894      0.000      -1.196      -1.176
ar.L5         -0.4339      0.003   -153.028      0.000      -0.439      -0.428
sigma2         0.0384      0.000    265.829      0.000       0.038       0.039
===================================================================================
Ljung-Box (L1) (Q):                 220.88   Jarque-Bera (JB):            402674.19
Prob(Q):                              0.00   Prob(JB):                         0.00
Heteroskedasticity (H):             573.13   Skew:                             0.62
Prob(H) (two-sided):                  0.00   Kurtosis:                        36.96
===================================================================================
```

Figure:35 summary of ARIMA model(5,2,0)

The AIC and BIC values are large so we can say that this is a good model.

Diagnostic test:

The ACF plot of residual error is shown below:

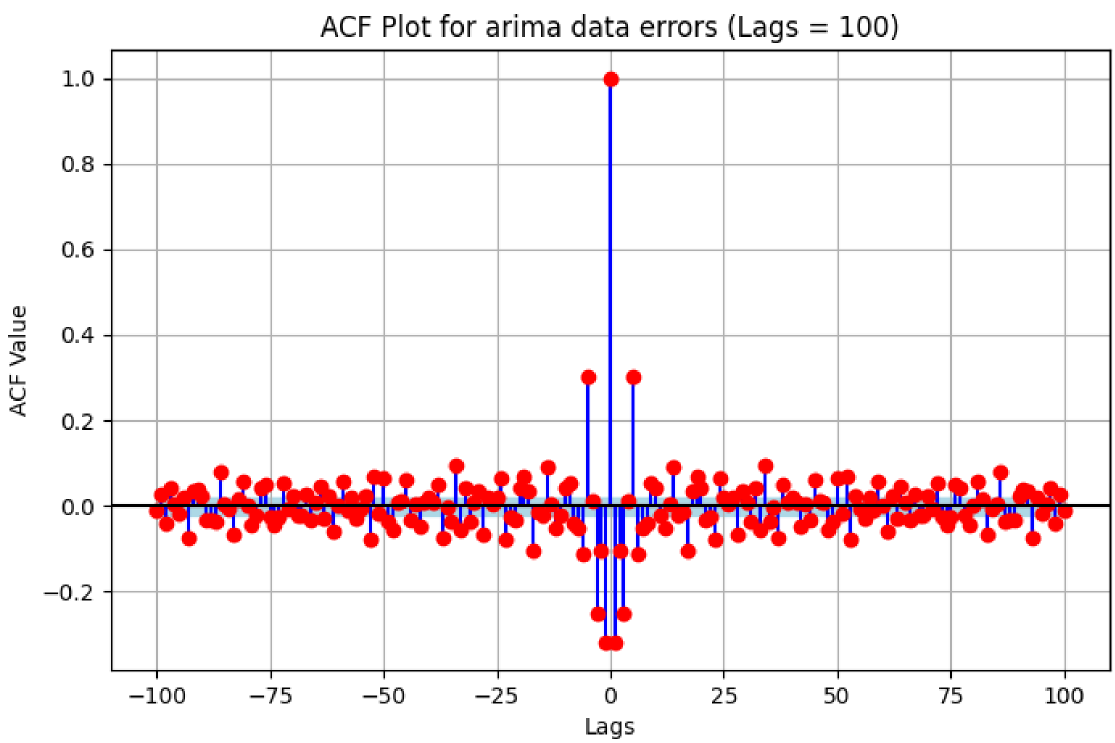


Figure: 36 ACF plot of residual error

```
Chi Squared test results
The residuals is white,  and the chi_square values is :1.121952246736736
```

```
parameters for confidence intervals :                  0         1
ar.L1   -1.940325 -1.930200
ar.L2   -2.229238 -2.208849
ar.L3   -1.914271 -1.889667
ar.L4   -1.196170 -1.175501
ar.L5   -0.439481 -0.428365
sigma2   0.038152  0.038718
zero/cancellation:
zeros : []
poles : [1.9352623945681398, 2.219043603522307, 1.9019689207570187, 1.1858355879149296, 0.43392304448426955]
```

