TIME SERIES ANALYSIS AND MODELLING

FINAL PROJECT REPORT ON
AIR QUALITY PREDICTION (Beijing pm2.5)

By

Adina Dingankar

The George Washington University

Author Note

This report was prepared for DATS 6450 taught by, Prof Reza Jafari

TABLE OF CONTENTS

ABSTRACT

Air Quality is the most concerning topic in day-to-day life that's when science dealing with technology has helped to understand the dynamics of it quite better. Such data is mainly related to Time Series domain. This project focusses on prediction of Air Quality for Beijing for the year 2010-2014. Different time series methods and models have been implemented using Python to forecast the prediction and all the associated functions are been developed in python. Since the data is complex hence during the EDA pre-processing step the data is been subset for the further analysis then it goes through the various steps of time series decomposition, singular value decomposition for feature selection, model building and analysis and then summarizing the performance metrics of the methods used to find the best model for the data.

INTRODUCTION

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

To check the stationarity of the data, we perform rolling and variance on the dependant variable and plot it against time to determine the stationarity of the data. And we perform statistical test also called as Augmented Dickey-Fuller test (ADF test). It is one of the most commonly used statistical test when it comes to analysing the stationary of a series. This test is hypothesis testing where we either reject the hypothesis to support that the DataFrame is stationary or do not reject the hypothesis to support that the DataFrame is not stationary, so similar test is KPSS test which works in the reverse order to ADF test the analysis of KPSS test is the p-value should be greater than 0.05 to prove that the data is stationary. Later with all these steps, we also plot the ACF/PACF plot to validate the result of stationary.

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

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

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

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

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

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

DESCRIPTION OF THE DATASET

The dataset used for this project is named as Air Quality Prediction (Beijing pm2.5), It is been sourced from UCI and Kaggle. The dataset is focussed on predicting the air quality in Beijing, China for the period 2010-2014. The entire dataset has 43,824 rows and 12 attributes which are as shown below:

year: year of data in this row

month: month of data in this row

day: day of data in this row

hour: hour of data in this row

pm2.5: PM2.5 concentration (ug/m^3)

DEWP: Dew Point (â„ƒ)

TEMP: Temperature (â„ƒ)

PRES: Pressure (hPa)

cbwd: Combined wind direction

Iws: Cumulated wind speed (m/s)

Is: Cumulated hours of snow

Ir: Cumulated hours of rain

The dependant variable used for the project is pm2.5 which is the pollution concentration and all the other attributes like DEWP, TEMP, Pres, IWS, IS, IR are the independent variables. Since the original dataset has NAN values in the pollution attribute over the period from 2010-2013 hence, we have extracted the data only for the period ranging from 01/01/2014 to 12/31/2014 for better forecasting this was the first step which we have used for the pre-processing of the dataset.

The second step of pre-processing is in our dataset the time attribute is widely spread into 5 sub columns since the prediction is taking place on the hourly basis hence, we have combined the columns year, month, day and hour to make a Timestep column for our day and all the plots will be graphed against the Timestep.

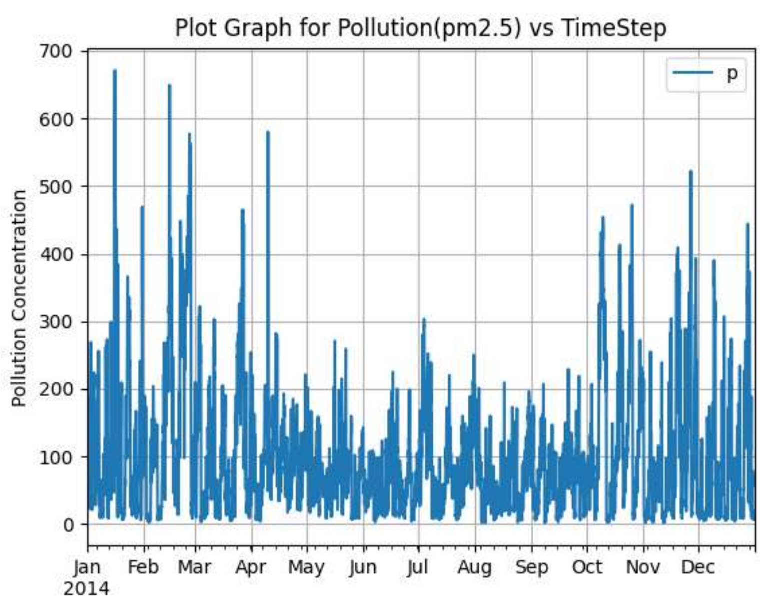


Fig1. Representation of the Pollution vs Timestep

From the plot we can see that data is trended over the period monthly from Jan 2014-Dec 2014 hence in order to validate our analysis we will plot ACF/ PACF to confirm the analysis.

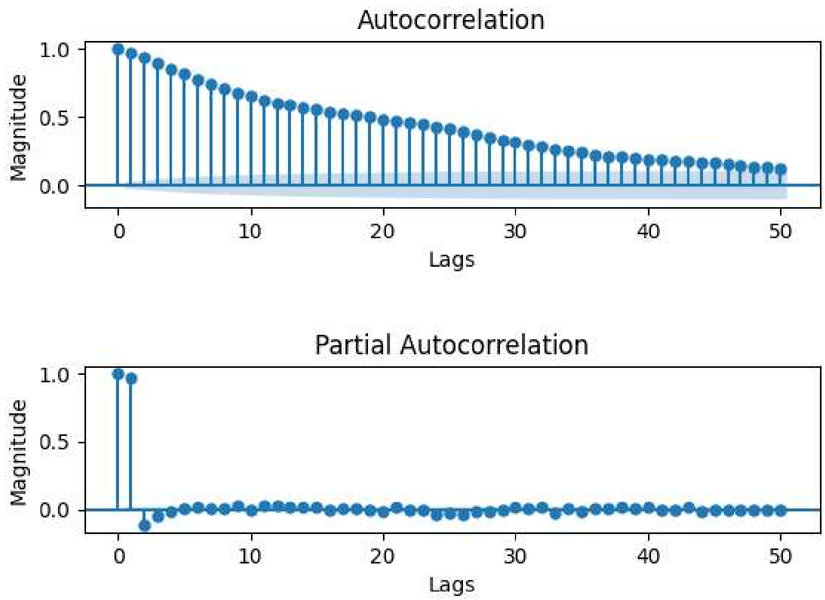


Fig2.ACF-PACF plot for lags =50

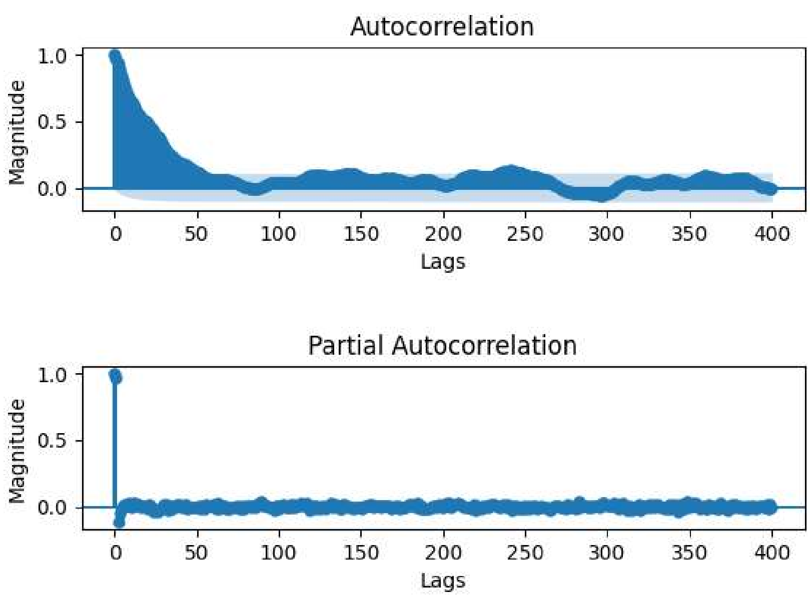


Fig3.ACF-PACF plot for lags =50

The ACF/PACF plot gives us information about auto-correlation values and partial auto-correlation values for 50 and 400 lags. And from the ACF plot, we can't see any decay hence we can say that our data is stationary. Now let's look the co-relation matrix

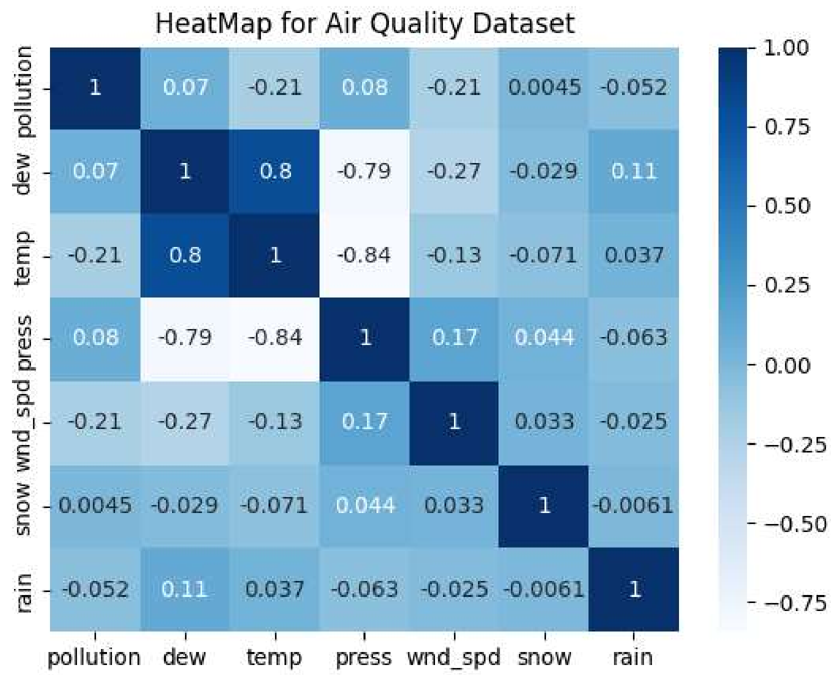


Fig4.Heat Map representation for the Air Quality Dataset

The heat map shows that the attributes temp and dew have the higher chances of correlation as we can clearly see that the pattern for both these columns is darker in association with the pollution attributes hence, we can confirm that two attributes have the correlation between each other.

Train-Test-Split (Dataset splitting using Sklearn Package)

```
#e. Split the dataset into train set (80%) and test set (20%).
data_train , data_test = train_test_split(data , shuffle=False ,
test_size=0.2)
X = data[['temp', 'press','wnd_spd', 'snow',  'rain']]
Y = data['pollution']
#X_svd = sm.add_constant(X)
X_train , X_test , y_train , y_test = train_test_split(X, Y ,
shuffle=False, test_size=0.2)
```

Fig5. Train-Test Split for the dataset

As we can clearly see that after performing pre-processing, we have renamed the columns and divided the data into 80% for training and 20% for the testing set. Since column wind_direction is a categorical column and it has no relevance with the analysis hence we have dropped the column before splitting the data for model training.

STATIONARITY

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

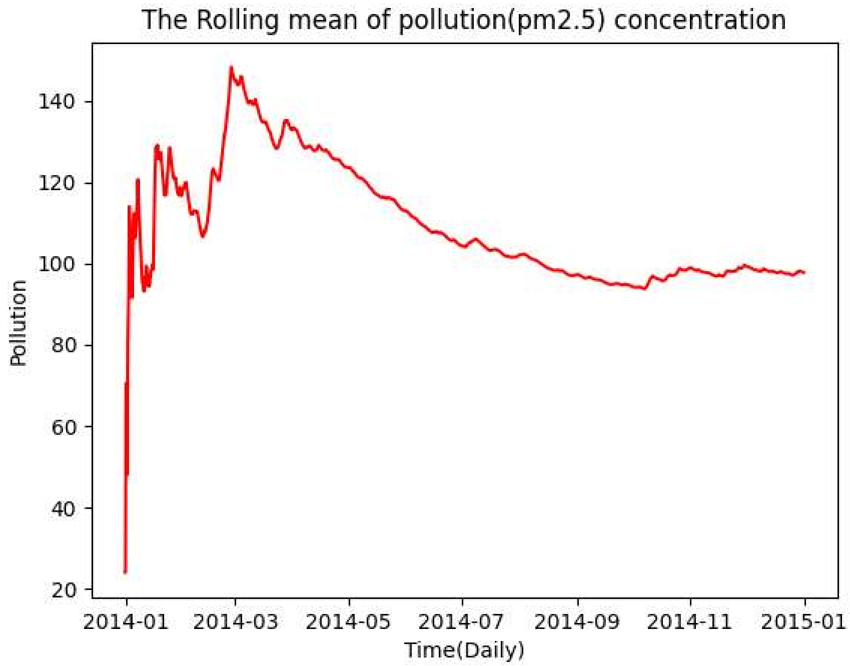


Fig6.Rolling Mean for the pollution variable (Before Differencing)

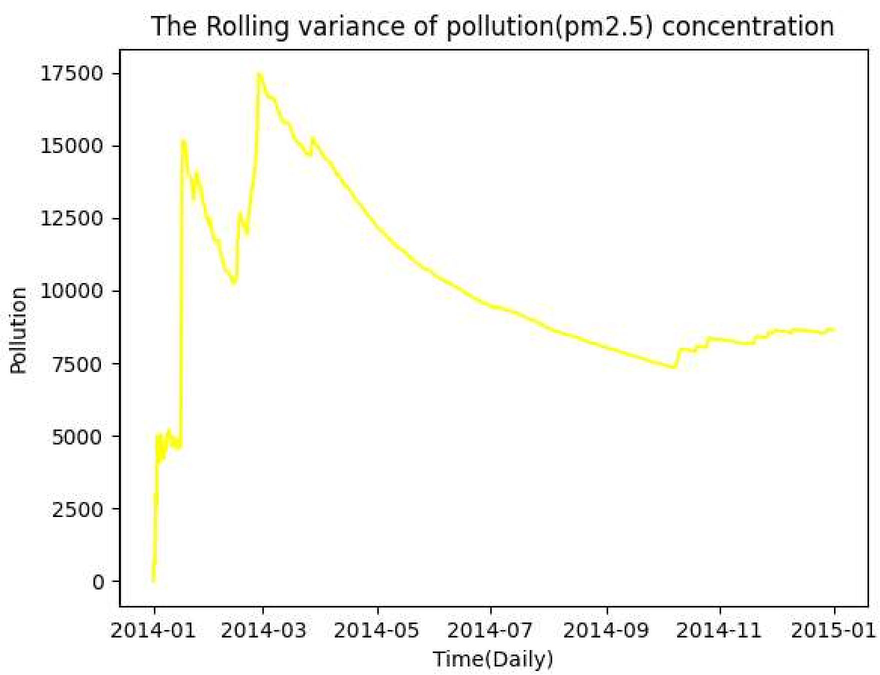


Fig7.Rolling Variance for the pollution variable (Before Differencing)

As we can observe from the rolling mean and variance of the dependant variable that mean isn't starting from zero this means the data isn't stationary hence, we will be performing the ADF and KPSS test to determine the stationarity.

ADF-TEST

```
ADF Statistic: -11.196151
p-value: 0.000000
Critical Values:
    1%: -3.431
    5%: -2.862
    10%: -2.567
```

ADF test tells us that p-value is 0.00 which is lesser than 0.05 hence we will reject the null hypothesis and hence we can claim that the data is stationary.

KPSS-TEST

```
Test Statistic             0.690718
p-value                    0.014389
Lags-Used                 54.000000
Critical Value (10%)       0.347000
Critical Value (5%)        0.463000
Critical Value (2.5%)      0.574000
Critical Value (1%)        0.739000
```

Since both of our stationarity test proves that the data is stationary but our plot of rolling mean and variance was contradicting to the analysis hence, we will be performing the non-seasonal differencing of order 1 on the data to see the difference in the rolling mean plot of the data.

