THE GEORGE WASHINGTON UNIVERSITY

WASHINGTON, DC

AIR QUALITY

Time Series Analysis & Modelling Term Project

Dinesh Kumar Padmanabhan

Professor Reza Jafari

DATS6450

Table of Contents

I. ABSTRACT 4

II. INTRODUCTION: 5

2.1 Problem Statement 6

2.2 Objective 6

2.3 Overview 6

III. DESCRIPTION OF DATASET 7

3.1 Resource 7

3.2 Overview of the Dataset 7

3.3 Pre-processing 8

3.4 Dependent variable v/s time 10

3.5 ACF of the dependent variable 11

3.6 Correlation Matrix 12

3.7 Split Train/Test 13

IV. STATIONARITY 13

V. TIME SERIES DECOMPOSITION 14

5.1 Additive Model 14

5.2 Multiplicative Model 15

5.3 Original v/s Seasonally adjusted 16

VI. HOLT- WINTERS METHOD 17

6.1 ACF of residuals: 17

VII. FEATURE SELECTION 19

VIII. MULTIPLE LINEAR REGRESSION 20

8.1. Linear Model with all Features: 20

8.2 ACF of Residuals 21

IX. ARMA MODEL (1) 23

9.1 GPAC Table 23

9.2 Parameter Estimation 24

9.3 ARMA (2,7) Model 25

9.4 Diagnostic Analysis 26

9.4.1 Chi Square Test 26

9.4.2 Simplification of Model 26

9.4.3 Zero/pole cancellation 27

9.4.4 Performance measures 27

X. ARMA MODEL (2) 29
10.1 Diagnostic Analysis 30
10.1.1 Chi Square Test 30
10.1.2 Simplification of Model 30
10.1.3 Zero/pole cancellation 31
10.1.4 Performance measures 31
XI. ARMA MODEL (3) 33
11.1 Diagnostic Analysis 34
11.1.1 Chi Square Test 34
11.1.2 Simplification of Model 34
11.1.3 Zero/pole cancellation 35
11.1.4 Performance measures 35
XII. ARMA MODEL (4) 37
12.1 Diagnostic Analysis 38
12.1.1 Chi Square Test 38
12.1.2 Simplification of Model 38
12.1.3 Zero/pole cancellation 39
12.1.4 Performance measures 39
XIII. BASE-MODELS 41
13.1 Average Model 41
13.2 Naïve Model 43
13.3 Drift Model 45
13.4 SIMPLE AND EXPONENTIAL SMOOTHING 47
XIV. FINAL MODEL SELECTION 49
XV. SUMMARY AND CONCLUSION 50
XVI. NEXT STEPS 56
APPENDIX 58

I. ABSTRACT

Controlling and preserving the better air excellence has become one of the most indispensable events in numerous manufacturing plus metropolitan regions at present. The excellence of air is harmfully affecting payable to the various forms of contamination affected via the transportation, power, fuels expenditures, etc. The installation of destructive fumes is spawning the severe hazard for the class of natural life in developed metropolises. Through cumulative air contamination, we require implementing competent air excellence monitoring models which gathers the statistics about the absorption of air impurities and be responsible for calculation of air contamination in each zone. Hence, air excellence estimation plus calculation has come to be a significant study area. The superiority of air is exaggerated by multi-dimensional influences comprising place, time plus indeterminate parameters. The intention of this development is to examine the time series based methods for air quality prediction.

Keywords- Air Quality Index, Linear Regression, Stepwise Regression, Base Models: Average, Naïve , Drift, Holt winter, ARMA, ARIMA Model, Python, pycharm.

II. INTRODUCTION:

Air is any of the greatest critical expected possessions for the endurance plus subsistence for the complete life on this globe. All forms of the life comprising plant life plus wildlife depending on air for their basic endurance. Thus, all breathing creatures require better excellence of air which is free of dangerous fumes for continuing their lives. The increasing populace, vehicles and productions are poisoning all the air at an alarming rate. Dissimilar elements are related with metropolitan air contamination. In order to estimate the air impurity, contaminant constraints are considered in the lower altitudes of the troposphere, which are meticulous. Air excellence sensor devices extent the attentions of particles that have an anthropogenic source and create the hazardous effects during or after the gulp of air by human being. Particles like PM2.5, CO, NO2, NO etc. disturbs the quality of air. Automobiles release enormous amounts of nitrogen oxides, carbon oxides, hydrocarbons and particulates when burning petrol and diesel. Therefore, we want to consider the added Parameters like number of vehicles along with the Pollutants data for prediction of pollution in specific regions of Italy.

2.1 Problem Statement

Humans are very sensitive to humidity, as the skin relies on the air to get rid of moisture. The process of sweating is your body's attempt to keep cool and maintain its current temperature. If the air is at 100-percent relative humidity, sweat will not evaporate into the air. As a result, we feel much hotter than the actual temperature when the relative humidity is high. If the relative humidity is low, we can feel much cooler than the actual temperature because our sweat evaporates easily, cooling us off. For example, if the air temperature e is 75 degrees Fahrenheit (24 degrees Celsius) and the relative humidity is zero percent, the air temperature feels like 69 degrees Fahrenheit (21 C) to our bodies. If the air temperature is 75 degrees Fahrenheit (24 C) and the relative humidity is 100 percent, we feel like it's 80 degrees (27 C) out.

2.2 Objective

To predict the Relative Humidity (RH) of a given point of time based on the all other attributes affecting the change in RH.

2.3 Overview

We will first introduce the data, then concepts, and structures of our various models and their theories. Then we'll explain each step of implementation and how we arrived in solving. Finally, we will explain the conclusion and the ideas for future improvement.

III. DESCRIPTION OF DATASET

3.1 Resource

The data is obtained from UCI Machine Learning Repository. Contains the responses of a gas multisensor device deployed on the field in an Italian city. Hourly responses averages are recorded along with gas concentrations references from a certified analyzer.

3.2 Overview of the Dataset

The dataset contains 9358 instances of hourly averaged responses from an array of 5 metal oxide chemical sensors embedded in an Air Quality Chemical Multisensor Device. The device was located on the field in a significantly polluted area, at road level, within an Italian city. Data were recorded from March 2004 to February 2005 (one year)representing the longest freely available recordings of on field deployed air quality chemical sensor devices responses. Ground Truth hourly averaged concentrations for CO, Non Metanic Hydrocarbons, Benzene, Total Nitrogen Oxides (NOx) and Nitrogen Dioxide (NO2) and were provided by a co-located reference certified analyzer. Following are the attributes of

Dependent Variable:

Relative Humidity (%)

Independent Variables:

0 Date (DD/MM/YYYY)

1 Time (HH.MM.SS)

2 True hourly averaged concentration CO in mg/m^3 (reference analyzer)

3 PT08.S1 (tin oxide) hourly averaged sensor response (nominally CO targeted)

4 True hourly averaged overall Non Metanic HydroCarbons concentration in microg/m^3 (reference analyzer)

5 True hourly averaged Benzene concentration in microg/m^3 (reference analyzer)

6 PT08.S2 (titania) hourly averaged sensor response (nominally NMHC targeted)

7 True hourly averaged NOx concentration in ppb (reference analyzer)

8 PT08.S3 (tungsten oxide) hourly averaged sensor response (nominally NOx targeted)

9 True hourly averaged NO2 concentration in microg/m^3 (reference analyzer)

10 PT08.S4 (tungsten oxide) hourly averaged sensor response (nominally NO2 targeted)

11 PT08.S5 (indium oxide) hourly averaged sensor response (nominally O3 targeted)

12 Temperature in Â°C

13 AH Absolute Humidity

3.3 Pre-processing

Importing data using Pandas, taking a look at the dimension of the dataset using df.shape **(9357, 13)**. Using the pd.read_csv() function the csv file is brought into a pandas DataFrame, with the *infer_datetime_format* parameter as *True*- because every new object is separated by a new line. We can already make few observations here, by looking at the head of the data: There are missing values across the dataframe, which need to be handled. Unnamed columns need to be removed Not so important, but some columns could do with some renaming- for removing spaces.

	Date	Time	CO(GT)	...	AH	Unnamed: 15	Unnamed: 16
0	10/03/2004	18.00.00	2,6	...	0,7578	NaN	NaN
1	10/03/2004	19.00.00	2	...	0,7255	NaN	NaN
2	10/03/2004	20.00.00	2,2	...	0,7502	NaN	NaN
3	10/03/2004	21.00.00	2,2	...	0,7867	NaN	NaN
4	10/03/2004	22.00.00	1,6	...	0,7888	NaN	NaN

First 5 rows of Dataframe

Firstly, we handle by removing unnamed columns for ease-of-access in pandas and then handle null values.

	Date	Time	CO(GT)	PT08.S1(CO)	...	PT08.S5(O3)	T	RH	AH
9357	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
9358	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
9359	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
9360	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
9361	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN

Some more important observations here, before we dive into performing the pre-processing tasks onto our data. Date, Time, CO(GT), C6H6, T, RH, AH is of types objects which needs to be addressed by changing from object to float. Also, for ease of access appending date and time.

Almost all columns surprisingly have a lot of NULL values and Nan Values. After estimating the sum of null values, null values are dropped. Also, it was observed there were lot of -200 values replacing -200 by nan and later this was handled by imputing default strategy mean and thus creating a new data frame.

```
Datatypes in Dataframe
 Date              object
Time               object
CO(GT)             object
PT08.S1(CO)       float64
NMHC(GT)          float64
C6H6(GT)           object
PT08.S2(NMHC)     float64
NOx(GT)           float64
PT08.S3(NOx)      float64
NO2(GT)           float64
PT08.S4(NO2)      float64
PT08.S5(O3)       float64
T                  object
RH                 object
AH                 object
dtype: object
```

Data types of dataframe

```
 Date                 0
Time                  0
CO(GT)             1683
PT08.S1(CO)         366
NMHC(GT)           8443
C6H6(GT)            366
PT08.S2(NMHC)       366
NOx(GT)            1639
PT08.S3(NOx)        366
NO2(GT)            1642
PT08.S4(NO2)        366
PT08.S5(O3)         366
T                   366
RH                  366
AH                  366
Datetime              0
dtype: int64
```

Dataframe with date and time appended

```
 CO(GT)           0
PT08.S1(CO)       0
NMHC(GT)          0
C6H6(GT)          0
PT08.S2(NMHC)     0
NOx(GT)           0
PT08.S3(NOx)      0
NO2(GT)           0
PT08.S4(NO2)      0
PT08.S5(O3)       0
T                 0
RH                0
AH                0
dtype: int64
```

Resampled Dataframe

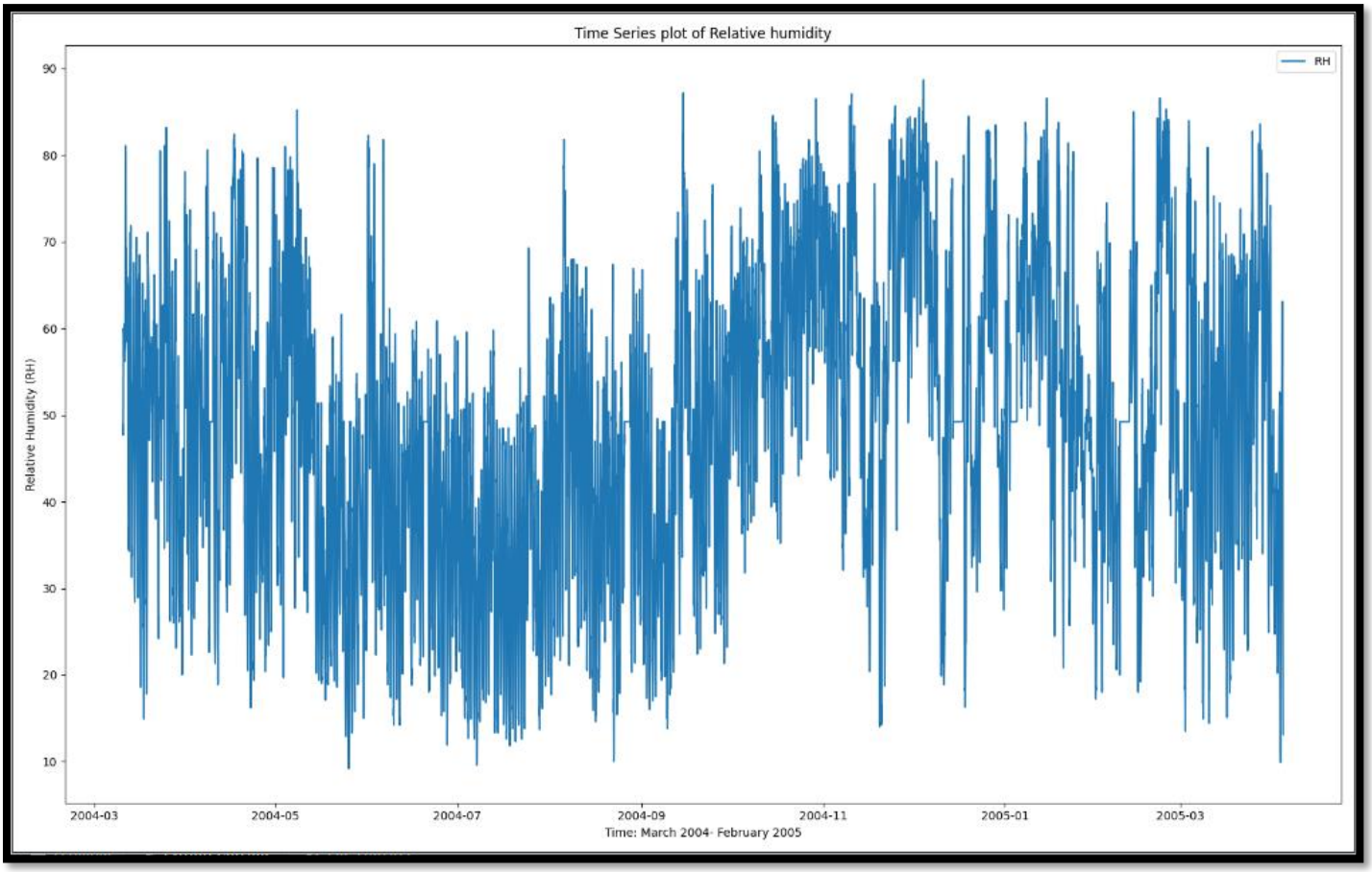
After performing the pre-processing thus, a cleaned dataset is created with dimension of (9357, 13)

```
                     CO(GT)  PT08.S1(CO)    NMHC(GT)  ...     T    RH      AH
2004-03-10 18:00:00     2.6       1360.0  150.000000  ...  13.6  48.9  0.7578
2004-03-10 19:00:00     2.0       1292.0  112.000000  ...  13.3  47.7  0.7255
2004-03-10 20:00:00     2.2       1402.0   88.000000  ...  11.9  54.0  0.7502
2004-03-10 21:00:00     2.2       1376.0   80.000000  ...  11.0  60.0  0.7867
2004-03-10 22:00:00     1.6       1272.0   51.000000  ...  11.2  59.6  0.7888
                      ...          ...         ...  ...   ...   ...     ...
2005-04-04 10:00:00     3.1       1314.0  218.811816  ...  21.9  29.3  0.7568
2005-04-04 11:00:00     2.4       1163.0  218.811816  ...  24.3  23.7  0.7119
2005-04-04 12:00:00     2.4       1142.0  218.811816  ...  26.9  18.3  0.6406
2005-04-04 13:00:00     2.1       1003.0  218.811816  ...  28.3  13.5  0.5139
2005-04-04 14:00:00     2.2       1071.0  218.811816  ...  28.5  13.1  0.5028
```

Cleaned dataset after pre-processing

3.4 Dependent variable v/s time

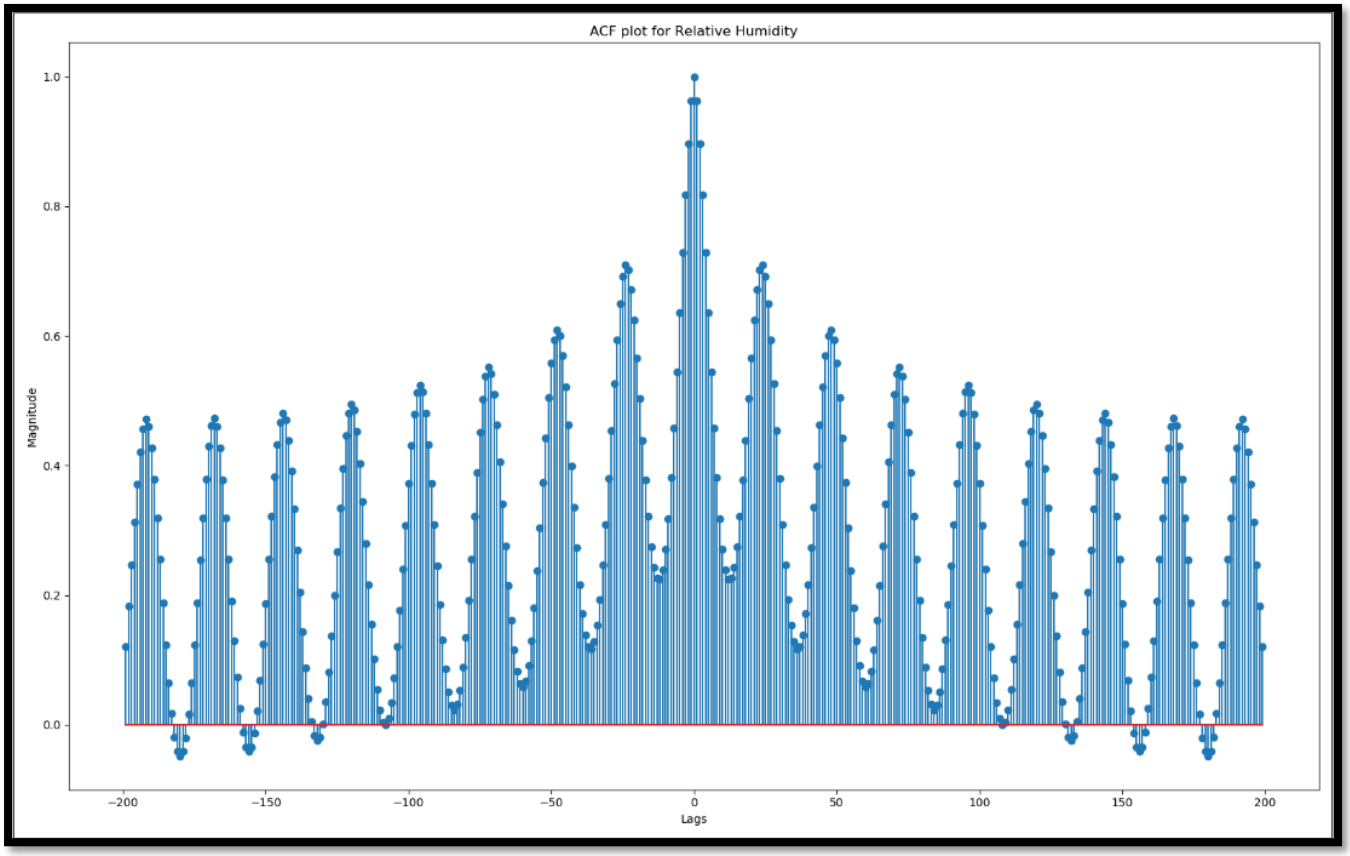
We plot the relative humidity (RH) over time from March 2004 to February 2005. RH data is hourly averaged data from 03/2004 to 04/2005. From the plot we can observe the seasonal pattern where in there is a rise and fall in the data values that repeats regularly over the same time period.



Dependent variable versus time

3.5 ACF of the dependent variable

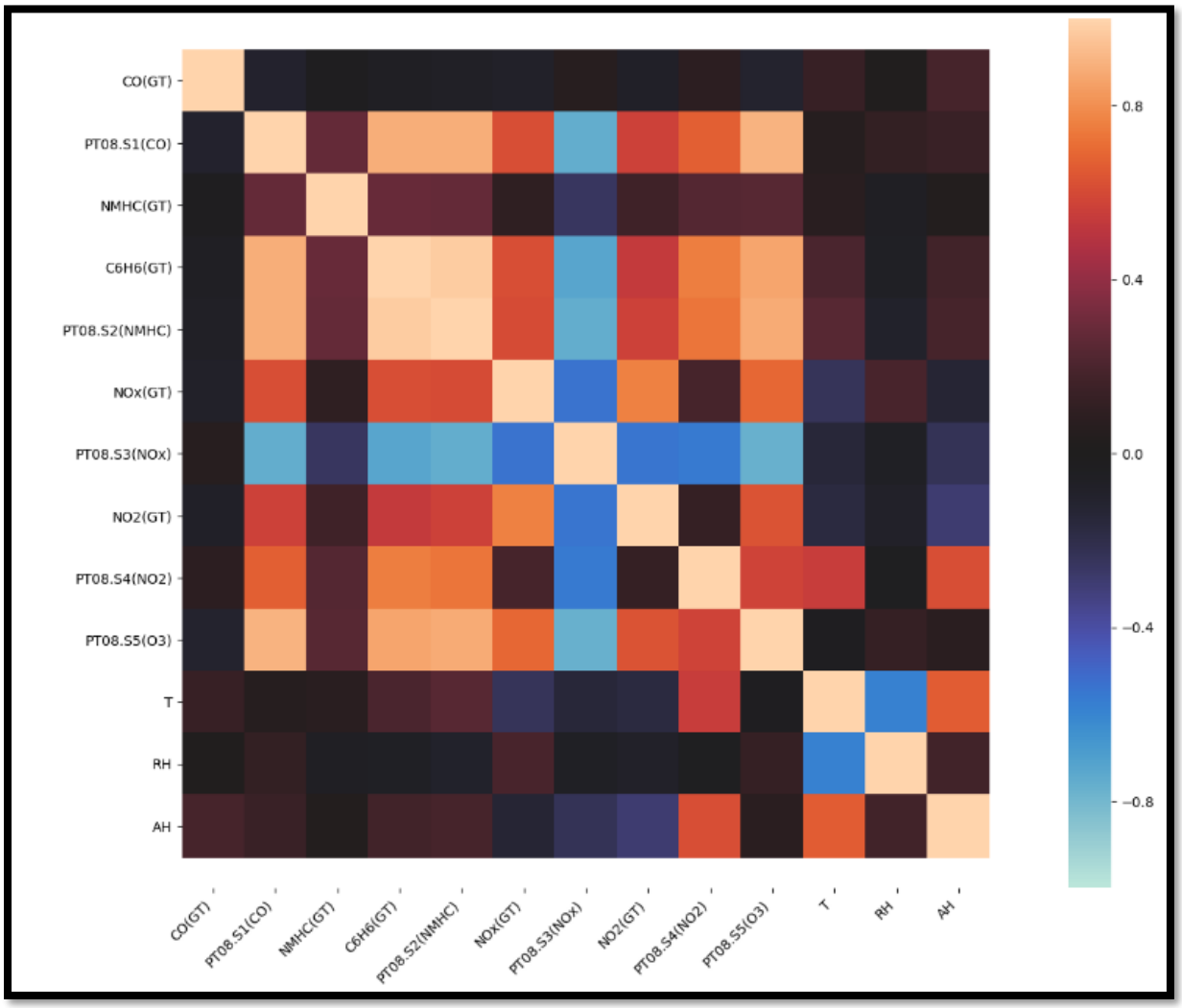
For plotting the ACF plot we have set the value of lag = 200. We notice that the ACF values show decay at lag = 200.



ACF of the dependent variable

3.6 Correlation Matrix

We plot the correlation coefficient matrix for the numerical columns and conclude that there is multicollinearity. Values closer to zero means there is no linear trend between the two variables. The close to 1 the correlation is the more positively correlated they are; that is as one increases so does the

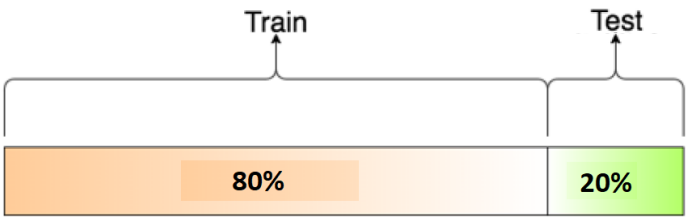


other and the closer to 1 the stronger this relationship is. A correlation closer to -1 is similar, but instead of both increasing one variable will decrease as the other increases. The diagonals are all 1/dark green because those squares are correlating each variable to itself (so it's a perfect correlation). For the rest the larger the number and darker the color the higher the correlation between the two variables. The plot is also symmetrical about the diagonal since the same two variables are being paired together in those squares.

Correlation Coefficient Matrix

3.7 Split Train/Test

We split the cleaned pre-processed data into train and test datasets.



The dimension for train dataset is :

(7485, 13)

The dimension for test dataset is:

(1872, 13)

IV. STATIONARITY

A Time series is said to be stationary if its statistical properties such as mean, variance remain constant over time. To check if relative humidity is stationary or not, we perform the ADF test.

```
#Ho: It is non stationary
#H1: It is stationary
```

ADF Test Statistic : -7.391164190222852

p-value : 7.993979270510891e-11

#Lags Used : 38

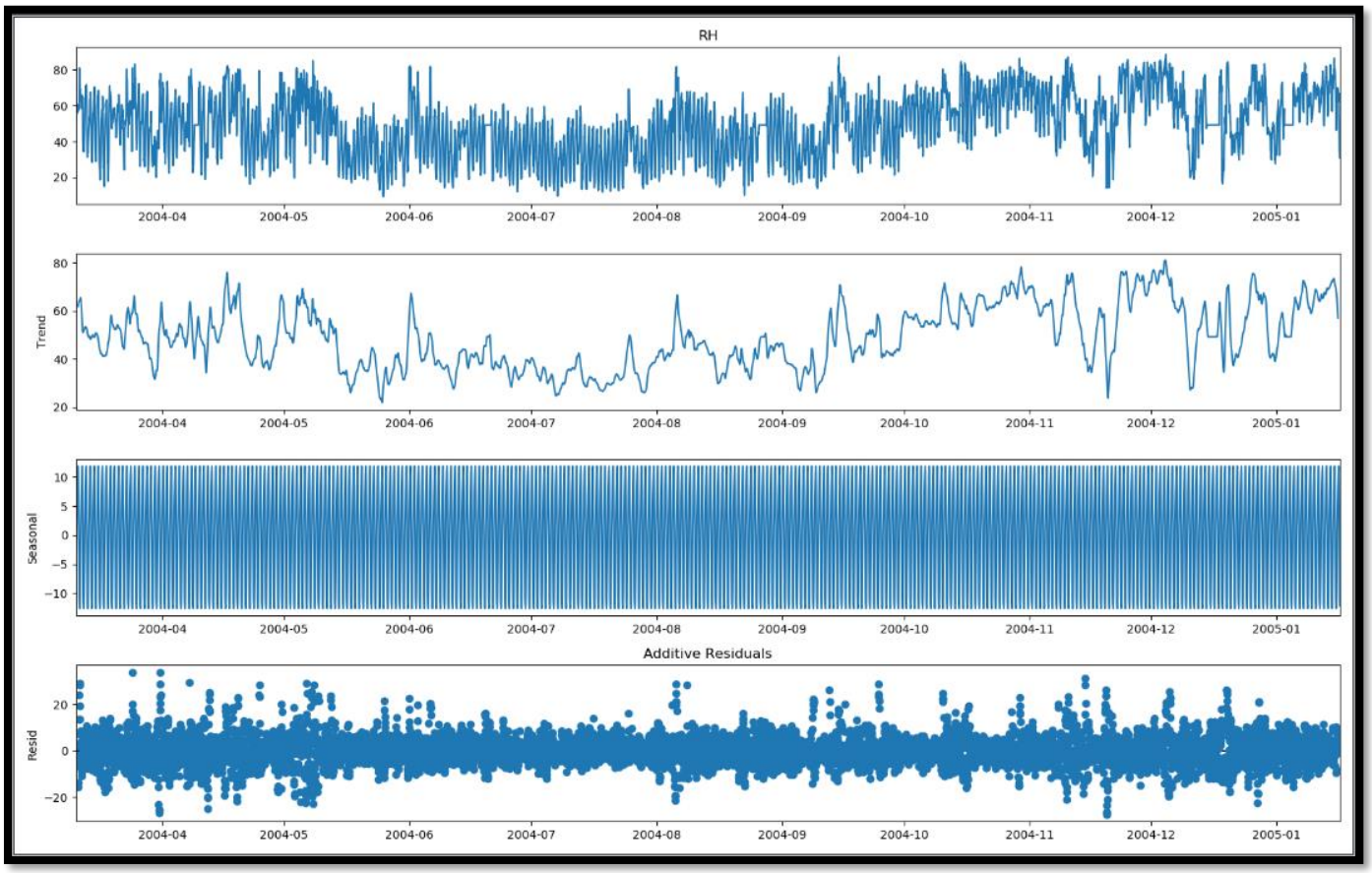
Number of Observations Used : 9318

Strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data has no unit root and is stationary

V. TIME SERIES DECOMPOSITION

5.1 Additive Model

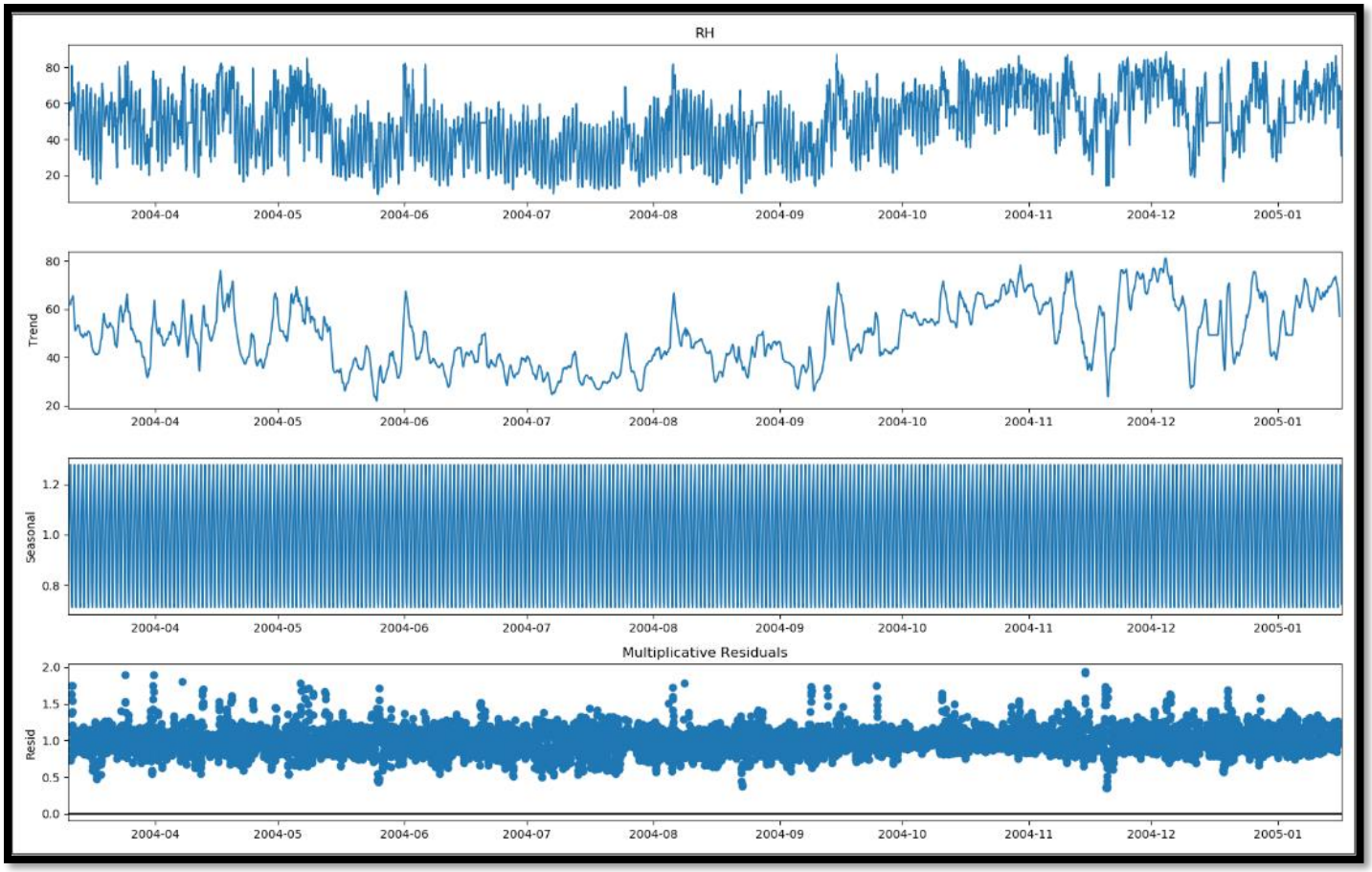
We will decompose relative humidity to comprehend whether trend and seasonality are additive or multiplicative. In additive model all the components are added together i.e. Level + Trend + Seasonality + Noise which is linear where it changes over time are consistent made by the same amount. A linear trend is a straight line and seasonality has the same frequency and amplitude from the plot we notice that the additive residuals have variance ranges from +20 to -20.