```
Variance of residual error: 0.022000701236954842
Variance of forecast error: 1570.9938891439213
```

Arima one step prediction plot:

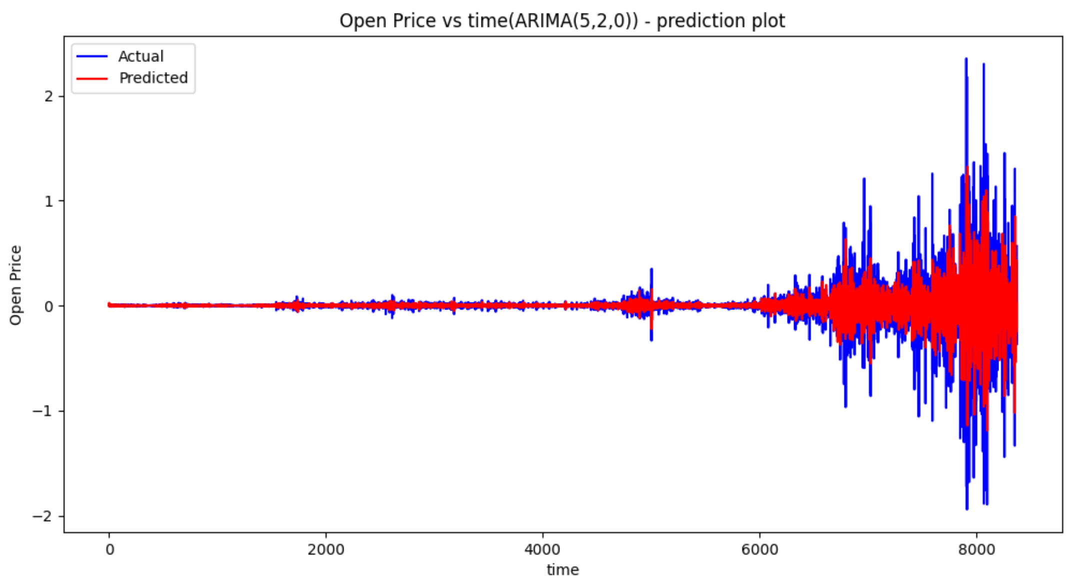


Figure:37 plot of actual and predicted data of arima model

The 1-step prediction of 1st 2500 samples plot is shown below:

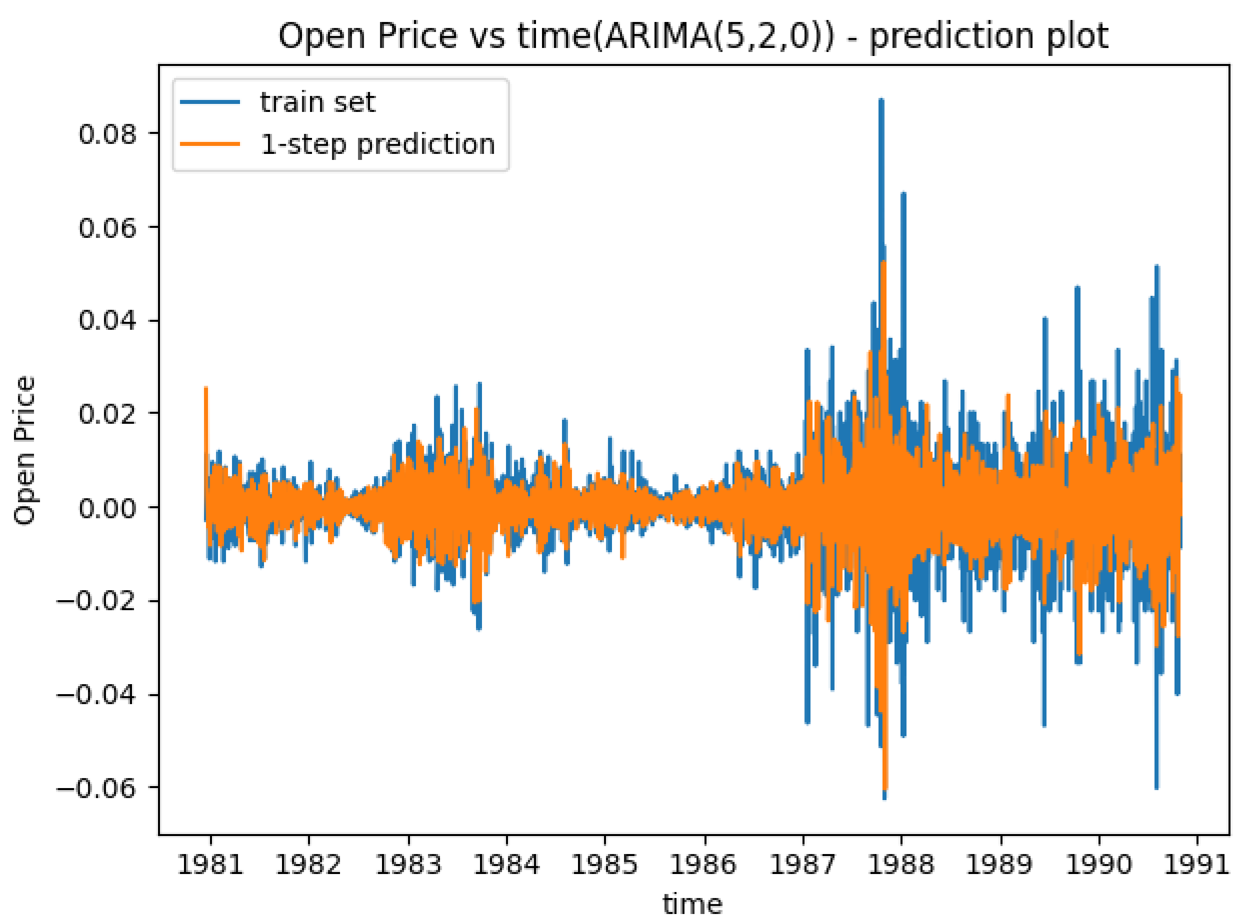


Figure:38 AQI vs time(ARIMA) for first 2500 samples

The h-step forecast plot is shown below:

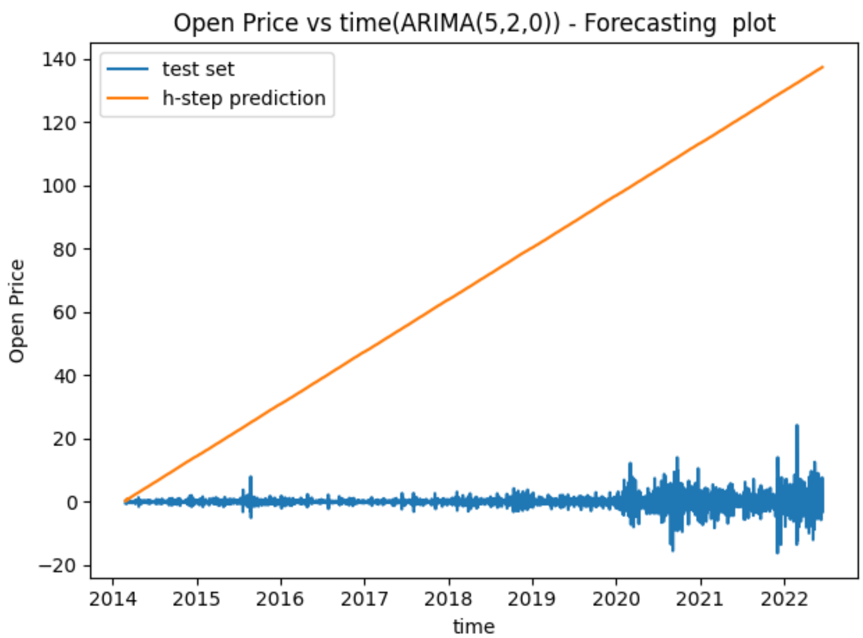


Figure:39 open price vs time(ARIMA) forecast

Based on the information provided, it appears that the ARMA(5,0,0) model performs better in terms of log likelihood, AIC, and BIC. These criteria suggest that the ARMA(5,0,0) model has a better fit to the data compared to the ARIMA(5,2,0) model. Additionally, the variance of the residual error in the ARMA model is lower, which is favorable.

So our best model is ARMA (5,0,0).

Summary and conclusion:

Data Pre-processing:

The initial dataset underwent preprocessing to handle missing values, followed by segmentation for Chennai-specific data analysis.

Stationarity Testing:

Stationarity checks were performed, necessitating a 1st order differentiation step to achieve stationarity within the data.

Base Models Evaluation:

Base models (Average, Naive, Drift, SES, Holt-Winters) were initially employed, but their performance fell short of expectations due to the dataset's complexity.

ARMA and ARIMA Model Selection:

GPAC table and auto ARIMA techniques were employed to determine the appropriate parameters for the ARMA and ARIMA models.

Model Construction and Comparison:

ARMA(5,0) and ARIMA(5,2,0) models were constructed and tested. Upon comparison, ARMA(5,0) exhibited superior performance.

Model Comparison Metrics:

ARMA(5,0) was deemed the better model due to its lower variance in both residual and forecast errors, showcasing its stronger predictive ability.

Conclusion:

The ARMA(5,0) model emerges as the preferred choice, demonstrating its capability to provide more accurate predictions for the dataset, with lower error variance and enhanced forecast reliability.

Appendix:

```python
import numpy as np
import pandas as pd
import seaborn as sns
import copy
import matplotlib.pyplot as plt
import warnings
import statsmodels.api as sm
warnings.filterwarnings("ignore")

pd.set_option("display.max.columns", None)
url = "C:/TimeSeries/final project/AAPL (1).csv"
data = pd.read_csv(url)
print(data)

print(data.columns)

#Data Cleaning
print('\n')
print(data.describe(include='all'))

#Data types
print(f'\n Data types:{data.dtypes}')

#Checking Null values
print('\nFinding NaN values\n', data.isnull().sum())

#Check wether the data is Equally sampled or not
data['Date'] = pd.to_datetime(data['Date'])
time_intervals = data['Date'].diff().dt.days
is_equally_sampled = time_intervals.nunique() == 1
fig, ax = plt.subplots(figsize=(10, 6))

if is_equally_sampled:
    # If data is equally sampled, plot as a bar chart
    ax.bar(data.index, time_intervals, width=0.5)
    ax.set_ylabel("Time Interval (in days)")
    ax.set_title("Time Intervals Between Equally Sampled Data Points")
else:
    # If data is not equally sampled, plot as a line chart
    ax.plot(time_intervals)
    ax.set_ylabel("Time Interval (in days)")
    ax.set_title("Time Intervals Between Data Points")
    ax.set_xlabel("Data Point Index")

plt.show()

#Plot for Dependent Variable VS Time

data['Date'] = pd.to_datetime(data['Date'])

print(f'\n Data types:{data.dtypes}')

# Create a plot of Open Price vs. Date
plt.figure(figsize=(15, 9))
plt.plot(data['Date'], data['Open'], label='AAPL', color='red', linewidth=1.0)
plt.title('Open Price vs. Date')
plt.xlabel('Date')
plt.ylabel('Open Price')
plt.grid(True)
plt.legend()
plt.show()
```

```
#ACF/PACF of the dependent variable

print("\n ACF Plot")
def ACF(data, lags):
    mean = np.mean(data)
    autocorrelations = []

    for lag in range(lags + 1):
        numerator = 0
        denominator = 0

        for i in range(lag, len(data)):
            numerator += (data[i] - mean) * (data[i - lag] - mean)
        for i in range(len(data)):
            denominator += (data[i] - mean) ** 2

        r = numerator / denominator
        autocorrelations.append(r)

    return autocorrelations

lags = 50
acf_values = ACF(data['Open'], lags)
left = acf_values[::-1]
right = acf_values[1:]
combine = left+right
confidence_interval= 1.96/np.sqrt(len(data))
plt.figure(figsize=(8, 5))
x_lags = list(range(-lags,lags+1))
plt.stem(x_lags, combine, markerfmt='ro', linefmt='b-', basefmt='r-')
plt.fill_between(x_lags, -confidence_interval, confidence_interval,
color='lightblue', alpha=1)
plt.xlabel('Lags')
plt.ylabel('ACF Value')
plt.title(f'ACF Plot for Open price (Lags = {lags})')
plt.axhline(0, color='black')
plt.grid()
plt.show()

from statsmodels.graphics.tsaplots import plot_acf , plot_pacf
fig = plt.figure()
plt.subplot(211)
plt.title('ACF of the open price')
sm.graphics.tsa.plot_acf(data['Open'], lags=100, ax=plt.gca())
plt.subplot(212)
plt.title('PACF of the open price')
sm.graphics.tsa.plot_pacf(data['Open'], lags=100, ax=plt.gca())
fig.tight_layout(pad=3)
plt.show()

#Heat map for correlation_matrix
correlation_matrix = data.corr()
plt.figure(figsize=(10, 6))
heatmap = sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm',
fmt=".2f", cbar=True, cbar_kws={'label': 'Correlation'})
plt.title('Correlation Heatmap for Apple stock data')
heatmap.set_xlabel('Features')
heatmap.set_ylabel('Features')
plt.show()

## data Stationarity check
```

```python
# ADF Test
from statsmodels.tsa.stattools import adfuller
def perform_adf_test(data, column_name):
    result = adfuller(data[column_name])
    print(f'ADF Test for {column_name}:')
    print(f'ADF Statistic: {result[0]}')
    print(f'p-value: {result[1]}')
    print(f'Critical Values:')
    for key, value in result[4].items():
        print(f'   {key}: {value}')

    # Determine stationarity based on p-value
    if result[1] <= 0.05:
        print(f'Result: {column_name} is likely stationary (p-value <= 0.05)\n')
    else:
        print(f'Result: {column_name} is likely non-stationary (p-value >
0.05)\n')

# Perform ADF test for  column
perform_adf_test(data, 'Open')

#KPSS Test
from statsmodels.tsa.stattools import kpss


# Function to perform KPSS test and print the results
def perform_kpss_test(data, column_name):
    result = kpss(data[column_name])
    print(f'KPSS Test for {column_name}:')
    print(f'KPSS Statistic: {result[0]}')
    print(f'p-value: {result[1]}')
    print(f'Lags Used: {result[2]}')
    print(f'Critical Values:')
    for key, value in result[3].items():
        print(f'   {key}: {value}')

    # Determine stationarity based on p-value
    if result[1] >= 0.05:
        print(f'Result: {column_name} is likely stationary (p-value >= 0.05)\n')
    else:
        print(f'Result: {column_name} is likely non-stationary (p-value <
0.05)\n')