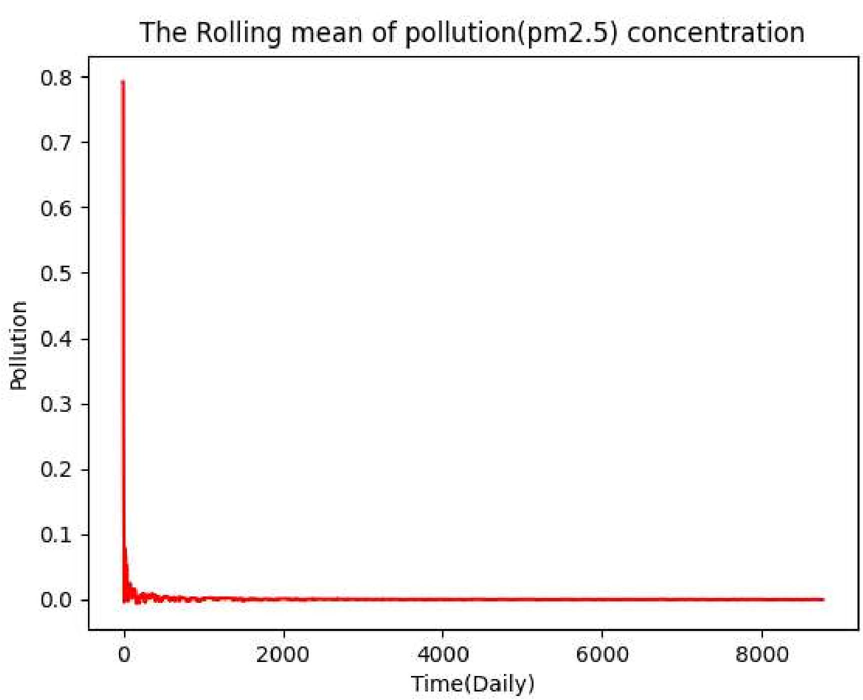


Fig8.Rolling Mean for the pollution variable (After Differencing)

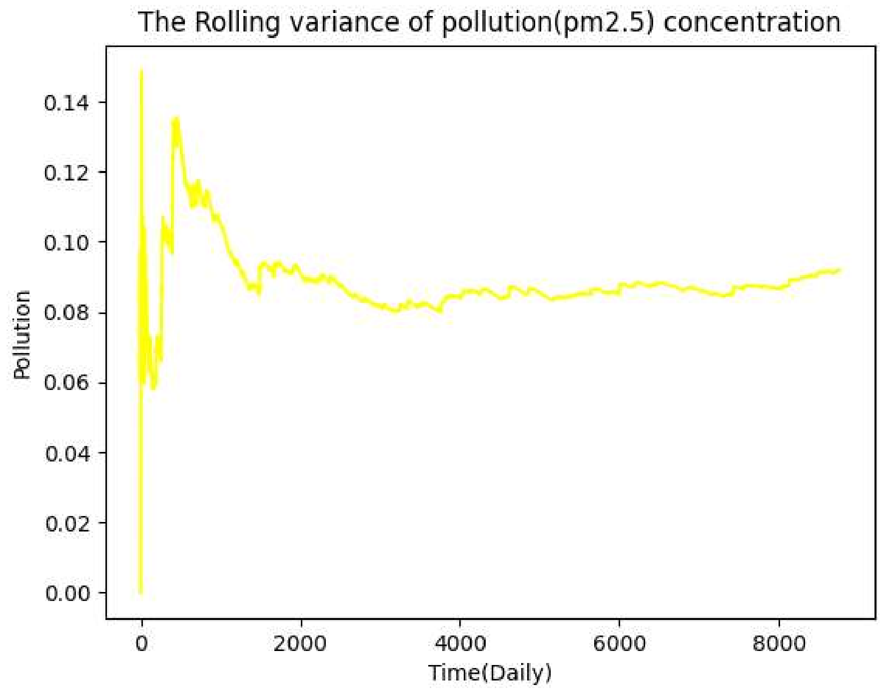


Fig9.Rolling Variance for the pollution variable (After Differencing)

After looking at the rolling mean and variance plot of differenced data we can claim that the mean is starting from zero and hence our data is stationary to be further proceeded for analysis.

TIME SERIES DECOMPOSITION

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

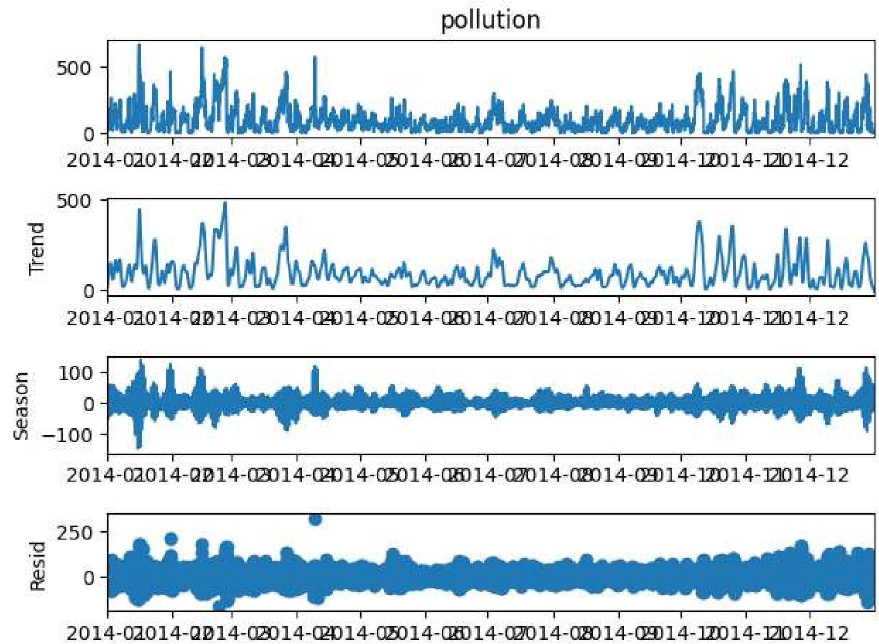


Fig10.Trend-Seasonal-Residual Plot for the Pollution

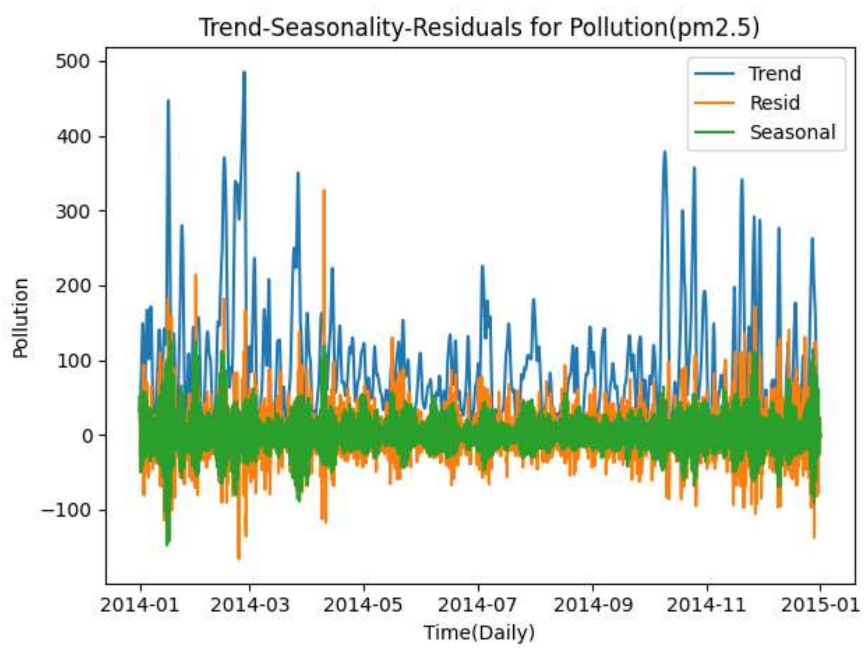


Fig11.Trend-Seasonal-Residual Plot for the Pollution

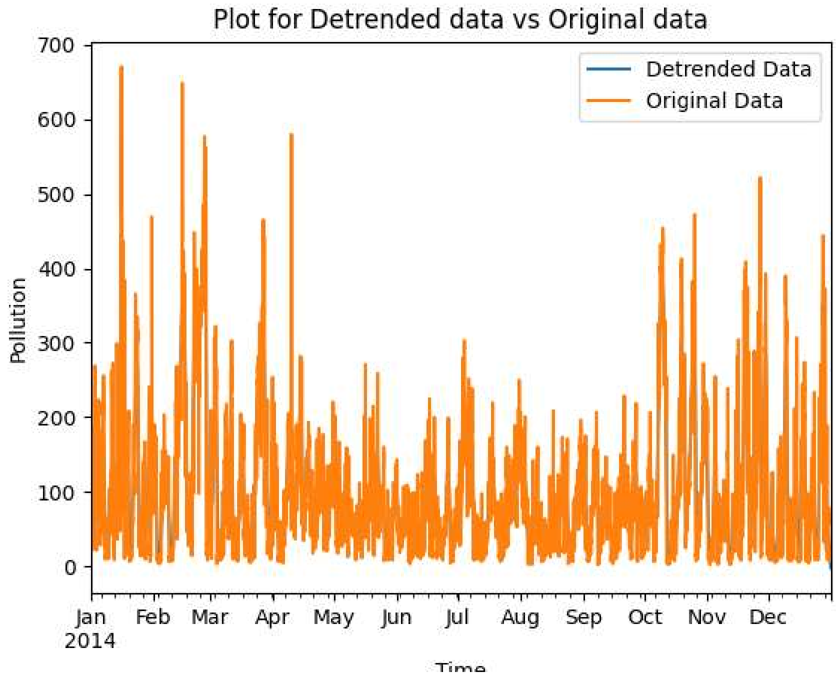


Fig12.Detrended Plot for the Pollution

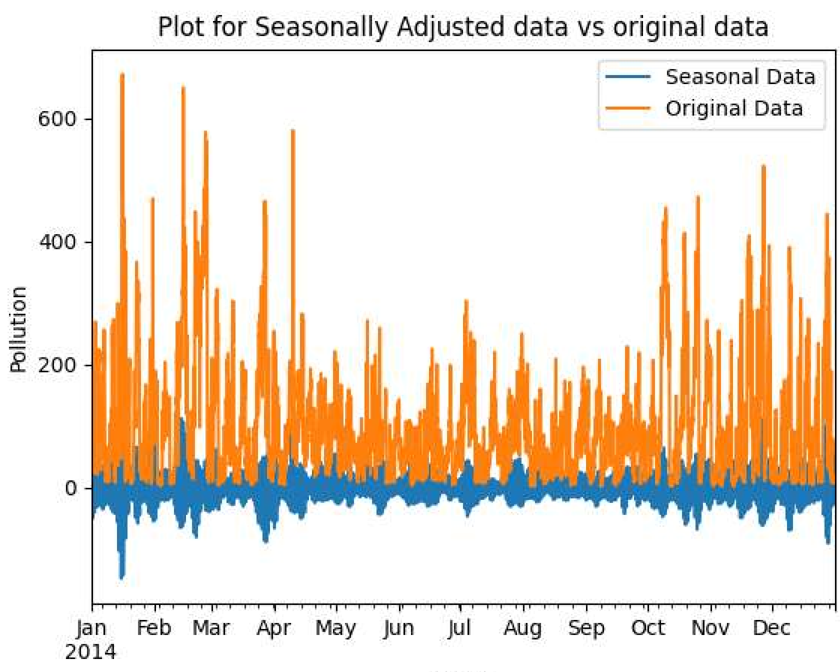


Fig13. Seasonally Adjusted Plot for the Pollution

```
The strength of trend for this data set is : 0.8835674448345994
The strength of seasonality for this data set is : 0.4108659225267651
```

After performing time series decomposition and analysing the plots we have understood that Strength of trend is close to 1 and also, we were able to see the trended spikes in the data for the period 2014 Jan -2014 Dec hence we performed the detrending on the data to feed to the base models for the prediction.

Hence, we can claim that our data is more of trended than seasonal and hence our model selection further going for ARMA will be on the basis of the conditions met previously.

HOLTS-WINTER-METHOD

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

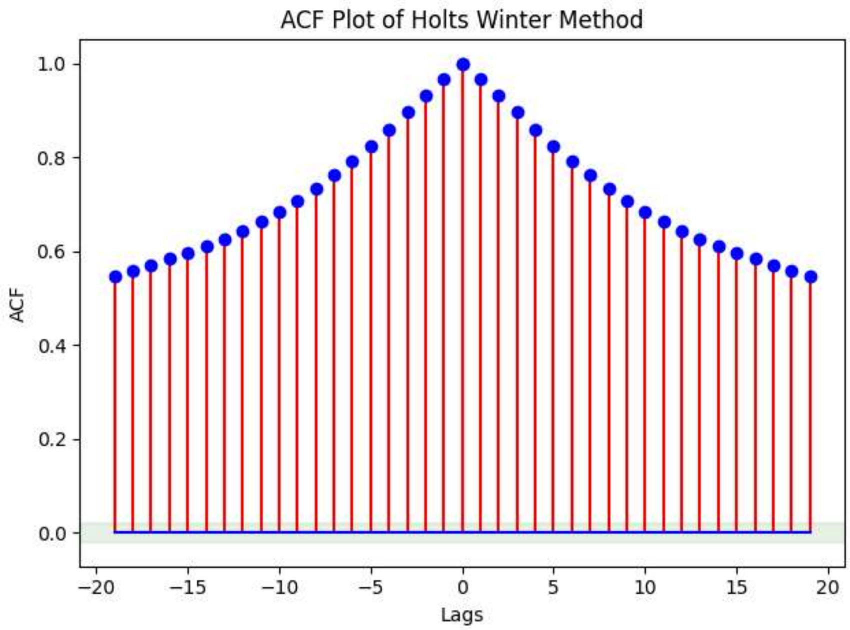


Fig14. ACF plot for Holts Winter Method

The ACF residuals of the Holts winter method is not white hence we will validate the other metrics of the model for analysis.

```
Mean square error for (training set) HLWM is  15756.02545754902
Enter the lags:>? 20
56434449.34768591
ACF is : [1.0, 0.968718181881095, 0.9330728484996474, 0.8965808456974041, 0.8596828449917657,

The Q value of residual using HWM is   77028.14023947318
The Mean of residual of HLWM is 87.76767363129004
The variance of residual of HLWM is 8052.860922900377
Mean square error for Holt Winter method is of testing set is  19439.818520414894
The Mean of forecast of HLWM is 92.3108672824683
The variance of forecast of HLWM is 10918.52230197342

 The ratio of resid vs forecast is 0.7375412807871352
```

The Mean square error of Testing data for holt's winter method is 19439.81 which is relatively high. Hence, we will compare with other models.

