## **DATS6313**

## **TIME SERIES ANALYSIS & MODELING**

**Dr. Reza Jafari**

**Final Term Project**

**Nusrat Nawshin**

**NN**

**03/29/2022**

Table of Content

|  |  |
| --- | --- |
| Contents | Page No. |
| Abstract | 4 |
| Introduction | 4 |
| Description of the dataset | 4 |
| Stationarity | 7 |
| Time series decomposition | 8 |
| Holt-Winter Method | 8 |
| Feature selection | 10 |
| Base models | 12 |
| Average method | 12 |
| Naïve method | 13 |
| Drift method | 13 |
| Simple Exponential Smoothing (SES) method | 14 |
| Multiple Linear Regression | 15 |
| General Partial Autocorrelation (GPAC) | 18 |
| ARMA | 18 |
| ARMA (1,0) | 19 |
| ARMA (3,1) | 20 |
| ARIMA | 21 |
| ARIMA (1,1,0) | 21 |
| ARIMA (3,1,1) | 22 |
| SARIMA | 23 |
| Levenberg Marquardt Algorithm | 23 |
| Diagnostic Analysis | 24 |
| Final Model Selection | 26 |
| Forecast | 26 |
| H-step prediction | 27 |
| Summary and Conclusion | 28 |
| Appendix | 29 |
| Source code of full analysis | 29 |
| Source code of toolbox | 55 |
| Source code of data cleaning | 67 |

**Table of Figures and Tables**

|  |  |
| --- | --- |
| Figure/Table No. | Page No. |
| 1. 1st five records of the dataset | 4 |
| 2. Dataset column data type | 5 |
| 3. Statistics of Numerical Columns Data | 5 |
| 4. Dependent variable avgAQI vs time | 6 |
| 5. ACF/PACF of dependent variable avgAQI | 6 |
| 6. Correlation Matrix of all features | 6 |
| 7. Rolling mean and variance of the dependent variable | 7 |
| 8. ADF test | 7 |
| 9. KPSS test | 7 |
| 10. ACF of average AQI | 8 |
| 11. STL decomposition | 8 |
| 12. Trend, Residuals and Seasonality of 'AvgAQI' | 8 |
| 13. Seasonality adjusted and detrended data | 9 |
| 14. 3-Moving Average of dependent variable | 9 |
| 15. Holt-Winter forecasting | 10 |
| 16. ACF of Holt-Winter Residuals | 10 |
| 17. ACF of Holt-Winter Forecast errors | 10 |
| 18. Singular value of the independent variables | 10 |
| 19. Cumulative explained variance vs number of components | 11 |
| 20. Reduced feature space explained variance ratio | 11 |
| 21. Reduced feature space correlation matrix | 11 |
| 22. Average forecasting | 12 |
| 23. ACF of Average method residuals | 12 |
| 24. Naïve method forecast | 13 |
| 25. ACF of Naïve method residuals | 13 |
| 26. Drift method forecasting | 14 |
| 27. ACF of Drift method residuals | 14 |
| 28. SES forecasting | 14 |
| 29. ACF of SES method residuals | 15 |
| Table 1: Multiple Linear Regression Models | 15 |
| Final Model Summary | 16 |
| 30. Multiple Linear Regression Model | 16 |
| 31. ACF of OLS residuals | 17 |
| 32. ACF of dependent variable Trainset | 18 |
| 33. GPAC table of y train set | 18 |
| 34. ARMA (1,0) model summary | 19 |
| 35.1. ACF of ARMA (1,0) residuals | 19 |
| 35.2. ACF of ARMA (1,0) forecast errors | 19 |
| 35.3 ACF of ARMA (1,0) Covariance Matrix | 19 |
| 36. ARMA (3,1) model summary | 20 |
| 37.1. ARMA (3,1) Residuals | 20 |
| 37.2 ARMA (3,1) forecast error | 20 |
| 37.3 ARMA (3,1) Covariance Matrix | 21 |
| 38. ARIMA (1,1,0) model summary | 21 |
| 39.1 ARIMA (1,1,0) residuals | 21 |
| 39.2 ARIMA (1,1,0) forecast errors | 21 |
| 39.3 ARIMA (1,1,0) covariance matrix | 21 |
| 40. ARIMA (3,1,1) model summary | 22 |
| 41.1 ARIMA (3,1,1) residuals | 22 |
| 41.2 ARIMA (3,1,1) forecast errors | 22 |
| 41.3 ARIMA (3,1,1) covariance matrix | 22 |
| 42. SARIMA (3,0,1) x (0,2,0,7) model summary | 23 |
| 43.1 SARIMA (3,0,1) x (0,2,0,7) residuals | 23 |
| 43.2 SARIMA (3,0,1) x (0,2,0,7) forecast errors | 23 |
| 44. LM algorithm estimated parameters | 24 |
| Table 2: Confidence Intervals | 24 |
| Table 3: Zero Pole Cancellation | 24 |
| Base models comparison: | 26 |
| 45. ACF of residuals of final model | 26 |
| 46. Forecast with final ARIMA (3,1,1) model | 27 |
| 47. 7-step prediction in ARIMA (3,1,1) model | 27 |
| 48. 30-step prediction on ARIMA (3,1,1) model | 28 |

Abstract:

The objective of this project is to apply course learning objectives to a real dataset for time series modelling and prediction.

Introduction:

Air pollution is a growing problem and a major worldwide health concern. Sources of air pollution are wide-ranging with seasonal and daily fluctuations. Troublesome anthropogenic inputs come from fossil fuel consumption for transportation, power supply, space heating and cooling, and certain industrial manufacturing processes, with crop-residue burning making a seasonal contribution in some areas [1]. This study uses air quality data from a compiled dataset for four air pollutants, Ozone (O3), Carbon monoxide (CO), Sulfur dioxide (SO2), and Nitrogen dioxide (NO2) of United States air from the period 2000 to 2021. The full data is collected from [Kaggle](https://www.kaggle.com/datasets/alpacanonymous/us-pollution-20002021) where the daily data is derived from the United States national Air Quality System (AQS) database maintained by the Environmental Protection Agency (EPA, 2021). This study specifically focuses on California Loss Angeles County. The Air Quality Index (AQI) 0-50 indicates good air quality whereas 401-500 indicates severe air quality [2].

Description of the dataset:

The original dataset from Kaggle ‘pollution\_2000\_2021.csv’ includes all the US states' air quality index data. It had 608,699 rows and 24 columns. For this analysis, I am considering only California state’s Loss Angeles County. I have also created my target column ‘avgAQI’ by taking the average of 'O3 AQI', 'CO AQI', 'SO2 AQI', and 'NO2 AQI' columns. The final dataset (figure 1) contains 7,852 rows and 24 columns and does not contain any missing values (figure 2). It has AQI data from 2000-01-01 to 2021-06-30. It has four categorical columns and 20 numeric columns. The first five records of the dataset are given below (figure 1).

Table

Description automatically generated

Figure 1: 1st five records of the dataset

Table

Description automatically generated

Figure 2: Dataset column data type

In figure 3 there is the statistical information of the air pollutant columns.

|  |
| --- |
|  |
| Figure 3: Statistics of Numerical Columns Data |

Here we can see that the mean average AQI is 23.7 which means Loss Angeles overall has a good air quality. But it reached a maximum AQI of 77.75 as well, which is coidered as moderate AQI [2]. From the graph of average AQI over the time (figure 4), it can be noticed that there are frequent ups and downs.

Chart

Description automatically generated

Figure 4: Dependent variable avgAQI vs time

A picture containing histogram

Description automatically generated

Figure 5: ACF/PACF of dependent variable avgAQI

Chart, treemap chart

Description automatically generated

Figure 6: Correlation Matrix of all features

In the ACF/PACF plots (figure 5), we can observe a tail off in the ACF plot and a cut off after 1. The Pearson correlation matrix, it can be noticed that average AQI has strong positive correlation with NO2 1st max value and NO2 mean. It has almost no correlation with NO2, SO2, CO and O3 1st mast hours.

For further analysis and model building the dataset is divided into train and test sets with an 80:20 split with ‘avgAQI’ as the target variable. The train set has 6281 number of rows, and the test set contains 1571 number of rows.

Stationarity:

To check the stationarity of the dependent variable, at first, I am calculating and plotting the rolling mean and variance and then performing ADF test.

A picture containing graphical user interface

Description automatically generated

Figure 7: Rolling mean and variance of the dependent variable

From the rolling mean and variance plots, rolling mean is downward slopping but rolling variance is stabilizes once all samples are included (figure 7).

A picture containing text

Description automatically generated

Figure 8: ADF test

The ADF p-value below a threshold (1% or 5%) suggests that we reject the null hypothesis and conclude that the data is stationary.

Text

Description automatically generated with medium confidence

Figure 9: KPSS test

The KPSS p-value below a threshold (1% or 5%) suggests that we reject the null hypothesis and conclude that the data is nonstationary. As the rolling mean, rolling variance and ADF is indicating the stationarity of the dependent variable, so I am considering the dataset is stationary and no difference is needed.

Chart, histogram

Description automatically generated

Figure 10: ACF of average AQI

The ACF of average AQI is decaying over the lags.

Time Series Decomposition:

STL is a versatile and robust method for decomposing time series. STL is an acronym for “Seasonal and Trend decomposition using Loess,” while Loess is a method for estimating nonlinear relationships [3].

A picture containing diagram

Description automatically generated

Figure 11: STL decomposition

Chart

Description automatically generated

Figure 12: Trend, Residuals and Seasonality of 'AvgAQI'

The strength of trend for this dataset is 0.745 and the strength of seasonality for this dataset is 0.387. Observing the graphs, trend and strength of seasonality values, this dataset has low seasonality (0.387) and is trended (0.745).

A picture containing chart

Description automatically generated

Figure 13: Seasonality adjusted and detrended data

Moving average is one of the classical methods for time series decomposition. Moving averages are averages calculated for consecutive data from overlapping subgroups of fixed length. Moving averages smoothen the time series by filtering out random fluctuations. The period of moving average depends on the type of data. For non-seasonal data, a shorter length, typically a 3 period or a 5-period moving average, is considered [4]. Here I am using 3-MA on this dataset (figure 14).

Chart

Description automatically generated

Figure 14: 3-Moving Average of dependent variable

Holt-Winter Method:

The Holt-Winters modifies the Holt linear technique so that it can be used in the presence of both trend and seasonality. As this dataset does not have seasonality, I am using the holt linear method with additive trend estimation.

Graphical user interface, chart

Description automatically generated

Figure 15: Holt-Winter forecasting

|  |  |
| --- | --- |
| Chart, histogram  Description automatically generated | Chart, histogram  Description automatically generated |
| Figure 16: ACF of Holt-Winter Residuals | Figure 17: ACF of Holt-Winter Forecast errors |

This forecast has a residual Error of mean of -0.018 and a variance of 44.217. The forecast error means is -13.883 and the forecast error variance is 44.645. The Q-Value of residual Error is 724.570, which is extremely high.

Feature Selection:

The singular values of the features set are calculated and shown below,

Text

Description automatically generated

Figure 18: Singular value of the independent variables

Here at least four features are correlated as their singular values is closer to zero (figure 18). The condition number for x is 899939.56. The condition number k < 100 indicates Weak Degree of co-linearity (DOC), 100 < κ < 1000 indicates Moderate to Strong DOC and κ > 1000 =⇒ Severe DOC. Here the condition number is remarkably high so there is a severe Degree of co-linearity.

Here I am using PCA for feature elimination. Using python sklearn package’s PCA () function with n\_components = ‘mle’, it reduced the features from 15 to 14 (figure 19 left), but more features can be removed as with 5 just features we are getting more than 90% explained variance.

|  |  |
| --- | --- |
| Chart, line chart  Description automatically generated | Chart, line chart  Description automatically generated |
| Figure 19: Cumulative explained variance vs number of component | |

Text

Description automatically generated

Figure 20: Reduced feature space explained variance ratio

Graphical user interface, application, Teams

Description automatically generated

Figure 21: Reduced feature space correlation matrix

Now there is no correlation between of the principal components as all Pearson correlation coefficients are zero (figure 21).

Base models:

Average Forecasting Method:

This method forecast by considering the historical average data. It assumes that all observations have equal importance and give them equal weights. This method is useful for forecasting short-term trends.

Chart

Description automatically generated

Figure 22: Average forecasting

Chart, histogram

Description automatically generated

Figure 23: ACF of Average method residuals

The model performance scores are: -

MSE of prediction: 83.06

MSE of forecast: 71.53

Variance of prediction error: 72.41

Variance of Forecast error: 44.63

Q-Value: 7350.11

The Q-value is extremely high. This model might not be good at predicting average AQI.

Naïve Forecasting Method:

The Naïve method assumes that the most important observation is the only one, and all previous observations provide no information for the future.

Chart

Description automatically generated

Figure 24. Naïve method forecast

A picture containing chart

Description automatically generated

Figure 25: ACF of Naïve method residuals

The model performance scores are: -

MSE of prediction: 44.99

MSE of forecast: 260.13

Variance of prediction error: 44.99

Variance of Forecast error: 44.63

Q-Value: 524.85

The Q-value of residuals are high in this model as well, but better than the average method.

Drift Forecasting Method:

The drift method uses the technique of drawing a line between the first and last observations and extrapolating it into the future for forecasting.

Chart

Description automatically generated

Figure 26: Drift method forecasting

A picture containing chart

Description automatically generated

Figure 27: ACF of Drift method residuals

The model performance scores are: -

MSE of prediction: 45.13

MSE of forecast: 297.64

Variance of prediction error: 45.13

Variance of Forecast error: 45.82

Q-Value: 521.68

The Q-value is still high but better than the previous two methods.

Simple Exponential Smoothing (SES) Method:

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.