# Perform KPSS test for each column
perform_kpss_test(data, 'Open')

#Rolling Mean and Rolling Variance
def Cal_rolling_mean_var(data, column_name):
    # Initialize lists to store rolling means and variances
    rolling_means = []
    rolling_variances = []

    # Create a figure with two subplots
    fig, axes = plt.subplots(2, 1, figsize=(10, 6))

    # Loop over the number of samples in the dataset
    for i in range(1, len(data) + 1):
        # Calculate rolling mean and variance up to the current sample
        rolling_mean = data[column_name][:i].mean()
        rolling_variance = data[column_name][:i].var()

        # Append the values to the respective lists
```

```
        rolling_means.append(rolling_mean)
        rolling_variances.append(rolling_variance)

    # Plot rolling means
    axes[0].plot(rolling_means, label='Rolling Mean')
    axes[0].set_title('Rolling Mean of {}'.format(column_name))
    axes[0].set_xlabel('Number of Samples')
    axes[0].set_ylabel('Mean')

    # Plot rolling variances
    axes[1].plot(rolling_variances, label='Rolling Variance', color='orange')
    axes[1].set_title('Rolling Variance of {}'.format(column_name))
    axes[1].set_xlabel('Number of Samples')
    axes[1].set_ylabel('Variance')
    axes[0].legend()
    axes[1].legend()
    plt.tight_layout()
    plt.show()

# Call the function to calculate and plot rolling mean and variance for "Sales"
column
Cal_rolling_mean_var(data, 'Open')

#Differencing
print("=======First Order Differencing======")
differenced_data = copy.deepcopy(data)
for i in range(1, len(differenced_data)):
    differenced_data.at[i, 'Open_1st_Difference'] = differenced_data.at[i,
'Open'] - differenced_data.at[i - 1, 'Open']

print("\n  After First order differencing\n", differenced_data)
perform_adf_test(differenced_data.dropna(), 'Open_1st_Difference')
perform_kpss_test(differenced_data.dropna(),'Open_1st_Difference')
Cal_rolling_mean_var(differenced_data, 'Open_1st_Difference')

print("=======Second Order Differencing======")
for i in range(1, len(differenced_data)):
    differenced_data.at[i, 'Open_2nd_Difference'] = differenced_data.at[i,
'Open_1st_Difference'] - differenced_data.at[i - 1, 'Open_1st_Difference']

print("\n  After Second order differencing\n", differenced_data)
perform_adf_test(differenced_data.dropna(), 'Open_2nd_Difference')
perform_kpss_test(differenced_data.dropna(), 'Open_2nd_Difference')
Cal_rolling_mean_var(differenced_data, 'Open_2nd_Difference')

data['Date'] = pd.to_datetime(data['Date'])

# Create a plot of Open Price vs. Date
plt.figure(figsize=(15, 9))
plt.plot(differenced_data['Date'], differenced_data['Open_2nd_Difference'],
label='AAPL', color='red', linewidth=1.0)
plt.title('Open Price vs. Date')
plt.xlabel('Date')
plt.ylabel('Open Price')
plt.grid(True)
plt.legend()
plt.show()


#STL Decomposition
from statsmodels.tsa.seasonal import STL, seasonal_decompose
Temp = data['Open']
```

```python
Temp = pd.Series(np.array(Temp), index=pd.date_range('1980-12-12',
                                                     periods=len(Temp),
                                                     freq = 'D'),)
STL = STL(Temp)
res = STL.fit()
plt.figure(figsize=(8, 6))

# Plot the original time series
plt.subplot(411)
plt.plot(res.observed, color='blue')
plt.title('Original Time Series')

# Plot the trend component
plt.subplot(412)
plt.plot(res.trend, color='green')
plt.title('Trend Component')

# Plot the seasonal component
plt.subplot(413)
plt.plot(res.seasonal, color='red')
plt.title('Seasonal Component')

# Plot the residual component
plt.subplot(414)
plt.plot(res.resid, color='purple')
plt.title('Residual Component')

plt.tight_layout()
plt.show()

#Dtrend and Seasonally Adjusted
T = res.trend
S = res.seasonal
R = res.resid

seasonal_adjusted = Temp-S
Detrended = Temp - T
plt.figure(figsize=(8,6))
plt.plot(Temp, label='Actual Data')
plt.plot(seasonal_adjusted, label='Adjusted Seasonal Data')
plt.title("Actual Data vs Adjusted Seasonal Data")
plt.xlabel("Date")
plt.ylabel('open price')
plt.grid()
plt.tight_layout()
plt.legend()
plt.show()

plt.figure(figsize=(8,6))
plt.plot(Temp, label='Actual Data')
plt.plot(Detrended, label='Detrended Data')
plt.title("Actual Data vs Detrended Data")
plt.xlabel("Date")
plt.ylabel('Open price')
plt.grid()
plt.tight_layout()
plt.legend()
plt.show()


def str_trend(T, S, R):
    FT = np.maximum(0, 1 - np.var(np.array(R)) / np.var(np.array(T + R)))
```

```
    print(f'\n Strength of trend for the raw data is {100 * FT:.3f}%')


str_trend(T, S, R)

def str_seasonal(T, S, R):
    FS = np.maximum(0, 1 - np.var(np.array(R)) / np.var(np.array(S + R)))
    print(f'Strength of seasonality for the raw data is {100 * FS:.3f}%')


str_seasonal(T, S, R)

#Split the dataset into train set (80%) and test set (20%).
from sklearn.model_selection import train_test_split

# Features (X) and Target Variable (y)
#X = differenced_data.drop(columns=['Open', 'Date'])  # Features excluding
'Open' and 'Date' columns
X = differenced_data[['High','Low','Close','Adj Close','Volume']]
X1 = differenced_data[['High','Low','Close','Adj Close','Volume','Open']]
y = differenced_data['Open_2nd_Difference']  # Target variable 'Open'
X = X.iloc[2:,:]
y = y[2:]



# Splitting the dataset into 80% train set and 20% test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42,shuffle = False)
date_train,date_test = train_test_split(differenced_data['Date'][2:],
test_size=0.2, random_state=42,shuffle = False)
yt,yf = train_test_split(data['Open'],shuffle=False,test_size=0.2)
print(f"X_train shape: {X_train.shape}, y_train shape: {y_train.shape}")
print(f"X_test shape: {X_test.shape}, y_test shape: {y_test.shape}")
print(f"date_train_test shape: {date_train.shape}, y_test shape:
{date_test.shape}")

from statsmodels.tsa.holtwinters import ExponentialSmoothing

# Holt-Winters method fitting on the train dataset
holtt = ExponentialSmoothing(yt, trend='mul', damped_trend=True, seasonal='mul',
seasonal_periods=12).fit()
holtf = holtt.forecast(steps=len(yf))
holtf = pd.DataFrame(holtf).set_index(yf.index)