Additive time series decomposition

5.2 Multiplicative Model

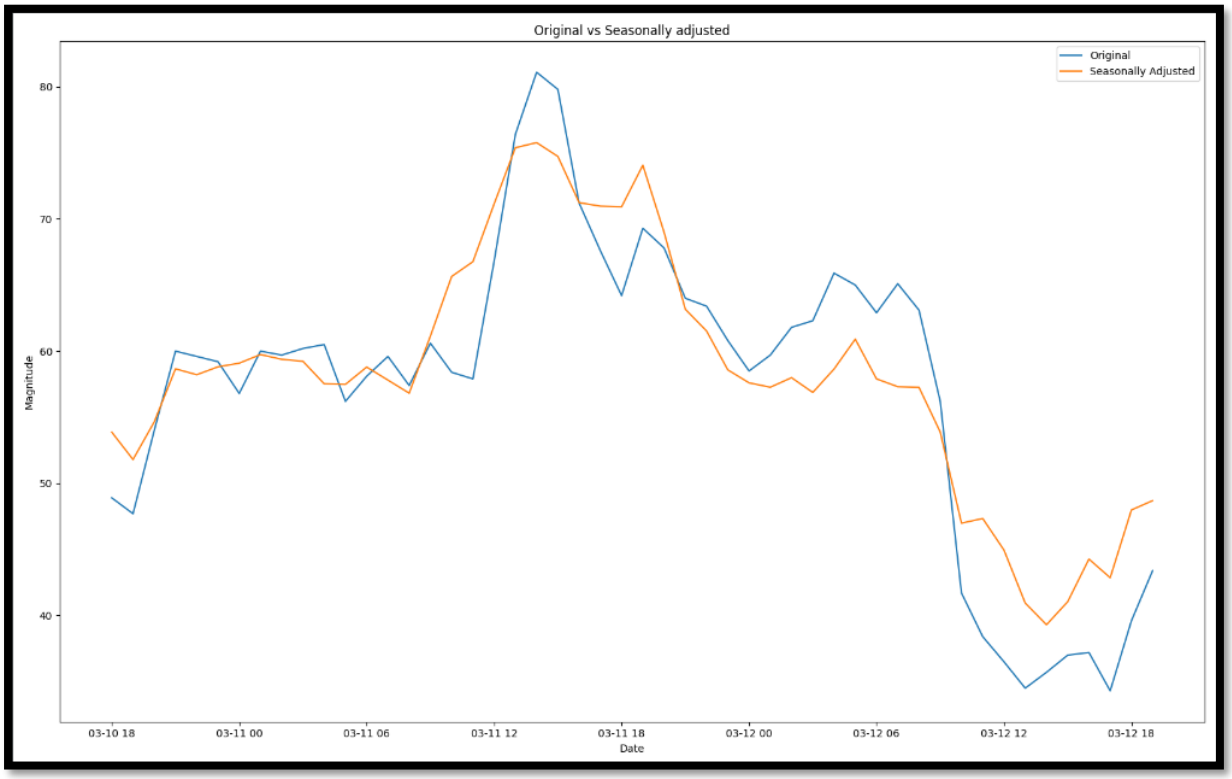
A multiplicative model suggests that the components are multiplied together as Level * Trend * Seasonality * Noise. Here the model is nonlinear, such as quadratic or exponential. Changes increase or decrease over time. A nonlinear trend is a curved line and the seasonality has an increasing or decreasing frequency and/or amplitude over time. The multiplicative decomposition best represents the relative humidity data and we see a strong seasonality component but there is no trend visible. We notice that the multiplicative have variance ranges from +00 to 2.0



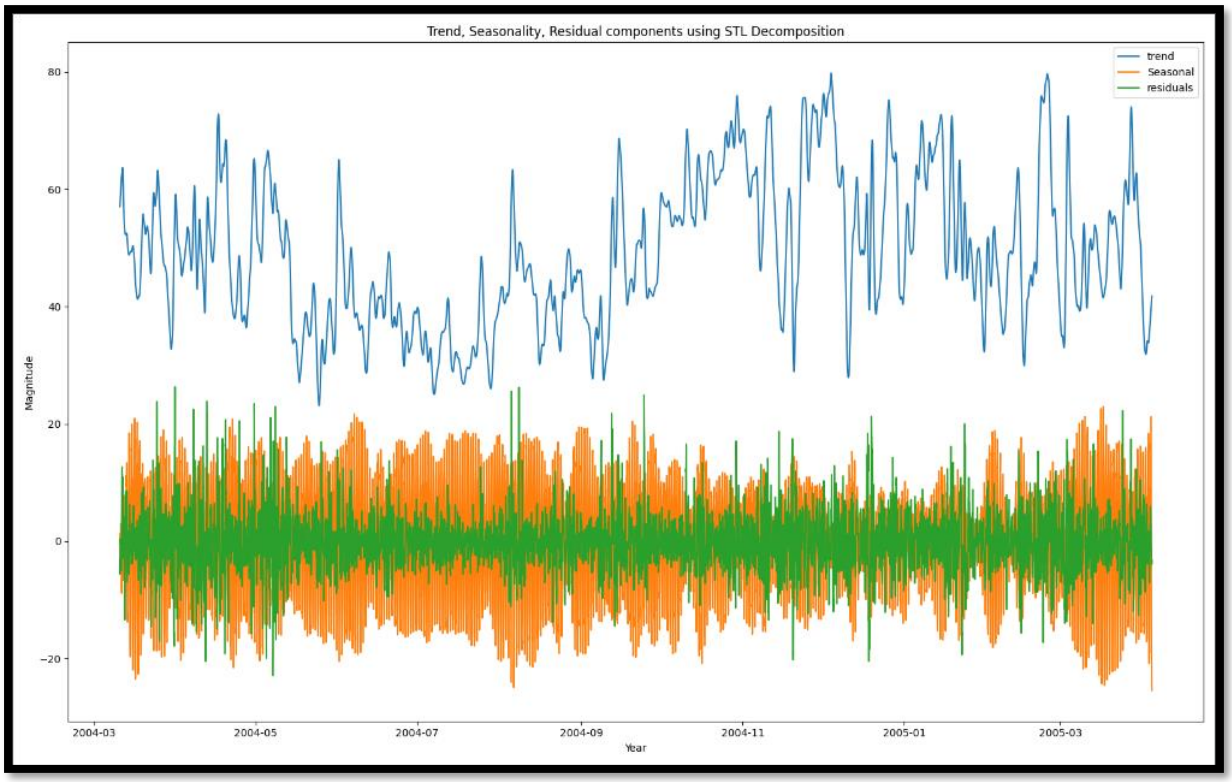
Multiplicative time series decomposition

5.3 Original v/s Seasonally adjusted

Strength of trend for Air quality dataset is **0.879**
Strength of seasonality for Air quality dataset is **0.807**



Comparison of original and seasonally adjusted



Plot of trend seasonal and residual

VI. HOLT- WINTERS METHOD

Based on the time series decomposition we will configure the Holt-Winters parameters for predicting the test data. We will set the seasonality to be multiplicative and set trend to be None.

The performance metrics for Holt Winter model is given below:

MSE: 298.017

RMSE: 17.263168886389312

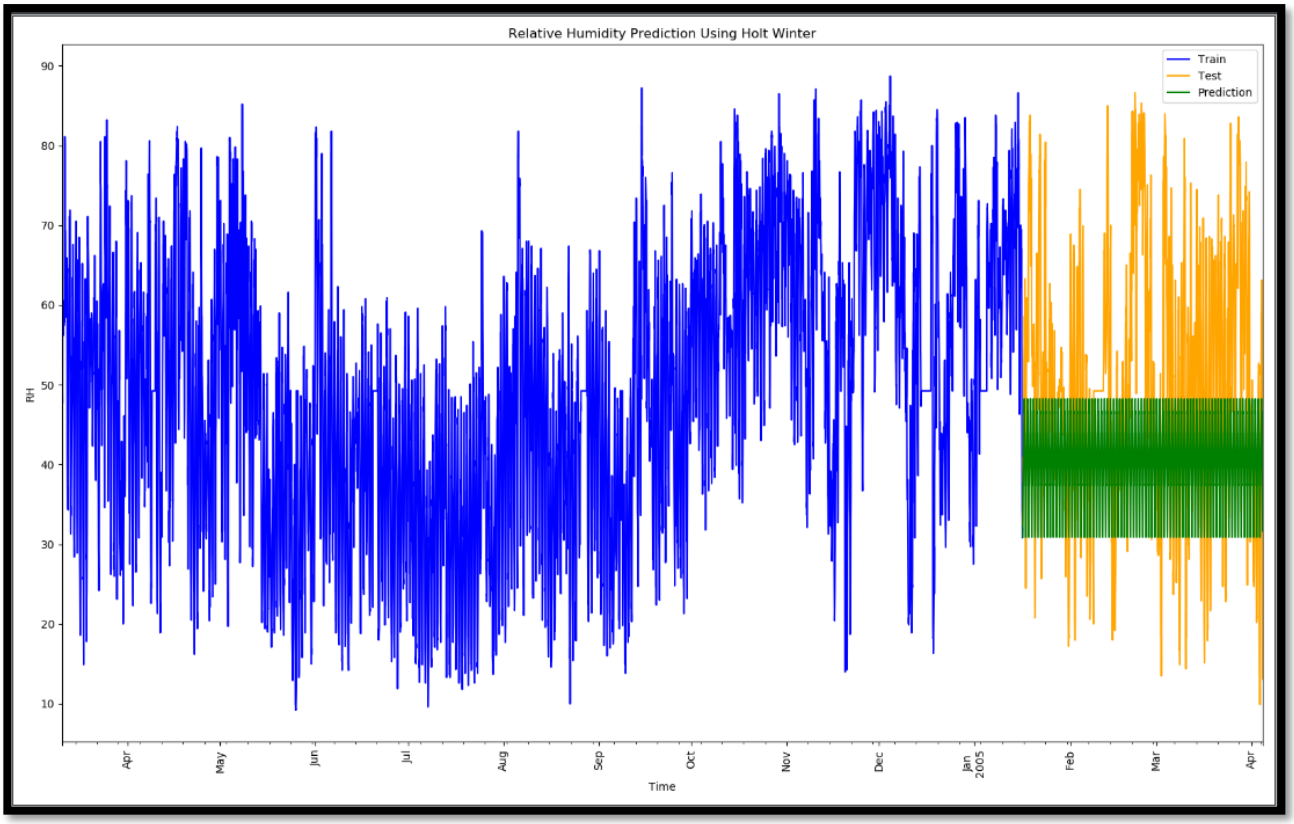
6.1 ACF of residuals:

We notice that the ACF plot for Holt Winter method resembles the white noise.

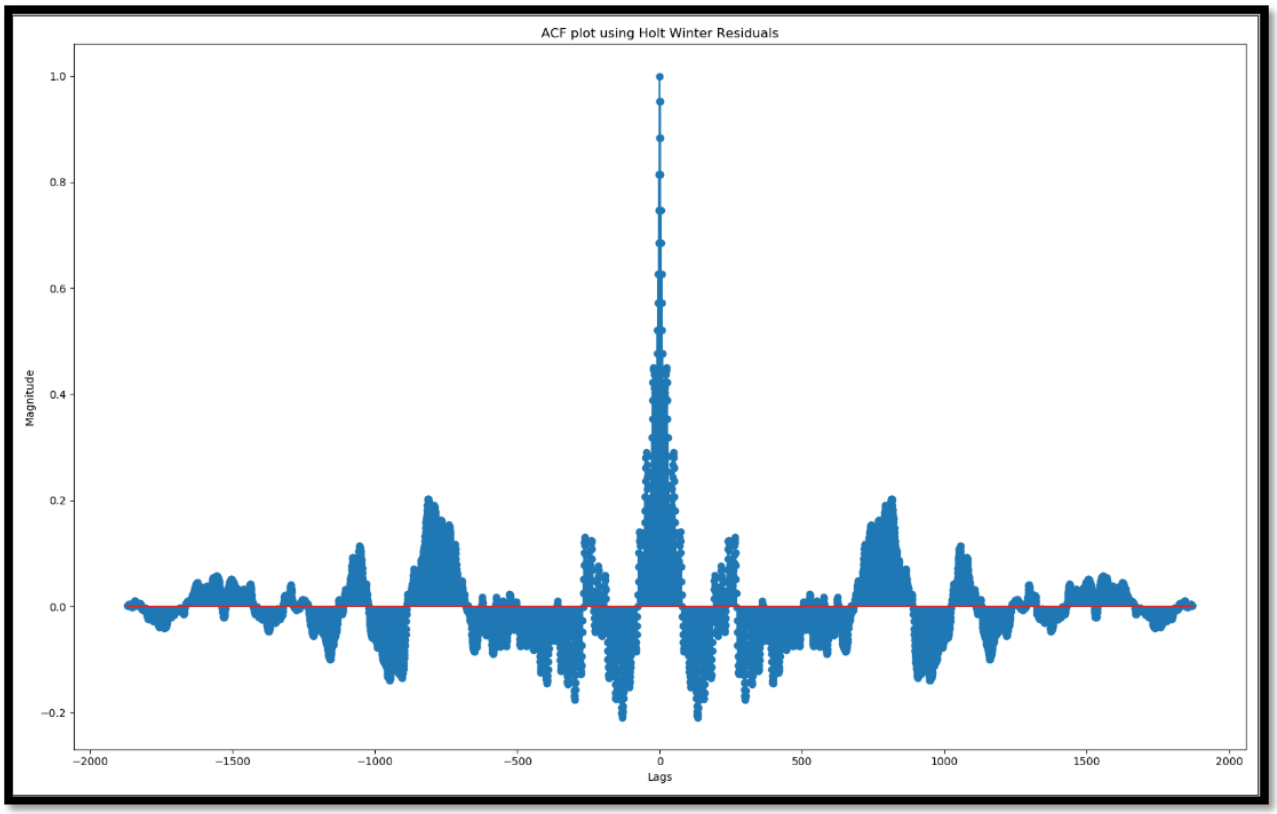
Variance of Residuals: 199.24900904219538

Mean of Residuals: 9.938232855164031

We can infer from the RMSE value and the graph above, that holt-winters method isn't suited for datasets with high variability. It is best suited for stable datasets. We can still improve our score by adopting different techniques. Now we will look at another technique and try to improve our score.



Plot of Prediction: Holt Winter model

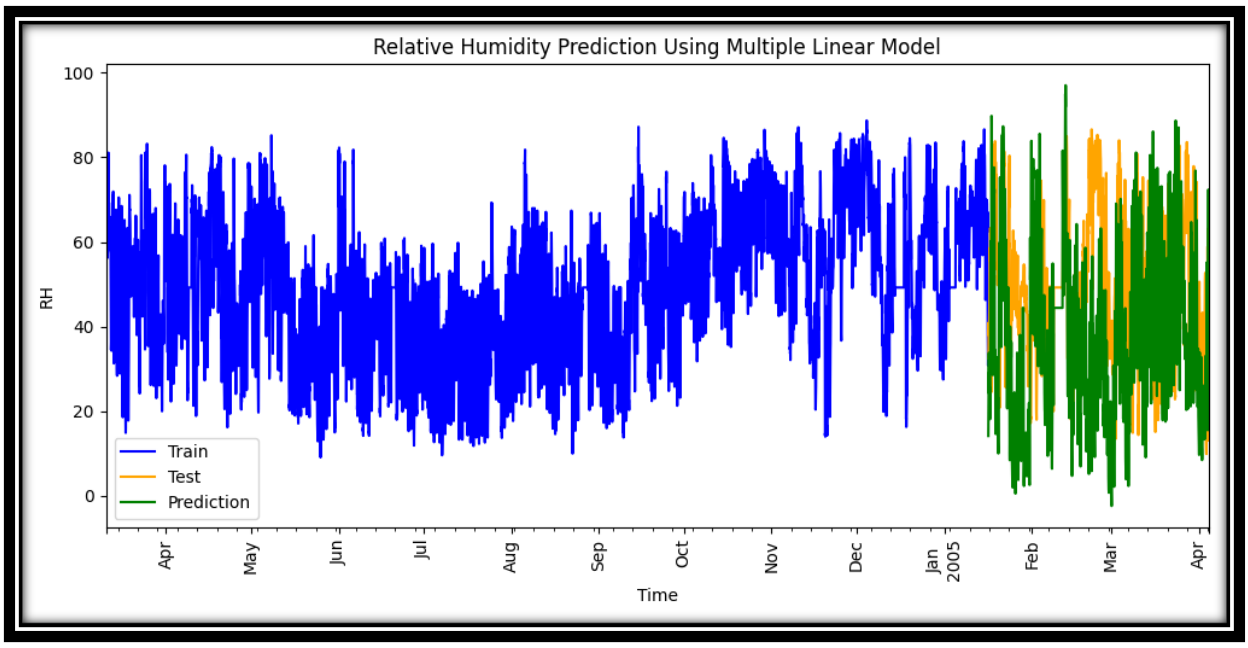


ACF Plot using Holt Winter residuals

VII. FEATURE SELECTION

In this process we are determining the most relevant features for a given problem in order to improve the generalization and the performance. Also, the plot shows the comparison between train, test and predictions.

```
The summary of linear model after feature selection:
                                 OLS Regression Results
==============================================================================================
Dep. Variable:                     RH   R-squared (uncentered):                   0.766
Model:                            OLS   Adj. R-squared (uncentered):              0.766
Method:                 Least Squares   F-statistic:                          1.223e+04
Date:                Wed, 16 Dec 2020   Prob (F-statistic):                        0.00
Time:                        22:49:53   Log-Likelihood:                         -34751.
No. Observations:                7485   AIC:                                  6.951e+04
Df Residuals:                    7483   BIC:                                  6.952e+04
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
PT08.S5(O3)   93.0566      1.324     70.287      0.000      90.461      95.652
T             25.1358      1.046     24.025      0.000      23.085      27.187
==============================================================================
Omnibus:                      104.811   Durbin-Watson:                   0.113
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               65.452
Skew:                          -0.070   Prob(JB):                     6.13e-15
Kurtosis:                       2.564   Cond. No.                         3.23
==============================================================================

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.
```



VIII. MULTIPLE LINEAR REGRESSION

8.1. Linear Model with all Features:

We will now perform multiple linear regression and for this we need to scale the data using sklearn.preprocessing's MinMaxScaler function and then compute the MSE values for the test data and the predicted values.

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                     RH   R-squared (uncentered):              0.979
Model:                            OLS   Adj. R-squared (uncentered):         0.979
Method:                 Least Squares   F-statistic:                     2.876e+04
Date:                Sun, 13 Dec 2020   Prob (F-statistic):                   0.00
Time:                        16:50:06   Log-Likelihood:                    -25759.
No. Observations:                7485   AIC:                             5.154e+04
Df Residuals:                    7473   BIC:                             5.162e+04
Df Model:                          12
Covariance Type:            nonrobust
==================================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------
AH              81.6056      0.826     98.849      0.000      79.987      83.224
C6H6(GT)      -150.5662      3.863    -38.973      0.000    -158.139    -142.993
CO(GT)           0.7902      0.222      3.554      0.000       0.354       1.226
NMHC(GT)        23.5147      1.489     15.796      0.000      20.596      26.433
NO2(GT)          7.6150      1.247      6.107      0.000       5.171      10.059
NOx(GT)         24.2378      1.432     16.923      0.000      21.430      27.045
PT08.S1(CO)      9.7269      1.608      6.050      0.000       6.575      12.879
PT08.S2(NMHC)   95.1796      3.151     30.205      0.000      89.003     101.357
PT08.S3(NOx)    72.7674      0.824     88.325      0.000      71.152      74.382
PT08.S4(NO2)    33.1811      1.505     22.049      0.000      30.231      36.131
PT08.S5(O3)     17.7374      1.518     11.683      0.000      14.761      20.713
T              -99.0968      0.773   -128.132      0.000    -100.613     -97.581
==============================================================================
Omnibus:                      252.640   Durbin-Watson:                   0.210
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              657.734
Skew:                           0.119   Prob(JB):                    1.50e-143
Kurtosis:                       4.433   Cond. No.                         62.3
==============================================================================

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.
```

Analysis of above model:

F-test: The Prob(F-statistics) < 0.05 thus our model performs better than the null model and it passes the F-test.

AIC: The AIC value is 5.154e+04.

BIC: The BIC value is 5.162e+04.

T-test: The t-test passes for all variables since the P(t) < 0.05 for all variables.

R-Squared: The R-squared value is 0.979

Adjusted R-squared: The adjusted R-squared value is 0.979.

MSE: **98.319**

RMSE: **9.915593779497021**

8.2 ACF of Residuals

We observe the residuals are decaying in ACF plot, but they do not resemble a white noise.

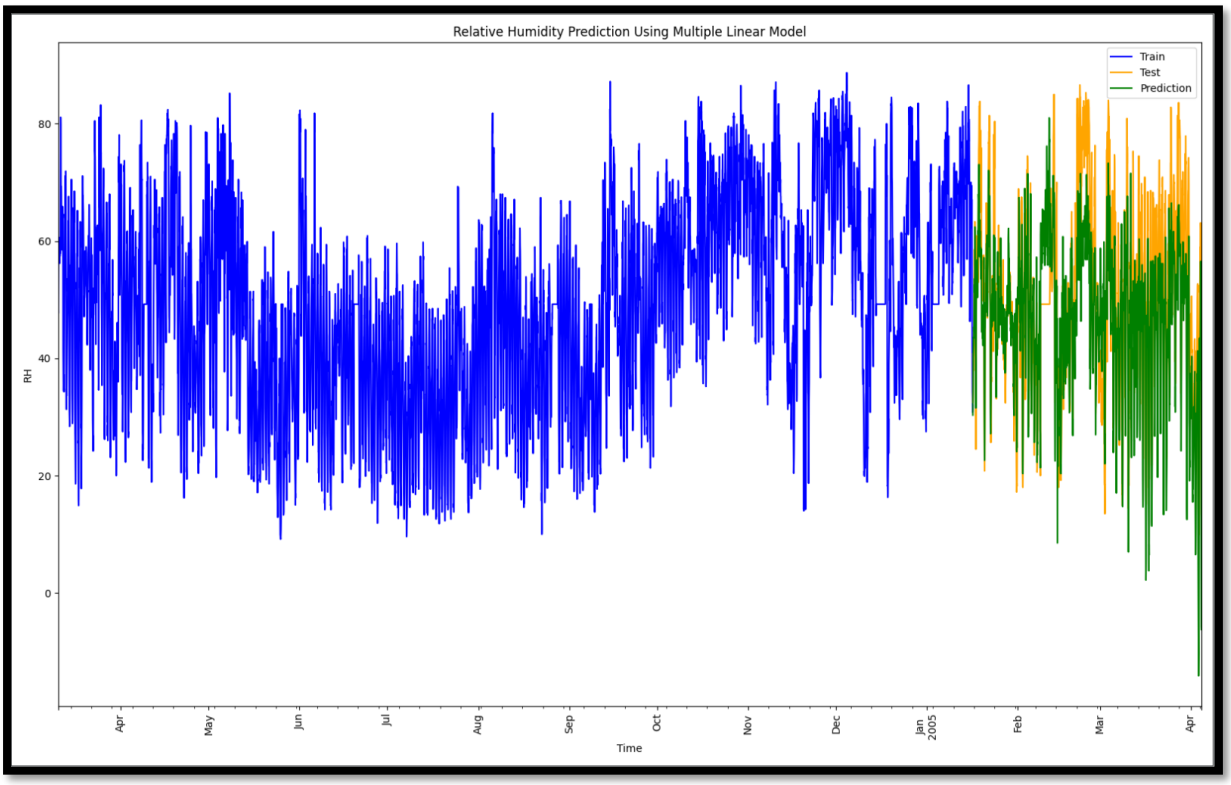
Variance of Residual: **84.01089803688629**

Mean of Residual: **3.782570390088166**

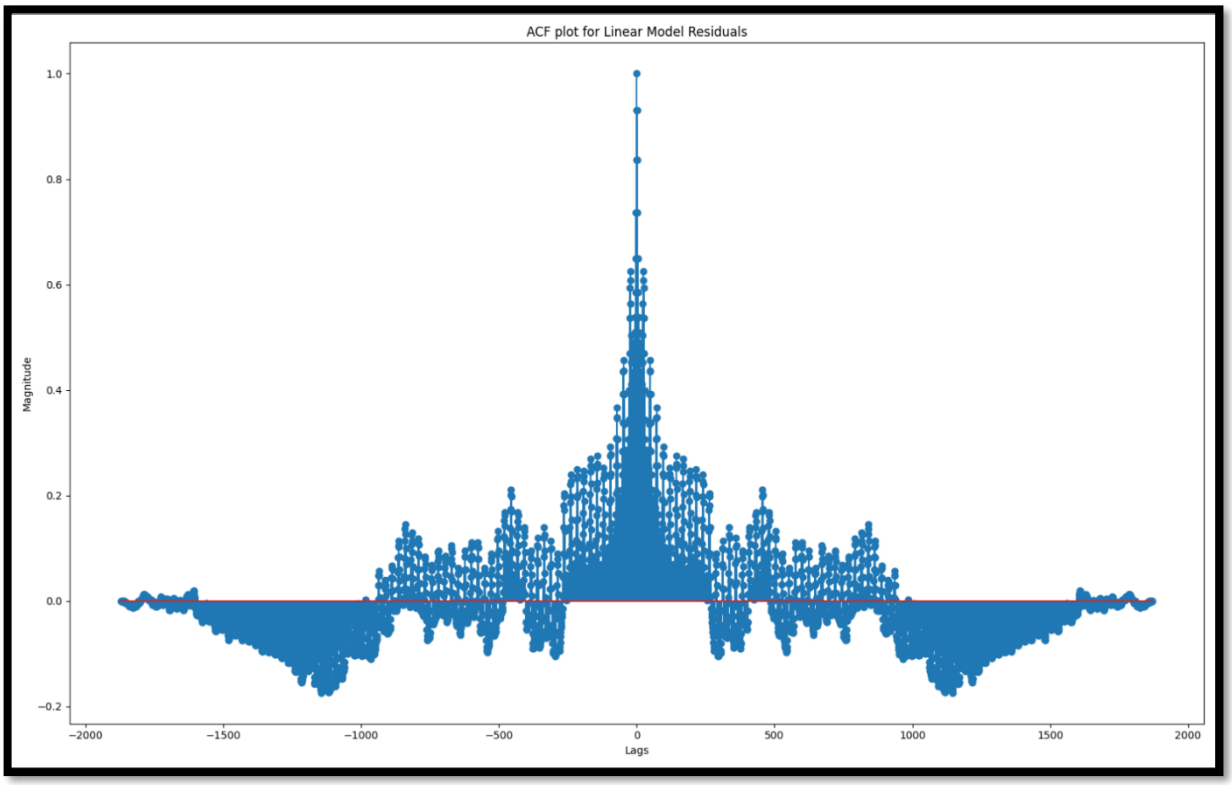
Q value of residuals: **1333.365**

We can infer from the RMSE value and the graph above, that multiple regression method is better than holt-winters method and it is suited for datasets with high variability. Let's see if we can still improve our score by adopting different techniques. Now we will look at another technique and try to improve our score.

The plot for linear model prediction along with actual predictions is shown below:



Plot of Prediction: Linear Regression



Plot of ACF: Regression residuals

IX. ARMA MODEL (1)

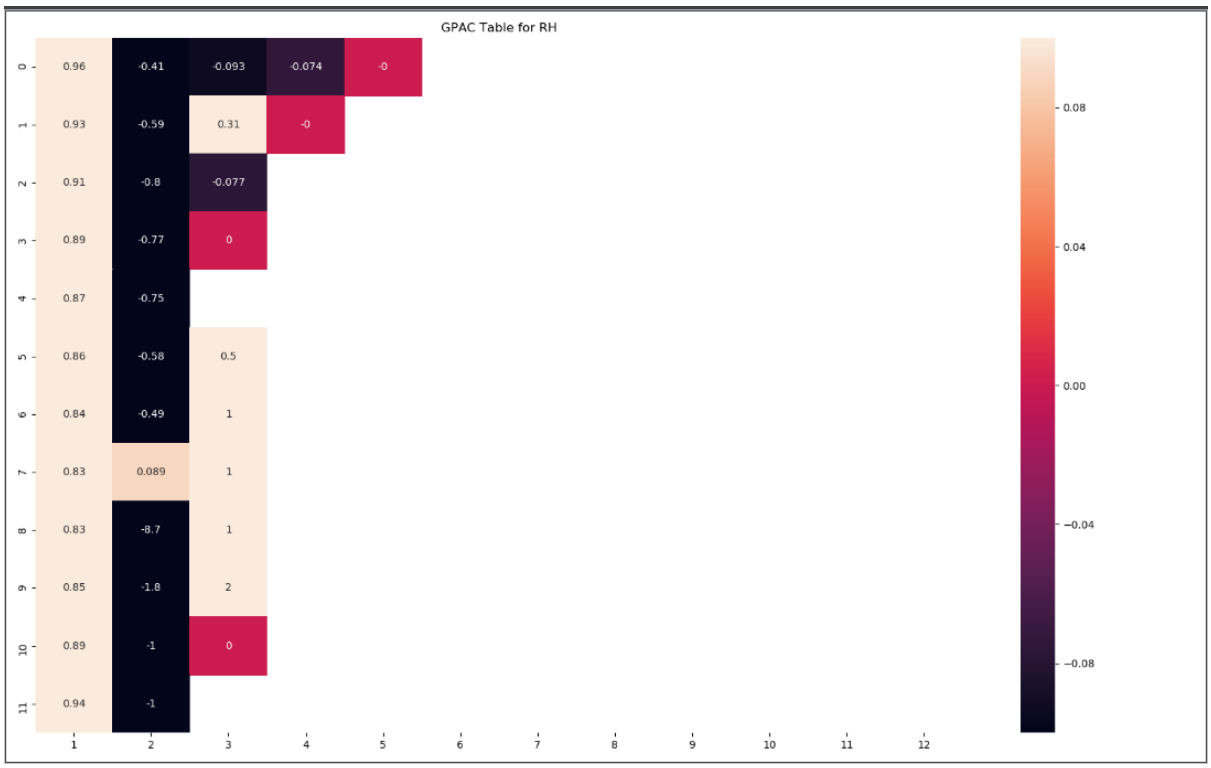
We will now predict the test data using the ARMA process. For this we will create the GPAC table and then find potential order of the ARMA process. After we find the order of ARMA process we will estimate its parameters.

We computed the mean of training data and subtracted it from the training data. This is done to relax the ARMA constraint. Since the ARMA model does not include an intercept, it might be a challenge to fit data with non-zero mean.

Once the order and parameters are estimated we will forecast the values and add the mean of training data. After adding mean, we check whether the residuals of forecasted and actual values are significant or not using the chi squared diagnostic test.

9.1 GPAC Table

The GPAC table with j=12, and k=12 is shown below:



From the GPAC table we consider the following orders for ARMA parameter estimation.

GPAC Table:

	1	2	3	4	5	6	7	8	9	10	11	12
0	0.96257	-0.40722	-0.09333	-0.07407	-0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	0.93151	-0.59044	0.30952	-0.00000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	0.91105	-0.80294	-0.07692	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	0.89167	-0.77010	0.00000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	0.87332	-0.74817	inf	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	0.85569	-0.58384	0.50000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
6	0.84191	-0.49266	1.00000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
7	0.83248	0.08936	1.00000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
8	0.83366	-8.66667	1.00000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
9	0.84864	-1.79121	2.00000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
10	0.88662	-1.04908	0.00000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
11	0.93962	-1.04386	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

The possible orders identified from GPAC for ARMA process are:

(n_a, n_b) = [(2, 3), (2, 7),(3,2),(3,6),(4, 0), (4, 2)]

9.2 Parameter Estimation

The estimated parameters based on the order n_a = 2 and n_b = 7 is:

ar.L1.RH	1.7873
ar.L2.RH	-0.7879
ma.L1.RH	-0.5042
ma.L2.RH	-0.1672
ma.L3.RH	-0.0464
ma.L4.RH	-0.0494
ma.L5.RH	-0.0370
ma.L6.RH	-0.0752
ma.L7.RH	-0.0784

9.3 ARMA (2,7) Model

Summary of ARMA (2,7) model is as follows. We can infer from the RMSE value that ARMA(2,7)

```
                              ARMA Model Results
==============================================================================
Dep. Variable:                     RH   No. Observations:                 9357
Model:                     ARMA(2, 7)   Log Likelihood              -26515.448
Method:                       css-mle   S.D. of innovations              4.115
Date:                Mon, 14 Dec 2020   AIC                          53050.897
Time:                        18:47:00   BIC                          53122.336
Sample:                    03-10-2004   HQIC                         53075.159
                         - 04-04-2005
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1.RH       1.7873      0.012    147.293      0.000       1.763       1.811
ar.L2.RH      -0.7879      0.012    -65.494      0.000      -0.812      -0.764
ma.L1.RH      -0.5042      0.015    -32.988      0.000      -0.534      -0.474
ma.L2.RH      -0.1672      0.012    -14.160      0.000      -0.190      -0.144
ma.L3.RH      -0.0464      0.012     -3.984      0.000      -0.069      -0.024
ma.L4.RH      -0.0494      0.013     -3.943      0.000      -0.074      -0.025
ma.L5.RH      -0.0370      0.012     -3.061      0.002      -0.061      -0.013
ma.L6.RH      -0.0752      0.012     -6.331      0.000      -0.098      -0.052
ma.L7.RH      -0.0784      0.011     -7.099      0.000      -0.100      -0.057
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            1.0032           +0.0000j            1.0032            0.0000
AR.2            1.2652           +0.0000j            1.2652            0.0000
MA.1            1.0174           -0.0000j            1.0174           -0.0000
MA.2            0.9048           -1.0446j            1.3819           -0.1364
MA.3            0.9048           +1.0446j            1.3819            0.1364
MA.4           -0.3780           -1.5056j            1.5523           -0.2891
MA.5           -0.3780           +1.5056j            1.5523            0.2891
MA.6           -1.5148           -0.6545j            1.6502           -0.4351
MA.7           -1.5148           +0.6545j            1.6502            0.4351
-----------------------------------------------------------------------------
```

method isn't suited for datasets with high variability, though the graph prediction matches and the best so far. We can still improve our score by adopting different orders. Now we will look at another order and try to improve our score.

9.4 Diagnostic Analysis

9.4.1 Chi Square Test

We noticed that four out of the six identified ARMA order from the GPAC table pass the chi squared test.

The residual is white, and the estimated order is n_a= 2 and n_b = 3

The residual is white, and the estimated order is n_a= 2 and n_b = 7

The residual is white, and the estimated order is n_a= 4 and n_b = 0

The residual is white, and the estimated order is n_a= 4 and n_b = 2

[(2, 3), (2, 7), (4, 0), (4, 2)]

9.4.2 Simplification of Model

We will now simplify the ARMA(2,7) model by checking if zeros are included in confidence interval or not. The confidence interval for the parameters are as shown below. We notice there are no zeros in the confidence interval band. Thus, no simplification needed.

	coef	[0.025	0.975]
ar.L1.RH	1.7873	1.763	1.811
ar.L2.RH	-0.7879	-0.812	-0.764
ma.L1.RH	-0.5042	-0.534	-0.474
ma.L2.RH	-0.1672	-0.190	-0.144
ma.L3.RH	-0.0464	-0.069	-0.024
ma.L4.RH	-0.0494	-0.074	-0.025
ma.L5.RH	-0.0370	-0.061	-0.013
ma.L6.RH	-0.0752	-0.098	-0.052
ma.L7.RH	-0.0784	-0.100	-0.057

9.4.3 Zero/pole cancellation

	Real	Imaginary	Modulus	Frequency
AR.1	1.0032	+0.0000j	1.0032	0.0000
AR.2	1.2652	+0.0000j	1.2652	0.0000
MA.1	1.0174	-0.0000j	1.0174	-0.0000
MA.2	0.9048	-1.0446j	1.3819	-0.1364
MA.3	0.9048	+1.0446j	1.3819	0.1364
MA.4	-0.3780	-1.5056j	1.5523	-0.2891
MA.5	-0.3780	+1.5056j	1.5523	0.2891
MA.6	-1.5148	-0.6545j	1.6502	-0.4351
MA.7	-1.5148	+0.6545j	1.6502	0.4351

We will also simplify model based on zero/pole cancellation by checking the roots of numerator and denominator. The roots of the AR and MA process are shown below. None of the roots are same, thus no zero/pole cancellation required. Hence the final ARMA model after simplification is ARMA(2,7).

9.4.4 Performance measures

The possible reason for poor performance of ARMA model is that our data contains seasonality.

MSE: 993.428

RMSE: 31.518692866297613

Mean of Residual Errors: 2.496568195729656

Variance of Residual Errors: 987.1946958235831

Biased or Unbiased models: Since the absolute value of mean of the residuals is greater than 0.05, we say that the model is biased. We can remove the bias by adding the mean to all the predictions.

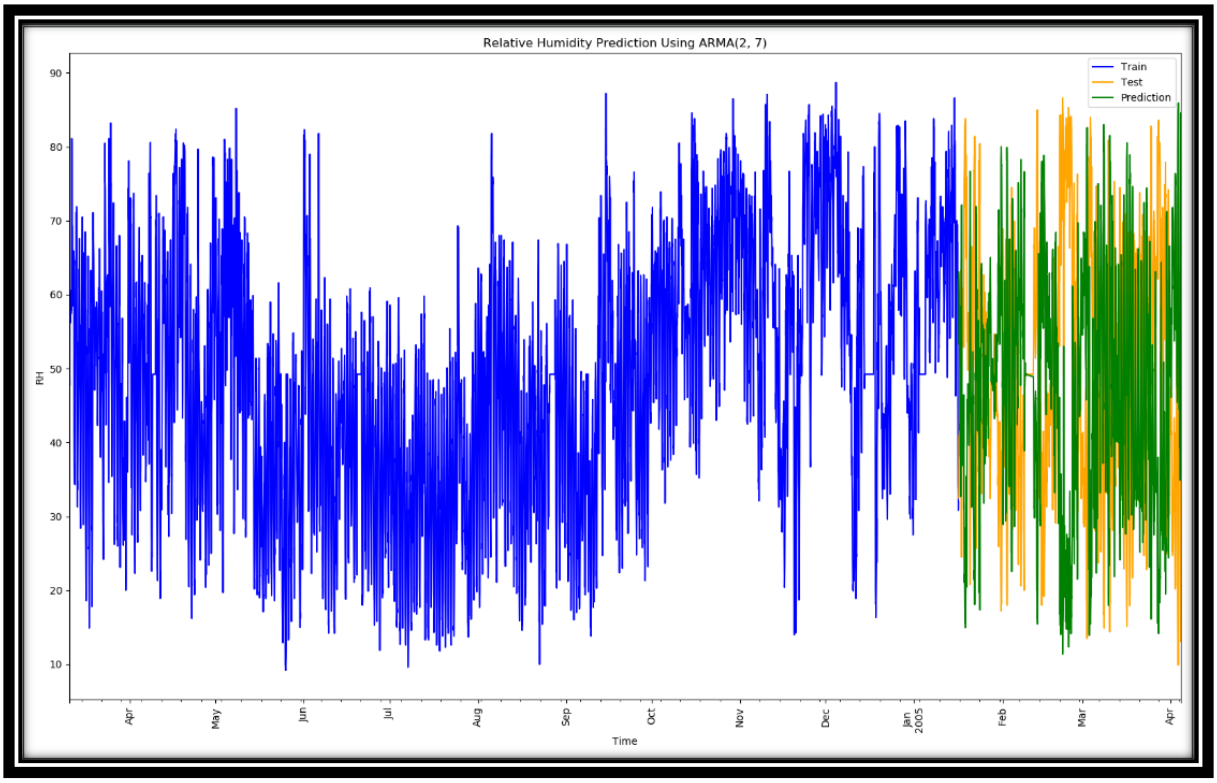
Variance of error of estimated parameters:

Estimated variance of error for na = 2 and nb = 7 is 16.93306246365715

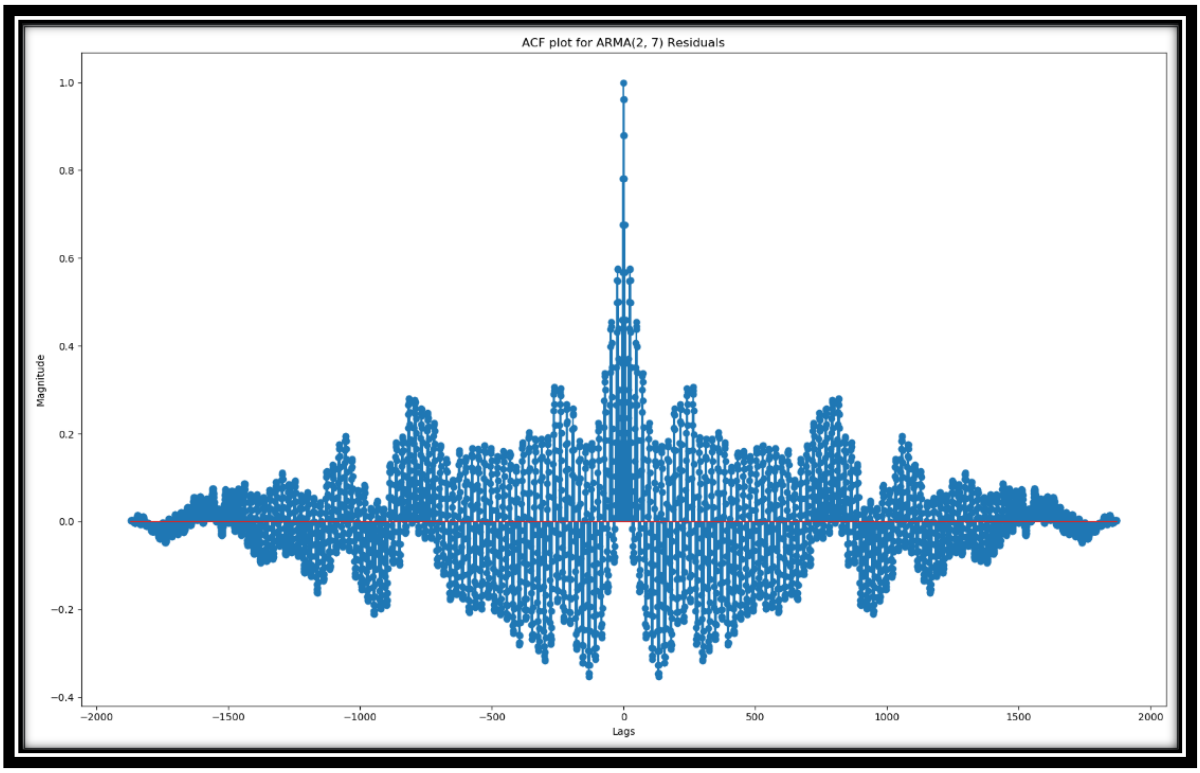
Covariance of estimated Parameters:

	ar.L1.RH	ar.L2.RH	ma.L1.RH	...	ma.L5.RH	ma.L6.RH	ma.L7.RH
ar.L1.RH	0.000147	-0.000146	-0.000136	...	0.000019	0.000025	0.000044
ar.L2.RH	-0.000146	0.000145	0.000135	...	-0.000019	-0.000025	-0.000044
ma.L1.RH	-0.000136	0.000135	0.000234	...	-0.000026	-0.000036	-0.000037
ma.L2.RH	-0.000029	0.000028	-0.000029	...	-0.000016	0.000007	-0.000016
ma.L3.RH	0.000006	-0.000007	-0.000026	...	0.000002	-0.000006	-0.000003
ma.L4.RH	0.000012	-0.000012	-0.000019	...	-0.000044	-0.000008	-0.000013
ma.L5.RH	0.000019	-0.000019	-0.000026	...	0.000147	-0.000054	-0.000012
ma.L6.RH	0.000025	-0.000025	-0.000036	...	-0.000054	0.000141	-0.000049
ma.L7.RH	0.000044	-0.000044	-0.000037	...	-0.000012	-0.000049	0.000122

The plot of forecasted values with the actual value and ACF is shown below.



Plot of Prediction: ARMA (2,7) Model



Plot of ACF:: ARMA(2,7) Residuals

X. ARMA MODEL (2)

Summary of ARMA (2,3) model is as follows. We can infer from the RMSE value that ARMA(2,3) method isn't suited for datasets with high variability, though the graph prediction matches and the best so far. We can still improve our score by adopting different orders. Now we will look at another order and try to improve our score.

```
                              ARMA Model Results
==============================================================================
Dep. Variable:                     RH   No. Observations:                 9357
Model:                     ARMA(2, 3)   Log Likelihood              -26582.179
Method:                       css-mle   S.D. of innovations              4.145
Date:                Tue, 15 Dec 2020   AIC                          53176.357
Time:                        01:46:18   BIC                          53219.221
Sample:                    03-10-2004   HQIC                         53190.914
                         - 04-04-2005
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1.RH       1.6624      0.021     80.623      0.000       1.622       1.703
ar.L2.RH      -0.7029      0.019    -36.581      0.000      -0.741      -0.665
ma.L1.RH      -0.3580      0.023    -15.738      0.000      -0.403      -0.313
ma.L2.RH      -0.0698      0.013     -5.368      0.000      -0.095      -0.044
ma.L3.RH       0.0477      0.012      4.027      0.000       0.024       0.071
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            1.1826           -0.1558j            1.1928           -0.0208
AR.2            1.1826           +0.1558j            1.1928            0.0208
MA.1           -3.1120           -0.0000j            3.1120           -0.5000
MA.2            2.2869           -1.2257j            2.5947           -0.0783
MA.3            2.2869           +1.2257j            2.5947            0.0783
-----------------------------------------------------------------------------
```

10.1 Diagnostic Analysis

10.1.1 Chi Square Test

We noticed that four out of the six identified ARMA order from the GPAC table pass the chi squared test.

The residual is white, and the estimated order is n_a= 2 and n_b = 3

The residual is white, and the estimated order is n_a= 2 and n_b = 7

The residual is white, and the estimated order is n_a= 4 and n_b = 0

The residual is white, and the estimated order is n_a= 4 and n_b = 2

[(2, 3), (2, 7), (4, 0), (4, 2)]

10.1.2 Simplification of Model

We will now simplify the ARMA(2,3) model by checking if zeros are included in confidence interval or not. The confidence interval for the parameters are as shown below. We notice there are no zeros in the

	coef	[0.025	0.975]
ar.L1.RH	1.6624	1.622	1.703
ar.L2.RH	-0.7029	-0.741	-0.665
ma.L1.RH	-0.3580	-0.403	-0.313
ma.L2.RH	-0.0698	-0.095	-0.044
ma.L3.RH	0.0477	0.024	0.071

confidence interval band. Thus, no simplification needed.

10.1.3 Zero/pole cancellation

We will also simplify model based on zero/pole cancellation by checking the roots of numerator and denominator. The roots of the AR and MA process are shown below. None of the roots are same, thus no zero/pole cancellation required. Hence the final ARMA model after simplification is ARMA(2,3).

	Real	Imaginary	Modulus	Frequency
AR.1	1.1826	-0.1558j	1.1928	-0.0208
AR.2	1.1826	+0.1558j	1.1928	0.0208
MA.1	-3.1120	-0.0000j	3.1120	-0.5000
MA.2	2.2869	-1.2257j	2.5947	-0.0783
MA.3	2.2869	+1.2257j	2.5947	0.0783

10.1.4 Performance measures

The possible reason for poor performance of ARMA model is that our data contains seasonality.

MSE: 1003.625

RMSE: 31.68004103532696

Mean of Residual Errors: 2.3864128547219856

Variance of Residual Errors: 997.9304283419611

Biased or Unbiased models: Since the absolute value of mean of the residuals is greater than 0.05, we say that the model is biased. We can remove the bias by adding the mean to all the predictions.

Variance of error of estimated parameters:

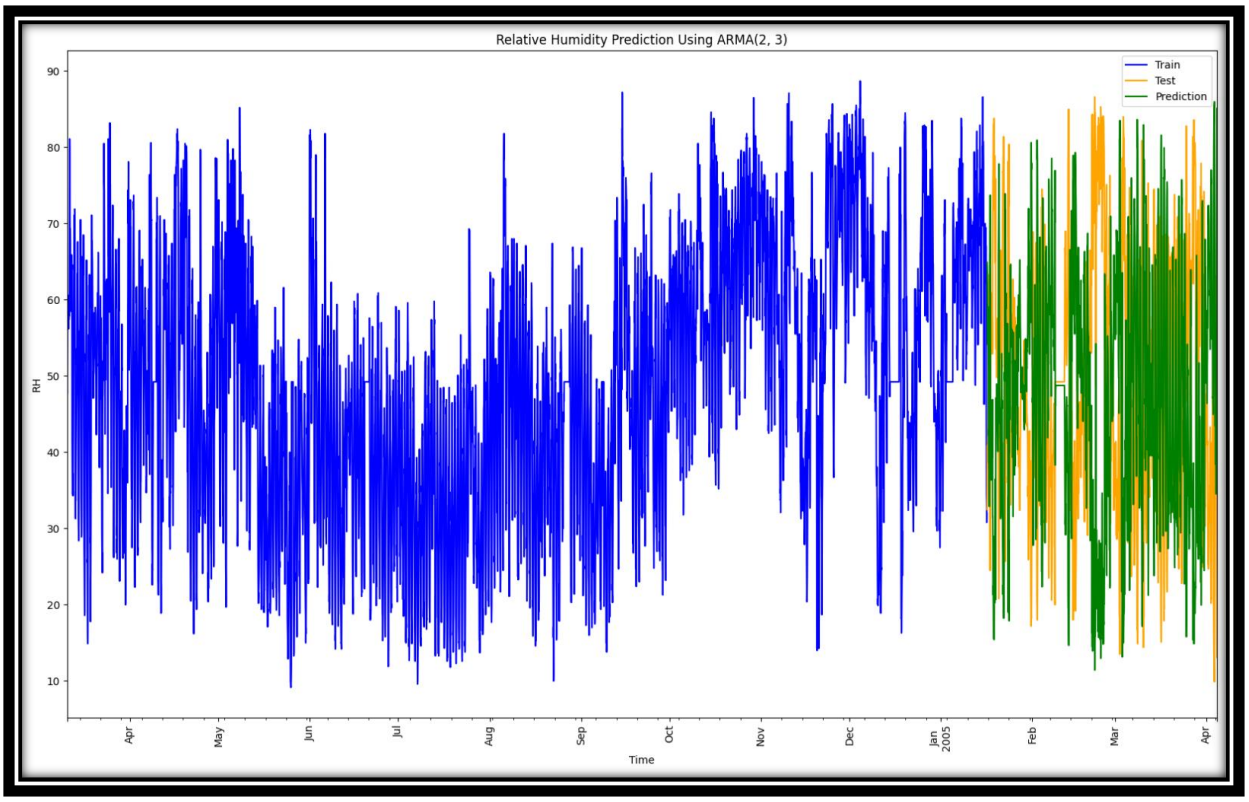
Estimated variance of error for na = 2 and nb = 3 is 17.176981270031806

Covariance of estimated Parameters:

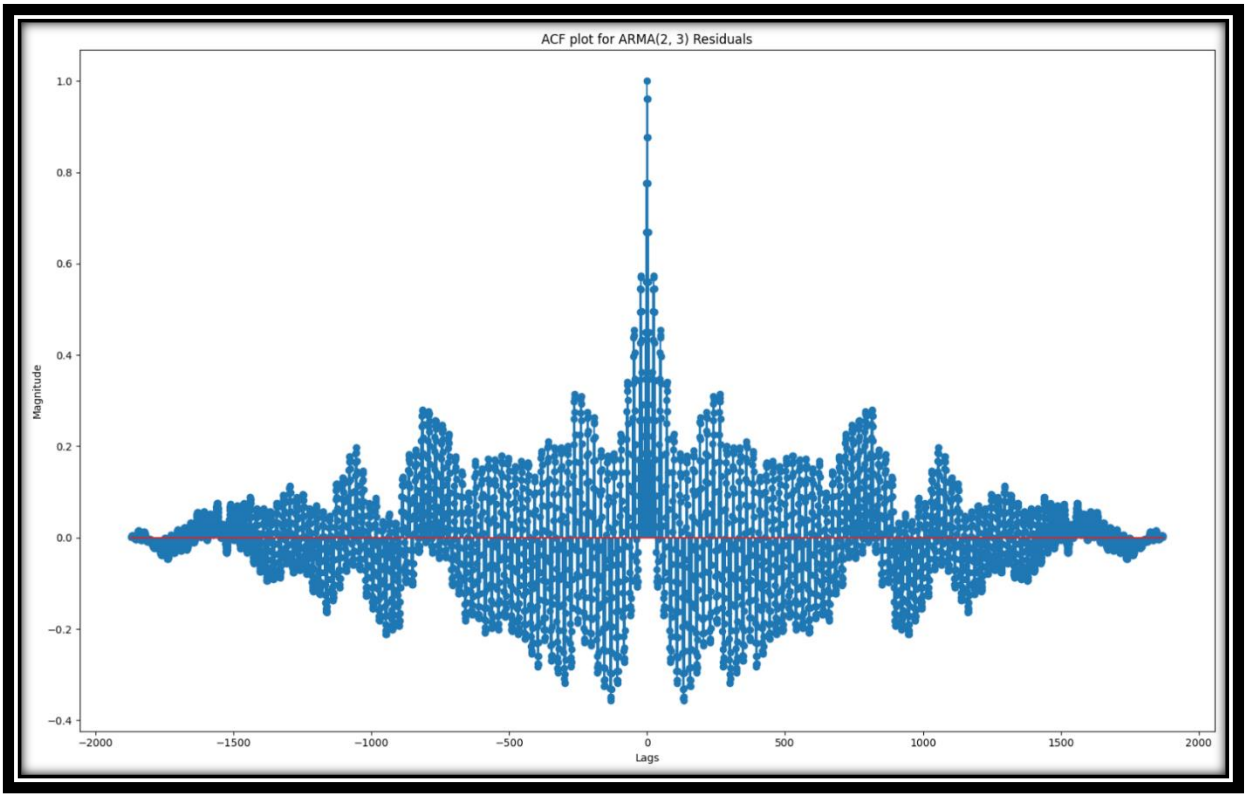
	ar.L1.RH	ar.L2.RH	ma.L1.RH	ma.L2.RH	ma.L3.RH
ar.L1.RH	0.000425	-0.000394	-0.000417	-0.000148	-0.000071
ar.L2.RH	-0.000394	0.000369	0.000388	0.000135	0.000059
ma.L1.RH	-0.000417	0.000388	0.000518	0.000109	0.000055
ma.L2.RH	-0.000148	0.000135	0.000109	0.000169	-0.000011
ma.L3.RH	-0.000071	0.000059	0.000055	-0.000011	0.000141

	ar.L1.RH	ar.L2.RH	ma.L1.RH	ma.L2.RH	ma.L3.RH
ar.L1.RH	0.000425	-0.000394	-0.000417	-0.000148	-0.000071
ar.L2.RH	-0.000394	0.000369	0.000388	0.000135	0.000059
ma.L1.RH	-0.000417	0.000388	0.000518	0.000109	0.000055
ma.L2.RH	-0.000148	0.000135	0.000109	0.000169	-0.000011
ma.L3.RH	-0.000071	0.000059	0.000055	-0.000011	0.000141

The plot of forecasted values with the actual value and ACF is shown below



Plot of Prediction: ARMA (2,3) Model



Plot of ACF:: ARMA(2,3) Residuals

XI. ARMA MODEL (3)

Summary of ARMA (4,0) model is as follows. We can infer from the RMSE value that ARMA(4,0) method isn't suited for datasets with high variability, though the graph prediction matches and the best so far. We can still improve our score by adopting different orders. Now we will look at another order and try to improve our score.

```
                              ARMA Model Results
==============================================================================
Dep. Variable:                     RH   No. Observations:                 9357
Model:                     ARMA(4, 0)   Log Likelihood              -26590.828
Method:                       css-mle   S.D. of innovations              4.148
Date:                Tue, 15 Dec 2020   AIC                          53191.656
Time:                        02:40:22   BIC                          53227.375
Sample:                    03-10-2004   HQIC                         53203.787
                         - 04-04-2005
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1.RH       1.3106      0.010    127.314      0.000       1.290       1.331
ar.L2.RH      -0.3088      0.017    -18.149      0.000      -0.342      -0.275
ar.L3.RH       0.0265      0.017      1.559      0.119      -0.007       0.060
ar.L4.RH      -0.0908      0.010     -8.815      0.000      -0.111      -0.071
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            1.2306           -0.1145j            1.2359           -0.0148
AR.2            1.2306           +0.1145j            1.2359            0.0148
AR.3           -1.0844           -2.4570j            2.6857           -0.3162
AR.4           -1.0844           +2.4570j            2.6857            0.3162
-----------------------------------------------------------------------------
```

11.1 Diagnostic Analysis

11.1.1 Chi Square Test

We noticed that four out of the six identified ARMA order from the GPAC table pass the chi squared test.

The residual is white, and the estimated order is n_a= 2 and n_b = 3

The residual is white, and the estimated order is n_a= 2 and n_b = 7

The residual is white, and the estimated order is n_a= 4 and n_b = 0

The residual is white, and the estimated order is n_a= 4 and n_b = 2

[(2, 3), (2, 7), (4, 0), (4, 2)]

11.1.2 Simplification of Model

We will now simplify the ARMA(4,0) model by checking if zeros are included in confidence interval or not. The confidence interval for the parameters are as shown below. We notice there are no zeros in the confidence interval band. Thus, no simplification needed.

	coef	[0.025	0.975]
ar.L1.RH	1.3106	1.290	1.331
ar.L2.RH	-0.3088	-0.342	-0.275
ar.L3.RH	0.0265	-0.007	0.060
ar.L4.RH	-0.0908	-0.111	-0.071

11.1.3 Zero/pole cancellation

We will also simplify model based on zero/pole cancellation by checking the roots of numerator and denominator. The roots of the AR and MA process are shown below. None of the roots are same, thus no zero/pole cancellation required. Hence the final ARMA model after simplification is ARMA(4,0).

	Real	Imaginary	Modulus	Frequency
AR.1	1.2306	-0.1145j	1.2359	-0.0148
AR.2	1.2306	+0.1145j	1.2359	0.0148
AR.3	-1.0844	-2.4570j	2.6857	-0.3162
AR.4	-1.0844	+2.4570j	2.6857	0.3162

11.1.4 Performance measures

The possible reason for poor performance of ARMA model is that our data contains seasonality.

MSE: 1003.063

RMSE: 31.67116985524848

Mean of Residual Errors: 2.390145156585325

Variance of Residual Errors: 997.3505220078016

Biased or Unbiased models: Since the absolute value of mean of the residuals is greater than 0.05, we say that the model is biased. We can remove the bias by adding the mean to all the predictions.

Variance of error of estimated parameters:

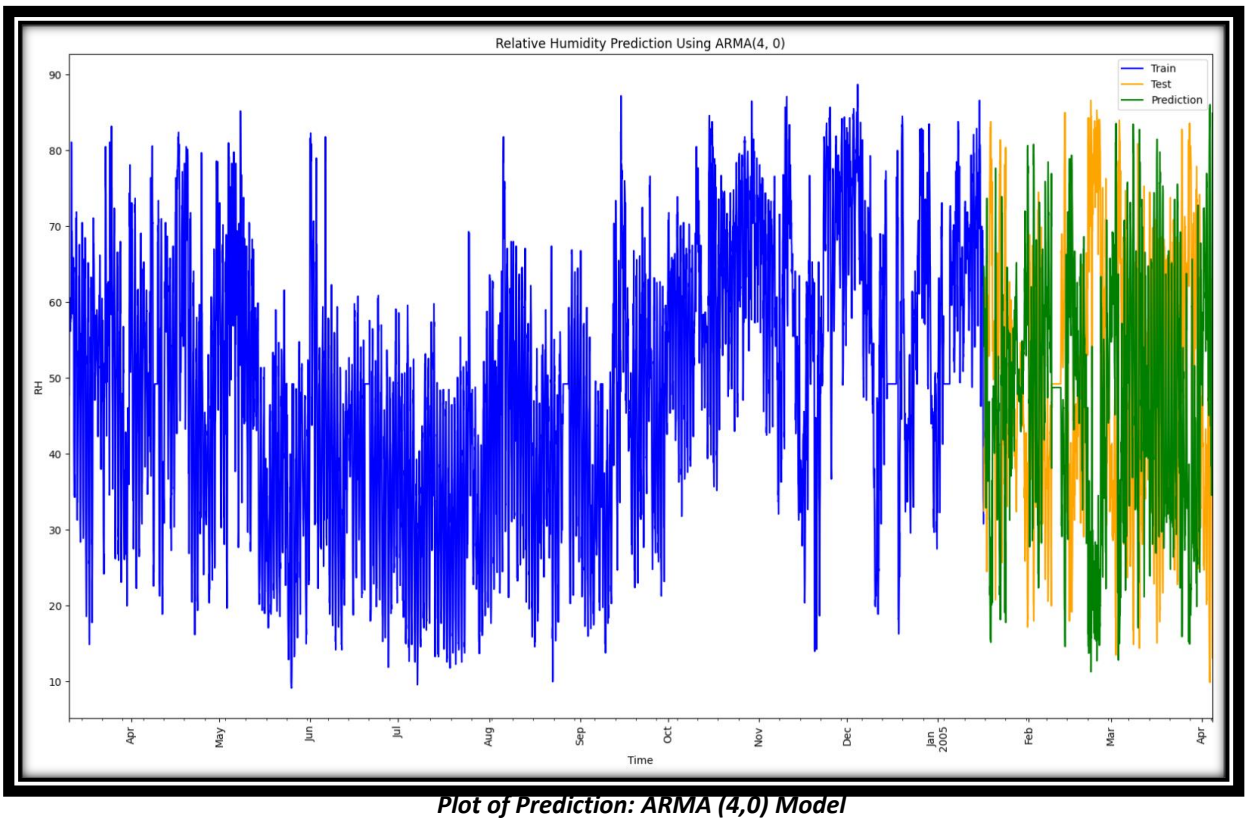
Estimated variance of error for na = 4 and nb = 0 is 17.20878463414576

Covariance of estimated Parameters:

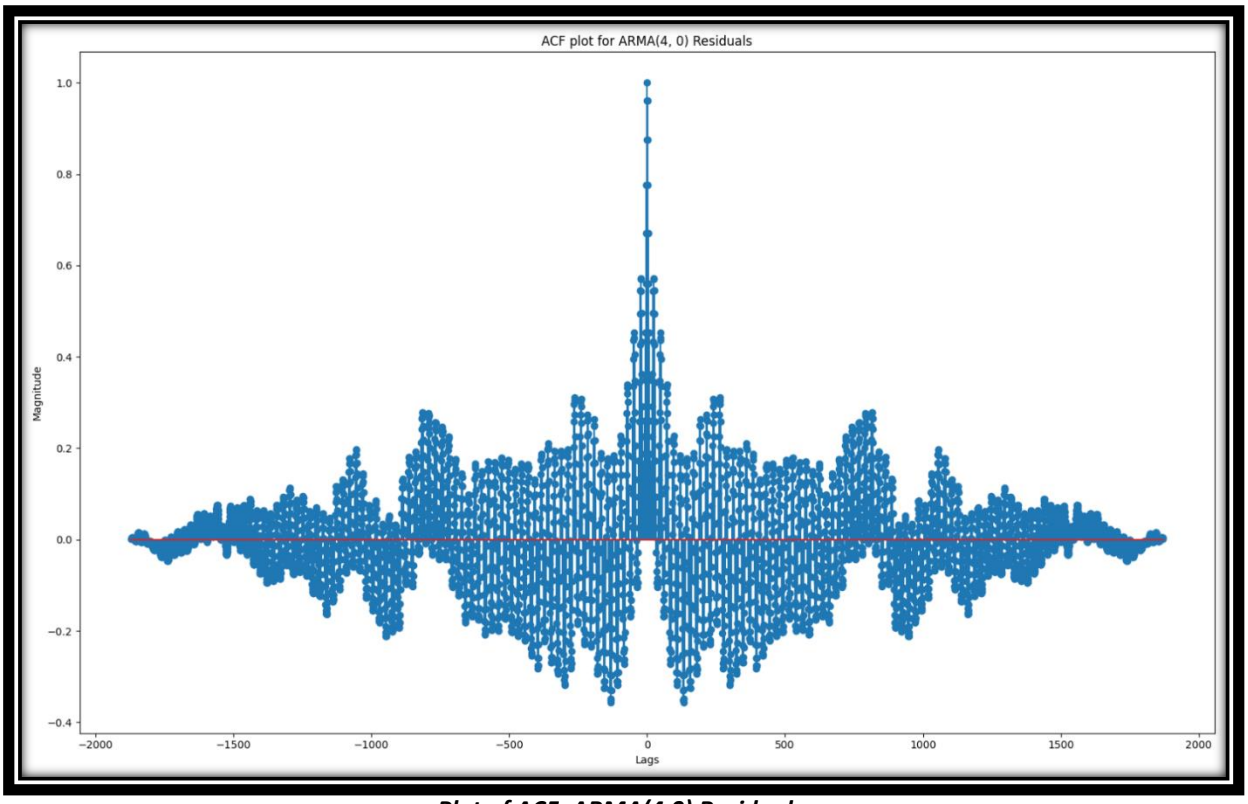
	ar.L1.RH	ar.L2.RH	ma.L1.RH	ma.L2.RH	ma.L3.RH
ar.L1.RH	0.000425	-0.000394	-0.000417	-0.000148	-0.000071
ar.L2.RH	-0.000394	0.000369	0.000388	0.000135	0.000059
ma.L1.RH	-0.000417	0.000388	0.000518	0.000109	0.000055
ma.L2.RH	-0.000148	0.000135	0.000109	0.000169	-0.000011
ma.L3.RH	-0.000071	0.000059	0.000055	-0.000011	0.000141

	ar.L1.RH	ar.L2.RH	ma.L1.RH	ma.L2.RH	ma.L3.RH
ar.L1.RH	0.000425	-0.000394	-0.000417	-0.000148	-0.000071
ar.L2.RH	-0.000394	0.000369	0.000388	0.000135	0.000059
ma.L1.RH	-0.000417	0.000388	0.000518	0.000109	0.000055
ma.L2.RH	-0.000148	0.000135	0.000109	0.000169	-0.000011
ma.L3.RH	-0.000071	0.000059	0.000055	-0.000011	0.000141

The plot of forecasted values with the actual value and ACF is shown below



Plot of Prediction: ARMA (4,0) Model



Plot of ACF: ARMA(4,0) Residuals

....

XII. ARMA MODEL (4)

Summary of ARMA (4,0) model is as follows. We can infer from the RMSE value that ARMA(2,7) method isn't suited for datasets with high variability, though the graph prediction matches and the best so far.

We can still improve our score by adopting different orders. Now we will look at another order and try to improve our score.

```
                              ARMA Model Results
==============================================================================
Dep. Variable:                     RH   No. Observations:                 9357
Model:                     ARMA(4, 0)   Log Likelihood              -26590.828
Method:                       css-mle   S.D. of innovations              4.148
Date:                Tue, 15 Dec 2020   AIC                          53191.656
Time:                        02:40:22   BIC                          53227.375
Sample:                    03-10-2004   HQIC                         53203.787
                         - 04-04-2005
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1.RH       1.3106      0.010    127.314      0.000       1.290       1.331
ar.L2.RH      -0.3088      0.017    -18.149      0.000      -0.342      -0.275
ar.L3.RH       0.0265      0.017      1.559      0.119      -0.007       0.060
ar.L4.RH      -0.0908      0.010     -8.815      0.000      -0.111      -0.071
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            1.2306           -0.1145j            1.2359           -0.0148
AR.2            1.2306           +0.1145j            1.2359            0.0148
AR.3           -1.0844           -2.4570j            2.6857           -0.3162
AR.4           -1.0844           +2.4570j            2.6857            0.3162
-----------------------------------------------------------------------------
```

12.1 Diagnostic Analysis

12.1.1 Chi Square Test

We noticed that four out of the six identified ARMA order from the GPAC table pass the chi squared test.

The residual is white, and the estimated order is n_a= 2 and n_b = 3

The residual is white, and the estimated order is n_a= 2 and n_b = 7

The residual is white, and the estimated order is n_a= 4 and n_b = 0

The residual is white, and the estimated order is n_a= 4 and n_b = 2

[(2, 3), (2, 7), (4, 0), (4, 2)]

12.1.2 Simplification of Model

We will now simplify the ARMA(4,0) model by checking if zeros are included in confidence interval or not. The confidence interval for the parameters are as shown below. We notice there are no zeros in the confidence interval band. Thus, no simplification needed.

	coef	[0.025	0.975]
ar.L1.RH	3.1017	nan	nan
ar.L2.RH	-3.4701	nan	nan
ar.L3.RH	1.5761	1.575	1.577
ar.L4.RH	-0.2103	nan	nan
ma.L1.RH	-1.9189	nan	nan
ma.L2.RH	0.9841	nan	nan

12.1.3 Zero/pole cancellation

We will also simplify model based on zero/pole cancellation by checking the roots of numerator and denominator. The roots of the AR and MA process are shown below. None of the roots are same, thus no zero/pole cancellation required. Hence the final ARMA model after simplification is ARMA(4,2).

	Real	Imaginary	Modulus	Frequency
AR.1	0.9669	-0.2590j	1.0010	-0.0417
AR.2	0.9669	+0.2590j	1.0010	0.0417
AR.3	1.0528	-0.0000j	1.0528	-0.0000
AR.4	4.5081	-0.0000j	4.5081	-0.0000
MA.1	0.9750	-0.2561j	1.0081	-0.0409
MA.2	0.9750	+0.2561j	1.0081	0.0409

12.1.4 Performance measures

The possible reason for poor performance of ARMA model is that our data contains seasonality.

MSE: 1004.539

RMSE: 31.69446323886871

Mean of Residual Errors: 2.4177438603919392

Variance of Residual Errors: 998.6936268531492

Biased or Unbiased models: Since the absolute value of mean of the residuals is greater than 0.05, we say that the model is biased. We can remove the bias by adding the mean to all the predictions.

Variance of error of estimated parameters:

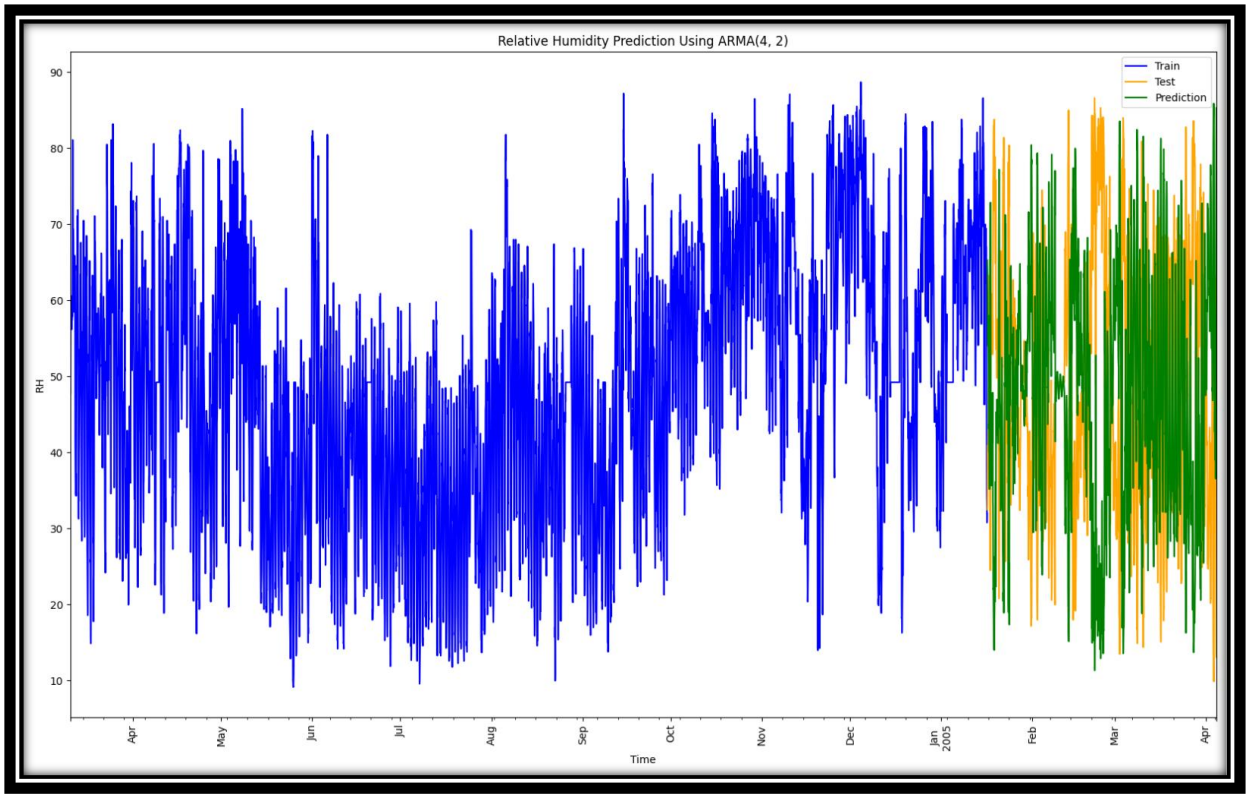
Estimated variance of error for na = 4 and nb = 2 is 15.331040468405101

Covariance of estimated Parameters:

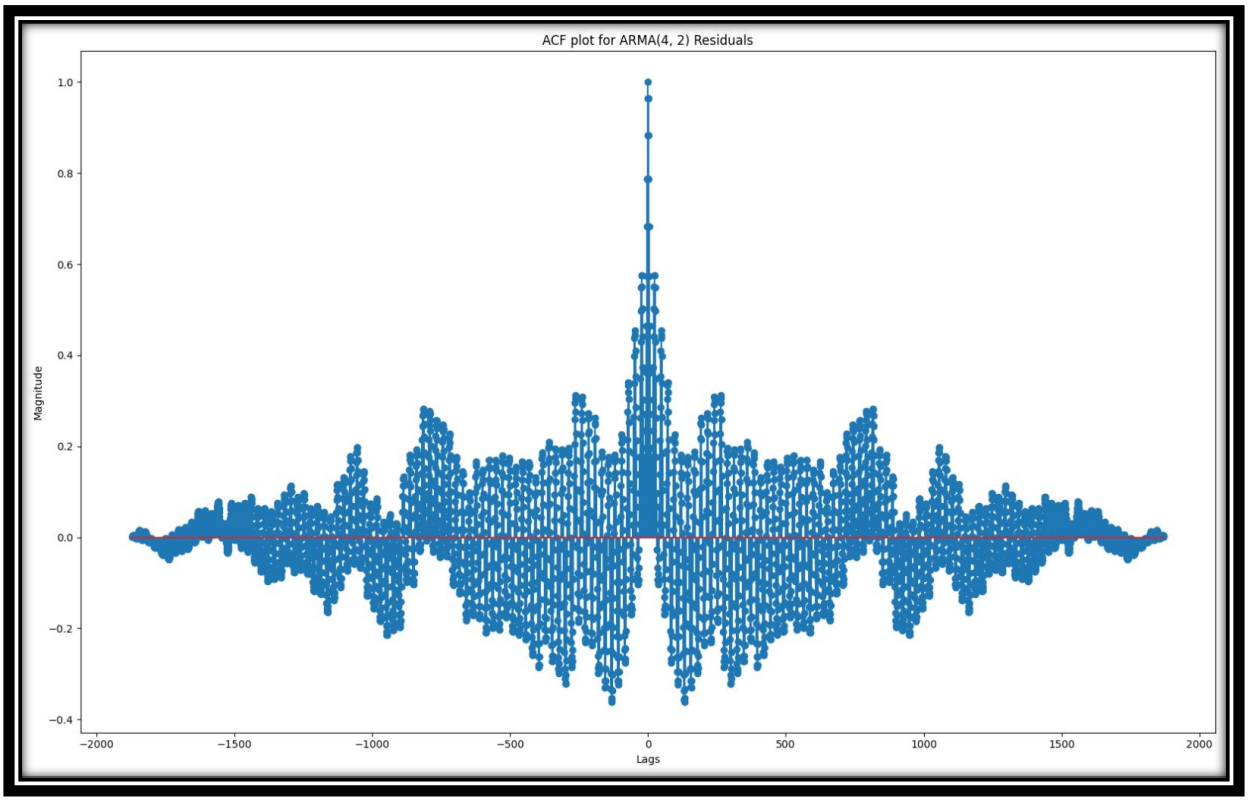
```
              ar.L1.RH      ar.L2.RH  ...      ma.L1.RH      ma.L2.RH
ar.L1.RH -1.411033e-06  1.646584e-06  ...  2.852952e-07 -2.382432e-07
ar.L2.RH  1.646584e-06 -1.782098e-06  ... -4.537540e-07  4.232904e-07
ar.L3.RH  6.685994e-07 -1.051925e-06  ... -2.123162e-07  2.116603e-07
ar.L4.RH -1.069152e-06  1.387772e-06  ...  2.902611e-07 -2.666720e-07
ma.L1.RH  2.852952e-07 -4.537540e-07  ... -5.727314e-06  8.150445e-06
ma.L2.RH -2.382432e-07  4.232904e-07  ...  8.150445e-06 -1.095549e-05

[6 rows x 6 columns]
```


```
              ar.L1.RH      ar.L2.RH  ...      ma.L1.RH      ma.L2.RH
ar.L1.RH -1.411033e-06  1.646584e-06  ...  2.852952e-07 -2.382432e-07
ar.L2.RH  1.646584e-06 -1.782098e-06  ... -4.537540e-07  4.232904e-07
ar.L3.RH  6.685994e-07 -1.051925e-06  ... -2.123162e-07  2.116603e-07
ar.L4.RH -1.069152e-06  1.387772e-06  ...  2.902611e-07 -2.666720e-07
ma.L1.RH  2.852952e-07 -4.537540e-07  ... -5.727314e-06  8.150445e-06
ma.L2.RH -2.382432e-07  4.232904e-07  ...  8.150445e-06 -1.095549e-05
```

The plot of forecasted values with the actual value and ACF is shown below



Plot of Prediction: ARMA (4,2) Model



Plot of ACF: ARMA(4,2) Residuals

XIII. BASE-MODELS

13.1 Average Model

We compute the mean of training data and perform h step predictions to match the size of the test data.

RMSE: **16.18168718026646**

MSE: **261.847**

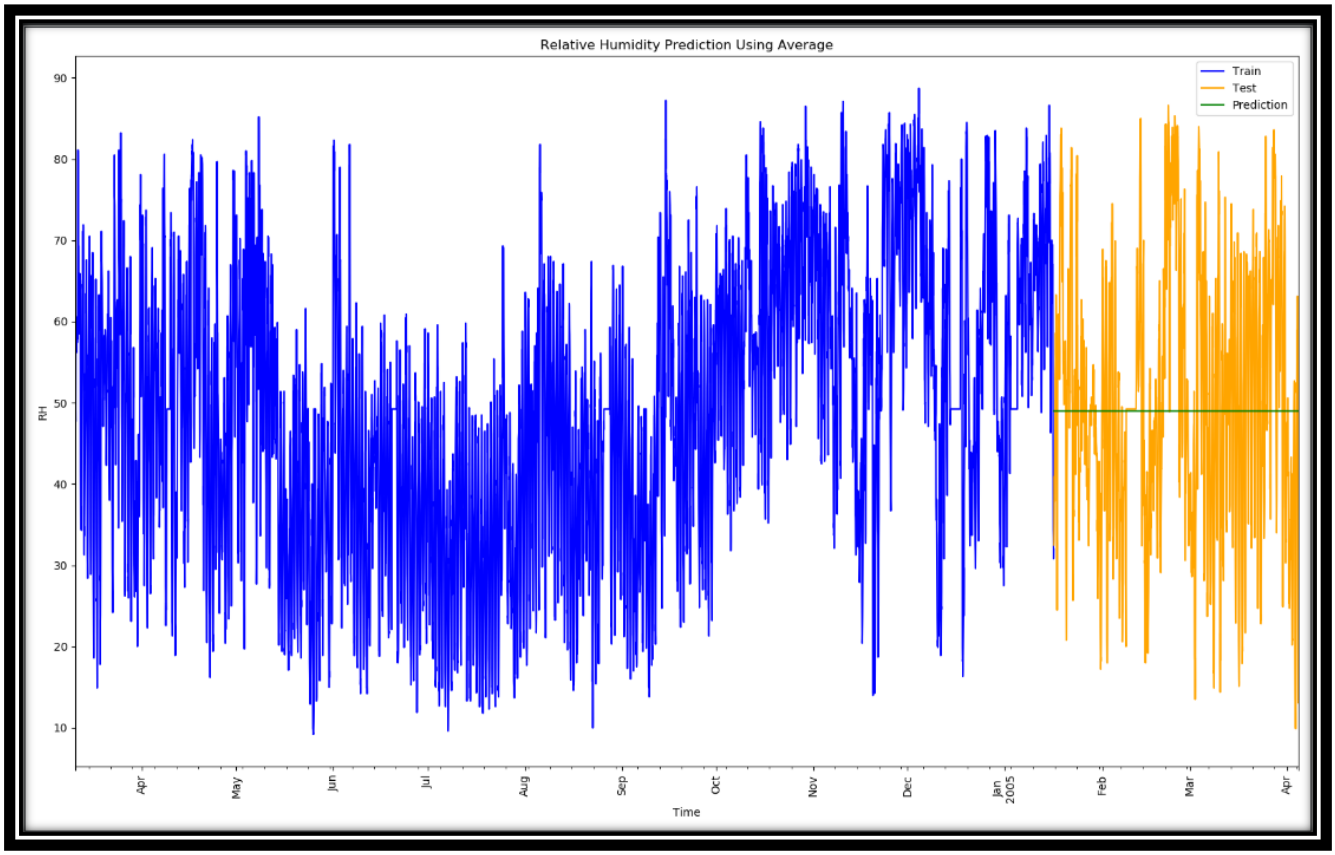
Residual Variance: **260.3322099679025**

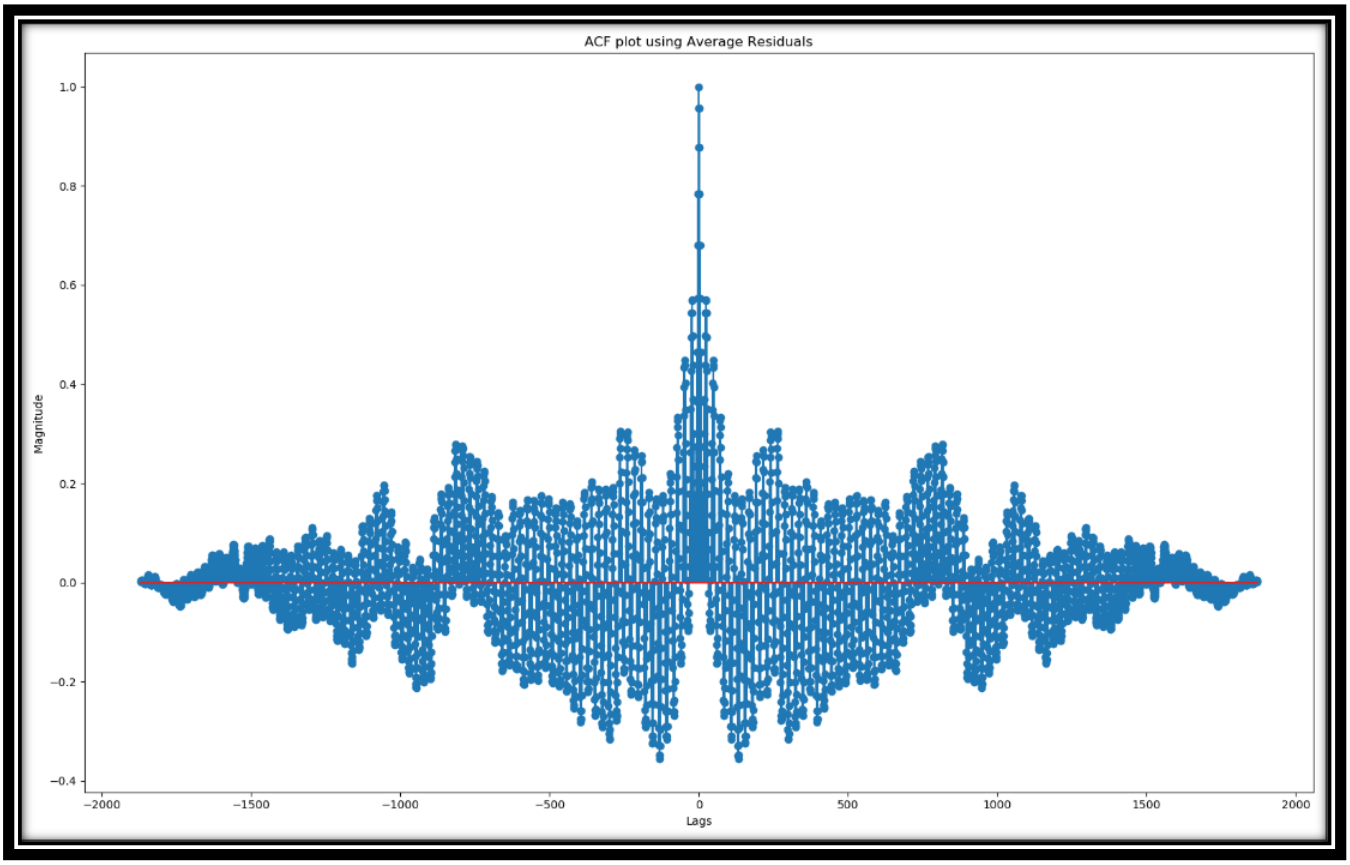
Residual Mean: **1.2309323048119354**

Plot of Prediction: The plot of forecasted values with the actual value is shown below:

Plot of ACF: The ACF of residuals is as shown in below plot and from the plot it can observed that the ACF plot does not resemble white noise.

We can infer from the RMSE value that average method isn't suited for datasets with high variability, though the graph prediction matches and the best so far. We can still improve our score by adopting different orders. Now we will look at another order and try to improve our score.





Plot of Prediction: Average

Plot of ACF: Average Residuals

13.2 Naïve Model

We compute the mean of training data and perform h step predictions to match the size of the test data.

RMSE: **24.33643359245557**

MSE: **592.262**

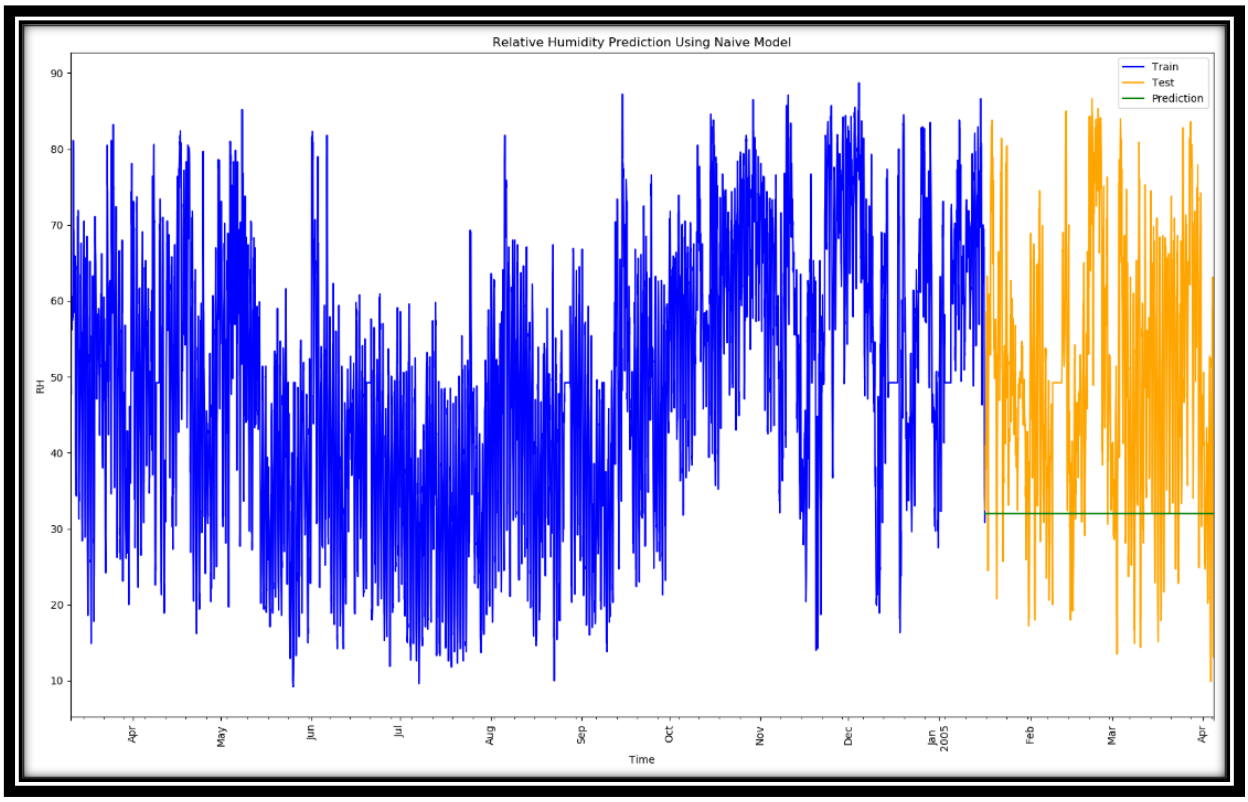
Residual Variance: **260.33220996790254**

Residual Mean: **18.218932304811933**

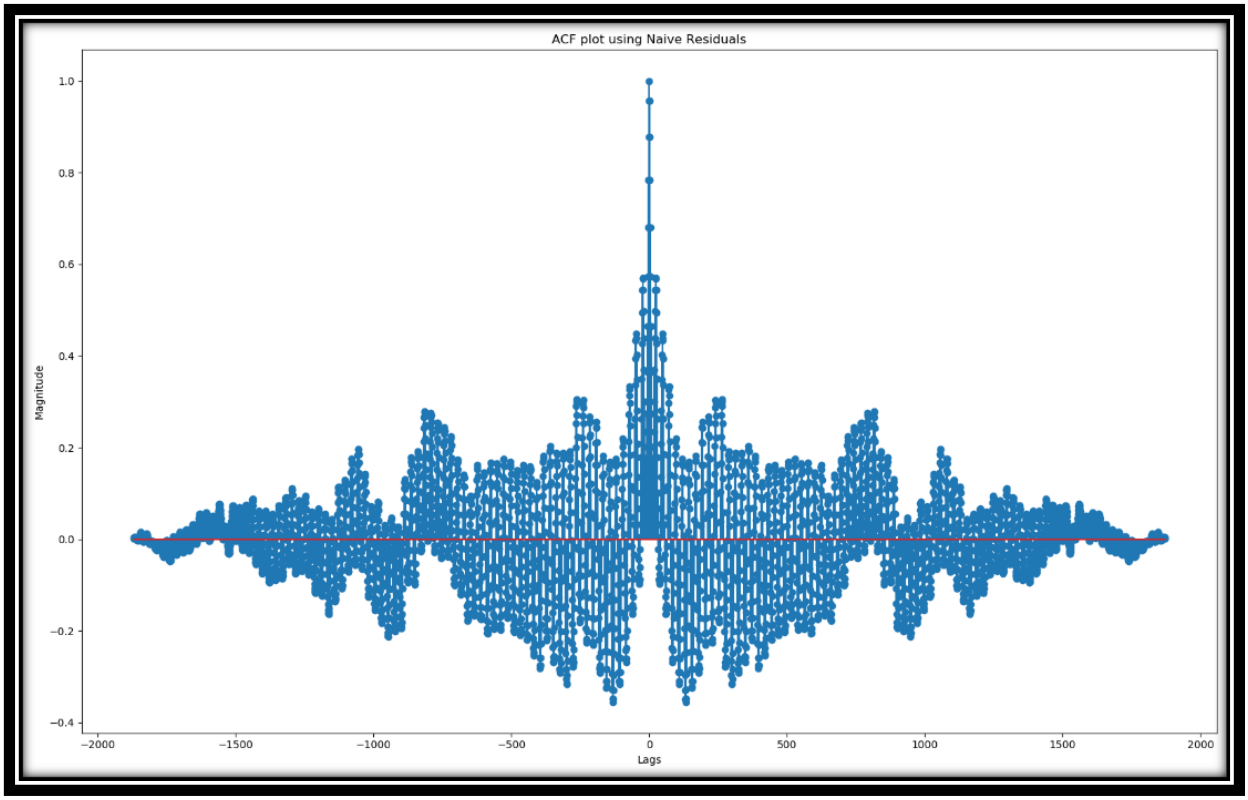
Plot of Prediction: The plot of forecasted values with the actual value is shown below:

Plot of ACF: The ACF of residuals is as follows:

We conclude that the ACF plot does not resemble white noise. We can see that this model didn't improve our score. Hence we can infer from the score that this method works best when the average at each time period remains constant. Though the score of Naive method is better than Average method, but this does not mean that the Naive method is better than Average method on all datasets. We should move step by step to each model and confirm whether it improves our model or not.



Plot of Prediction: Naive



Plot of ACF: Naive Residuals

13.3 Drift Model

We compute the mean of training data and perform h step predictions to match the size of the test data.

RMSE: **25.99463406166742**

MSE: **675.721**

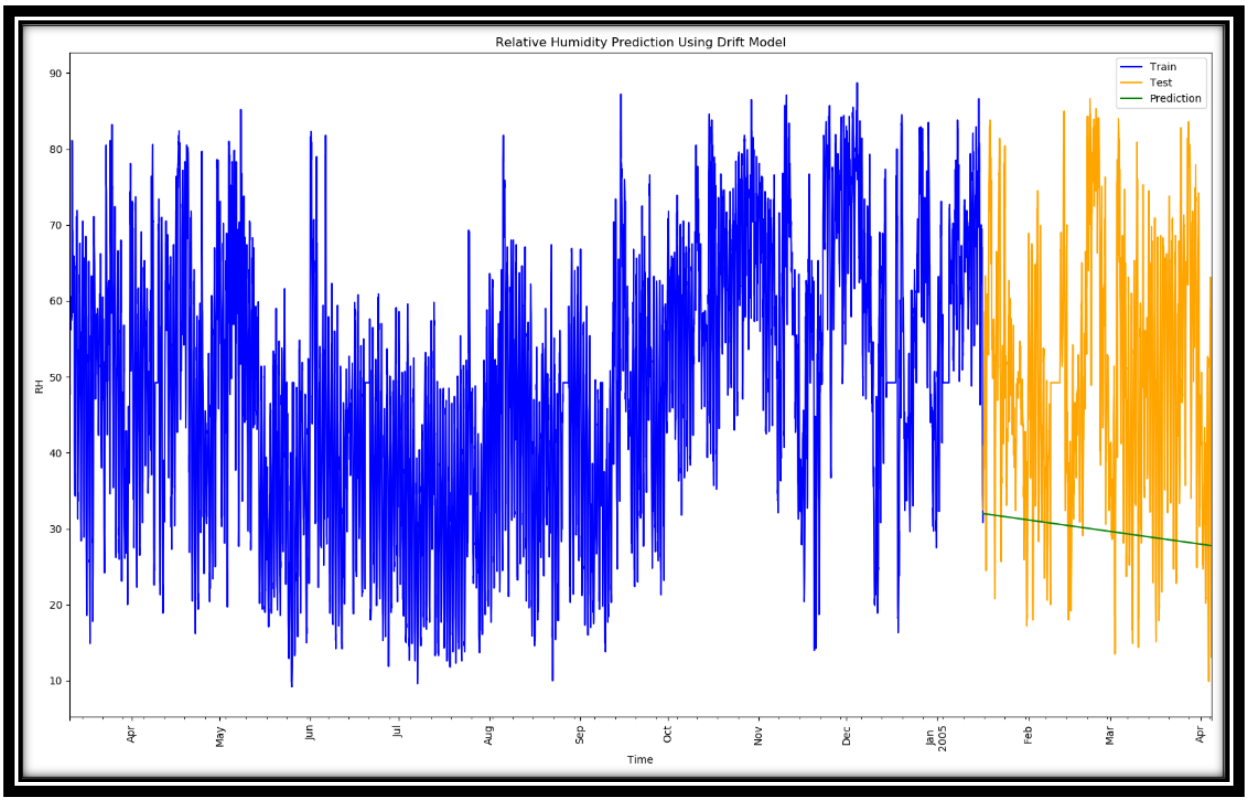
Residual Variance: **262.26196979202473**

Residual Mean: **20.333690317632445**

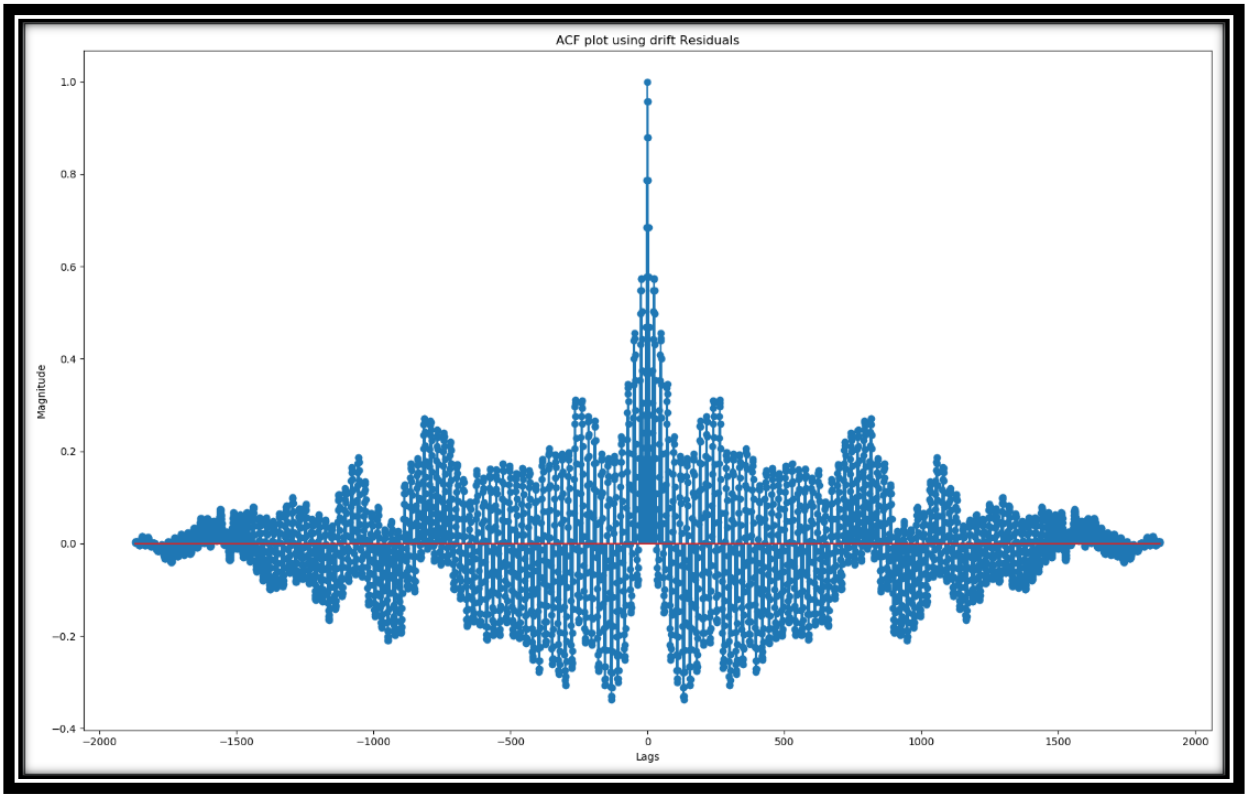
Plot of Prediction: The plot of forecasted values with the actual value is shown below:

Plot of ACF: The ACF of residuals is as follows:

We conclude that the ACF plot does not resemble white noise. We can see that this model didn't improve our score. Hence we can infer from the score that this method works best when the average at each time period remains constant. Though the score of naive method is better than drift method, but this does not mean that the average method is better than naive method on all datasets. We should move step by step to each model and confirm whether it improves our model or not.



Plot of Prediction: Drift



Plot of ACF: Drift Residuals

13.4 SIMPLE AND EXPONENTIAL SMOOTHING

We compute the mean of training data and perform h step predictions to match the size of the test data.

RMSE: **24.13006423530613**

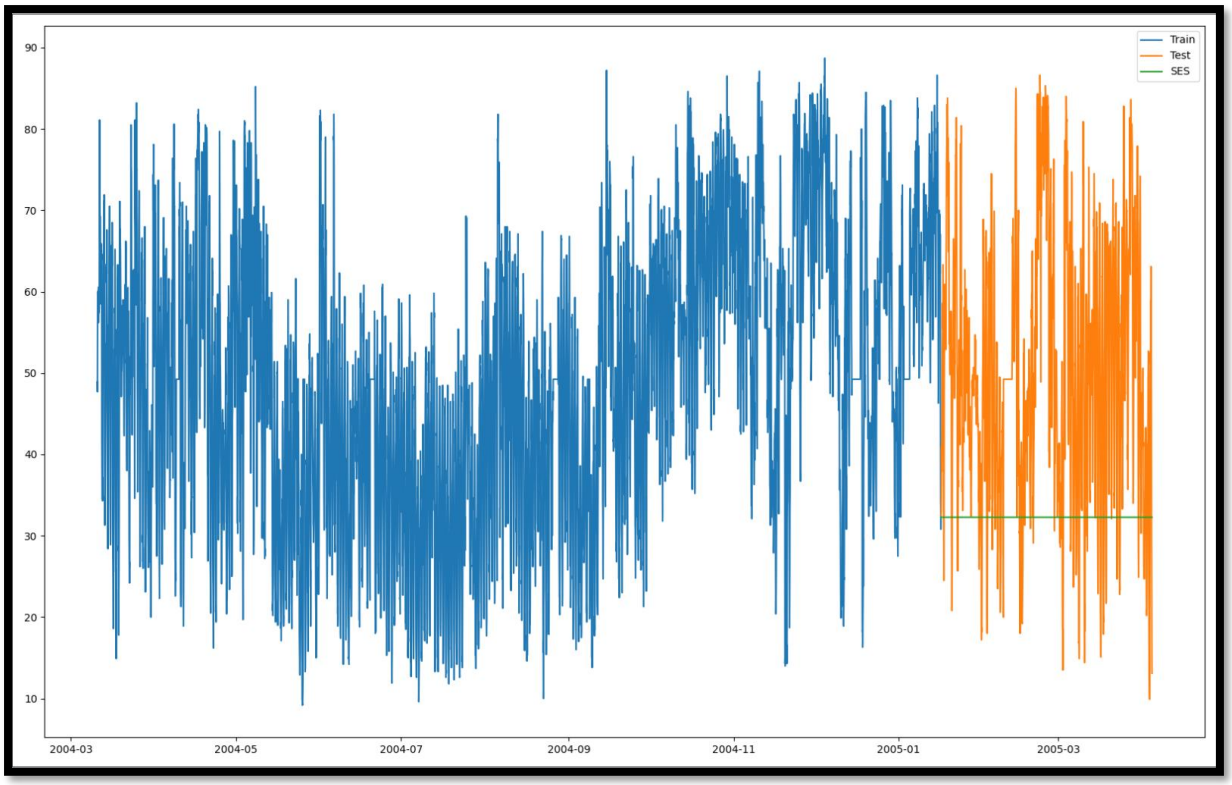
MSE: **582.26**

Residual Variance: **260.33220996790254**

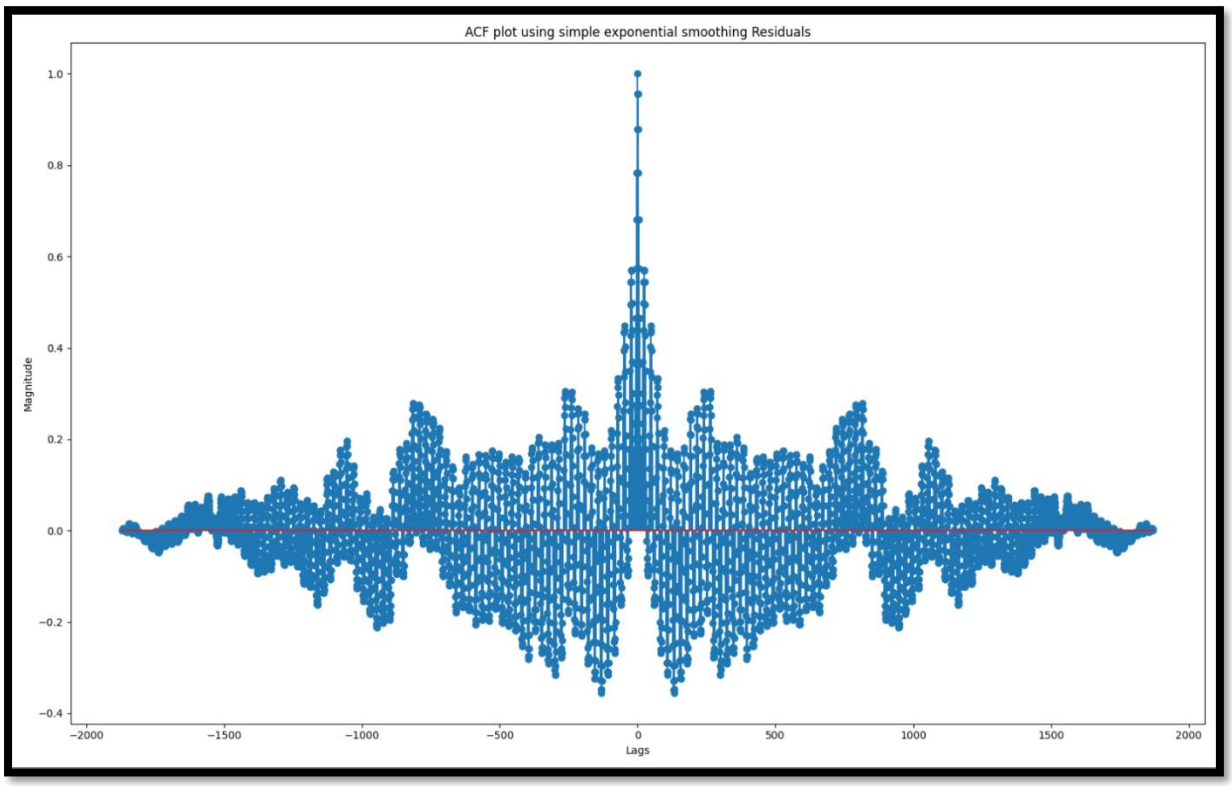
Residual Mean: **17.942352454161597**

Plot of Prediction: The plot of forecasted values with the actual value is shown below:

Plot of ACF: The ACF of residuals is as follows. It can be observed that the ACF plot does not resemble white noise. We can see that implementing Simple exponential model with alpha as 0.6 generates a better model till now. We can tune the parameter using the validation set to generate even a better Simple exponential model.



Plot of Prediction: Simple Exponential smoothing



Plot of ACF: SES Residuals

XIV. FINAL MODEL SELECTION

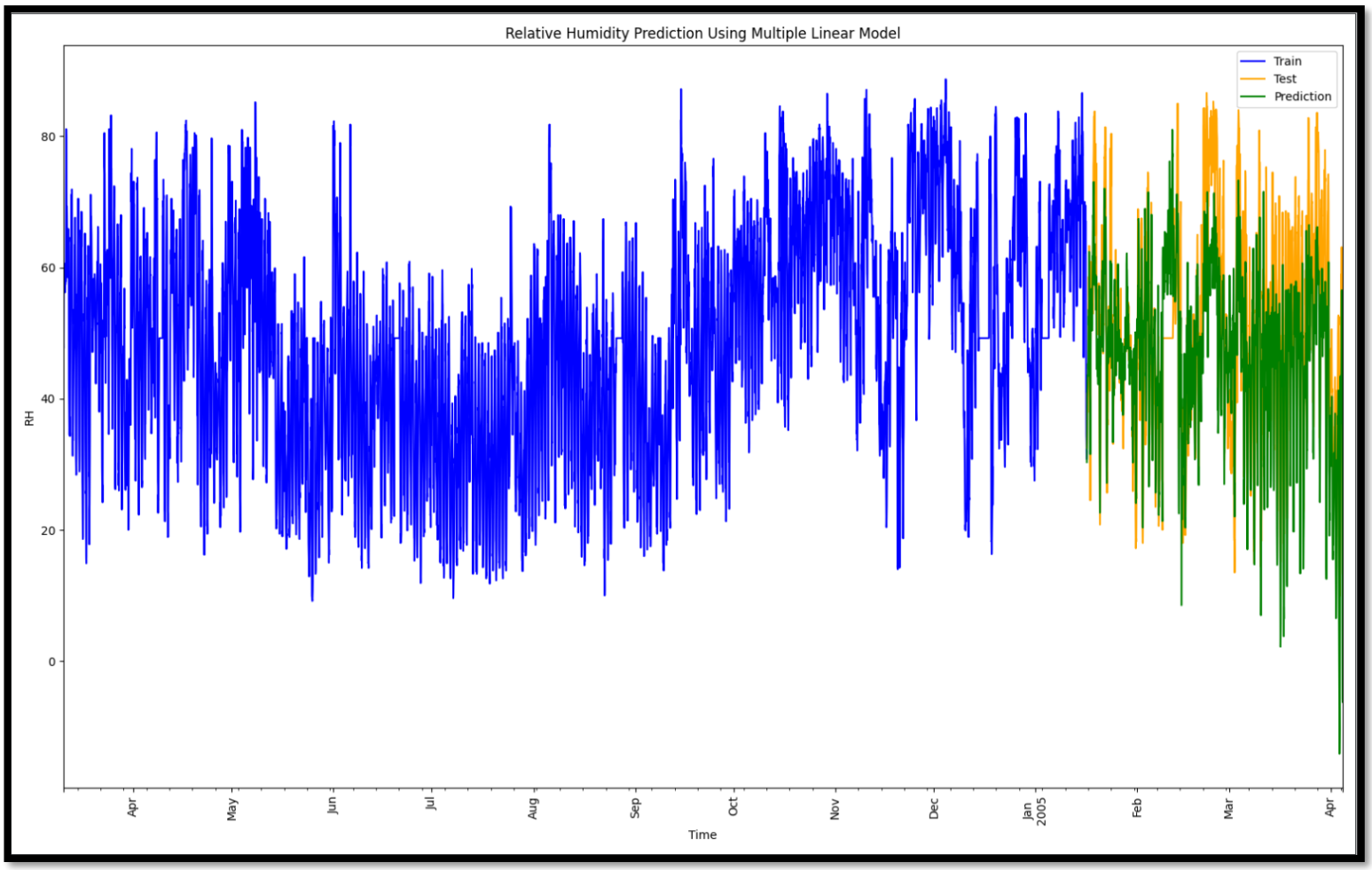
In our analysis for predicting relative humidity we have used the models such as Average Method, Naïve Method, Drift Method, Holt Winter Model, Multiple Linear Regression model and the ARMA model. The mean-squared error is determined by the residual sum of squares resulting from comparing the predictions y hat with the observed outcomes y. Since the MSE is based on squared residuals, it is on the scale of the squared outcomes. Thus, the root of the MSE, which is on the scale of the outcome, is often used to report model fit. We can compare these models on the basis of their RMSE scores and provide conclusions.

The performance metrics for all the models is shown:

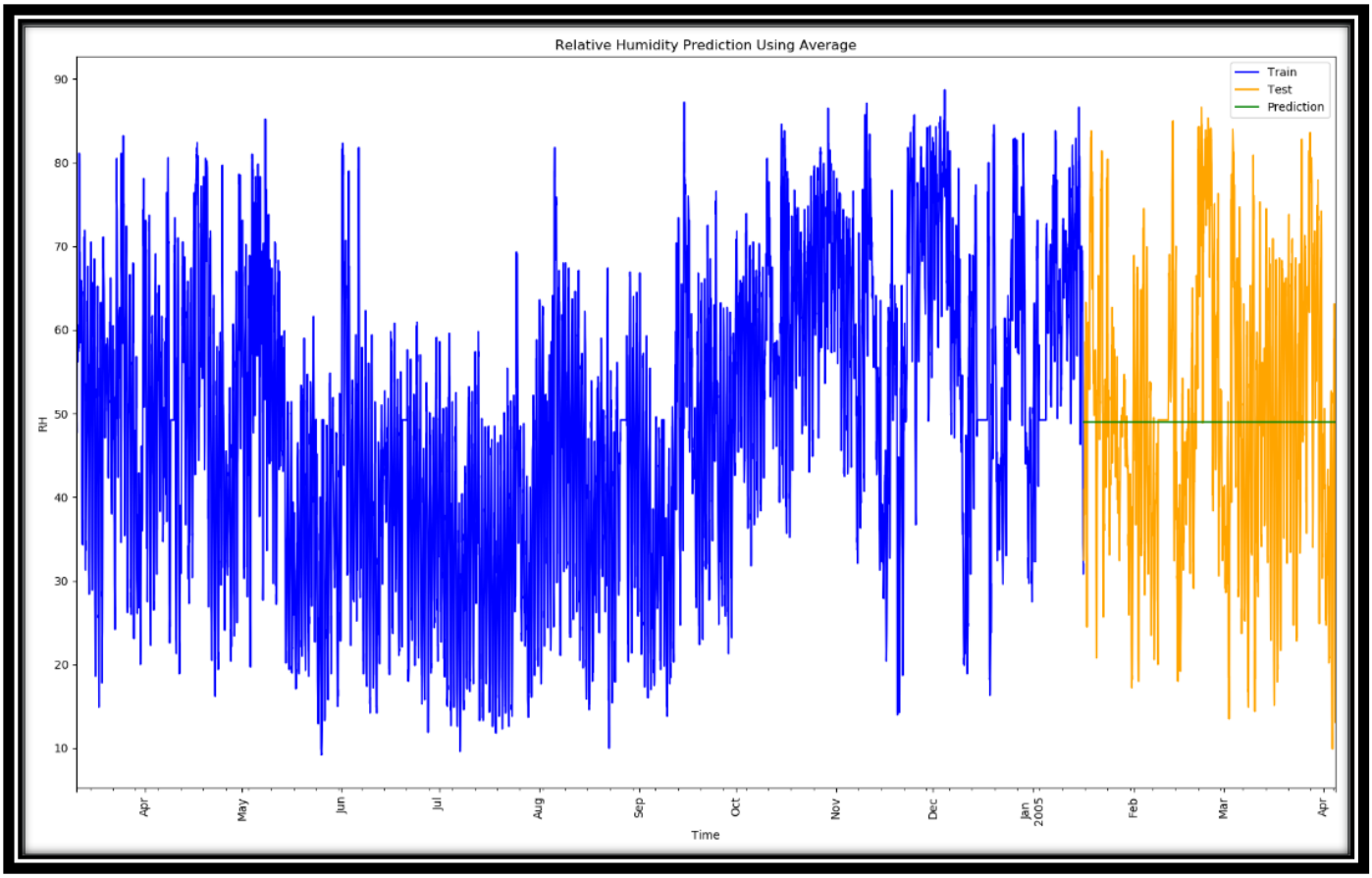
	Model	MSE	RMSE	Residual Mean	Residual Variance
0	Multiple Linear Regression Model	98.319	9.915594	3.782570	84.010898
1	Average Model	261.847	16.181687	1.230932	260.332210
2	Holt Winter Model	298.017	17.263169	9.938233	199.249009
3	Simple Exponential Smoothing Model	582.260	24.130064	17.942352	260.332210
4	Naive Model	592.262	24.336434	18.218932	260.332210
5	Drift Model	675.721	25.994634	20.333690	262.261970
6	ARMA(2, 7) Model	993.428	31.518693	2.496568	987.194696
7	ARMA(4, 0) Model	1003.063	31.671170	2.390145	997.350522
8	ARMA(2, 3) Model	1003.625	31.680041	2.386413	997.930428
9	ARMA(4, 2) Model	1004.539	31.694463	2.417744	998.693627

XV. SUMMARY AND CONCLUSION

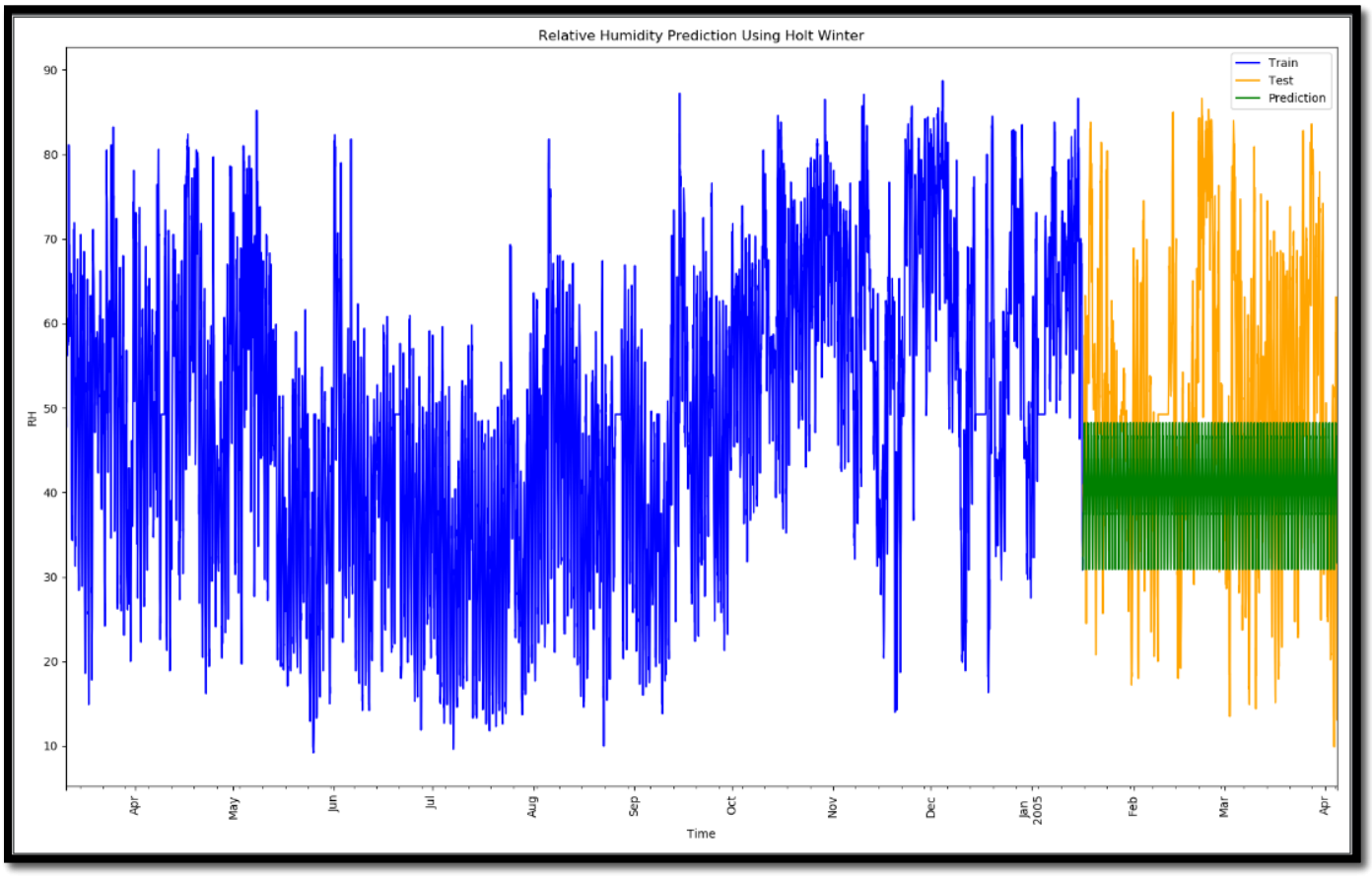
When we consider RMSE to be performance metric, we conclude that multiple linear regression model has the smallest RMSE and thus performs the best. Hence for relative humidity prediction the multiple linear regression model is the best model since the it has the lowest RMSE value when compared to all other models. The plots for predicted and actual values for all models are shown as in the order of final model selection screenshot.



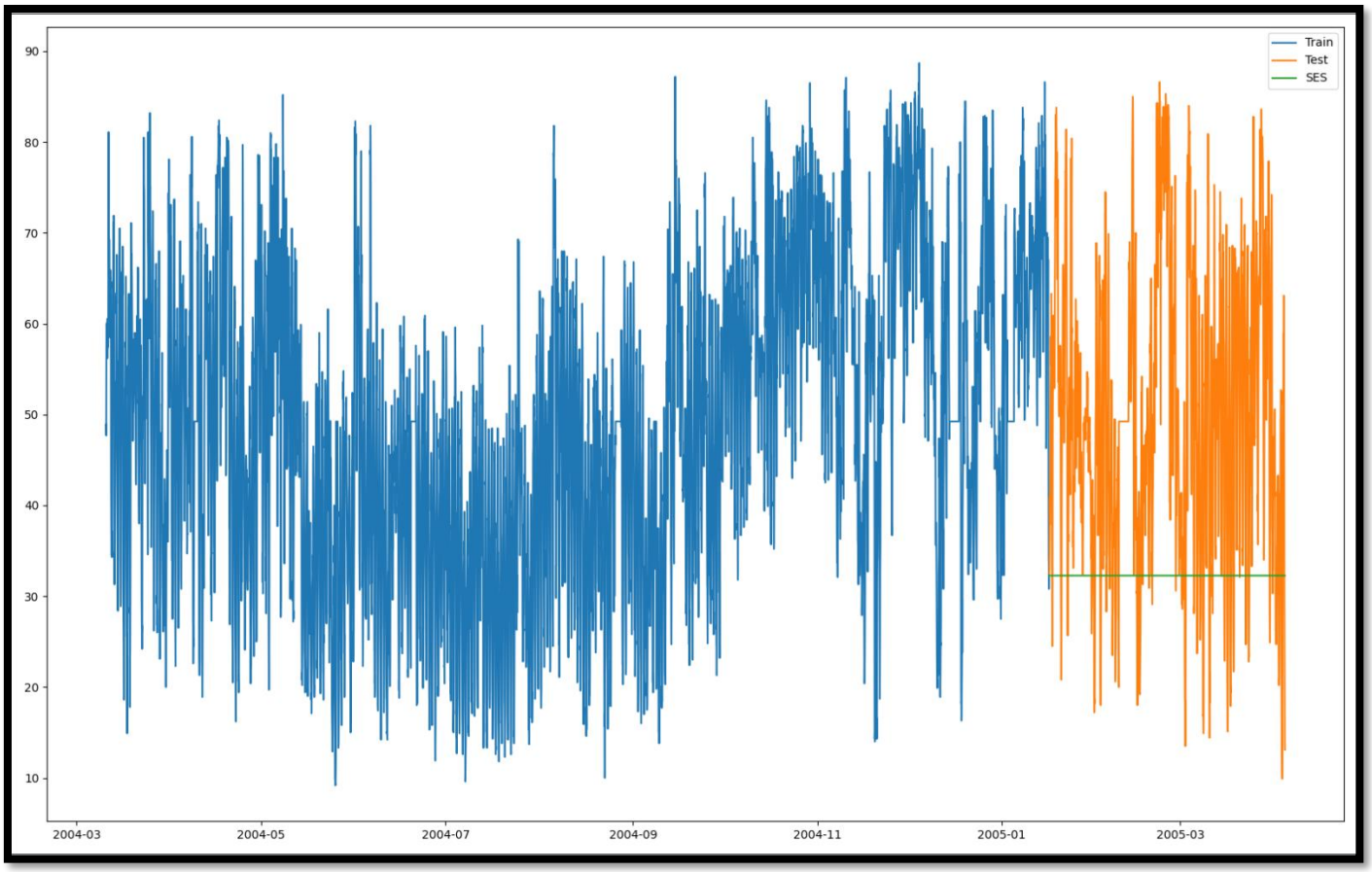
Plot of Prediction v/s actual : Multiple Linear Regression Model



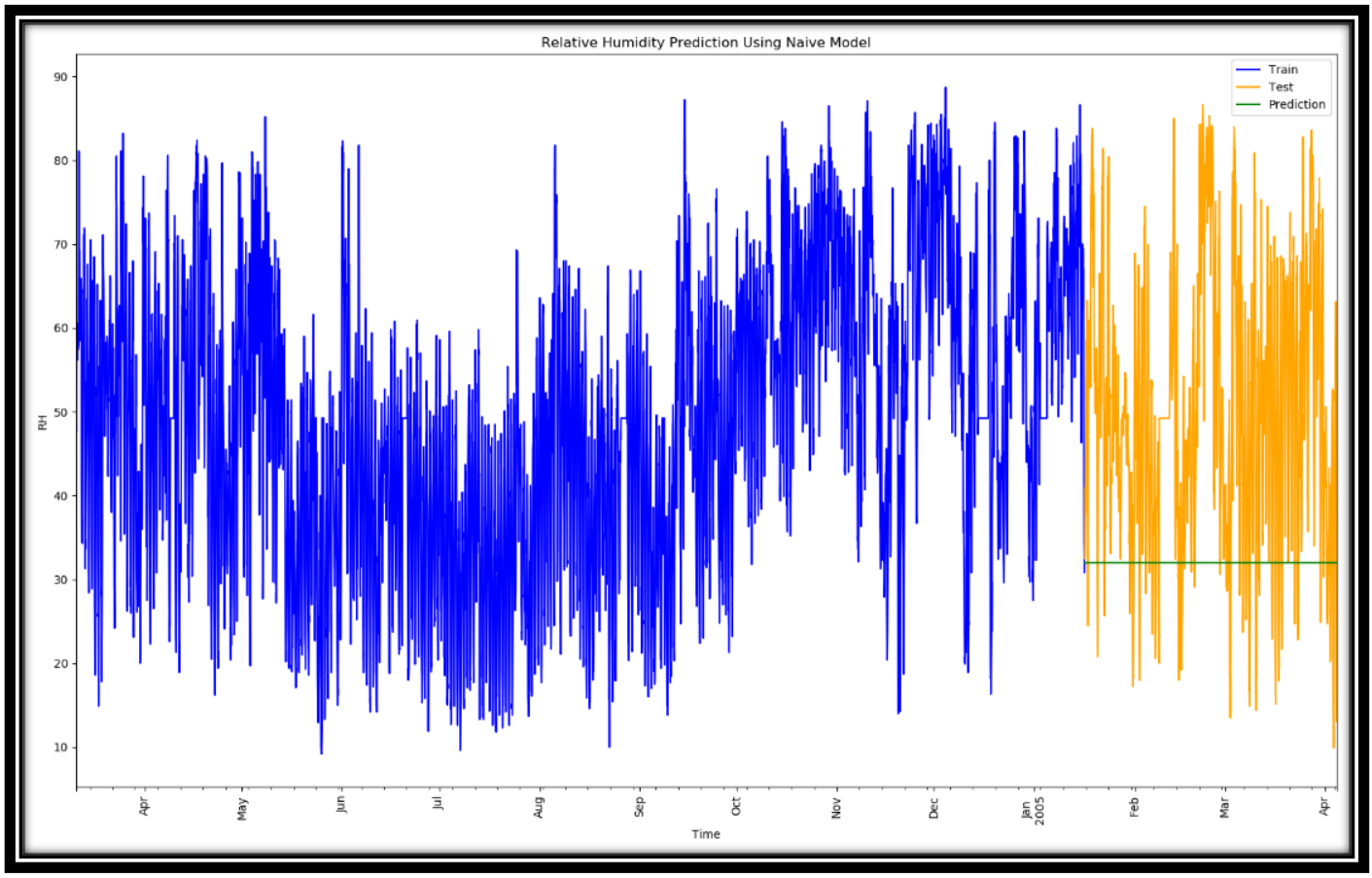
Plot of Prediction v/s actual : Average Model



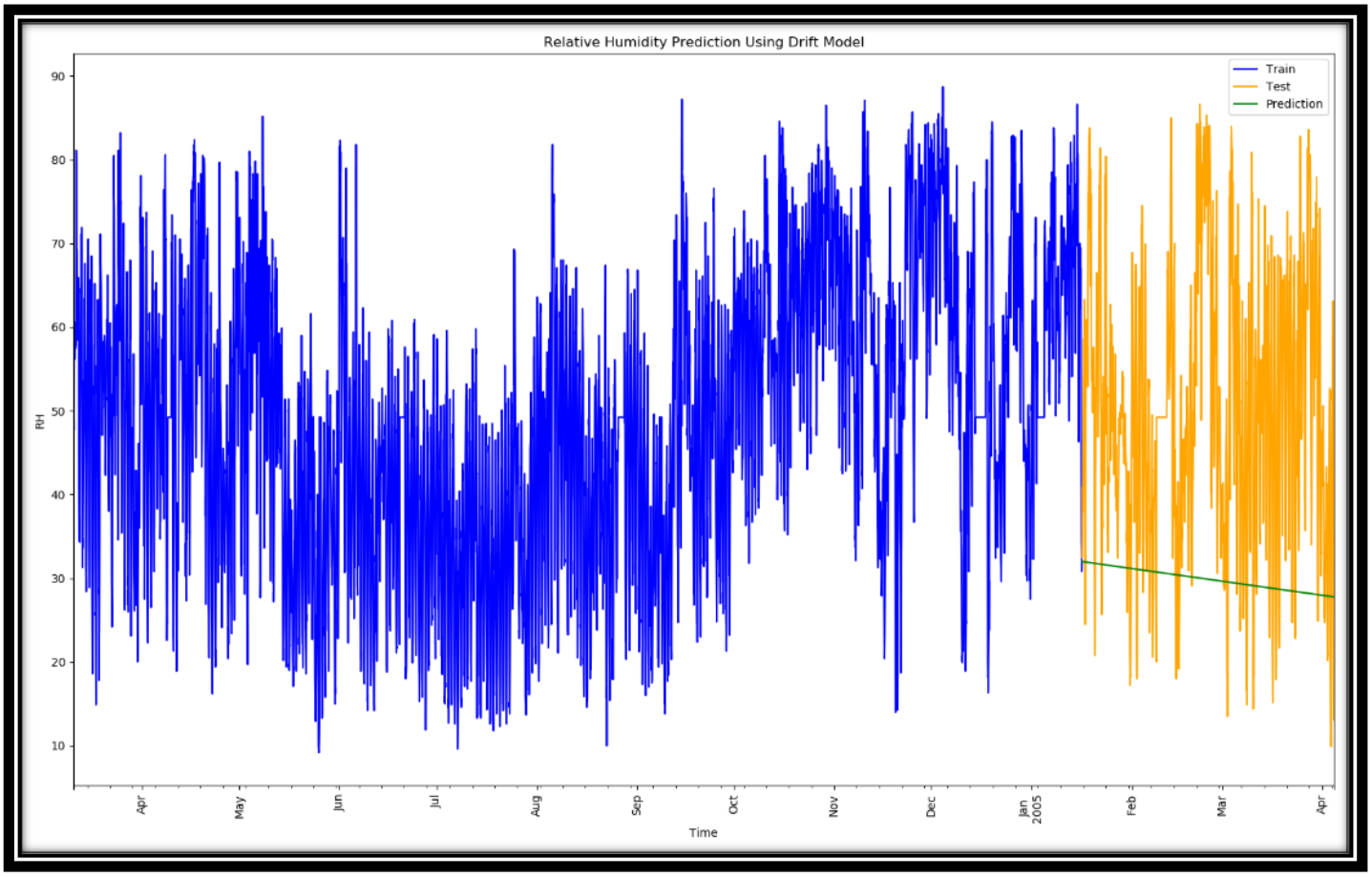
Plot of Prediction v/s actual : Holt Winter Model



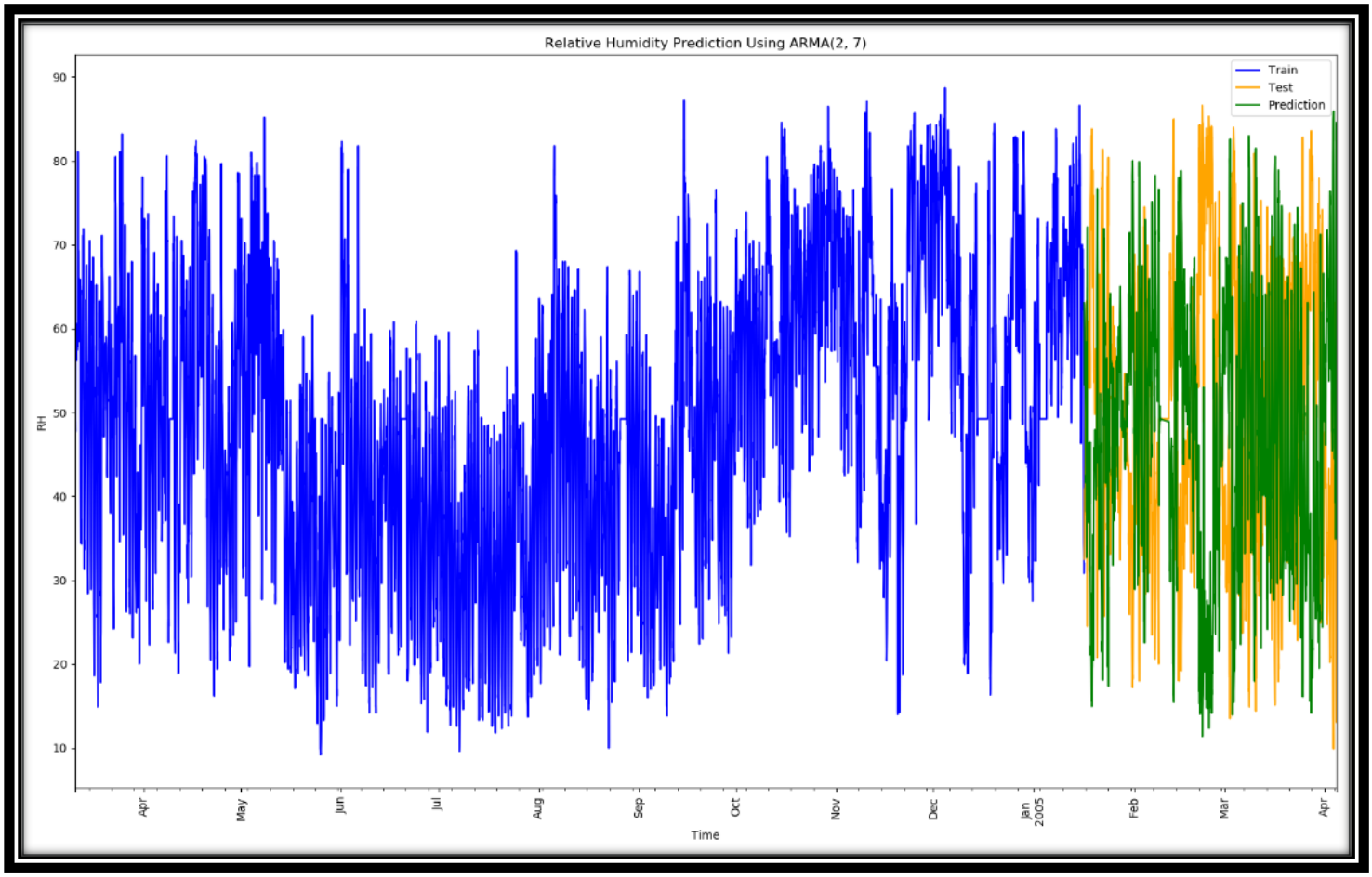
Plot of Prediction v/s actual : SES Model



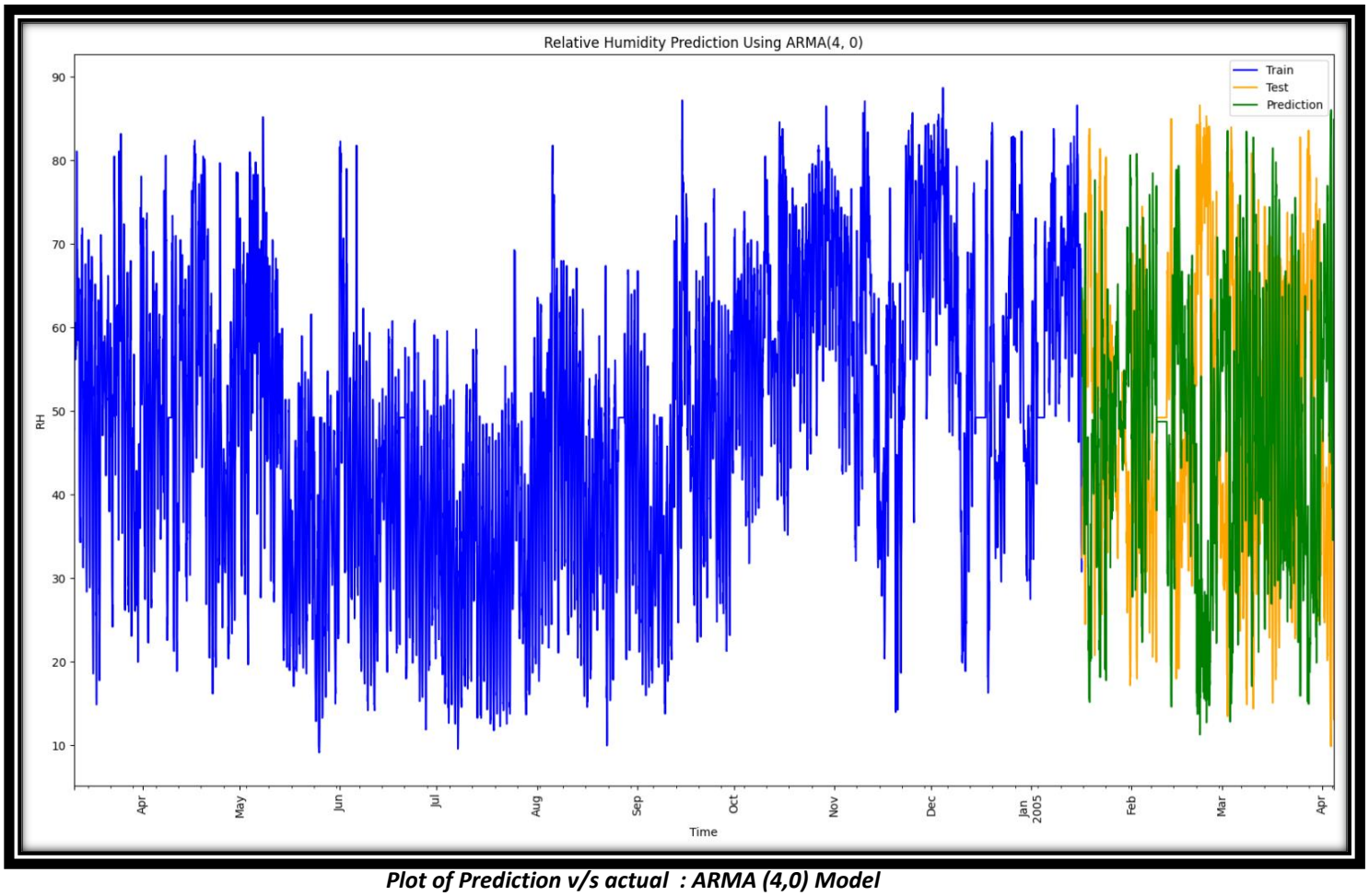
Plot of Prediction v/s actual : Naive Model



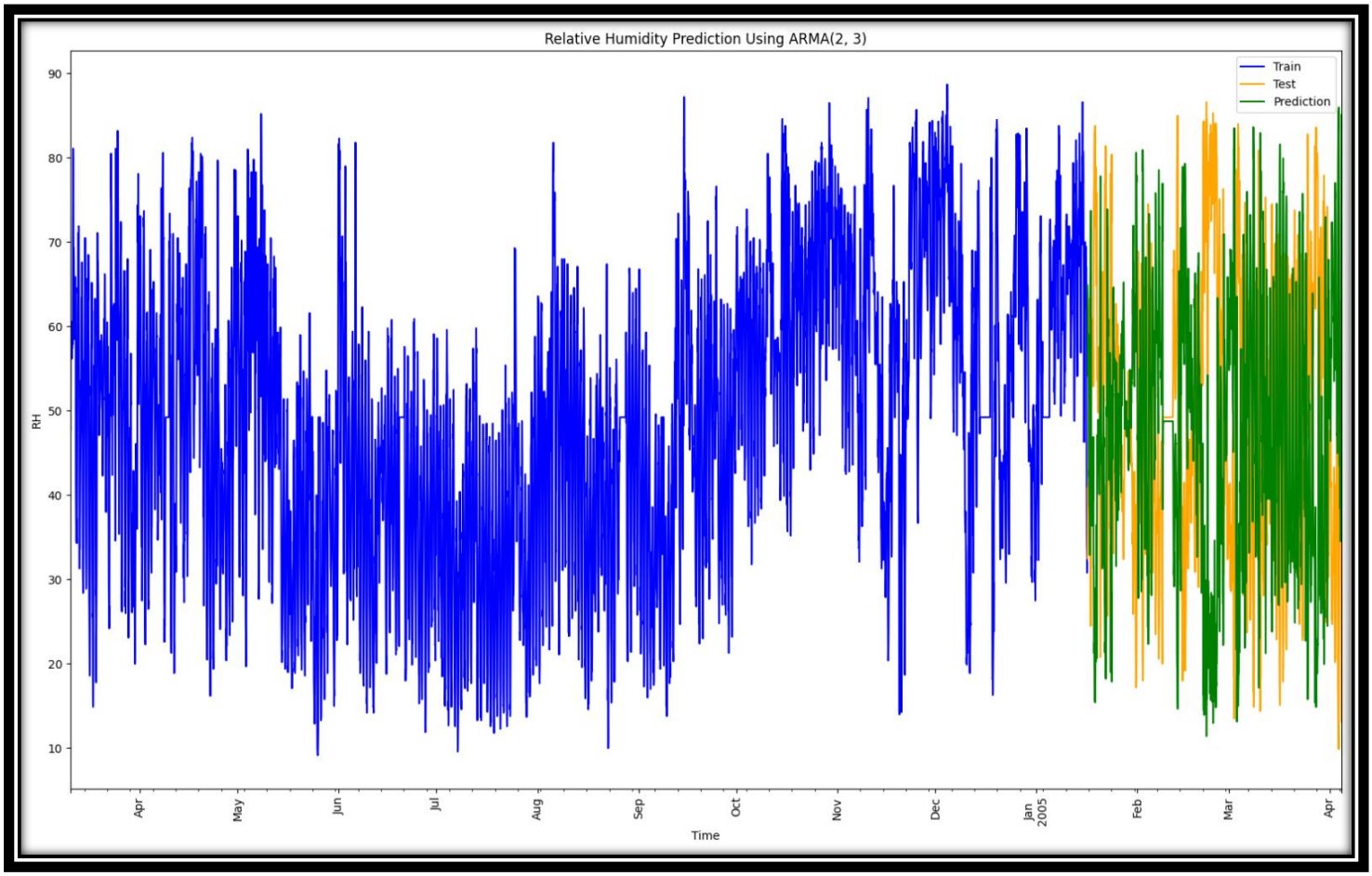
Plot of Prediction v/s actual : Drift Model



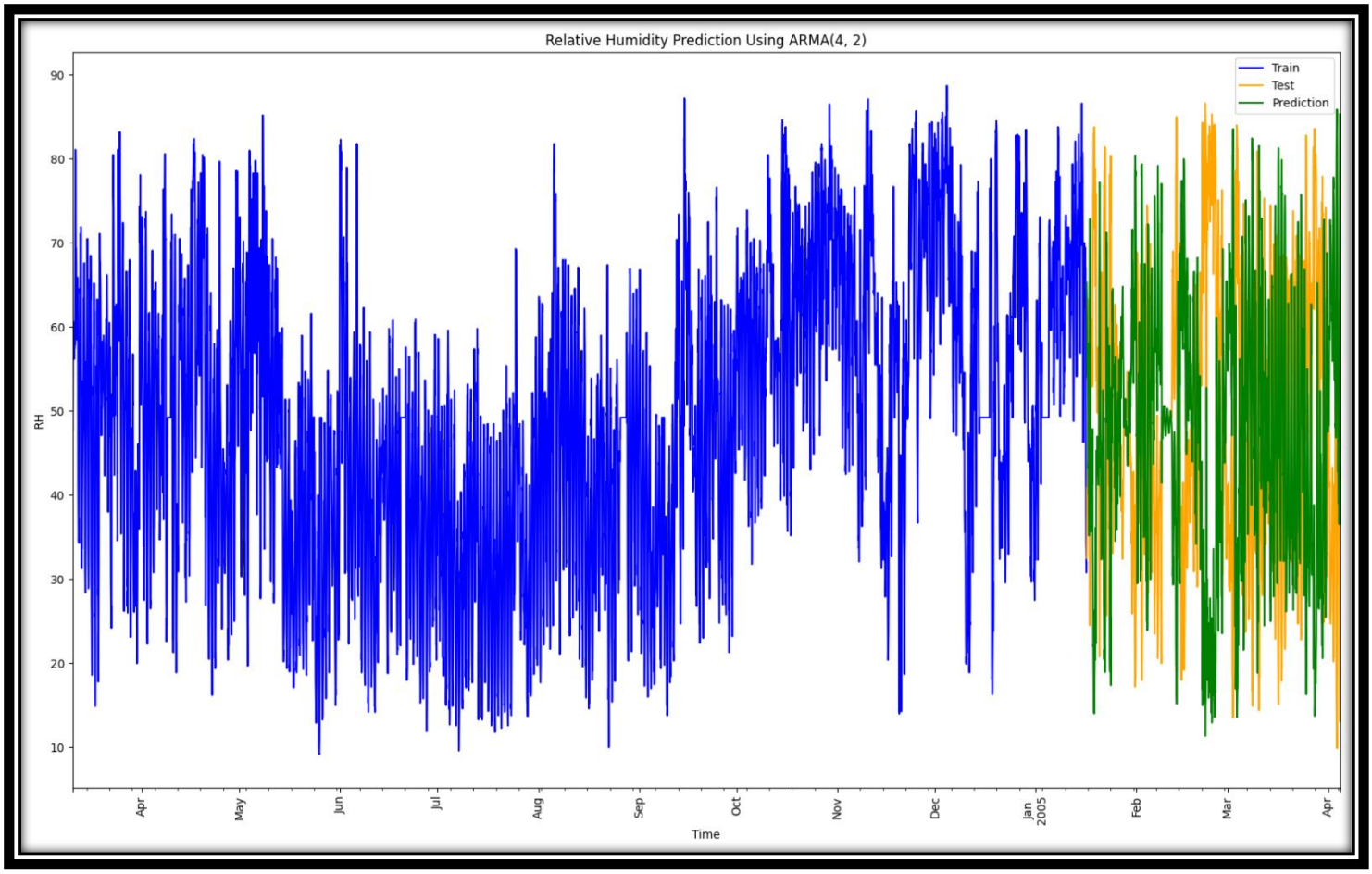
Plot of Prediction v/s actual : ARMA (2,7) Model



Plot of Prediction v/s actual : ARMA (4,0) Model



Plot of Prediction v/s actual : ARMA (2,3) Model

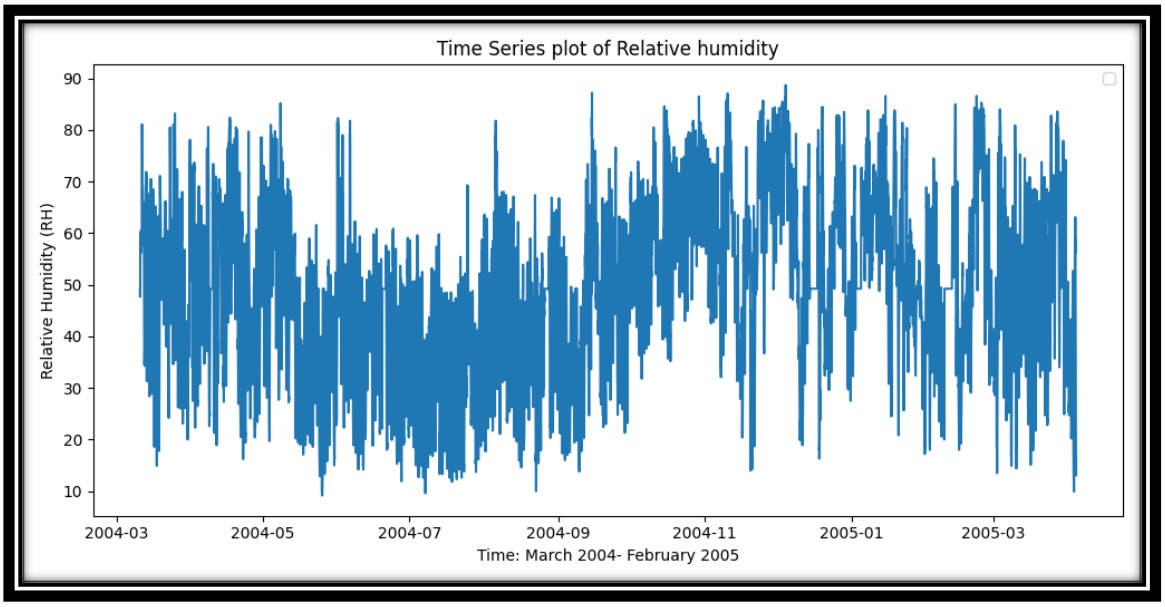


Plot of Prediction v/s actual : ARMA (4,2) Model

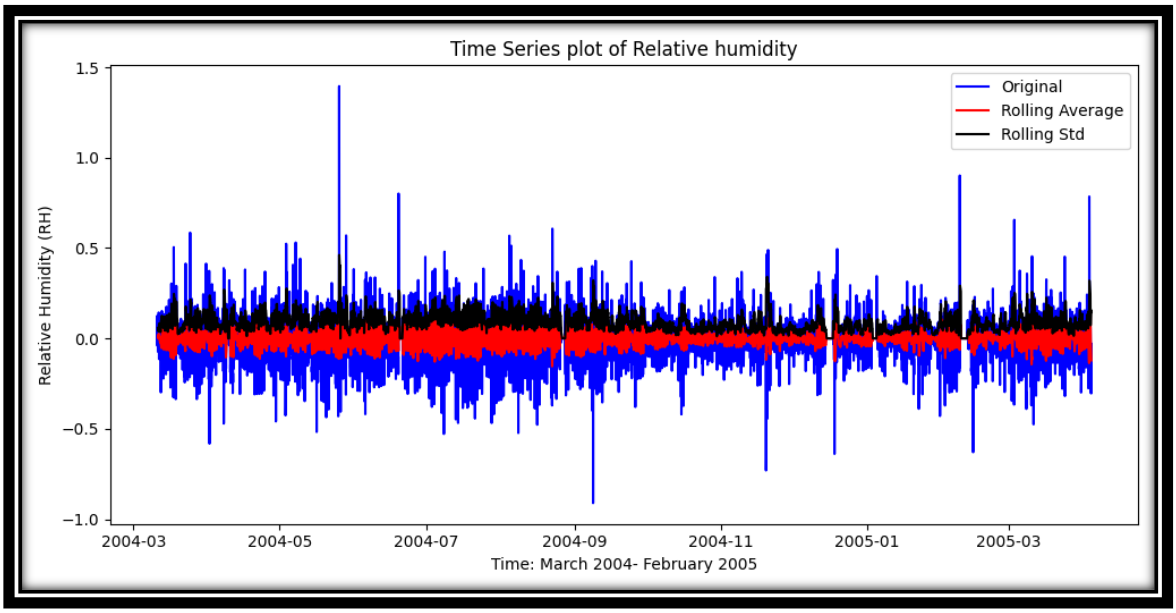
XVI. NEXT STEPS

We can tune the parameters to achieve a better model. Another common Time series model that is very popular is ARIMA. It stands for Autoregressive Integrated Moving average**.** While exponential smoothing models were based on a description of trend and seasonality in the data, ARIMA models aim to describe the correlations in the data with each other. An improvement over ARIMA is Seasonal ARIMA. It considers the seasonality of dataset just like Holt' Winter method.

We plot the relative humidity (RH) over time from March 2004 to February 2005. RH data is hourly averaged data from 03/2004 to 04/2005. From the plot we can observe the seasonal pattern where in there is a rise and fall in the data values that repeats regularly over the same time period.



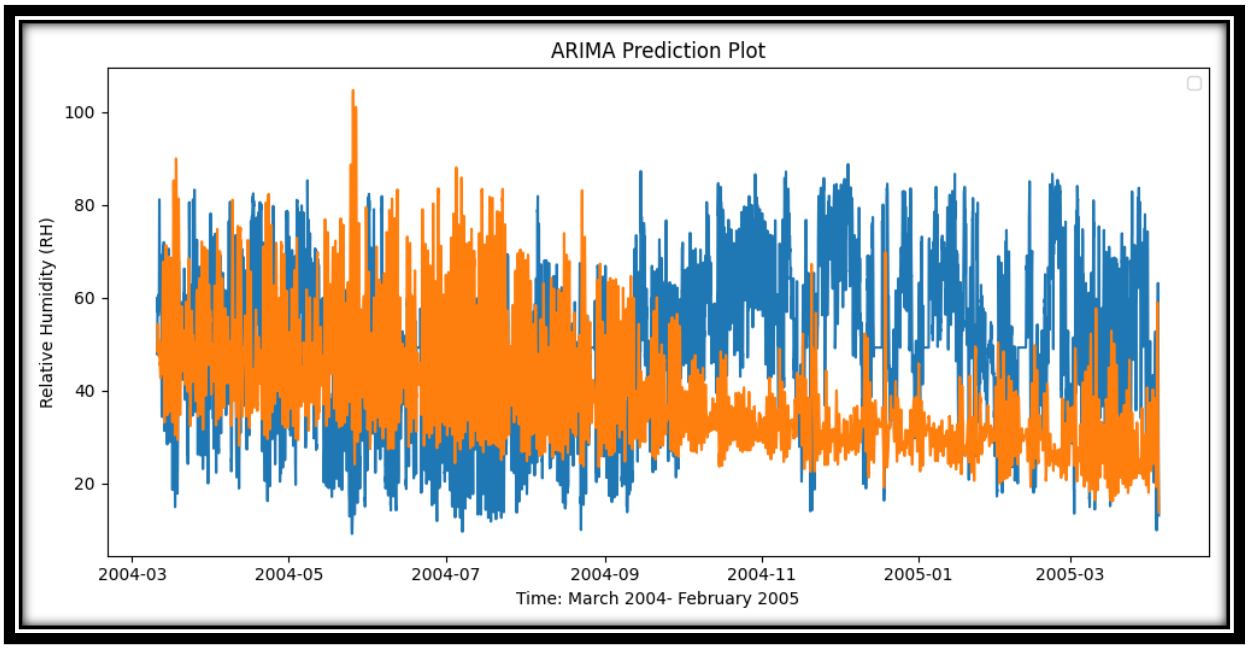
In the plot below we take the first difference and plot the rolling mean and average and notice that that after differencing there in same level.



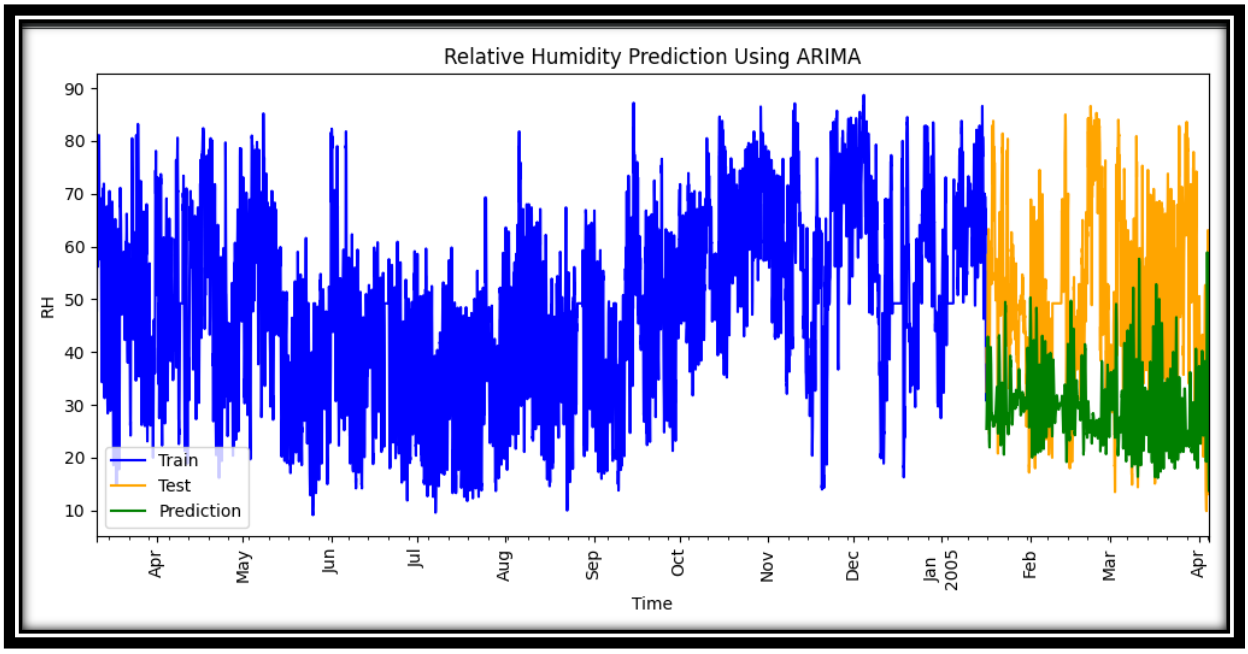
The MSE for ARIMA model is:**643.421**

The RMSE for ARIMA model is:**25.365744617495462.**

We can see that implementing ARIMA model with d as 1 generates a better model till now. We can tune the parameter using the validation set to generate even a better model. For future scope we may want to explore other models like SARIMA.



Plot of Arima Prediction



Plot of Arima prediction with train and test data

APPENDIX

```
import os
import statsmodels.api as sm
import pandas as pd
from requests import get
from io import BytesIO
from zipfile import ZipFile
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime
from statsmodels.tsa.seasonal import STL
from statsmodels.tsa.holtwinters import SimpleExpSmoothing

from functions import *
from pylab import rcParams
# from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.tsa.stattools import adfuller
from pandas.plotting import register_matplotlib_converters
from sklearn.model_selection import train_test_split
import statsmodels.tsa.holtwinters as ets
from sklearn.preprocessing import MinMaxScaler
import warnings
warnings.filterwarnings("ignore")
register_matplotlib_converters()


# Objective
# %%-----------------------------------------------------------------------------------------------------------
```

```
print("To predict the RELATIVE HUMIDITY of a given point of time based on the all other "
      "attributes affecting the change in RH")



# LOAD THE DATASET
# %%-----------------------------------------------------------------------------------------------------------
print('\n')
print(20 * "-" + "loading dataset..." + 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)
print(20 * "-" + "Download Complete!" + 20 * "-")



# PREPROCESSING
# %%-----------------------------------------------------------------------------------------------------------
print('\n')
print(20 * "-" + "PREPROCESSING" + 20 * "-")
df.describe()


# Investigate the first 5 rows
print(df.head(5))


# Removing last two Unnamed columns
df = df.drop(['Unnamed: 15','Unnamed: 16'], axis = 1)
```

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


# Changing the datatype from object to float
print('Datatypes in Dataframe\n',df.dtypes)
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)


print('\n')
# Dimension of the Dataset
print('Shape of the Dataset before null value removal:',df.shape)
# print('Shape of the Dataset:\n',df.shape)

# Handling null values
print('\n')
# Estimating null values
print(df.isnull().sum())

# Removing null values
null_data = df[df.isnull().any(axis=1)]
print('\n',null_data.head())
df= df.dropna()
# print(df.head)
```

```
print('Shape of the Dataset after null value removal',df.shape)

# 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)

# Handling Nan values
# Creating processed dataframe
print(df.isnull().sum())
print(df.head)
# 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')


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("cleaned dataset after preprocessing:\n", df)

print(df.shape)

# df.index = datetime

# df.to_csv("AQI.csv")

# print('CLEANDED DATASET:\n')

# print('CLEANDED DATASET:\n',df.head)


# Section 5: Description of the dataset. Describe the independent variable(s) and dependent
variable:

# %%-----------------------------------------------------------------------------------------------------------


# split into train and test(20%) dataset

train, test = split_df_train_test(df, 0.2)

print()


print("The dimension of train data is:")
```

```
print(train.shape)

# dimension of test data
print()

print("The dimension of test data is:")
print(test.shape)


# 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())

# dependent variable v/s time
plt.figure(figsize=(16,10))
plt.plot(df_rh,  label = 'RH')
plt.xlabel('Time: March 2004- February 2005')
plt.ylabel('Relative Humidity (RH)')
plt.title('Time Series plot of Relative humidity')
plt.legend(loc='best')
plt.show()

# ACF of the dependent variable.
autocorrelation = cal_auto_correlation(list(df["RH"]), 200)
plot_acf(autocorrelation, "ACF plot for Relative Humidity")


# auto corrleation using custom function
```

```
y = df['RH']
k = 20
acfcal = auto_corr_cal(y,k)
acfplotvals = acfcal[::-1] + acfcal[1:]
plt.figure(figsize=(16,10))
plt.stem(range(-(k - 1), k), acfplotvals)
plt.xlabel('Lags')
plt.ylabel('Magnitude')
plt.title('ACF using custom function 1')
plt.show()

# heatmap
corr = df.corr()
ax = sns.heatmap(corr, vmin=-1, vmax=1, center=0, square=True)
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
ax.set_xticklabels(ax.get_xticklabels(), rotation=45, horizontalalignment='right')
rcParams['figure.figsize'] = 16, 10
plt.show()

# section 6 : #Stationarity
# %%-----------------------------------------------------------------------------------------------------------
print('\n')
test_result = adfuller(df['RH'])
adfuller_test(df['RH'])

# section 7 : # Time series decomposition
# %%-----------------------------------------------------------------------------------------------------------
from pylab import rcParams
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')
```

```
plt.ylabel('Magnitude')
plt.title('Trend, Seasonality, Residual components using STL Decomposition')
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')
plt.ylabel('Magnitude')
plt.title('Original vs Seasonally adjusted')
plt.legend()
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))



# section 8 : # Base Models
# %%-----------------------------------------------------------------------------------------------------------------
# #performance for all the models
result_performance = pd.DataFrame(
    {"Model": [], "MSE": [], "RMSE": [], "Residual Mean": [], "Residual Variance": []})
```

```
print(20 * "-" + "AVERAGE MODEL" + 20 * "-")
# %%----------------------------------------------------------------------------------------------------------
average_predictions = generic_average_method(train["RH"], len(test["RH"]))
print(average_predictions)
print(test["RH"])
x = test["RH"] - average_predictions
print(x)

avg_mse = cal_mse(test["RH"], average_predictions)
print()
print("The MSE for Average model is:")
print(avg_mse)



avg_rmse = np.sqrt(avg_mse)
print()
print("The RMSE for Average model is:")
print(avg_rmse)

# forecast errors for average model
residuals_avg = cal_forecast_errors(test["RH"], average_predictions)

# average residual variance
avg_variance = np.var(residuals_avg)
print()
print("The Variance of residual for Average model is:")
print(avg_variance)

# Average residual mean
```

```
avg_mean = np.mean(residuals_avg)
print()
print("The Mean of residual for Average model is:")
print(avg_mean)

# Average residual ACF
residual_autocorrelation_average = cal_auto_correlation(residuals_avg, len(average_predictions))
plot_acf(residual_autocorrelation_average, "ACF plot using Average Residuals")



# add the results to common dataframe
result_performance = result_performance.append(
pd.DataFrame(
        {"Model": ["Average Model"], "MSE": [avg_mse], "RMSE": [avg_rmse],
         "Residual Mean": [avg_mean], "Residual Variance": [avg_variance]}))

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

plot_multiline_chart_pandas_using_index([train, test, average_df], "RH",
                                        ["Train", "Test", "Prediction"], ["Blue", "Orange", "Green"],
                                        "Time", "RH",
                                        "Relative Humidity Prediction Using Average",
                                        rotate_xticks=True)

print(20 * "-" + "NAIVE MODEL" + 20 * "-")
# %%-----------------------------------------------------------------------------------------------------------
# naive model
naive_predictions = generic_naive_method(train["RH"], len(test["RH"]))
```

```
# print(naive_predictions)
naive_mse = cal_mse(test["RH"], naive_predictions)
print()
print("The MSE for Naive model is:")
print(naive_mse)

naive_rmse = np.sqrt(naive_mse)
print()
print("The RMSE for Naive model is:")
print(naive_rmse)

# forecast errors for naive model
residuals_naive = cal_forecast_errors(test["RH"], naive_predictions)

# naive residual variance
naive_variance = np.var(residuals_naive)
print()
print("The Variance of residual for Naive model is:")
print(naive_variance)

# naive residual mean
naive_mean = np.mean(residuals_naive)
print()
print("The Mean of residual for Naive model is:")
print(naive_mean)

# naive residual ACF
residual_autocorrelation_naive = cal_auto_correlation(residuals_naive, len(naive_predictions))
plot_acf(residual_autocorrelation_naive, "ACF plot using Naive Residuals")
```

```
# add the results to common dataframe
result_performance = result_performance.append(
pd.DataFrame(
        {"Model": ["Naive Model"], "MSE": [naive_mse], "RMSE": [naive_rmse],
         "Residual Mean": [naive_mean], "Residual Variance": [naive_variance]}))

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

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

print(20 * "-" + "DRIFT MODEL" + 20 * "-")
# %%-----------------------------------------------------------------------------------------------------------------
# drift model
drift_predictions = generic_drift_method(train["RH"], len(test["RH"]))
# print(drift_predictions)
drift_mse = cal_mse(test["RH"], drift_predictions)
print()
print("The MSE for drift model is:")
print(drift_mse)

drift_rmse = np.sqrt(drift_mse)
print()
print("The RMSE for Drift model is:")
print(drift_rmse)
```

```
# forecast errors for drift model
residuals_drift = cal_forecast_errors(test["RH"], drift_predictions)

# drift residual variance
drift_variance = np.var(residuals_drift)
print()
print("The Variance of residual for Drift model is:")
print(drift_variance)

# drift residual mean
drift_mean = np.mean(residuals_drift)
print()
print("The Mean of residual for drift model is:")
print(drift_mean)


# drift residual ACF
residual_autocorrelation_drift = cal_auto_correlation(residuals_drift, len(drift_predictions))
plot_acf(residual_autocorrelation_drift, "ACF plot using drift Residuals")


# add the results to common dataframe
result_performance = result_performance.append(
pd.DataFrame(
        {"Model": ["Drift Model"], "MSE": [drift_mse], "RMSE": [drift_rmse],
         "Residual Mean": [drift_mean], "Residual Variance": [drift_variance]}))

# plot the predicted vs actual data
drift_df = test.copy(deep=True)
```

```
drift_df["RH"] = drift_predictions


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



print(20 * "-" + "HOLT WINTER MODEL" + 20 * "-")
# %%-----------------------------------------------------------------------------------------------------------------


# holt winter prediction
holt_winter_prediction = generic_holt_linear_winter(train["RH"], test["RH"], None, None,
                                                    "mul", None)
# print(holt_winter_prediction)


# holt winter mse
holt_winter_mse = cal_mse(test["RH"], holt_winter_prediction)



print()
print("The MSE for Holt Winter model is:")
print(holt_winter_mse)


# holt winter rmse
holt_winter_rmse = np.sqrt(holt_winter_mse)
print()
print("The RMSE for Holt Winter model is:")
print(holt_winter_rmse)
```

```
# holt winter residual

residuals_holt_winter = cal_forecast_errors(list(test["RH"]), holt_winter_prediction)

residual_autocorrelation_holt_winter = cal_auto_correlation(residuals_holt_winter,
len(holt_winter_prediction))


# holt winter residual variance

holt_winter_variance = np.var(residuals_holt_winter)

print()

print("The Variance of residual for Holt Winter model is:")

print(holt_winter_variance)


# holt winter residual mean

holt_winter_mean = np.mean(residuals_holt_winter)

print()

print("The Mean of residual for Holt Winter model is:")

print(holt_winter_mean)


# holt winter residual ACF

plot_acf(residual_autocorrelation_holt_winter, "ACF plot using Holt Winter Residuals")



# add the results to common dataframe

result_performance = result_performance.append(

pd.DataFrame(

        {"Model": ["Holt Winter Model"], "MSE": [holt_winter_mse], "RMSE": [holt_winter_rmse],

         "Residual Mean": [holt_winter_mean], "Residual Variance": [holt_winter_variance]}))


# plot the predicted vs actual data

holt_winter_df = test.copy(deep=True)
```

```
holt_winter_df["RH"] = holt_winter_prediction


plot_multiline_chart_pandas_using_index([train, test, holt_winter_df], "RH",
                                        ["Train", "Test", "Prediction"], ["Blue", "Orange", "Green"],
                                        "Time", "RH",
                                        "Relative Humidity Prediction Using Holt Winter",
                                        rotate_xticks=True)



print(20 * "-" + "SIMPLE AND EXPONENTIAL SMOOTHING" + 20 * "-")
# %%-----------------------------------------------------------------------------------------------------------------
from statsmodels.tsa.api import ExponentialSmoothing, SimpleExpSmoothing, Holt
y_hat_avg = test.copy()
fit2 = SimpleExpSmoothing(np.asarray(train["RH"])).fit(smoothing_level=0.6,optimized=False)
y_hat_avg['SES'] = fit2.forecast(len(test))
plt.figure(figsize=(16,10))
plt.plot(train['RH'], label='Train')
plt.plot(test['RH'], label='Test')
plt.plot(y_hat_avg['SES'], label='SES')
plt.legend(loc='best')
plt.show()

# sse mse
sse_mse = cal_mse(test["RH"], y_hat_avg['SES'])



print()
print("The MSE for Holt Winter model is:")
print(sse_mse)
```

```
# sse rmse
sse_rmse = np.sqrt(sse_mse)
print()
print("The RMSE for Holt Winter model is:")
print(sse_rmse)

# sse residual
residuals_sse = cal_forecast_errors(list(test["RH"]), y_hat_avg['SES'])
residual_autocorrelation_sse = cal_auto_correlation(residuals_sse, len(y_hat_avg['SES']))

# sse residual variance
sse_variance = np.var(residuals_sse)
print()
print("The Variance of residual for Holt Winter model is:")
print(sse_variance)

# sse residual mean
sse_mean = np.mean(residuals_sse)
print()
print("The Mean of residual for Holt Winter model is:")
print(sse_mean)

# sse residual ACF
plot_acf(residual_autocorrelation_sse, "ACF plot using simple exponential smoothing Residuals")


# add the results to common dataframe
result_performance = result_performance.append(
pd.DataFrame(
        {"Model": [" Simple Exponential Smoothing Model"], "MSE": [sse_mse], "RMSE": [sse_rmse],
```

```
        "Residual Mean": [sse_mean], "Residual Variance": [sse_variance]}))


print(20 * "-" + "MULTIPLE LINEAR REGRESSION" + 20 * "-")
# %%-----------------------------------------------------------------------------------------------------------
# combining train and test data
combined_data = train.append(test)

lm_combined = combined_data.copy(deep=True)
# separate train and test data
lm_train = lm_combined[:len(train)]
lm_test = lm_combined[len(train):]

# Scaling the data using MixMax Scaler
mm_scaler = MinMaxScaler()
lm_train_mm_scaled = pd.DataFrame(
mm_scaler.fit_transform(lm_train[np.setdiff1d(lm_train.columns, ["RH"])]),
    columns=np.setdiff1d(lm_train.columns, ["RH"]))
lm_train_mm_scaled.set_index(lm_train.index, inplace=True)
lm_train_mm_scaled["RH"] = lm_train["RH"]

lm_test_mm_scaled = pd.DataFrame(mm_scaler.transform(lm_test[np.setdiff1d(lm_test.columns,
["RH"])]),
                                 columns=np.setdiff1d(lm_test.columns, ["RH"]))
lm_test_mm_scaled.set_index(lm_test.index, inplace=True)
lm_test_mm_scaled["RH"] = lm_test["RH"]

print(lm_train_mm_scaled.columns)

# linear model using all variables
```

```
basic_model = normal_equation_using_statsmodels(
lm_train_mm_scaled[np.setdiff1d(lm_train_mm_scaled.columns, "RH")],
lm_train_mm_scaled["RH"], intercept=False)

print()
print("The summary of linear model with all variables is:")
print(basic_model.summary())


print(lm_train_mm_scaled.columns)
features_1 = np.setdiff1d(lm_train_mm_scaled.columns,
                          ["AH", "C6H6(GT)", "CO(GT)", "NMHC(GT)",
                           "NO2(GT)", "NOx(GT)", "PT08.S1(CO)", "PT08.S2(NMHC)",
                           "PT08.S3(NOx)", "PT08.S4(NO2)"])

pruned_model =
normal_equation_using_statsmodels(lm_train_mm_scaled[np.setdiff1d(features_1, "RH")],
                                  lm_train_mm_scaled["RH"], intercept=False)

print("The summary of linear model after feature selection:")
print(pruned_model.summary())

# linear model predictions
lm_predictions = normal_equation_prediction_using_statsmodels(pruned_model,
lm_test_mm_scaled[
    np.setdiff1d(features_1, "RH")], intercept=False)

# linear model mse
lm_mse = cal_mse(test["RH"], lm_predictions)

print()
```

```
print("The MSE for Linear Model is:")

print(lm_mse)


# linear model rmse

lm_rmse = np.sqrt(lm_mse)

print()

print("The RMSE for Linear Model model is:")

print(lm_rmse)


# plot the actual vs predicted values

lm_predictions_scaled = lm_test_mm_scaled.copy(deep=True)

lm_predictions_scaled["RH"] = lm_predictions



plot_multiline_chart_pandas_using_index([lm_train_mm_scaled, lm_test_mm_scaled,
lm_predictions_scaled],

                                        "RH",

                                        ["Train", "Test", "Prediction"], ["Blue", "Orange", "Green"],

                                        "Time", "RH",

                                        "Relative Humidity Prediction Using Multiple Linear Model",

                                        rotate_xticks=True)


features = lm_train_mm_scaled.columns


# linear model predictions

lm_predictions = normal_equation_prediction_using_statsmodels(basic_model,
lm_test_mm_scaled[

np.setdiff1d(features, "RH")], intercept=False)


# linear model mse

lm_mse = cal_mse(test["RH"], lm_predictions)
```

```
print()
print("The MSE for Linear Model is:")
print(lm_mse)

# linear model rmse
lm_rmse = np.sqrt(lm_mse)
print()
print("The RMSE for Linear Model model is:")
print(lm_rmse)

# linear model residual
residuals_lm = cal_forecast_errors(list(test["RH"]), lm_predictions)
residual_autocorrelation = cal_auto_correlation(residuals_lm, len(lm_predictions))

# linear model residual variance
lm_variance = np.var(residuals_lm)
print()
print("The Variance of residual for Linear Model model is:")
print(lm_variance)

# linear model residual mean
lm_mean = np.mean(residuals_lm)
print()
print("The Mean of residual for Linear Model model is:")
print(lm_mean)

# linear model residual ACF
plot_acf(residual_autocorrelation, "ACF plot for Linear Model Residuals")
```

```
# linear model Q value
Q_value_lm = box_pierce_test(len(test), residuals_lm, len(test))
print()
print("The Q Value of residuals for Linear Model model is:")
print(Q_value_lm)

# add the results to common dataframe
result_performance = result_performance.append(
    pd.DataFrame(
      {"Model": ["Multiple Linear Regression Model"], "MSE": [lm_mse], "RMSE": [lm_rmse],
       "Residual Mean": [lm_mean], "Residual Variance": [lm_variance]}))

# plot the actual vs predicted values
lm_predictions_scaled = lm_test_mm_scaled.copy(deep=True)
lm_predictions_scaled["RH"] = lm_predictions

plot_multiline_chart_pandas_using_index([lm_train_mm_scaled, lm_test_mm_scaled,
lm_predictions_scaled],
                                        "RH",
                                        ["Train", "Test", "Prediction"], ["Blue", "Orange", "Green"],
                                        "Time", "RH",
                                        "Relative Humidity Prediction Using Multiple Linear Model",
                                        rotate_xticks=True)

# Section 11: ARMA (ARIMA or SARIMA) model
# %%-----------------------------------------------------------------------------------------------------------
print('# %%-----------------------------------------------------------------------------------------------------------')
j = 12
k = 12
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")



# # estimate the order of the process

# # the possible orders identified from GPAC table don't pass the chi square test

possible_order2 = [(2, 3), (2, 7), (4, 0), (4, 2)]


print()

print("The possible orders identified from GPAC for ARMA process are:")

print(possible_order2)

print()

print("We noticed that none of the identified ARMA order from the GPAC table 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 = [(2, 7)]
possible_order = [(2, 3)]
# possible_order = [(4, 0)]
# possible_order = [(4, 2)]

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

n_a = 2
n_b = 3

model = statsmodels_estimate_parameters(n_a, n_b, y)
print(model.summary())

# ARMA predictions
arma_prediction = statsmodels_predict_ARMA_process(model, "2005-01-16 15:00:00", "2005-04-04
14:00:00")

# # add the subtracted mean back into the predictions
arma_prediction = np.add(y_mean, arma_prediction)
# print(arma_prediction)

# def cal_mse(actual_values, predicted_values):
#     return np.round(np.mean(np.square(np.subtract(predicted_values, actual_values))), 3)

# ARMA mse
```

```
arma_mse = cal_mse(test["RH"], arma_prediction)

print(f"The MSE for ARMA({n_a}, {n_b}) model is:")
print(arma_mse)

# ARMA rmse
arma_rmse = np.sqrt(arma_mse)
print()

print(f"The RMSE for ARMA({n_a}, {n_b}) model is:")
print(arma_rmse)


# ARMA residual
residuals_arma = cal_forecast_errors(list(test["RH"]), arma_prediction)

# ARMA residual variance
arma_variance = np.var(residuals_arma)
print()
print("The Variance of residual for ARMA model is:")
print(arma_variance)

# ARMA residual mean
arma_mean = np.mean(residuals_arma)
print()
print(f"The Mean of residual for ARMA({n_a}, {n_b}) model is:")
print(arma_mean)

# ARMA residual ACF
residual_autocorrelation_arma = cal_auto_correlation(residuals_arma, len(arma_prediction))
```

```
plot_acf(residual_autocorrelation_arma, f"ACF plot for ARMA({n_a}, {n_b}) Residuals")

# ARMA covariance matrix
print()
print(statsmodels_print_covariance_matrix(model, n_a, n_b))

# ARMA estimated variance of error
statsmodels_print_variance_error(model, n_a, n_b)



# add the results to common dataframe
result_performance = result_performance.append(
    pd.DataFrame(
      {"Model": [f"ARMA({n_a}, {n_b}) Model"], "MSE": [arma_mse], "RMSE": [arma_rmse],
       "Residual Mean": [arma_mean], "Residual Variance": [arma_variance]}))

# 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"Relative Humidity Prediction Using ARMA({n_a}, {n_b})",
                                        rotate_xticks=True)
# Section Extra: ARIMA (ARIMA or SARIMA) model
# %%-----------------------------------------------------------------------------------------------------------------

#Determine the rolling statistics
rolmean = df_rh.rolling(window=12).mean()
```

```
rolstd = df_rh.rolling(window=12).std()
# print(rolmean,rolstd)
from matplotlib import pyplot
from pandas.plotting import autocorrelation_plot
rcParams['figure.figsize'] = 15,10
autocorrelation_plot(df_rh)
plt.xlabel('Lags')
plt.ylabel('Magnitude')
plt.title('Time Series plot of Relative humidity')
plt.legend(loc='best')
plt.show()

pyplot.show()

rcParams['figure.figsize'] = 10,5
pyplot.plot(df_rh)
plt.xlabel('Time: March 2004- February 2005')
plt.ylabel('Relative Humidity (RH)')
plt.title('Time Series plot of Relative humidity')
plt.legend(loc='best')
plt.show()


#Estimate trend
df_RH_logScale = np.log(df_rh)
plt.plot(df_RH_logScale)
plt.xlabel('Time: March 2004- February 2005')
plt.ylabel('Relative Humidity (RH)')
plt.title('Time Series plot of Relative humidity in Log scale')
plt.legend(loc='best')
```

```
plt.show()


from statsmodels.tsa.stattools import adfuller
def test_stationary(timeseries):
    movingAverage = timeseries.rolling(window=12).mean()
    movingStd = timeseries.rolling(window=12).std()


    # Plot rolling
    orig = plt.plot(timeseries, color='blue', label='Original')
    mean = plt.plot(movingAverage, color='red', label='Rolling Average')
    std = plt.plot(movingStd, color='black', label='Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean & Standard Deviation')
    plt.xlabel('Time: March 2004- February 2005')
    plt.ylabel('Relative Humidity (RH)')
    plt.title('Time Series plot of Relative humidity')
    plt.legend(loc='best')
    plt.show()


    plt.show(block=False)


    # perform Dickey Fuller Test
    print('Result of Dickey Fuller Test')
    dftest = adfuller(timeseries['RH'], autolag='AIC')
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic', 'p-value', '#Lags Used', 'Number of
observation Used'])
    for key, value in dftest[4].items():
        dfoutput['Critical Value (%s)' % key] = value
    print(dfoutput)
```

```
exponentialDecayWeightedAverage =
df_RH_logScale.ewm(halflife=12,min_periods=0,adjust=True).mean()

plt.plot(df_RH_logScale)

plt.plot(exponentialDecayWeightedAverage,color='red')

plt.xlabel('Time: March 2004- February 2005')

plt.ylabel('Relative Humidity (RH)')

plt.title('Time Series plot of Relative humidity in Log scale')

plt.legend(loc='best')

plt.show()



datasetLogScaleMinusMovingExponentialDecayAverage = df_RH_logScale -
exponentialDecayWeightedAverage

test_stationary(datasetLogScaleMinusMovingExponentialDecayAverage)

datasetLogDiffShifting  = df_RH_logScale-df_RH_logScale.shift()

plt.plot(datasetLogDiffShifting)

plt.xlabel('Time: March 2004- February 2005')

plt.ylabel('Relative Humidity (RH)')

plt.title('Time Series plot of Relative humidity')

plt.legend(loc='best')

plt.show()

plt.show()


datasetLogDiffShifting.dropna(inplace=True)

test_stationary(datasetLogDiffShifting)


from statsmodels.tsa.arima_model import ARIMA

#ARIMMA Model

model = ARIMA(df_RH_logScale,order=(2, 1, 3))
```

```
# print(model)
result_ARIMA= model.fit(disp=-1)
plt.plot(datasetLogDiffShifting)
plt.plot(result_ARIMA.fittedvalues,color='red')
plt.title('RSS: %.4f'%sum((result_ARIMA.fittedvalues-datasetLogDiffShifting["RH"])**2))
print('Plotting ARIMA Model')
plt.xlabel('Time: March 2004- February 2005')
plt.ylabel('Relative Humidity (RH)')
plt.title('Time Series plot of Relative humidity')
plt.legend(loc='best')
plt.show()

prediction_ARIMA_diff = pd.Series(result_ARIMA.fittedvalues,copy=True)
prediction_ARIMA_diff_cumsum = prediction_ARIMA_diff.cumsum()
prediction_ARIMA_log = pd.Series(df_RH_logScale['RH'].iloc[0], index=df_RH_logScale.index)
prediction_ARIMA_log = prediction_ARIMA_log.add(prediction_ARIMA_diff_cumsum,fill_value =0)

prediction_ARIMA = np.exp(prediction_ARIMA_log)
plt.plot(df_rh)
plt.plot(prediction_ARIMA)
plt.xlabel('Time: March 2004- February 2005')
plt.ylabel('Relative Humidity (RH)')
plt.title('ARIMA Prediction Plot')
plt.legend(loc='best')
plt.show()



result_ARIMA.plot_predict(1,264)
plt.xlabel('Time: March 2004- February 2005')
```

```
plt.ylabel('Relative Humidity (RH)')
plt.title('Prediction v/s Forecast')
plt.legend(loc='best')
plt.show()




print(result_ARIMA.summary().tables[1])

# ARMA mse
arima_mse = cal_mse(test["RH"], prediction_ARIMA)

print(f"The MSE for ARIMA  model is:")
print(arima_mse)

# ARMA rmse
arima_rmse = np.sqrt(arima_mse)
print()

print(f"The RMSE for ARIMA model is:")
print(arima_rmse)

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

plot_multiline_chart_pandas_using_index([train, test, arima_df], "RH",
                                        ["Train", "Test", "Prediction"], ["Blue", "Orange", "Green"],
                                        "Time", "RH",
                                        f"Relative Humidity Prediction Using ARIMA",
```

```
                                rotate_xticks=True)


# ------------------------------------------Final Performance Metrics----------------------
print()
print("The performance metrics for all the models is shown:")
print(result_performance.sort_values(["RMSE"]).reset_index(drop=True).to_string())



#Autocorrelation Function
import numpy as np
import pandas as pd
import seaborn as sns
import statsmodels.tsa.holtwinters as ets
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from statsmodels.tsa.stattools import adfuller
from pylab import rcParams
import statsmodels.api as sm
from scipy.stats import chi2
# %%-------------------------------------------------------------------------------------------------------
def split_df_train_test(df, test_size, random_seed=42):
    train, test = train_test_split(df, shuffle=False, test_size=test_size, random_state=random_seed)
    return train, test

# %%-------------------------------------------------------------------------------------------------------


def auto_corr(y,k):
    T = len(y)
    y_mean = np.mean(y)
```

```
    res_num = 0
    res_den = 0
    for t in range(k,T):
        res_num += (y[t] - y_mean) * (y[t-k] - y_mean)

    for t in range(0,T):
        res_den += (y[t] - y_mean)**2

    res = res_num/res_den
    return res

def auto_corr_cal(y,k):
    res = []
    for t in range(0,k):
        result = auto_corr(y,t)
        res.append(result)
    return res

def cal_auto_correlation(input_array, number_of_lags, precision=3):

    mean_of_input = np.mean(input_array)
    result = []
    denominator = np.sum(np.square(np.subtract(input_array, mean_of_input)))
    for k in range(0, number_of_lags):
        numerator = 0
        for i in range(k, len(input_array)):
            numerator += (input_array[i] - mean_of_input) * (input_array[i - k] - mean_of_input)
        if denominator != 0:
            result.append(np.round(numerator / denominator, precision))
    return result
```

```
def plot_acf(autocorrelation, title_of_plot, x_axis_label="Lags", y_axis_label="Magnitude"):
    # make a symmetric version of autocorrelation using slicing
    symmetric_autocorrelation = autocorrelation[:0:-1] + autocorrelation
    x_positional_values = [i * -1 for i in range(0, len(autocorrelation))][:0:-1] + [i for i in
                                                                                      range(0, len(autocorrelation))]
    # plot the symmetric version using stem
    rcParams['figure.figsize'] = 16, 10
    plt.stem(x_positional_values, symmetric_autocorrelation, use_line_collection=True)
    plt.xlabel(x_axis_label)
    plt.ylabel(y_axis_label)
    plt.title(title_of_plot)
    plt.figure(figsize=(16, 10))
    plt.show()


# %%-----------------------------------------------------------------------------------------------------------
def adfuller_test(RH):
    result = adfuller(RH)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations Used']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )
    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data has no
unit root and is stationary")
    else:
        print("weak evidence against null hypothesis, time series has a unit root, indicating it is non-
stationary ")
# %%-----------------------------------------------------------------------------------------------------------


def get_max_denominator_indices(j, k_scope):
```

```
    # create denominator indexes based on formula for GPAC
    denominator_indices = np.zeros(shape=(k_scope, k_scope), dtype=np.int64)

    for k in range(k_scope):
        denominator_indices[:, k] = np.arange(j - k, j + k_scope - k)

    return denominator_indices

def get_apt_denominator_indices(max_denominator_indices, k):
    apt_denominator_indices = max_denominator_indices[-k:, -k:]
    return apt_denominator_indices


def get_numerator_indices(apt_denominator_indices, k):
    numerator_indices = np.copy(apt_denominator_indices)
    # take the 0,0 indexed value and then create a range of values from (indexed_value+1,
indexed_value+k)
    indexed_value = numerator_indices[0, 0]
    y_matrix = np.arange(indexed_value + 1, indexed_value + k + 1)

    # replace the last column with this new value
    numerator_indices[:, -1] = y_matrix

    return numerator_indices

def get_ACF_by_index(numpy_indices, acf):
    # select values from an array based on index specified
    result = np.take(acf, numpy_indices)
    return result
```

```
def get_phi_value(denominator_indices, numerator_indices, ry, precision=5):
    # take the absolute values since when computing phi value, we use ACF and ACF is symmetric in
nature
    denominator_indices = np.abs(denominator_indices)
    numerator_indices = np.abs(numerator_indices)

    # replace the indices with the values of ACF
    denominator = get_ACF_by_index(denominator_indices, ry)
    numerator = get_ACF_by_index(numerator_indices, ry)

    # take the determinant
    denominator_det = np.round(np.linalg.det(denominator), precision)
    numerator_det = np.round(np.linalg.det(numerator), precision)

    # divide it and return the value of phi
    return np.round(np.divide(numerator_det, denominator_det), precision)

def create_gpac_table(j_scope, k_scope, ry, precision=5):
    # initialize gpac table
    gpac_table = np.zeros(shape=(j_scope, k_scope), dtype=np.float64)
    for j in range(j_scope):
        # create the largest denominator
        max_denominator_indices = get_max_denominator_indices(j, k_scope)

        for k in range(1, k_scope + 1):
            #  slicing largest denominator as required
            apt_denominator_indices = get_apt_denominator_indices(max_denominator_indices, k)

            # for numerator replace denominator's last columnn with index starting from j+1 upto k times
            numerator_indices = get_numerator_indices(apt_denominator_indices, k)
```

```
            # compute phi value
            phi_value = get_phi_value(apt_denominator_indices, numerator_indices, ry, precision)
            gpac_table[j, k - 1] = phi_value

    gpac_table_pd = pd.DataFrame(data=gpac_table, columns=[k for k in range(1, k_scope + 1)])

    return gpac_table_pd


def plot_heatmap(corr_df, title, xticks=None, yticks=None, x_axis_rotation=0, annotation=True):
    sns.heatmap(corr_df, annot=annotation)
    plt.title(title)
    if xticks is not None:
        plt.xticks([i for i in range(len(xticks))], xticks, rotation=x_axis_rotation)
    if yticks is not None:
        plt.yticks([i for i in range(len(yticks))], yticks)
    plt.show()
# %%-----------------------------------------------------------------------------------------------------------
# average method
def generic_average_method(input_data, step_ahead):
    # returns a flat prediction
    return [np.round(np.mean(input_data), 3) for i in range(0, step_ahead)]

# naive method
def generic_naive_method(input_data, step_ahead):
    """Predicts using naive method for specified steps"""
    return [input_data[-1] for i in range(0, step_ahead)]

# drift method
```

```
def generic_drift_method(input_data, step_ahead):
    predicted_values = []

    for i in range(0, step_ahead):
        predicted_value = input_data[-1] + (i + 1) * ((input_data[-1] - input_data[0]) / (len(input_data) -
1))

        predicted_values.append(round(predicted_value, 3))

    return predicted_values

# holt_linear_winter_method
def generic_holt_linear_winter(train_data, test_data, seasonal_period: int, trend="mul",
seasonal="mul",
                        trend_damped=False):
    """ Works best for data with trend and seasonality"""
    holt_winter = ets.ExponentialSmoothing(train_data, trend=trend, seasonal=seasonal,
                                   seasonal_periods=seasonal_period, damped=trend_damped).fit()
    holt_winter_forecast = list(holt_winter.forecast(len(test_data)))
    return holt_winter_forecast

# %%-----------------------------------------------------------------------------------------------------------
# mean square error
def cal_mse(actual_values, predicted_values):
    return np.round(np.mean(np.square(np.subtract(predicted_values, actual_values))), 3)

# forecast errors is difference between  observed values and predicted values
def cal_forecast_errors(actual_values, predicted_values):
    # forecast errors is difference between  observed values and predicted values
    return np.subtract(actual_values, predicted_values)
```

```
# %%-----------------------------------------------------------------------------------------------------------------

def box_pierce_test(number_of_samples, residuals, lags):

    return round(number_of_samples * np.sum(np.square(cal_auto_correlation(residuals, lags +
1)[1:])), 3)


# %%-----------------------------------------------------------------------------------------------------------------

def plot_multiline_chart_pandas_using_index(list_of_dataframes, y_axis_common_data,
list_of_label, list_of_color,

                                            x_label, y_label, title_of_plot, rotate_xticks=False):

    for i, df in enumerate(list_of_dataframes):

        df[y_axis_common_data].plot(label=list_of_label[i], color=list_of_color[i])

    plt.legend(loc='best')

    plt.xlabel(x_label)

    plt.ylabel(y_label)

    plt.title(title_of_plot)

    if rotate_xticks:

        plt.xticks(rotation=90)

    plt.figure(figsize=(16, 10))

    plt.show()

# %%-----------------------------------------------------------------------------------------------------------------

def plot_multiline_chart_pandas(list_of_dataframes, x_axis_common_data, y_axis_common_data,
list_of_label,

                                list_of_color,

                                x_label, y_label, title_of_plot, rotate_xticks=False):

    """Plots multiple line charts into single chart. This API uses list of pandas data having same x_axis
label and

    same y_axis label """

    for i, df in enumerate(list_of_dataframes):

        plt.plot(df[x_axis_common_data], df[y_axis_common_data], label=list_of_label[i],
color=list_of_color[i])
```

```
    plt.legend(loc='best')
    plt.xlabel(x_label)
    plt.ylabel(y_label)
    plt.title(title_of_plot)
    if rotate_xticks:
        plt.xticks(rotation=90)
    plt.show()


def chi_square_test(Q, lags, n_a, n_b, alpha=0.01):
    dof = lags - n_a - n_b
    chi_critical = chi2.isf(alpha, df=dof)

    if Q < chi_critical:
        print(f"The residual is white and the estimated order is n_a= {n_a} and n_b = {n_b}")
    else:
        print(f"The residual is not white with n_a={n_a} and n_b={n_b}")

    return Q < chi_critical

def Q_value(y, autocorrelation_of_residuals):
    """ Computes Q value for comparing with chi_critical for Chi Square Test. Same as
box_pierce_test(..)"""
    Q = len(y) * np.sum(np.square(autocorrelation_of_residuals[1:]))
    return Q

def statsmodels_estimate_parameters(n_a, n_b, y, trend="nc"):
    model = sm.tsa.ARMA(y, (n_a, n_b)).fit(trend=trend, disp=0)
    return model
```

```
def statsmodels_print_parameters(model, n_a, n_b):
    # print the parameters which are estimated
    for i in range(n_a):
        print("The AR coefficients a {}".format(i), "is:", model.params[i])
    print()
    for i in range(n_b):
        print("The MA coefficients b {}".format(i), "is:", model.params[i + n_a])
    print()


def statsmodels_print_covariance_matrix(model, n_a, n_b):
    print(f"Estimated covariance matrix for n_a = {n_a} and n_b = {n_b}: \n{model.cov_params()}")
    print()
    return model.cov_params()


def statsmodels_print_variance_error(model, n_a, n_b):
    print(f"Estimated variance of error for n_a = {n_a} and n_b = {n_b}: \n{model.sigma2}")
    print()
    return model.sigma2


def statsmodels_print_confidence_interval(model, n_a, n_b):
    # confidence interval
    print(
        f"The confidence interval for estimated parameters for n_a = {n_a} and n_b = {n_b}: \n
{model.conf_int()}")
    print()
    return model.conf_int()
```

```
def statsmodels_predict_ARMA_process(model, start, stop):
    model_hat = model.predict(start=start, end=stop)
    return model_hat


def statsmodels_plot_predicted_true(y, model_hat, n_a, n_b):
    true_data = pd.DataFrame({"Magnitude": y, "Samples": [i for i in range(len(y))]})
    fitted_data = pd.DataFrame({"Magnitude": model_hat, "Samples": [i for i in
range(len(model_hat))]})


    plot_multiline_chart_pandas([true_data, fitted_data], "Samples", "Magnitude", ["True data",
"Fitted data"],
                                ["red", "blue"], "Samples", "Magnitude",
                                f"ARMA process with n_a={n_a} and n_b={n_b}")


def statsmodels_print_roots_AR(model):
    print("Real part:")
    for root in model.arroots:
        print(root.real)
    print("Imaginary part:")
    for root in model.arroots:
        print(root.imag)


def statsmodels_print_roots_MA(model):
    print("Real part:")
    for root in model.maroots:
        print(root.real)
```

```
    print("Imaginary part:")
    for root in model.maroots:
        print(root.imag)



def gpac_order_chi_square_test(possible_order_ARMA, train_data, start, stop, lags,
actual_outputs):
    results = []


    for n_a, n_b in possible_order_ARMA:
        try:
            # estimate the model parameters
            model = statsmodels_estimate_parameters(n_a, n_b, train_data)



            # performing h step predictions
            predictions = statsmodels_predict_ARMA_process(model, start=start, stop=stop)


            # # add mean back to the forecast values
            # predictions = np.add(mean_to_add, predictions)


            # calculate forecast errors
            residuals = cal_forecast_errors(actual_outputs, predictions)


            # autocorrelation of residuals
            re = cal_auto_correlation(residuals, lags)


            # compute Q value for chi square test
            Q = Q_value(actual_outputs, re)
```

```
            # checking the chi square test
            if chi_square_test(Q, lags, n_a, n_b):
                results.append((n_a, n_b))

        except Exception as e:
            # print(e)
            pass

    return results

def normal_equation_using_statsmodels(train_feature_list, train_target_list, intercept=True):
    if intercept:
        train_feature_list = sm.add_constant(train_feature_list)

    model = sm.OLS(train_target_list, train_feature_list)
    results = model.fit()
    return results

def normal_equation_prediction_using_statsmodels(OLS_model, test_feature_list, intercept=True):
    if intercept:
        test_feature_list = sm.add_constant(test_feature_list, has_constant='add')

    predicted_values_OLS = OLS_model.predict(test_feature_list)
    return predicted_values_OLS
```