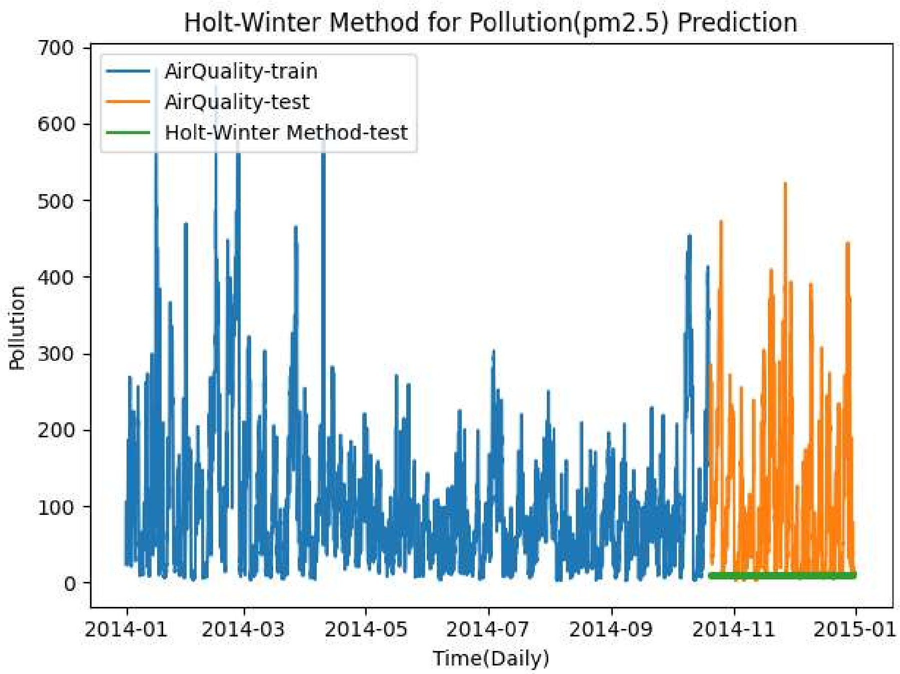


Fig15. Plot for Holts Winter Method for Pollution Prediction

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

FEATURE SELECTION

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

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

Let's check the singular values of our model

```
SingularValues = [7.21798051e+09 5.80064281e+06 9.13750162e+05 9.97645496e+03
 4.29183166e+03 1.79348949e-01]
```

```
The condition number constant (original data) = 200612.7058783319
```

Based on the above results, we can see that there are some numbers closer to zero which indicates that they have to be removed. From this we can say that most probably we have to remove 1 to 2 features because the value of singular values is closer to 0 (in this case the values are not more than 5). Similarly, the condition number greater than 1000 represents there is severe degree of collinearity. Thus, based on the results we implemented OLS calculation and get the summary for regression.

```
------------------------------------------------------------------------------
Dep. Variable:              pollution   R-squared:                       0.114
Model:                            OLS   Adj. R-squared:                  0.113
Method:                 Least Squares   F-statistic:                     180.1
Date:                Wed, 15 Dec 2021   Prob (F-statistic):          6.24e-181
Time:                        12:34:11   Log-Likelihood:                -41039.
No. Observations:                7008   AIC:                         8.209e+04
Df Residuals:                    7002   BIC:                         8.213e+04
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       1886.7901    199.683      9.449      0.000    1495.351    2278.229
temp          -3.4161      0.165    -20.764      0.000      -3.739      -3.094
press         -1.7018      0.195     -8.744      0.000      -2.083      -1.320
wnd_spd       -0.4784      0.035    -13.571      0.000      -0.548      -0.409
snow          -1.6309      1.292     -1.263      0.207      -4.163       0.901
rain          -4.9589      0.848     -5.846      0.000      -6.622      -3.296
==============================================================================
Omnibus:                     2418.768   Durbin-Watson:                   0.078
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             9085.837
Skew:                           1.705   Prob(JB):                         0.00
Kurtosis:                       7.415   Cond. No.                     2.01e+05
==============================================================================
```

As we can see that the attribute snow has p-value 0.207 hence that feature must be eliminated to see the changes in the condition number and accuracy of the model.

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              pollution   R-squared:                       0.114
Model:                            OLS   Adj. R-squared:                  0.113
Method:                 Least Squares   F-statistic:                     224.8
Date:                Wed, 15 Dec 2021   Prob (F-statistic):         8.65e-182
Time:                        12:34:12   Log-Likelihood:                -41040.
No. Observations:                7008   AIC:                         8.209e+04
Df Residuals:                    7003   BIC:                         8.212e+04
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       1877.8324    199.565      9.410      0.000    1486.624    2269.041
temp          -3.3995      0.164    -20.728      0.000      -3.721      -3.078
press         -1.6932      0.195     -8.705      0.000      -2.075      -1.312
wnd_spd       -0.4813      0.035    -13.681      0.000      -0.550      -0.412
rain          -4.9512      0.848     -5.837      0.000      -6.614      -3.288
==============================================================================
Omnibus:                     2424.012   Durbin-Watson:                   0.078
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             9118.787
Skew:                           1.708   Prob(JB):                         0.00
Kurtosis:                       7.422   Cond. No.                     2.00e+05
==============================================================================
```

After removing the snow column, we can still observe that the condition number hasn't yet reduced to 100 to claim that model is a good one to also the adjusted Rsquare is 0.114 which shows that the model is working relatively poor on this data hence we will be dropping the column const to see any changes.

Dropping Column Const

```
                            OLS Regression Results
==============================================================================================
Dep. Variable:                pollution   R-squared (uncentered):                   0.585
Model:                              OLS   Adj. R-squared (uncentered):              0.585
Method:                   Least Squares   F-statistic:                              2469.
Date:                  Wed, 15 Dec 2021   Prob (F-statistic):                        0.00
Time:                          12:34:12   Log-Likelihood:                         -41084.
No. Observations:                  7008   AIC:                                  8.218e+04
Df Residuals:                      7004   BIC:                                  8.220e+04
Df Model:                             4
Covariance Type:              nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
temp          -2.1001      0.089    -23.592      0.000      -2.275      -1.926
press          0.1371      0.002     75.279      0.000       0.133       0.141
wnd_spd       -0.5001      0.035    -14.150      0.000      -0.569      -0.431
rain          -4.4734      0.852     -5.251      0.000      -6.143      -2.803
==============================================================================
Omnibus:                       2315.498   Durbin-Watson:                   0.075
Prob(Omnibus):                    0.000   Jarque-Bera (JB):             8310.855
Skew:                             1.644   Prob(JB):                         0.00
Kurtosis:                         7.202   Cond. No.                         851.
==============================================================================
```

Now in this OLS chart we can clearly see that our model's adjusted R square has increased to 0.5% which is relatively better than previous performance also the condition number has drastically reduced in addition to that all the other features have p-value as 0.00 hence there aren't any features left for the further elimination.

MULTIPLE LINEAR REGRESSION

```
                                 OLS Regression Results
=========================================================================================
Dep. Variable:              pollution   R-squared (uncentered):                    0.585
Model:                            OLS   Adj. R-squared (uncentered):               0.585
Method:                 Least Squares   F-statistic:                               2469.
Date:                Wed, 15 Dec 2021   Prob (F-statistic):                         0.00
Time:                        12:34:12   Log-Likelihood:                          -41084.
No. Observations:                7008   AIC:                                   8.218e+04
Df Residuals:                    7004   BIC:                                   8.220e+04
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
temp          -2.1001      0.089    -23.592      0.000      -2.275      -1.926
press          0.1371      0.002     75.279      0.000       0.133       0.141
wnd_spd       -0.5001      0.035    -14.150      0.000      -0.569      -0.431
rain          -4.4734      0.852     -5.251      0.000      -6.143      -2.803
==============================================================================
Omnibus:                     2315.498   Durbin-Watson:                   0.075
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             8310.855
Skew:                           1.644   Prob(JB):                         0.00
Kurtosis:                       7.202   Cond. No.                         851.
==============================================================================
```

Now we will be performing the T-test to validate the performance.

T -Test:

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

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

```
T-test P values :  temp       1.761649e-118
press       0.000000e+00
wnd_spd     7.690290e-45
rain        1.560067e-07
```

Here the p-value of t-test for all of the features is less than 0.05. Thus, we reject null hypothesis and support that the features of the final model are significant. In further words it can be said that there shouldn't be any feature elimination.

F-test:

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

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

```
F-statistic:  2468.5915784032004
Probability of observing value at least as high as F-statistic  0.0
```

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

Let’s have a look on the performance metrics of the Multiple Linear Regression Model.

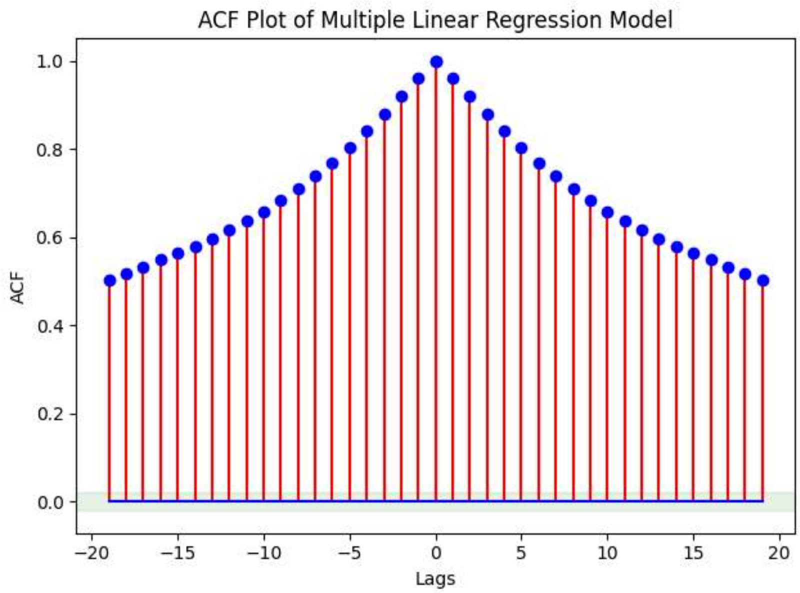


Fig16. Plot for Multiple Linear Regression for Pollution Prediction

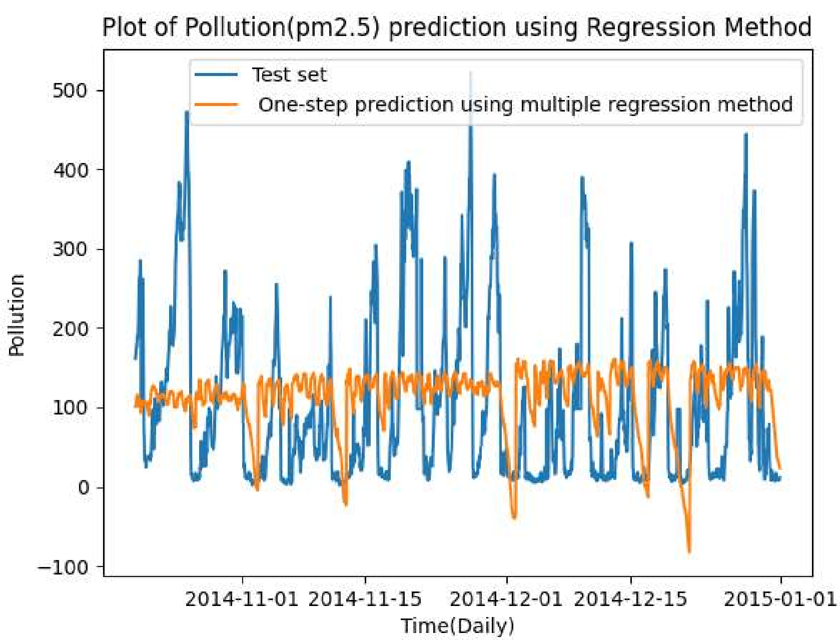


Fig17. Plot for Multiple Linear Regression for One Step ahead Prediction

```
Mean square error of training set for multiple regression is  7237.0985325532865
RMSE for training set using multiple regression is : 7237.0985325532865
Enter the lags:>? 20
50717570.28997721
ACF is : [1.0, 0.9622483954235185, 0.9215697178896395, 0.8807899497591605, 0.841043426605065,
The Q value of residual of regression is 72486.53245904138
The mean of residuals is 0.04811835557659166
The variance of residual is 7237.096217177143

Mean square error for testing set multiple regression is  9916.488634717207
RMSE for testing set using multiple regression is :99.58156774582939
The mean of forecast of multiple regression is -12.503436803929896
The variance of foreacast of multiple regression is 9760.152702807338

 The ratio of resid vs forecast is 0.7414941587026112
```

We can observe that MSE of the training set is lesser than the testing set but the Q-value of the model is relatively higher than expected hence we will further move to compare with other models.

BASE MODEL

Average Method

We will start with the Average method to perform the model training on the data.

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

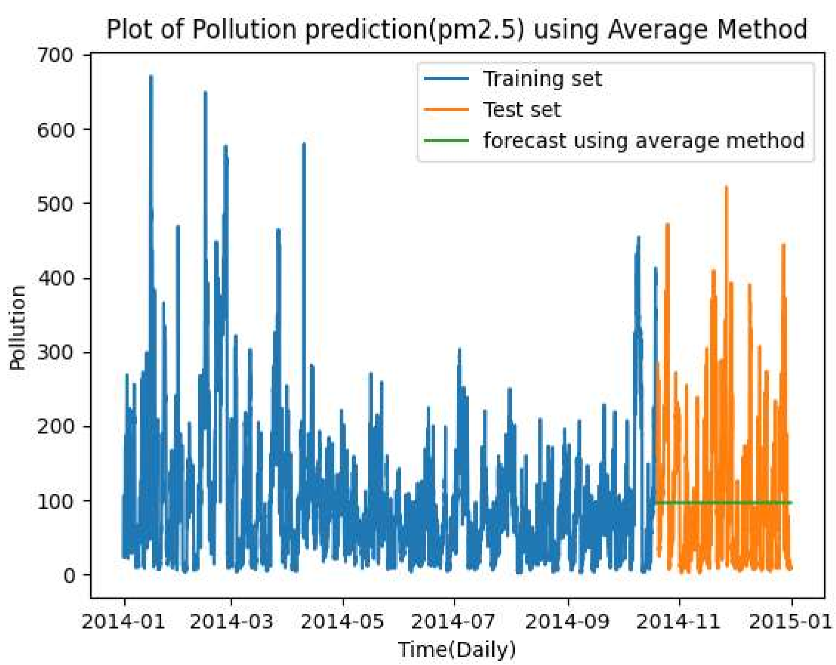


Fig18. Average method plot for Pollution VS Timestep

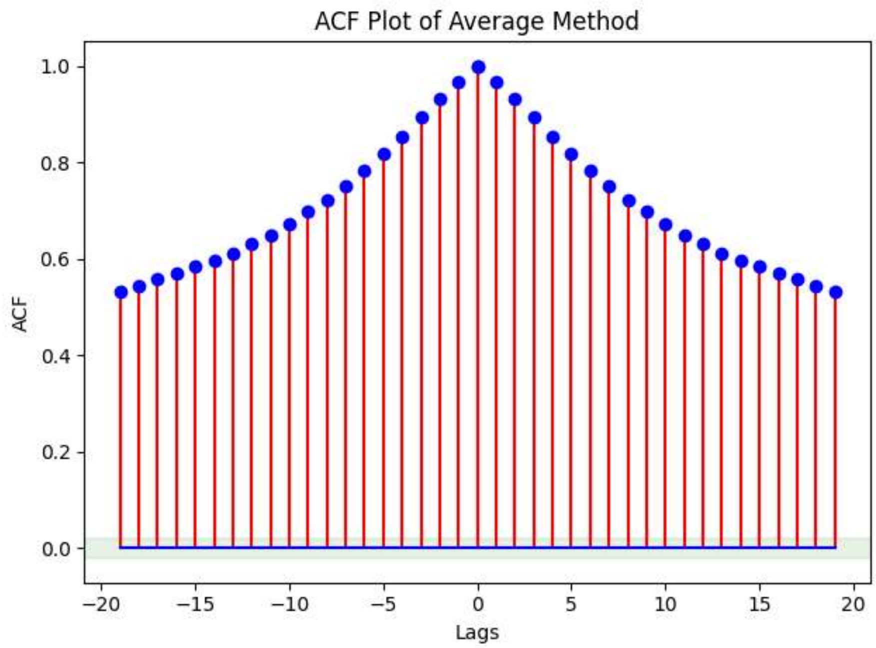


Fig19. ACF plot for Average Method

We will now verify the model performance metrics to observe how it did perform on the data.

