DATS6313

TIME SERIES ANALYSIS & MODELING

Dr. Reza Jafari

Final Term Project

Nusrat Nawshin

NN

05/1/2022

Table of Content

Table of Figures and Tables

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 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).

Date	Year	Month	Day	Address	State	County	City	O3 Mean	O3 1st Max Value	O3 1st Max Hour	...	CO AQI	SO2 Mean	SO2 1st Max Value	SO2 1st Max Hour	SO2 AQI	NO2 Mean	NO2 1st Max Value	NO2 1st Max Hour	NO2 AQI	avgAQI
2000-01-01	2000	1	1	228 W. PALM AVE., BURBANK	California	Los Angeles	Burbank	0.016529	0.028	10	...	34.0	0.000000	0.0	0	0.0	25.869565	41.0	23	39	24.75
2000-01-02	2000	1	2	228 W. PALM AVE., BURBANK	California	Los Angeles	Burbank	0.030706	0.040	7	...	9.0	0.000000	0.0	0	0.0	13.086957	32.0	0	30	19.00
2000-01-03	2000	1	3	228 W. PALM AVE., BURBANK	California	Los Angeles	Burbank	0.012235	0.023	9	...	42.0	0.434783	2.0	19	3.0	43.000000	73.0	21	71	34.25
2000-01-04	2000	1	4	228 W. PALM AVE., BURBANK	California	Los Angeles	Burbank	0.007588	0.013	8	...	50.0	0.869565	3.0	8	4.0	46.913043	70.0	9	68	33.50
2000-01-05	2000	1	5	228 W. PALM AVE., BURBANK	California	Los Angeles	Burbank	0.021176	0.032	18	...	39.0	0.521739	4.0	9	6.0	46.130435	145.0	9	109	46.00

Figure 1: 1st five records of the dataset

```
Data columns (total 24 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Year                7852 non-null   int64
 1   Month               7852 non-null   int64
 2   Day                 7852 non-null   int64
 3   Address             7852 non-null   object
 4   State               7852 non-null   object
 5   County              7852 non-null   object
 6   City                7852 non-null   object
 7   O3 Mean             7852 non-null   float64
 8   O3 1st Max Value    7852 non-null   float64
 9   O3 1st Max Hour     7852 non-null   int64
 10  O3 AQI              7852 non-null   int64
 11  CO Mean             7852 non-null   float64
 12  CO 1st Max Value    7852 non-null   float64
 13  CO 1st Max Hour     7852 non-null   int64
 14  CO AQI              7852 non-null   float64
 15  SO2 Mean            7852 non-null   float64
 16  SO2 1st Max Value   7852 non-null   float64
 17  SO2 1st Max Hour    7852 non-null   int64
 18  SO2 AQI             7852 non-null   float64
 19  NO2 Mean            7852 non-null   float64
 20  NO2 1st Max Value   7852 non-null   float64
 21  NO2 1st Max Hour    7852 non-null   int64
 22  NO2 AQI             7852 non-null   int64
 23  avgAQI              7852 non-null   float64
dtypes: float64(11), int64(9), object(4)
```

Figure 2: Dataset column data type

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

	O3 Mean	O3 1st Max Value	O3 1st Max Hour	CO Mean	CO 1st Max Value	CO 1st Max Hour	SO2 Mean	SO2 1st Max Value	SO2 1st Max Hour	NO2 Mean	NO2 1st Max Value	NO2 1st Max Hour	avgAC
count	7852.000000	7852.000000	7852.000000	7852.000000	7852.000000	7852.000000	7852.000000	7852.000000	7852.000000	7852.000000	7852.000000	7852.000000	7852.00000
mean	0.025035	0.040974	10.193454	0.654280	0.958788	7.645313	0.855750	1.858087	7.815079	26.028570	41.572529	12.255222	23.71857
std	0.011394	0.016837	2.043756	0.504741	0.776945	6.878804	1.333562	3.106859	5.998339	12.565829	18.320828	7.381765	8.93208
min	0.000000	0.000000	7.000000	-0.314286	-0.300000	0.000000	-0.466667	-0.300000	0.000000	0.269565	1.200000	0.000000	7.50000
25%	0.016235	0.029000	9.000000	0.316667	0.400000	2.000000	0.108696	0.500000	4.000000	16.349457	29.100000	7.000000	17.75000
50%	0.025454	0.041000	10.000000	0.500000	0.700000	7.000000	0.395833	1.000000	7.000000	24.817391	39.800000	11.000000	21.75000
75%	0.033412	0.051000	11.000000	0.829167	1.200000	10.000000	1.043478	2.100000	12.000000	33.826087	51.000000	19.000000	27.25000
max	0.067000	0.128000	23.000000	5.037500	6.200000	23.000000	17.636364	130.000000	23.000000	94.521739	180.000000	23.000000	77.75000

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 considered 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.

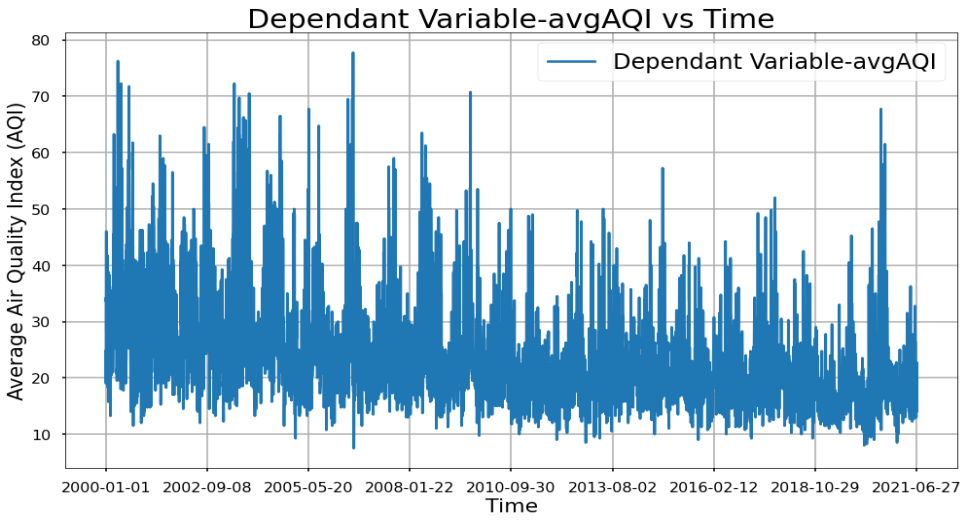

Figure 4: Dependent variable 'avgAQI' vs time

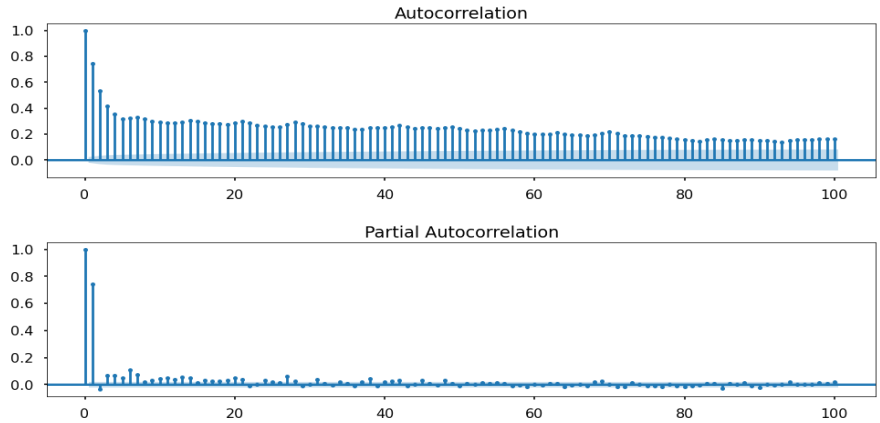

Figure 5: ACF/PACF of dependent variable 'avgAQI'