# Plotting the results
plt.plot(date_train, yt[2:], label='Train Data')
plt.plot(date_test, yf, label='Test Data')
plt.plot(date_test, holtf, label='Holt Winter Forecast')
plt.title('Holt Winter (Prediction Plot)')
plt.xlabel('Time')
plt.ylabel('Value')
plt.legend()
plt.show()

#============feature Selection===================
def lse(Y, Z):
    return np.linalg.inv(Y.T.dot(Y)).dot(Y.T).dot(Z)
from numpy import linalg as LA
print("Feature Selection...\n")
H = np.matmul(X1.T, X1)
print(f'Shape of H is {H.shape}')
```

```python
s, d, v = np.linalg.svd(H)
print(f'Singular Values {d}')
print(f'The condition number is {LA.cond(X)}')
print('unknown coefficients :',lse(X_train, y_train))

print("===================================OLS
MODEL=================================")

# X = sm.add_constant(X_train)
model_1 = sm.OLS(y_train, X_train)
output1 = model_1.fit()

print(output1.summary())

print("=========================== Volume dropped
=================================")

X_train2 = X_train.drop(['Volume'],axis=1)
model2 = sm.OLS(y_train,X_train2)
output2 = model2.fit()
print(output2.summary())

print("=========================== Adj Close dropped
=================================")

X_train3 = X_train2.drop(['Adj Close'],axis=1)
model3 = sm.OLS(y_train,X_train3).fit()
# output3 = model3.fit()
print(model3.summary())

X_test_new = X_test.drop(['Volume','Adj Close'],axis=1)
y_pred_ols = model3.predict(X_test_new)

def inverse_difference(original_data, forecast, interval=1):
    new_data = np.zeros(len(original_data))
    for i in range(1, len(forecast)):
        new_data[i] = forecast[i - interval] + original_data[i - interval]
    new_data = new_data[1:]
    return new_data
open = data['Open']
y_pred_ols_inv =
inverse_difference(open[len(y_train)+1:].values,np.array(y_pred_ols),1)

# print(len(date_test))
# print(len(y_pred_ols_inv))
plt.plot(date_test, open[len(y_train)+2:].values.flatten(), label='Test Data')
plt.plot(date_test, y_pred_ols_inv, label='OLS Method Forecast')
plt.title('OLS (Prediction Plot) --> open vs time')
plt.xlabel('date')
plt.ylabel('open')
plt.legend()
plt.show()
print("\n")

plt.plot(date_test, y_test.values.flatten(), label='Test Data')
plt.plot(date_test, y_pred_ols.values.flatten(), label='OLS Method Forecast')
plt.title('OLS (Prediction Plot) --> open vs Time ')
plt.xlabel('date')
plt.ylabel('open price')
plt.legend()
plt.show()
print("\n")
```

```
ols_err = y_test-y_pred_ols
# Diagnostic Testing
print('Mean of residual error (OLS) method :', np.mean(ols_err))
print(f'variance of residual error (OLS) method :',np.var(ols_err))

#Basic Methods
#Average method
Average_train_df = pd.DataFrame()
Average_test_df = pd.DataFrame()
Average_train_df['open'] = copy.deepcopy(y_train)
Average_test_df['open'] = copy.deepcopy(y_test)

x = y_train.values
y = y_test.values

x_pred_1_step = [np.nan] * len(x)
y_pred_h_step = [np.nan] * len(y)
errors_1_step = [np.nan] * len(x)
errors_h_step = []

# perform one-step ahead prediction and calculate errors
for t in range(2, len(x)):
    x_pred_1_step[t] = np.mean(x[:t])
    errors_1_step[t] = (x[t] - x_pred_1_step[t])

# Perform h-step forecast and calculate errors

for h in range(len(y)):
    y_pred_h_step[h] = np.mean(x[:len(x)])
    errors_h_step.append(y[h] - y_pred_h_step[h])

Average_train_df['1_step_prediction'] = x_pred_1_step
Average_train_df['e'] = errors_1_step
Average_train_df['squared_error'] = Average_train_df['e'] ** 2
print("\nAverage calculation for train set")
print(Average_train_df)

Average_test_df['h_step_prediction'] = y_pred_h_step
Average_test_df['h_error'] = errors_h_step
Average_test_df['squared_error'] = Average_test_df['h_error'] ** 2
print("\n Average calculation for test set")
print(Average_test_df)

plt.plot(date_train, y_train, label='Train Data')
plt.plot(date_test, y_test, label='Test Data')
plt.plot(date_test,Average_test_df['h_step_prediction'], label='Average Method
Forecast')
plt.title('Average Method (Prediction Plot) --> Open Price vs time')
plt.xlabel('Time')
plt.ylabel('open')
plt.legend()
plt.show()

# Naive method
print("/n")
Naive_train_df = pd.DataFrame()
Naive_test_df = pd.DataFrame()
Naive_train_df['open'] = copy.deepcopy(y_train)
Naive_test_df['open'] = copy.deepcopy(y_test)
```

```
x = y_train.values
y = y_test.values

x_pred_1_step = [np.nan] * len(x)
y_pred_h_step = [np.nan] * len(y)
errors_1_step = [np.nan] * len(x)
errors_h_step = []

# perform one-step ahead prediction and calculate errors
for t in range(2, len(x)):
    x_pred_1_step[t] = x[t - 1]
    errors_1_step[t] = (x[t] - x_pred_1_step[t])

# Perform h-step forecast and calculate errors

for h in range(len(y)):
    y_pred_h_step[h] = x[t]
    errors_h_step.append(y[h] - y_pred_h_step[h])

Naive_train_df['1_step_prediction'] = x_pred_1_step
Naive_train_df['e'] = errors_1_step
Naive_train_df['squared_error'] = Naive_train_df['e'] ** 2
print("\nNaive calculation for train set")
print(Naive_train_df)
print("\n")