Chart

Description automatically generated

Figure 28: SES forecasting

Chart, histogram

Description automatically generated

Figure 29: ACF of SES method residuals

The model performance scores are: -

MSE of prediction: 44.193

MSE of forecast: 236.877

Variance of prediction error: 44.190

Variance of Forecast error: 44.630

Q-Value: 714.82

Multiple Linear Regression:

When there are two or more predictor variables, the model is called a multiple regression model. This model assumes that the relationship between the forecast variable and the predictor variables satisfies linear equations. The model’s performance is evaluated by using R-squared, adjusted R-squared, CV, AIC, BIC, T-test, F-test, condition number etc. R-squared is an intuitive measure of how well your linear model fits a set of observations. For example, R2 = 80% means that 80% of the variation in y can be explained by the relationship between x and y. The remaining 20% of the variation is unexplained and it is due to error. Every time a predictor is added to a model, the R-squared increases and does not necessarily improve the model performance. Akaike’s Information Criterion (AIC) tests how well the model fits the data set without overfitting it. In AIC and BIC measures, we want to find the model with lowest values while for adjusted R- squared, we seek the model with highest value. For this Multiple Regression model building I have created a full model with all the features as avgAQI as the dependent variable and reduced the features for a best model with a good, adjusted R-squared value and low condition number by observing based on the p-values and standard errors. All the models and their corresponding scores are highlighted in the following table.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Table 1: Multiple Linear Regression Models | | | | | |
| Model | Feature eliminated | Adj. R-squared | AIC | BIC | Condition Number |
| Full OLS Model | -- | 93.2% | 2.872e+04 | 2.882e+04 | 1.05e+06 |
| Model 2 | Month | 93.2% | 2.872e+04 | 2.882e+04 | 1.05e+06 |
| Model 3 | SO2 1st Max Hour | 93.2% | 2.871e+04 | 2.881e+04 | 1.00e+06 |
| Model 4 | Day | 93.2% | 2.872e+04 | 2.880e+04 | 1.00e+06 |
| Model 5 | O3 1st Max Hour | 93.2% | 2.872e+04 | 2.880e+04 | 1.00e+06 |
| Model 6 | NO2 1st Max Hour | 93.2% | 2.873e+04 | 2.881e+04 | 1.00e+06 |
| Model 7 | CO 1st Max Hour | 93.2% | 2.875e+04 | 2.881e+04 | 9.97e+05 |
| Model 8 | Year | 93.2% | 2.877e+04 | 2.883e+04 | 1.49e+04 |
| Model 9 | CO Mean | 93.1% | 2.881e+04 | 2.887e+04 | 1.46e+04 |
| Model 10 | SO2 Mean | 93.1% | 2.886e+04 | 2.890e+04 | 1.46e+04 |
| Model 11 | O3 Mean | 92.9% | 2.898e+04 | 2.902e+04 | 3.82e+03 |
| Model 12 | NO2 Mean | 92.6% | 2.928e+04 | 2.931e+04 | 3.20e+03 |
| Model 13 | SO2 1st Max Value | 91.0% | 3.049e+04 | 3.052e+04 | 3.20e+03 |
| Model 14 | NO2 1st Max Value | 79.8% | 3.554e+04 | 3.556e+04 | 100 |

Now the high condition number is reduced, and the final model contains only 2 features, O3 1st Max Value and CO 1st Max Value. The adjusted r squared is 79.83% which means 79.83% of the variation in y can be explained by the relationship between the features and dependent variable. The OLS model summary is given below.

|  |
| --- |
| Final Model Summary |
|  |
| A picture containing graphical user interface  Description automatically generated |
| Figure 30: Multiple Linear Regression Model |

Hypothesis Tests:

**T Test** p-values:

const 1.742061e-63

O3 1st Max Value 0.000000e+00

CO 1st Max Value 0.000000e+00

As the p-values of the T test is less than the significant level alpha = 0.05, we reject the null hypothesis and conclude that there is a statistically significant relationship between the predictor variable and the response variable.

**F Test** p-value: 0.0

As the p-value of the F test is less than the significant level alpha = 0.05, we can reject the null-hypothesis and conclude that final model provides a better fit than the intercept-only model.

AIC: 35540.00

BIC: 35560.24

RMSE: -

Residual: 4.095

Forecast: 3.670

R-Squared Value: 79.84%

Adj-R Squared Value: 79.83%

Overall, the final model's performance is surprisingly good. In this final model, 79.83% variation in dependent variable 'avgAQI' can be explained by the independent variables. The RMSE values are low as well.

Chart, histogram

Description automatically generated

Figure 31: ACF of OLS residuals

The model performance scores are: -

MSE of Residual Error: 16.769

MSE of Forecast Error: 13.466

Q-Value of Residual Error: 11030.564

Mean of residuals: 0.00

Variance of residuals: 16.77

General Partial Autocorrelation (GPAC):

A picture containing histogram

Description automatically generated

Figure 32: ACF of dependent variable Trainset

Chart, treemap chart

Description automatically generated

Figure 33: GPAC table of y train set

Observing the patterns ARMA (1,0) and ARMA (3,1) can be selected for farther analysis.

ARMA:

Autoregressive moving average (ARMA (na, nb)) models are the combination of AR(na) and MA(nb) models. Compared with the pure autoregressive (AR) or moving average (MA) models, ARMA models provide the most effective linear model of stationary time series since they can model the unknown process with the minimum number of parameters.

ARMA (1,0):

Table

Description automatically generated

Figure 34: ARMA (1,0) model summary

The AR coefficient a0 is: -0.97. As the interval does not contain zero in it, it is statistically important.

|  |  |
| --- | --- |
| A picture containing chart  Description automatically generated | Chart, histogram  Description automatically generated |
| Figure 35.1: ACF of ARMA (1,0) residuals | Figure 35.2: ACF of ARMA (1,0) forecast errors |

MSE of Residuals: 44.36

MSE of Forecast Error: 11.01

Q-Value: 514.35

|  |
| --- |
|  |
| Figure 35.3: ACF of ARMA (1,0) Covariance Matrix |

ARMA (3,1):

Table

Description automatically generated

Figure 36: ARMA (3,1) model summary

The AR coefficient a0 = -1.69, a1 = 0.79, a2 = -0.10 and the MA coefficient b0 is -0.97. As none of the intervals does not contain zero in it, they are statistically important.

|  |  |
| --- | --- |
| A picture containing histogram  Description automatically generated | Chart, histogram  Description automatically generated |
| Figure 37.1: ARMA (3,1) Residuals | Figure 37.2: ARMA (3,1) forecast error |

MSE of Residuals: 37.50

MSE of Forecast Error: 46.60

Q-Value: 167.51

|  |
| --- |
|  |
| Figure 37.3: ARMA (3,1) Covariance Matrix |

ARIMA:

ARIMA (1,1,0):

Table

Description automatically generated

Figure 38: ARIMA (1,1,0) model summary

The AR coefficient a1 is: 0.10. As the interval does not contain zero in it, it is statistically important.

|  |  |
| --- | --- |
| Chart  Description automatically generated with medium confidence | A picture containing chart  Description automatically generated |
| Figure 39.1: ARIMA (1,1,0) residuals | Figure 39.2: ARIMA (1,1,0) forecast errors |

MSE of Residuals: 0.62

MSE of Forecast Error: 0.00

Q-Value: 651.29

|  |
| --- |
|  |
| Figure 39.3: ARIMA (1,1,0) covariance matrix |

Among ARMA (1,0) model ARMA (3,1), ARMA (1.0) has lower Q-value, but ARMA (3,1) is better at forecasting.

ARIMA (3,1,1):

Table

Description automatically generated

Figure 40: ARIMA (3,1,1) model summary

The AR coefficients: a1 = -0.69, a2 = 0.09, a3 = 0.01 and MA coefficient b1 = -0.97. Here interval of AR coefficient a2 contains zero, it is statistically not important in this model.

|  |  |
| --- | --- |
| Histogram  Description automatically generated | Chart, histogram  Description automatically generated |
| Figure 41.1: ARIMA (3,1,1) residuals | Figure 41.2: ARIMA (3,1,1) forecast errors |

MSE of Residuals: 7.72

MSE of Forecast Error: 0.03

Q-Value: 167.33

|  |
| --- |
|  |
| Figure 41.3: ARIMA (3,1,1) covariance matrix |

Among ARIMA (1,1,0) model ARMA (3,1,1), ARIMA (3,1,1) has lower Q value, but ARMA (1,1,0) is better at forecasting.

SARIMA:

Here I am performing SARIMA (3,0,1) x (0,2,0,7) even though my dataset is not seasonal.

Table

Description automatically generated

Figure 42: SARIMA (3,0,1) x (0,2,0,7) model summary

|  |  |
| --- | --- |
| Chart  Description automatically generated | Chart, histogram  Description automatically generated |
| Figure 43.1: SARIMA (3,0,1) x (0,2,0,7) residuals | Figure 43.2: SARIMA (3,0,1) x (0,2,0,7) forecast errors |

MSE of Residuals: 155.87

MSE of Forecast Error: 386.41

Q-Value: 702.40

Levenberg Marquardt algorithm:

I am using this LM algorithm to estimate the coefficients of ARMA (3,1) models.

Text

Description automatically generated

Figure 44: LM algorithm estimated parameters

The coefficients are statistically important as the interval does not include 0.

Diagnostic Analysis

|  |  |  |
| --- | --- | --- |
| Table 2: Confidence Intervals | | |
|  |  |  |
|  |  |  |

Among all the confidence intervals of the six model’s coefficients, only ARIMA (3,1,1) includes zero in the interval of a2. Which indicates it is statistically not important in this model.

|  |  |
| --- | --- |
| Table 3: Zero Pole Cancellation | |
|  |  |
|  |  |
|  | |

None of the models have zero pole cancellations as the numerators and denominators are not close.

Q-Values of Residual Error:

OLS: 11030.564

ARMA (1,0): 514.351

ARMA (3,1): 167.514

ARIMA (1,1,0): 651.286

ARIMA (3,1,1): 167.332

SARIMA (3,0,1) x (0,2,0,7): 702.396

Variance of Residual Errors:

OLS: 16.77

ARMA (1,0): 43.71

ARMA (3,1): 37.50

ARIMA (1,1,0): 44.69

ARIMA (3,1,1): 37.59

SARIMA: 155.87

Variance of Forecast Errors:

OLS: 9.67

ARMA (1,0): 52.02

ARMA (3,1): 45.02

ARIMA (1,1,0): 0.00

ARIMA (3,1,1): 0.03

SARIMA: 278.43

MSE of Residuals:

OLS: 16.77

ARMA (1,0): 44.36

ARMA (3,1): 37.50

ARIMA (1,1,0): 44.69

ARIMA (3,1,1): 37.59

SARIMA: 155.87

MSE of Forecasts:

OLS: 13.47

ARMA (1,0): 409.85

ARMA (3,1): 46.60

ARIMA (1,1,0): 0.00

ARIMA (3,1,1): 0.03

SARIMA: 386.41

Final Model Selection

Base models comparison:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Model Name | Q-value | Variance of Errors | | MSE | |
| Residuals | Forecasts | Residuals | Forecasts |
| Holt-Winter | 724.570 | 44.217 | 44.645 | 44.217 | 237.388 |
| Average | 7350.11 | 72.41 | 44.63 | 83.06 | 71.53 |
| Naive | 524.85 | 44.99 | 44.63 | 44.99 | 260.13 |
| Drift | 521.68 | 45.13 | 45.82 | 45.13 | 297.64 |
| SES | 714.82 | 44.190 | 44.63 | 44.193 | 236.877 |
| Best Model | Drift | SES | SES | SES | Average |

Among the base models, in terms of q values, Drift method is the best model but by considering MSE and other metrices SES is the best among all.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Model Name | Q-value | Variance of Errors | | MSE | | Ratio of variance of test set/forecasted |
| Residuals | Forecasts | Residuals | Forecasts |
| OLS | 11030.564 | 16.77 | 9.67 | 16.77 | 13.47 | 1.14 |
| ARMA (1,0) | 514.351 | 43.71 | 52.02 | 44.36 | 409.85 | 4.22 |
| ARMA (3,1) | 167.514 | 37.50 | 45.02 | 37.50 | 46.60 | 20.88 |
| ARIMA (1,1,0) | 651.286 | 44.69 | 0.00 | 44.69 | 0.00 | 1.00 |
| ARIMA (3,1,1) | 167.332 | 37.59 | 0.03 | 37.59 | 0.03 | 1.00 |
| SARIMA | 702.396 | 155.87 | 278.43 | 155.87 | 386.41 | 0.19 |
| Best model: | ARIMA (3,1,1) | OLS | ARIMA (1,1,0) | OLS | ARIMA (1,1,0) | ARIMA (1,1,0)  ARIMA (3,1,1) |

In terms of lowest q-value, ARIMA (3,1,1) is the best model among these 6 models. But ARIMA (1,1,0) is best at forecasting. As the difference between variance of errors and MSE of the two ARIMA models does not differ that much, overall, the best model is ARIMA (3,1,1). It has the low q-value, MSE and variance of errors and the ratio of test set by forecasted set is 1.

Forecast

The best model is ARIMA (3,1,1). The AR coefficients are: a1 = -0.69, a2 = 0.09, a3 = 0.01 and MA coefficient is b1 = -0.97. The equation of model is:

y(t) – 0.69 y(t-1) + 0.09 y(t-2) + 0.01 y(t-3) = e(t) – 0.97 e(t-1)

Histogram

Description automatically generated

Figure 45: ACF of residuals of final model

The code for 1-step forecasting is:

y\_train\_diff = y\_train.diff().dropna()

y\_hat = []

for i in range(1, len(y\_train\_diff)):

    if i==1:

      y\_hat.append((0.69\*y\_train\_diff[i-1]) -(0.97 \*y\_train\_diff[i-1]))

    elif i == 2:

      y\_hat.append((0.69\*y\_train\_diff[i-1]) - (0.09 \* y\_train\_diff[i-2]) - (0.97\*(y\_train\_diff[i-1] - y\_hat[0])))

    else:

      y\_hat.append((0.69\*y\_train\_diff[i-1]) - (0.09\*y\_train\_diff[i-2]) -(0.01\*y\_train\_diff[i-3]) - (0.97\*(y\_train\_diff[i-1] - y\_hat[-1])) )

y\_hat\_inv\_diff = inverse\_diff(y\_train.values,np.array(y\_hat),1)

Chart

Description automatically generated

Figure 46: Forecast with final ARIMA (3,1,1) model

h-step ahead prediction:

Chart, histogram

Description automatically generated

Figure 47: 7-step prediction in ARIMA (3,1,1) model

Here on the 7-step prediction, the variance of test set is 44.63, variance of predicted set is 49.81 and the ratio is 0.90.

Chart, line chart

Description automatically generated

Figure 48: 30-step prediction on ARIMA (3,1,1) model

Here on the 30-step prediction, the variance of test set is 44.63, variance of predicted set is 43.48 and the ratio is 1.03.

As the ratio is around 1 this model is good at multi step prediction. This model is good at predict next week as well as next month’s average AQI.