```
MSE of prediction error (training set) using average method is : 8073.581353026532
Enter the lags:>? 20
54938701.69496809
ACF is : [1.0, 0.9677892245885512, 0.9308812089200558, 0.8929594871522525, 0.8545765858765549, 0.818246827167
The Q value of residual using average method is 75297.67931445534
The mean of residuals using average method is -15.265514796872154
The variaince of residual using average method is 7840.545411013009

MSE of forecast (testing set) using average method is : 10988.730041071889
Mean of forecast error is: 4.5267135611325156
Variance of forecast error is: 10968.238905407348

 The ratio of resid vs forecast of average method is 0.7148408672195875
```

We can see that mean square error of the training set is lesser than the testing data but the q-value of the data is very high hence we will go ahead to perform naïve method on the data.

NAIVE Method

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

Let’s plot the ACF and Naïve method plot to understand the model better.

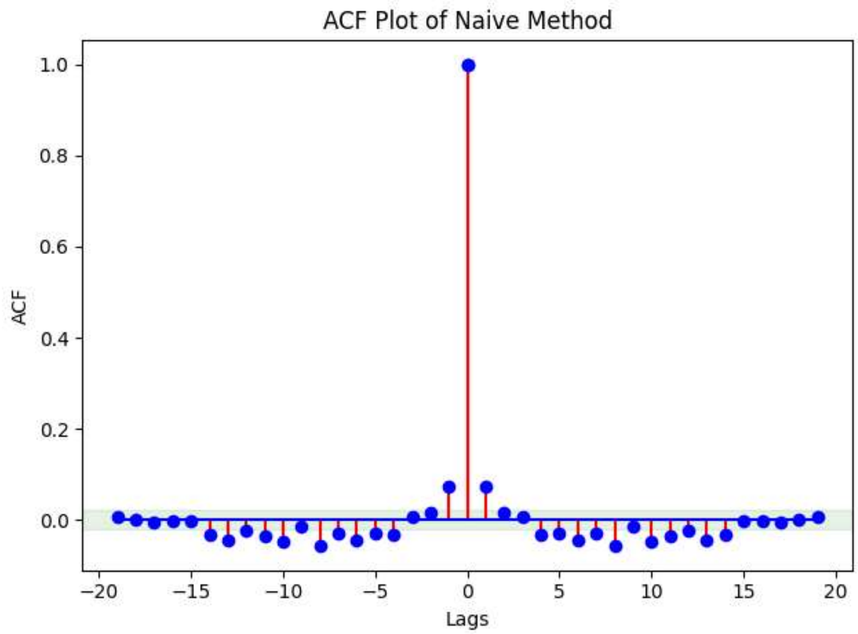


Fig20. ACF plot for Naive Method

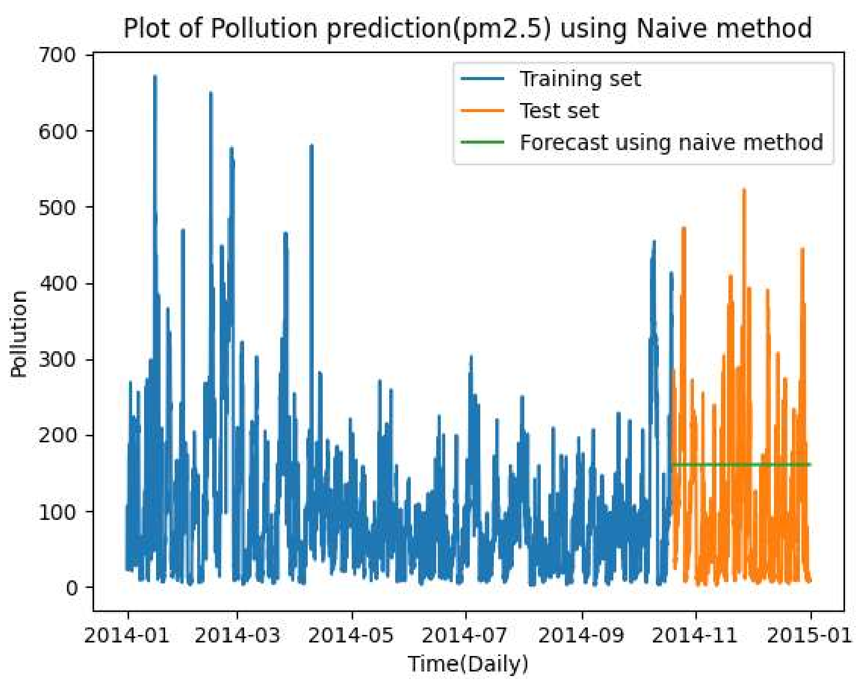


Fig21. ACF plot for Naive Method

```
MSE of prediction error (training set) using naive method is : 503.5219178722312
Enter the lags:>? 20
3528175.3999233353
ACF is : [1.0, 0.07288474687998511, 0.015541221291732755, 0.007256824931380603, -0.0318288419404406, -0.03105
The Q value of residual using naive method is 7154.857592952213
The mean of residuals using naive method is 0.01955187669473384
The variance of residual using naive method is 503.5215355963489

MSE of prediction error (testing set) using naive method is : 14525.65422306404
Mean of forecast error using naive method is: -59.64407194061027
Variance of forecast error using naive method is: 10968.23890540735

 The ratio of resid vs forecast of naive method is 0.04590723633382132
```

The Residuals for the training data are white which proves that the model is a good estimator but still we will perform the different base models in order to determine the best fit.

DRIFT Method

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

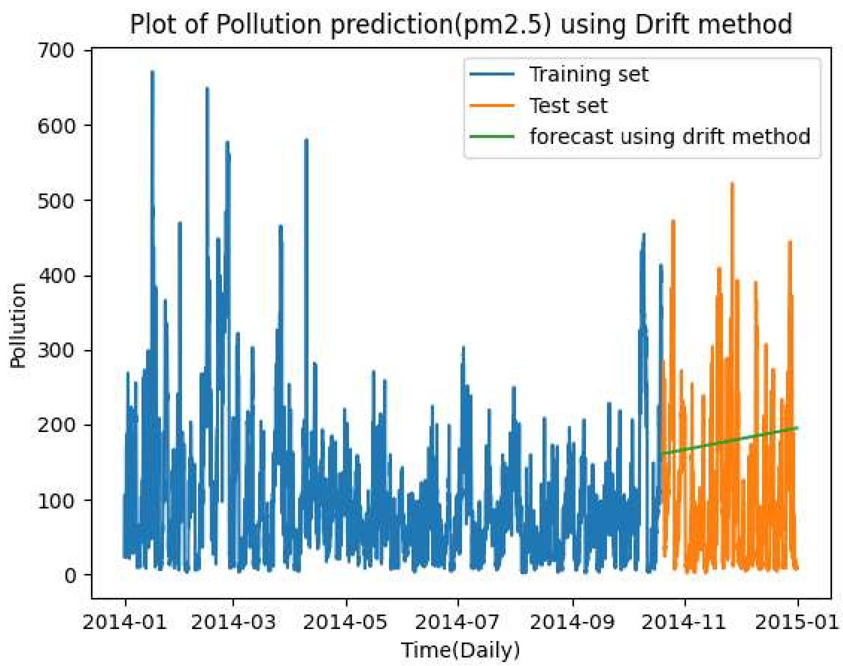


Fig22. Plot for Drift Method for the Pollution Prediction

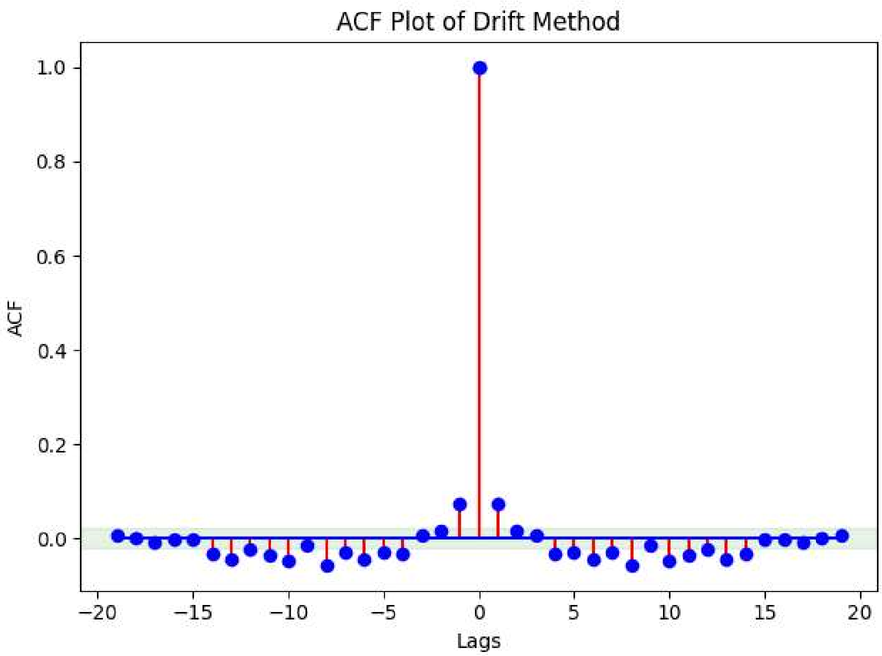


Fig23. ACF Plot for Drift Method for the Pollution Prediction

```
MSE of prediction error (training set) using drift method is : 504.3090529970017
Enter the lags:>? 20
3533145.5471837744
ACF is : [1.0, 0.0732401676893391, 0.015866062707883547, 0.007513804706171743, -0.03146823043715925,
The Q value of residual using drift method is 7153.199791983748
The mean of residuals using drift method is -0.07895813255433069
The variance of residual using drift method is 504.30281861030517

MSE of prediction error of testing set using drift method is : 17168.86521234145
Mean of forecast error using drift method is: -76.7812918635445
Variance of forecast error using drift method is: 11273.49843210665

 The ratio of resid vs forecast of drift method is 0.044733480174535975
```

From the result we can see that MSE of the testing set is higher than the training set which is our forecasting data also the q value is 7153 which is relatively better than Naïve model and also the residuals are white. Still, we will proceed further with other base models.

Simple Smoothing Exponential (SES)

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

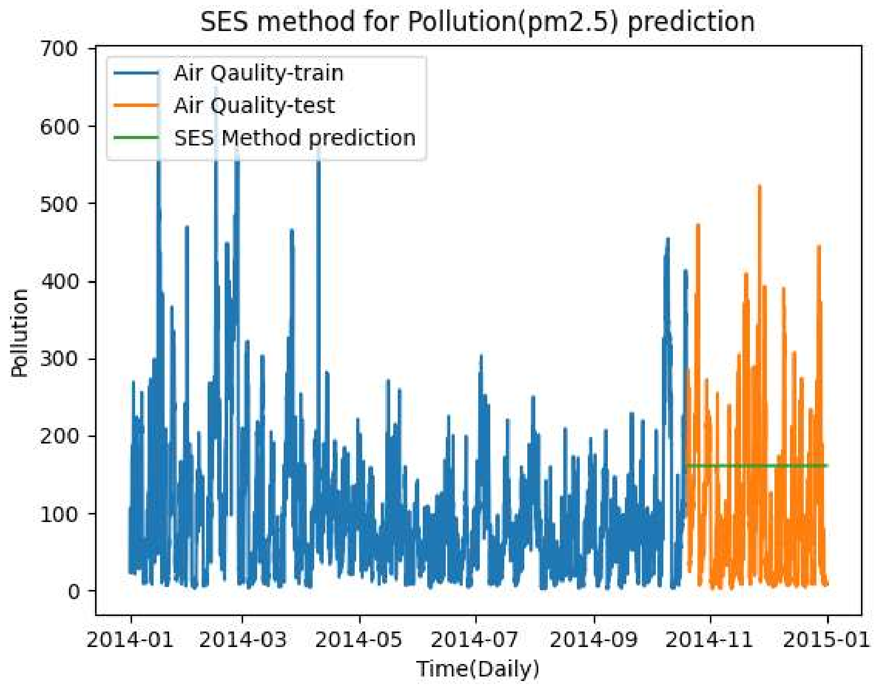


Fig24. Plot for Simple Smoothing Exponential for the Pollution Prediction

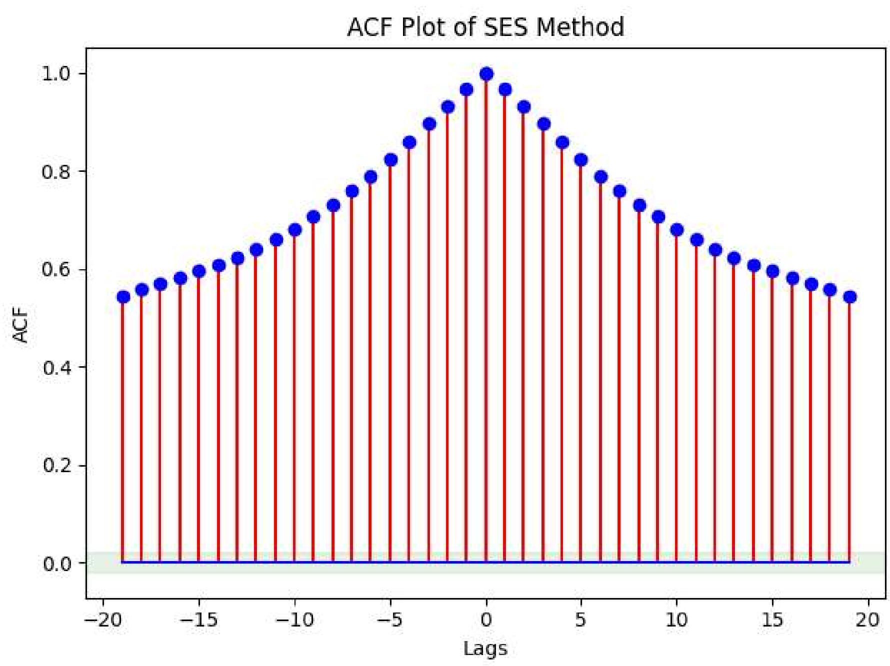


Fig25. ACF Plot for SES for the Pollution Prediction

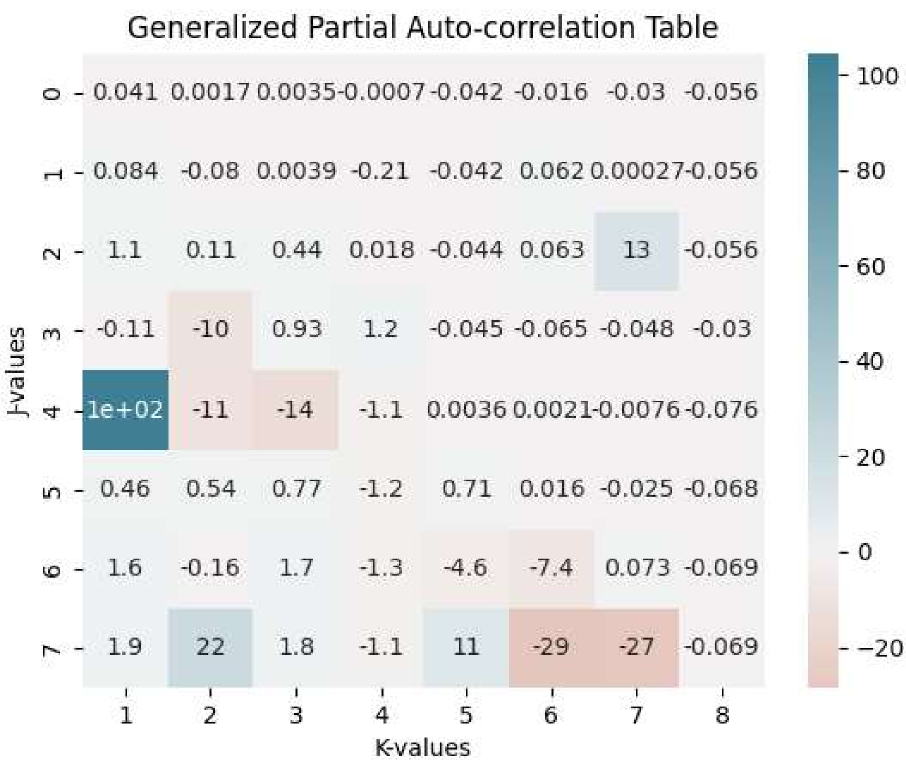
```
Mean square error for (training set) simple exponential smoothing is  12193.061668977316
Enter the lags:>? 20
56513875.80900519
ACF is : [1.0, 0.9687014913409793, 0.9328707953597192, 0.8960637014616821, 0.8588065541915251, 0.823552589192
The Q value of residual using SES method is 76798.48626426388
Mean of residual using SES method is: -64.2562608741581
Variance of residual using SES method is: 8064.194607449459
Mean square error for (testing set) simple exponential smoothing is  14535.857727626319
Mean of forecast error using SES method is: -59.72954731302566
Variance of forecast error using SES method is: 10968.238905407348

