THE GEORGE WASHINGTON UNIVERSITY

WASHINGTON, DC

Time series Analysis and Modeling

DATS 6313-10

Final Project

Instructor Name: Reza Jafari

Submitted by: Sara Sanchez

Date: 04-May-2022

Table of content

I. Abstract.

In the present final project, I'm going to use the Air Quality Data set from the UCI Machine Learning Repository. The main goal of this project is to develop a prediction model for our dependent variable, Relative Humidity. This project will begin by preprocessing the dataset and the time series decomposition, then developing several models and evaluating them to choose the best one.

II. Introduction:

In this report I'm going to make use of an Air Quality Data set from the UCI machine learning repository. This dataset has 9,358 instances "of hourly averaged responses from an array of 5 metal oxide chemical sensors embedded in an Air Quality Chemical Multisensor Device" (UCI, n.d.). The records are taken from a device located in an Italian city from March 2004 to February 2005.

The main goal of this project is to understand which factors influence Relative Humidity throughout the study period. To develop this, I will use several models to make predictions like Average, Naïve, Drift, Simple Exponential Smoothing, Holts Winter, Multiple Linear Regression, ARMA, and SARIMA. It should be highlighted that the SARIMA model will be used because this dataset has strong seasonality. Given this, it will be an analysis based on the performance of these models to decide which one is the best.

III. Data Description

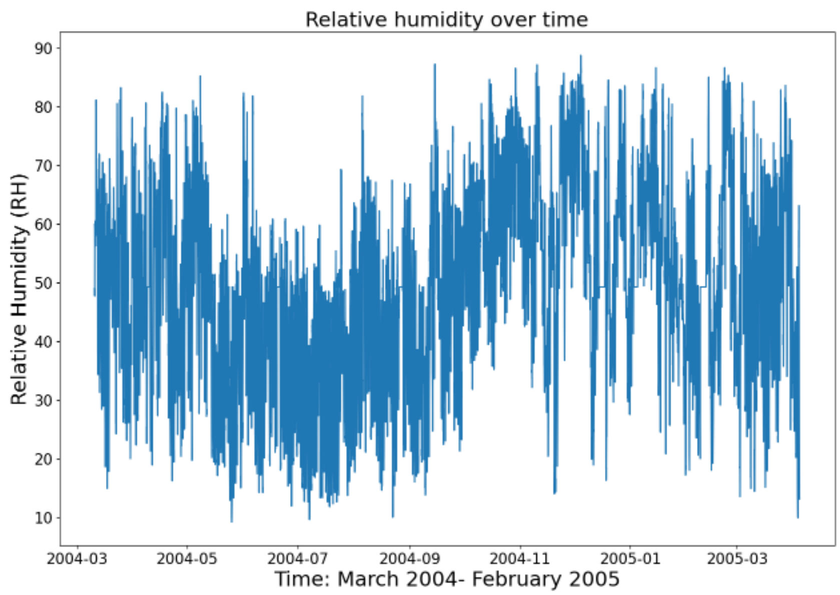
Table 1: Features details

Features	Description
Date	Date (DD/MM/YYYY)
Time	Time (HH.MM.SS)
CO(GT)	True hourly averaged concentration CO in mg/m^3 (reference analyzer)
PT08.S1(CO)	PT08.S1 (tin oxide) hourly averaged sensor response (nominally CO targeted)
NMHC(GT)	True hourly averaged overall Non Metanic HydroCarbons concentration in microg/m^3 (reference analyzer)
C6H6(GT)	True hourly averaged Benzene concentration in microg/m^3 (reference analyzer)
PT08.S2(NMHC)	PT08.S2 (titania) hourly averaged sensor response (nominally NMHC targeted)
NOx(GT)	True hourly averaged NOx concentration in ppb (reference analyzer)
PT08.S3(NOx)	PT08.S3 (tungsten oxide) hourly averaged sensor response (nominally NOx targeted)
NO2(GT)	True hourly averaged NO2 concentration in microg/m^3 (reference analyzer)
PT08.S4(NO2)	PT08.S4 (tungsten oxide) hourly averaged sensor response (nominally NO2 targeted)
PT08.S5(O3)	PT08.S5 (indium oxide) hourly averaged sensor response (nominally O3 targeted)
T	Temperature in Â°C
RH	Relative Humidity (%) **(Dependent Variable)**
AH	AH Absolute Humidity

As part of the pre-processing, First, I had to change the data type of some variables from object to float; second, I dropped the unnamed columns; third, I removed the null values. With the latter, the database reduced its figures by 114 rows. Finally, I realized that we have some -200 values in some features, so I replaced this with NaN as these are the missing values in all the UCI datasets. I use the mean to fill these values to solve this new NaN problem.

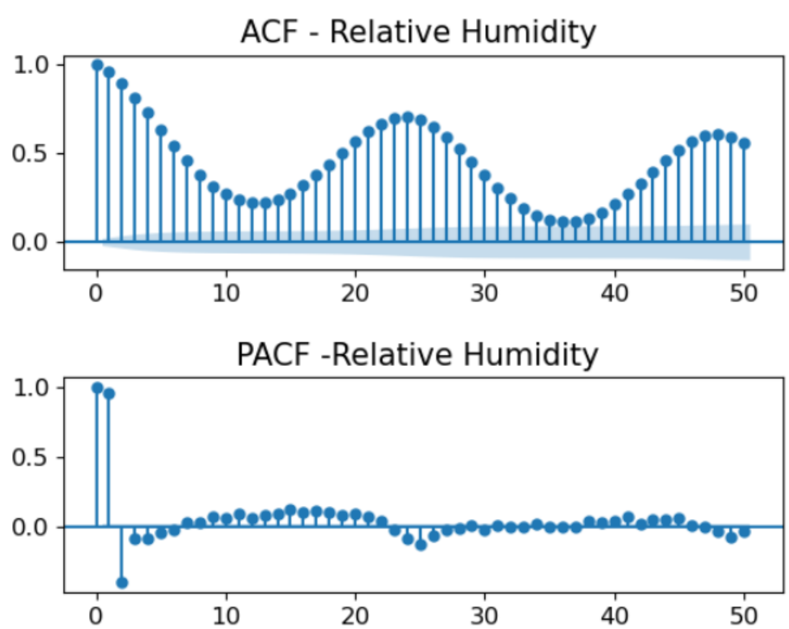
The following plot shows that the RH has a seasonal pattern because the numbers had almost the same values over the study period (March 2004 to February 2005). Also, we will develop an additive decomposition because the variation is almost at the same level.

Graph 1



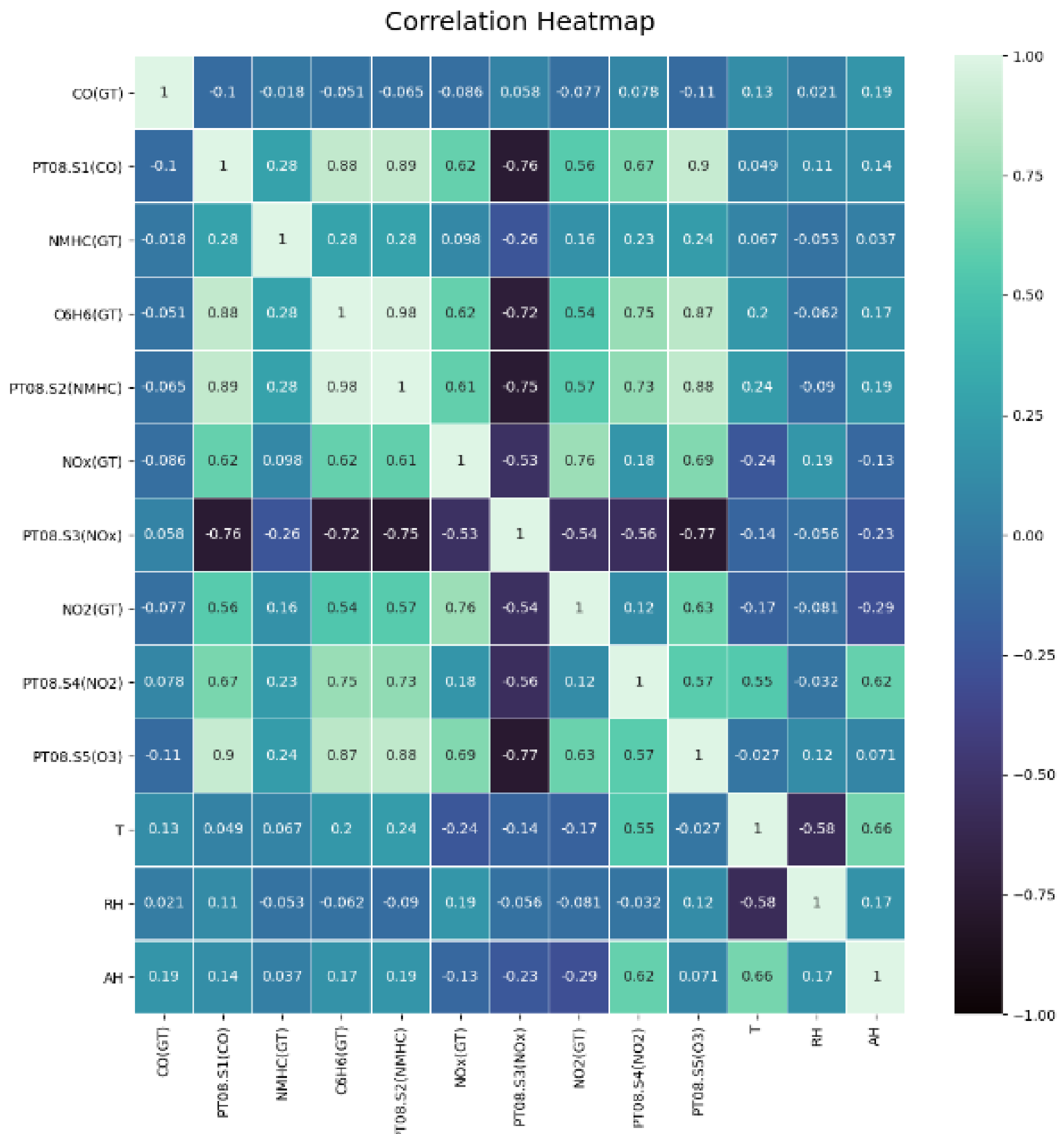
The dependent variable has strong seasonality upon the ACF/ PACF plot.

Graph 2



From the heatmap, it is possible to observe multicollinearity; this means that there is a high intercorrelation among several independent variables.

Graph 3



Then we are going to split the dataset into train set (80%) and test set (20%)

```
Train Shape: (7485, 13)
Test Shape: (1872, 13)
```

IV. Checking stationarity

To check stationarity, it is necessary to make use of the ADF and KPSS test. The ADF test determines if there is a unit root in the sequence: If the series is stationary, no unit root; otherwise, there is a unit root. Therefore, null hypothesis ($H0$) of the ADF test is a unit root. If the statistical significance of the test is less than the three confidence levels (10%, 5%, 1%), then there is a certainty (90%, 95, 99%) to reject the null hypothesis ($H0 = unit\ root = not\ stationary$). On the other hand, for the KPSS test it is necessary to check for the p-value and test if it is higher than 1%, 5%. So if there is this case we fail the reject the null hypothesis ($H0$= stationarity). This means that these series are stationary and there is a certainty of 99% and 95%.

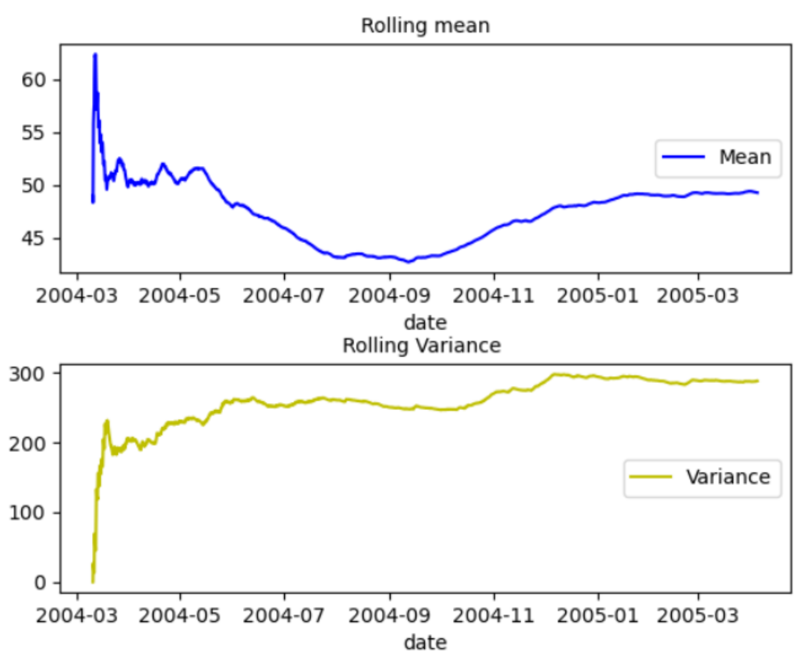
So for this dataset it can be observed that from the ADF statistics we reject the null hypothesis and the series has no unit root and is stationary. From the KPSS test the dataset is also stationary.

```
ADF Statistic: -7.391164
p-value: 0.000000
Critical Values:
    1%: -3.431
    5%: -2.862
    10%: -2.567
Reject the null hypothesis(Ho). Series has no unit root and is stationary
```

```
Results of KPSS Test:
Test Statistic             2.963095
p-value                    0.010000
LagsUsed                  52.000000
Critical Value (10%)       0.347000
Critical Value (5%)        0.463000
Critical Value (2.5%)      0.574000
Critical Value (1%)        0.739000
dtype: float64
```

From the rolling mean and variance, they are constant over time.

Graph 4



From the ACF/PACF plots, we have observed that the series has strong seasonality; this is why it is necessary to apply differencing.

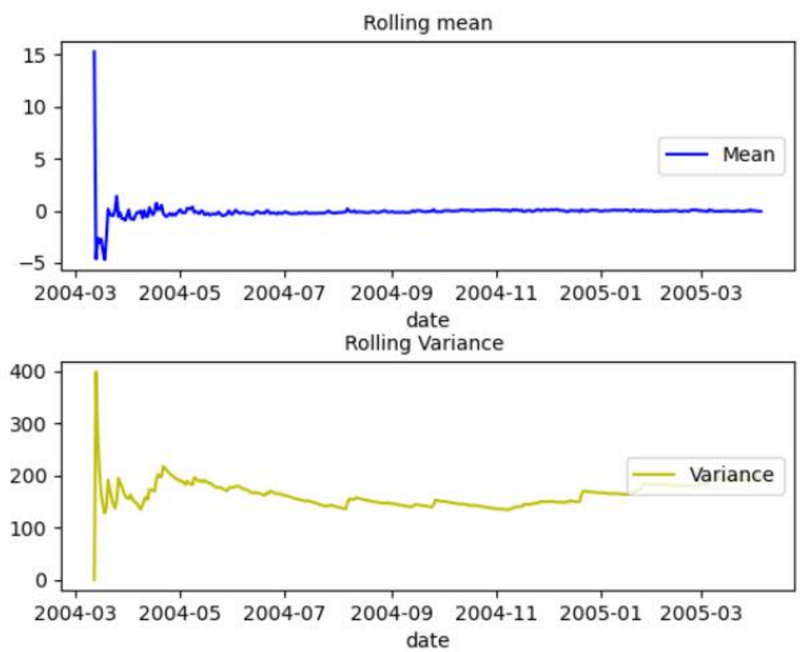
Seasonal differencing: Considering that the dataset is hourly, it is necessary to calculate a seasonal difference every 24 periods.
The ADF and KPSS tests revealed that the series is stationary for this seasonal difference.

```
ADF Statistic: -9.538226
p-value: 0.000000
Critical Values:
    1%: -3.448
    5%: -2.869
    10%: -2.571
Reject the null hypothesis(Ho). Series has no unit root and is stationary
```

```
Results of KPSS Test:
Test Statistic              0.117078
p-value                     0.100000
LagsUsed                   53.000000
Critical Value (10%)        0.347000
Critical Value (5%)         0.463000
Critical Value (2.5%)       0.574000
Critical Value (1%)         0.739000
```

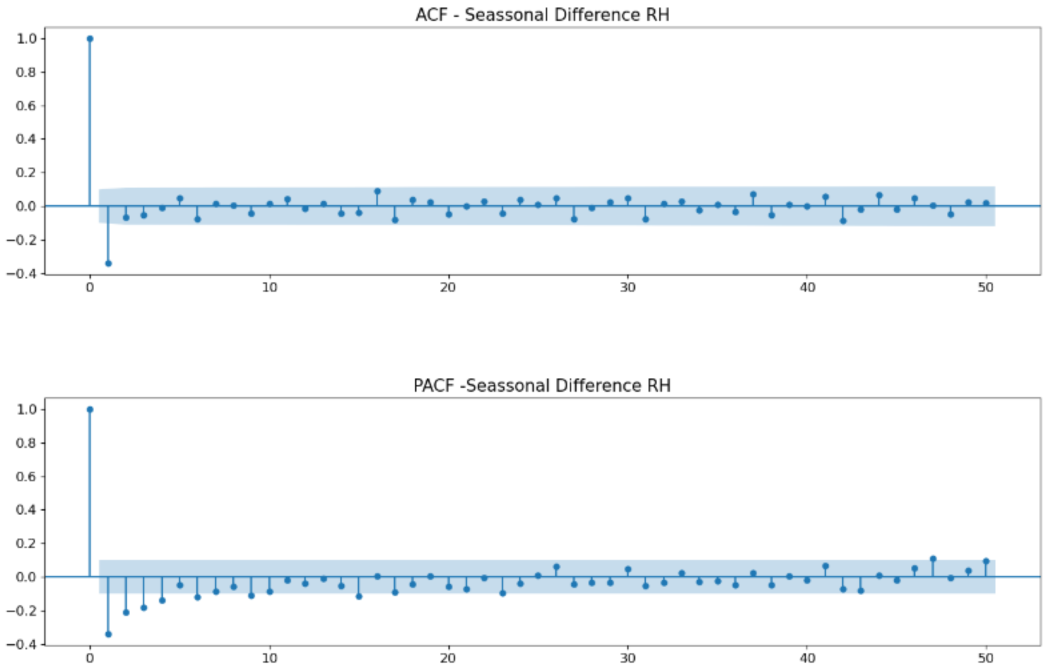
From the rolling mean and variance the series is almost constant over the time. The latter means that the series is stationary.

Graph 5: Rolling mean and variance after seasonal differencing



The ACF/ PACF plot of the difference data reveals that the series has no strong seasonality.

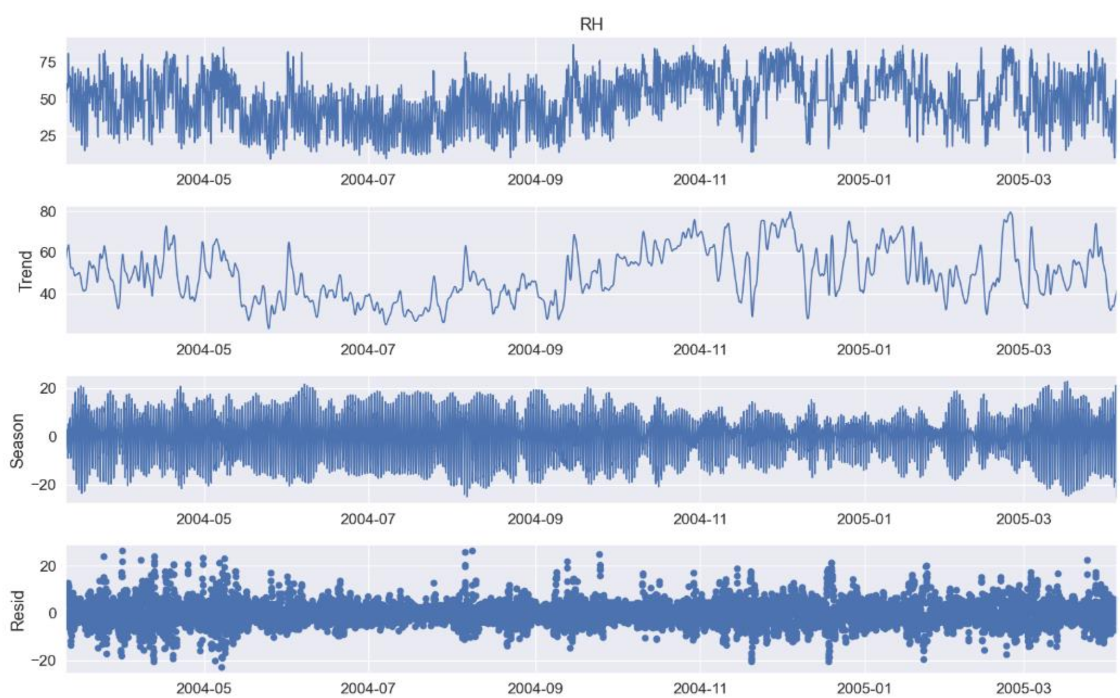
Graph 6: ACF/PACF after seasonal differencing



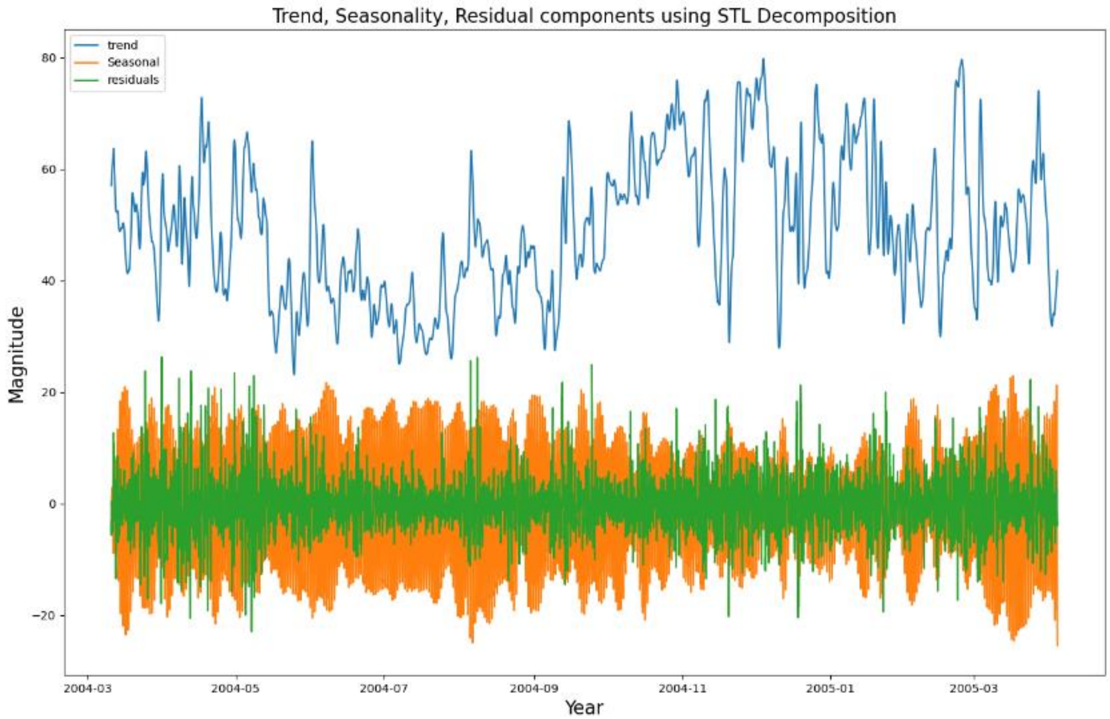
V. Time Series Decomposition

As we mentioned before, the time series decomposition will be the additive (Level+Trend+Seasonality+Noise).

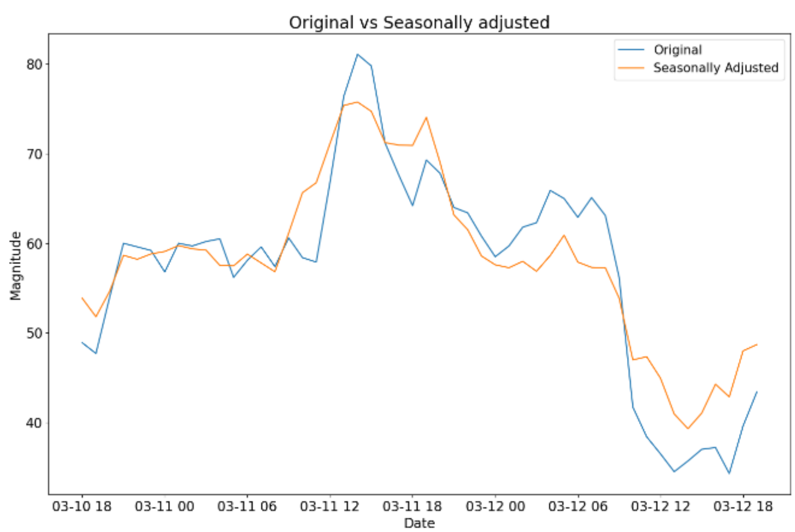
Graph 7: Series Decomposition



Graph 8: Trend, Seasonality, Residual components using STL decomposition



Graph 9: Original vs Seasonally adjusted




```
Strength of trend for Air quality dataset is 0.879

Strength of seasonality for Air quality dataset is 0.807
```

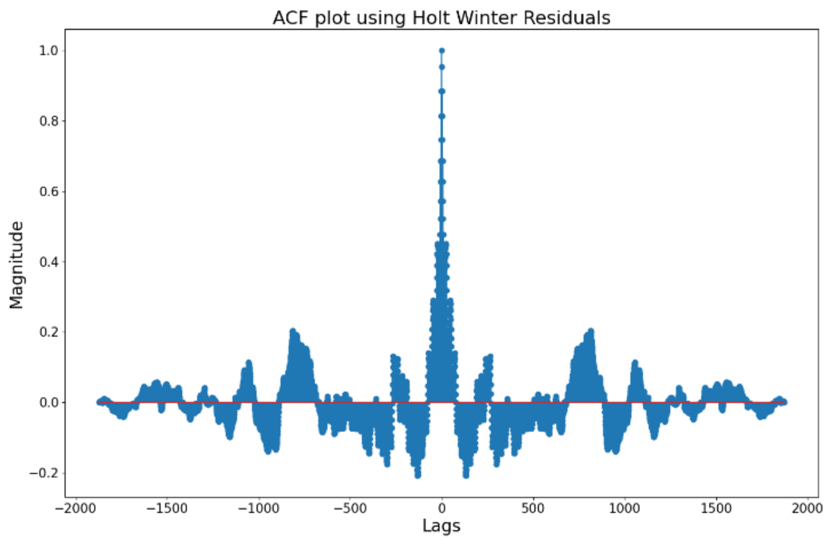
VI. Holt-Winters method

We will use the Holt-Winter method to find a perfect fit of the train and make the prediction in the test. Also, the test shows a higher MSE. This is why this model is not good because this does not capture all the fluctuation.

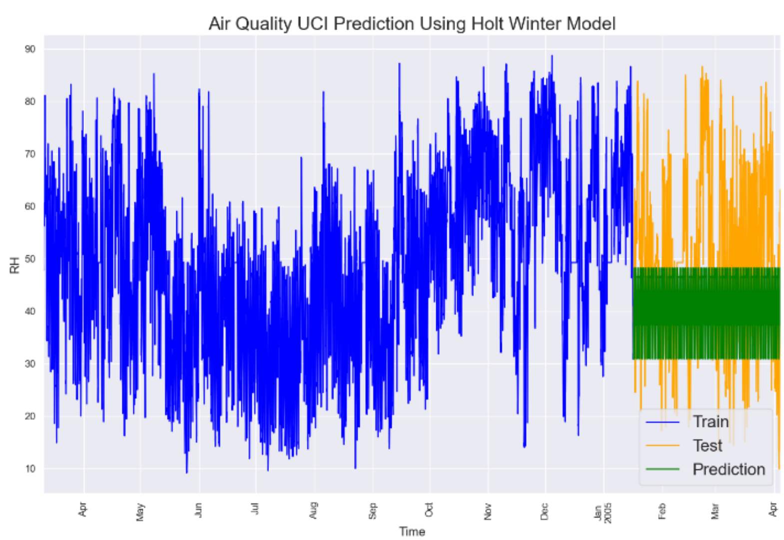
```
Holt Winter Method: MSE  of prediction errors (Train):  19.48152040573896
Holt Winter Method: RMSE of prediction errors (Train):  4.413787535183242
Holt Winter Method: MSE  of forecast errors (Test):  298.016539337517
Holt Winter Method: RMSE of forecast errors (Test):  17.26315554403415
Holt Winter Method: Variance of Residual of forecast (Test) : 199.2488647408672
Holt Winter Method: Mean of Residual of forecast (Test): 9.938192722857094
```

From the ACF plot and the figures above, we can see that the Holt-Winter is not the best model.

Graph 10: ACF using Holt Winter Residuals



Graph 11: Air Quality prediction using Holt Winter Model



VII. Feature selection

This section will calculate the SVD and the condition number and develop regression models. From these singular values, we can detect zero, which means that some features are highly correlated. The latter is supported by the high condition number, which also suggests the presence of multi-collinearity.

```
SingularValues =  [4.95272352e+10 1.70004381e+09 1.26169277e+09 3.76719664e+08
 1.01987005e+08 5.47698284e+07 4.19885038e+07 3.33440032e+07
 3.93091020e+06 2.57655299e+05 3.05449944e+04 3.26306567e+02]
The condition number is  12319.955385024095
```

First, we are going to observe the behavior of the model (p-values) with all the features, then we are going to remove them one by one in order of the highest p-values:

Regression model with all the features:

Table 2: OLS Regression

OLS Regression Results

Dep. Variable:	RH	R-squared (uncentered):	0.987
Model:	OLS	Adj. R-squared (uncentered):	0.987
Method:	Least Squares	F-statistic:	4.919e+04
Date:	Sat, 30 Apr 2022	Prob (F-statistic):	0.00
Time:	16:41:34	Log-Likelihood:	-23784.
No. Observations:	7485	AIC:	4.759e+04
Df Residuals:	7473	BIC:	4.767e+04
Df Model:	12		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
CO(GT)	0.0006	0.000	3.610	0.000	0.000	0.001
PT08.S1(CO)	0.0135	0.001	15.513	0.000	0.012	0.015
NMHC(GT)	0.0033	0.001	3.339	0.001	0.001	0.005
C6H6(GT)	-1.6980	0.037	-45.856	0.000	-1.771	-1.625
PT08.S2(NMHC)	0.0339	0.001	29.384	0.000	0.032	0.036
NOx(GT)	0.0157	0.001	21.564	0.000	0.014	0.017
PT08.S3(NOx)	0.0104	0.000	30.786	0.000	0.010	0.011
NO2(GT)	-0.0241	0.003	-7.061	0.000	-0.031	-0.017
PT08.S4(NO2)	0.0122	0.001	22.974	0.000	0.011	0.013
PT08.S5(O3)	-0.0017	0.001	-3.173	0.002	-0.003	-0.001
T	-2.3287	0.014	-169.999	0.000	-2.356	-2.302
AH	34.5833	0.321	107.714	0.000	33.954	35.213

Omnibus:	479.490	Durbin-Watson:	0.163
Prob(Omnibus):	0.000	Jarque-Bera (JB):	607.479
Skew:	0.607	Prob(JB):	1.22e-132
Kurtosis:	3.688	Cond. No.	1.23e+04

Removing PT08.S5(O3):

<u>Table 3: OLS Regression Removing PT08.S5(O3)</u>

```
                            OLS Regression Results
=======================================================================================
Dep. Variable:                     RH   R-squared (uncentered):                   0.987
Model:                            OLS   Adj. R-squared (uncentered):              0.987
Method:                 Least Squares   F-statistic:                          5.359e+04
Date:                Sat, 30 Apr 2022   Prob (F-statistic):                        0.00
Time:                        17:01:40   Log-Likelihood:                         -23789.
No. Observations:                7485   AIC:                                  4.760e+04
Df Residuals:                    7474   BIC:                                  4.768e+04
Df Model:                          11
Covariance Type:            nonrobust
================================================================================
                   coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
CO(GT)           0.0006      0.000      3.856      0.000       0.000       0.001
PT08.S1(CO)      0.0125      0.001     15.470      0.000       0.011       0.014
NMHC(GT)         0.0034      0.001      3.397      0.001       0.001       0.005
C6H6(GT)        -1.7052      0.037    -46.105      0.000      -1.778      -1.633
PT08.S2(NMHC)    0.0329      0.001     29.654      0.000       0.031       0.035
NOx(GT)          0.0158      0.001     21.570      0.000       0.014       0.017
PT08.S3(NOx)     0.0108      0.000     33.125      0.000       0.010       0.011
NO2(GT)         -0.0258      0.003     -7.642      0.000      -0.032      -0.019
PT08.S4(NO2)     0.0123      0.001     23.408      0.000       0.011       0.013
T               -2.3144      0.013   -178.782      0.000      -2.340      -2.289
AH              34.4307      0.318    108.395      0.000      33.808      35.053
==============================================================================
Omnibus:                      489.147   Durbin-Watson:                   0.163
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              619.196
Skew:                           0.617   Prob(JB):                    3.49e-135
Kurtosis:                       3.680   Cond. No.                     1.11e+04
==============================================================================
```

Removing NMHC(GT):

<u>Table 4: OLS Regression Removing PT08.S5(O3)</u>

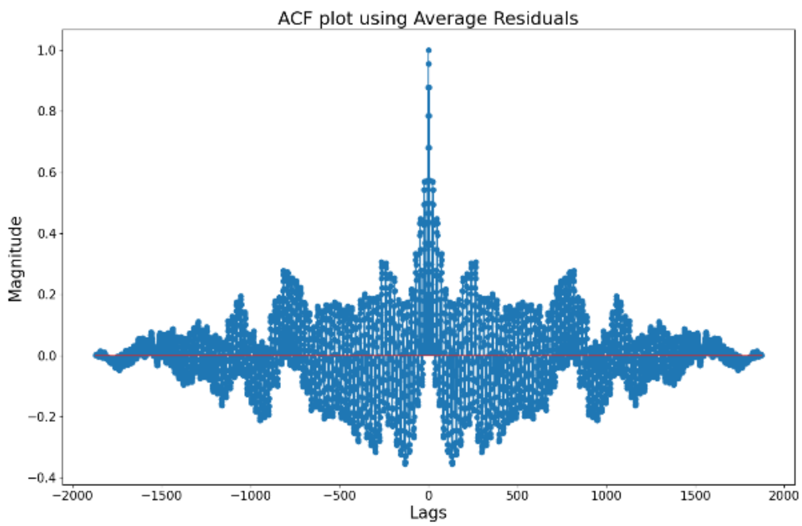
```
                            OLS Regression Results
=======================================================================================
Dep. Variable:                     RH   R-squared (uncentered):                   0.987
Model:                            OLS   Adj. R-squared (uncentered):              0.987
Method:                 Least Squares   F-statistic:                          5.887e+04
Date:                Sat, 30 Apr 2022   Prob (F-statistic):                        0.00
Time:                        17:05:23   Log-Likelihood:                         -23795.
No. Observations:                7485   AIC:                                  4.761e+04
Df Residuals:                    7475   BIC:                                  4.768e+04
Df Model:                          10
Covariance Type:            nonrobust
=================================================================================
                   coef    std err          t      P>|t|      [0.025      0.975]
---------------------------------------------------------------------------------
CO(GT)           0.0006      0.000      3.876      0.000       0.000       0.001
PT08.S1(CO)      0.0127      0.001     15.830      0.000       0.011       0.014
C6H6(GT)        -1.7151      0.037    -46.484      0.000      -1.787      -1.643
PT08.S2(NMHC)    0.0333      0.001     30.100      0.000       0.031       0.035
NOx(GT)          0.0156      0.001     21.357      0.000       0.014       0.017
PT08.S3(NOx)     0.0108      0.000     33.278      0.000       0.010       0.011
NO2(GT)         -0.0249      0.003     -7.416      0.000      -0.032      -0.018
PT08.S4(NO2)     0.0124      0.001     23.597      0.000       0.011       0.013
T               -2.3144      0.013   -178.661      0.000      -2.340      -2.289
AH              34.4295      0.318    108.315      0.000      33.806      35.053
==============================================================================
Omnibus:                      497.707   Durbin-Watson:                   0.163
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              631.717
Skew:                           0.624   Prob(JB):                    6.67e-138
Kurtosis:                       3.686   Cond. No.                     1.10e+04
==============================================================================
```

VIII. Base- Models

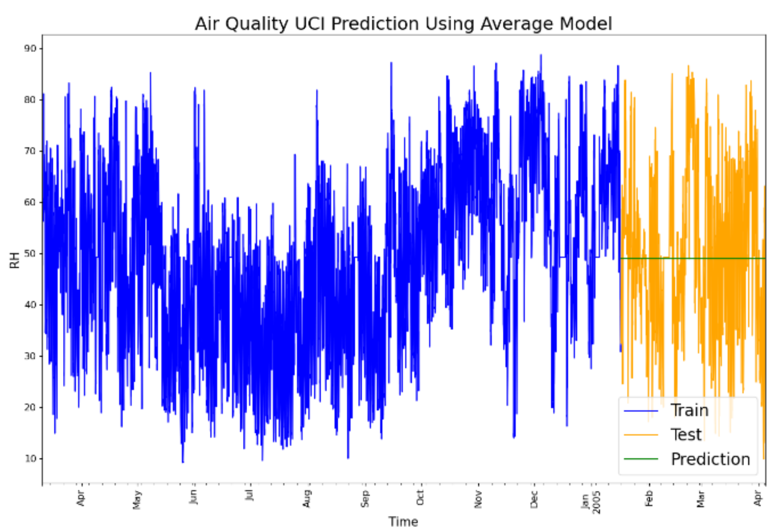
Average Model: From the RMSE we can conclude that the average is not a suitable model for this data, also the graph of the prediction is not accurate with the real data.

```
The MSE for Average model is :  261.8476
The RMSE for Average model is:  16.1817
The Variance of residual for Average model is:  260.3322
The Mean of residual for Average model is:  1.231
The Q value of Residual for Average model is:  310975.648
```

Graph 12: ACF plot using Average Residuals



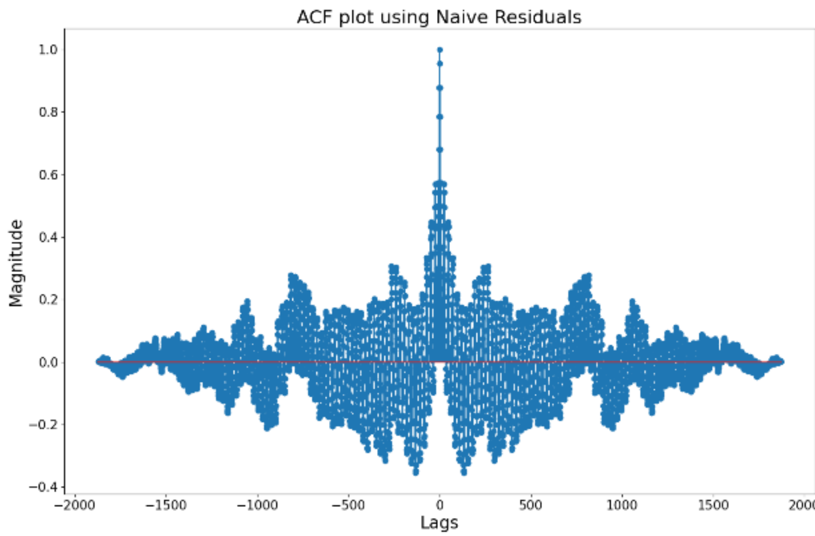
Graph 13: Air Quality using Average method



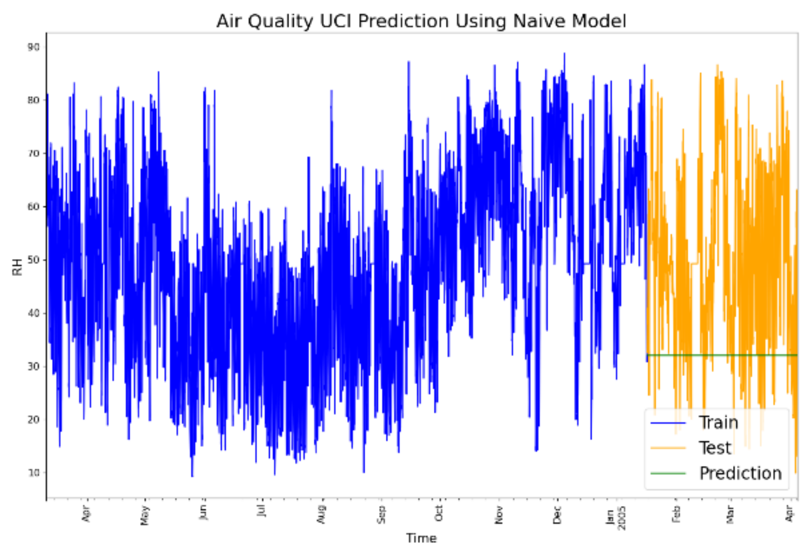
Naïve Model: The ACF plot is not according to white noise. Also, the RMSE shows that the Naïve shows better behavior than the average, but this is not enough.

```
The MSE for Naive Model is :  592.2617
The RMSE for Naive Model is:  24.3364
The Variance of residual for Naive Model is:  260.3322
The Mean of residual for Naive Model is:  18.2189
The Q value of Residual for Naive Model is:  310975.648
```

Graph 14: ACF plot using Naïve Residuals



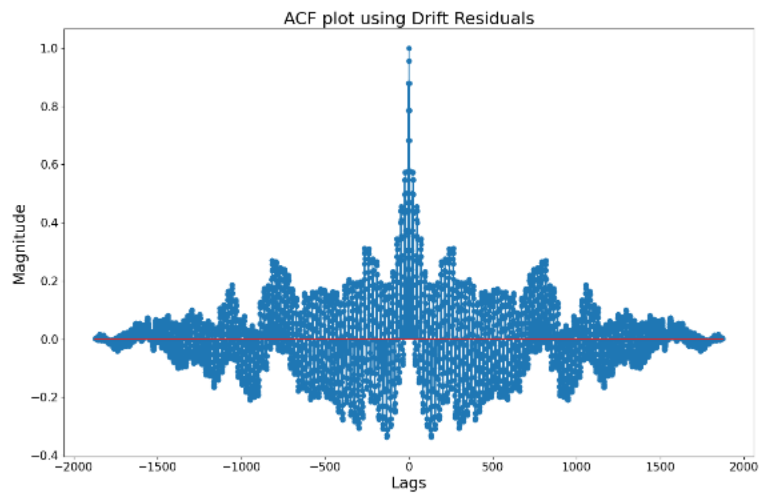
Graph 15: Air Quality prediction using Naïve Model



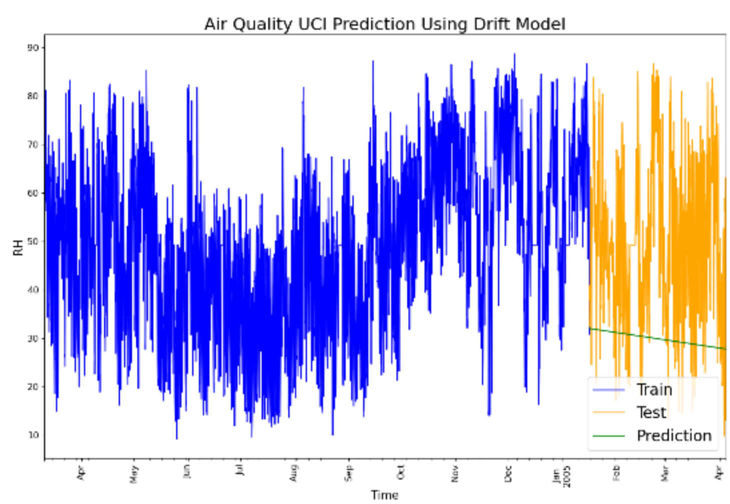
Drift Model: The ACF plot is not according to white noise. Also, the prediction is not accurate with the real data.

```
The MSE for Drift Model is :  675.7209
The RMSE for Drift Model is:  25.9946
The Variance of residual for Drift Model is:  262.262
The Mean of residual for Drift Model is:  20.3337
The Q value of Residual for Drift Model is:  305120.1339
```

Graph 16: ACF plot using drift Residuals



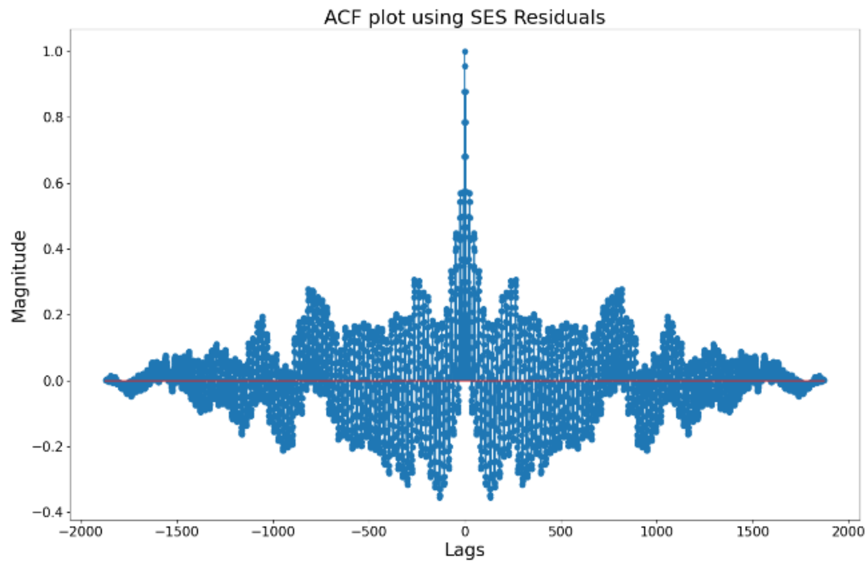
Graph 17: Air Quality Prediction using Drift Model



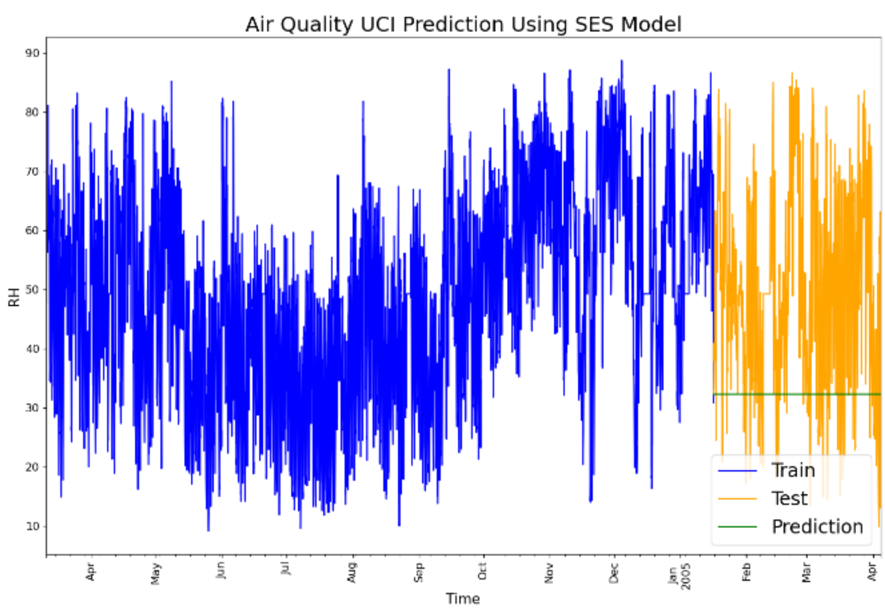
Simple Exponential Smoothing: The ACF plot is not according to white noise. Also, the prediction is not accurate with the real data.

```
The MSE for SES Model is :  582.2602
The RMSE for SES Model is:  24.1301
The Variance of residual for SES Model is:  260.3322
The Mean of residual for SES Model is:  17.9424
The Q value of Residual for SES Model is:  310975.648
```

Graph 18: ACF plot using SES Residuals



Graph 19: Air Quality Prediction Using SES model



IX. Multiple Linear Regression

The F-test is less than 0.05, which means that this model performs better than the null model. A similar conclusion is observed for all the p-values since they are lower than 0.05.

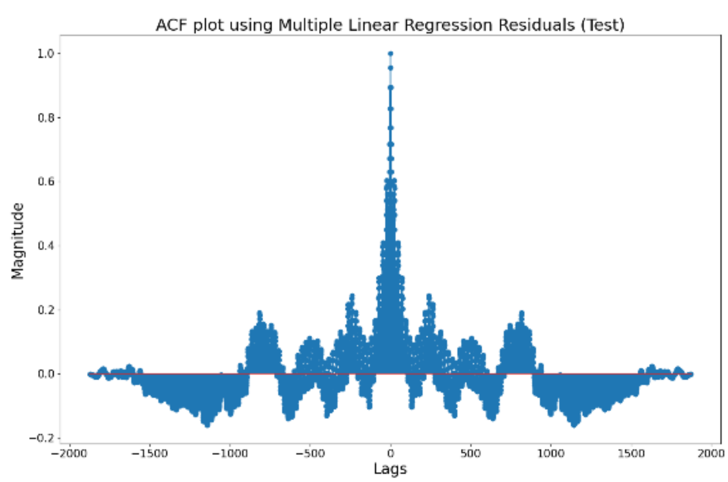
```
                                 OLS Regression Results
=======================================================================================
Dep. Variable:                     RH   R-squared (uncentered):                   0.987
Model:                            OLS   Adj. R-squared (uncentered):              0.987
Method:                 Least Squares   F-statistic:                          4.919e+04
Date:                Mon, 02 May 2022   Prob (F-statistic):                        0.00
Time:                        11:32:36   Log-Likelihood:                         -23784.
No. Observations:                7485   AIC:                                  4.759e+04
Df Residuals:                    7473   BIC:                                  4.767e+04
Df Model:                          12
Covariance Type:            nonrobust
=================================================================================
                    coef    std err          t      P>|t|      [0.025      0.975]
---------------------------------------------------------------------------------
CO(GT)            0.0006      0.000      3.610      0.000       0.000       0.001
PT08.S1(CO)       0.0135      0.001     15.513      0.000       0.012       0.015
NMHC(GT)          0.0033      0.001      3.339      0.001       0.001       0.005
C6H6(GT)         -1.6980      0.037    -45.856      0.000      -1.771      -1.625
PT08.S2(NMHC)     0.0339      0.001     29.384      0.000       0.032       0.036
NOx(GT)           0.0157      0.001     21.564      0.000       0.014       0.017
PT08.S3(NOx)      0.0104      0.000     30.786      0.000       0.010       0.011
NO2(GT)          -0.0241      0.003     -7.061      0.000      -0.031      -0.017
PT08.S4(NO2)      0.0122      0.001     22.974      0.000       0.011       0.013
PT08.S5(O3)      -0.0017      0.001     -3.173      0.002      -0.003      -0.001
T                -2.3287      0.014   -169.999      0.000      -2.356      -2.302
AH               34.5833      0.321    107.714      0.000      33.954      35.213
==============================================================================
Omnibus:                      479.490   Durbin-Watson:                   0.163
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              607.479
Skew:                           0.607   Prob(JB):                    1.22e-132
Kurtosis:                       3.688   Cond. No.                     1.23e+04
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[3] The condition number is large, 1.23e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

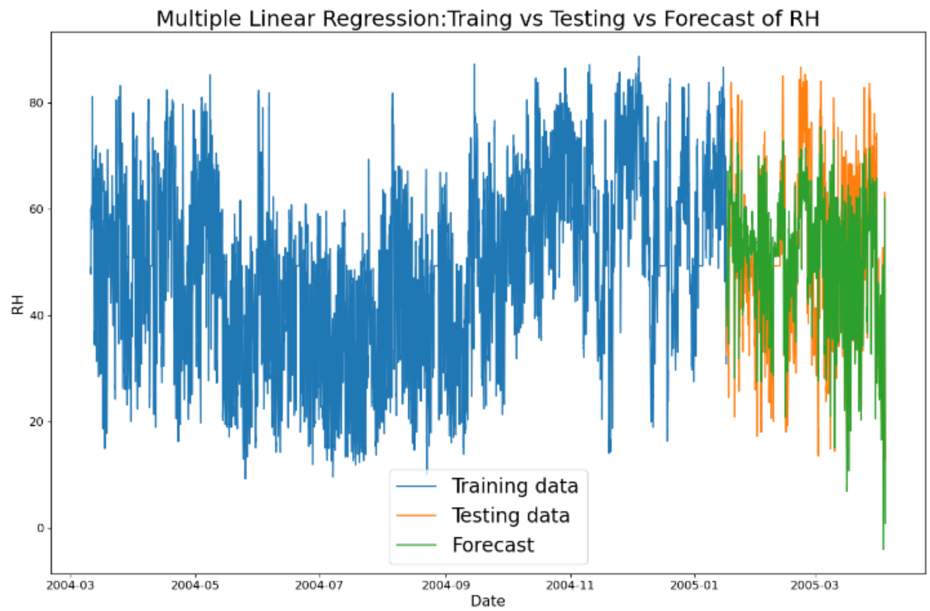
According to the RMSE, we can see that this is a good model with better figures than the others so far.

```
The MSE for Multiple Linear Regression Model is :  60.433
The RMSE for Multiple Linear Regression Model is:  7.7739
The Variance of residual for Multiple Linear Regression Model is:  60.0224
The Mean of residual for Multiple Linear Regression Model is:  -0.6408
The Q value of Residual for Multiple Linear Regression  Model is:  45559.2679
```

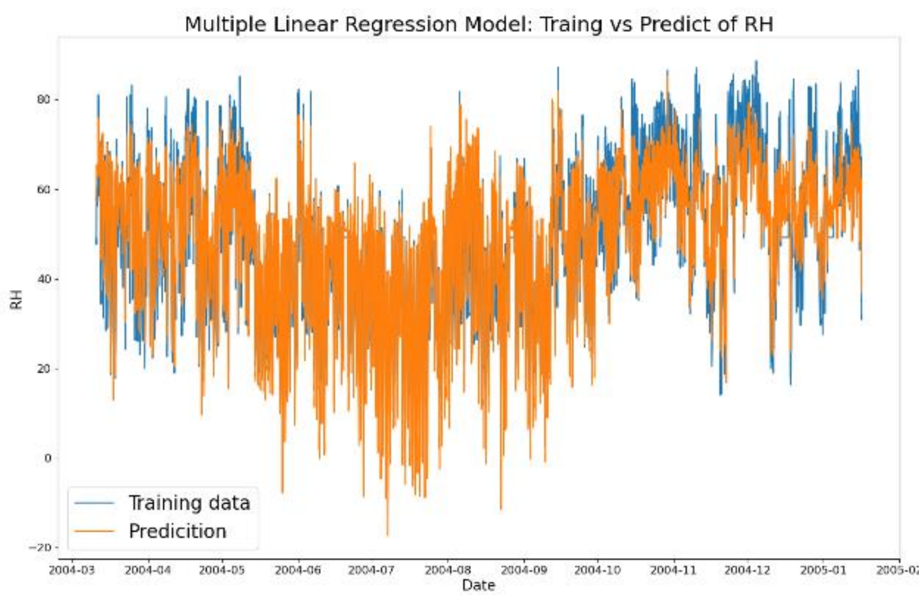
Graph 20: ACF plot using Multiple Linear Regression Residuals Test



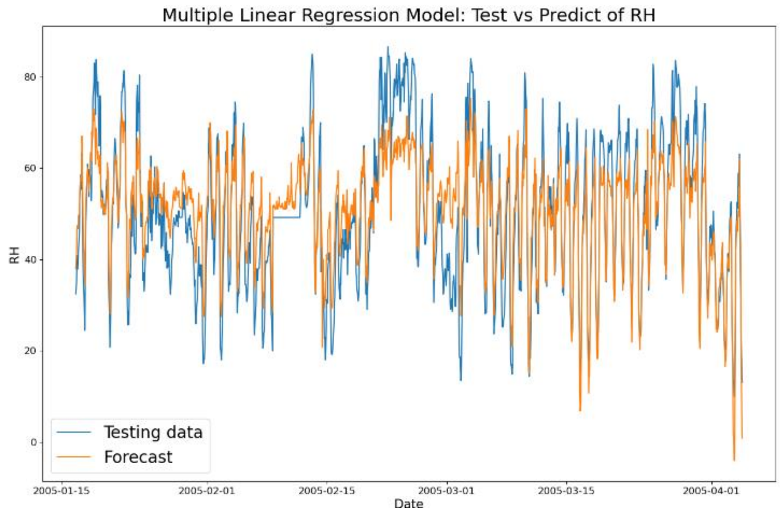
Graph 21: Multiple Linear Regression: Training vs testing Vs Forecast RH



Graph 22: Multiple Linear Regression Model: Training vs Predict of RH



Graph 23: Multiple Linear Regression Model: Test vs Predict of RH



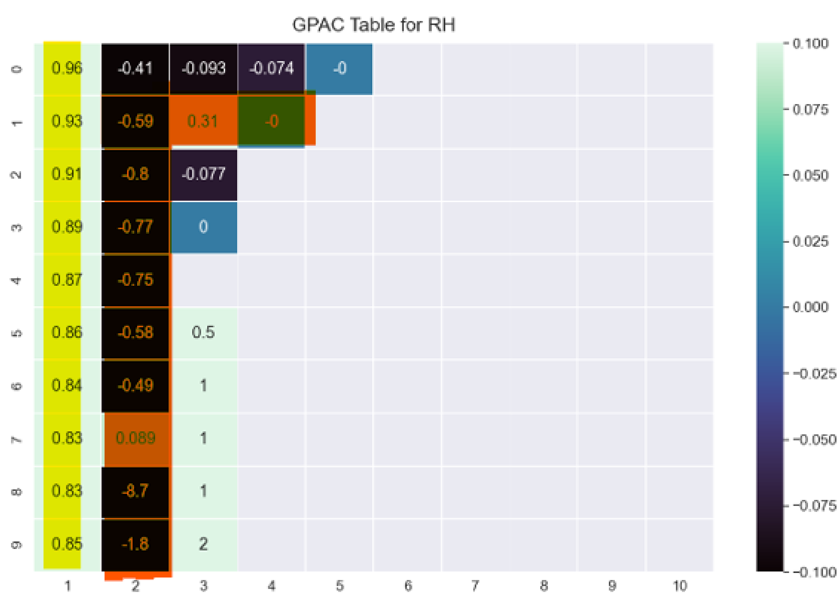
The graphs above show that the model's prediction is accurate with the actual data.

X. ARMA Models

To see the order of the ARMA models, we will use the GPAC table, and then the parameters of each model will be calculated.

A GPAC table with j=10 and k=10 is developed:

Table 5: GPAC table



From this GPAC, the ARMA orders obtained are (1,0) and (2,1).

Applying the chi-square test, we observed that the residuals from the detected ARMA orders are not white.

```
None of the identified ARMA orders pass the chi-squared test.

The residual is not white with n_a=1 and n_b=0
The residual is not white with n_a=2 and n_b=1
```

ARMA (1,0)

We can observe a description of the ARMA model below.

Also, from the confidence interval for each coefficient, we can observe no zeros on the confidence interval, so no simplification is needed.

Table 6: ARMA (1,0) Model result

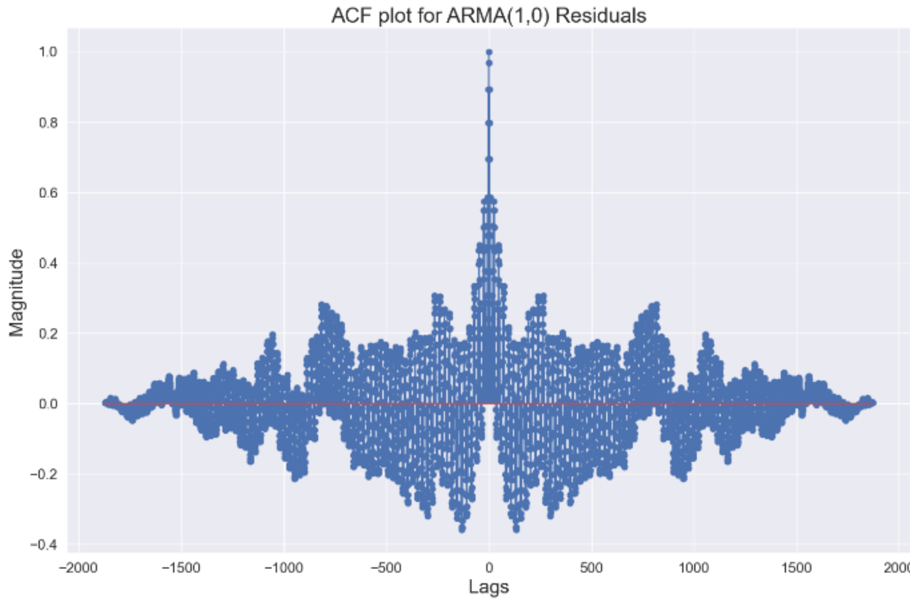
```
                              ARMA Model Results
==============================================================================
Dep. Variable:                     RH   No. Observations:                 9357
Model:                     ARMA(1, 0)   Log Likelihood              -27529.364
Method:                       css-mle   S.D. of innovations              4.586
Date:                Wed, 04 May 2022   AIC                          55062.729
Time:                        00:04:44   BIC                          55077.016
Sample:                    03-10-2004   HQIC                         55067.581
                         - 04-04-2005
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1.RH       0.9629      0.003    345.193      0.000       0.957       0.968
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            1.0385           +0.0000j            1.0385            0.0000
```

```
The MSE for ARMA(1, 0) model is:  986.9027
The RMSE for ARMA(1, 0) model is:  31.415
The Variance of residual for ARMA(1, 0) model is: 981.0166
The Mean of residual for ARMA(1, 0) model is: 2.4261
```

We can say that because the absolute value of the mean residual is more significant than 0.05, this is the case of a biased model.

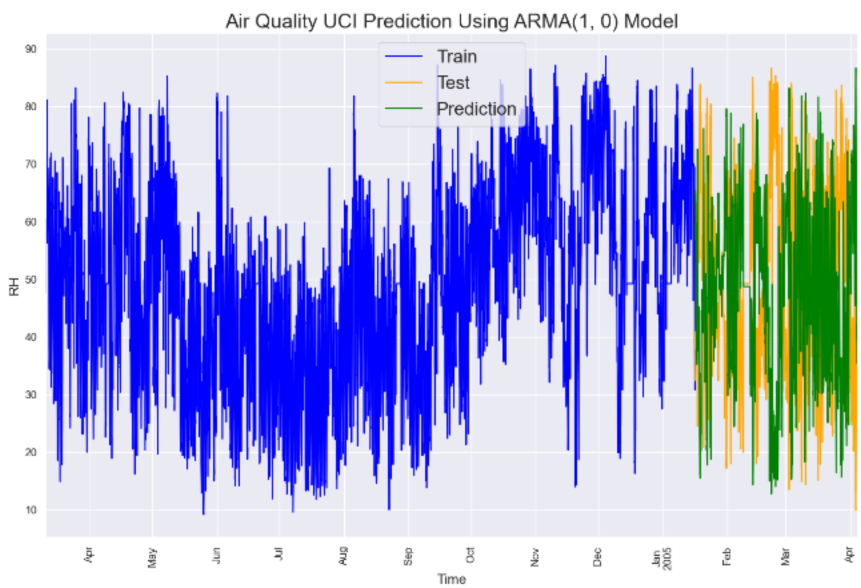
From the ACF plot we can see that this model does not works well.

Graph 24: ACF plot for ARMA(1,0) Residuals



Here we have the prediction plot for ARMA(1,0)

Graph 25: Air quality Prediction using ARMA(1,0) model



As we can see the parameter estimation using the LM algorithm is accurate with the real value.

```
                         PARAMETER ESTIMATED
================================================================================
LM - The AR coefficient a0 is: 0.9960090800559419
The AR coefficient a0 is: 0.995995649538989
```

```
                    Estimated Covariance Matrix
                           LM algorithm
==============================================================================
[[1.07117668e-06]]
==============================================================================
==============================================================================
                    Estimated Variance of Error
                           LM algorithm
==============================================================================
[[21.60334515]]
```

Also, the confidence interval includes the calculated coefficient.

```
                              Confidence Interval for Theta
==============================================================================
  θi-2*sqrt(cov(θ)ii)    |            θi            |   θi ± 2*sqrt(cov(θ)ii)
==============================================================================
  0.9939391267400481         0.9960090800559419         0.9980790333718357
==============================================================================
```

From the output, the zero/pole cancelation cannot be done because none of the roots are the same. So the model is still ARMA (1,0).

```
==============================================================================
                       Zero / Pole Cancelation
==============================================================================
Root of numerator "Zeros" :  []
Root of denominator "Poles" :  [-0.99600908]
```

ARMA (2,1): We can observe a description of the ARMA model below.

Also, from the confidence interval for each coefficient, we can observe no zeros on the confidence interval, so no simplification is needed.

<u>Table 7: ARMA (2,1) Model result</u>

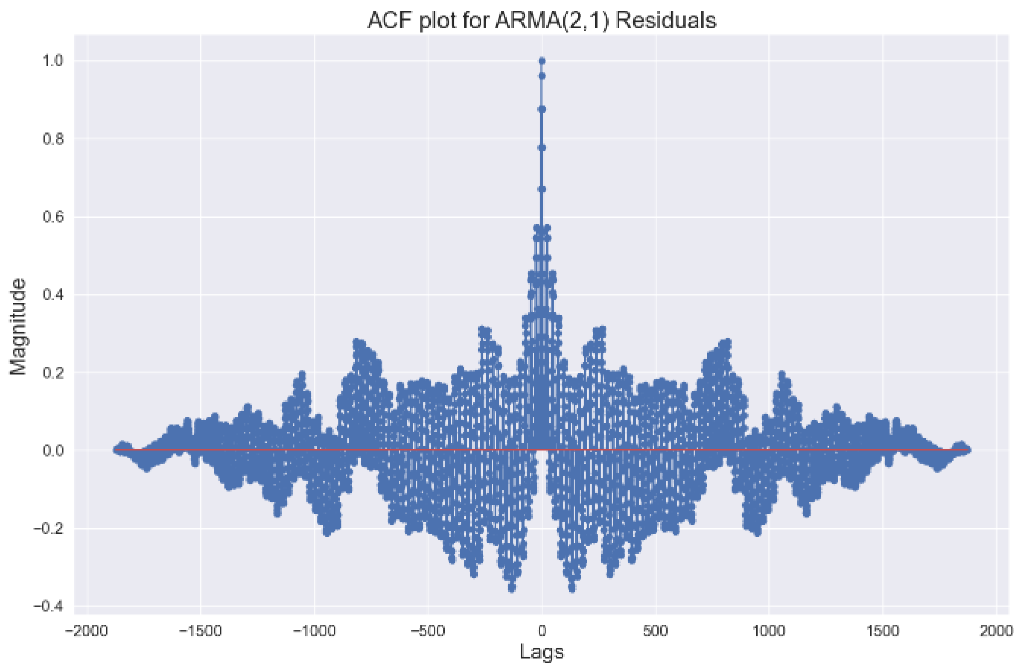
```
                              ARMA Model Results
==============================================================================
Dep. Variable:                     RH   No. Observations:                 9357
Model:                     ARMA(2, 1)   Log Likelihood              -26603.418
Method:                       css-mle   S.D. of innovations              4.154
Date:                Wed, 04 May 2022   AIC                          53214.837
Time:                        00:12:15   BIC                          53243.412
Sample:                    03-10-2004   HQIC                         53224.542
                         - 04-04-2005
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1.RH       1.6266      0.019     85.550      0.000       1.589       1.664
ar.L2.RH      -0.6685      0.018    -36.681      0.000      -0.704      -0.633
ma.L1.RH      -0.3351      0.025    -13.540      0.000      -0.384      -0.287
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            1.2166           -0.1258j            1.2230           -0.0164
AR.2            1.2166           +0.1258j            1.2230            0.0164
MA.1            2.9844           +0.0000j            2.9844            0.0000
-----------------------------------------------------------------------------
```

```
The MSE for ARMA(2, 1) model is:  1003.0689
The RMSE for ARMA(2, 1) model is:  31.6713
The Variance of residual for ARMA(2, 1) model is: 997.3603
The Mean of residual for ARMA(2, 1) model is: 2.3893
```

We can say that because the absolute value of the mean residual is more significant than 0.05, this is the case of a biased model.

From the ACF plot we can see that this model does not works well

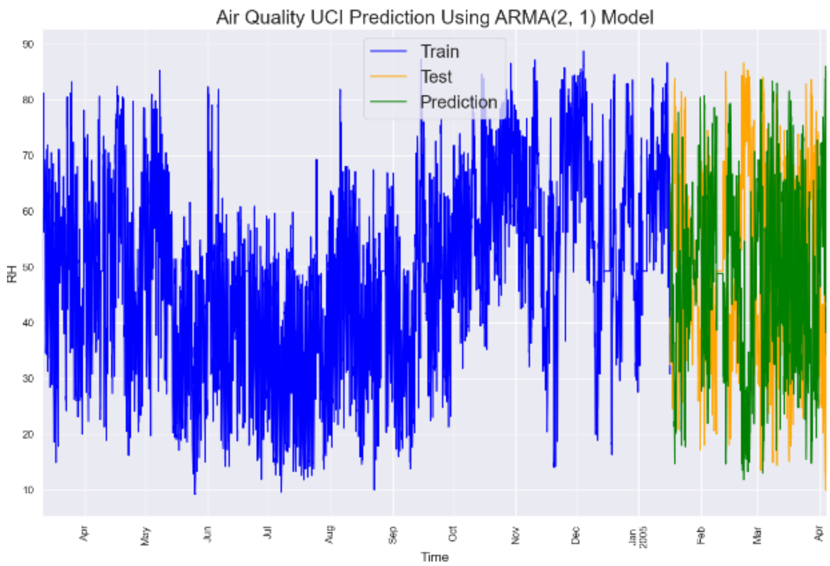
Graph 26: ACF plot ARMA (2,1) residuals




```
Estimated covariance matrix for n_a = 2 and n_b = 1:
          ar.L1.RH  ar.L2.RH  ma.L1.RH
ar.L1.RH  0.000362 -0.000345 -0.000430
ar.L2.RH -0.000345  0.000332  0.000409
ma.L1.RH -0.000430  0.000409  0.000612

Estimated variance of error for n_a = 2 and n_b = 1:
17.255173716438684
```

Graph 27: Air Quality prediction using ARMA(2,1) model



As we can see, the estimation of the LM algorithm's parameters is accurate with the real values.

```
                           PARAMETER ESTIMATED
================================================================================
LM - The AR coefficient a0 is: 1.5249896053962666
LM - The AR coefficient a1 is: -0.5298550819910275
LM - The MA coefficient b0 is: -0.1843455766129109
The AR coefficient a0 is: 1.5190248961591442
The AR coefficient a1 is: -0.5239233881532351
The MA coefficient b0 is: -0.17136892003716545
```

```
                         Estimated Covariance Matrix
                                LM algorithm
=================================================================================
[[ 0.00065341 -0.00065055  0.0006993 ]
 [-0.00065055  0.00064831 -0.00069609]
 [ 0.0006993  -0.00069609  0.00087758]]
=================================================================================
```

```
                       Estimated Variance of Error
                                LM algorithm
================================================================================
[[18.52121036]]
================================================================================
```

The estimated parameters are inside the confidence interval.

```
                       Confidence Interval for Theta
================================================================================
   θi-2*sqrt(cov(θ)ii)   |            θi            |   θi ± 2*sqrt(cov(θ)ii)
================================================================================
   1.4738657635138719         1.5249896053962666         1.5761134472786613
   -0.5807788982518889        -0.5298550819910275        -0.47893126573016614
   -0.24359364107571463       -0.1843455766129109        -0.12509751215010717
================================================================================
```

From the output, the zero/pole cancelation cannot be done because none of the roots are the same. So the model is still ARMA(2,1).

```
                        Zero / Pole Cancelation
================================================================================
Root of numerator "Zeros" :  [0.18434558]
Root of denominator "Poles" :  [0.98952617 0.53546343]
```

XI. SARIMA (0,0,0) x (0,1,1,24)

After the seasonal differencing of 24 periods, the rolling mean and rolling variance is not diverge, so it is stationary. Then, by looking at the ACF/PACF plots, the ACF is cut-off at lag 1, and the PACF is tailing-off, which indicates an MA process. This explains why we will test the SARIMA with AR(0) component and MA(1) and 24 seasonal periods.

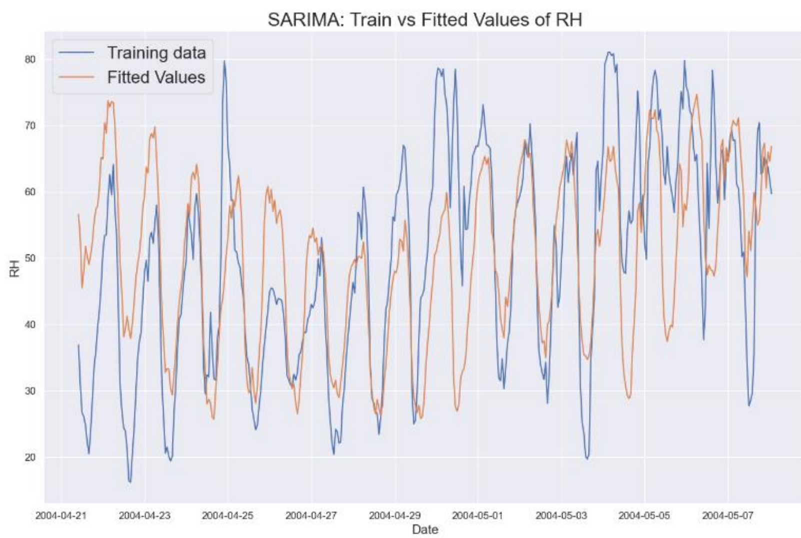
Table 8: SARIMA Model result

```
                                     SARIMAX Results
==================================================================================
Dep. Variable:                              RH   No. Observations:            7485
Model:             SARIMAX(0, 1, [1], 24)   Log Likelihood          -28612.806
Date:                     Wed, 04 May 2022   AIC                      57229.612
Time:                             00:42:17   BIC                      57243.447
Sample:                         03-10-2004   HQIC                     57234.365
                              - 01-16-2005
Covariance Type:                       opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ma.S.L24      -0.6081      0.008    -73.520      0.000      -0.624      -0.592
sigma2       125.2868      1.544     81.151      0.000     122.261     128.313
===================================================================================
Ljung-Box (L1) (Q):                 6459.24   Jarque-Bera (JB):               892.57
Prob(Q):                               0.00   Prob(JB):                         0.00
Heteroskedasticity (H):                1.12   Skew:                             0.33
Prob(H) (two-sided):                   0.00   Kurtosis:                         4.56
===================================================================================
```

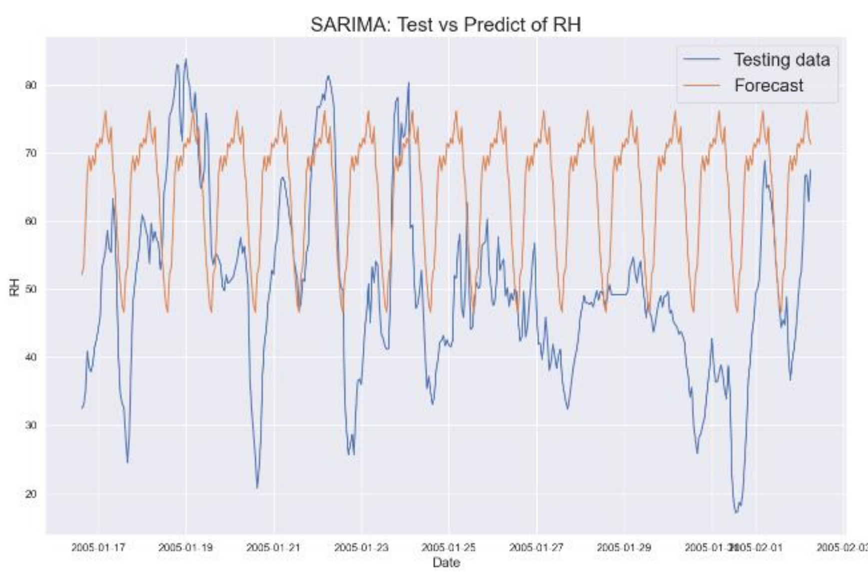
```
The MSE for SARIMA Model is :  418.8581
The RMSE for SARIMA Model is:  20.466
The Variance of residual for SARIMA Model is:  207.8946
The Mean of residual for SARIMA Model is:  -14.5246
The Variance of residual for SARIMA Model (Train) is:  137.6386
The Mean of residual for SARIMA Model (Train) is:  0.2366
The Q value of Residual for SARIMA Model is:  178853.9642
```

From the following plots, we have the SARIMA 1-step and h-step prediction, and it is possible to observe that the values are not too accurate. Also, according to the MSE 418.85, this value is high; this is why this is not a good model.

Graph 28: SARIMA: Train vs Fitted values of RH



Graph 29: SARIMA: Test vs Predicted of RH



XII.Models' comparison

We just summary all the ten models developed along this project. To choose the best model we have to look at the MSE to be the lower, so for this project the best one is the multiple linear regression.

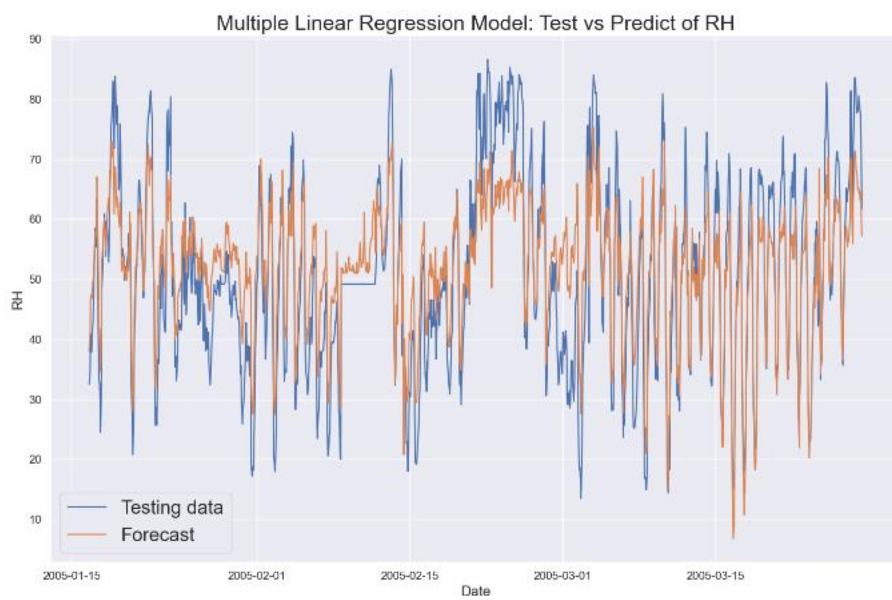
Table 3: Models Summary

BASE MODEL COMPARISON

Model	MSE	RMSE	Residual Mean	Residual Variance	Train Residual Mean	Train Residual Variance	Q Value
Average Model	261.847603	16.181706	1.231013	260.332210	-2.269931e-09	294.757911	310975.648032
Naive Model	592.261704	24.336428	18.218932	260.332210	1.698792e+01	294.757911	310975.648032
Drift Model	675.720932	25.994633	20.333690	262.261970	2.544018e+01	367.999923	305120.133891
Simple Exponential Smoothing Model	582.260222	24.130069	17.942352	260.332210	-3.701496e-03	34.940240	310975.648032
Holt Winter Model	298.016539	17.263156	9.938193	199.248865	-5.502696e-02	19.478492	174779.944494
Multiple Linear Regression Model	60.433018	7.773868	-0.640819	60.022369	1.103621e-01	33.677514	45559.267884
ARMA(1, 0) Model	986.902672	31.415007	2.426130	981.016567	2.185490e-03	1115.137897	316119.924675
ARMA(2, 1) Model	1003.068885	31.671263	2.389274	997.360256	9.273044e-04	1125.326443	315440.233711
SARIMA (0, 0, 0) (0, 1, 1, 24) Model	418.858124	20.466024	-14.524584	207.894582	2.365511e-01	137.638637	178853.964200

As we can see from the following plot, the Multiple Linear Regression model has the best accuracy with the data.

Graph 30: Multiple Linear Regression Model: Test vs Predict of RH



According to this the equation for the model would be

$$Y = 0.0006 * CD(GT) + 0.0135 * PT08.S1(CO) + 0.0033 * NMHC(GT) - 1.6980 * C6H6(G) + 0.0339 * PT08.S2(NMHC) + 0.0157 * NOx(GT) + 0.0104 * PT08.S3(N0x) - 0.0241 * NO2(GT) + 0.0122 * PT08.S4(NO2) - 0.0017 * PT08.S4(NO2) - 2.3287 * T + 34.5833 * AH$$

XIII. Conclusion

In conclusion, we have built ten models and tested all their characteristics. Therefore, we can conclude that the best model is the multiple linear regression due to highlighted attributes.

XIV. Appendix

XV.

```
from requests import get
from io import BytesIO
from zipfile import ZipFile
from datetime import datetime
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.seasonal import STL
from statsmodels.tsa.holtwinters import SimpleExpSmoothing
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from pylab import rcParams
from pandas.plotting import register_matplotlib_converters
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error

import os
import pandas as pd
import statsmodels.api as sm
import statsmodels.tsa.holtwinters as ets
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
from scipy.stats import chi2
from pylab import rcParams

from Final_toolbox import *

warnings.filterwarnings("ignore")
register_matplotlib_converters()




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

# if "AirQualityUCI" not in os.listdir():
#     request = get('https://archive.ics.uci.edu/ml/machine-learning-
databases/00360/AirQualityUCI.zip')
#     zip_file = ZipFile(BytesIO(request.content))
#     zip_file.extractall()
# print('\n')
df = pd.read_csv("AirQualityUCI.csv", sep =
```

```
';',infer_datetime_format=True)

#===========================================================================
============================
# Question 6
# a. Pre-processing dataset: Dataset cleaning for missing observation.
#    You must follow the data cleaning techniques for time series
dataset.
#===========================================================================
============================

# Print the head of the dataset
print(df.head(10))

# Description of the data
df.describe()
df.shape #9471 rows and 17 colums

# Summary of dataframe
print('Summary of Dataframe:\n',df.info)

# Changing the datatype from object to float: number that is not an
integer
df['CO(GT)'] = df['CO(GT)'].str.replace(',', '.').astype(float)
df['C6H6(GT)'] = df['C6H6(GT)'].str.replace(',','.').astype(float)
df['T'] = df['T'].str.replace(',', '.').astype(float)
df['RH'] = df['RH'].str.replace(',', '.').astype(float)
df['AH'] = df['AH'].str.replace(',', '.').astype(float)

# Drop the Unnamed columns
df = df.drop(['Unnamed: 15','Unnamed: 16'], axis = 1)
# Null values per feature
print(df.isnull().sum())
# Removing null values
null_data = df[df.isnull().any(axis=1)] #all null values
print(null_data.head())
df= df.dropna()
print('Shape after null values:\n',df.shape) #(9357, 15)
# Replacing -200 with nan
df = df.replace(-200,np.nan)
print(df.isnull().sum())
# Appending date and time
print(df.index)
df.loc[:,'Datetime'] = df['Date'] + ' ' + df['Time']
DateTime = []
for x in df['Datetime']:
    DateTime.append(datetime.strptime(x,'%d/%m/%Y %H.%M.%S'))
datetime = pd.Series(DateTime)
df.index = datetime
# print(df.head())
# print('AFTER',df.dtypes)
df = df.replace(-200, np.nan)

# Process for NaN values: fill this NaN values with the mean
# Creating processed dataframe
print(df.isnull().sum())
print(df.head)
```

```python
# SD = df['Date']
# ST = df['Time']
S0 = df['CO(GT)'].fillna(df['PT08.S1(CO)'].mean())
S1 = df['PT08.S1(CO)'].fillna(df['PT08.S1(CO)'].mean())
S2 = df['NMHC(GT)'].fillna(df['NMHC(GT)'].mean())
S3 = df['C6H6(GT)'].fillna(df['C6H6(GT)'].mean())
S4 = df['PT08.S2(NMHC)'].fillna(df['PT08.S1(CO)'].mean())
S5 = df['NOx(GT)'].fillna(df['NOx(GT)'].mean())
S6 = df['PT08.S3(NOx)'].fillna(df['PT08.S1(CO)'].mean())
S7 = df['NO2(GT)'].fillna(df['NO2(GT)'].mean())
S8 = df['PT08.S4(NO2)'].fillna(df['PT08.S1(CO)'].mean())
S9 = df['PT08.S5(O3)'].fillna(df['PT08.S1(CO)'].mean())
S10 = df['T'].fillna(df['T'].mean())
S11 = df['RH'].fillna(df['RH'].mean())
S12 = df['AH'].fillna(df['AH'].mean())
print('Handling nan with mean\n',df.isnull().sum())
print('\n')

#This values does not have any NaN values
df = pd.DataFrame({'CO(GT)':S0,'PT08.S1(CO)':S1,'NMHC(GT)':S2,
'C6H6(GT)':S3, 'PT08.S2(NMHC)':S4, 'NOx(GT)':S5,
                   'PT08.S3(NOx)':S6, 'NO2(GT)':S7,  'PT08.S4(NO2)':S8,
'PT08.S5(O3)':S9, 'T':S10, 'RH':S11, 'AH':S12 })


print("Shape after preprocessing:\n",df.shape) #(9357, 13)

#======================================================================
===========================
# Question 6
# b. Plot of the dependent variable versus time.
#======================================================================
===========================
# Created Dataframe for Dependent variable and time
df_rh = pd.DataFrame({'RH':S11})

# df.to_csv("AirQuality_processed_rh.csv")
print('Dataframe for Dependent variable and time\n',df_rh.head())

plt.figure(figsize=(15,10))
plt.plot(df_rh,  label = 'RH')
plt.xlabel('Time: March 2004- February 2005', fontsize=22)
plt.ylabel('Relative Humidity (RH)', fontsize=22)
plt.title('Relative humidity over time', fontsize=22)
plt.tick_params(axis='x', labelsize=16)
plt.tick_params(axis='y', labelsize=16)
#plt.legend(loc='best')
plt.show()


#======================================================================
===========================
# Question 6
# c. ACF/PACF of the dependent variable.
#======================================================================
===========================
```

```
ACF_PACF_Plot(y = df_rh,
              title1 = 'ACF - Relative Humidity ',
              title2 = 'PACF -Relative Humidity ',
              nlags = 50)


#========================================================================
===========================
# Question 6
# d. Correlation Matrix with seaborn heatmap with the Pearson’s
correlation coefficient.
#========================================================================
===========================

plt.figure()
fig, ax = plt.subplots(figsize=(12,12))
heatmap=sns.heatmap(df.corr(), vmin=-1, vmax=1,
annot=True,cmap="mako",linewidth=0.3, linecolor='w')
heatmap.set_title('Correlation Heatmap', fontdict={'fontsize':18},
pad=17);
plt.show()

#========================================================================
===========================
# Question 6
# e. Split the dataset into train set (80%) and test set (20%).
#========================================================================
===========================

# Always consider suffle=False, so the time dependency is not lost:
# Split the training and testing in 80% and 20%

train, test = train_test_split(df, shuffle=False, test_size = 0.2)