Summary and Conclusion

This project has used diverse types of forecasting techniques to predict Loss Angeles county’s average Air Quality Index including the simple forecasting models, multiple linear regression model, ARMA, ARIMA and SARIMA models. In the simple models, Simple Exponential Smoothing forecasting was better than the other five models by considering the lowest variance of errors and MSE. But as the dataset is not seasonal Holt-Winter and SARIMA models are not appropriate for forecasting in this case. Among the ARMA and ARIMA models ARIMA model with AR (3), MA (1) with differencing order 1, performed better than the other by considering the lowest q-value of residuals, MSE, ratio of variance of test set vs forecasted set. From this model I farther generated the model equation and built 1-step and multi-step prediction functions and the model is exceptionally good at predict next week as well as next month’s average AQI. Overall, the models did not have white q-value of residuals possibly because of the nature of the dataset. More advanced machine learning technique of forecasting like LSTM, XGboost etc. can achieve that.

Appendix

Source code of full analysis

#%%

from sympy import rotations

from toolbox import \*

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

plt.style.use('seaborn-poster')

from statsmodels.tsa.seasonal import STL

import statsmodels.tsa.holtwinters as ets

import statsmodels.api as sm

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error

from numpy import linalg as LA

import warnings

warnings.filterwarnings('ignore')

#%%

########### Description of the dataset

# a. Pre-processing dataset:

df = pd.read\_csv('data/AQI\_CA\_LA.csv', index\_col='Date')

print ('This dataset is about the Air Quality Index of California Loss Angeles county.')

print ("1st 5 values of the dataset: \n",df.head())

#%%

print ('Details of dataset: \n')

print(df.info ())

print(f"The dataset contains {df.shape[0]} number of rows and \

{df.shape[1]} columns and does not contain any missing values.\

It has the AQI data from {df.index[0]} to {df.index[-1]}. ")

#%%

print(df.describe())

# %%

########## b. plotting dependent variable vs time.

print ("For this time series analysis, my dependant variable is 'avgAQI'\

 which is the the average Air Quality Index of O3, CO, SO2 and NO2. ")

plt.figure(figsize=(14,8))

plt.plot(df.index, df['avgAQI'], label='Dependant Variable-avgAQI')

plt.xticks(df.index[::981])

plt.xlabel('Time', fontsize=22)

plt.ylabel('Average Air Quality Index (AQI)', fontsize=22)

plt.tight\_layout()

plt.title('Dependant Variable-avgAQI vs Time',  fontsize=30)

plt.legend(fontsize=24)

plt.grid()

plt.show()

#%%

######### c. ACF/PACF of the dependent variable

ACF\_PACF\_Plot(df.avgAQI, 100)

# %%

######### d. Correlation Matrix with seaborn heatmap with the Pearson’s correlation coefficient

# df2 = df.copy()

# df2.drop(columns=['Year','Month','Day','Address','State','City','County'], inplace=True)

dff = df.drop(['O3 AQI','CO AQI','SO2 AQI','NO2 AQI'], axis=1)

plt.figure(figsize=(14,14))

sns.heatmap(dff.corr(), vmin=-1,vmax=1,cmap='RdBu\_r', annot=True)

plt.title('Correlation Matrix of AQI Dataset')

plt.show()

# %%

######### e. Split the dataset into train set (80%) and test set (20%)

# train,test=train\_test\_split(df,test\_size=0.2,shuffle=False)

# print ("Train set: ", train.shape)

# print("Test set: ", test.shape)

X = df.copy()

X = X.drop(['avgAQI', 'O3 AQI','CO AQI','SO2 AQI','NO2 AQI'], axis=1)

y = df['avgAQI']

x\_train,x\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.2,shuffle=False)

print("Train set: ", x\_train.shape)

print("Test set: ", x\_test.shape)

# %%

######### 7. Stationarity Check

# Original dataset-rolling mean variance

print('Original dataset-rolling mean & variance')

Cal\_rolling\_mean\_var(df['avgAQI'])

print('The rolling mean is downward slopping but rolling variance is stabilizes once all samples are included.')

# %%

# ADF Test

ADF\_Cal(df['avgAQI'])

print('The ADF p-value below a threshold (1% or 5%) suggests that we reject the null hypothesis and conclude that the data is stationary.')

# %%

# KPSS Test

kpss\_test(df['avgAQI'])

print('The KPSS p-value below a threshold (1% or 5%) suggests that we reject the null hypothesis and conclude that the data is non stationary.')

# %%

# ACF

acf(df.avgAQI,100,plot=True, title='ACF of avgAQI')

# %%

# 1st order differentiation

# df2 = df.copy()

# df2['avgAQI'] = df2['avgAQI'].diff(1)

# Cal\_rolling\_mean\_var(df2['avgAQI'])

# ADF\_Cal(df2['avgAQI'].dropna())

# kpss\_test(df2['avgAQI'].dropna())

# print('After 1st order differenciation, the mean of dependant variable is zero\

#  and both ADF and KPSS tests indicates stationarity.')

# ACF\_PACF\_Plot(df2['avgAQI'].dropna(), 100)

# %%

# log transform then 1nd order differentiation

# df3 = df.copy()

# df3['avgAQI'] = df3['avgAQI'].transform(np.log).diff(1).dropna()

# Cal\_rolling\_mean\_var(df3['avgAQI'])

# ADF\_Cal(df3['avgAQI'].dropna())

# kpss\_test(df3['avgAQI'].dropna())

# ACF\_PACF\_Plot(df3['avgAQI'].dropna(), 50)

# %%

######### 8- Time series Decomposition:

aqi = df['avgAQI']

aqi=pd.Series(np.array(df['avgAQI']),index = pd.date\_range('2000-01-01',periods= len(df)))

STL = STL(aqi)

res = STL.fit()

fig = res.plot()

plt.xlabel("Time (Year)")

plt.suptitle('STL Decomposition', y=1.05)

plt.show()

# %%

T = res.trend

S = res.seasonal

R = res.resid

adj\_seasonal = df['avgAQI'] - S

detrended\_Temp = df['avgAQI'] - T

plt.figure()

plt.plot(T,label="Trend")

plt.plot(R,label="Residual")

plt.plot(S,label="Seasonal")

plt.xlabel("Time")

plt.ylabel("STL")

plt.legend(loc='upper right')

plt.title("Trend, Residuals and Seasonality of 'AvgAQI'")

plt.show()

# %%

# Strength of trend

F\_t = np.maximum(0, 1 - np.var(np.array(R)) / np.var(np.array(T) + np.array(R)))

print(f"The strength of trend for this dataset is {F\_t:.3f}")

# %%

# Strength of seasonality

F = np.maximum(0, 1 - np.var(np.array(R)) / np.var(np.array(S) + np.array(R)))

print(f"The strength of seasonality for this dataset is {F:.3f}")

# %%

print(f'Observing the graphs, trend, and strength of seasonality values,\

  this dataset has low seasonality ({F:.3f}) and is trended ({F\_t:.3f}).')

# %%

#seasonally adjusted data

seasonally\_adj=aqi-S

#Detrended data

detrended=aqi-T

fig, axes = plt.subplots(nrows=2, ncols=1, figsize=(12, 8))

axes[0].plot(df.index,df.avgAQI,label="Original")

axes[0].plot(df.index,seasonally\_adj,label="adjusted")

axes[0].set\_xlabel("Time")

axes[0].set\_xticks(df.index[::1500])

axes[0].set\_ylabel("Average AQI")

axes[0].set\_title("Seasonality adjusted vs. Original")

axes[0].legend(loc='upper right')

axes[1].plot(df.index,df.avgAQI,label="Original")

axes[1].plot(df.index,detrended,label="Detrended")

axes[1].set\_xlabel("Time")

axes[1].set\_xticks(df.index[::1500])

axes[1].set\_ylabel("Average AQI")

axes[1].set\_title("Detrended vs. Original Data")

axes[1].legend(loc = 'upper right')

plt.tight\_layout()

plt.show()

#%%

# Moving Average (3)

dfma = df.copy()

dfma['avgAQI\_MA'] = dfma['avgAQI'].rolling(3).mean()

dfma.dropna(inplace=True)

dfma[['avgAQI', 'avgAQI\_MA']].plot(label='AQI',

                                  figsize=(16, 8))

plt.title('3-MA')

plt.grid()

plt.show()

# %%

######## 9. Holt-Winters method:

x\_train,x\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.2,shuffle=False)

holtt = ets.ExponentialSmoothing(y\_train,  trend='add', damped\_trend=True, seasonal=None).fit()

# holtt\_predt=holtt.forecast(steps=len(y\_train))

holtt\_predt=holtt.fittedvalues

holtt\_df=pd.DataFrame(holtt\_predt,columns=['avgAQI']).set\_index(y\_train.index)

holtt\_forcst=holtt.forecast(steps=len(y\_test))

holtf\_df=pd.DataFrame(holtt\_forcst,columns=['avgAQI']).set\_index(y\_test.index)

plt.figure(figsize=(16,8))

plt.plot(y\_train.index,y\_train, label='Train', color = 'b')

plt.plot(y\_test.index,y\_test, label='Test', color = 'g')

# plt.plot(holtt\_df.index,holtt\_df['avgAQI'],label='Holts winter prediction', color = 'skyblue', linestyle='dashed')

plt.plot(holtf\_df.index,holtf\_df['avgAQI'],label='Holts winter forecast', color = 'r')

plt.xticks(holtt\_df.index.values[::1300])

plt.legend(loc = 'upper right')

plt.xlabel("Time")

plt.ylabel("Average AQI")

plt.title("Holts Winter Method")

plt.grid()

plt.show()

# MSE

holtt\_mse = mean\_squared\_error(y\_train, holtt\_df['avgAQI'])

print(f"Holt Winter Train set MSE: {holtt\_mse:.3f}")

holtf\_mse = mean\_squared\_error(y\_test, holtf\_df['avgAQI'])

print(f"Holt Winter Test set MSE: {holtf\_mse:.3f}")

#%%

# ERRORS

res\_err = y\_train - holtt\_df['avgAQI']

print(f"Residual Error Mean {np.mean(res\_err):.3f}")

print(f"Residual Error Variance {np.var(res\_err):.3f}")

fcst\_err = y\_test - holtf\_df['avgAQI']

print(f"Forecast error Mean: {np.mean(fcst\_err):.3f}")

print(f"Forecast error Variance: {np.var(fcst\_err):.3f}")

acf(res\_err, 50,plot=True, title='ACF of Prediction Error')

acf(fcst\_err, 50,plot=True, title='ACF of Forecast Error')

# Q-Value

# re = acf(res\_err, 50, plot=False)

# res\_q = len(y\_train)\*np.sum(np.square(re[50+2:]))

# res\_q = len(y\_train)\*np.sum(res\_q)

res\_q = q\_value(res\_err, 50, len(y\_train))

print(f"Q-Value of Residual Error: {res\_q:.3f}")

#%%

qstar,pvalue=sm.stats.acorr\_ljungbox(res\_err,lags=[50])

# print(f"Q\*-Value of Residual Error: {qstar[0]:.3f}")

# if res\_q < qstar:

#   print("The residual is white as Q < Q\*")

# else:

#   print("The residual is not white as Q > Q\* ")

if pvalue > 0.05:

  print("The residual is white as p > 0.05")

else:

  print("The residual is not white as p < 0.05 ")

# %%

######## 10. Feature Selection

svd\_df = df.select\_dtypes(include='number')

svd\_df.drop(['O3 AQI','CO AQI','SO2 AQI','NO2 AQI'], axis=1, inplace=True)

sdv\_df = svd\_df.drop('avgAQI', axis=1)

X = sm.add\_constant(sdv\_df)

Y = df['avgAQI']

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X,Y, shuffle=False, test\_size=0.2)

Xx = X.values

H = np.matmul(Xx.T, Xx)

\_,d,\_ = np.linalg.svd(H)

print(f'singular value for X are :- \n{pd.DataFrame(d)}')

# Any singular values close to zero means that one or more

# features are correlated. The correlated feature(s) needs to be

# detected and removed from the feature space.

print('\nAt least 4 features are correlated as their singular values is closer to zero.')

print(f'\nThe condition number for x is {LA.cond(Xx):.2f}') # its k -> if small <100 then good

# The κ < 100 =⇒ Weak Degree of Co-linearity(DOC)

# • The 100 < κ < 1000 =⇒ Moderate to Strong DOC

# • The κ > 1000 =⇒ Severe DOC

print("\nAs the condition number is very high there is a severe Degree of Co-linearity.")

#%%

print("I am using PCA for feature elimination.")

from sklearn.decomposition import PCA

pca=PCA(n\_components='mle',svd\_solver='full')

pca.fit(sdv\_df)

aqi\_pca=pca.transform(sdv\_df)

print("Original Dimension:",sdv\_df.shape)

print("Transformed dimension:", aqi\_pca.shape)

print("Explained variance ratio:\n",pca.explained\_variance\_ratio\_)

x=np.arange(1,len(np.cumsum(pca.explained\_variance\_ratio\_))+1,1)

plt.plot(x,np.cumsum(pca.explained\_variance\_ratio\_))

plt.xticks(x)

plt.xlabel("Number of Components")

plt.ylabel("Cumulative Explained Variance")

plt.title("Cumulative Explained Variance vs Number of Components")

plt.grid()

plt.show()

print('PCA already reduced the features from 15 to 14, but\

 more feature can be removed as with 5 just features we are getting\

 more than 90% explained variance. ')

# Making new reduced feature space with 5 components

pcaf=PCA(n\_components=5,svd\_solver='full')

pcaf.fit(sdv\_df)

reduced\_aqi\_pcaf=pcaf.transform(sdv\_df)

print("\nOriginal Dimension:",sdv\_df.shape)

print("Transformed Dimension:",reduced\_aqi\_pcaf.shape)

print("Explained variance ratio:\n",pcaf.explained\_variance\_ratio\_)

x=np.arange(1,len(np.cumsum(pcaf.explained\_variance\_ratio\_))+1,1)

plt.plot(x,np.cumsum(pcaf.explained\_variance\_ratio\_))

plt.xticks(x)

plt.xlabel("Number of Components")

plt.ylabel("Cumulative Explained Variance")

plt.title("Cumulative Explained Variance Vs Number of Components")

plt.suptitle("Reduced Feature Space", fontsize = 22)

plt.grid()

plt.show()

PlayStore\_pcaf\_df=pd.DataFrame(reduced\_aqi\_pcaf).corr()

column=[]

for i in range(reduced\_aqi\_pcaf.shape[1]):

    column.append(f'Pricipal Component {i+1}')

plt.figure(figsize=(8,6))

sns.heatmap(PlayStore\_pcaf\_df,annot=True, xticklabels=column,yticklabels=column)

plt.title("Correlation Coefficient of Reduced Feature Space")

plt.show()

#%%

# OLS

# Full model

model=sm.OLS(Y\_train,X\_train).fit()

print(model.summary())

#%%

# maximum p of model 1

max\_p=pd.DataFrame(model.pvalues,columns=['P\_Values']).set\_index(model.pvalues.index)

max\_p = max\_p[max\_p['P\_Values'] == max\_p['P\_Values'].max()]

print(f"\nMaximum p-value: \n{max\_p}")

print("The highest P-value is on column 'Month'.Let's drop this feature\

 from the train set and rebuilt the model.")