 The ratio of resid vs forecast of SES method is 0.7352314876615066
```

Since the Q-value of the SES method is very high in comparison to the Naïve and Drift method hence we will be proceeding the Auto Regressive Models approach.

ARMA and ARIMA model

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

We have implemented GPAC first on the differenced dataset to understand the need of ARIMA model for our data and here's the plot of gpac.



J \ K	1	2	3	4	5	6	7	8
0	0.041	0.0017	0.0035	-0.0007	-0.042	-0.016	-0.03	-0.056
1	0.084	-0.08	0.0039	-0.21	-0.042	0.062	0.00027	-0.056
2	1.1	0.11	0.44	0.018	-0.044	0.063	13	-0.056
3	-0.11	-10	0.93	1.2	-0.045	-0.065	-0.048	-0.03
4	1e+02	-11	-14	-1.1	0.0036	0.0021	-0.0076	-0.076
5	0.46	0.54	0.77	-1.2	0.71	0.016	-0.025	-0.068
6	1.6	-0.16	1.7	-1.3	-4.6	-7.4	0.073	-0.069
7	1.9	22	1.8	-1.1	11	-29	-27	-0.069

Fig25. GPAC plot for the Differenced Dataset

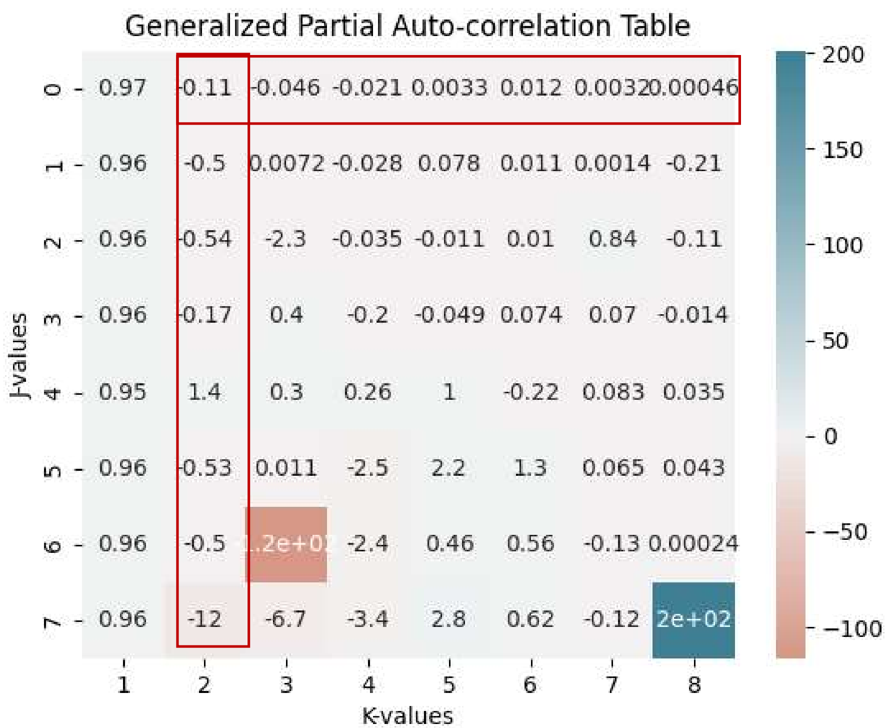


Fig26. GPAC plot for the Original Dataset

Let's See the ARMA model for the pattern (2,0)

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

```
For our estimated ARMA (2,0):
Mue is high now and cannot go higher than that!!!
The estimated parameters >>>  [-1.08914979  0.10564303]

 The estimated co-variance matrix is [[ 0.00011291 -0.00011122]
 [-0.00011122  0.00011291]]

 The estimated variance of error is 533.0709507292935
The standard deviation for a1 is 0.010625790130836506
The standard deviation for a2 is 0.010625792262655493

 The confidence interval for parameters are:
-1.1104013698556658 < a1 < -1.0678982093323197
0.0843914454109563 < a2 < 0.12689461446157826

The roots of the numerators are []
The roots of dinominators are [0.98151744 0.10763235]
```

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

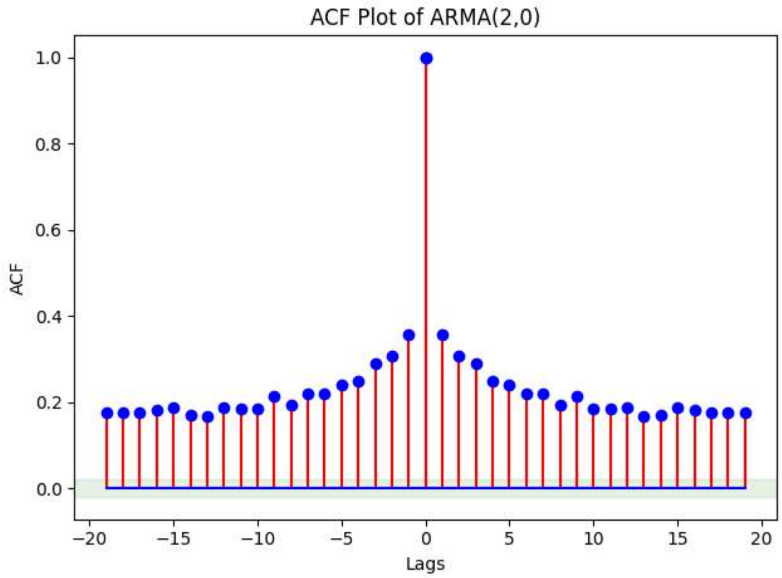


Fig27. ACF plot for ARMA (2,0)

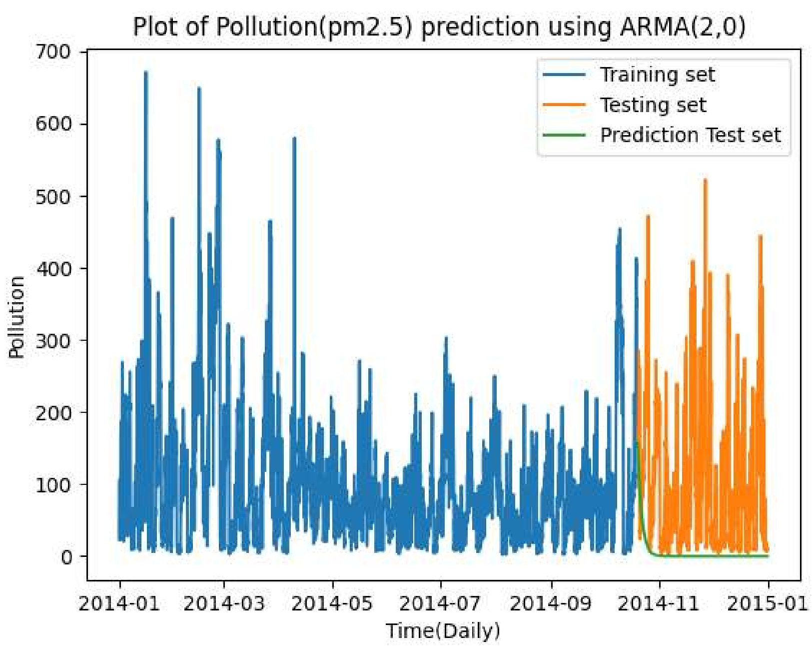


Fig28. Plot for ARMA (2, 0) for Pollution Prediction

Let's see the Q value.

```
The Q value is   13462.084317699166
The chi-critical is 7284.312853053468

The Q value is not less than 7284.312853053468 (chi-critical) so, the residual is not white
[-2.13959495 -2.18950613 -0.19565574  0.62631167  0.35217872]
```

Let's perform the One-step ahead prediction to see the values obtained for the residuals

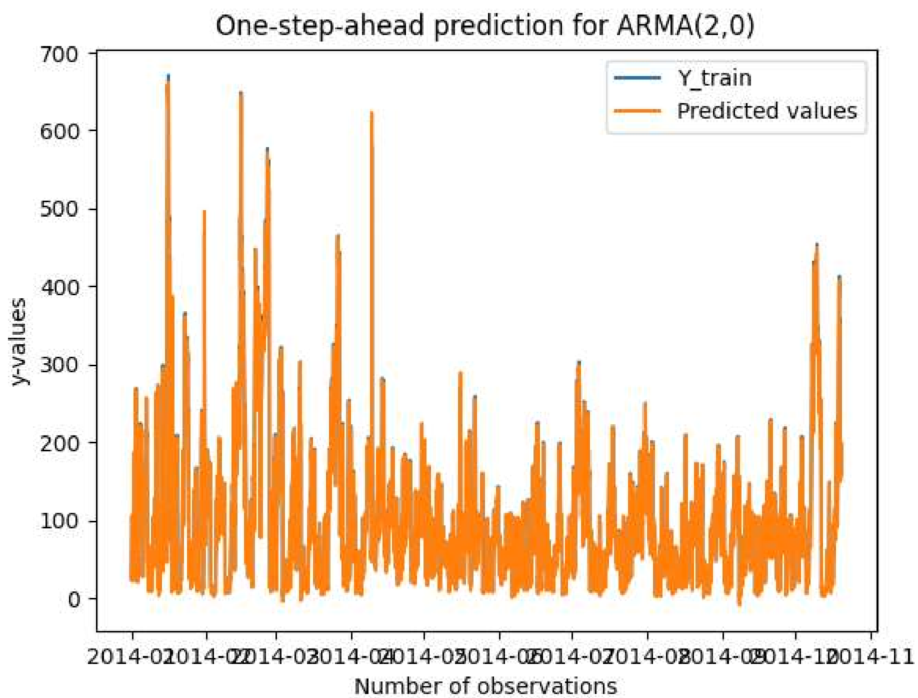


Fig29. Plot for ARMA (2, 0) for One Step Ahead Prediction

```
MSE of training data for ARMA(2,0) is 9.480024573151569
Mean of residual(training data) with ARMA(2,0) is 1.5946004909173956
The variance of residual(training data) with ARMA(2,0) is 6.937273847517569

MSE of forecast for ARMA(2,0) is 20643.461581370106
The mean of testing data is 96.5269077647003
The variance of testing data is 11326.017658755132
```

After looking at the q-value of the model we can see that it is greater than chi-critical and hence the residual is not white. In addition to that we did perform one step ahead prediction we can see that mean of the residual is not closer to 0 but since the q value predicts that the residual is not white hence, we need to discard the model and proceed further to observe the other potential pattern.

ARMA (2,1)

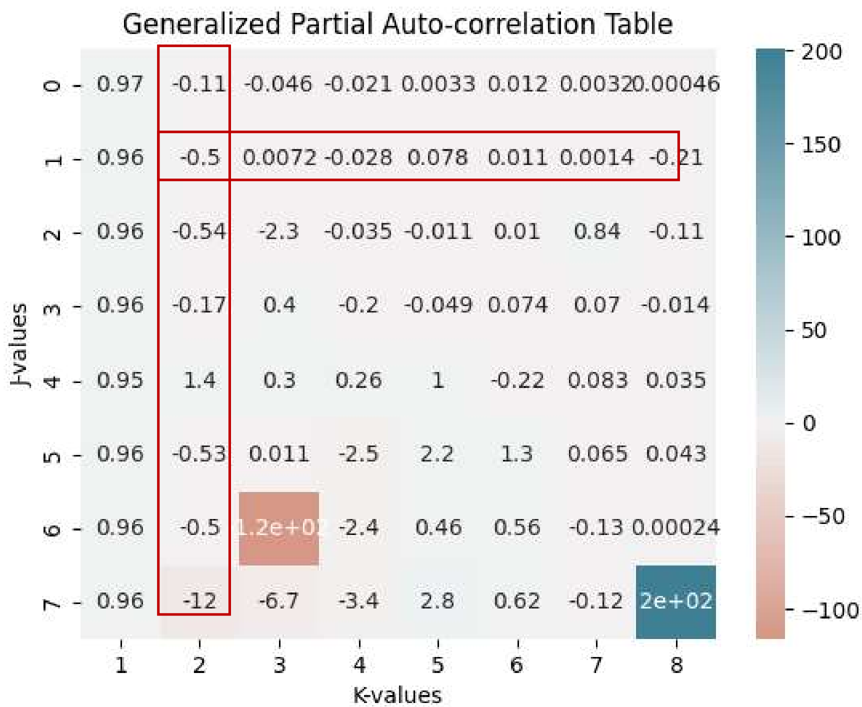


Fig30. GPAC for ARMA (2,1)

We have formed the pattern ARMA (2,1) for our model

```
The estimated parameters >>>  [-1.36241583  0.37486606 -0.27555201]

 The estimated co-variance matrix is [[ 0.0073506  -0.00723448  0.00758019]
 [-0.00723448  0.00712197 -0.00745975]
 [ 0.00758019 -0.00745975  0.00792275]]

 The estimated variance of error is 532.5044943508112
The standard deviation for a1 is 0.0857356512612694
The standard deviation for a2 is 0.08439177918672627
The standard deviation for b2 is 0.0890098490238727

 The confidence interval for parameters are:
-1.533887134121611 < a1 < -1.190944529076535
0.20608250099269493 < a2 < 0.5436496177396
-0.45357170835540384 < b1 < -0.09753231225991077

The roots of the numerators are [0.27555201]
The roots of dinominators are [0.97983505 0.38258078]
```

The estimated parameters for our model ARMA (2,1) are -1.36, 0.37, -0.27 and we have got the roots of the denominator and we don't have zero/pole cancellation but the confidence intervals of the parameters don't include zero hence we can say that the parameters are statistically significant.

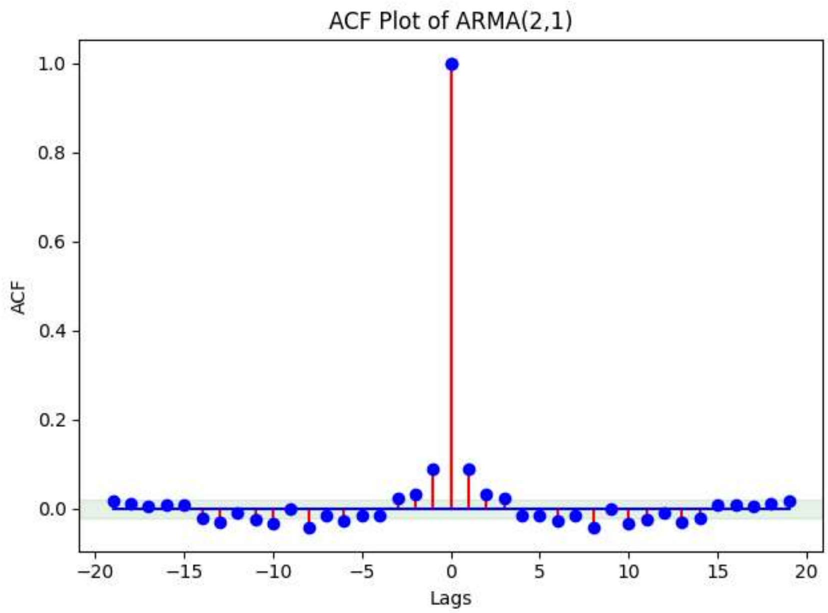


Fig31. Representation of ACF plot for ARMA (2, 1)

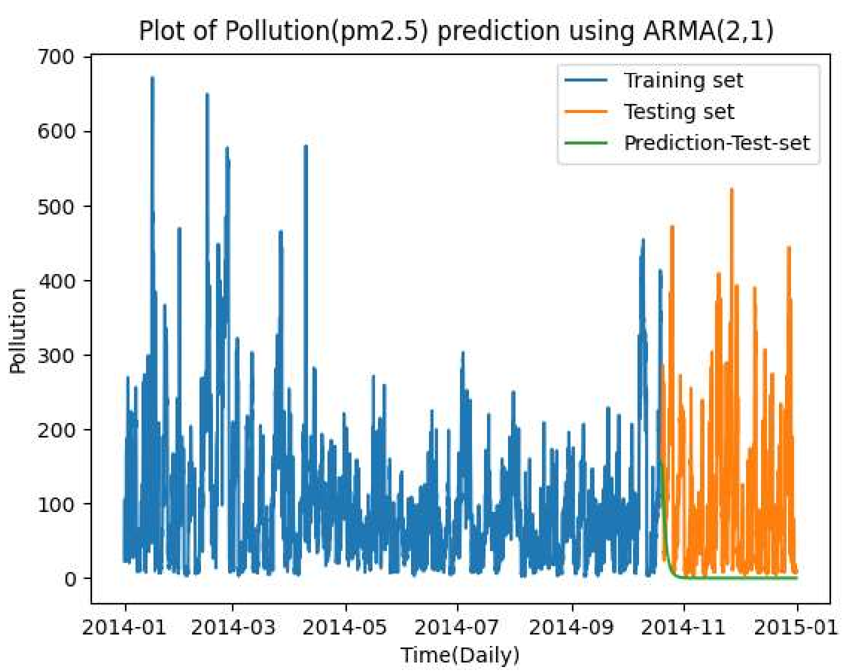


Fig32. Representation of Pollution Prediction for ARMA (2, 1)

Now we will perform the One step ahead prediction and determine the Q-value

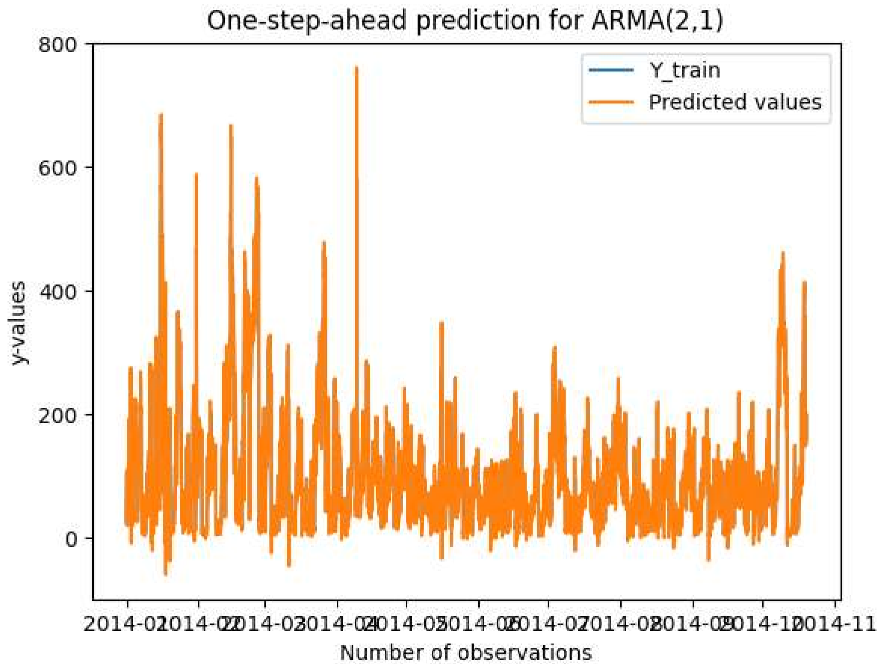


Fig33. Representation of One Step Ahead Prediction for ARMA (2, 1)

```
The Q value is   7125.963528507974
The chi-critical is 7283.29319928651

 The Q value is less than 7283.29319928651 (chi-critical) so, the residual is white
[ -8.69797996 -10.21125365  -3.68912791  -1.00281435  -2.39022654]
```

The Q-Value of the model ARMA (2,1) is lesser than chi-critical and also the residual is white hence the model can be said as the good estimator.

Now we will see the mean and variance of the residuals.

```
MSE of training data for ARMA(2,1) is 71.09467485370998
Mean of residual(training data) with ARMA(2,1) is 1.1969336980185743
The variance of residual(training data) with ARMA(2,1) is 69.66202457625756

MSE of forecast for ARMA(2,1) is 20703.52226497346
The mean of testing set error is 97.12587236779149
The variance of testing set error is 11270.08718176896

The ratio of variance of residual to variance of forecast is 0.006181143362311006
```

As we have seen that the mean of the residual is close to 1 hence it cant be said as a significantly good model. Now we will try to compare the models to find the best fit amongst all.

FINAL MODEL SELECTION

Metrics	Holts Winter	Regression	Average	Naïve	Drift	SES	ARMA (2,0)	ARMA (2,1)
MSE-Residual	15756.0254	7327.098	8073.581	503.521	504.3	12193.06	9.48	71.09
Q-value	77028.14	72486.532	75297.67	7154.85	7153.199	76798.48	13462.08	7152.96
Mean Residual	87.76	0.048	-15.2655	0.019	-0.0789	-64.256	1.59	1.19
Var Residual	8052.86	7237.09	7840.54	503.52	504.3	8064.194	6.93727	69.66
MSE test	19439.81	9916.488	10988.73	14525.65	17168.86	14535.86	20643.461	20703.522
Mean Test	92.31	-12.503	4.526	-59.64	-76.781	-59.729	96.5269	97.1258
Var test	10918.5	9760.152	10968.23	10968.23	11273.49	10968.2	11326.017	11270.087
Ratio (var of resid/var of test)	0.73	0.74	0.71	0.04	0.044	0.73	0.006	0.006

Fig34. Representation of Final Model selection metrics

As we precisely focus on Q-value of all the models we will get a clearer understanding that for the models whose q value is reasonably greater than the chi-critical which proves that models do has the residuals which are not white and hence the ACF plot for those respective models did not converge over the lag period. On this discussion we will see now that amongst 8 models we have got three models named Naïve, Drift and ARMA (2,1) whose Q value is in the same scale of range and whose residuals are white.

Hence our further step of analysis is to focus on the MSE of the training data and mean of the residuals to understand which model we will go ahead with.

Metrics	Naïve	Drift	ARMA (2,1)
MSE-Residual	503.521	504.3	71.09
Q-value	7154.85	7153.199	7152.96
Mean Residual	0.019	-0.0789	1.19
Var Residual	503.52	504.3	69.66
MSE test	14525.65	17168.86	20703.522
Mean Test	-59.64	-76.781	97.1258
Var test	10968.23	11273.49	11270.087
Ratio (var of resid/var of test)	0.04	0.044	0.006

Fig35. Representation of Final Model Selection Metrics

So, as we have discarded all the previous models and have proceeded further with Naïve, Drift and ARMA (2,1) so let's have a look what does the results say as we can see that mean of residual of drift and Naïve is very close by to 0 whereas for ARMA it is approximately 1 whereas on the same hand the mean of the residuals of Naïve and Drift is similar and ARMA is reasonably less so hence we will take the mean of the residual metric into the consideration and proceed further ahead with drift model since the mean of residual is also quite closer to 0 so does the q -value is lesser in comparison to Naïve and Drift hence we will be performing the forecast function using Drift Method.

FORECAST FUNCTION

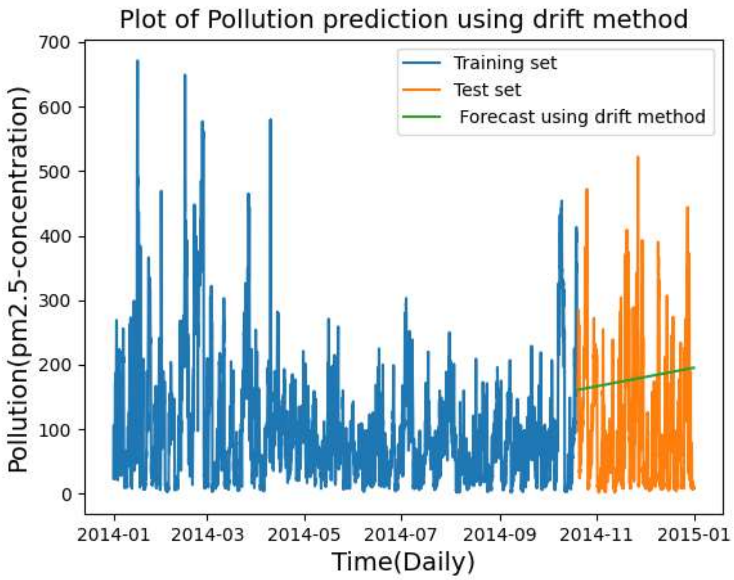


Fig36. Representation of Drift Method Forecast Function

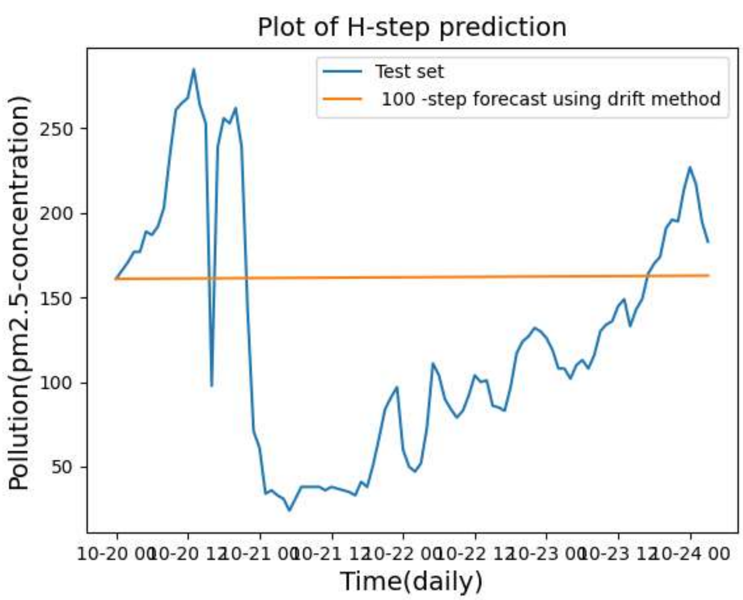


Fig37. Representation of Drift Method H-step Prediction

CONCLUSION

In conclusion, this paper is a detail report for the project of explanation of time series analysis of Air Quality Prediction. This Final Project report includes calculation, explanation and comparison of Holt-winter method, Multiple Regression method, Base models (Average method, Drift method, Naïve method, SES method), ARMA method and ARIMA method. From all the comparison results, Drift was the best model according to the model was trained. However, the performance for testing set was not like the model predicted for training set. The benchmark model out trained the other models. Even though Drift was best fit for the training data, it did not predict precisely for testing set. Based on the results, drift method is also not too bad. However, from this project analysis we should keep in mind that in real world heavy Neural Network models like LSTM will also be used for the implementation to give a better prediction in comparison to the drift method.

REFERENCES

https://archive.ics.uci.edu/ml/datasets/Beijing+PM2.5+Data

https://www.statsmodels.org/0.8.0/generated/statsmodels.tsa.arima_model.ARMA.html

https://pawarbi.github.io/blog/forecasting/r/python/rpy2/altair/fbprophet/ensemble_forecast/uncertainty/simulation/2020/04/21/timeseries-part2.html

https://medium.com/analytics-vidhya/time-series-forecasting-a-complete-guide-d963142da33f

https://otexts.com/fpp2/holt-winters.html

APPENDIX

Project code

```
import numpy as np
import pandas as pd
import datetime
import matplotlib.pyplot as plt
from Toolbox import *
from sklearn.model_selection import train_test_split
import statsmodels.tsa.holtwinters as ets
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from scipy.stats import chi2
from scipy import signal
import keras
from keras.preprocessing.sequence import TimeseriesGenerator
from keras.models import Sequential
from keras.layers import Dense, LSTM
from sklearn.metrics import mean_squared_error

#6- Description of the dataset. Describe the independent variable(s) and dependent variable:
#a. Pre-processing dataset: Dataset cleaning for missing observation. You must follow the
#data cleaning techniques for time series dataset.
#b. Plot of the dependent variable versus time.
#c. ACF/PACF of the dependent variable.
#d. Correlation Matrix with seaborn heatmap with the Pearson’s correlation coefficient.
#e. Split the dataset into train set (80%) and test set (20%).

#Pre-processing dataset: Dataset cleaning for missing observation. You must follow the data
cleaning techniques for time series dataset."
# Loading and reading the dataset into a dataframe:
df = pd.read_csv('AirQualityBeijing.csv')

#printing first ten rows of the dataset
print(df.head(10))

#printing the info of the dataset
print(df.info())

##since the data is having missing values over the period starting from 2010 - 2013 hence we
will focusing only for the year 2014.
data = df[35064:43824]

#printing the first 5 rows of the data to see.
print(data.head())

#checking the dependant column pm2.5 for null values.
null_pm = data['pm2.5'].isna().sum()
print(null_pm)
```

```
#We will be imputing the missing values by using mean of the dependant column.
mean_pm = np.mean(data['pm2.5'])
print(mean_pm)
data['pm2.5']= data['pm2.5'].fillna(mean_pm)
print(data['pm2.5'])
#Reforming the time column in the specific format yyyy:mm:dd:h:m:s
data['Time'] = data.apply(lambda x : datetime.datetime(year=x['year'], month=x['month'],
day=x['day'], hour=x['hour']), axis=1)
data.drop(columns=['year', 'month', 'day', 'hour', 'No'], inplace=True)
data.time = pd.to_datetime(data.Time)
data = data.set_index('Time')
data.columns = ['pollution', 'dew', 'temp', 'press', 'wnd_dir', 'wnd_spd', 'snow', 'rain']
print(data.head())

#Validating the unique values of the column cbwd
unique_cbwd = data['wnd_dir'].unique()
print(unique_cbwd)
#Dropping the column cbwd:
del data['wnd_dir']
print(data.head())

#b. Plot of the dependent variable versus time.
data['pollution'].plot()
plt.xlabel('Time(Daily)')
plt.ylabel('Pollution Concentration')
plt.title('Plot Graph for Pollution(pm2.5) vs TimeStep')
plt.grid()
plt.legend('pollution')
plt.figure(figsize=(20,20))
plt.show()

#c. ACF/PACF of the dependent variable.
dependant_variable = data['pollution']
mean_dep = np.mean(dependant_variable)
var = dependant_variable - mean_dep
ACF_PACF_Plot(dependant_variable,50)
ACF_PACF_Plot(dependant_variable,400)

#d. Correlation Matrix with seaborn heatmap with the Pearson’s correlation coefficient.
#Plotting heat map:
corr_data = data.corr()
sns.heatmap(corr_data , annot=True, cmap="Blues")
plt.title("HeatMap for Air Quality Dataset")
plt.show()


#e. Split the dataset into train set (80%) and test set (20%).
data_train , data_test = train_test_split(data , shuffle=False , test_size=0.2)
X = data[['temp', 'press','wnd_spd', 'snow',  'rain']]
Y = data['pollution']
```

```
#X_svd = sm.add_constant(X)
X_train , X_test , y_train , y_test = train_test_split(X, Y , shuffle=False, test_size=0.2)


#7- Stationarity: Check for a need to make the dependent variable stationary. If the dependent
#variable is not stationary, you need to use the techniques discussed in class to make it
stationary.
#Perform ACF/PACF analysis for stationarity. You need to perform ADF-test & kpss-test
and plot
#the rolling mean and variance for the raw data and the transformed data.

#Plotting the Rolling Mean and Variance for pm2.5:
rollingmean_pm = data['pollution']
cal_rolling_mean_var(rollingmean_pm , data.index)

##performing the ADF-cal and KPSS test to check the stationarity of the dependant variable.
ADF_Cal(dependant_variable)
kpss_test(dependant_variable)

#Observation: Since the data is stationary hence we will proceed further with the steps.
#using differencing technique

log_transformed_data = np.log(data['pollution'])
data_diff1 = log_transformed_data.diff()[1:]
cal_rolling_mean_var(data_diff1 , range(len(data_diff1)))

#8- Time series Decomposition: Approximate the trend and the seasonality and plot the
detrended
#and the seasonally adjusted data set. Find the out the strength of the trend and seasonality.
Refer
#to the lecture notes for different type of time series decomposition techniques.

from statsmodels.tsa.seasonal import STL
import matplotlib.pyplot as plt
temp = data['pollution']
#df = pd.Series(np.array(data['pollution']), index = pd.date_range(start = '2014-01-01',
periods =len(data['pollution']), freq='b'), name= 'Pollution Concentration plot')
STL = STL(temp)
res = STL.fit()
res.plot()
plt.show()

#Calculating the Trend , resid and seasonal by plotting a graph.
T = res.trend
R = res.resid
S = res.seasonal
plt.plot(T , label = 'Trend')
plt.plot(R , label = 'Resid')
plt.plot(S , label = 'Seasonal')
plt.title("Trend-Seasonality-Residuals for Pollution(pm2.5)")
```

```
plt.xlabel('Time(Daily)')
plt.ylabel('Pollution')
plt.legend()
plt.tight_layout()
plt.show()

#calculating the strength of the trend of data:
import numpy as np
v=1 - (np.var(R)/np.var(T + R))
strength_trend = np.max([0,v])
print("The strength of trend for this data set is :", strength_trend)

#calculating the strength of seasonality of the data:
v2=1 - (np.var(R)/np.var(S + R))
strength_seasonal = np.max([0,v2])
print("The strength of seasonality for this data set is :",strength_seasonal)

#Observation: Since the strength of trend is 0.88 which is close to 1 hence we can say that the
data is trended and so will detrend the data and plot the graph against time to see the
observations made.
#calculating the detrended data and plot is vs original data.
detrended= data['pollution'] - T
print("The strength of seasonality for this data set is :",detrended)
fig = T.plot(label = 'Detrended Data')
fig = temp.plot(label = 'Original Data')
plt.title("Plot for Detrended data vs Original data")
fig.legend()
plt.ylabel('Pollution')
plt.show()

#calculating the seasonally adjusted data plot  vs original data.
seasonality = data['pollution'] - S
print("The strength of seasonality for this data set is :",seasonality)
fig = S.plot(label = 'Seasonal Data')
fig = temp.plot(label = 'Original Data')
plt.title("Plot for Seasonally Adjusted data vs original data")
plt.ylabel('Pollution')
fig.legend()
plt.show()


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

train_HLWM =
ets.ExponentialSmoothing(data['pollution'],trend='mul',damped_trend=True,seasonal='mul').f
it()
HLWM_prediction_train = train_HLWM.forecast(steps=len(data_train['pollution']))
test_HLWM = train_HLWM.forecast(steps=len(data_test['pollution']))
test_predict_HLWM = pd.DataFrame(test_HLWM).set_index(data_test['pollution'].index)
resid_HLWM = np.subtract(y_train.values,np.array(HLWM_prediction_train))
```

```
forecast_error_HLWM = np.subtract(y_test.values,np.array(test_HLWM))
MSE_HLWM = np.square(resid_HLWM).mean()
print("Mean square error for (training set) HLWM is ", MSE_HLWM)
acf_resid = auto_correlation(resid_HLWM)
#plotting the acf plot for holts winter:
var1 =np.arange(0,20)
m=1.96/np.sqrt(len(data.pollution))
plt.stem(var1,acf_resid,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.stem(-1*var1,acf_resid,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.xlabel('Lags')
plt.ylabel('ACF')
plt.title('ACF Plot of Holts Winter Method')
plt.axhspan(-m,m,alpha = .1, color = 'green')
plt.tight_layout()
plt.show()
Q  = (len(resid_HLWM)) *np.sum(np.square(acf_resid))
print("\nThe Q value of residual using HWM is  ",Q)
print(f"The Mean of residual of HLWM is {np.mean(resid_HLWM)}")
print(f"The variance of residual of HLWM is {np.var(resid_HLWM)}")
MSE =
np.square(np.subtract(data_test['pollution'].values,np.ndarray.flatten(test_HLWM.values))).m
ean()
print("Mean square error for Holt Winter method is of testing set is ", MSE)
print(f"The Mean of forecast of HLWM is {np.mean(forecast_error_HLWM)}")
print(f"The variance of forecast of HLWM is {np.var(forecast_error_HLWM)}")
print(f"\n The ratio of resid vs forecast is {(np.var(resid_HLWM)) / (
np.var(forecast_error_HLWM)  )}")
plt.plot(data_train['pollution'],label= "AirQuality-train")
plt.plot(data_test['pollution'],label= "AirQuality-test")
plt.plot(test_predict_HLWM,label= "Holt-Winter Method-test")
plt.legend(loc='upper left')
plt.title('Holt-Winter Method for Pollution(pm2.5) Prediction')
plt.xlabel('Time(Daily)')
plt.ylabel('Pollution')
plt.tight_layout()
plt.show()



#Feature Selection:
from numpy import linalg as LA
X_m = X_train.values
y_m = y_train.values
x_svd = sm.add_constant(X_m)
H_vector = np.matmul(x_svd.T, x_svd)
s, d, v = np.linalg.svd(H_vector)
print(f"SingularValues = {d}")
print(f"The condition number constant (original data) = {LA.cond(x_svd)}")

#Feature Selection - OLS:
```

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

#We will be dropping the feature Is:
X_train.drop('snow',axis = 1, inplace=True)
model_1 = sm.OLS(y_train , X_train).fit()
X_test.drop('snow', axis = 1 ,inplace=True)
predictions_1 = model_1.predict(X_test)
print(model_1.summary())

#Will be dropping the constant to check the model summary:
X_train.drop('const',axis = 1, inplace=True)
model_final = sm.OLS(y_train , X_train).fit()
X_test.drop('const', axis=1 , inplace=True)
predictions_2 = model_final.predict(X_test)
print(model_final.summary())


prediction_test = predictions_2
# predictions_2 is the final prediction for x test based on my multiple regression model
#so I will name it as prediction_test

#Multiple Linear Regression:
#performing the f-test on the model obtained after backward-stepwise regression.
f_test = model_final.fvalue
print('\nF-statistic: ', f_test)
print("Probability of observing value at least as high as F-statistic ",model_final.f_pvalue)

#Performing the t-test on the model obtained after backward-stepwise regression.
t_Test =model_final.pvalues
print("\nT-test P values : ", t_Test)

#now I need to predict based on train set so.
model_train = sm.OLS(y_train, X_train).fit()
prediction_train =model_train.predict(X_train)
training_residual = np.subtract(y_train,prediction_train)

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

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

```
    Q_value = len(x) * sum(Q_calc_autocorr)
    return Q_value

#calling auto-corelation function
np_acf_calc_residuals = auto_correlation(training_residual)
var1 =np.arange(0,20)
m=1.96/np.sqrt(len(data.pollution))
plt.stem(var1,np_acf_calc_residuals,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.stem(-1*var1,np_acf_calc_residuals,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.xlabel('Lags')
plt.ylabel('ACF')
plt.title('ACF Plot of Multiple Linear Regression Model')
plt.axhspan(-m,m,alpha = .1, color = 'green')
plt.tight_layout()
plt.show()
#calling function to calculate Q values
Q  = (len(training_residual)) *(np.sum(np.square(np_acf_calc_residuals)))
print(f"The Q value of residual of regression is {Q}")
print(f"The mean of residuals is {np.mean(training_residual)}")
print(f"The variance of residual is {np.var(training_residual)}")
testing_error_regression= np.subtract(y_test, prediction_test)
MSE = np.square(testing_error_regression).mean()
print("\nMean square error for testing set multiple regression is ", MSE)
print(f"RMSE for testing set using multiple regression is :{np.sqrt(MSE)} ")
print(f"The mean of forecast of multiple regression is {np.mean(testing_error_regression)}")
print(f"The variance of foreacast of multiple regression is
{np.var(testing_error_regression)}")
print(f"\n The ratio of resid vs forecast is
{np.var(training_residual)/np.var(testing_error_regression)}")

plt.plot(y_test, label = 'Test set')
plt.plot(prediction_test, label = ' One-step prediction using multiple regression method')
plt.xlabel("Time(Daily)")
plt.ylabel("Pollution")
plt.title("Plot of Pollution(pm2.5) prediction using Regression Method")
plt.legend()
plt.show()


#Base Models:

#Average Method:

y_predict_train_set = []
value = 0
for i in range(len(y_train)):
    if i != 0:
        value = value + y_train[i - 1]
        t_value  = i
        y_each_predict = value / i
```

```
        y_predict_train_set.append(y_each_predict)
    else:
        continue

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

y_preidction_average= pd.DataFrame(y_predict_test_set).set_index(y_test.index)
plt.plot(y_train, label = 'Training set')
plt.plot(y_test, label = 'Test set')
plt.plot(y_preidction_average, label = 'forecast using average method')
plt.xlabel("Time(Daily)")
plt.ylabel("Pollution")
plt.title("Plot of Pollution prediction(pm2.5) using Average Method")
plt.legend()
plt.show()
#now lets find out the MSE of our prediction error --on training set using average method
error_train_set_avg = np.subtract(y_train[1:], y_predict_train_set)
#print(error_train_set_avg)
def calc_MSE(x):
    MSE = np.square(np.array(x)).mean()
    return MSE
MSE_train_set =calc_MSE(error_train_set_avg)
#MSE_train_set = np.sum((var_square_train_set_array) ** 2 )/ len(t_train_set)
print(f"\nMSE of prediction error (training set) using average method is : {MSE_train_set}")
#calling auto-corelation function
np_acf_calc_residuals_average = auto_correlation(error_train_set_avg)
var1 =np.arange(0,20)
m=1.96/np.sqrt(len(data.pollution))
plt.stem(var1,np_acf_calc_residuals_average,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.stem(-1*var1,np_acf_calc_residuals_average,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.xlabel('Lags')
plt.ylabel('ACF')
plt.title('ACF Plot of Average Method')
plt.axhspan(-m,m,alpha = .1, color = 'green')
plt.tight_layout()
plt.show()
#calling function to calculate Q values
Q_residual  = (len(error_train_set_avg))
*(np.sum(np.square(np_acf_calc_residuals_average)))
print(f"The Q value of residual using average method is {Q_residual}")
print(f"The mean of residuals using average method is {np.mean(error_train_set_avg)}")
print(f"The variaince of residual using average method is {np.var(error_train_set_avg)}")
error_test_set_avg = np.subtract(y_test, y_predict_test_set)
MSE_train_set =calc_MSE(error_test_set_avg)
print(f"\nMSE of forecast (testing set) using average method is : {MSE_train_set}")
print(f"Mean of forecast error is: {np.mean(np.array(error_test_set_avg))}")
print(f"Variance of forecast error is: {np.var(np.array(error_test_set_avg))}")
```

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

#Naive method:

print("****  Naive Method   ****")
y_predict_train_set_naive = []
value = 0
for i in range(len(y_train[1:])):
    y_predict_train_set_naive.append(y_train[i])
#print(y_predict_train_set_naive)
y_predict_test_set_naive= [y_train[-1] for i in y_test]
y_prediction_naive_test= pd.DataFrame(y_predict_test_set_naive).set_index(y_test.index)
#print(y_predict_test_set_naive)
plt.plot(y_train, label = 'Training set')
plt.plot(y_test, label = 'Test set')
plt.plot(y_prediction_naive_test, label = 'Forecast using naive method')
plt.xlabel("Time(Daily)")
plt.ylabel("Pollution")
plt.title("Plot of Pollution prediction(pm2.5) using Naive method")
plt.legend()
plt.show()
error_train_set_naive = np.subtract(y_train[1:], y_predict_train_set_naive)
error_test_set_naive = np.subtract(y_test, y_predict_test_set_naive)
MSE_train_set_naive =calc_MSE(error_train_set_naive)
print(f"\nMSE of prediction error (training set) using naive method is :
{MSE_train_set_naive}")
#calling auto-corelation function
np_acf_calc_residuals_naive = auto_correlation(error_train_set_naive)
var1 =np.arange(0,20)
m=1.96/np.sqrt(len(data.pollution))
plt.stem(var1,np_acf_calc_residuals_naive,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.stem(-1*var1,np_acf_calc_residuals_naive,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.xlabel('Lags')
plt.ylabel('ACF')
plt.title('ACF Plot of Naive Method')
plt.axhspan(-m,m,alpha = .1, color = 'green')
plt.tight_layout()
plt.show()

#calling function to calculate Q values
Q_residual  = (len(error_train_set_naive))
*(np.sum(np.square(np_acf_calc_residuals_naive)))
print(f"The Q value of residual using naive method is  {Q_residual}")
print(f"The mean of residuals using naive method is  {np.mean(error_train_set_naive)}")
print(f"The variance of residual using naive method is  {np.var(error_train_set_naive)}")
MSE_test_set_naive =calc_MSE(error_test_set_naive)
print(f"\nMSE of prediction error (testing set) using naive method is :
{MSE_test_set_naive}")
print(f"Mean of forecast error using naive method is:
```

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

#Drift method

print("***** Drift Method   ********")
y_predict_train_set_drift = []
value = 0
for i in range(len(y_train)):
    if i > 1:
        slope_val = (y_train[i - 1] - y_train[0]) / (i-1)
        y_each_predict = (slope_val * i) + y_train[0]
        y_predict_train_set_drift.append(y_each_predict)
    else:
        continue
y_predict_test_set_drift= []
for h in range(len(y_test)):
    slope_val = (y_train[-1] - y_train[0] ) /( len(y_train) - 1 )
    y_predict_each = y_train[-1] + ((h +1) * slope_val)
    y_predict_test_set_drift.append(y_predict_each)

y_preidction_drift= pd.DataFrame(y_predict_test_set_drift).set_index(y_test.index)
plt.plot(y_train, label = 'Training set')
plt.plot(y_test, label = 'Test set')
plt.plot(y_preidction_drift, label = 'forecast using drift method')
plt.xlabel("Time(Daily)")
plt.ylabel("Pollution")
plt.title("Plot of Pollution prediction(pm2.5) using Drift method")
plt.legend()
plt.show()
error_train_set_drift = np.subtract(y_train[2:], y_predict_train_set_drift)
error_test_set_drift = np.subtract(y_test, y_predict_test_set_drift)
MSE_train_set_drift =calc_MSE(error_train_set_drift)
print(f"\nMSE of prediction error (training set) using drift method is :
{MSE_train_set_drift}")
#calling auto-corelation function
np_acf_calc_residuals_drift = auto_correlation(error_train_set_drift)
var1 =np.arange(0,20)
m=1.96/np.sqrt(len(data.pollution))
plt.stem(var1,np_acf_calc_residuals_drift,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.stem(-1*var1,np_acf_calc_residuals_drift,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.xlabel('Lags')
plt.ylabel('ACF')
plt.title('ACF Plot of Drift Method')
plt.axhspan(-m,m,alpha = .1, color = 'green')
plt.tight_layout()
plt.show()
```

```
#calling function to calculate Q values
Q_residual  = (len(error_train_set_drift)) *(np.sum(np.square(np_acf_calc_residuals_drift)))
print(f"The Q value of residual using drift method is {Q_residual}")
print(f"The mean of residuals using drift method is {np.mean(error_train_set_drift)}")
print(f"The variance of residual using drift method is {np.var(error_train_set_drift)}")
MSE_test_set_drift =calc_MSE(error_test_set_drift)
print(f"\nMSE of prediction error of testing set using drift method is : {MSE_test_set_drift}")
print(f"Mean of forecast error using drift method is:
{np.mean(np.array(error_test_set_drift))}")
print(f"Variance of forecast error using drift method is:
{np.var(np.array(error_test_set_drift))}")
print(f"\n The ratio of resid vs forecast of drift method is {( np.var(error_train_set_drift) ) / (
np.var(np.array(error_test_set_drift)) )}")

#Simple and exponential smoothing
SES = ets.ExponentialSmoothing(y_train,trend=None,damped=False,seasonal=None).fit()
SES_predict_train= SES.forecast(steps=len(y_train))
SES_predict_test= SES.forecast(steps=len(y_test))
predict_test_SES = pd.DataFrame(SES_predict_test).set_index(y_test.index)
resid_SES = np.subtract(y_train.values,np.array(SES_predict_train))
forecast_error_Ses = np.subtract(y_test.values,np.array(SES_predict_test))
MSE_SES = np.square(resid_SES).mean()
print("Mean square error for (training set) simple exponential smoothing is ", MSE_SES)
plt.plot(y_train,label= "Air Qaulity-train")
plt.plot(y_test,label= "Air Quality-test")
plt.plot(predict_test_SES,label= "SES Method prediction")
plt.legend(loc='upper left')
plt.title('SES method for Pollution(pm2.5) prediction')
plt.xlabel('Time(Daily)')
plt.ylabel('Pollution')
plt.show()
#calling auto-corelation function
np_acf_calc_residuals_SES = auto_correlation(resid_SES)
var1 =np.arange(0,20)
m=1.96/np.sqrt(len(data.pollution))
plt.stem(var1,np_acf_calc_residuals_SES,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.stem(-1*var1,np_acf_calc_residuals_SES,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.xlabel('Lags')
plt.ylabel('ACF')
plt.title('ACF Plot of SES Method')
plt.axhspan(-m,m,alpha = .1, color = 'green')
plt.tight_layout()
plt.show()
#calling function to calculate Q values
Q_residual  = (len(resid_SES)) *(np.sum(np.square(np_acf_calc_residuals_SES)))
print(f"The Q value of residual using SES method is {Q_residual}")
print(f"Mean of residual using SES method is: {np.mean(np.array(resid_SES))}")
print(f"Variance of residual using SES method is: {np.var(np.array(resid_SES))}")
MSE_SES = np.square(forecast_error_Ses).mean()
print("Mean square error for (testing set) simple exponential smoothing is ", MSE_SES)
```

```
print(f"Mean of forecast error using SES method is:
{np.mean(np.array(forecast_error_Ses))}")
print(f"Variance of forecast error using SES method is:
{np.var(np.array(forecast_error_Ses))}")
print(f"\n The ratio of resid vs forecast of SES method is {( np.var(np.array(resid_SES)) ) / (
np.var(np.array(forecast_error_Ses)) )}")


#ARMA and ARIMA and SARIMA model:
#a. Preliminary model development procedures and results. (ARMA model order
#determination). Pick at least two orders using GPAC table.
#b. Should include discussion of the autocorrelation function and the GPAC. Include a plot of
#the autocorrelation function and the GPAC table within this section).
#c. Include the GPAC table in your report and highlight the estimated order.
ry = auto_correlation(data_diff1)
ry1 = ry[::-1]
ry2 = np.concatenate((np.reshape(ry1,20),ry[1:]))
na_order =8
nb_order =8
calc_Gpac(na_order, nb_order, ry2)

#Since we dont see any patterns after feeding the differenced variable to the auto-correlation
which proves that we dont have ARIMA model so we will be feeding the dependant variable
to check the gpac pattern.
y = data['pollution']
ry = auto_correlation(y)
ry1 = ry[::-1]
ry2 = np.concatenate((np.reshape(ry1,20),ry[1:]))
na_order =8
nb_order =8
calc_Gpac(na_order, nb_order, ry2)

#we will perform ARMA(2,0) looking at the pattern obtained from the gpac.

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

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

theta, cov,SSE_count = calc_LMA(count, dependant_variable, na,nb, theta, delta, u, u_max)
Y_train , Y_test = train_test_split(y, test_size= 0.2, shuffle =False)
y_predict=[]
for i in range(len(Y_train)):
    if i ==0:
```

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

resid = np.subtract(np.array(Y_train),np.array(y_predict))
acf_resid_arma = auto_correlation(resid)
var1 =np.arange(0,20)
m=1.96/np.sqrt(len(data.pollution))
plt.stem(var1,acf_resid_arma,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.stem(-1*var1,acf_resid_arma,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.xlabel('Lags')
plt.ylabel('ACF')
plt.title('ACF Plot of ARMA(2,0)')
plt.axhspan(-m,m,alpha = .1, color = 'green')
plt.tight_layout()
plt.show()
Q  = (len(resid)) *(np.sum(np.square(acf_resid_arma)))
DOF = len(resid) - na -nb
alfa =0.01
chi_critical = chi2.ppf(1-alfa, DOF)
print("\nThe Q value is  ",Q)
print(f"The chi-critical is {chi_critical}")

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

print(resid[:5])
print(f"\nMSE of training data for ARMA(2,0) is {np.square(resid).mean()}")
print(f"Mean of residual(training data) with ARMA(2,0) is {np.mean(resid)}")
print(f"The variance of residual(training data) with ARMA(2,0) is {np.var(resid)}")
y_predict = pd.DataFrame(y_predict).set_index(Y_train.index)
plt.plot(Y_train, label='Y_train')
plt.plot(y_predict, label ='Predicted values')
plt.xlabel('Number of observations')
plt.ylabel('y-values')
plt.title("One-step-ahead prediction for ARMA(2,0)")
plt.legend()
plt.show()

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

```
    elif i ==2:
        predict = ( -theta[0] * y_prediction_test[1] ) + (-theta[1] * y_prediction_test[0])
    else:
        predict = (-theta[0] * y_prediction_test[i - 1]) + (-theta[1] * y_prediction_test[i-2])
    y_prediction_test.append(predict)
y_prediction_test = pd.DataFrame(y_prediction_test).set_index(Y_test.index)
forecast_error = np.subtract(np.array(Y_test), np.array(y_prediction_test))
print(f"\nMSE of forecast for ARMA(2,0) is {np.square(forecast_error).mean()}")
print(f"The mean of testing data is {np.mean(forecast_error)}")
print(f"The variance of testing data is {np.var(forecast_error)}")
ratio = np.var(resid)/np.var(forecast_error)
print(f"\nThe ratio of variance of residual to variance of forecast is {ratio}")
plt.plot(Y_train, label='Training set')
plt.plot(Y_test, label = 'Testing set')
plt.plot(y_prediction_test, label ='Prediction Test set')
plt.xlabel("Time(Daily)")
plt.ylabel("Pollution")
plt.title("Plot of Pollution(pm2.5) prediction using ARMA(2,0) ")
plt.legend()
plt.show()

#Second pattern of gpac(2,1):

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

print("\nFor our estimated ARMA (2,1): ")
theta, cov,SSE_count = calc_LMA(count, dependant_variable, na,nb, theta, delta, u, u_max)
Y_train , Y_test = train_test_split(y, test_size= 0.2, shuffle =False)
y_predict=[]
for i in range(len(Y_train)):
    if i ==0:
        predict = (-theta[0]) * Y_train[i]
    else:
        predict = ( - theta[0] * Y_train[i] ) + ( -theta[1] * Y_train[i-1])
    y_predict.append(predict)

resid = np.subtract(np.array(Y_train),np.array(y_predict))
acf_resid_ar = auto_correlation(resid)
var1 =np.arange(0,20)
m=1.96/np.sqrt(len(data.pollution))
plt.stem(var1,acf_resid_ar,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.stem(-1*var1,acf_resid_ar,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.xlabel('Lags')
```

```
plt.ylabel('ACF')
plt.title('ACF Plot of ARMA(2,1)')
plt.axhspan(-m,m,alpha = .1, color = 'green')
plt.tight_layout()
plt.show()
Q  = (len(resid)) *(np.sum(np.square(acf_resid_ar)))
DOF = len(resid) - na -nb
alfa =0.01
chi_critical = chi2.ppf(1-alfa, DOF)
print("\nThe Q value is  ",Q)
print(f"The chi-critical is {chi_critical}")

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

print(resid[:5])
print(f"\nMSE of training data for ARMA(2,1) is {np.square(resid).mean()}")
print(f"Mean of residual(training data) with ARMA(2,1) is {np.mean(resid)}")
print(f"The variance of residual(training data) with ARMA(2,1) is {np.var(resid)}")
y_predict = pd.DataFrame(y_predict).set_index(Y_train.index)
plt.plot(Y_train, label='Y_train')
plt.plot(y_predict, label ='Predicted values')
plt.xlabel('Number of observations')
plt.ylabel('y-values')
plt.title("One-step-ahead prediction for ARMA(2,1)")
plt.legend()
plt.show()

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

y_prediction_test = pd.DataFrame(y_prediction_test).set_index(Y_test.index)
forecast_error = np.subtract(np.array(Y_test), np.array(y_prediction_test))
print(f"\nMSE of forecast for ARMA(2,1) is {np.square(forecast_error).mean()}")
print(f"The mean of testing set error is {np.mean(forecast_error)}")
print(f"The variance of testing set error is {np.var(forecast_error)}")
ratio = np.var(resid)/np.var(forecast_error)
```

```
print(f"\nThe ratio of variance of residual to variance of forecast is {ratio}")
plt.plot(Y_train, label='Training set')
plt.plot(Y_test, label = 'Testing set')
plt.plot(y_prediction_test, label ='Prediction-Test-set')
plt.xlabel("Time(Daily)")
plt.ylabel("Pollution")
plt.title("Plot of Pollution(pm2.5) prediction using ARMA(2,1) ")
plt.legend()
plt.show()

#Forecast function:

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

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

    return y_predict_test_set_drift

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

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


plt.plot(Y_train, label = 'Training set')
plt.plot(Y_test, label = 'Test set')
plt.plot(y_predict_test_set_drift, label = ' Forecast using drift method')
plt.xlabel("Time(Daily)", fontsize= 14)
plt.ylabel("Pollution(pm2.5-concentration)", fontsize =14)
plt.title("Plot of Pollution prediction using drift method", fontsize =14)
plt.legend()
plt.show()

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

step =100
y_predict_h_step_drift = forecast_function(Y_train, step)

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

plt.plot(Y_test[:100], label = 'Test set')
plt.plot(y_predict_h_step_drift, label = ' 100 -step forecast using drift method')
plt.xlabel("Time(daily)", fontsize= 14)
plt.ylabel("Pollution(pm2.5-concentration)", fontsize =14)
```

```
plt.title("Plot of H-step prediction", fontsize =14)
plt.legend()
plt.show()
```

Toolbox:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from pandas import Series
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
from Toolbox import *
import matplotlib.pyplot as plt
from scipy import signal
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
from statsmodels.tsa.stattools import pacf
import seaborn as sns

#Rolling mean and variance
def cal_rolling_mean_var(x,y):
   rmean = [x[0]]
   rvariance =[]
   for k in range(1, len(x)):
      result_mean = np.mean(x[:k])
      rmean.append(result_mean)
   for j in range(1,len(x)+1):
      result_variance = np.var(x[:j])
      rvariance.append(result_variance)
   print(f"The rolling mean of is :", rmean[:10])
   print(f"The rolling variance of is :", rvariance[:10])

##To plot the rolling mean :
   plt.plot(y, rmean , color='Red')
   plt.ylabel(f'Pollution')
   plt.xlabel(f'Time(Daily)')
   plt.title(f"The Rolling mean of pollution(pm2.5) concentration")
   plt.figure()
   plt.show()
##To plot the rolling variance:
   plt.plot(y , rvariance , color='Yellow')
   plt.ylabel(f'Pollution')
   plt.xlabel(f'Time(Daily)')
   plt.title(f"The Rolling variance of pollution(pm2.5) concentration")
   plt.figure()
```

```python
    plt.show()


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


#KPSS test
from statsmodels.tsa.stattools import kpss
def kpss_test(timeseries):
     print(f'Results of KPSS Test for {timeseries}:')
     kpsstest = kpss(timeseries, regression='c', nlags="auto")
     kpss_output = pd.Series(kpsstest[0:3], index=['Test Statistic', 'p-value', 'Lags-Used'])
     for key, value in kpsstest[3].items():
         kpss_output['Critical Value (%s)' % key] = value
     print(kpss_output)

#first order differencing
def first_order_differencing(dataset, interval=1):
    difference = []
    difference[0]=difference.append(np.nan)
    for i in range(interval , len(dataset)):
       value = dataset[i]-dataset[i - interval]
       difference.append(value)
    return Series(difference)


#Second order differencing
def second_order_differencing(dataset ,interval=2):
    difference = []
    difference[0]=difference.append(np.nan)
    difference[1]=difference.append(np.nan)
    for i in range(interval , len(dataset)+1):
        value = dataset[i]-dataset[i - 1]
        difference.append(value)
    return Series(difference)


#Third order differencing
def third_order(dataset , interval=3):
```

```
    difference = []
    difference[0]=difference.append(np.nan)
    difference[1]=difference.append(np.nan)
    difference[2]=difference.append(np.nan)
    for i in range(interval ,len(dataset)+2):
        value = dataset[i]-dataset[i-1]
        difference.append(value)
    return Series(difference)
import math

#Correlation Co-efficient
def correlation_coefficient_cal(data1,data2):
    n = len(data1)
    #for calculating the mean:
    data1_mean = np.mean(data1)
    data2_mean = np.mean(data2)
    numerator = 0
    for i, j in zip(data1,data2):
        numerator+= (i-data1_mean) * (j-data2_mean)
    sum1=0
    sum2=0
    for i in data1:
        sum1+=(i-data1_mean)**2
    sqrt_data1 = np.sqrt(sum1)
    for j in data2:
         sum2+=(j-data2_mean)**2
    sqrt_data2 = np.sqrt(sum2)
    denominator=sqrt_data1*sqrt_data2
    rho = numerator/denominator
    return (rho)


#Auto Correlation
def auto_correlation(y):
    lags = int(input("Enter the lags:"))
    T = len(y)
    numerator = 0
    denominator = 0
    list_acf = []
    y_mean = np.mean(y)
    d_t=0
    for tho in range(d_t, T):
        denominator += (y[tho] - y_mean) ** 2
    print(denominator)
    for i in range(0 , lags):
        for t in range(i , T):
            numerator+= (y[t]-y_mean)*(y[t-i]-y_mean)
```

```
        acf = numerator/denominator
        numerator=0
        list_acf.append(acf)

    print("ACF is :",list_acf)
    return list_acf

#Moving Average
import numpy as np
# define moving average function
def moving_avg(x):
    n = int(input("Enter the n value :"))
    if (n % 2) != 0:
        cumsum = np.cumsum(np.insert(x, 0, 0))
        return (cumsum[n:] - cumsum[:-n]) / float(n)
    elif (n % 2) == 0:
        m= 2
        cumsum = np.cumsum(np.insert(x, 0, 0))
        result_four = (cumsum[n:] - cumsum[:-n]) / float(n)
        cumsum_kfold = np.cumsum(np.insert(result_four, 0, 0))
        result_2ma = (cumsum_kfold[m:] - cumsum_kfold[:-m]) / float(m)
        return result_2ma


##ARMA process :
def ARMA():
    samples = int(input("Enter the samples :"))
    mean_wn = float(input("Enter the mean of white noise :"))
    variance = float(input("Enter the variance:"))
    na = int(input("Enter the ar order:"))
    nb = int(input("Enter the ma order :"))
    an = [0]*na
    bn = [0]*nb
    for i in range(na):
        an[i] = float(input("Enter the co-efficient :"))

    for j in range(nb):
        bn[j] = float(input("Enter the co-efficient :"))

    max_order = max(na , nb)
    num = [0]*(max_order+1)
    den = [0]*(max_order+1)
    for i in range(na+1):
        if i ==0:
            den[i]=1
        else:
```

```
        den[i]=an[i-1]

    arparams = np.array(an)
    print(arparams)
    maparams = np.array(bn)
    print(maparams)
    na=len(arparams)
    nb=len(maparams)
    ar = np.r_[1 , arparams]
    ma = np.r_[1 , maparams]
    arma_process = sm.tsa.ArmaProcess(ar, ma)
    if mean_wn == 0:
        y = arma_process.generate_sample(samples)
    else:
        mean_y = mean_wn * (1 + np.sum(bn)) / (1 + np.sum(an))
        y = arma_process.generate_sample(samples, scale=np.sqrt(variance) + mean_y)
    return y


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

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

                # dinominator matrix and numerator matrix have same values except for the last
column so:
```

```
            dinomator_matrix = np.array(dinom_matrix)
            numerator_matrix = np.array(dinom_matrix)

            # updating values for last column of numerator matrix

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

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

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

    print(df)

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



#ACF-PACF plot
from statsmodels.graphics.tsaplots import plot_acf , plot_pacf
def ACF_PACF_Plot(y,lags):
    acf = sm.tsa.stattools.acf(y, nlags=lags)
    pacf = sm.tsa.stattools.pacf(y, nlags=lags)
    fig = plt.figure()
    plt.subplot(211)
    plt.title('Auto Correlation Plot')
    plot_acf(y, ax=plt.gca(), lags=lags)
    plt.ylabel("Magnitude")
    plt.xlabel("Lags")
    plt.subplot(212)
    plt.title("Partial Auto Correlation Plot")
    plot_pacf(y, ax=plt.gca(), lags=lags)
    plt.ylabel("Magnitude")
    plt.xlabel("Lags")
    fig.tight_layout(pad=3)
    plt.show()


from scipy import signal
```

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

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

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


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

    theta_e =np.array(theta)

    return theta_e

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

    X=[]

    n =na +nb

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

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

```
    return A,g, SSE

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

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

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

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

        n =na+nb

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

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

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

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

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


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

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

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

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


#zero/pole

num_root =[1]
den_root= [1]

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

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

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

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

```
    #plt.show()

    return theta,co_variance, SSE_count

def SARIMA_func():
    T=int(input('Enter number of samples'))
    mean=eval(input('Enter mean of white nosie'))
    var=eval(input('Enter variance of white noise'))
    na = int(input("Enter AR process order"))
    nb = int(input("Enter MA process order"))
    naparam = [0] * na
    nbparam = [0] * nb
    for i in range(0, na):
        naparam[i] = float(input(f"Enter the coefficient of AR:a{i + 1}"))
    for i in range(0, nb):
        nbparam[i] = float(input(f"Enter the coefficient of MA:b{i + 1}"))
    while len(naparam) < len(nbparam):
        naparam.append(0)
    while len(nbparam) < len(naparam):
        nbparam.append(0)
    ar = np.r_[1, naparam]
    ma = np.r_[1, nbparam]
    e=np.random.normal(mean,np.sqrt(var),T)
    system=(ma,ar,1)
    t,process=signal.dlsim(system,e)
    a=[a[0] for a in process]
    return a


def difference(y,interval):
    diff=[]
    for i in range(interval,len(y)):
        value=y[i]-y[i-interval]
        diff.append(value)
    return diff
```