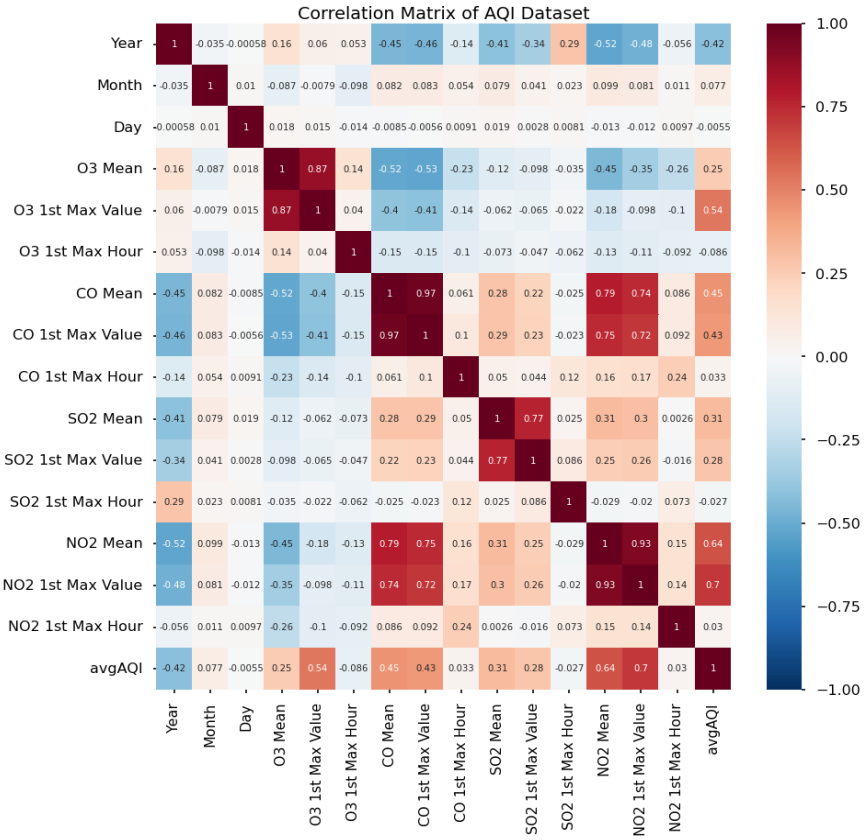

Figure 6: Correlation Matrix of all features

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.

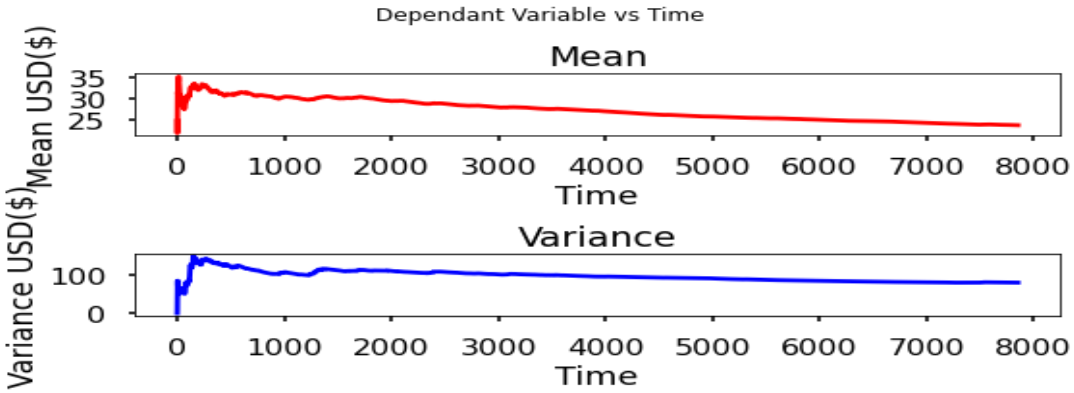

Figure 7: Rolling mean and variance of the dependent variable

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

```
ADF Statistic: -7.463589
p-value: 0.000000
Critical Values:
        1%: -3.431
        5%: -2.862
        10%: -2.567
```

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.

```
Results of KPSS Test:
Test Statistic            8.815354
p-value                   0.010000
Lags Used                46.000000
Critical Value (10%)      0.347000
Critical Value (5%)       0.463000
Critical Value (2.5%)     0.574000
Critical Value (1%)       0.739000
dtype: float64
```

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.

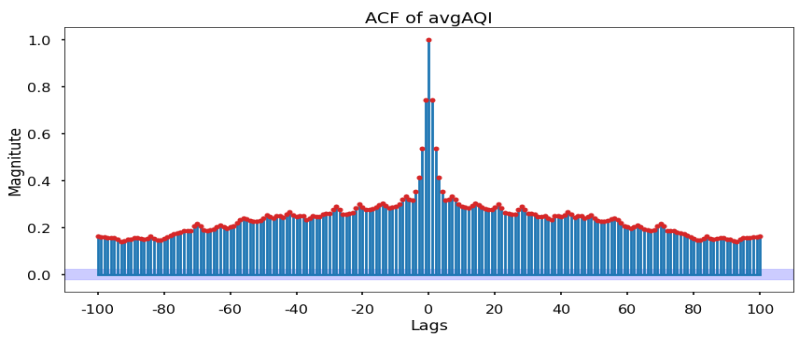

Figure 10: ACF of average AQI

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].

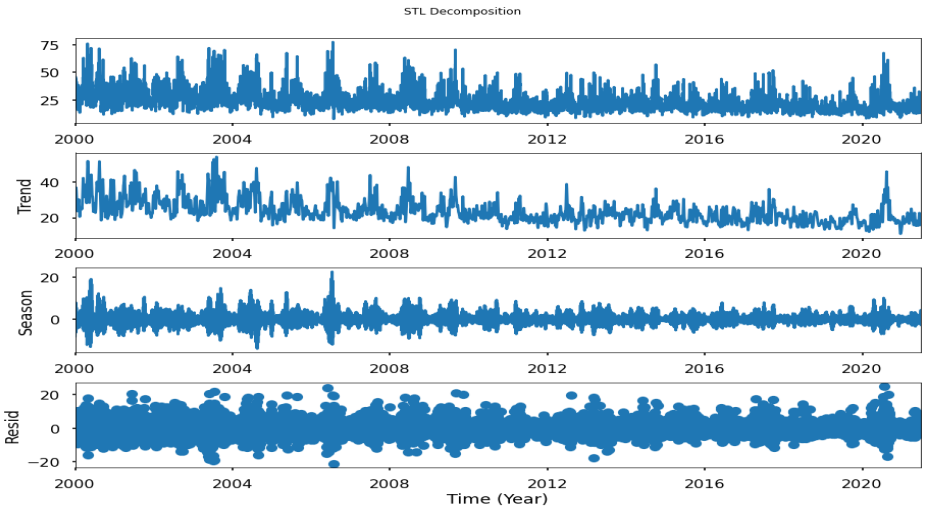

Figure 11: STL decomposition

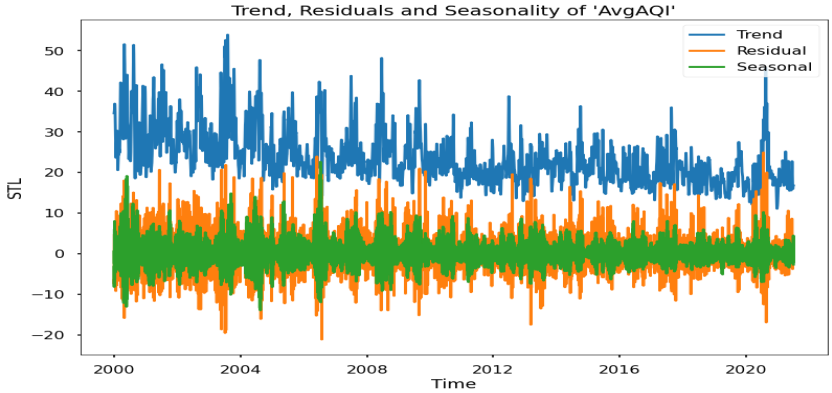

Figure 12: Trend, Residuals and Seasonality of 'AvgAQI'

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).

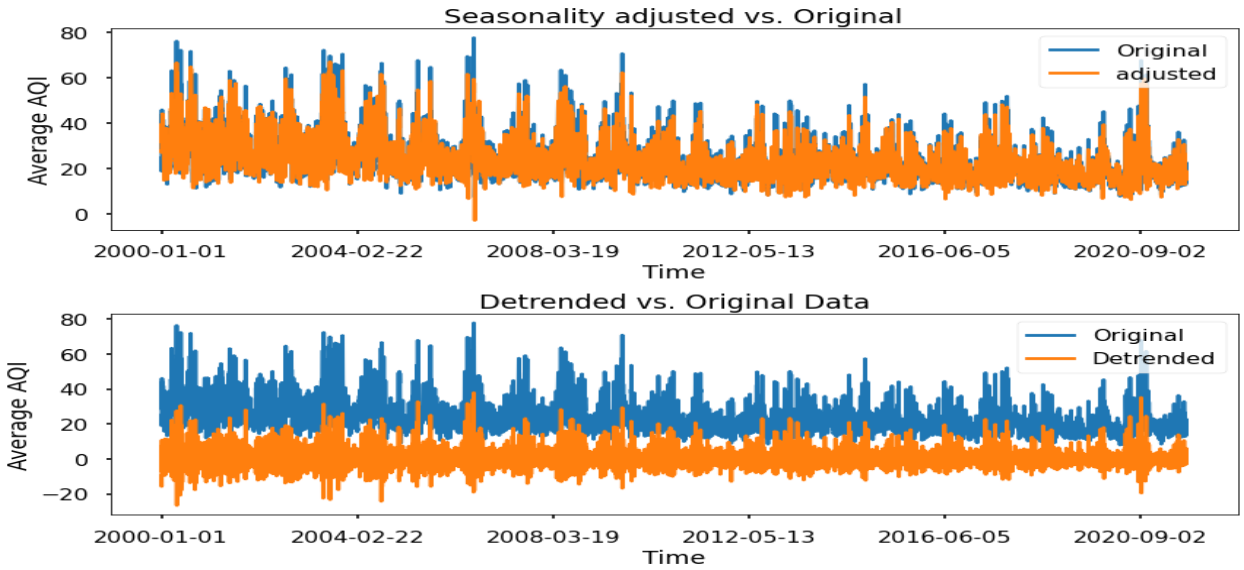

Figure 13: Seasonality adjusted and detrended data

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).