#%%

# model 2

X\_train.drop(['Month'], axis=1, inplace=True)

model2 = sm.OLS(Y\_train,X\_train).fit()

print('Model 2: After dropping Month: \n',model2.summary())

#%%

# max p

max\_p=pd.DataFrame(model2.pvalues,columns=['P\_Values']).set\_index(model2.pvalues.index)

max\_p = max\_p[max\_p['P\_Values'] == max\_p['P\_Values'].max()]

print(f"\nMaximum p-value: \n{max\_p}")

print("The highest P-value is on column 'SO2 1st Max Hour'.Let's drop this feature\

 from the train set and build the 3rd model.")

#%%

# model 3

X\_train.drop(['SO2 1st Max Hour'], axis=1, inplace=True)

model3 = sm.OLS(Y\_train,X\_train).fit()

print('Model 3: After dropping SO2 1st Max Hour: \n',model3.summary())

#%%

# max p\_values

max\_p=pd.DataFrame(model3.pvalues,columns=['p\_values']).set\_index(model3.pvalues.index)

max\_p = max\_p[max\_p['p\_values'] == max\_p['p\_values'].max()]

print(f"\nMaximum p\_values: \n{max\_p}")

print("The highest p\_values is on column 'Day'.Let's drop this feature\

 from the train set and build the 4th model.")

#%%

# Model 4

X\_train.drop(['Day'], axis=1, inplace=True)

model4 = sm.OLS(Y\_train,X\_train).fit()

print('Model 4: After dropping Day: \n',model4.summary())

#%%

# max p

max\_p=pd.DataFrame(model4.pvalues,columns=['P\_Value']).set\_index(model4.pvalues.index)

max\_p = max\_p[max\_p['P\_Value'] == max\_p['P\_Value'].max()]

print(f"\nMaximum p-value: \n{max\_p}")

print("The highest P-value is on column 'O3 1st Max Hour'.Let's drop this feature\

 from the train set and build the 5th model.")

#%%

# Model 5

X\_train.drop(['O3 1st Max Hour'], axis=1, inplace=True)

model5 = sm.OLS(Y\_train,X\_train).fit()

print('Model 5: After dropping O3 1st Max Hour: \n',model5.summary())

#%%

# max p

max\_p=pd.DataFrame(model5.pvalues,columns=['P\_Value']).set\_index(model5.pvalues.index)

max\_p = max\_p[max\_p['P\_Value'] == max\_p['P\_Value'].max()]

print(f"\nMaximum P\_Value: \n{max\_p}")

print("The highest P\_Value is on column 'NO2 1st Max Hour'.Let's drop this feature\

 from the train set and build the 6th model.")

#%%

# Model 6

X\_train.drop(['NO2 1st Max Hour'], axis=1, inplace=True)

model6 = sm.OLS(Y\_train,X\_train).fit()

print('Model 6: After dropping NO2 1st Max Hour: \n',model6.summary())

#%%

# max p\_values

max\_p=pd.DataFrame(model6.pvalues,columns=['p\_values']).set\_index(model6.pvalues.index)

max\_p = max\_p[max\_p['p\_values'] == max\_p['p\_values'].max()]

print(f"\nMaximum p\_values: \n{max\_p}")

print("The highest p\_values is on column 'CO 1st Max Hour'.Let's drop this feature\

 from the train set and build the 7th model.")

#%%

# Model 7

X\_train.drop(['CO 1st Max Hour'], axis=1, inplace=True)

model7 = sm.OLS(Y\_train,X\_train).fit()

print('Model 7: After dropping CO 1st Max Hour: \n',model7.summary())

#%%

# max P\_Value

max\_p=pd.DataFrame(model7.pvalues[1:],columns=['P\_Value']).set\_index(model7.pvalues.index[1:])

max\_p = max\_p[max\_p['P\_Value'] == max\_p['P\_Value'].max()]

print(f"\nMaximum p-value: \n{max\_p}")

print("The highest P\_Value is on column 'Year'.Let's drop this feature\

 from the train set and buil the 8th model.")

#%%

# Model 8

X\_train.drop(['Year'], axis=1, inplace=True)

model8 = sm.OLS(Y\_train,X\_train).fit()

print('Model 8: After dropping Year: \n',model8.summary())

#%%

# max p\_value

max\_p=pd.DataFrame(model8.pvalues,columns=['p\_value']).set\_index(model8.pvalues.index)

max\_p = max\_p[max\_p['p\_value'] == max\_p['p\_value'].max()]

print(f"\nMaximum p\_value: \n{max\_p}")

print("The highest p\_value is on column 'CO Mean'.Let's drop this feature\

 from the train set and build the 9th model.")

#%%

# Model 9

X\_train.drop(['CO Mean'], axis=1, inplace=True)

model9 = sm.OLS(Y\_train,X\_train).fit()

print('Model 9: After dropping CO Mean: \n',model9.summary())

#%%

# max p

max\_p=pd.DataFrame(model9.pvalues,columns=['P\_Value']).set\_index(model9.pvalues.index)

max\_p = max\_p[max\_p['P\_Value'] == max\_p['P\_Value'].max()]

print(f"\nMaximum p-value: \n{max\_p}")

print("The highest P-value is on column 'SO2 Mean'.Let's drop this feature\

 from the train set and build the 10th model.")

#%%

# Model 10

X\_train.drop(['SO2 Mean'], axis=1, inplace=True)

model10 = sm.OLS(Y\_train,X\_train).fit()

print('Model 10: After dropping SO2 Mean: \n',model10.summary())

#%%

# max p

max\_p=pd.DataFrame(model10.pvalues,columns=['P\_Value']).set\_index(model10.pvalues.index)

max\_p = max\_p[max\_p['P\_Value'] == max\_p['P\_Value'].max()]

print(f"\nMaximum p-value: \n{max\_p}")

print("The highest P-value is on column 'O3 Mean'.Let's drop this feature\

 from the train set and build the 11th model.")

#%%

# Model 11

X\_train.drop(['O3 Mean'], axis=1, inplace=True)

model11 = sm.OLS(Y\_train,X\_train).fit()

print('Model 11: After dropping O3 Mean: \n',model11.summary())

#%%

# max p\_value

max\_p=pd.DataFrame(model11.pvalues,columns=['p\_value']).set\_index(model11.pvalues.index)

max\_p = max\_p[max\_p['p\_value'] == max\_p['p\_value'].max()]

print(f"\nMaximum p\_value: \n{max\_p}")

print("The highest p\_value is on column 'NO2 Mean'.Let's drop this feature\

 from the train set and build the 12th model.")

#%%

# Model 12

X\_train.drop(['NO2 Mean'], axis=1, inplace=True)

model12 = sm.OLS(Y\_train,X\_train).fit()

print('Model 12: After dropping NO2 Mean: \n',model12.summary())

#%%

# max p\_value

max\_p=pd.DataFrame(model12.pvalues,columns=['p\_value']).set\_index(model12.pvalues.index)

max\_p = max\_p[max\_p['p\_value'] == max\_p['p\_value'].max()]

print(f"\nMaximum p\_value: \n{max\_p}")

print("The highest p\_value is on column 'SO2 1st Max Value'.Let's drop this feature\

 from the train set and build the 13th model.")

#%%

# Model 13

X\_train.drop(['SO2 1st Max Value'], axis=1, inplace=True)

model13 = sm.OLS(Y\_train,X\_train).fit()

print('Model 13: After dropping SO2 1st Max Value: \n',model13.summary())

#%%

# max p\_value

max\_p=pd.DataFrame(model13.pvalues,columns=['p\_value']).set\_index(model13.pvalues.index)

max\_p = max\_p[max\_p['p\_value'] == max\_p['p\_value'].max()]

print(f"\nMaximum p\_value: \n{max\_p}")

print('All the pvalues are 0 but the condition number is high. Lets drop the highest std err O3 1st Max Value')

#%%

# Model 14

X\_train2 = X\_train.copy()

X\_train2.drop(['O3 1st Max Value'], axis=1, inplace=True)

model14 = sm.OLS(Y\_train,X\_train2).fit()

print('Model 14: After dropping O3 1st Max Value: \n',model14.summary())

#%%

print('After dropping O3 1st Max Value, the adjusted r squared drops a lot. So instead of dropping O3 1st Max Value, lets drop CO 1st Max Value')

#%%

# Model 15

X\_train3 = X\_train.copy()

X\_train3.drop(['CO 1st Max Value'], axis=1, inplace=True)

model15 = sm.OLS(Y\_train,X\_train3).fit()

print('Model 15: After dropping CO 1st Max Value: \n',model15.summary())

print('Now by dropping CO 1st Max Value, the condition number is still high. So instead of dropping CO 1st Max Value, lets drop NO2 1st Max Value')

#%%

# Model 16

X\_train4 = X\_train.copy()

X\_train4.drop(['NO2 1st Max Value'], axis=1, inplace=True)

model16 = sm.OLS(Y\_train,X\_train4).fit()

print('Model 16: After dropping NO2 1st Max Value: \n',model16.summary())

#%%

# Final model - model16

print(f"Now the high condition number is reduced and the final mmodel contains only 2 features,\

 O3 1st Max Value and CO 1st Max Value. The adjusted r squared is  {((model16.rsquared\_adj)\*100):2f}%")

X\_train.drop(['NO2 1st Max Value'], axis=1, inplace=True)

X\_test = X\_test[['const','O3 1st Max Value','CO 1st Max Value']]

#%%

final\_model = sm.OLS(Y\_train,X\_train).fit()

# %%

######## 11. Base model

# average

# 1 - step

# df\_avg = pd.DataFrame(data = {'yt':Y\_train})

df\_avg = pd.DataFrame(data = {'yt':y\_train})

df\_avg['y\_hat'] = avg\_pred(df\_avg.index,df\_avg['yt'])

df\_avg['e'] = df\_avg['yt']-df\_avg['y\_hat']

df\_avg['e^2'] = round(df\_avg['e']\*\*2, 2)

# h - step

# df\_avg\_h = pd.DataFrame(data = {'yt+h':Y\_test})

df\_avg\_h = pd.DataFrame(data = {'yt+h':y\_test})

df\_avg\_h['y\_hat'] = np.mean(df\_avg['yt'])

df\_avg\_h['e'] = df\_avg\_h['yt+h']-df\_avg\_h['y\_hat']

df\_avg\_h['e^2'] = round(df\_avg\_h['e']\*\*2, 2)

plt.figure(figsize=(16,6))

plt.plot(df\_avg.index,df\_avg['yt'], label='Training Dataset', color='b')

plt.plot(df\_avg\_h.index,df\_avg\_h['yt+h'], label = 'Testing Dataset', color='g')

plt.plot(df\_avg\_h.index,df\_avg\_h['y\_hat'], label = 'Avg Method H-step prediction', color='r', linestyle='dashed')

# plt.xticks(df.index[::981])

plt.xticks(ticks=range(0,len(df))[::981], labels = df.index[::981])

plt.title('Average Method & Forecast')

plt.xlabel('time')

plt.ylabel('Values')

plt.legend()

plt.show()

# MSE of prediction

MSE\_pred = round(np.mean(df\_avg['e^2']),2)

print("MSE of prediction: ", MSE\_pred)

# MSE of forecast

MSE\_forecast = round(np.mean(df\_avg\_h['e^2']),2)

print("MSE of forecast: ", MSE\_forecast)

# Variance error of prediction

var\_err\_pred = round(np.var(df\_avg['e']),2)

print("Variance of prediction error: ", var\_err\_pred)

# Variance error of Forecast

var\_err\_forecast = round(np.var(df\_avg\_h['e']),2)

print("Variance of Forecast error: ", var\_err\_forecast)

# ACF

acf(df\_avg['e'], 50, plot= True, title="ACF of Average Method Residuals")

# Prediction Q

q\_avg = q\_value(df\_avg['e'],50,len(df\_avg))

print(f"Q-Value: {q\_avg:.2f}")

qstar\_avg,pvalue\_avg=sm.stats.acorr\_ljungbox(df\_avg['e'],lags=[50])

# print(f"Q\*-Value of Residual Error: {qstar\_avg[0]:.2f}")

# if res\_q < qstar\_avg:

#   print("The residual is white as Q < Q\*")

# else:

#   print("The residual is not white as Q > Q\* ")

if pvalue\_avg > 0.05:

  print("The residual is white as p > 0.05")

else:

  print("The residual is not white as p < 0.05 ")

#%%

# Naive

# df\_niv = pd.DataFrame(data = {'yt':Y\_train})

df\_niv = pd.DataFrame(data = {'yt':y\_train})

df\_niv['y\_hat'] = naive\_forecast(np.arange(0,len(df\_niv.index)),df\_niv['yt'])

df\_niv['e'] = df\_niv['yt']-df\_niv['y\_hat']

df\_niv['e^2'] = round(df\_niv['e']\*\*2, 2)

# h - step

# df\_niv\_h = pd.DataFrame(data = {'yt+h':Y\_test})

df\_niv\_h = pd.DataFrame(data = {'yt+h':y\_test})

df\_niv\_h['y\_hat'] = df\_niv['yt'].iloc[-1]

df\_niv\_h['e'] = df\_niv\_h['yt+h']-df\_niv\_h['y\_hat']

df\_niv\_h['e^2'] = round(df\_niv\_h['e']\*\*2, 2)

plt.figure(figsize=(16,6))

plt.plot(df\_niv.index,df\_niv['yt'], label='Training Dataset', color='b')

plt.plot(df\_niv\_h.index,df\_niv\_h['yt+h'], label = 'Testing Dataset', color='g')

plt.plot(df\_niv\_h.index,df\_niv\_h['y\_hat'], label = 'Naive Method H-step prediction', color='r', linestyle='dashed')