Naive_test_df['h_step_prediction'] = y_pred_h_step
Naive_test_df['e'] = errors_h_step
Naive_test_df['squared_error'] = Naive_test_df['e'] ** 2
print("Naive calculation for test set")
print(Naive_test_df)

plt.plot(date_train, y_train, label='Train Data')
plt.plot(date_test, y_test, label='Test Data')
plt.plot(date_test,Naive_test_df['h_step_prediction'], label='Naive Method
Forecast')
plt.title('Naive Method (Prediction Plot) --> Open price vs time')
plt.xlabel('Time')
plt.ylabel('open')
plt.legend()
plt.show()


#Drift Method


print("/n")
Drift_train_df = pd.DataFrame()
Drift_test_df = pd.DataFrame()
Drift_train_df['open'] = copy.deepcopy(y_train)
Drift_test_df['open'] = copy.deepcopy(y_test)


x = y_train.values
y = y_test.values

x_pred_1_step = [np.nan] * len(x)
y_pred_h_step = [np.nan] * len(y)
errors_1_step = [np.nan] * len(x)
errors_h_step = []
h=1
# perform one-step ahead prediction and calculate errors
```

```
for t in range(2, len(x)):
    num = x[t-1]-x[0]
    den = t-1
    x_pred_1_step[t] = x[t-1] + (h * num / den)
    errors_1_step[t] = (x[t] - x_pred_1_step[t])

# Perform h-step forecast and calculate errors
for h in range(len(y)):
    T = len(x)
    h_num = x[T-1]-x[0]
    h_den = T-1
    H = h+1
    y_pred_h_step[h] = x[t]+(H *h_num/h_den)
    errors_h_step.append(y[h] - y_pred_h_step[h])

Drift_train_df['1_step_prediction'] =x_pred_1_step
Drift_train_df['e'] = errors_1_step
Drift_train_df['squared_error'] = Drift_train_df['e'] ** 2
print("\nDrift calculation for train set")
print(Drift_train_df)

Drift_test_df['h_step_prediction'] =y_pred_h_step
Drift_test_df['e'] = errors_h_step
Drift_test_df['squared_error'] = Drift_test_df['e'] ** 2
print("\nDrift calculation for train set")
print(Drift_test_df)

plt.plot(date_train, y_train, label='Train Data')
plt.plot(date_test, y_test, label='Test Data')
plt.plot(date_test,Drift_test_df['h_step_prediction'], label='Drift Method
Forecast')
plt.title('Drift Method (Prediction Plot) --> Open price vs time')
plt.xlabel('Time')
plt.ylabel('open')
plt.legend()
plt.show()

#SES Method
SES_train_df = pd.DataFrame()
SES_test_df = pd.DataFrame()
SES_train_df['open'] = copy.deepcopy(y_train)
SES_test_df['open'] = copy.deepcopy(y_test)


x = y_train.values
y = y_test.values

x_pred_1_step = [np.nan] * len(x)
y_pred_h_step = [np.nan] * len(y)
errors_1_step = [np.nan] * len(x)
errors_h_step = []

alpha = 0.5
IC = x[0]
# perform one-step ahead prediction and calculate errors
for t in range(1, len(x)):
    x_pred_1_step[t] = x[t - 1] * alpha + (1 - alpha) * IC
    IC = x_pred_1_step[t]
    errors_1_step[t] = (x[t] - x_pred_1_step[t])

# Perform h-step forecast and calculate errors
```

```
for h in range(len(y)):
    T = len(x)
    y_pred_h_step[h] = x[T - 1] * alpha + (1 - alpha) * IC
    errors_h_step.append(y[h] - y_pred_h_step[h])

SES_train_df['1_step_prediction'] = x_pred_1_step
SES_train_df['e'] = errors_1_step
SES_train_df['squared_error'] = SES_train_df['e'] ** 2
print("\nSES calculation for train set")
print(SES_train_df)


SES_test_df['h_step_prediction'] = y_pred_h_step
SES_test_df['e'] = errors_h_step
SES_test_df['squared_error'] = SES_test_df['e'] ** 2
print("SES calculation for train set")
print(SES_test_df)

plt.plot(date_train, y_train, label='Train Data')
plt.plot(date_test, y_test, label='Test Data')
plt.plot(date_test,SES_test_df['h_step_prediction'], label='SES Method
Forecast')
plt.title('SES Method (Prediction Plot) --> Open price vs time')
plt.xlabel('Time')
plt.ylabel('open')
plt.legend()
plt.show()



#ARMA Model
print("\nARMA Model...\n")
def calculate_gpac_values(ry, nb, na):
    na += 1  # Adjusting for zero-indexing
    gpac_values = np.empty((nb, na))  # Initialize the GPAC values array

    for ar_order in range(1, na):
        numerator_matrix = np.empty((ar_order, ar_order))
        denominator_matrix = np.empty((ar_order, ar_order))
        for ma_order in range(nb):
            for i in range(ar_order):
                for j in range(ar_order):
                    if j < ar_order - 1:
                        numerator_matrix[i][j] = ry[abs(ma_order + (i - j))]
                        denominator_matrix[i][j] = ry[abs(ma_order + (i - j))]
                    else:
                        numerator_matrix[i][j] = ry[abs(ma_order + i + 1)]
                        denominator_matrix[i][j] = ry[abs(ma_order + (i - j))]

            numerator_det = round(np.linalg.det(numerator_matrix), 6)
            denominator_det = round(np.linalg.det(denominator_matrix), 6)

            if denominator_det == 0.0:
                gpac_values[ma_order][ar_order] = np.inf
            else:
                gpac_values[ma_order][ar_order] = round((numerator_det /
denominator_det), 3)

    gpac_dataframe = pd.DataFrame(gpac_values[:, 1:])  # Exclude the first
column
    gpac_dataframe.columns = [f'{i}' for i in range(1, na)]
```

```
    return gpac_dataframe

def plot_gpac_heatmap(gpac_dataframe, process):
    plt.figure(figsize=(10, 8))
    sns.heatmap(gpac_dataframe, annot=True)
    plt.title(f'GPAC Table {process}')
    plt.xlabel("AR order (na)")
    plt.ylabel("MA order (nb)")
    plt.show()