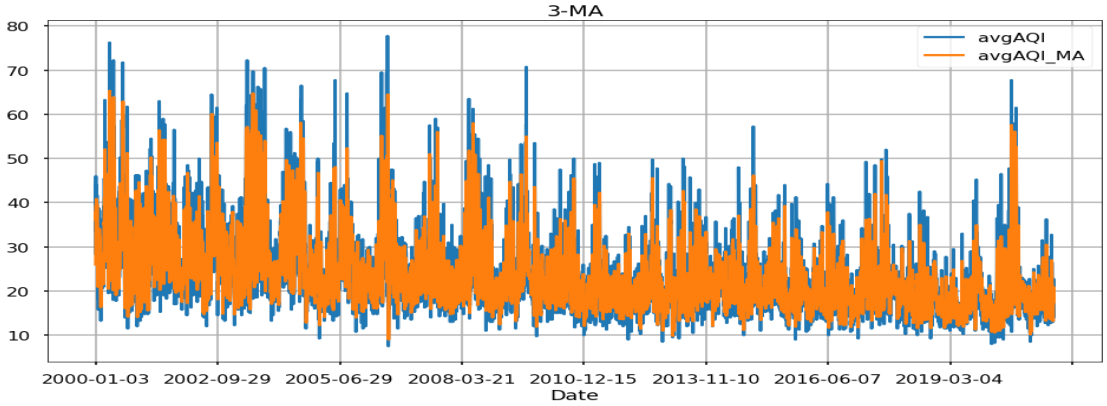

Figure 14: 3-Moving Average of dependent variable

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.

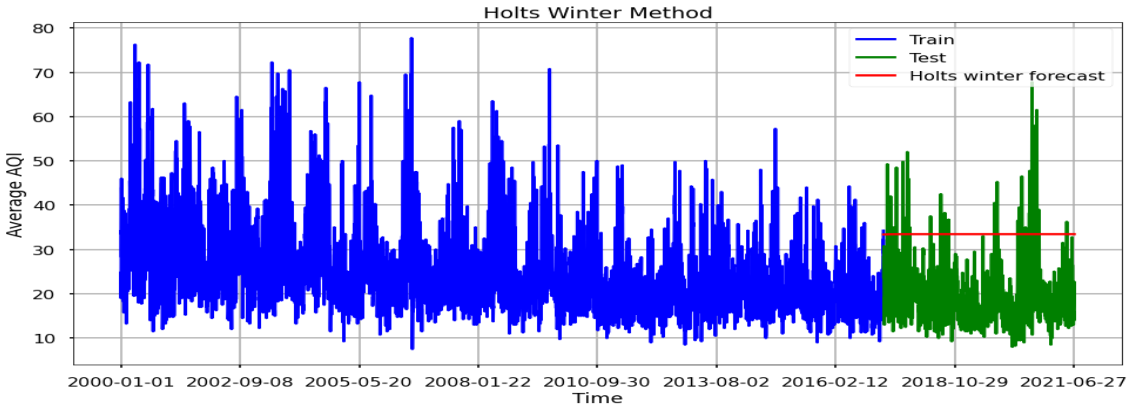

Figure 15: Holt-Winter forecasting

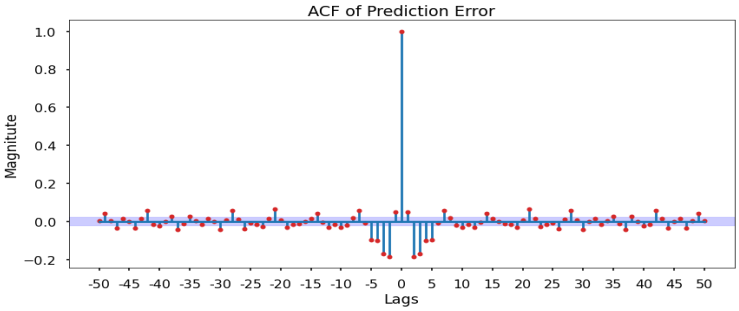

Figure 16: ACF of Holt-Winter Residuals

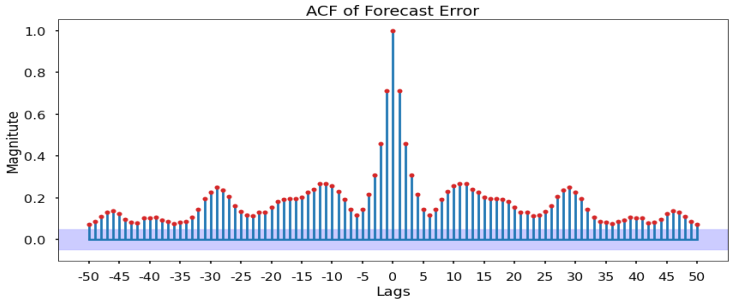

Figure 17: ACF of Holt-Winter Forecast errors

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,

singular value for X are :-

	0
0	3.175529e+10
1	3.815006e+06
2	6.085185e+05
3	4.889580e+05
4	3.067766e+05
5	2.642439e+05
6	1.204909e+05
7	9.147243e+04
8	7.665125e+04
9	3.147486e+04
10	4.953629e+03
11	2.709006e+03
12	6.700568e+01
13	2.091666e+00
14	9.809790e-02
15	3.920933e-02

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.

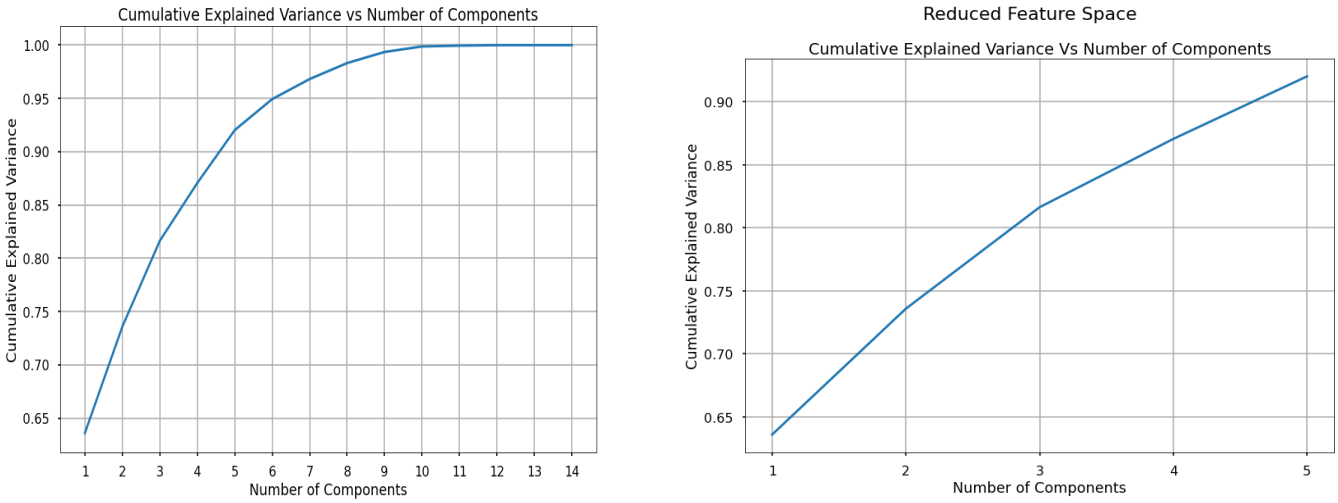

Figure 19: Cumulative explained variance vs number of component

```
Original Dimension: (7852, 15)
Transformed Dimension: (7852, 5)
Explained variance ratio:
 [0.63590751 0.09992183 0.08052794 0.05423488 0.04965561]
```

Figure 20: Reduced feature space explained variance ratio

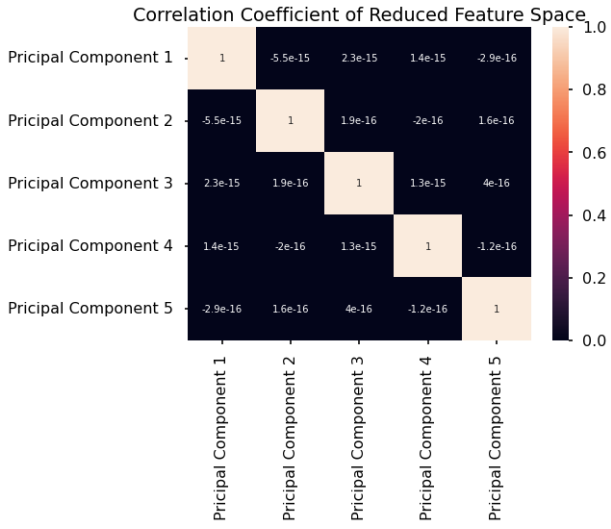

Figure 21: Reduced feature space correlation matrix

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.

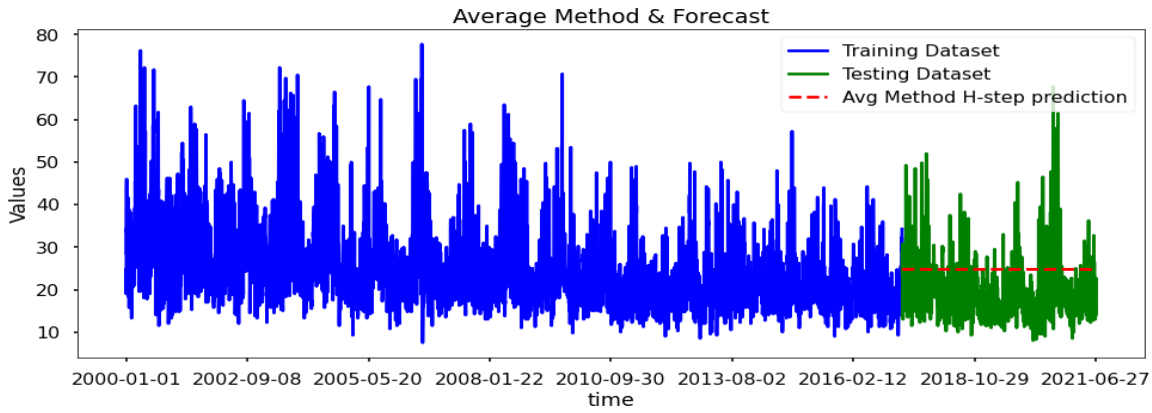

Figure 22: Average forecasting

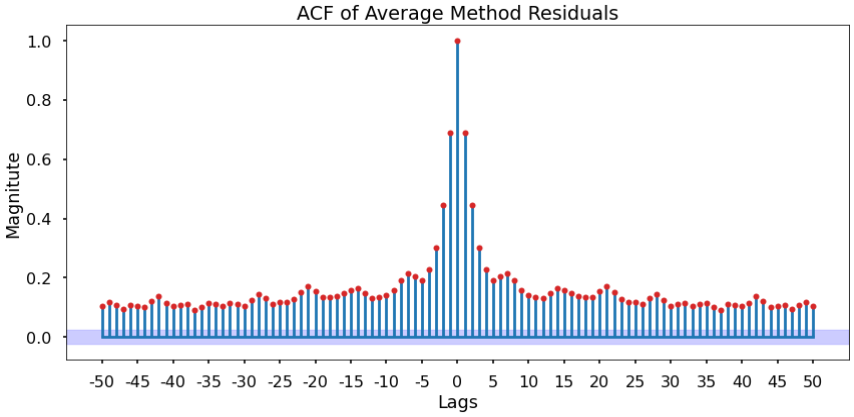

Figure 23: ACF of Average method residuals

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.

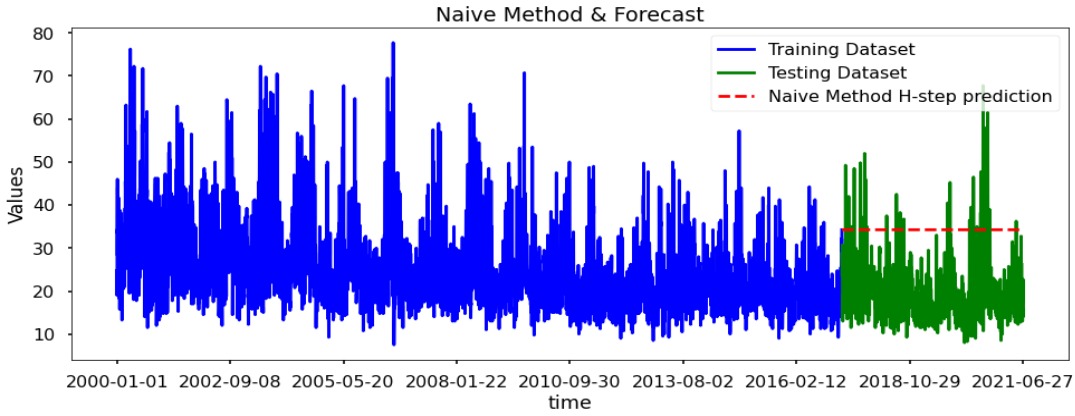

Figure 24. Naïve method forecast

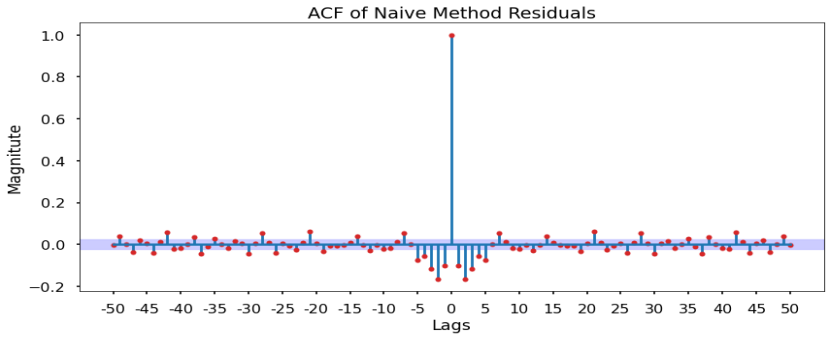

Figure 25: ACF of Naïve method residuals

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.

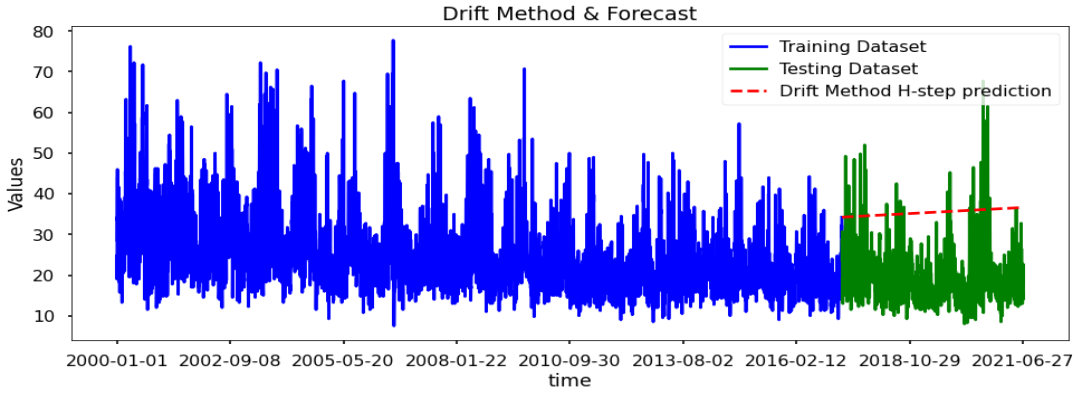

Figure 26: Drift method forecasting

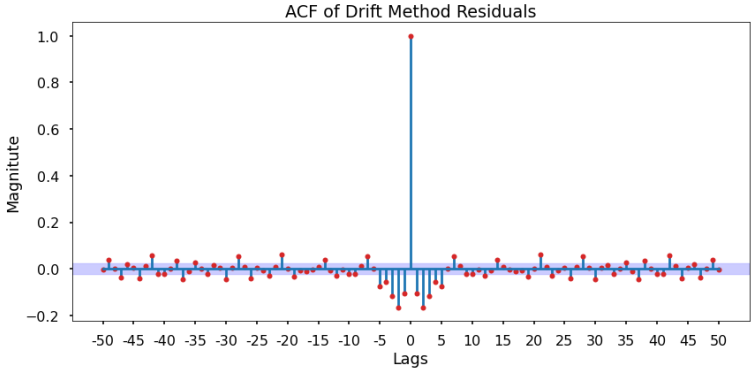

Figure 27: ACF of Drift method residuals

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.

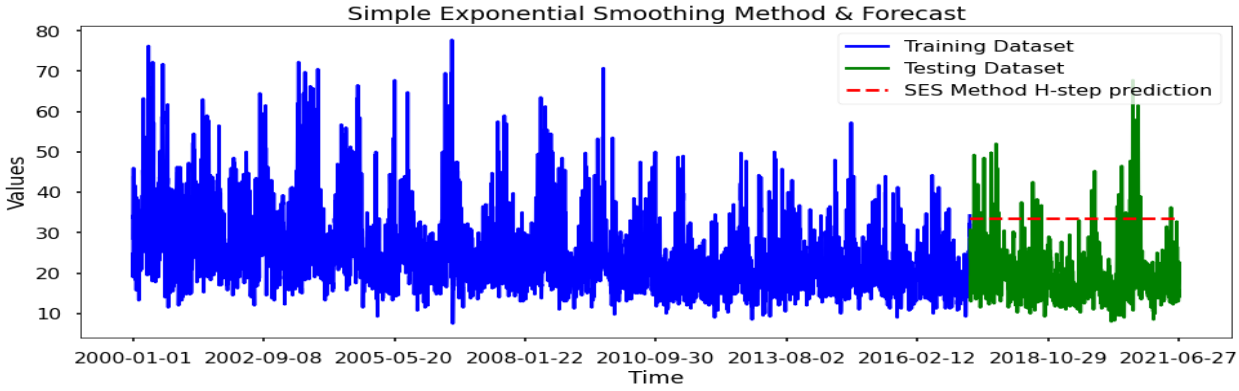

Figure 28: SES forecasting

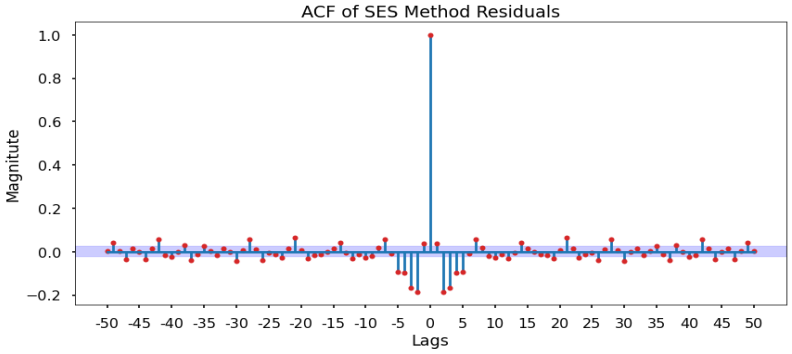

Figure 29: ACF of SES method residuals

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

OLS Regression Results

Dep. Variable:	avgAQI	R-squared:	0.798
Model:	OLS	Adj. R-squared:	0.798
Method:	Least Squares	F-statistic:	1.243e+04
Date:	Thu, 28 Apr 2022	Prob (F-statistic):	0.00
Time:	23:01:08	Log-Likelihood:	-17767.
No. Observations:	6281	AIC:	3.554e+04
Df Residuals:	6278	BIC:	3.556e+04
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-3.1477	0.185	-17.012	0.000	-3.510	-2.785
O3 1st Max Value	461.4465	3.270	141.112	0.000	455.036	467.857
CO 1st Max Value	8.7080	0.069	125.596	0.000	8.572	8.844

Omnibus:	890.596	Durbin-Watson:	0.973
Prob(Omnibus):	0.000	Jarque-Bera (JB):	3195.281
Skew:	0.692	Prob(JB):	0.00
Kurtosis:	6.209	Cond. No.	100.

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

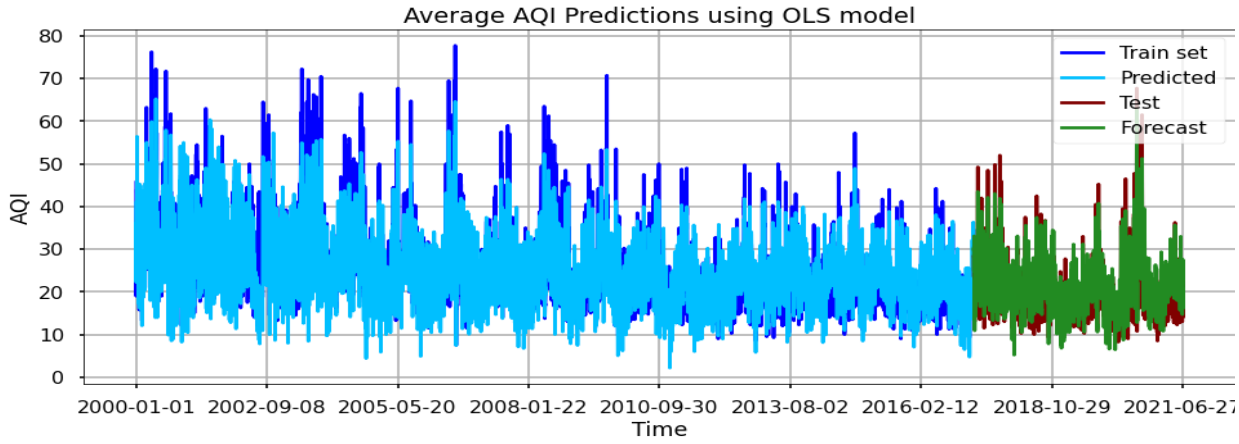

Figure 30: Multiple Linear Regression Model

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.

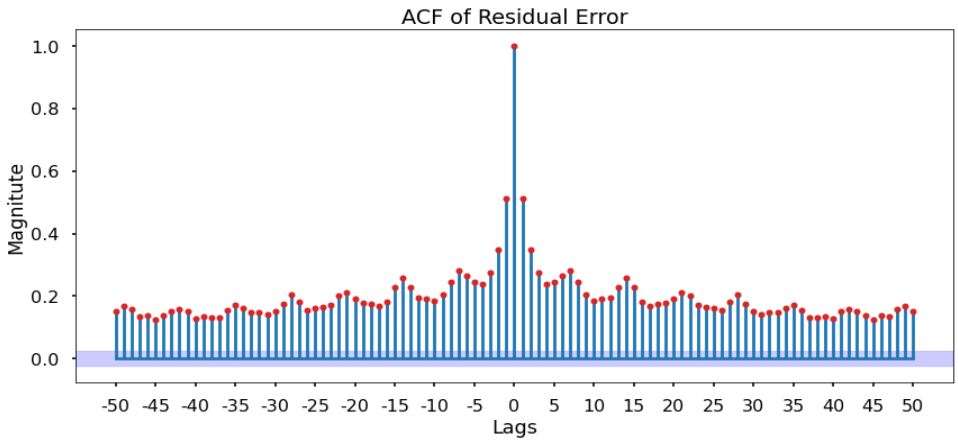

Figure 31: ACF of OLS residuals

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):