#Train and test Shape
print('Train Shape:',train.shape) #(7485, 13)
print('Test Shape:',test.shape) #(1872, 13)



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

test_result = adfuller(df['RH'])

# ADF TEST
```

```
ADF_Cal(df['RH'])

# KPSS TEST
KPSS_test(df["RH"])

# ROLLING MEAN AND VAR
rolling_mean, rolling_var = cal_rolling_mean_var(df["RH"],start="2004-
03-10 18:00:00", end="2005-04-04 14:00:00")

###### 1st Difference:
# DIFFERENCING RH variable
#difference_RH = differencing(df['RH'], 1)

#seasonal_diff_24
difference_RH_seasonal = differencing_l(df['RH'], 24)

# ADF TEST
#ADF_Cal(difference_RH)
ADF_Cal(difference_RH_seasonal)

# KPSS TEST
#KPSS_test(difference_RH)
KPSS_test(difference_RH_seasonal)

# ROLLING MEAN AND VAR
#rolling_mean, rolling_var =
cal_rolling_mean_var(difference_RH,start="2004-03-10 18:00:00",
end="2005-04-04 14:00:00")
rolling_mean, rolling_var =
cal_rolling_mean_var(difference_RH_seasonal,start="2004-03-10
18:00:00", end="2005-04-04 14:00:00")


# ACF and PACF of DIFFERENCE_RH
#ACF_PACF_Plot(difference_RH, 'ACF Difference of RH variable', 'PACF
Difference of RH variable' , 50)
ACF_PACF_Plot(y = difference_RH_seasonal,
              title1 = 'ACF - Seassonal Difference RH',
              title2 = 'PACF -Seassonal Difference RH',
              nlags = 50)

# ####### 2nd Difference
# # DIFFERENCING RH variable
# difference_2_RH = differencing(difference_RH, 1)
# # ADF TEST
# ADF_Cal(difference_2_RH)
#
# # KPSS TEST
# KPSS_test(difference_2_RH)
#
# # ROLLING MEAN AND VAR
# rolling_mean, rolling_var =
cal_rolling_mean_var(difference_2_RH,start="2004-03-10 18:00:00",
end="2005-04-04 14:00:00")
#
# # ACF and PACF of 1ST DIFFERENCE_RH
# ACF_PACF_Plot(difference_2_RH, 'ACF Difference of RH variable', 'PACF
```

```
Difference of RH variable', 50)



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

rcParams['figure.figsize'] = 16, 10
decomposition = sm.tsa.seasonal_decompose(train["RH"],
model='additive')

fig = decomposition.plot()
plt.title('Additive Residuals')
plt.show()

# rcParams['figure.figsize'] = 16, 10
# decomposition = sm.tsa.seasonal_decompose(train["RH"],
model='multiplicative')
# fig = decomposition.plot()
# plt.title('Multiplicative Residuals')
# plt.show()

y = df['RH'].astype(float)
print(y)
STL = STL(y)
res = STL.fit()
fig = res.plot()
# plt.fig(figsize=(16,10))
plt.show()

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

plt.figure(figsize=(16, 10))
plt.plot(T, label='trend')
plt.plot(S, label='Seasonal')
plt.plot(R, label='residuals')
plt.xlabel('Year', fontsize=16)
plt.ylabel('Magnitude', fontsize=16)
plt.title('Trend, Seasonality, Residual components using STL
Decomposition', fontsize=16)
plt.legend()
plt.show()

adjusted_seasonal = y - S
```

```
plt.figure(figsize=(16, 10))
plt.plot(y[:50], label='Original')
plt.plot(adjusted_seasonal[:50], label='Seasonally Adjusted')
plt.xlabel('Date', fontsize=16)
plt.ylabel('Magnitude', fontsize=16)
plt.title('Original vs Seasonally adjusted', fontsize=20)
plt.tick_params(axis='x', labelsize=16)
plt.tick_params(axis='y', labelsize=16)
plt.legend(loc='best', fontsize=15)
plt.show()

# Measuring strength of trend and seasonality
F = np.max([0, 1 - np.var(np.array(R)) / np.var(np.array(T + R))])
print('Strength of trend for Air quality dataset is', round(F, 3))

FS = np.max([0, 1 - np.var(np.array(R)) / np.var(np.array(S + R))])
print('Strength of seasonality for Air quality dataset is', round(FS,
3))

#
========================================================================
==========================
# QUESTION 9 :
# Holt-Winters method: Using the Holt-Winters method try to find the
best fit
# using the train dataset and make a prediction using the test set.
#
========================================================================
==========================

# Holt's Winter Seasonal Trend
# ===============================
print('=' * 20, 'HOLT WINTERS METHOD', '=' * 20)

holt_winter_model = ExponentialSmoothing(train["RH"],
seasonal='mul').fit()
# holt_winter_model = ExponentialSmoothing(train["RH"],
seasonal='multiplicative', trend='multiplicative').fit()

hw_train_pred = holt_winter_model.fittedvalues
hw_test_pred = list(holt_winter_model.forecast(len(test["RH"])))

# holt winter MSE and RESIDUAL ERROR
# =====================================
hw_residual_error_test = np.subtract(test["RH"].values,
np.array(hw_test_pred))
hw_residual_error_train = np.subtract(train["RH"].values,
np.array(hw_train_pred))

# Holt Winter mse
hw_mse_test = mean_squared_error(test["RH"].values, hw_test_pred)
hw_mse_train = mean_squared_error(train["RH"].values, hw_train_pred)

# holt winter rmse
hw_rmse_train = np.sqrt(hw_mse_train)
hw_rmse_test = np.sqrt(hw_mse_test)
```

```
# holt winter residual variance
hw_residual_variance_test = np.var(hw_residual_error_test)

# holt winter residual mean
hw_residual_mean_test = np.mean(hw_residual_error_test)

print("Holt Winter Method: MSE  of prediction errors (Train): ",
hw_mse_train)
print("Holt Winter Method: RMSE of prediction errors (Train): ",
hw_rmse_train)

print("Holt Winter Method: MSE  of forecast errors (Test): ",
hw_mse_test)
print("Holt Winter Method: RMSE of forecast errors (Test): ",
hw_rmse_test)

print("Holt Winter Method: Variance of Residual of forecast (Test) :",
hw_residual_variance_test)
print("Holt Winter Method: Mean of Residual of forecast (Test):",
hw_residual_mean_test)
print('\n')

# holt winter residual ACF


hw_residual_error_test_ACF = calc_acf(hw_residual_error_test,
len(hw_test_pred))

# calculate ACF
# ================================================
####  inbuilt function
# fig = plt.figure()
# plot_acf( hw_residual_error_test_ACF,  ax = plt.gca(),
lags=len(hw_residual_error_test_ACF)-1)
# plt.title('ACF of Residuals Error', fontsize=15)
# plt.tick_params(axis='x', labelsize=12)
# plt.tick_params(axis='y', labelsize=12)
# plt.show()

plot_acfx(hw_residual_error_test_ACF, "ACF plot using Holt Winter
Residuals")

# Calculate Q value for Residual Error
# Q =
len(test['RH'])*np.sum(np.square(hw_residual_error_test_ACF[100:]))
# print('The Q value of Holt Winter forecast is ', Q)


#
=======================================================================
==========================
# Question 10
# Feature selection: You need to have a section in your report that
explains how the feature
# selection was performed and whether the collinearity exits not.
# Backward stepwise regression along with SVD and condition number is
needed.
```

```
# You must explain that which feature(s) need to be eliminated and why.
# You are welcome to use other methods like PCA or random forest for
feature elimination.
#
=======================================================================
==========================


# Divide the dataset in features and target
x = df[['CO(GT)', 'PT08.S1(CO)', 'NMHC(GT)', 'C6H6(GT)',
'PT08.S2(NMHC)', 'NOx(GT)',
        'PT08.S3(NOx)', 'NO2(GT)', 'PT08.S4(NO2)', 'PT08.S5(O3)', 'T',
'AH']]
y = df[['RH']]

x_train, x_test, y_train, y_test = train_test_split(x, y,
shuffle=False, test_size=0.2)

# ================================
# Feature Selection
# ================================
# simgular values
from numpy import linalg as LA

X = x_train.values
H = np.matmul(X.T, X)
s, d, v = np.linalg.svd(H)
print('SingularValues = ', d)  #

condition_num = LA.cond(X)
print('The condition number is ', condition_num)  # features are
correlated in this dataset

# **************** All Variables ****************
model_0 = sm.OLS(y_train, x_train).fit()
print(model_0.summary())

# **************** Removing PT08.S5(O3) ****************
x_train.drop(columns='PT08.S5(O3)', axis=1, inplace=True)
model_1 = sm.OLS(y_train, x_train).fit()
print(model_1.summary())

# **************** Removing  NMHC(GT) *****************
x_train.drop(columns='NMHC(GT)', axis=1, inplace=True)
model_2 = sm.OLS(y_train, x_train).fit()
print(model_2.summary())

#
=======================================================================
==========================
# Question 11
# Base-models: average, naïve, drift, simple and exponential smoothing.
# You need to perform an h-step prediction based on the base models and
compare the SARIMA model performance with
# the base model predication.
#
=======================================================================
```

```
==========================

# Performance for all the models
base_model_columns = ["Model", "MSE", "RMSE","Residual Mean","Residual Variance",
                      "Train Residual Mean","Train Residual Variance", "Q Value"]
base_model_results = pd.DataFrame(columns=base_model_columns)

# ===============================================#
# **************** AVERAGE METHOD **************#
# ===============================================#
print(20 * "=" + " AVERAGE METHOD " + 20 * "=")


# Compute Prediction
average_pred_test = average_method(train["RH"], len(test["RH"]))
average_pred_train = average_method(train["RH"], len(train["RH"]))

# Compute Residual Error
average_residual_test = np.subtract(test["RH"].values, np.array(average_pred_test))
average_residual_train = np.subtract(train["RH"].values, np.array(average_pred_train))

# Compute Residual Variance
average_residual_variance = np.var(average_residual_test)
average_residual_variance_train = np.var(average_residual_train)
# Compute Residual Mean
average_residual_mean = np.mean(average_residual_test)
average_residual_mean_train = np.mean(average_residual_train)

# Compute MSE
average_mse_test = mean_squared_error(test["RH"].values, average_pred_test)
# Compute RMSE
average_rmse_test = np.sqrt(average_mse_test)

# Average residual ACF
average_residual_acf = calc_acf(average_residual_test, len(average_pred_test))

k = len(train) + len(test)
average_Q_value = k * np.sum(np.array(average_residual_acf) ** 2)

print("The MSE for Average model is : ", round(average_mse_test, 4))
print("The RMSE for Average model is: ", round(average_rmse_test, 4))
print("The Variance of residual for Average model is: ", round(average_residual_variance, 4))
print("The Mean of residual for Average model is: ", round(average_residual_mean, 4))
print('The Q value of Residual for Average model is: ', round(average_Q_value, 4))

print(60 * "=")

plot_acfx(average_residual_acf, "ACF plot using Average Residuals")
```

```
# add the results to common dataframe
values = ["Average Model",
          average_mse_test,
          average_rmse_test,
          average_residual_mean,
          average_residual_variance,
          average_residual_mean_train,
          average_residual_variance_train,
          average_Q_value]

base_model_results = base_model_results.append(pd.DataFrame([values],
columns=base_model_columns))

# plot the predicted vs actual data
average_df = test.copy(deep=True)
average_df["RH"] = average_pred_test

plot_multiline_chart_pandas_using_index([train, test, average_df],
"RH",
                                        ["Train", "Test",
"Prediction"], ["Blue", "Orange", "Green"],
                                        "Time", "RH",
                                        "Air Quality UCI Prediction
Using Average Model",
                                        rotate_xticks=True)

# =====================================#
# *********** NAIVE METHOD ***********#
# =====================================#
print(20 * "=" + "NAIVE MODEL" + 20 * "=")


# Compute Prediction
naive_pred_test = naive_method(train["RH"], len(test["RH"]))
naive_pred_train = naive_method(train["RH"], len(train["RH"]))

# Compute Residual Error
naive_residual_test = np.subtract(test["RH"].values,
np.array(naive_pred_test))
naive_residual_train = np.subtract(train["RH"].values,
np.array(naive_pred_train))
# Compute Residual Variance
naive_residual_variance = np.var(naive_residual_test)
naive_residual_variance_train = np.var(naive_residual_train)
# Compute Residual Mean
naive_residual_mean = np.mean(naive_residual_test)
naive_residual_mean_train = np.mean(naive_residual_train)# Compute MSE
naive_mse_test = mean_squared_error(test["RH"].values, naive_pred_test)
# Compute RMSE
naive_rmse_test = np.sqrt(naive_mse_test)

# Average residual ACF
naive_residual_acf = calc_acf(naive_residual_test,
len(naive_pred_test))

k = len(train) + len(test)
```

```
naive_Q_value = k * np.sum(np.array(naive_residual_acf) ** 2)

print("The MSE for Naive Model is : ", round(naive_mse_test, 4))
print("The RMSE for Naive Model is: ", round(naive_rmse_test, 4))
print("The Variance of residual for Naive Model is: ",
round(naive_residual_variance, 4))
print("The Mean of residual for Naive Model is: ",
round(naive_residual_mean, 4))
print('The Q value of Residual for Naive Model is: ',
round(naive_Q_value, 4))

print(60 * "=")

plot_acfx(naive_residual_acf, "ACF plot using Naive Residuals")

# add the results to common dataframe
values = ["Naive Model",
          naive_mse_test,
          naive_rmse_test,
          naive_residual_mean,
          naive_residual_variance,
          naive_residual_mean_train,
          naive_residual_variance_train,
          naive_Q_value]

base_model_results = base_model_results.append(pd.DataFrame([values],
columns=base_model_columns))

# plot the predicted vs actual data
naive_df = test.copy(deep=True)
naive_df["RH"] = naive_pred_test

plot_multiline_chart_pandas_using_index([train, test, naive_df], "RH",
                                        ["Train", "Test",
"Prediction"], ["Blue", "Orange", "Green"],
                                        "Time", "RH",
                                        "Air Quality UCI Prediction
Using Naive Model",
                                        rotate_xticks=True)

# ======================================#
# *********** DRIFT METHOD ***********#
# ======================================#

print(20 * "=" + " DRIFT MODEL " + 20 * "=")

# Compute Prediction
drift_pred_test = drift_method(train["RH"], len(test["RH"]))
drift_pred_train = drift_method(train["RH"], len(train["RH"]))
# Compute Residual Error
drift_residual_test = np.subtract(test["RH"].values,
np.array(drift_pred_test))
drift_residual_train = np.subtract(train["RH"].values,
np.array(drift_pred_train))
# Compute Residual Variance
drift_residual_variance = np.var(drift_residual_test)
drift_residual_variance_train = np.var(drift_residual_train)
```

```
# Compute Residual Mean
drift_residual_mean = np.mean(drift_residual_test)
drift_residual_mean_train = np.mean(drift_residual_train)
# Compute MSE
drift_mse_test = mean_squared_error( test["RH"].values,
drift_pred_test)
# Compute RMSE
drift_rmse_test = np.sqrt(drift_mse_test)

# Average residual ACF
drift_residual_acf = calc_acf(drift_residual_test,
len(drift_pred_test))

k = len(train) + len(test)
drift_Q_value = k * np.sum(np.array(drift_residual_acf) ** 2)

print("The MSE for Drift Model is : ", round(drift_mse_test, 4))
print("The RMSE for Drift Model is: ", round(drift_rmse_test, 4))
print("The Variance of residual for Drift Model is: ",
round(drift_residual_variance, 4))
print("The Mean of residual for Drift Model is: ",
round(drift_residual_mean, 4))
print('The Q value of Residual for Drift Model is: ',
round(drift_Q_value, 4))

print(60 * "=")
plot_acfx(drift_residual_acf, "ACF plot using Drift Residuals")

# add the results to common dataframe
values = ["Drift Model",
          drift_mse_test,
          drift_rmse_test,
          drift_residual_mean,
          drift_residual_variance,
          drift_residual_mean_train,
          drift_residual_variance_train,
          drift_Q_value]

base_model_results = base_model_results.append(pd.DataFrame([values],
columns=base_model_columns))

# plot the predicted vs actual data
drift_df = test.copy(deep=True)
drift_df["RH"] = drift_pred_test

plot_multiline_chart_pandas_using_index([train, test, drift_df], "RH",
                                        ["Train", "Test",
"Prediction"], ["Blue", "Orange", "Green"],
                                        "Time", "RH",
                                        "Air Quality UCI Prediction
Using Drift Model",
                                        rotate_xticks=True)

# ============================================================#
# *********** SIMPLE EXPONENTIAL SMOTHING METHOD ***********#
# ============================================================#
```

```
print(20 * "=" + "SIMPLE EXPONENTIAL SMOOTHING" + 20 * "=")

# Compute Prediction
ses_model =
SimpleExpSmoothing(np.asarray(train["RH"])).fit(smoothing_level=0.6,opt
imized=False)
ses_pred_test = ses_model.forecast(len(test))
ses_pred_train = ses_model.fittedvalues
# Compute Residual Error
ses_residual_test = np.subtract(test["RH"].values,
np.array(ses_pred_test))
ses_residual_train = np.subtract(train["RH"].values,
np.array(ses_pred_train))
# Compute Residual Variance
ses_residual_variance = np.var(ses_residual_test)
ses_residual_variance_train = np.var(ses_residual_train)
# Compute Residual Mean
ses_residual_mean = np.mean(ses_residual_test)
ses_residual_mean_train = np.mean(ses_residual_train)
# Compute MSE
ses_mse_test = mean_squared_error( test["RH"].values, ses_pred_test)
# Compute RMSE
ses_rmse_test = np.sqrt(ses_mse_test)


# Average residual ACF
ses_residual_acf = calc_acf(ses_residual_test, len(ses_pred_test))

k = len(train) + len(test)
ses_Q_value = k * np.sum(np.array(ses_residual_acf) ** 2)

print("The MSE for SES Model is : ", round(ses_mse_test, 4))
print("The RMSE for SES Model is: ", round(ses_rmse_test, 4))
print("The Variance of residual for SES Model is: ",
round(ses_residual_variance, 4))
print("The Mean of residual for SES Model is: ",
round(ses_residual_mean, 4))
print('The Q value of Residual for SES Model is: ', round(ses_Q_value,
4))

print(60 * "=")

plot_acfx(ses_residual_acf, "ACF plot using SES Residuals")

# add the results to common dataframe
values = ["Simple Exponential Smoothing Model",
          ses_mse_test,
          ses_rmse_test,
          ses_residual_mean,
          ses_residual_variance,
          ses_residual_mean_train,
          ses_residual_variance_train,
          ses_Q_value]

base_model_results = base_model_results.append(pd.DataFrame([values],
columns=base_model_columns))
```

```
# plot the predicted vs actual data
ses_df = test.copy(deep=True)
ses_df["RH"] = ses_pred_test

plot_multiline_chart_pandas_using_index([train, test, ses_df], "RH",
                                        ["Train", "Test",
"Prediction"], ["Blue", "Orange", "Green"],
                                        "Time", "RH",
                                        "Air Quality UCI Prediction
Using SES Model",
                                        rotate_xticks=True)

# =====================================================#
# *************** HOLT WINTER METHOD ***************#
# =====================================================#

print(20 * "=" + " HOLT WINTER METHOD " + 20 * "=")

# Compute Prediction
holtwinter_model = ExponentialSmoothing(train["RH"],
seasonal='mul').fit()
holtwinter_pred_test = holtwinter_model.forecast(len(test))
holtwinter_pred_train = holtwinter_model.fittedvalues
# Compute Residual Error
holtwinter_residual_test = np.subtract(test["RH"].values,
np.array(holtwinter_pred_test))
holtwinter_residual_train = np.subtract(train["RH"].values,
np.array(holtwinter_pred_train))
# Compute Residual Variance
holtwinter_residual_variance = np.var(holtwinter_residual_test)
holtwinter_residual_variance_train = np.var(holtwinter_residual_train)
# Compute Residual Mean
holtwinter_residual_mean = np.mean(holtwinter_residual_test)
holtwinter_residual_mean_train = np.mean(holtwinter_residual_train)
# Compute MSE
holtwinter_mse_test = mean_squared_error( test["RH"].values,
holtwinter_pred_test)
# Compute RMSE
holtwinter_rmse_test = np.sqrt(holtwinter_mse_test)


# Average residual ACF
holtwinter_residual_acf = calc_acf(holtwinter_residual_test,
len(holtwinter_pred_test))

k = len(train) + len(test)
holtwinter_Q_value = k * np.sum(np.array(holtwinter_residual_acf) ** 2)

print("The MSE for Holt Winter Model is : ", round(holtwinter_mse_test,
4))
print("The RMSE for Holt Winter Model is: ",
round(holtwinter_rmse_test, 4))
print("The Variance of residual for Holt Winter Model is: ",
round(holtwinter_residual_variance, 4))
print("The Mean of residual for Holt Winter Model is: ",
round(holtwinter_residual_mean, 4))
print('The Q value of Residual for Holt Winter Model is: ',
```

```
round(holtwinter_Q_value, 4))

print(60 * "=")
plot_acfx(holtwinter_residual_acf, "ACF plot using Holt Winter
Residuals")

# add the results to common dataframe
values = ["Holt Winter Model",
          holtwinter_mse_test,
          holtwinter_rmse_test,
          holtwinter_residual_mean,
          holtwinter_residual_variance,
          holtwinter_residual_mean_train,
          holtwinter_residual_variance_train,
          holtwinter_Q_value]

base_model_results = base_model_results.append(pd.DataFrame([values],
columns=base_model_columns))

# plot the predicted vs actual data
holtwinter_df = test.copy(deep=True)
holtwinter_df["RH"] = holtwinter_pred_test

plot_multiline_chart_pandas_using_index([train, test, holtwinter_df],
"RH",
                                        ["Train", "Test",
"Prediction"], ["Blue", "Orange", "Green"],
                                        "Time", "RH",
                                        "Air Quality UCI Prediction
Using Holt Winter Model",
                                        rotate_xticks=True)

print(' ' * 45, 'BASE MODEL COMPARISON')
print('=' * 110)
print(base_model_results.to_string())
print('=' * 110)

#========================================================================
===========================
# Question 12
# Develop the multiple linear regression model that represent the
dataset.
# Check the accuracy of the developed model.
# a. You need to include the complete regression analysis into your
report.
# Perform one-step ahead prediction and compare the performance versus
the test set.
#========================================================================
===========================

#Divide the dataset in features and target
x =
df[['CO(GT)','PT08.S1(CO)','NMHC(GT)','C6H6(GT)','PT08.S2(NMHC)','NOx(G
T)',

'PT08.S3(NOx)','NO2(GT)','PT08.S4(NO2)','PT08.S5(O3)','T','AH']]
y = df[['RH']]
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y,
shuffle=False, test_size=0.2)

# building the model
model = sm.OLS(y_train['RH'],x_train).fit()
# predict values for x_test
ml_pred_test = model.predict(x_test)
ml_pred_train = model.predict(x_train)

print(ml_pred_test)
print(model.summary())


print(20 * "=" + " MULTIPLE LINEAR REGRESSION " + 20 * "=")
# Compute Residual Error
ml_residual_test  = np.subtract(y_test[ 'RH'].values,
np.array(ml_pred_test))
ml_residual_train = np.subtract(y_train["RH"].values,
np.array(ml_pred_train))
# Compute Residual Variance
ml_residual_variance_test  = np.var(ml_residual_test)
ml_residual_variance_train = np.var(ml_residual_train)
# Compute Residual Mean
ml_residual_mean_test  = np.mean(ml_residual_test)
ml_residual_mean_train = np.mean(ml_residual_train)
# Compute MSE
ml_mse_test  = mean_squared_error( y_test[ "RH"].values, ml_pred_test)
ml_mse_train = mean_squared_error( y_train["RH"].values, ml_pred_train)
# Compute RMSE
ml_rmse_test  = np.sqrt(ml_mse_test )
ml_rmse_train = np.sqrt(ml_mse_train)



# Average residual ACF
ml_residual_test_acf  = calc_acf(ml_residual_test , len(ml_pred_test))
# ml_residual_train_acf = calc_acf(ml_residual_train,
len(ml_pred_train))

ml_test_Q_value  = len(x_test)  * np.sum(np.array(
ml_residual_test_acf)**2)
# ml_train_Q_value = len(x_train) * np.sum(np.array(
ml_residual_train_acf)**2)


print("The MSE for Multiple Linear Regression Model is : ", round(
ml_mse_test , 4) )
print("The RMSE for Multiple Linear Regression Model is: ", round(
ml_rmse_test, 4) )
print("The Variance of residual for Multiple Linear Regression Model
is: ", round(ml_residual_variance_test, 4) )
print("The Mean of residual for Multiple Linear Regression Model is: ",
round(ml_residual_mean_test, 4) )
print('The Q value of Residual for Multiple Linear Regression  Model
is: ', round ( ml_test_Q_value, 4))
print(60 * "=" )
```

```
plot_acfx(ml_residual_test_acf, "ACF plot using Multiple Linear
Regression Residuals (Test)")
# plot_acfx(ml_residual_train_acf, "ACF plot using Multiple Linear
Regression Residuals (Train)")

# add the results to common dataframe
values = ["Multiple Linear Regression Model",
          ml_mse_test,
          ml_rmse_test,
          ml_residual_mean_test,
          ml_residual_variance_test,
          ml_residual_mean_train,
          ml_residual_variance_train,
          ml_test_Q_value]

base_model_results = base_model_results.append( pd.DataFrame([values],
columns = base_model_columns ) )

print(' '* 45, 'BASE MODEL COMPARISON' )
print('=' * 170)
print( base_model_results.to_string() )
print('=' * 170)

# plot result

# TRAIN, TEST AND PREDICTED

dates_train = pd.date_range(start='2004-03-10 18:00:00', end='2005-01-
16 14:00:00', periods=len(y_train))
dates_test  = pd.date_range(start='2005-01-16 15:00:00', end='2005-04-
04 14:00:00', periods=len( y_test))

fig, ax = plt.subplots()
plt.title('Multiple Linear Regression:Traing vs Testing vs Forecast of
RH', fontsize=22)
ax.plot(dates_train, y_train['RH'],label='Training data')
ax.plot(dates_test,  y_test[ 'RH'],label='Testing data')
ax.plot(dates_test,  ml_pred_test,   label='Forecast')
plt.xlabel('Date', fontsize=15)
plt.ylabel('RH', fontsize=15)
plt.tick_params(axis='x', labelsize=12)
plt.tick_params(axis='y', labelsize=12)
plt.legend(loc='best', fontsize=20)
plt.show()

# TRAIN AND PREDICTED

fig, ax = plt.subplots()
plt.title('Multiple Linear Regression: Traing vs Predict of RH',
fontsize=22)
ax.plot(dates_train,y_train['RH'],label='Training data')
ax.plot(dates_train,ml_pred_train,label='Predicition')
plt.xlabel('Date', fontsize=15)
plt.ylabel('RH', fontsize=15)
plt.tick_params(axis='x', labelsize=12)
plt.tick_params(axis='y', labelsize=12)
plt.legend(loc='best', fontsize=20)
```

```
plt.show()

# TEST AND PREDICTED

fig, ax = plt.subplots()
plt.title('Multiple Linear Regression Model: Test vs Predict of RH',
fontsize=22)
ax.plot(dates_test, y_test['RH'] ,label='Testing data')
ax.plot(dates_test, ml_pred_test ,label='Forecast')
plt.xlabel('Date', fontsize=15)
plt.ylabel('RH', fontsize=15)
plt.tick_params(axis='x', labelsize=12)
plt.tick_params(axis='y', labelsize=12)
plt.legend(loc='best', fontsize=20)
plt.show()


#=======================================================================
===========================
# Question 13
# ARMA and ARIMA and SARIMA model order determination:
# Develop an ARMA, ARIMA and SARIMA model that represent the dataset.
# a. Preliminary model development procedures and results.
#    (ARMA model order determination).
#    Pick at least two orders using GPAC table.
# b. Should include discussion of the autocorrelation function and the
GPAC.
#    Include a plot of the autocorrelation function and the GPAC table
within this section).
# c. Include the GPAC table in your report and highlight the estimated
order.
#=======================================================================
===========================

# ARMA (ARIMA or SARIMA) model

j = 10
k = 10
lags = j + k

y_mean = np.mean(train['RH'])
y = np.subtract(y_mean, df['RH'])
actual_output = np.subtract(y_mean, test['RH'])

# autocorrelation of RH
ry = auto_corr_cal(y, lags)

# create GPAC Table
gpac_table = create_gpac_table(j, k, ry)
print()
print("GPAC Table:")
print(gpac_table.to_string())
print()

plot_heatmap(gpac_table, "GPAC Table for RH")
```

```
# possible orders of the process
possible_order2 = [(1,0),(2, 1)]

print()
print("The possible orders identified from GPAC for ARMA process are:")
print(possible_order2)
print()
print("None of the identified ARMA orders pass the chi-squared test.")
print()

# checking which orders pass the GPAC test
print(gpac_order_chi_square_test(possible_order2, y, '2004-03-10
18:00:00', '2005-01-16 14:00:00',
lags,actual_output))


possible_order = [(1, 0)]
possible_order = [(2, 1)]


# checking which orders pass the chi-square test
gpac_order_chi_square_test(possible_order, y, '2004-03-10 18:00:00',
'2005-01-16 14:00:00',
                                        lags, actual_output)


#**********************************************************************
********
# ARMA(1,0) model
#**********************************************************************
********

n_a = 1
n_b = 0

model = sm.tsa.ARMA(y, (n_a, n_b)).fit(trend='nc', disp=0)
print(model.summary())

# ARMA predictions
arma_prediction = model.predict(start="2005-01-16 15:00:00", end="2005-
04-04 14:00:00")
# arma_prediction = model.forecast(len(test['RH']))[1]

arma_prediction_train = model.fittedvalues[:len(train)]
# add the subtracted mean back into the predictions
arma_prediction = np.add(y_mean, arma_prediction)
arma_prediction_train = np.add(y_mean, arma_prediction_train)


# Compute Residual Error
arma_residual_test = np.subtract(test["RH"].values,
np.array(arma_prediction))
arma_residual_train = np.subtract(train["RH"].values,
np.array(arma_prediction_train))
# Compute Residual Variance
arma_residual_variance = np.var(arma_residual_test)
arma_residual_variance_train = np.var(arma_residual_train)
```

```
# Compute Residual Mean
arma_residual_mean = np.mean(arma_residual_test)
arma_residual_mean_train = np.mean(arma_residual_train)
# Compute MSE
arma_mse_test = mean_squared_error( test["RH"].values, arma_prediction)
# Compute RMSE
arma_rmse_test = np.sqrt(arma_mse_test)

print(f"The MSE for ARMA({n_a}, {n_b}) model is: " ,
round(arma_mse_test, 4) )
print(f"The RMSE for ARMA({n_a}, {n_b}) model is: ",
round(arma_rmse_test, 4))
print(f"The Variance of residual for ARMA({n_a}, {n_b}) model is:",
round(arma_residual_variance, 4))
print(f"The Mean of residual for ARMA({n_a}, {n_b}) model is:",
round(arma_residual_mean, 4))


# Average residual ACF
arma_residual_acf = calc_acf(arma_residual_test, len(arma_prediction))
plot_acfx(arma_residual_acf, f"ACF plot for ARMA({n_a},{n_b})
Residuals")

k=len(train) + len(test)
arma_Q_value  = k * np.sum(np.array( arma_residual_acf)**2)


print(f"Estimated covariance matrix for n_a = {n_a} and n_b = {n_b}:
\n{model.cov_params()}\n")
print(f"Estimated variance of error for n_a = {n_a} and n_b = {n_b}:
\n{model.sigma2}\n")


# add the results to common dataframe
values = [f"ARMA({n_a}, {n_b}) Model",
          arma_mse_test,
          arma_rmse_test,
          arma_residual_mean,
          arma_residual_variance,
          arma_residual_mean_train,
          arma_residual_variance_train,
          arma_Q_value]

base_model_results = base_model_results.append( pd.DataFrame([values],
columns = base_model_columns ) )


print(' '* 45, 'BASE MODEL COMPARISON' )
print('=' * 170)
print( base_model_results.to_string() )
print('=' * 170)


# plot the predicted vs actual data
arma_df = test.copy(deep=True)
arma_df["RH"] = arma_prediction
```

```
plot_multiline_chart_pandas_using_index([train, test, arma_df], "RH",
                                         ["Train", "Test",
"Prediction"], ["Blue", "Orange", "Green"],
                                         "Time", "RH",
                                         f"Air Quality UCI
Prediction Using ARMA({n_a}, {n_b}) Model",
                                         rotate_xticks=True)



# chi square
#chi_square_test(arma_Q_value, lags, n_a, n_b, alpha=0.01)


#**********************************************************************
********
# ARMA(2,1) model
#**********************************************************************
********

n_a = 2
n_b = 1

model = sm.tsa.ARMA(y, (n_a, n_b)).fit(trend='nc', disp=0)
print(model.summary())

# ARMA predictions
arma_prediction = model.predict(start="2005-01-16 15:00:00", end="2005-
04-04 14:00:00")
arma_prediction_train = model.fittedvalues[:len(train)]
# add the subtracted mean back into the predictions
arma_prediction = np.add(y_mean, arma_prediction)
arma_prediction_train = np.add(y_mean, arma_prediction_train)


# Compute Residual Error
arma_residual_test = np.subtract(test["RH"].values,
np.array(arma_prediction))
arma_residual_train = np.subtract(train["RH"].values,
np.array(arma_prediction_train))
# Compute Residual Variance
arma_residual_variance = np.var(arma_residual_test)
arma_residual_variance_train = np.var(arma_residual_train)
# Compute Residual Mean
arma_residual_mean = np.mean(arma_residual_test)
arma_residual_mean_train = np.mean(arma_residual_train)
# Compute MSE
arma_mse_test = mean_squared_error( test["RH"].values, arma_prediction)
# Compute RMSE
arma_rmse_test = np.sqrt(arma_mse_test)

print(f"The MSE for ARMA({n_a}, {n_b}) model is: " ,
round(arma_mse_test, 4) )
print(f"The RMSE for ARMA({n_a}, {n_b}) model is: ",
round(arma_rmse_test, 4))
print(f"The Variance of residual for ARMA({n_a}, {n_b}) model is:",
round(arma_residual_variance, 4))
```

```
print(f"The Mean of residual for ARMA({n_a}, {n_b}) model is:",
round(arma_residual_mean, 4))

# Average residual ACF
arma_residual_acf = calc_acf(arma_residual_test, len(arma_prediction))
plot_acfx(arma_residual_acf, f"ACF plot for ARMA({n_a},{n_b})
Residuals")

k=len(train) + len(test)
arma_Q_value  = k * np.sum(np.array( arma_residual_acf)**2)


print(f"Estimated covariance matrix for n_a = {n_a} and n_b = {n_b}:
\n{model.cov_params()}\n")
print(f"Estimated variance of error for n_a = {n_a} and n_b = {n_b}:
\n{model.sigma2}\n")


# add the results to common dataframe
values = [f"ARMA({n_a}, {n_b}) Model",
          arma_mse_test,
          arma_rmse_test,
          arma_residual_mean,
          arma_residual_variance,
          arma_residual_mean_train,
          arma_residual_variance_train,
          arma_Q_value]

base_model_results = base_model_results.append( pd.DataFrame([values],
columns = base_model_columns ) )




print(' '* 45, 'BASE MODEL COMPARISON' )
print('=' * 170)
print( base_model_results.to_string() )
print('=' * 170)


# plot the predicted vs actual data
arma_df = test.copy(deep=True)
arma_df["RH"] = arma_prediction

plot_multiline_chart_pandas_using_index([train, test, arma_df], "RH",
                                         ["Train", "Test",
"Prediction"], ["Blue", "Orange", "Green"],
                                         "Time", "RH",
                                         f"Air Quality UCI
Prediction Using ARMA({n_a}, {n_b}) Model",
                                         rotate_xticks=True)

# chi square
#chi_square_test(arma_Q_value, lags, n_a, n_b, alpha=0.01)


#======================================================================
```

```
===========================
# Question 14
# Estimate ARMA model parameters using the Levenberg Marquardt
algorithm.
# Display the parameter estimates, the standard deviation of the
parameter
# estimates and confidence intervals.
#========================================================================
===========================

#####################
#### ARMA (1,0) #####
#####################

na, nb = (1, 0)

# ================================================
# LM ALGORITM
# ================================================
lm_params, ro2, cov_theta, iterations = LM_algoritmh(y=train['RH'],
                                                     n_a=na,
                                                     n_b=nb,
                                                     num_iter=30,
                                                     delta=1e-6,
                                                     flip_val=True)

# parameter estimated
print(' ' * 27, ' PARAMETER ESTIMATED ')
print('=' * 80)

lm_den = [1.] + [lm_params[i] for i in range(na)]
lm_num = [1.] + [lm_params[i + na] for i in range(nb)]
for i in range(na):
    print('LM - The AR coefficient a{}'.format(i), 'is:', lm_params[i])
for i in range(nb):
    print('LM - The MA coefficient b{}'.format(i), 'is:', lm_params[i +
na])

# ARMA MODEL to compare
model = sm.tsa.ARMA(train["RH"], (na, nb)).fit(trend='nc', disp=0)   #
hacer del train

for i in range(na):
    print('The AR coefficient a{}'.format(i), 'is:', model.params[i])
for i in range(nb):
    print('The MA coefficient b{}'.format(i), 'is:', model.params[i +
na])

# Estimated covariance matrix of the estimated parameters.
print(' ' * 25, 'Estimated Covariance Matrix')
print(' ' * 30, 'LM algorithm')
print('=' * 80)
print(np.matrix(cov_theta))
print('=' * 80)
# Estimated variance of error.
print(' ' * 23, 'Estimated Variance of Error')
print(' ' * 30, 'LM algorithm')
```

```
print('=' * 80)
print(np.matrix(ro2))
print('=' * 80)
# Confidence interval for each estimated parameter(s).
# θi ± 2*sqrt(cov(θ)ii)
print(' ' * 23, 'Confidence Interval for Theta')
print('=' * 80)
print('   θi-2*sqrt(cov(θ)ii)    |             θi              |   θi ±
2*sqrt(cov(θ)ii)')
print('=' * 80)
for i in range(len(lm_params)):
    sqrt_ro = np.sqrt(cov_theta[i, i])
    theta_i = lm_params[i]
    theta_lower = theta_i - 2 * sqrt_ro
    theta_upper = theta_i + 2 * sqrt_ro
    print('  ', str(theta_lower).ljust(26),
          str(theta_i).ljust(26),
          str(theta_upper))

print('=' * 80)

#======================================================================
===========================
# Question 15
# Diagnostic Analysis: Make sure to include the followings:
# a. Diagnostic tests (confidence intervals, zero/pole cancellation,
chi-square test).
# b. Display the estimated variance of the error and the estimated
covariance of the estimated parameters.
# c. Is the derived model biased or this is an unbiased estimator?
# d. Check the variance of the residual errors versus the variance of
the forecast errors.
# e. If you find out that the ARIMA or SARIMA model may better
represents the dataset, then you can find the model accordingly. You
are not constraint only to use of ARMA model. Finding an ARMA model is
a minimum requirement and making the model better is always welcomed.
#======================================================================
===========================

# Confidence interval for each estimated parameter(s).
# θi ± 2*sqrt(cov(θ)ii)
print(' '*23,'Confidence Interval for Theta')
print('=' * 80)
print('   θi-2*sqrt(cov(θ)ii)    |             θi              |   θi ±
2*sqrt(cov(θ)ii)')
print('=' * 80)
for i in range( len(lm_params) ):
    sqrt_ro = np.sqrt(cov_theta[i,i])
    theta_i = lm_params[i]
    theta_lower = theta_i - 2 * sqrt_ro
    theta_upper = theta_i + 2 * sqrt_ro
    print('  ', str(theta_lower).ljust(26),
          str(theta_i).ljust(26),
          str(theta_upper) )
print('=' * 80)

# Zero/Pole Cancelation
```

```
#***************************************************
zeros, poles, _ = zero_pole_plot(lm_num, lm_den)

print(' '*23,'Zero / Pole Cancelation')
print('=' * 80)
print('Root of numerator "Zeros" : ', zeros)
print('Root of denominator "Poles" : ', poles)

# Estimated covariance matrix of the estimated parameters.
print(' '*25,'Estimated Covariance Matrix')
print(' '*30, 'LM algorithm' )
print('=' * 80)
print(np.matrix( cov_theta ) )
print('=' * 80)
# Estimated variance of error.
print(' '*23,'Estimated Variance of Error')
print(' '*30, 'LM algorithm' )
print('=' * 80)
print(np.matrix( ro2 ) )
print('=' * 80)

#####################
#### ARMA (2,1) #####
#####################

na, nb = (2,1)

#==================================================
# LM ALGORITM
#==================================================
lm_params, ro2, cov_theta, iterations = LM_algoritmh(y = train['RH'],
                                                     n_a = na,
                                                     n_b = nb,
                                                     num_iter = 30,
                                                     delta = 1e-6,
                                                     flip_val=True)



# parameter estimated
print(' '*27,' PARAMETER ESTIMATED ')
print('=' * 80)

lm_den = [1.] + [ lm_params[i] for i in range(na)]
lm_num = [1.] + [ lm_params[i+na] for i in range(nb)]
for i in range(na):
    print('LM - The AR coefficient a{}'.format(i), 'is:', lm_params[i])
for i in range(nb):
    print('LM - The MA coefficient b{}'.format(i), 'is:',
lm_params[i+na])


# ARMA MODEL to compare
model = sm.tsa.ARMA(train["RH"],(na,nb)).fit(trend='nc',disp=0) # hacer
del train

for i in range(na):
```

```
    print('The AR coefficient a{}'.format(i), 'is:', model.params[i])
for i in range(nb):
    print('The MA coefficient b{}'.format(i), 'is:',
model.params[i+na])


# Estimated covariance matrix of the estimated parameters.
print(' '*25,'Estimated Covariance Matrix')
print(' '*30, 'LM algorithm' )
print('=' * 80)
print(np.matrix( cov_theta ) )
print('=' * 80)
# Estimated variance of error.
print(' '*23,'Estimated Variance of Error')
print(' '*30, 'LM algorithm' )
print('=' * 80)
print(np.matrix( ro2 ) )
print('=' * 80)
# Confidence interval for each estimated parameter(s).
# θi ± 2*sqrt(cov(θ)ii)
print(' '*23,'Confidence Interval for Theta')
print('=' * 80)
print('   θi-2*sqrt(cov(θ)ii)   |            θi             |   θi ±
2*sqrt(cov(θ)ii)')
print('=' * 80)
for i in range( len(lm_params) ):
    sqrt_ro = np.sqrt(cov_theta[i,i])
    theta_i = lm_params[i]
    theta_lower = theta_i - 2 * sqrt_ro
    theta_upper = theta_i + 2 * sqrt_ro
    print('  ', str(theta_lower).ljust(26),
          str(theta_i).ljust(26),
          str(theta_upper) )

print('=' * 80)

#======================================================================
===========================
# Question 15
# Diagnostic Analysis: Make sure to include the followings:
# a. Diagnostic tests (confidence intervals, zero/pole cancellation,
chi-square test).
# b. Display the estimated variance of the error and the estimated
covariance of the estimated parameters.
# c. Is the derived model biased or this is an unbiased estimator?
# d. Check the variance of the residual errors versus the variance of
the forecast errors.
# e. If you find out that the ARIMA or SARIMA model may better
represents the dataset, then you can find the model accordingly. You
are not constraint only to use of ARMA model. Finding an ARMA model is
a minimum requirement and making the model better is always welcomed.
#======================================================================
===========================

# Confidence interval for each estimated parameter(s).
# θi ± 2*sqrt(cov(θ)ii)
print(' '*23,'Confidence Interval for Theta')
```

```
print('=' * 80)
print('   θi-2*sqrt(cov(θ)ii)    |             θi              |   θi ±
2*sqrt(cov(θ)ii)')
print('=' * 80)
for i in range( len(lm_params) ):
    sqrt_ro = np.sqrt(cov_theta[i,i])
    theta_i = lm_params[i]
    theta_lower = theta_i - 2 * sqrt_ro
    theta_upper = theta_i + 2 * sqrt_ro
    print('  ', str(theta_lower).ljust(26),
          str(theta_i).ljust(26),
          str(theta_upper) )
print('=' * 80)

# Zero/Pole Cancelation
#*******************************************************
zeros, poles, _ = zero_pole_plot(np.array(lm_num) , np.array(lm_den) )

print(' '*23,'Zero / Pole Cancelation')
print('=' * 80)
print('Root of numerator "Zeros" : ', zeros)
print('Root of denominator "Poles" : ', poles)

# Estimated covariance matrix of the estimated parameters.
print(' '*25,'Estimated Covariance Matrix')
print(' '*30, 'LM algorithm' )
print('=' * 80)
print(np.matrix( cov_theta ) )
print('=' * 80)
# Estimated variance of error.
print(' '*23,'Estimated Variance of Error')
print(' '*30, 'LM algorithm' )
print('=' * 80)
print(np.matrix( ro2 ) )
print('=' * 80)

#======================================================================
===========================
# Question 18
# Forecast function: Once the final mode is picked (SARIMA), the
forecast
# function needs to be developed and included in your report.
#======================================================================
===========================

#**********************************************************************
********
# SARIMA MODEL (0,0,0)  (n_a, d , nb)
#**********************************************************************
********

order_x = (0,0,0)
seasonal_order_x = (0,1,1,24)

sarima= sm.tsa.statespace.SARIMAX(train["RH"],
                                  order = order_x, seasonal_order =
seasonal_order_x ).fit()
```

```python
print(sarima.summary())

sarima_pred_test = sarima.forecast(len(test['RH']))
sarima_pred_train = sarima.fittedvalues
# Compute Residual Error
sarima_residual_test = np.subtract(test["RH"].values,
np.array(sarima_pred_test))
sarima_residual_train = np.subtract(train["RH"].values,
np.array(sarima_pred_train))
# Compute Residual Variance
sarima_residual_variance = np.var(sarima_residual_test)
sarima_residual_variance_train = np.var(sarima_residual_train)
# Compute Residual Mean
sarima_residual_mean=np.mean(sarima_residual_test)
sarima_residual_mean_train = np.mean(sarima_residual_train)
# Compute MSE
sarima_mse_test = mean_squared_error( test["RH"].values,
sarima_pred_test)
# Compute RMSE
sarima_rmse_test = np.sqrt(sarima_mse_test)



# Average residual ACF
sarima_residual_acf = calc_acf(sarima_residual_test,
len(sarima_pred_test))


k = len(train) + len(test)
sarima_Q_value = k * np.sum(np.array( sarima_residual_acf)**2)


print("The MSE for SARIMA Model is : ", round( sarima_mse_test , 4) )
print("The RMSE for SARIMA Model is: ", round( sarima_rmse_test, 4) )
print("The Variance of residual for SARIMA Model is: ",
round(sarima_residual_variance, 4) )
print("The Mean of residual for SARIMA Model is: ",
round(sarima_residual_mean, 4) )
print("The Variance of residual for SARIMA Model (Train) is: ",
round(sarima_residual_variance_train, 4) )
print("The Mean of residual for SARIMA Model (Train) is: ",
round(sarima_residual_mean_train, 4) )
print('The Q value of Residual for SARIMA Model is: ', round (
sarima_Q_value, 4))


print(60 * "=" )
plot_acfx(sarima_residual_acf, "ACF plot using SARIMA Residuals")

# add the results to common dataframe
values = [f"SARIMA {order_x} {seasonal_order_x} Model",
          sarima_mse_test,
          sarima_rmse_test,
          sarima_residual_mean,
          sarima_residual_variance,
          sarima_residual_mean_train,
```

```
          sarima_residual_variance_train,
          sarima_Q_value]

base_model_results = base_model_results.append( pd.DataFrame([values],
columns = base_model_columns ) )


dates_train = pd.date_range(start='2004-03-10 18:00:00', end='2005-01-
16 14:00:00', periods=len(y_train))
dates_test  = pd.date_range(start='2005-01-16 15:00:00', end='2005-04-
04 14:00:00', periods=len( y_test))

# TEST AND PREDICTED
start = 1000
range_x = range(start, start + 400)
fig, ax = plt.subplots()
plt.title('SARIMA: Train vs Fitted Values of RH', fontsize=22)
ax.plot(dates_train[range_x], y_train['RH'].values[range_x]
,label='Training data')
ax.plot(dates_train[range_x], sarima_pred_train.values[range_x]
,label='Fitted Values')
plt.xlabel('Date', fontsize=15)
plt.ylabel('RH', fontsize=15)
plt.tick_params(axis='x', labelsize=12)
plt.tick_params(axis='y', labelsize=12)
plt.legend(loc='best', fontsize=20)
plt.show()



# TEST AND PREDICTED
start = 0
range_x = range(start, start + 400)
fig, ax = plt.subplots()
plt.title('SARIMA: Test vs Predict of RH', fontsize=22)
ax.plot(dates_test[range_x], y_test['RH'].values[range_x]
,label='Testing data')
ax.plot(dates_test[range_x], sarima_pred_test.values[range_x]
,label='Forecast')
plt.xlabel('Date', fontsize=15)
plt.ylabel('RH', fontsize=15)
plt.tick_params(axis='x', labelsize=12)
plt.tick_params(axis='y', labelsize=12)
plt.legend(loc='best', fontsize=20)
plt.show()



print(' '* 80, 'BASE MODEL COMPARISON' )
print('=' * 170)
print( base_model_results.to_string() )
print('=' * 170)

#======================================================================
===========================
# Question 19
# h-step ahead Predictions: You need to make a multiple step ahead
```

```
prediction
# for the duration of the test data set. Then plot the predicted values
# versus the true value (test set) and write down your observations.
#======================================================================
===========================

# h-step ahead prediction - using Multiple Linear Regression


#
***********************************************************************
********
# MULTIPLE LINEAR REGRESSION
#
***********************************************************************
********
# TRAIN, TEST AND PREDICTED

dates_train = pd.date_range(start='2004-03-10 18:00:00', end='2005-01-
16 14:00:00', periods=len(y_train))
dates_test  = pd.date_range(start='2005-01-16 15:00:00', end='2005-04-
04 14:00:00', periods=len( y_test))


# TRAIN AND PREDICTED
h_ahead = 1700
range_x = range( h_ahead)


# TEST AND PREDICTED

fig, ax = plt.subplots()
plt.title('Multiple Linear Regression Model: Test vs Predict of RH',
fontsize=22)
ax.plot( dates_test[range_x]
,y_test['RH'].values[range_x],label='Testing data')
ax.plot( dates_test[range_x],
ml_pred_test.values[range_x],label='Forecast')
plt.xlabel('Date', fontsize=15)
plt.ylabel('RH', fontsize=15)
plt.tick_params(axis='x', labelsize=12)
plt.tick_params(axis='y', labelsize=12)
plt.legend(loc='best', fontsize=20)
plt.show()

#************************************************************
# SARIMA OBSERVATIONS
#************************************************************

dates_train = pd.date_range(start='2004-03-10 18:00:00', end='2005-01-
16 14:00:00', periods=len(y_train))
dates_test  = pd.date_range(start='2005-01-16 15:00:00', end='2005-04-
04 14:00:00', periods=len( y_test))

# TEST AND PREDICTED
h_ahead = 100
```

```
range_x = range(0, h_ahead)
fig, ax = plt.subplots()
plt.title('SARIMA: Train vs Fitted Values of RH', fontsize=22)
ax.plot(dates_train[range_x], y_train['RH'].values[range_x]
,label='Training data')
ax.plot(dates_train[range_x], sarima_pred_train.values[range_x]
,label='Fitted Values')
plt.xlabel('Date', fontsize=15)
plt.ylabel('RH', fontsize=15)
plt.tick_params(axis='x', labelsize=12)
plt.tick_params(axis='y', labelsize=12)
plt.legend(loc='best', fontsize=20)
plt.show()



# TEST AND PREDICTED
fig, ax = plt.subplots()
plt.title('SARIMA: Test vs Predict of RH', fontsize=22)
ax.plot(dates_test[range_x], y_test['RH'].values[range_x]
,label='Testing data')
ax.plot(dates_test[range_x], sarima_pred_test.values[range_x]
,label='Forecast')
plt.xlabel('Date', fontsize=15)
plt.ylabel('RH', fontsize=15)
plt.tick_params(axis='x', labelsize=12)
plt.tick_params(axis='y', labelsize=12)
plt.legend(loc='best', fontsize=20)
plt.show()
```