# plt.xticks(df.index[::981])

plt.xticks(ticks=range(0,len(df))[::981], labels = df.index[::981])

plt.title('Naive Method & Forecast')

plt.xlabel('time')

plt.ylabel('Values')

plt.legend()

plt.show()

# MSE of prediction

MSE\_pred\_nv = round(np.mean(df\_niv['e^2']),2)

print("MSE of prediction: ", MSE\_pred\_nv)

# MSE of forecast

MSE\_forecast\_nv = round(np.mean(df\_niv\_h['e^2']),2)

print("MSE of forecast: ", MSE\_forecast\_nv)

# Variance error of prediction

var\_err\_pred\_nv = round(np.var(df\_niv['e']),2)

print("Variance of prediction error: ", var\_err\_pred\_nv)

# Variance error of Forecast

var\_err\_forecast\_nv = round(np.var(df\_niv\_h['e']),2)

print("Variance of Forecast error: ", var\_err\_forecast\_nv)

# ACF

acf(df\_niv['e'], 50, plot= True, title="ACF of Naive Method Residuals")

# Prediction Q

q\_niv = q\_value(df\_niv['e'],50,len(df\_niv))

print(f"Q-Value: {q\_niv:.2f}")

qstar\_niv,pvalue\_niv=sm.stats.acorr\_ljungbox(df\_niv['e'][1:],lags=[50])

# print(f"Q\*-Value of Residual Error: {qstar\_niv[0]:.2f}")

# if q\_niv < qstar\_niv:

#   print("The residual is white as Q < Q\*")

# else:

#   print("The residual is not white as Q > Q\* ")

if pvalue\_niv > 0.05:

  print("The residual is white as p > 0.05")

else:

  print("The residual is not white as p < 0.05 ")

#%%

# Drift

# df\_drft = pd.DataFrame(data = {'yt':Y\_train})

df\_drft = pd.DataFrame(data = {'yt':y\_train})

df\_drft['y\_hat'] = drift\_predict(np.arange(0,len(df\_drft.index)),df\_drft['yt'],1)

df\_drft['e'] = df\_drft['yt']-df\_drft['y\_hat']

df\_drft['e^2'] = round(df\_drft['e']\*\*2, 2)

# h - step

# df\_drft\_h = pd.DataFrame(data = {'yt+h':Y\_test})

df\_drft\_h = pd.DataFrame(data = {'yt+h':y\_test})

df\_drft\_h['y\_hat'] = drift\_forecast(Y\_train[0],Y\_train[-1],len(Y\_train),len(df\_drft\_h))

df\_drft\_h['e'] = df\_drft\_h['yt+h']-df\_drft\_h['y\_hat']

df\_drft\_h['e^2'] = round(df\_drft\_h['e']\*\*2, 2)

plt.figure(figsize=(16,6))

plt.plot(df\_drft.index,df\_drft['yt'], label='Training Dataset', color='b')

plt.plot(df\_drft\_h.index,df\_drft\_h['yt+h'], label = 'Testing Dataset', color='g')

plt.plot(df\_drft\_h.index,df\_drft\_h['y\_hat'], label = 'Drift Method H-step prediction', color='r', linestyle='dashed')

# plt.xticks(df.index[::981])

plt.xticks(ticks=range(0,len(df))[::981], labels = df.index[::981])

plt.title('Drift Method & Forecast')

plt.xlabel('time')

plt.ylabel('Values')

plt.legend()

plt.show()

# MSE of prediction

MSE\_pred\_df = round(np.mean(df\_drft['e^2']),2)

print("MSE of prediction: ", MSE\_pred\_df)

# MSE of forecast

MSE\_forecast\_df = round(np.mean(df\_drft\_h['e^2']),2)

print("MSE of forecast: ", MSE\_forecast\_df)

# Variance error of prediction

var\_err\_pred\_df = round(np.var(df\_drft['e']),2)

print("Variance of prediction error: ", var\_err\_pred\_df)

# Variance error of Forecast

var\_err\_forecast\_df = round(np.var(df\_drft\_h['e']),2)

print("Variance of Forecast error: ", var\_err\_forecast\_df)

# ACF

acf(df\_drft['e'], 50, plot= True, title="ACF of Drift Method Residuals")

# Prediction Q

q\_dft = q\_value(df\_drft['e'],50,len(df\_drft))

print(f"Q-Value: {q\_dft:.2f}")

qstar\_dft,pvalue\_dft=sm.stats.acorr\_ljungbox(df\_drft['e'][2:],lags=[50])

# print(f"Q\*-Value of Residual Error: {qstar\_dft[0]:.2f}")

# if q\_dft < qstar\_dft:

#   print("The residual is white as Q < Q\*")

# else:

#   print("The residual is not white as Q > Q\* ")

if pvalue\_dft > 0.05:

  print("The residual is white as p > 0.05")

else:

  print("The residual is not white as p < 0.05 ")

#%%

# Simple Exponential Smoothing (SES)

ses = ets.ExponentialSmoothing(y\_train, trend =None, damped\_trend = False, seasonal = None).fit()

ses\_pred = pd.DataFrame(ses.fittedvalues).set\_index(y\_train.index)

ses\_frcst = pd.DataFrame(ses.forecast(steps=len(y\_test))).set\_index(y\_test.index)

ses\_pred\_err = y\_train - ses\_pred[0].values

ses\_frcst\_err = y\_test - ses\_frcst[0].values

plt.figure(figsize=(16,6))

# plt.plot(Y\_train.index,Y\_train.values, label='Training Dataset', color='b')

plt.plot(y\_train.index,y\_train.values, label='Training Dataset', color='b')

# plt.plot(Y\_test.index,Y\_test, label = 'Testing Dataset', color='g')

plt.plot(y\_test.index,y\_test, label = 'Testing Dataset', color='g')

plt.plot(ses\_frcst.index,ses\_frcst[0].values, label = 'SES Method H-step prediction', color='r', linestyle='dashed')

# plt.xticks(df.index[::981])

plt.xticks(ticks=range(0,len(df))[::981], labels = df.index[::981])

plt.title('Simple Exponential Smoothing Method & Forecast')

plt.xlabel('Time')

plt.ylabel('Values')

plt.legend(loc='upper right')

plt.show()

# MSE of prediction

# MSE\_ses\_pred = mean\_squared\_error(Y\_train,ses\_pred)

MSE\_ses\_pred = mean\_squared\_error(y\_train,ses\_pred)

print(f"MSE of prediction: {MSE\_ses\_pred:.3f}")

# MSE of forecast

# MSE\_ses\_frcst = mean\_squared\_error(Y\_test, ses\_frcst)

MSE\_ses\_frcst = mean\_squared\_error(y\_test, ses\_frcst)

print(f"MSE of forecast: {MSE\_ses\_frcst:.3f}")

# Variance error of prediction

var\_err\_ses\_pred = round(np.var(ses\_pred\_err),2)

print(f"Variance of prediction error: {var\_err\_ses\_pred:.3f}")

# Variance error of Forecast

var\_err\_ses\_frcst = round(np.nanvar(ses\_frcst\_err),2)

print(f"Variance of Forecast error: {var\_err\_ses\_frcst:.3f}")

# ACF

acf(ses\_pred\_err, 50, plot= True, title="ACF of SES Method Residuals")

# Prediction Q

q\_ses = q\_value(ses\_pred\_err,50,len(ses\_pred\_err))

print(f"Q-Value: {q\_ses:.2f}")

qstar\_ses,pvalue\_ses=sm.stats.acorr\_ljungbox(ses\_pred\_err,lags=[50])

# print(f"Q\*-Value of Residual Error: {qstar\_ses[0]:.2f}")

# if q\_ses < qstar\_ses:

#   print("The residual is white as Q < Q\*")

# else:

#   print("The residual is not white as Q > Q\* ")

if pvalue\_ses > 0.05:

  print("The residual is white as p > 0.05")

else:

  print("The residual is not white as p < 0.05 ")

# %%

######## 12. Multiple Linear Regression

print('From the backward stepwise feature selection we got our Final Multiple Linear Regression model.')

# final\_model = sm.OLS(Y\_train,X\_train).fit()

print('Final Model: \n', final\_model.summary())

#%%

# Predictions

# pred=final\_model.predict(X\_train)

pred = final\_model.fittedvalues

#Residual error

res\_err=Y\_train-pred

#Forecasts

forecast=final\_model.predict(X\_test)

#Forecast error

fcst\_err=Y\_test-forecast

#%%

# Prediction Plot

plt.figure(figsize=(16,6))

Y\_train.plot(label='Train set')

plt.plot(pred.index,pred, label='Predicted')

plt.title('AvgAQI Prediction using OLS model')

plt.ylabel('AQI')

plt.xlabel('Time')

plt.grid()

plt.legend(loc='upper right')

plt.show()

# Forecast Plot

plt.figure(figsize=(16,6))

Y\_test.plot(label='Test set')

plt.plot(forecast.index,forecast, label='Forecast')

plt.title('AvgAQI Forecast using OLS model')

plt.ylabel('AQI')

plt.xlabel('Time')

plt.grid()

plt.legend(loc='upper right')

plt.show()

# together

plt.figure(figsize=(16,6))

Y\_train.plot(label='Train set', color = 'b')

plt.plot(pred.index,pred, label='Predicted', color = 'deepskyblue')

plt.plot(Y\_test, label='Test', color = 'maroon')

plt.plot(forecast.index,forecast, label='Forecast', color = 'forestgreen')

plt.xticks(ticks=range(0,len(df))[::981], labels = df.index[::981])

plt.title('Average AQI Predictions using OLS model')

plt.ylabel('AQI')

plt.xlabel('Time')

plt.grid()

plt.legend(loc='upper right')

plt.show()

#%%

# Hypothesis Testing

# T test

print(f"\nT Test p-values: \n{final\_model.pvalues}")

print("As the p-values of the T test is less than the significant level\

 alpha = 0.05, we reject the null hypothesis and conclude that there is a\

 statistically significant relationship between the predictor variable and the response variable.")

# F test

print(f"\nF Test p-value: {final\_model.f\_pvalue}")

print("As the p-value of the F test is less than the significant level\

 alpha = 0.05, we can reject the null-hypothesis and conclude that final\

 model provides a better fit than the intercept-only model.")

#%%

# AIC, BIC, RMSE, R^2, Adj R^2

print(f"AIC: {final\_model.aic:.2f}")

print(f"BIC: {final\_model.bic:.2f}")

print(f"RMSE:-")

print(f'\tResidual: {mean\_squared\_error(Y\_train, pred,squared=False):.3f}')

print(f'\tFoercast: {mean\_squared\_error(Y\_test, forecast,squared=False):.3f}')

print(f"R-Squared Value: {(final\_model.rsquared\*100):.2f}%")

print(f"Adj-R Squared Value: {(final\_model.rsquared\_adj\*100):.2f}%")

print(f"\nOverall the final model's performance is pretty good. In this final model, {(final\_model.rsquared\_adj\*100):.2f}%\

  variation in dependandant variable 'avgAQI' can be explained by the independant variables.\

 The RMSE values are low as well.")

#%%

# ACF of ERRORS

acf(res\_err, 50, plot = True, title='ACF of Residual Error')

# acf(fcst\_err, 50, plot = True, title='ACF of Forecast Error')

# Model Performance

# MSE

print(f'\nMSE of Residual Error: {mean\_squared\_error(Y\_train, pred):.3f}')

print(f'\nMSE of Foercast Error: {mean\_squared\_error(Y\_test, forecast):.3f}')

# Q

q\_res\_ols = q\_value(res\_err, 50, len(Y\_train))

print(f'\nQ-Value of Residual Error: {q\_res\_ols:.3f}')

# Prediction Q

# q\_dft = q\_value(df\_drft['e'],50,len(df\_drft))

# print(f"Q-Value: {q\_dft:.2f}")

qstar\_ols,pvalue\_ols=sm.stats.acorr\_ljungbox(res\_err,lags=[50])

# print(f"Q\*-Value of Residual Error: {qstar\_ols[0]:.2f}")

# if q\_res\_ols < qstar\_ols:

#   print("The residual is white as Q < Q\*")

# else:

#   print("The residual is not white as Q > Q\* ")

if pvalue\_ols > 0.05:

  print("The residual is white as p > 0.05")

else:

  print("The residual is not white as p < 0.05 ")

# Mean Variance of residual

print(f'Mean of residuals: {np.nanmean(res\_err):.2f}')

print(f'Variance of residuals: {np.var(res\_err):.2f}')

# %%

######## 13. ARMA, ARIMA, SARIMA

# ARMA

# finding order

# re = acf(Y\_train, 50, plot=False)

# Cal\_GPAC2(re[50:],7,7)

ACF\_PACF\_Plot(Y\_train, 50)

re = sm.tsa.stattools.acf(Y\_train.values, nlags = 50)

Cal\_GPAC(re[:],8,8)

print('Observing the patterns ARMA(1,0) and ARMA(3,1) can be selected for farther analysis.')

#%%

# ARMA(1,0)

na = 1

nb = 0

arma10 = sm.tsa.ARMA(Y\_train, (na,nb)).fit(trend='nc', disp=0)

# coefficients

for i in range(na):

  print(f"The AR coefficient a{i} is: {-arma10.params[i]:.2f}")

for i in range(nb):

  print(f"The MA coefficient b{i} is {arma10.params[i+na]:.2f}")

print(arma10.summary())

# confidance interval

print('Confidance Interval: ')

print(arma10.conf\_int())

print('As the interval does not contain zero in it, it is statistically important.')