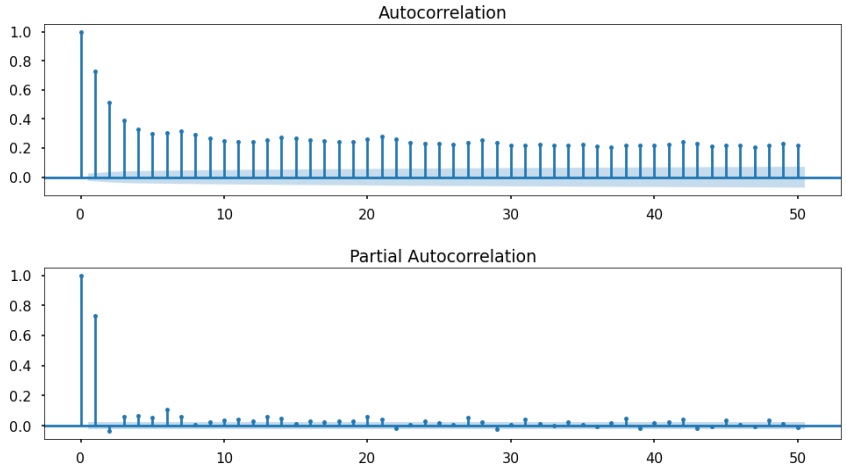

Figure 32: ACF of dependent variable Trainset

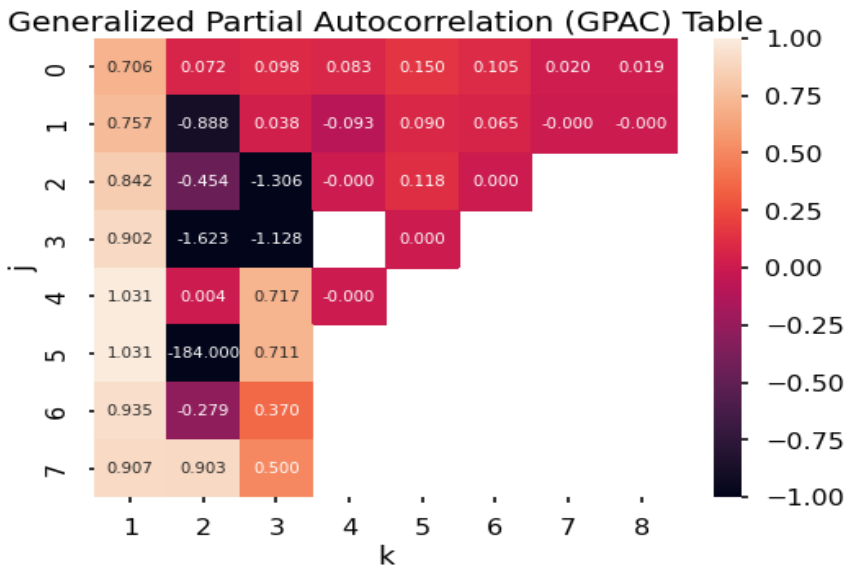

Figure 33: GPAC table of y train set

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):

```
The AR coefficient a0 is: -0.97
                              ARMA Model Results
==============================================================================
Dep. Variable:                 avgAQI   No. Observations:                 6281
Model:                     ARMA(1, 0)   Log Likelihood              -20816.940
Method:                       css-mle   S.D. of innovations              6.653
Date:                Fri, 29 Apr 2022   AIC                          41637.880
Time:                        17:24:26   BIC                          41651.371
Sample:                             0   HQIC                         41642.555

==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1.avgAQI     0.9677      0.003    304.798      0.000       0.961       0.974
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            1.0334           +0.0000j            1.0334            0.0000
-----------------------------------------------------------------------------
```

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.

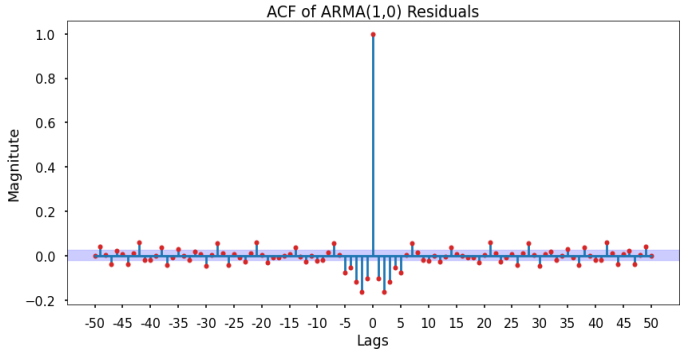

Figure 35.1: ACF of ARMA (1,0) residuals

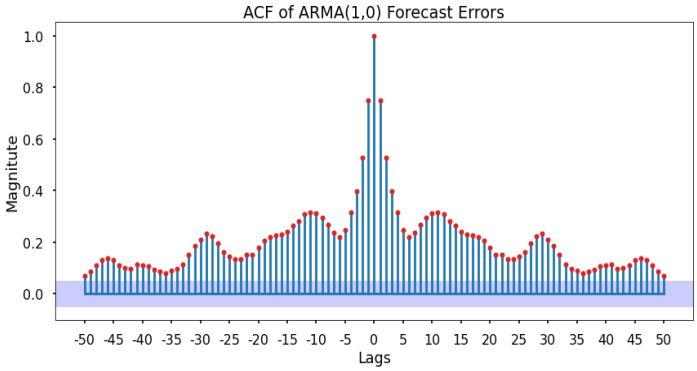

Figure 35.2: ACF of ARMA (1,0) forecast errors

MSE of Residuals: 44.36
MSE of Forecast Error: 11.01
Q-Value: 514.35

	ar.L1.avgAQI
ar.L1.avgAQI	0.00001

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

ARMA (3,1):

ARMA Model Results

Dep. Variable:	avgAQI	No. Observations:	6281
Model:	ARMA(3, 1)	Log Likelihood	-20287.826
Method:	css-mle	S.D. of innovations	6.115
Date:	Thu, 28 Apr 2022	AIC	40585.652
Time:	23:47:28	BIC	40619.379
Sample:	0	HQIC	40597.338

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1.avgAQI	1.6911	0.001	3069.716	0.000	1.690	1.692
ar.L2.avgAQI	-0.7913	0.001	-680.060	0.000	-0.794	-0.789
ar.L3.avgAQI	0.1002	0.001	97.040	0.000	0.098	0.102
ma.L1.avgAQI	-0.9683	0.004	-263.105	0.000	-0.976	-0.961

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	1.0002	+0.0000j	1.0002	0.0000
AR.2	2.0648	+0.0000j	2.0648	0.0000
AR.3	4.8327	+0.0000j	4.8327	0.0000
MA.1	1.0327	+0.0000j	1.0327	0.0000

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.

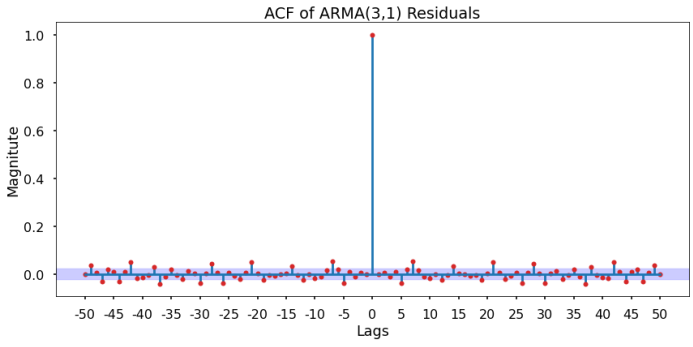

Figure 37.1: ARMA (3,1) Residuals

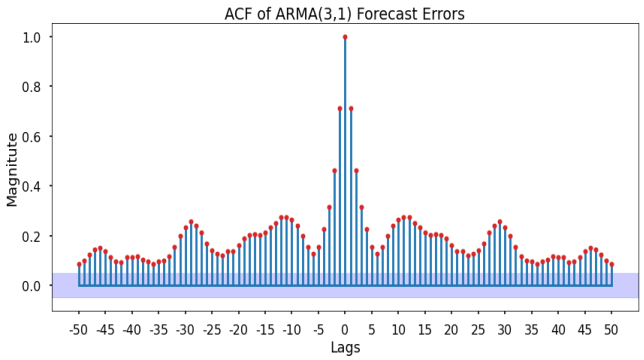

Figure 37.2: ARMA (3,1) forecast error

MSE of Residuals: 37.50
MSE of Forecast Error: 46.60
Q-Value: 167.51

	ar.L1.avgAQI	ar.L2.avgAQI	ar.L3.avgAQI	ma.L1.avgAQI
ar.L1.avgAQI	3.034740e-07	-2.948282e-07	-5.832162e-09	-9.003170e-08
ar.L2.avgAQI	-2.948282e-07	1.354002e-06	-1.058600e-06	-1.442148e-07
ar.L3.avgAQI	-5.832162e-09	-1.058600e-06	1.066144e-06	1.766435e-07
ma.L1.avgAQI	-9.003170e-08	-1.442148e-07	1.766435e-07	1.354512e-05

Figure 37.3: ARMA (3,1) Covariance Matrix

ARIMA:

The Autoregressive Integrated Moving Average (ARIMA) model is a combination of the differenced autoregressive model with the moving average model. The AR part of ARIMA shows that the time series is regressed on its own past data. The MA part of ARIMA indicates that the forecast error is a linear combination of past respective errors [5].

ARIMA (1,1,0):

```
The AR coefficient a1 is: 0.10
                             ARIMA Model Results
==============================================================================
Dep. Variable:                 D.avgAQI   No. Observations:                 6280
Model:                 ARIMA(1, 1, 0)   Log Likelihood              -20829.568
Method:                       css-mle   S.D. of innovations              6.672
Date:                Thu, 28 Apr 2022   AIC                          41665.136
Time:                        23:57:16   BIC                          41685.371
Sample:                             1   HQIC                         41672.147

==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0015      0.076      0.020      0.984      -0.148       0.151
ar.L1.D.avgAQI -0.1033      0.013     -8.231      0.000      -0.128      -0.079
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1           -9.6788           +0.0000j            9.6788            0.5000
-----------------------------------------------------------------------------
```

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.