# from statsmodels.tsa.stattools import acf
# acf_values = acf(differenced_data['Open_2nd_Difference'].dropna().values,
lags)
# acf_left = acf_values[::-1]
# acf_right = acf_values[1:]
# combine_gpac = acf_left + acf_right


from statsmodels.tsa.stattools import acf
y2 = differenced_data['Open_2nd_Difference']

acf_values = acf(y2[2:], 100)


gpac =calculate_gpac_values(acf_values,7,7)
plot_gpac_heatmap(gpac, 'Open price')



import statsmodels.api as sm
from statsmodels.graphics.tsaplots import plot_acf , plot_pacf
fig = plt.figure()
plt.subplot(211)
plt.title('ACF of the generated data')
sm.graphics.tsa.plot_acf(differenced_data['Open_2nd_Difference'].dropna(),
lags=100, ax=plt.gca())
plt.subplot(212)
plt.title('PACF of the generated data')
sm.graphics.tsa.plot_pacf(differenced_data['Open_2nd_Difference'].dropna(),
lags=100, ax=plt.gca())
fig.tight_layout(pad=3)
plt.show()

from pmdarima import auto_arima
stepwise_fit = auto_arima(differenced_data['Open_2nd_Difference'].dropna(),
trace=True,
suppress_warnings=True)


# from sktime.forecasting.arima import AutoARIMA
# y = differenced_data['Open_2nd_Difference'].dropna()
# # p--->AR non-seasonal
# # q---->MA non-seasonal
# # P --->AR seasonal
# # W---->MA seasonal
# # d --- order of non-seasonal differencing
# # D --- order of seasonal differencing
# forecaster =  AutoARIMA(start_p = 0,
#                          max_p = 20,
#                          start_q = 0,
#                          max_q = 20,
```

```
#                           max_d = 5,
#
#                           stationary = True,
#                           n_fits = 20,
#                           stepwise = False
# )
# forecaster = forecaster.fit(y)
# print(forecaster.summary())


# ARMA MODEL
print("ARMA model with na = 5 and nb = 0")
na = 5
nb = 0

Arma_model = sm.tsa.ARIMA(y_train, order=(na, 0, nb), trend='c').fit()

print(Arma_model.summary())

# Print AR and MA coefficients
for i in range(na):
    print(f'The AR Coefficient a{i+1} is: {Arma_model.params[i]}')

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

# Forecast into the future
prediction = Arma_model.predict(start=0, end=8373)  # Adjust the range as needed

# Plot actual vs. predicted
fig = plt.figure(figsize=(12, 6))
plt.title('ARMA Predictions')
plt.plot(y_train, label='Actual', color='blue')
plt.plot(prediction, label='Predicted', color='red')
plt.title('Open Price vs time(ARMA(5,0)) - prediction plot')
plt.xlabel('time')
plt.ylabel('Open Price')
plt.legend()
plt.show()


Arma_df = pd.DataFrame()
Arma_df['y_train'] = y_train
Arma_df['y_train+1'] = prediction
Arma_df['error'] = Arma_df['y_train'] - Arma_df['y_train+1']
Arma_df['error_square'] = Arma_df['error']**2

Arma_df.reset_index(drop=True, inplace=True)
print(Arma_df)

lags = 100
acf_values = ACF(Arma_df['error'], lags)
left = acf_values[::-1]
right = acf_values[1:]
combine = left+right
confidence_interval= 1.96/np.sqrt(len(Arma_df))
plt.figure(figsize=(8, 5))
x_lags = list(range(-lags,lags+1))
plt.stem(x_lags, combine, markerfmt='ro', linefmt='b-', basefmt='r-')
plt.fill_between(x_lags, -confidence_interval, confidence_interval,
color='lightblue', alpha=1)
plt.xlabel('Lags')
```

```
plt.ylabel('ACF Value')
plt.title(f'ACF Plot for arma data errors (Lags = {lags})')
plt.axhline(0, color='black')
plt.grid()
plt.show()


plt.plot(date_train[:2500],Arma_df['y_train'][:2500], label = 'train set')
plt.plot(date_train[:2500],Arma_df['y_train+1'][:2500], label = '1-step
prediction')
plt.title('Open Price vs time(ARMA(5,0)) - prediction plot')
plt.xlabel('time')
plt.ylabel('Open Price')
plt.legend()
plt.tight_layout()
plt.show()


# diagnostic testing
from scipy.stats import chi2

print('parameters for confidence intervals :',Arma_model.conf_int())

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

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

print(f'zeros : {zeros}')
print(f'poles : {poles}')
Q = len(y_train)*np.sum(np.square(acf_values[lags:]))
Degree_of_freedom = lags-na-nb
alpha = 0.01
chi_critical = chi2.ppf(1-alpha, Degree_of_freedom)
print('Chi Squared test results')
if Q<chi_critical:
    print(f'The residuals is white,  and the chi_square values is :{Q}')
else:
    print(f'The residual is NOT white, chi squared value :{Q}')


# =========== h step prediction ================
forecast = Arma_model.forecast(steps=len(y_test))

Arma_df_forecast = pd.DataFrame()
Arma_df_forecast['y_test'] = y_test
Arma_df_forecast['y_test+h_step'] = forecast
Arma_df_forecast['forecast_error'] = Arma_df_forecast['y_test'] -
Arma_df_forecast['y_test+h_step']
Arma_df_forecast['forecast_error_square'] =
Arma_df_forecast['forecast_error']**2

plt.plot(date_test,y_test, label = 'test set')
plt.plot(date_test,forecast, label = 'h-step prediction')
plt.title('Open Price vs time(ARMA(5,0)) - Forecasting  plot')
plt.xlabel('time')
plt.ylabel('Open Price')
plt.legend()
```

```
plt.tight_layout()
plt.show()

print(Arma_df_forecast)
res_var = Arma_df['error'].var()
print(f'variance of residual error : {res_var}')
forecast_var = Arma_df_forecast['forecast_error'].var()
print(f'variance of forecast error : {forecast_var}')

#
# #=====Arma(4,5)
# Arma_model_2 = sm.tsa.ARIMA(y_train, order=(4, 0, 5), trend='c').fit()
#
# print(Arma_model_2.summary())
#
# # Print AR and MA coefficients
# for i in range(na):
#     print(f'The AR Coefficient a{i+1} is: {Arma_model_2.params[i]}')
#
# for i in range(nb):
#     print(f'The MA Coefficient b{i+1} is: {Arma_model_2.params[i+na]}')
#
# # Forecast into the future
# prediction_2 = Arma_model_2.predict(start=0, end=8373)  # Adjust the range as
needed
#
# # Plot actual vs. predicted
# fig = plt.figure(figsize=(12, 6))
# plt.title('ARMA Predictions')
# plt.plot(y_train, label='Actual', color='blue')
# plt.plot(prediction_2, label='Predicted', color='red')
# plt.legend()
# plt.show()
#
#
# Arma_df_2 = pd.DataFrame()
# Arma_df_2['y_train'] = y_train
# Arma_df_2['y_train+1'] = prediction_2
# Arma_df_2['error'] = Arma_df_2['y_train'] - Arma_df_2['y_train+1']
# Arma_df_2['error_square'] = Arma_df_2['error']**2
#
# Arma_df_2.reset_index(drop=True, inplace=True)
# print(Arma_df)
#
# lags = 100
# acf_res = acf(Arma_df_2['error'].values, 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(4,5))")
# plt.xlabel("Lags")
# plt.ylabel("ACF values")
# plt.grid()
# plt.legend(["ACF"], loc='upper right')
# plt.show()
#
#
#
#
# # diagnostic testing
# from scipy.stats import chi2
```

```
#
# print('parameters for confidence intervals :',Arma_model_2.conf_int())
#
# poles = []
# for i in range(na):
#     poles.append(-(Arma_model_2.params[i]))
#
# print('zero/cancellation:')
# zeros = []
# for i in range(nb):
#     zeros.append(-(Arma_model_2.params[i+na]))
#
# print(f'zeros : {zeros}')
# print(f'poles : {poles}')
# Q = len(y_train)*np.sum(np.square(acf_res[lags:]))
# Degree_of_freedom = lags-4-5
# alpha = 0.01
# chi_critical = chi2.ppf(1-alpha, Degree_of_freedom)
# print('Chi Squared test results')
# if Q<chi_critical:
#     print(f'The residuals is white,  and the chi_square values is :{Q}')
# else:
#     print(f'The residual is NOT white, chi squared value :{Q}')

#==============ARIMA===========================
na = 5
d = 2
nb = 0

Arima_model = sm.tsa.ARIMA(y_train, order=(na, d, nb)).fit()

print(Arima_model.summary())

# Print AR and MA coefficients
for i in range(na):
    print(f'The AR Coefficient a{i+1} is: {Arima_model.params[i]}')

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

# Forecast into the future
Arima_prediction = Arima_model.predict(start=0, end=8373)  # Adjust the range as
needed

# Plot actual vs. predicted
fig = plt.figure(figsize=(12, 6))
plt.title('ARIMA Predictions')
plt.plot(y_train, label='Actual', color='blue')
plt.plot(Arima_prediction, label='Predicted', color='red')
plt.title('Open Price vs time(ARIMA(5,2,0)) - prediction plot')
plt.xlabel('time')
plt.ylabel('Open Price')
plt.legend()
plt.show()


Arima_df = pd.DataFrame()
Arima_df['y_train'] = y_train
Arima_df['y_train+1'] = Arima_prediction
Arima_df['error'] = Arima_df['y_train'] - Arima_df['y_train+1']
Arima_df['error_square'] = Arima_df['error']**2
```

```python
Arima_df.reset_index(drop=True, inplace=True)
print(Arima_df)

lags = 100
acf_values = ACF(Arima_df['error'], lags)
left = acf_values[::-1]
right = acf_values[1:]
combine = left+right
confidence_interval= 1.96/np.sqrt(len(Arima_df))
plt.figure(figsize=(8, 5))
x_lags = list(range(-lags,lags+1))
plt.stem(x_lags, combine, markerfmt='ro', linefmt='b-', basefmt='r-')
plt.fill_between(x_lags, -confidence_interval, confidence_interval,
color='lightblue', alpha=1)
plt.xlabel('Lags')
plt.ylabel('ACF Value')
plt.title(f'ACF Plot for arima data errors (Lags = {lags})')
plt.axhline(0, color='black')
plt.grid()
plt.show()


plt.plot(date_train[:2500],Arima_df['y_train'][:2500], label = 'train set')
plt.plot(date_train[:2500],Arima_df['y_train+1'][:2500], label = '1-step
prediction')
plt.title('Open Price vs time(ARIMA(5,2,0)) - prediction plot')
plt.xlabel('time')
plt.ylabel('Open Price')
plt.legend()
plt.tight_layout()
plt.show()


# diagnostic testing
from scipy.stats import chi2

print('parameters for confidence intervals :',Arima_model.conf_int())

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

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

print(f'zeros : {zeros}')
print(f'poles : {poles}')
Q = len(y_train)*np.sum(np.square(acf_values[lags:]))
Degree_of_freedom = lags-na-nb
alpha = 0.01
chi_critical = chi2.ppf(1-alpha, Degree_of_freedom)
print('Chi Squared test results')
if Q<chi_critical:
    print(f'The residuals is white,  and the chi_square values is :{Q}')
else:
    print(f'The residual is NOT white, chi squared value :{Q}')


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

```
Arima_forecast = Arima_model.forecast(steps=len(y_test))

Arima_df_forecast = pd.DataFrame()
Arima_df_forecast['y_test'] = y_test
Arima_df_forecast['y_test+h_step'] = Arima_forecast
Arima_df_forecast['forecast_error'] = Arima_df_forecast['y_test'] -
Arima_df_forecast['y_test+h_step']
Arima_df_forecast['forecast_error_square'] =
Arima_df_forecast['forecast_error']**2

plt.plot(date_test,Arima_df_forecast['y_test'], label = 'test set')
plt.plot(date_test,Arima_df_forecast['y_test+h_step'], label = 'h-step
prediction')
plt.title('Open Price vs time(ARIMA(5,2,0)) - Forecasting  plot')
plt.xlabel('time')
plt.ylabel('Open Price')
plt.legend()
plt.tight_layout()
plt.show()

print(Arima_df_forecast)
# res_var = Arima_df_forecast['error'].var()
print(f"Variance of residual error: {Arima_df['error'].var()}")
print(f"Variance of forecast error:
{Arima_df_forecast['forecast_error'].var()}")
```

References:

Data Set:

https://www.kaggle.com/datasets/meetnagadia/applestock-price-

from-19802021

https://plotly.com/

Class lecture videos and lecture material

Labs and assignment code