# Prediction

arma10\_pred = arma10.fittedvalues

arma10\_residuals = Y\_train - arma10\_pred

# Forecast

arma10\_for = arma10.predict(start=len(Y\_train), end = len(df)-1)

arma10\_ferr = pd.DataFrame(Y\_test.values - arma10\_for).set\_index(Y\_test.index)

arma10\_ferr=pd.Series(np.array(arma10\_ferr[0]),index = pd.date\_range(Y\_test.index[0],periods= len(Y\_test)))

# ACF of Residuals

acf(arma10\_residuals, 50, plot=True, title= "ACF of ARMA(1,0) Residuals")

acf(arma10\_ferr, 50, plot=True, title= "ACF of ARMA(1,0) Forecast Errors")

# MSE

arma10\_p\_mse = mean\_squared\_error(Y\_train, arma10\_pred)

print(f"MSE of Residuals: {arma10\_p\_mse:.2f}")

arma10\_f\_mse = mean\_squared\_error(Y\_test, arma10\_ferr)

print(f"MSE of Forecast Error: {arma10\_f\_mse:.2f}")

# Q-Value

arma10\_q = q\_value(arma10\_residuals, 50, len(Y\_train))

print(f"Q-Value: {arma10\_q:.2f}")

# Covariance Matrix

print('Covariance Matrix: \n', arma10.cov\_params())

#%%

# ARMA(3,1)

na = 3

nb = 1

arma31 = sm.tsa.ARMA(Y\_train, (na,nb)).fit(trend='nc', disp=0)

# coefficients

for i in range(na):

  print(f"The AR coefficient a{i} is: {-arma31.params[i]:.2f}")

for i in range(nb):

  print(f"The MA coefficient b{i} is {arma31.params[i+na]:.2f}")

print(arma31.summary())

# confidance interval

print('Confidance Interval: ')

print(arma31.conf\_int())

print('As the interval does not contain zero in it, it is statistically important.')

# Prediction

arma31\_pred = arma31.fittedvalues

arma31\_residuals = Y\_train - arma31\_pred

# Forecast

arma31\_for = arma31.predict(start=len(Y\_train), end = len(df)-1)

arma31\_ferr = pd.DataFrame(Y\_test.values - arma31\_for).set\_index(Y\_test.index)

arma31\_ferr=pd.Series(np.array(arma31\_ferr[0]),index = pd.date\_range(Y\_test.index[0],periods= len(Y\_test)))

# ACF of Residuals

acf(arma31\_residuals, 50, plot=True, title= "ACF of ARMA(3,1) Residuals")

acf(arma31\_ferr, 50, plot=True, title= "ACF of ARMA(3,1) Forecast Errors")

# MSE

arma31\_p\_mse = mean\_squared\_error(Y\_train, arma31\_pred)

print(f"MSE of Residuals: {arma31\_p\_mse:.2f}")

arma31\_f\_mse = mean\_squared\_error(Y\_test, arma31\_for)

print(f"MSE of Forecast Error: {arma31\_f\_mse:.2f}")

# Q-Value

arma31\_q = q\_value(arma31\_residuals, 50, len(Y\_train))

print(f"Q-Value: {arma31\_q:.2f}")

# Covariance Matrix

print('Covariance Matrix: \n', arma31.cov\_params())

print("\nAmong ARMA(1,0) model ARMA(3,1), ARMA(1.0) has lower Q valure but ARMA(3,1) is better at forecasting.")

#%%

# ARIMA

# ARIMA(1,1,0)

na = 1

d = 1

nb = 0

arima110 = sm.tsa.ARIMA(endog=Y\_train, order=(na,d,nb)).fit()

# coefficients

for i in range(1,na+1):

  print(f"The AR coefficient a{i} is: {-arima110.params[i]:.2f}")

for i in range(1,nb+1):

  print(f"The MA coefficient b{i} is {arima110.params[i+na]:.2f}")

print(arima110.summary())

# confidance interval

print('Confidance Interval: ')

print(arima110.conf\_int())

print('As the interval does not contain zero in it, it is statistically important.')

# Prediction

arima110\_pred = arima110.fittedvalues

arima110\_predict = inverse\_diff(Y\_train.values,np.array(arima110\_pred),1)

arima110\_residuals = Y\_train[1:] - arima110\_predict

# Forecast

arima110\_for = arima110.predict(start=len(Y\_train), end = len(df)-1)

arima110\_for = inverse\_diff(Y\_test.values,np.array(arima110\_for),1)

arima110\_ferr = pd.DataFrame(Y\_test.values[:-1] - arima110\_for).set\_index(Y\_test.index[:-1])

arima110\_ferr=pd.Series(np.array(arima110\_ferr[0]),index = pd.date\_range(Y\_test.index[0],periods= len(Y\_test)-1))

# ACF of Residuals

acf(arima110\_residuals, 50, plot=True, title= "ACF of ARIMA(1,1,0) Residuals")

acf(arima110\_ferr, 50, plot=True, title= "ACF of ARIMA(1,1,0) Forecast Errors")

# # MSE

arima110\_p\_mse = mean\_squared\_error(Y\_train[:-1], arima110\_predict)

print(f"MSE of Residuals: {arima110\_p\_mse:.2f}")

arima110\_f\_mse = mean\_squared\_error(Y\_test[:-1], arima110\_for)

print(f"MSE of Forecast Error: {arima110\_f\_mse:.2f}")

# Q-Value

arima110\_q = q\_value(arima110\_residuals, 50, len(Y\_train))

print(f"Q-Value: {arima110\_q:.2f}")

# Covariance Matrix

print('Covariance Matrix: \n', arima110.cov\_params())

#%%

# ARIMA(3,1,1)

na = 3

d = 1

nb = 1

arima311 = sm.tsa.ARIMA(endog=Y\_train, order=(na,d,nb)).fit()

# coefficients

for i in range(1,na+1):

  print(f"The AR coefficient a{i} is: {-arima311.params[i]:.2f}")

for i in range(1,nb+1):

  print(f"The MA coefficient b{i} is {arima311.params[i+na]:.2f}")

print(arima311.summary())

# confidance interval

print('Confidance Interval: ')

print(arima311.conf\_int())

print("\nHere interval of AR coefficient a2 contains zero, it is statistically not important in this model.")

# Prediction

arima311\_pred = arima311.fittedvalues

arima311\_predict = inverse\_diff(Y\_train.values,np.array(arima311\_pred),1)

arima311\_residuals = Y\_train[1:] - arima311\_predict

# Forecast

arima311\_for = arima311.predict(start=len(Y\_train), end = len(df)-1)

arima311\_for = inverse\_diff(Y\_test.values,np.array(arima311\_for),1)

arima311\_ferr = pd.DataFrame(Y\_test.values[:-1] - arima311\_for).set\_index(Y\_test.index[:-1])

arima311\_ferr=pd.Series(np.array(arima311\_ferr[0]),index = pd.date\_range(Y\_test.index[0],periods= len(Y\_test)-1))

# ACF of Residuals

acf(arima311\_residuals, 50, plot=True, title= "ACF of ARIMA(3,1,1) Residuals")

acf(arima311\_ferr, 50, plot=True, title= "ACF of ARIMA(3,1,1) Forecast Errors")

# # MSE

arima311\_p\_mse = mean\_squared\_error(Y\_train[:-1], arima311\_predict)

print(f"MSE of Residuals: {arima311\_p\_mse:.2f}")

arima311\_f\_mse = mean\_squared\_error(Y\_test[:-1], arima311\_for)

print(f"MSE of Forecast Error: {arima311\_f\_mse:.2f}")

# Q-Value

arima311\_q = q\_value(arima311\_residuals, 50, len(Y\_train))

print(f"Q-Value: {arima311\_q:.2f}")

# Covariance Matrix

print('Covariance Matrix: \n', arima311.cov\_params())

print("\nAmong ARIMA(1,1,0) model ARMA(3,1,1), ARIMA(3,1,1) has lower Q valure but ARMA(1,1,0) is better at forecasting.")

#%%

# SARIMA

sarima= sm.tsa.statespace.SARIMAX(Y\_train,order=(3,0,1),seasonal\_order=(0,2,0,7),

                                    enforce\_stationarity=False,

                                    enforce\_invertibility=False)

sarima\_results=sarima.fit()

print(sarima\_results.summary())

# Prediction

sarima\_pred\_ = sarima\_results.get\_prediction(start=0, end=len(Y\_train), dynamic=False)

sarima\_pred = sarima\_pred\_.predicted\_mean

sarima\_residuals = Y\_train - sarima\_pred.values[1:]

# Forecast

sarima\_fore = sarima\_results.predict(start=0, end =len(Y\_test))

sarima\_ferr =Y\_test - sarima\_fore.values[1:]

# ACF

acf(sarima\_residuals,50,plot=True,title="ACF of SARIMA Residuals")

acf(sarima\_ferr, 50, plot=True, title="ACF of SARIMA Forecast Errors")

# # MSE

sarima\_pred\_mse = mean\_squared\_error(Y\_train, sarima\_pred[1:])

print(f"MSE of Residuals: {sarima\_pred\_mse:.2f}")

sarima\_fore\_mse = mean\_squared\_error(Y\_test, sarima\_fore[1:])

print(f"MSE of Forecast Error: {sarima\_fore\_mse:.2f}")

# Q-Value

sarima\_q = q\_value(sarima\_residuals, 50, len(Y\_train))

print(f"Q-Value: {sarima\_q:.2f}")

# Covariance Matrix

print('Covariance Matrix: \n', sarima\_results.cov\_params())

# %%

######## 14. LMA

# AR(3) MA(1)

SSE,cov,teta\_hat,var = LMA(Y\_train,3,1)

print("Estimated ARMA(3,1) model parameters using the LM Algorithm are:- \n", teta\_hat)

print(f"\nStandard deviation of parameter estimates: {np.std(teta\_hat):.2f}")

conf\_int(cov, teta\_hat, 3, 1)

print('\nThe coefficients are statistically important as the interval does not include 0.')

#%%

# coefficents from ARMA(3,1)

print(f"{-arma31.params[:3].values} {arma31.params[-1]}")

# %%

######## 15. Diagnostic Analysis

# confidance intervals

print("####  Confidance Intervals:  ####\n")

print("\nOLS:- \n", final\_model.conf\_int())

print("\nARMA(1,0):-\n", arma10.conf\_int())

print("\nARMA(3,1):-\n", arma31.conf\_int())

print("\nARIMA(1,1,0):-\n", arima110.conf\_int())

print("\nARIMA(3,1,1):-\n", arima311.conf\_int())

print("\nSARIMA:-\n", sarima\_results.conf\_int())

#%%

# zero/pole cancellation

print("####  Zero/Pole cancellations:  ####\n")

# print("\nOLS:- \n", zero\_pole(final\_model.params, na))

print("\nARMA(1,0):-\n")

zero\_pole(arma10.params, 1)

print("\nARMA(3,1):-\n")

zero\_pole(arma31.params, 3)

print("\nARIMA(1,1,0):-\n")

zero\_pole(arima110.params, 1)

print("\nARIMA(3,1,1):-\n")

zero\_pole(arima311.params, 3)

print("\nSARIMA:-\n")

zero\_pole(sarima\_results.params[:-1], 3)

print("None of the models have zero pole cancellations.")

#%%

# chi sq test

def chi\_sq(lags,na,nb, q, alpha=0.01):

  from scipy.stats import chi2

  DOF= lags - na - nb

  chi\_critical = chi2.ppf(1-alpha,DOF)

  print(f"\tQ-Value: {q:.2f}\n\tChi Critical Value: {chi\_critical:.2f}")

  if q < chi\_critical:

    print('The residual is white')

  else:

      print('The residual is not white')

  return None

print("\nARMA(1,0):-")

chi\_sq(50,1,0,arma10\_q)

print("\nARMA(3,1):-")

chi\_sq(50,3,1,arma31\_q)

print("\nARIMA(1,1,0):-\n")

chi\_sq(50,1,0,arima110\_q)

print("\nARIMA(3,1,1):-\n")

chi\_sq(50,3,0,arima311\_q)

print("\nSARIMA:-\n")

chi\_sq(50,3,0,sarima\_q)

#%%

# Q-Values

print("Q-Values of Residual Error:")

print(f'\tOLS: {q\_res\_ols:.3f}')

print(f'\tARMA(1,0): {arma10\_q:.3f}')

print(f'\tARMA(3,1): {arma31\_q:.3f}')

print(f'\tARIMA(1,1,0): {arima110\_q:.3f}')

print(f'\tARIMA(3,1,1): {arima311\_q:.3f}')

print(f'\tSARIMA(3,0,1) x (0,2,0,7) : {sarima\_q:.3f}')

#%%

# variance of residual error

print("\nVariance of Residual Errors: ")

print(f"\tOLS: {np.var(res\_err):.2f}")

print(f"\tARMA(1,0): {np.var(arma10\_residuals):.2f} ")

print(f"\tARMA(3,1): {np.var(arma31\_residuals):.2f} ")

print(f"\tARIMA(1,1,0): {np.var(arima110\_residuals):.2f} ")

print(f"\tARIMA(3,1,1): {np.var(arima311\_residuals):.2f} ")

print(f"\tSARIMA: {np.var(sarima\_residuals):.2f} ")

# variance of forecast error

print("\nVariance of Forecast Errors: ")

print(f"\tOLS: {np.var(fcst\_err):.2f}")

print(f"\tARMA(1,0): {np.var(arma10\_ferr):.2f} ")

print(f"\tARMA(3,1): {np.var(arma31\_ferr):.2f} ")

print(f"\tARIMA(1,1,0): {np.var(arima110\_ferr):.2f} ")

print(f"\tARIMA(3,1,1): {np.var(arima311\_ferr):.2f} ")

print(f"\tSARIMA: {np.var(sarima\_ferr):.2f} ")

# MSE

# MSE of residuals

print("\nMSE of Residuals: ")

print(f"\tOLS: {np.mean(np.square(res\_err)):.2f}")

print(f"\tARMA(1,0): {np.mean(np.square(arma10\_residuals)):.2f} ")

print(f"\tARMA(3,1): {np.mean(np.square(arma31\_residuals)):.2f} ")

print(f"\tARIMA(1,1,0): {np.mean(np.square(arima110\_residuals)):.2f} ")

print(f"\tARIMA(3,1,1): {np.mean(np.square(arima311\_residuals)):.2f} ")

print(f"\tSARIMA: {np.mean(np.square(sarima\_residuals)):.2f} ")

# MSE of forecasts

print("\nMSE of Forecasts: ")

print(f"\tOLS: {np.mean(np.square(fcst\_err)):.2f}")

print(f"\tARMA(1,0): {np.mean(np.square(arma10\_ferr)):.2f} ")

print(f"\tARMA(3,1): {np.mean(np.square(arma31\_ferr)):.2f} ")

print(f"\tARIMA(1,1,0): {np.nanmean(np.square(arima110\_ferr)):.2f} ")

print(f"\tARIMA(3,1,1): {np.mean(np.square(arima311\_ferr)):.2f} ")

print(f"\tSARIMA: {np.mean(np.square(sarima\_ferr)):.2f} ")