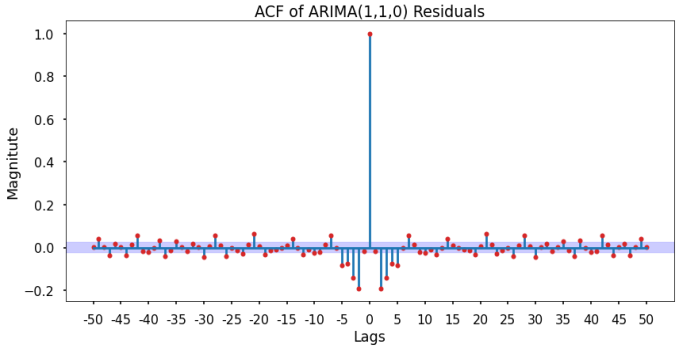

Figure 39.1: ARIMA (1,1,0) residuals

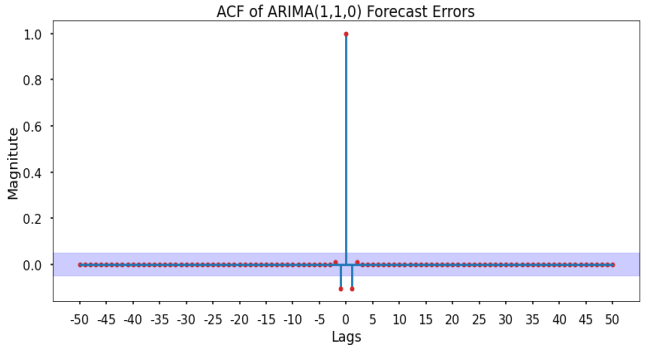

Figure 39.2: ARIMA (1,1,0) forecast errors

MSE of Residuals: 0.62
MSE of Forecast Error: 0.00
Q-Value: 651.29

	const	ar.L1.D.avgAQI
const	5.822658e-03	-1.552343e-08
ar.L1.D.avgAQI	-1.552343e-08	1.575424e-04

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):

ARIMA Model Results

Dep. Variable:	D.avgAQI	No. Observations:	6280
Model:	ARIMA(3, 1, 1)	Log Likelihood	-20283.191
Method:	css-mle	S.D. of innovations	6.115
Date:	Fri, 29 Apr 2022	AIC	40578.382
Time:	00:04:49	BIC	40618.853
Sample:	1	HQIC	40592.404

	coef	std err	z	P>|z|	[0.025	0.975]
const	-0.0015	0.006	-0.243	0.808	-0.013	0.010
ar.L1.D.avgAQI	0.6898	0.013	51.244	0.000	0.663	0.716
ar.L2.D.avgAQI	-0.0950	0.015	-6.202	0.000	-0.125	-0.065
ar.L3.D.avgAQI	-0.0084	0.013	-0.632	0.527	-0.034	0.018
ma.L1.D.avgAQI	-0.9677	0.005	-204.481	0.000	-0.977	-0.958

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	2.5023	+0.0000j	2.5023	0.0000
AR.2	2.8569	+0.0000j	2.8569	0.0000
AR.3	-16.6946	+0.0000j	16.6946	0.5000
MA.1	1.0333	+0.0000j	1.0333	0.0000

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.

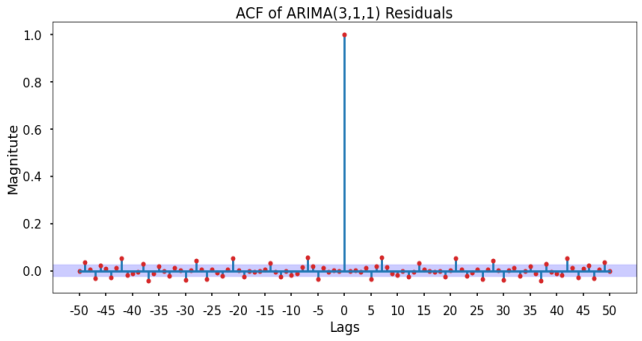

Figure 41.1: ARIMA (3,1,1) residuals

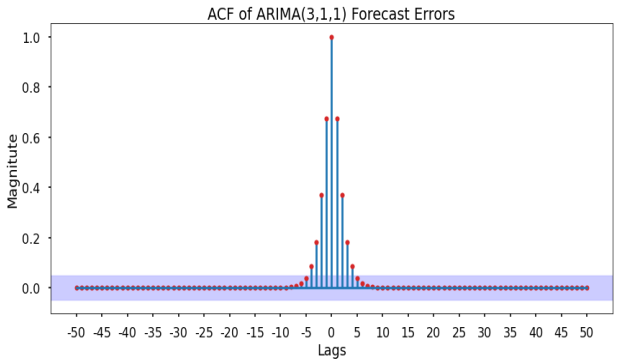

Figure 41.2: ARIMA (3,1,1) forecast errors

MSE of Residuals: 7.72
MSE of Forecast Error: 0.03
Q-Value: 167.33

	const	ar.L1.D.avgAQI	ar.L2.D.avgAQI	ar.L3.D.avgAQI	ma.L1.D.avgAQI
const	3.658209e-05	-3.142393e-07	-3.529802e-08	-2.137903e-07	3.297526e-07
ar.L1.D.avgAQI	-3.142393e-07	1.811851e-04	-1.057727e-04	3.508518e-05	-2.215263e-05
ar.L2.D.avgAQI	-3.529802e-08	-1.057727e-04	2.345227e-04	-1.064320e-04	-4.266430e-06
ar.L3.D.avgAQI	-2.137903e-07	3.508518e-05	-1.064320e-04	1.757893e-04	-1.913922e-05
ma.L1.D.avgAQI	3.297526e-07	-2.215263e-05	-4.266430e-06	-1.913922e-05	2.239785e-05

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:

SARIMA stands for Seasonal Autoregressive Integrated Moving Average. It's very much like ARIMA but with seasonal order. Here I am performing SARIMA (3,0,1) x (0,2,0,7) even though my dataset is not seasonal.

SARIMAX Results

Dep. Variable:	avgAQI	No. Observations:	6281
Model:	SARIMAX(3, 0, 1)x(0, 2, [], 7)	Log Likelihood	-24906.664
Date:	Fri, 29 Apr 2022	AIC	49823.329
Time:	00:15:39	BIC	49857.042
Sample:	0	HQIC	49835.011
	- 6281		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	1.4765	0.009	162.734	0.000	1.459	1.494
ar.L2	-0.5170	0.016	-32.304	0.000	-0.548	-0.486
ar.L3	-0.1114	0.010	-11.572	0.000	-0.130	-0.093
ma.L1	-1.0000	0.380	-2.629	0.009	-1.746	-0.254
sigma2	166.1940	63.152	2.632	0.008	42.419	289.969

Ljung-Box (L1) (Q):	1.67	Jarque-Bera (JB):	1370.80
Prob(Q):	0.20	Prob(JB):	0.00
Heteroskedasticity (H):	0.39	Skew:	-0.23
Prob(H) (two-sided):	0.00	Kurtosis:	5.25

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

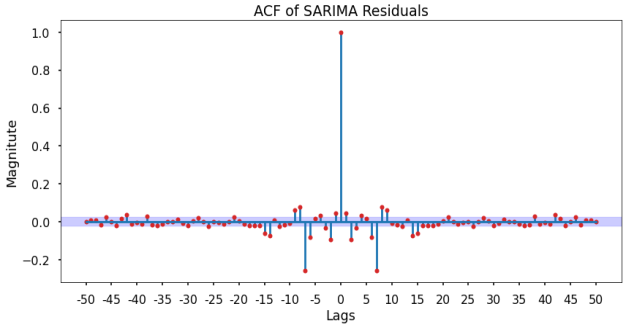

Figure 43.1: SARIMA (3,0,1) x (0,2,0,7) residuals

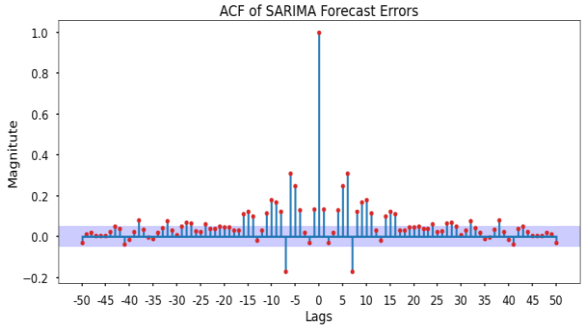

Figure 43.2: SARIMA (3,0,1) x (0,2,0,7) forecast errors

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.

```
Estimated ARMA(3,1) model parameters using the LM Algorithm are:-
 [-1.68920529  0.78915477 -0.09984795 -0.95828647]

Standard deviation of parameter estimates: 0.93
Confidence Interval:
-1.715605 <a1< -1.662805
0.743820 <a2< 0.834489
-0.125828 <a3< -0.073868
-0.967120 <b1< -0.949453
```

Figure 44: LM algorithm estimated parameters

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

Diagnostic Analysis

Table 2: Confidence Intervals

OLS:-

	0	1
const	-3.510406	-2.784970
O3 1st Max Value	455.036036	467.856992
CO 1st Max Value	8.572090	8.843924

ARMA(1,0):-

	0	1
ar.L1.avgAQI	0.9615	0.973946

ARMA(3,1):-

	0	1
ar.L1.avgAQI	1.689980	1.692140
ar.L2.avgAQI	-0.793609	-0.789047
ar.L3.avgAQI	0.098174	0.102222
ma.L1.avgAQI	-0.975536	-0.961109

SARIMA:-

	0	1
ar.L1	1.458763	1.494330
ar.L2	-0.548329	-0.485598
ar.L3	-0.130315	-0.092566
ma.L1	-1.745562	-0.254447
sigma2	42.419002	289.968962

ARIMA(1,1,0):-

	0	1
const	-0.148034	0.151082
ar.L1.D.avgAQI	-0.127919	-0.078718

ARIMA(3,1,1):-

	0	1
const	-0.013326	0.010383
ar.L1.D.avgAQI	0.663384	0.716148
ar.L2.D.avgAQI	-0.124995	-0.064964
ar.L3.D.avgAQI	-0.034365	0.017607
ma.L1.D.avgAQI	-0.977009	-0.958457

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

ARMA(1,0):-

```
The roots of numerator(poles):
 []
The roots of denominator(zero):
 [-0.9677227]
```

ARMA(3,1):-

```
The roots of numerator(poles):
 [0.96832265]
The roots of denominator(zero):
 [-2.09218168+0.j          0.2005609
+0.08756138j  0.2005609 -0.08756138j]
```

```
ARIMA(1,1,0):-

The roots of numerator(poles):
 [0.10331831]
The roots of denominator(zero):
 [-0.00152395]
```

```
ARIMA(3,1,1):-

The roots of numerator(poles):
 [ 0.98793266 -0.97955363]
The roots of denominator(zero):
 [-0.06637933+0.8385581j -0.06637933-
0.8385581j  0.13423043+0.j       ]
```

```
SARIMA:-

The roots of numerator(poles):
 [1.00000472]
The roots of denominator(zero):
 [-1.73720369  0.41517029 -0.15451348]
```

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:

$$y(t) - 0.69\, y(t-1) + 0.09\, y(t-2) + 0.01\, y(t-3) = e(t) - 0.97\, e(t-1)$$

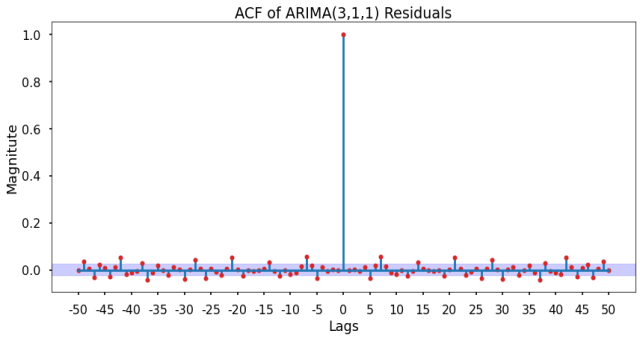

Figure 45: ACF of residuals of final model

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)
```

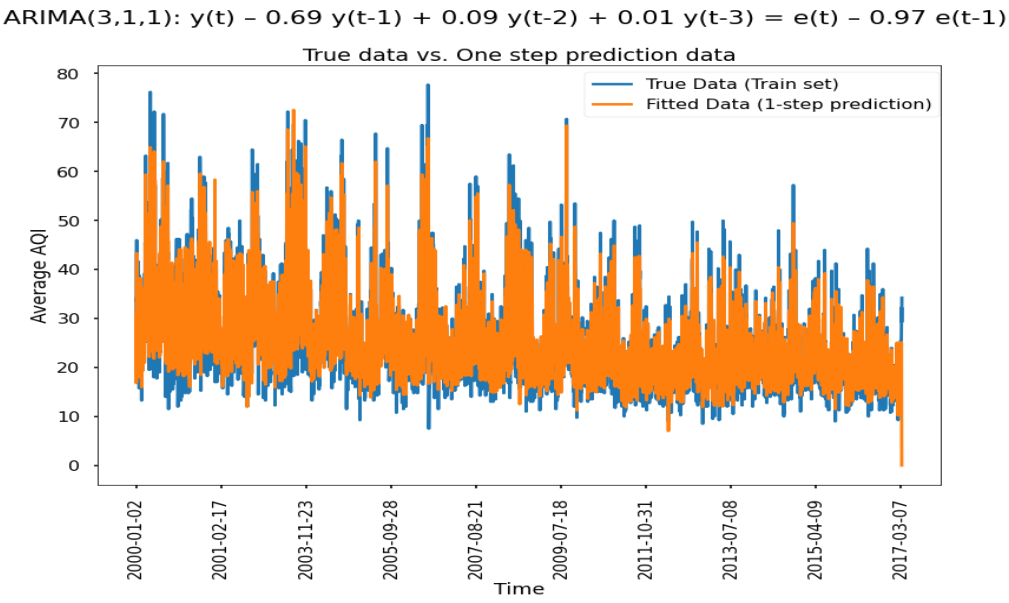

Figure 46: Forecast with final ARIMA (3,1,1) model

h-step ahead prediction:

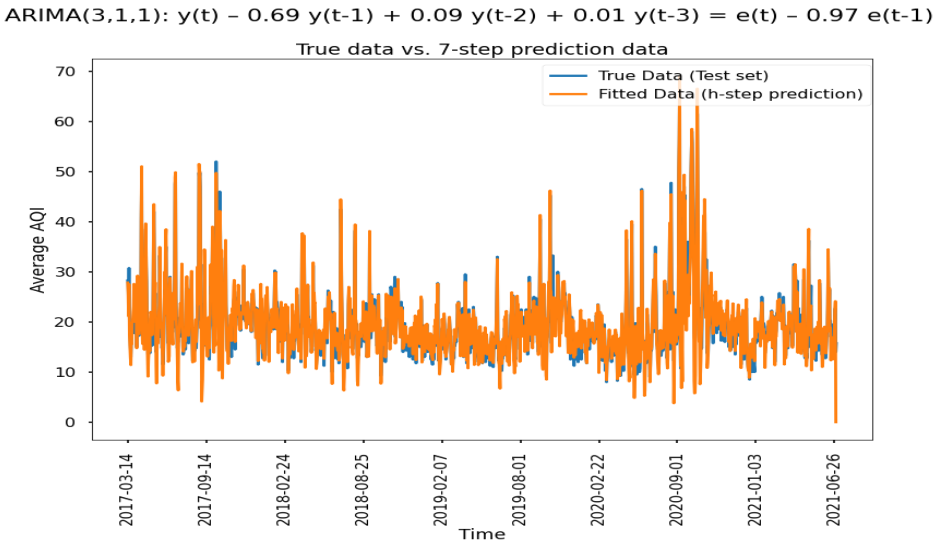

Figure 47: 7-step prediction in ARIMA (3,1,1) model

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.

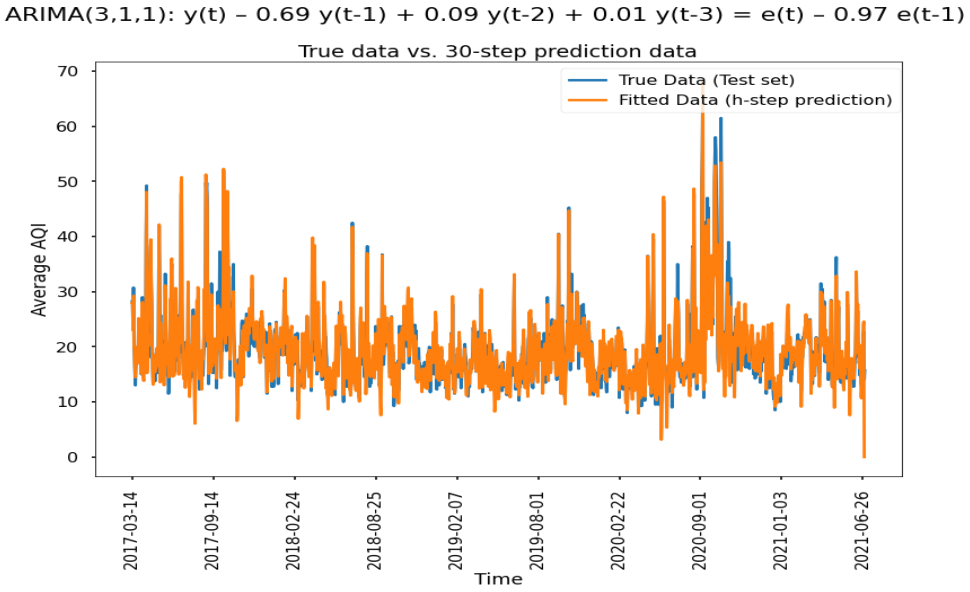

Figure 48: 30-step prediction on ARIMA (3,1,1) model

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

[5] https://www.sciencedirect.com/topics/mathematics/autoregressive-integrated-moving-average

Link to dataset: US Pollution 2000-2021 | Kaggle

Link to GitHub Repository: https://github.com/NusratNawshin/Air-Quality-Index-Prediction