#%%

# ratio of test set by forecast

print("\nRatio of test set variance by forecast variance: ")

print(f"\tOLS: {np.var(Y\_test)/np.var(forecast):.2f}")

print(f"\tARMA(1,0): {np.var(Y\_test)/np.var(arma10\_for):.2f} ")

print(f"\tARMA(3,1): {np.var(Y\_test)/np.var(arma31\_for):.2f} ")

print(f"\tARIMA(1,1,0): {np.var(Y\_test)/np.var(arima110\_for):.2f} ")

print(f"\tARIMA(3,1,1): {np.var(Y\_test)/np.var(arima311\_for):.2f} ")

print(f"\tSARIMA: {np.var(Y\_test)/np.var(sarima\_fore):.2f} ")

# %%

######## 17. Final model Selection

print("The final model is ARIMA(3,1,1)")

# %%

######## 18. Forecast Function

y\_train\_diff = y\_train.diff(1).dropna()

y\_hat = []

for i in range(1,len(y\_train\_diff)):

    if i==1:

      y\_hat.append((0.69\*y\_train\_diff[i-1]) -(0.97 \*y\_train\_diff[i-1]))

    elif i == 2:

      y\_hat.append((0.69\*y\_train\_diff[i-1]) - (0.09 \* y\_train\_diff[i-2]) - (0.97\*(y\_train\_diff[i-1] - y\_hat[0])))

    else:

      y\_hat.append((0.69\*y\_train\_diff[i-1]) - (0.09\*y\_train\_diff[i-2]) -(0.01\*y\_train\_diff[i-3]) - (0.97\*(y\_train\_diff[i-1] - y\_hat[-1])) )

y\_hat\_inv\_diff = inverse\_diff(y\_train.values,np.array(y\_hat),1)

#%%

plt.plot(y\_train,label='True Data (Train set)')

plt.plot(y\_hat\_inv\_diff,label='Fitted Data (1-step prediction)')

plt.title('True data vs. One step prediction data')

plt.xticks(ticks=range(0,len(y\_train\_diff))[::697], labels = y\_train\_diff.index[::697], rotation = 90)

plt.suptitle("ARIMA(3,1,1): y(t) – 0.69 y(t-1) + 0.09 y(t-2) + 0.01 y(t-3) = e(t) – 0.97 e(t-1)", fontsize=22)

plt.legend(loc='upper right', bbox\_to\_anchor=(1.01,1))

plt.xlabel('Time')

plt.ylabel('Average AQI')

plt.show()

# %%

######## 19. h-step Prediction

def h\_step(h,y\_train, y\_test, y):

  y\_hat = []

  for i in range(len(y\_train),len(y)):

    if i==len(y\_train):

      y\_hat.append((0.69\*y[i-h]) -(0.97 \*y[i-h]))

    elif i == (len(y\_train)+1):

      y\_hat.append((0.69\*y[i-h]) - (0.09 \* y[i-h-1]) - (0.97\*(y[i-h] - y\_hat[0])))

    else:

      y\_hat.append((0.69\*y[i-h]) - (0.09\*y[i-h-1]) -(0.01\*y[i-h-2]) - (0.97\*(y[i-h] - y\_hat[-1])) )

  return y\_hat

h=30

arima311\_hstep = h\_step(h,y\_train, y\_test,df['avgAQI'].diff(1).dropna())

arima311\_hstep\_inv\_diff = inverse\_diff(y\_test.values,np.array(arima311\_hstep),1)

plt.plot(y\_test,label='True Data (Test set)')

plt.plot(arima311\_hstep\_inv\_diff,label='Fitted Data (h-step prediction)')

plt.title(f'True data vs. {h}-step prediction data')

plt.xticks(ticks=range(0,len(y\_test))[::174], labels = y\_test.index[::174], rotation = 90)

plt.suptitle("ARIMA(3,1,1): y(t) – 0.69 y(t-1) + 0.09 y(t-2) + 0.01 y(t-3) = e(t) – 0.97 e(t-1)", fontsize=22)

plt.legend(loc='upper right', bbox\_to\_anchor=(1.01,1))

plt.xlabel('Time')

plt.ylabel('Average AQI')

plt.show()

#%%

# variance of test set vs vaarience of predicted set

print(f"Variance of test set: {np.var(y\_test):.2f}")

print(f"Variance of predicted set: {np.var(arima311\_hstep\_inv\_diff):.2f}")

print(f"Ratio: {np.var(y\_test)/np.var(arima311\_hstep\_inv\_diff):.2f}")

# %%

######## 20. Summary  and Conclusion

# Among the ARMA and ARIMA models ARIMA model with AR (3), MA (1) with differencing order 1,

#  performed better than the other by considering the lowest q-value of residuals, MSE, ratio

#  of variance of test set vs forecasted set. From this model I farther generated the model

#  equation and built 1-step and multi-step prediction functions and the model is exceptionally

#  good at predict next week as well as next month’s average AQI. Overall, the models did not

#  have white q-value of residuals possibly because of the nature of the dataset. More advanced

#  machine learning technique of forecasting like LSTM, XGboost etc. can achieve that.

# %%

Source code of toolbox

#%%

import matplotlib.pyplot as plt

import pandas as pd

from statsmodels.tsa.stattools import adfuller

import os

import numpy as np

from statsmodels.tsa.stattools import kpss

import math

import seaborn as sns

from scipy.signal import dlsim

from statsmodels.graphics.tsaplots import plot\_acf,plot\_pacf

import statsmodels.api as sm

import warnings

warnings.filterwarnings('ignore')

# %%

def Cal\_rolling\_mean\_var(column):

    """

    To calculate and plot rolling mean and rolling variance of a column

        Parameter:

            coulmn (list): list of coulmn values

        Variables:

            rolling\_mean (list): a list containing the rolling means

            rolling\_var (list): a list containing the rolling variances

        returns:

            None

    """

    rolling\_mean = list()

    rolling\_var = list()

    for i in range(1,len(column)+1):

        mean=np.mean(column[:i])

        rolling\_mean.append(mean)

        var=np.var(column[:i])

        rolling\_var.append(var)

    fig, axes = plt.subplots(nrows=2, ncols=1, figsize=(8, 4))

    axes[0].plot(rolling\_mean, color='r')

    axes[0].set\_title('\nMean')

    axes[0].set(xlabel='Time', ylabel='Mean USD($)')

    axes[1].plot(rolling\_var, color='b')

    axes[1].set\_title('Variance')

    axes[1].set(xlabel='Time', ylabel='Variance USD($)')

    plt.tight\_layout()

    plt.suptitle(f'Dependant Variable vs Time')

    plt.show()

    print(f'Final rolling mean: {rolling\_mean[-1]:.4f}')

    print(f'Final rolling variance: {rolling\_var[-1]:.4f}')

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))

# print('\nThe null hypothesis of the ADF is that there is a unit root, with\

#  the alternative that there is no unit root.The p-value below a threshold\

#  (1% or 5%) suggests we reject the null hypothesis (stationary) and a p-value\

#  above the threshold suggests we fail to reject the null hypothesis (non-stationary).\n')

def kpss\_test(timeseries):

    print ('Results of KPSS Test:')

    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)

# print('The null and alternate hypothesis for the KPSS test are opposite that of the ADF test.\

#  The p-value below a threshold (1% or 5%) suggests we reject the null hypothesis (non-stationary)\

#  and a p-value above the threshold suggests we fail to reject the null hypothesis (stationary).')

def pearson\_correlation\_coeff(x, y):

    df = pd.DataFrame({'x': x, 'y': y}, columns=['x', 'y'])

    x\_bar = df['x'].mean()

    y\_bar = df['y'].mean()

    x\_xbar = df['x'] - x\_bar

    y\_ybar = df['y'] - y\_bar

    x\_xbar\_sq = x\_xbar \*\* 2

    y\_ybar\_sq = y\_ybar \*\* 2

    corr =(sum(x\_xbar \* y\_ybar))/(math.sqrt(sum(x\_xbar\_sq) \* sum(y\_ybar\_sq)))

    return corr

# acf (R\_of\_y, acf\_plot, acf)

def R\_of\_y(y: list, tau: int):

    """[summary]

    to calculate r(y)

    Args:

        y (list): [list of values to calculate acf]

        tau (int): [number of lags to calculate]

    Returns:

        [acf]: [acf of yt value on lag = tau ]

    """

    # y = y.astype(float)

    y\_bar = np.nanmean(y)

    numerator\_0 = []

    numerator\_1 = []

    denominator = []

    for i, n in enumerate(y):

        denominator.append((n - y\_bar)\*\*2)

        if i >= abs(tau):

            numerator\_0.append(n-y\_bar)

            if tau < 0:

                numerator\_1.append(y[i+tau]-y\_bar)

            else:

                numerator\_1.append(y[i-tau]-y\_bar)

    denominator=np.nan\_to\_num(denominator)

    numerator\_0=np.nan\_to\_num(numerator\_0)

    numerator\_1=np.nan\_to\_num(numerator\_1)

    acf = np.dot(numerator\_0, numerator\_1)/np.nansum(denominator)

    return acf

def acf\_plot(acf, tau, y, title='Autocorrelation of White Noise'):

    """ to plot acf

    Args:

        acf (list): [list of calculated acf values]

        tau (list): [range of tau]

        y (list): [original sample]

        title (string): [Title for the plot; default -> 'Autocorrelation of White Noise']

    Returns:

        NONE

    """

    fig, ax = plt.subplots(figsize=(12,6))

    markerline, stemline, baseline = plt.stem(acf, markerfmt='C3o', basefmt='C0-')

    plt.setp(markerline, markersize = 6)

    plt.ylabel('Magnitute')

    plt.xlabel('Lags')

    if max(tau) <= 10:

        plt.xticks(ticks=range(0,len(acf))[::1], labels = tau[::1])

    if max(tau) > 10 and max(tau) < 100:

        plt.xticks(ticks=range(0,len(acf))[::5], labels = tau[::5])

    elif max(tau) >= 100 and max(tau) < 200:

        plt.xticks(ticks=range(0,len(acf))[::20], labels = tau[::20])

    elif max(tau) >= 200 and max(tau) < 600:

        plt.xticks(ticks=range(0,len(acf))[::50], labels = tau[::50])

    else:

        plt.xticks(ticks=range(0,len(acf))[::200], labels = tau[::200])

    plt.title(f'{title}')

    # ax.fill\_between(range(0,len(acf)),confint[0],confint[1], alpha=0.25)

    m = 1.96/np.sqrt(len(y))

    plt.axhspan(ymin=-m,ymax=m,alpha=0.2,color='b')

    plt.tight\_layout()

    plt.show()

def acf(y: list, taus: int, plot: bool, title='Autocorrelation of White Noise'):

    """ to calculated all the acf values

    Args:

        y (list): [list of numbers to calculate acf]

        taus (list): [lags range]

        plot (bool): [to call plot functions]

        title (string): [Title for the plot; default -> 'Autocorrelation of White Noise']

    Returns:

        list: [list of acf values]

    """

    if taus > 0:

        taus = list(range(-taus,taus+1))

    else:

        taus = list(range(taus, abs(taus)+1))

    acf\_list = []

    for t in taus:

        acf\_list.append(R\_of\_y(y,t))

    if taus[-1] <0:

        acf\_list = acf\_list[::-1]

    if plot == True:

        acf\_plot(acf\_list,taus,y, title)

    else:

        return acf\_list

def Cal\_GPAC(acf: list, len\_j: int, len\_k: int):

    ''' [Calculate and Plot GPAC Table]

    Args:

        acf (list): list of acf values

        len\_j (int): number of rows of the GPAC table

        len\_k (int): number of columns of the GPAC table

    Returns:

        NONE

    '''

    len\_k = len\_k + 1

    gpac = np.empty(shape=(len\_j, len\_k))

    for k in range(1, len\_k):

        num = np.empty(shape=(k, k))

        den = np.empty(shape=(k, k))

        for j in range(0, len\_j):

            for row in range(0, k):

                for col in range(0, k):

                    if col < k - 1:

                        num[row][col] = acf[np.abs(j+(row-col))]

                        den[row][col] = acf[np.abs(j+(row-col))]

                    else:

                        num[row][col] = acf[np.abs(j+row+1)]

                        den[row][col] = acf[np.abs(j+(row-col))]

            num\_determinant = round(np.linalg.det(num),5)

            denom\_determinant = round(np.linalg.det(den),5)

            if denom\_determinant == 0:

                gpac[j][k] = float('inf')

            else:

                gpac[j][k] = round(num\_determinant/denom\_determinant,3)

    gpac = pd.DataFrame(gpac[:, 1:])

    gpac.columns = [i for i in range(1, len\_k)]

    print("GPAC TABLE: \n",gpac)

    plt.figure(figsize=(8,6))

    sns.heatmap(gpac, annot=True, fmt=".3f")

    plt.xlabel('k')

    plt.ylabel('j')

    plt.title('Generalized Partial Autocorrelation (GPAC) Table')

    plt.show()

def rolling\_mean(y:list):

    mean=[]

    s = pd.Series(y)

    for i in range(len(s)):

        mean.append(np.mean( s.head(i) ))

    return mean

def rolling\_variance(y:list):

    var=[]

    s = pd.Series(y)

    for i in range(len(s)):

        var.append(np.var( s.head(i) ))

    return var

def q\_value(y: list,lags: int, t:int):

    """ to calculate q value from a given list of residuals

    Args:

        y (list): list of residuals to calculate acf

        lags (list): number of lags

        t (int): length of trainset

    Returns:

        calculated q value

    """

    r=acf(y,lags, plot= False)

    rk=np.square(r[lags+2:])

    return t\*(np.sum(rk))

# Average Prediction

def avg\_pred(t: list, yt: list):

    """

    to calculate average predictions

    Args:

        t (list): [list of times]

        yt (list): [list of y values]

    Returns:

        [forecast]: [list of forecasts]

    """

    forecast = []

    for i,v in zip(t, yt):

        if i == 0:

            forecast.append(np.nan)

        else:

            forecast.append(round(np.nanmean(yt[:i]),2))

    return forecast

# for h step -> yhat = average of trainset yt[1:]

# Naive

def naive\_forecast(t: list, yt: list):

    """

    to calculate naive forecast

    Args:

        t (list): [list of times]

        yt (list): [list of y values]

    Returns:

        [forecast]: [list of forecasts]

    """

    forecast = []

    forecast.append(np.nan)

    for i,v in zip(t, yt):

        if i >= 0:

            forecast.append(yt[i])

    forecast.pop(-1)

    return forecast

# for h step -> yhat = last trainset yt

# Drift

def drift\_predict(t: list, yt: list, h):

    """

    to calculate drift predict

    Args:

        t (list): [list of times]

        yt (list): [list of y values]

        h (int): [step]

    Returns:

        [predicts]: [list of predicts]

    """

    predicts = []

    for i,v in zip(t, yt):

        if i == 0:

            predicts.append(np.nan)

        elif i == 1:

            predicts.append(np.nan)

        else:

            predicts.append(round(yt[i-1]+((h\*(yt[i-1]-yt[0]))/(i-1)),2))

    return predicts

def drift\_forecast(y\_begin, y\_end, t, h):

    """

    to calculate drift forecast

    Args:

        y\_begin (number): [1st value of yt series]

        y\_end (number): [last value of yt series]

        t (int): [time of y\_end]

        h (int): [lag of prediction]

    Returns:

        [forecast]: [list of forecasts]

    """

    forecast = []

    for i in range(1,h+1):

        forecast.append(round(y\_end+((i\*(y\_end-y\_begin))/(t-1)),2))

    return forecast

# Simple Exponential Smoothing

def ses\_predict(t, yt, alpha, initial):

    """

    to calculate Simple Exponential Smoothing forecast

    Args:

        alpha (float): [damping factor -> 0 ≤ alpha ≤ 1]

        initial (number): [1st value of yt series or initial condition]

        t (list): [list of times]

        yt (list): [list of values at t times]

    Returns:

        [predict]: [list of predictions]

    """

    predict = []

    for t, v in zip(t,yt):

        if t == 0:

            predict.insert(t,initial)

        elif t > 0:

            y\_hat = predict[-1]

            predict.insert(t,round(alpha\*yt[t-1] + (1-alpha)\* y\_hat,2))

    return predict

# for h step -> yhat = predict using last trainset row

def moving\_avg(y):

    m = int(input("Enter the value of m:"))

    result = pd.DataFrame()

    def calculate\_mavg(m,y):

        m\_avg\_list = pd.DataFrame(columns=['t', f'{m}MA'])

        k = int(np.floor((m - 1) / 2))

        for t in range(k, (len(y) - k)):

            s = 0

            for j in range(-k, k + 1):

                s = s + y[t + j]

            m\_avg = round(s / m, 2)

            m\_avg\_list.loc[len(m\_avg\_list)] = [t, m\_avg]

        return m\_avg\_list

    def even\_mavg(y,m):

        cumsum = np.cumsum(np.insert(y, 0, 0))

        return (cumsum[m:] - cumsum[:-m]) / float(m)

    if m < 0:

        print("Invalid value of m")

    elif m == 1 | m == 2:

        print('Not acceptable m value. Please enter more than 2.')

    elif m%2 != 0:

        result = calculate\_mavg(m,y)

        # print(result)

    if m % 2 == 0:

        mv1 = even\_mavg(y,m)

        m2 = int(input("Enter the value of second order m:"))

        mv2 = even\_mavg(mv1,m2)

        tstart=((m-1)/2+(m2-1)/2)+1

        result['t'] = np.arange(tstart,len(mv2)+tstart)

        result[f'{m}MA'] = mv2

    return result

def b\_hat\_calc(x,y):

    return np.matmul(np.matmul(np.linalg.inv(np.matmul(x.T,x)),x.T),y)

def AR():

    N = int(input('Enter number of samples: '))

    na = int(input("Enter order of the AR process: "))

    input\_n\_string = input("Enter a list of numerators separated by space: ")

    num = list(map(float,input\_n\_string.split()))

    input\_d\_string = input("Enter a list of denominators separated by space: ")

    den = list(map(float,input\_d\_string.split()))

    np.random.seed(123)

    e = np.random.normal(0,1,N)

    system = (num,den,1)

    \_,y = dlsim(system,e)

    T = len(y)- na -1

    vars = []

    for a in range(na, 0, -1):

        vars.append(y[a-1:a+T])

    X = np.hstack(vars)

    Y = np.array(y[na:(na+T)+1])

    def b\_hat\_calc(x,y):

        return np.matmul(np.matmul(np.linalg.inv(np.matmul(x.T,x)),x.T),y)

    b\_hat4 = b\_hat\_calc(-X,Y)

    print(f"Sample {N}: The unknown coefficients are: \n {pd.DataFrame(b\_hat4, columns=['b\_k'])}")

def ARMA():

    T = int(input('Enter the number of data samples: '))

    mean = int(input('Enter the mean of white noise: '))

    var = int(input('Enter the variance of white noise: '))

    ar\_ord = int(input('Enter AR order: '))

    ma\_ord = int(input('Enter MA order: '))

    input\_d\_string = input("Enter the coefficients of AR: hint:- Enter a list separated by space: ")

    an = list(map(float,input\_d\_string.split())) # denominators

    input\_n\_string = input("Enter the coefficients of MA: hint:- Enter a list separated by space: ")

    bn = list(map(float,input\_n\_string.split())) # numerators

    arparams = np.array(an)

    maparams = np.array(bn)

    ar = np.r\_[arparams]

    ma = np.r\_[maparams]

    arma\_process = sm.tsa.ArmaProcess(ar,ma)

    print('Is this a stationary process: ', arma\_process.isstationary)

    mean\_y = mean \* (1 + np.sum(bn)) / (1 + np.sum(an))

    y = arma\_process.generate\_sample(T, scale = np.sqrt(var) + mean\_y)

    lags = int(input('Enter Lag size for ACF: '))

    ry = arma\_process.acf(lags = lags+1)

    return y,ry

def Cal\_GPAC(acf: list, len\_j: int, len\_k: int):

    ''' [Calculate and Plot GPAC Table]

    Args:

        acf (list): list of acf values

        len\_j (int): number of rows of the GPAC table

        len\_k (int): number of columns of the GPAC table

    Returns:

        NONE

    '''

    len\_k = len\_k + 1

    gpac = np.empty(shape=(len\_j, len\_k))

    for k in range(1, len\_k):

        num = np.empty(shape=(k, k))

        den = np.empty(shape=(k, k))

        for j in range(0, len\_j):

            for row in range(0, k):

                for col in range(0, k):

                    if col < k - 1:

                        num[row][col] = acf[np.abs(j+(row-col))]

                        den[row][col] = acf[np.abs(j+(row-col))]

                    else:

                        num[row][col] = acf[np.abs(j+row+1)]

                        den[row][col] = acf[np.abs(j+(row-col))]

            num\_determinant = round(np.linalg.det(num),5)

            denom\_determinant = round(np.linalg.det(den),5)

            if denom\_determinant == 0:

                gpac[j][k] = float('inf')

            else:

                gpac[j][k] = round(num\_determinant/denom\_determinant,3)

    gpac = pd.DataFrame(gpac[:, 1:])

    gpac.columns = [i for i in range(1, len\_k)]

    print("GPAC TABLE: \n",gpac)

    plt.figure(figsize=(8,6))

    sns.heatmap(gpac, annot=True, fmt=".3f", vmin=-1,vmax=1)

    plt.xlabel('k')

    plt.ylabel('j')

    plt.title('Generalized Partial Autocorrelation (GPAC) Table')

    plt.show()

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(figsize=(14,8))

    plt.subplot(211)

    plt.title('ACF/PACF of the raw data')

    plot\_acf(y, ax = plt.gca(), lags =lags)

    plt.subplot(212)

    plot\_pacf(y, ax=plt.gca(), lags =lags)

    fig.tight\_layout(pad = 3)

    plt.show()

# LM ALGORITHM

def cal\_e(teta,na,y):

    numerator=[1]+list(teta[na:])

    denominator=[1]+list(teta[:na])

    if len(numerator)!=len(denominator):

        while len(numerator)<len(denominator):

            numerator.append(0)

        while len(denominator)<len(numerator):

            denominator.append(0)

    system=(denominator,numerator,1)

    \_,e=dlsim(system,y)

    e=[i[0] for i in e]

    return np.array(e)

def step0(na,nb):

    teta\_o=np.zeros(shape=(na+nb,1))

    return teta\_o.flatten()

def step1(delta,na,nb,teta,y):

    x=[]

    e\_teta=cal\_e(teta,na,y)

    SSE\_0=np.dot(e\_teta.T,e\_teta)

    for i in range(na+nb):

        teta\_delta = teta.copy()

        teta\_delta[i]=teta[i]+delta

        en=cal\_e(teta\_delta,na,y)

        xi=(e\_teta-en)/delta

        x.append(xi)

    X=np.transpose(x)

    A=np.dot(X.T,X)

    G=np.dot(X.T,e\_teta)

    return A,G,SSE\_0

def step2(A,G,mu,na,nb,teta,y):

    n=na+nb

    I=np.identity(n)

    dteta1=A+(mu\*I)

    dteta\_inv=np.linalg.inv(dteta1)

    delta\_teta=np.dot(dteta\_inv,G)

    teta\_new=teta+delta\_teta

    e=cal\_e(teta\_new,na,y)

    SSE\_new=np.dot(e.T,e)

    if np.isnan(SSE\_new):

        SSE\_new=10\*\*10

    return SSE\_new,delta\_teta,teta\_new

def step3(max\_iter, mu, delta, epsilon, mu\_max, na, nb, y):

    iter=0

    teta=step0(na,nb)

    SSE=[]

    while iter<max\_iter:

        A,G,SSE\_0=step1(delta, na, nb, teta, y)

        if iter == 0:

            SSE.append(SSE\_0)

        SSE\_new,delta\_teta,teta\_new=step2(A, G, mu, na, nb, teta,y)

        SSE.append(SSE\_new)

        if SSE\_new<SSE\_0:

            if np.linalg.norm(delta\_teta)<epsilon:

                teta\_hat=teta\_new

                var=SSE\_new/(len(y)-A.shape[0])

                A\_inv=np.linalg.inv(A)

                cov=var\*A\_inv

                return SSE,cov,teta\_hat,var

            else:

                teta=teta\_new

                mu=mu/10

        while SSE\_new>=SSE\_0:

            mu = mu \* 10

            if mu > mu\_max:

               print('Mu limit exceeded')

               return None, None, None, None

            SSE\_new, delta\_teta, teta\_new = step2(A, G, mu, na,nb, teta, y)

        iter += 1

        teta = teta\_new

        if iter > max\_iter:

            print('Maximum iterations exceeded')

            return None,None,None,None

def LMA(y, na, nb):

    SSE,cov,teta\_hat,var = step3(100,0.01,1e-6,1e-3,1e10,na,nb,y)

    return SSE,cov,teta\_hat,var

def conf\_int(cov,teta,na,nb):

    print("Confidence Interval:")

    for i in range(na):

        right = teta[i] + 2\*np.sqrt(cov[i][i])

        left = teta[i] - 2\*np.sqrt(cov[i][i])

        print(f'{left:.6f} <a{i+1}< {right:.6f}')

    for i in range(nb):

        right = teta[na+i] + 2\*np.sqrt(cov[na+i][na+i])

        left = teta[na+i] - 2\*np.sqrt(cov[na+i][na+i])

        print(f'{left:.6f} <b{i+1}< {right:.6f}')

def zero\_pole(teta,na):

    y\_den=[1]+list(teta[:na])

    e\_num=[1]+list(teta[na:])

    zeros=np.roots(e\_num)

    poles=np.roots(y\_den)

    print("The roots of numerator(poles): \n",zeros)

    print("The roots of denominator(zero): \n",poles)

def plot\_SSE(SSE):

    iter=np.arange(0,len(SSE))

    plt.plot(iter,SSE)

    plt.xlabel('Number of iterations')

    plt.ylabel('SSE')

    plt.title('Sum square error vs. No. of iterations')

    plt.show()

def inverse\_diff(y,y\_hat,interval=1):

    y\_orginal = np.zeros(len(y))

    for i in range(1,len(y\_hat)):

        y\_orginal[i] = y\_hat[i-interval] + y[i-interval]

    y\_orginal = y\_orginal[1:]

    return y\_orginal

#%%

Source code of data preparation: -

#%%

import pandas as pd

df = pd.read\_csv('data/pollution\_2000\_2021.csv')

df.head()

#%%

# keeping only California data

df = df[df['State'] == 'California']

df.reset\_index(inplace=True)

df.drop(columns=['index'], inplace=True)

#%%

print(df.shape)

#%%

# making date column by merging Year, month, and day

df['Date'] = df[['Year', 'Month','Day']].apply(lambda x: '{}-{}-{}'.format(x[0], x[1], x[2]), axis=1)

#%%

# creating target coulmn avgAQI from 4 AQI data columns

df['avgAQI'] = df[['O3 AQI', 'CO AQI', 'SO2 AQI', 'NO2 AQI']].mean(axis=1)

#%%

# keeping only Los Angeles county for further analysis

df2 = df[df['County'] == 'Los Angeles']

df2.reset\_index(inplace=True)

df2.drop(columns=['index'], inplace=True)

#%%

# making date as index

df3 = df2.copy()

df3.Date=pd.to\_datetime(df3.Date)

df3.set\_index('Date',inplace=True)

# keeping non duplicated data

finaldf=df3[~df3.index.duplicated(keep='first')]

print(df.shape)

print(finaldf.Year.value\_counts().sort\_index())

#%%

import matplotlib.pyplot as plt

plt.figure(figsize=(14,8))

finaldf['avgAQI'].plot()

plt.xlabel('Time', fontsize=22)

plt.ylabel('Average Air Quality Index (AQI)', fontsize=22)

plt.tight\_layout()

plt.title('Dependant Variable-avgAQI vs Time',  fontsize=30)

plt.legend(fontsize=24)

plt.show()

#%%

# storing dataframe in csv

finaldf.to\_csv('data/AQI\_CA\_LA.csv', index=True, index\_label='Date')

# %%

References

[1] <https://www.sciencedirect.com/science/article/pii/S2667259621000023>

[2] <https://fit.thequint.com/health-news/explaining-air-quality-index#read-more>

[3] <https://otexts.com/fpp2/stl.html>

[4] <https://www.datascienceinstitute.net/blog/time-series-decomposition-in-r>

Link to dataset: [US Pollution 2000-2021 | Kaggle](https://www.kaggle.com/datasets/alpacanonymous/us-pollution-20002021)

Link to GitHub Repository: