DATS 6450

Time Series Modelling & Analysis

Instructor Name: Dr. Reza Jafari

Term Project

Weather Forecasting - Hourly Temperature of Seattle City

Gayathri Chandrasekaran
Graduate Student, Data Science Program
George Washington University

12/16/2020

Table of Contents

Abstract:

In this project, we are predicting the hourly temperature of the Seattle city in Kelvin. Both conventional base model and linear ARMA (ARIMA/ SARIMA) model techniques are used to forecast the temperature. A comparative analysis on the forecast and predicted errors statistics is performed to choose the best model for the given data. H step prediction model is built, and temperature values are forecasted for the h-step (size of the test set).

Introduction:

Weather forecast is the branch of science to predict the conditions of the atmosphere for the given location and time. This is more relatable as this helps to plan everyday travel and other related activities. Weather warnings are the most important forecasts as they protect life and property from adverse damage. In this project, I have used hourly temperature data of the Seattle city and built a prediction model to forecast the upcoming temperature. I have made use of various time series model techniques like average method, naïve method, drift method, simple exponential smoothing, holts' linear method, holts winter method and ARMA methods to build the prediction model. Also, I have performed multivariate regression analysis on the dataset to check the linear dependency of the target variable with the regressor. A comparative study is performed to determine the best model by evaluating the results of these models like MSE values, variance & mean of the predicted error & forecast error, Q value and chi square test results. Using this best model,the h step forecast is performed on the test set.

Data Description

The Data used in the project is taken from the Kaggle. This data corresponds to the historical hourly temperature of the top US states from 2012 to 2017. The scope of this project is restricted to the Seattle city temperature data.

Data Source:

[Historical Hourly Weather Data 2012-2017 | Kaggle]

This dataset contains below dependent and independent variables.

Dependent Variable:

Temperature – hourly temperature of the Seattle city in Kelvin scale.

Independent Variable:

Humidity – concentration of water vapor present in the atmosphere at given hour

Wind speed – speed of the air movement at given hour

Wind Direction – direction of the air movement 0 - 360

Pressure – Atmospheric pressure for given hour at Seattle

Weather Description – categorical description of the weather at a given hour – It has 24 categories like sky is clear, rainy, snow, fog, etc.

Datetime – "2012-10-01 13:00:00" Format

Sample Dataset information:

```
First five observations from dataset
                      Humidity  Wind Speed  ...  Weather Description  Temperature
datetime                                    ...
2012-10-01 13:00:00       81.0           0  ...         sky is clear   281.800000
2012-10-01 14:00:00       80.0           0  ...         sky is clear   281.797217
2012-10-01 15:00:00       80.0           0  ...         sky is clear   281.789833
2012-10-01 16:00:00       79.0           0  ...         sky is clear   281.782449
2012-10-01 17:00:00       79.0           0  ...         sky is clear   281.775065

[5 rows x 6 columns]
Full data
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 45252 entries, 2012-10-01 13:00:00 to 2017-11-30 00:00:00
Data columns (total 6 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Humidity             44964 non-null  float64
 1   Wind Speed           45252 non-null  int64
 2   Wind Direction       45252 non-null  int64
 3   Pressure             45240 non-null  float64
 4   Weather Description  45252 non-null  object
 5   Temperature          45250 non-null  float64
dtypes: float64(3), int64(2), object(1)
```

Data Preprocessing:

Missing Value Ratio:

From the data set information, few missing values are observed. Hence, comparing the proportion of the missing values present in the dataset is calculated to make a decision on the missing value imputation method.

```
Humidity               0.636436
Wind Speed             0.000000
Wind Direction         0.000000
Pressure               0.026518
Weather Description    0.000000
Temperature            0.004420
```

From the above results, I could observe that less than 1% of the data is missing in the dataset, hence I have planned to forward fill the data as its hourly data there won't be much variation to the quantitative values from the previous hour.

Data sampling approach:

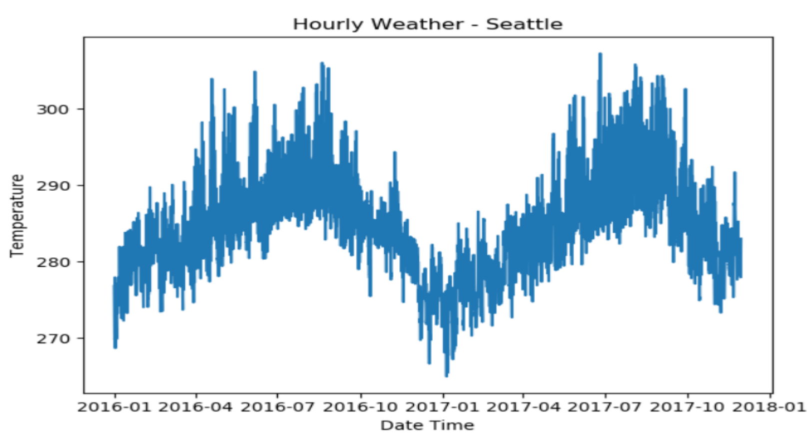
Initial dataset contains almost 45242 samples. For computational ease, I have sampled the last two data for this project. The data from 01/2016 to 11/2017 is used in this project.

The below snippet shows the information on the sampled data.

```
Data structure - Seattle Dataset
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 16777 entries, 2016-01-01 00:00:00 to 2017-11-30 00:00:00
Data columns (total 6 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Humidity             16777 non-null  float64
 1   Wind Speed           16777 non-null  int64
 2   Wind Direction       16777 non-null  int64
 3   Pressure             16777 non-null  float64
 4   Weather Description  16777 non-null  object
 5   Temperature          16777 non-null  float64
dtypes: float64(3), int64(2), object(1)
```

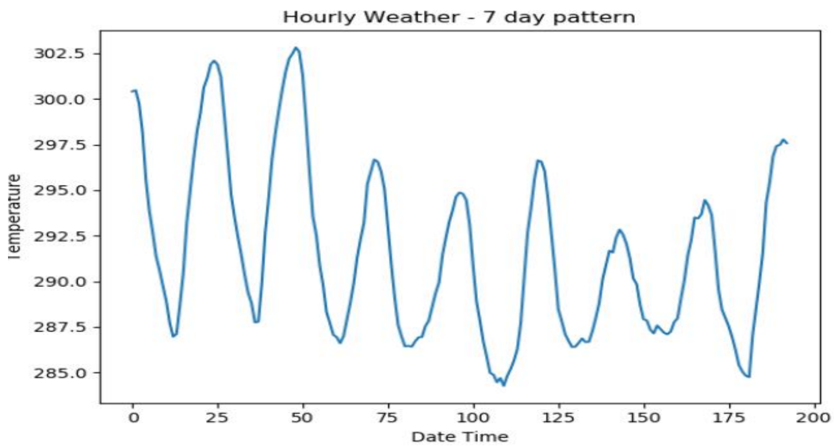
As stated earlier, weather description contains 24 categories, to reduce the load on the regressor model and most of the description can be correlated from other factors like humidity, wind speed and pressure. Hence this categorical variable is dropped from the dataset.

Temperature pattern over time



From the above plot, we could observe strong trend, seasonality, and cyclic pattern present in the dataset.

Let’s take a closer look at the seasonal part of the dataset, to understand more on the hourly data pattern. For this, I have sampled 7-day data from the dataset and plotted the data.

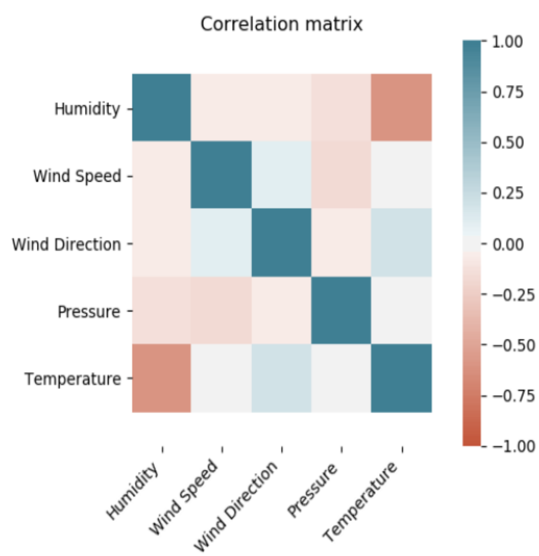


This plot uses the data from July 28, 2016 00:00:00 to Aug 04,2016 23:00:00. We could see a repeating pattern every 24-time cycle. Also, seasonal spike is not the same, this information is helpful while selecting “add” or “mul” decomposition values in holts’ winter/linear method.

Due to this cyclic nature and multiple seasonality, there is possibility that data may be highly non-linear. To overcome this issue, I have resampled the data as per seasonal order and developed four models. Hence, data is sub-sampled as Spring data – March to May, Summer data – June to August, Fall data – September to November and Winter data – December to Feb. For this split, I have used 2016 data only.

Correlation Matrix:

Let’s understand correlation between regressor (i.e.) independent variable with the temperature variable.



From the correlation heat map, except humidity, none of the variables have correlation with the target. Also, there is no multicollinearity within dependent variables like humidity, wind speed, wind direction, pressure.

Train and Test Split:

I have split sampled data into a train and test set with a split ratio of 80:20. After the train & test split, number of samples in each group is given as

Spring Data:

Train Samples: 1767

Test Samples: 442

Summer Data:

Train Samples: 1767

Test Samples: 442

Fall Data:

Train Samples: 1748

Test Samples: 437

Winter Data:

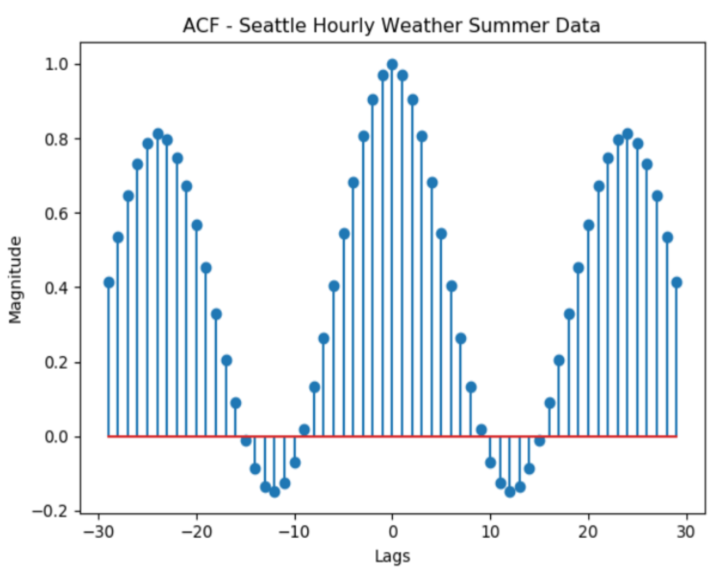
Train Samples: 1728

Test Samples: 433

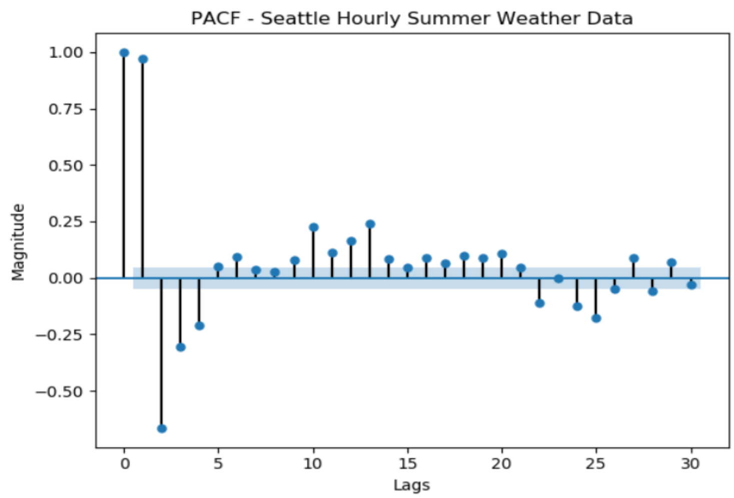
In this project, I have used Summer data to built the model and run the prediction over it.

Stationarity Test:

ACF Plots:



PACF Plots:

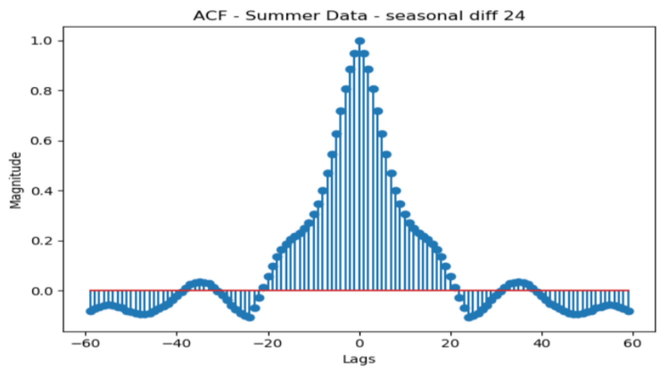


ADF Test Results:

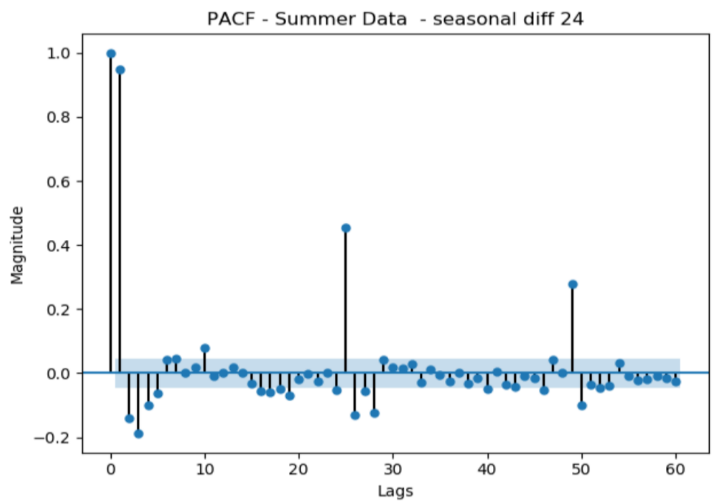
```
Hourly Weather Analysis - Summer Data
ADF Statistics: -3.211735
p value: 0.019315
Critical Values:
     1% : -3.434
     5% : -2.863
     10% : -2.568
```

Although ADF test result suggests data is stationary with p value less than significance value of 0.05 and confidence interval of 95% as ASDF stats is less than 5% of the CI value, ACF plot lags values are decaying slowly with repeating pattern of 24 lags suggests that data is not stationary. This calls for the transformation like differencing.

Due to the seasonal nature of the data, I have performed seasonal differencing of period 24 over the data set. Let's check the ACF plot for this differenced data, ACF Plot:

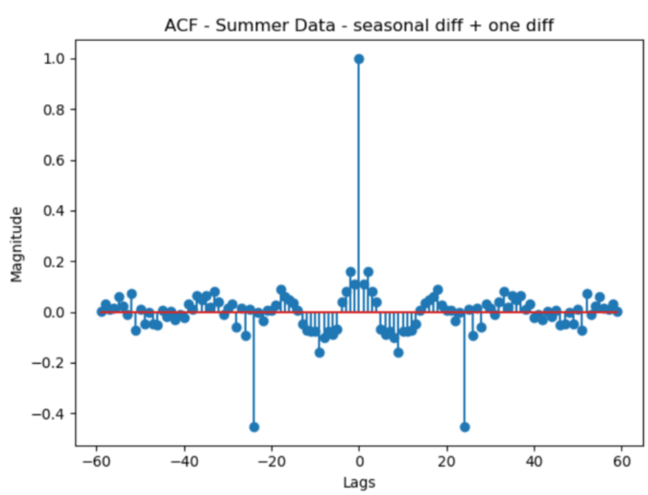


PCF Plot:

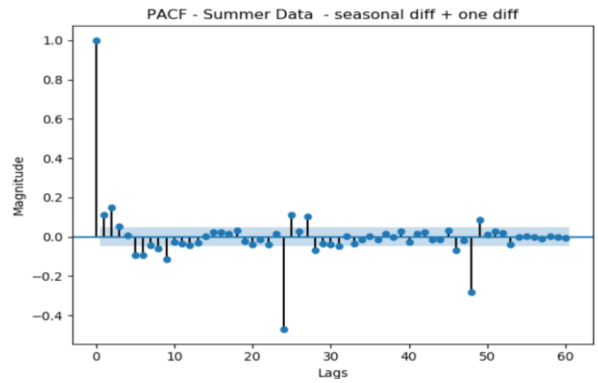


From the above plots, we could see that seasonal difference transformation has adjusted the repeating pattern to the maximum extent. However, we could still see the ACF is decaying slowly, hence I have used normal differential transformation on the data.

ACF Plots – Seasonal difference + one difference transformation



PACF Plot:



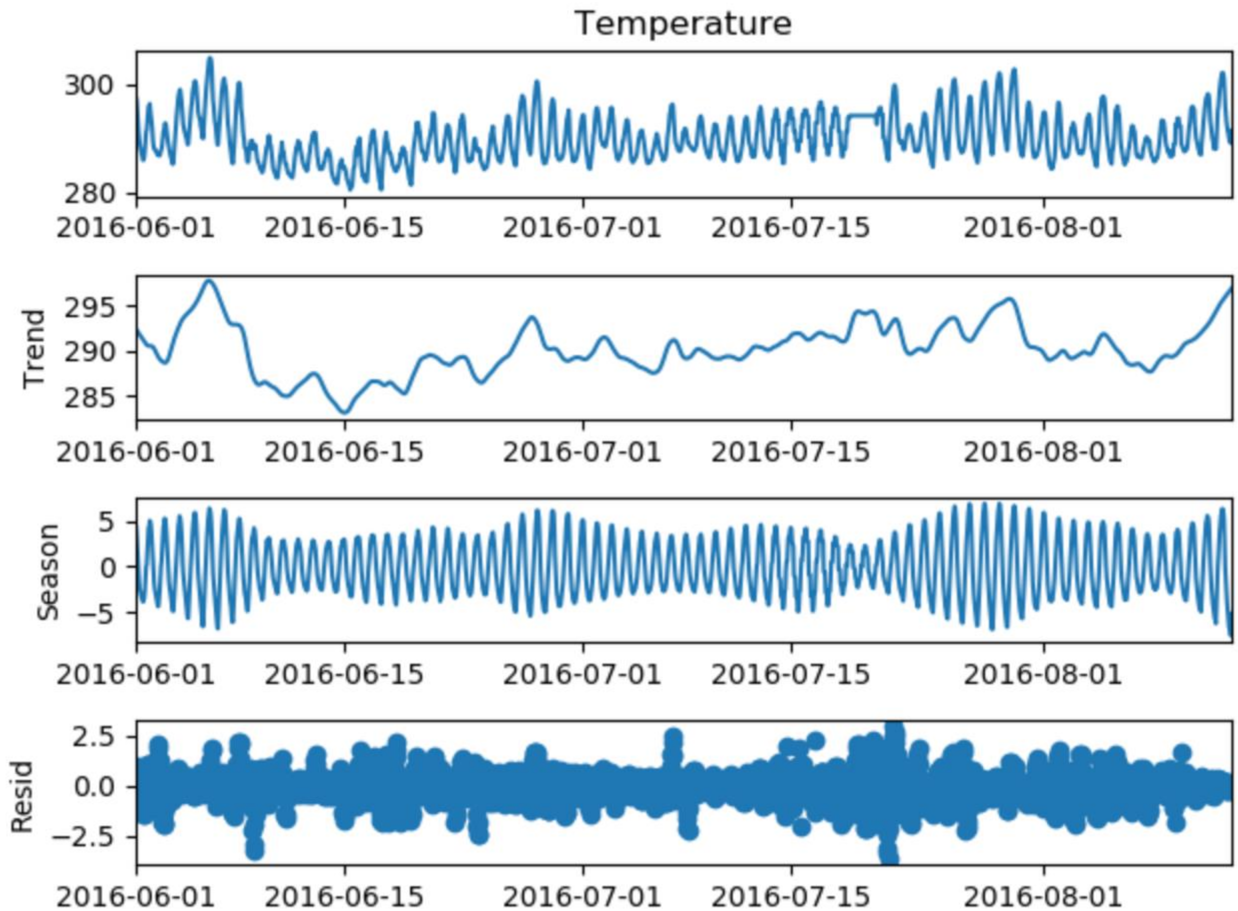
ADF Test Results on Transformed data:

```
ADF Statistics: -13.475400
p value: 0.000000
Critical Values:
     1% : -3.434
     5% : -2.863
     10% : -2.568
```

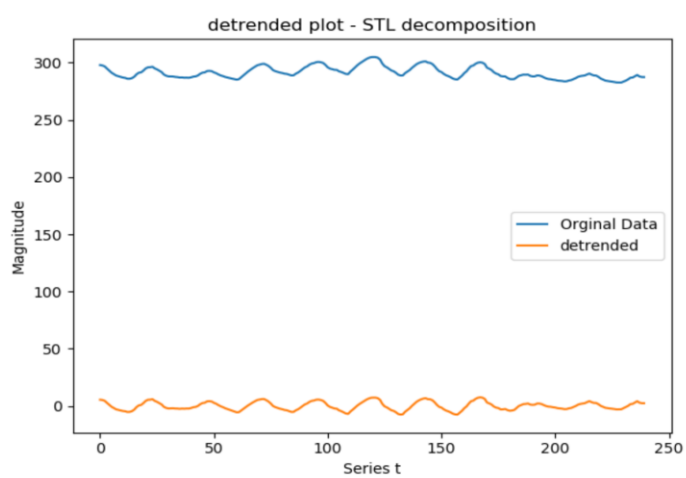
From the above plots and ADF tests we could see data became stationary with p value less then the significance value plus the ADF stats is far lower than 1% CI value suggesting more than 99% confidence interval.

Time Series Decomposition:

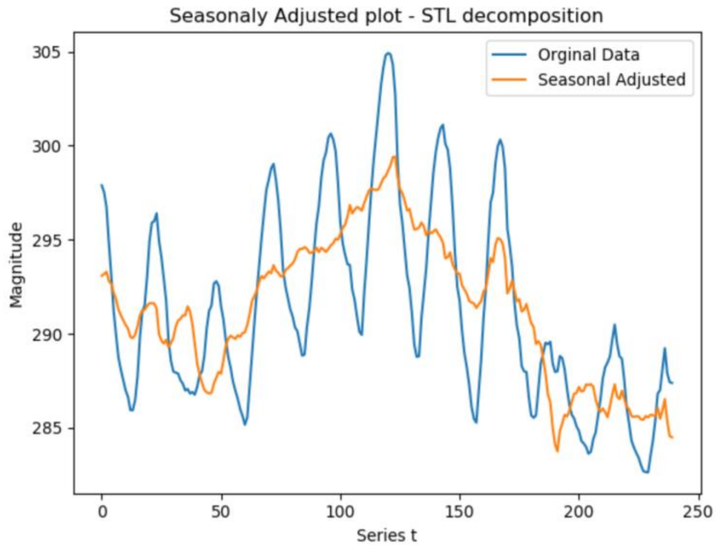
I have used the additive STL decomposition to approximate trend, seasonality from the original dataset.



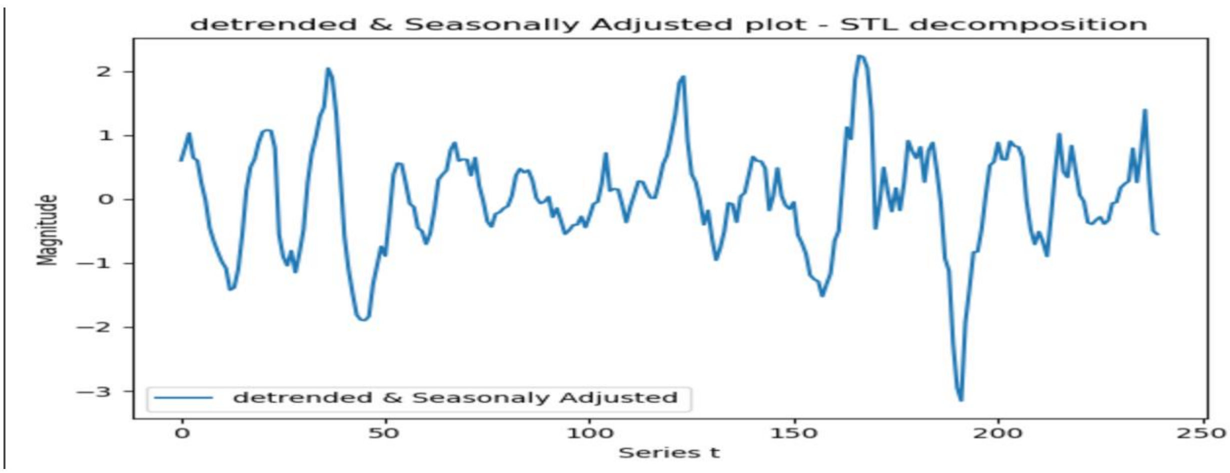
Detrended Plot: [First 10 days of the summer dataset]



Seasonally Adjusted Plot: [First 10 days of the summer dataset]



Detrended & seasonal Adjusted Plot:



From the plots, we could infer that STL decomposition works well for our data and able to capture most of the trends and seasonality present in our data. Variability present in the dataset is captured and removed from the data. Looks like both trend and seasonal components dominate our data, we will confirm the same as below

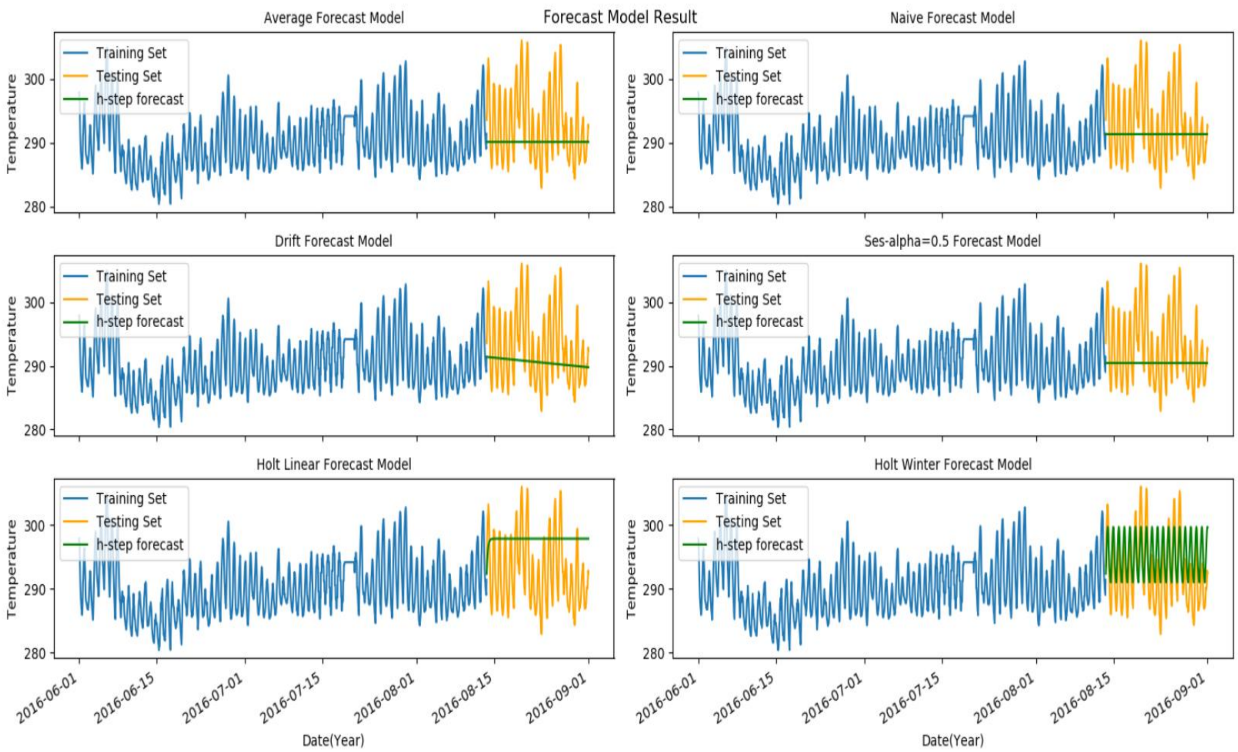
```
The strength of trend for this data set is  0.9301867127909873

The strength of seasonality for this data set is 0.9432048243530327
```

We could see data has both trend and seasonality to the maximum, we could say there are higher chances that data might be nonlinear.

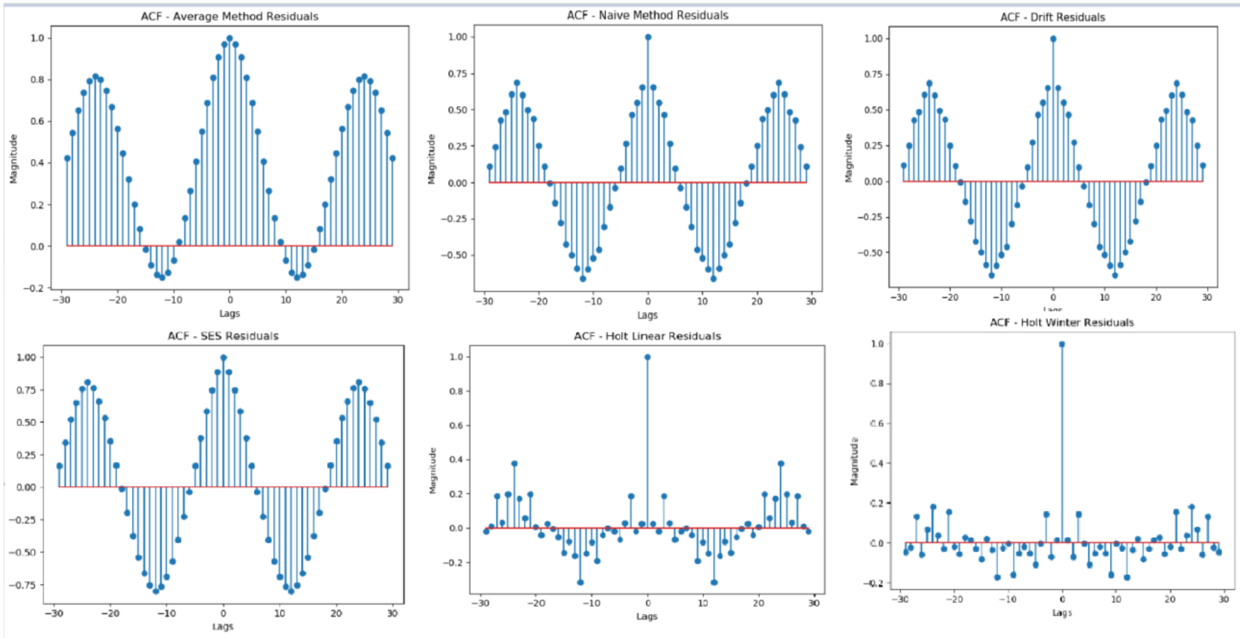
Conventional Basic model:

We have applied basic models like average, naïve, drift, SES, holts linear and holts winter method to our train set and made an h step prediction over the test data. All the basic stats values like MSE, Mean, Variance and Q value is calculated on the prediction error & forecast errors to do the comparative analysis over the model.

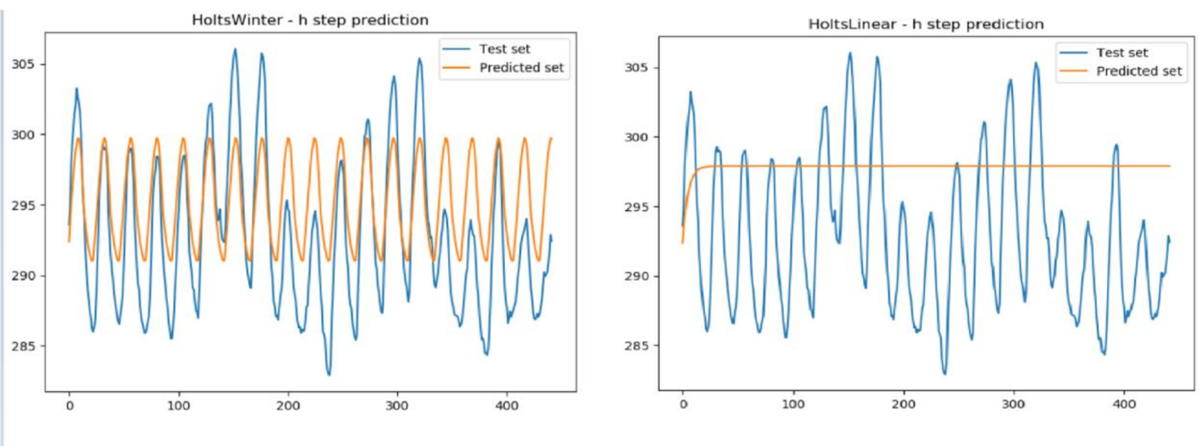
Forecast Model Plots:



From the above model, we could see holts winter method seems to work better for the given data. Let's check the residual plot to confirm the one step prediction results.



ACF plots also reveal that the holt's winter one step prediction is almost equal to the impulse response (i.e.) white noise. Next closest model is holt's linear model. Let's give a closer look to the h step prediction of these models to understand the pattern.



	Average Forecast	Naive	Drift	SES-alpha = 0.5
MSE_pred	18.82	1.07	1.07	2.94
MSE_Forecast	36.68	31.44	33.74	35.09
Mean_pred	0.31	-0.00	0.02	-0.01
Mean_Forecast	2.69	1.42	2.24	2.38
Variance_pred	18.72	1.06	1.07	2.94
Variance_Forecast	29.41	29.41	28.73	29.41
Q Value	15155.44	9744.10	9687.65	16098.93
Correlation coefficient	1.00	1.00	1.00	1.00

	Holts_Linear	Holts_winter
MSE_pred	0.58	0.37
MSE_Forecast	55.24	18.20
Mean_pred	-0.00	0.00
Mean_Forecast	-5.00	-2.36
Variance_pred	0.58	0.37
Variance_Forecast	30.23	12.62
Q Value	984.36	356.36
Correlation coefficient	1.00	0.85

Among base models, MSE of the holt's winter method is lowest and the mean of the prediction error is almost 0 and variance is 0.37. Although it could not be able to capture exact variability, there exists some correlation between the predicted values and actual values. With Q value far less than other models, suggests holts winter method outperforms other basic conventional forecast models.

Multiple Linear Regression Model:

Let's perform multiple regression analysis on the data with dependent variables like humidity, pressure, wind direction, wind speed.

Linear Regression Model with all features:

```
Model Summary
                            OLS Regression Results
==============================================================================
Dep. Variable:            Temperature   R-squared:                       0.359
Model:                            OLS   Adj. R-squared:                  0.359
Method:                 Least Squares   F-statistic:                     1877.
Date:                Thu, 17 Dec 2020   Prob (F-statistic):               0.00
Time:                        01:55:07   Log-Likelihood:                -40812.
No. Observations:               13421   AIC:                         8.163e+04
Df Residuals:                   13416   BIC:                         8.167e+04
Df Model:                           4
Covariance Type:            nonrobust
==================================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------
Intercepts       386.6060      6.115     63.218      0.000     374.619     398.593
Humidity          -0.1985      0.002    -82.267      0.000      -0.203      -0.194
Wind Speed        -0.1319      0.028     -4.795      0.000      -0.186      -0.078
Wind Direction     0.0106      0.000     22.716      0.000       0.010       0.012
Pressure          -0.0868      0.006    -14.548      0.000      -0.098      -0.075
==============================================================================
Omnibus:                     1534.953   Durbin-Watson:                   0.142
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             2416.433
Skew:                          -0.823   Prob(JB):                         0.00
Kurtosis:                       4.269   Cond. No.                     1.44e+05
==============================================================================
```

```
Linear Regression Results
Mean of the Residuals:  2.4132765262666235e-12
Variance of the Residuals:  25.645025486525455
MSE Residuals:  5.823903592229501e-24
Q Value:  254373.16875910715
Mean of the Forecast Error:  2.4354751738285785
Variance of the Forecast Error:  26.955679495909692
MSE Forecast Error:  5.931539322335345
Correlation coefficients predicted value and test set: 0.73
correlation coefficients predicted value and original set: 0.6
```

From the above linear regression model result, R squared value is 0.359 – this model able to capture 36% of the variability present in the data. The p value of all features are less than the significant value 0.05, thus all features are contributing to model results. The MSE, variance, Q value suggesting that model couldn't be able to perform well on both on train and test set. This might be since data is highly non-linear. Hence, this method is not suitable for our dataset.

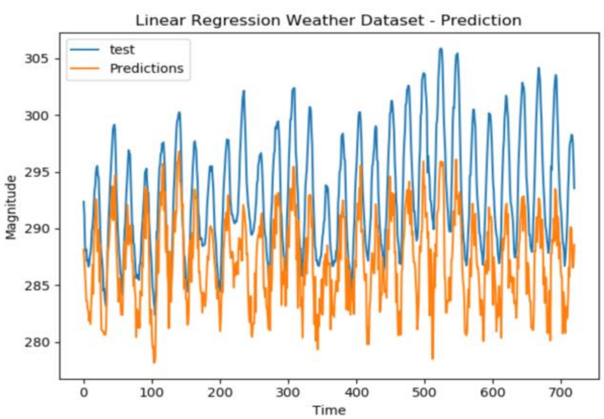
Hypothesis Test Results:

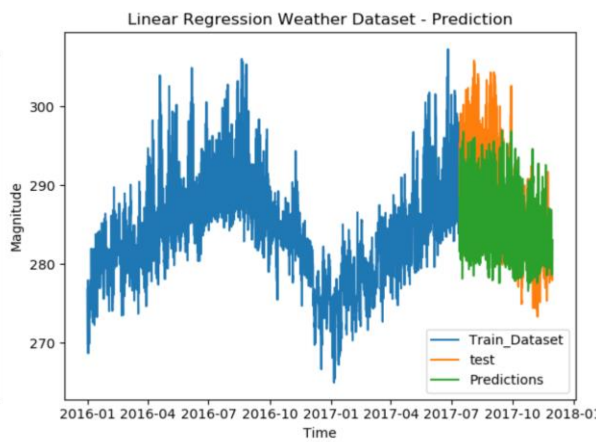
F Test → F statistic value is far less than the significant value of 0.05, hence we could say this model performs better than the intercept only model and reject the null hypothesis.

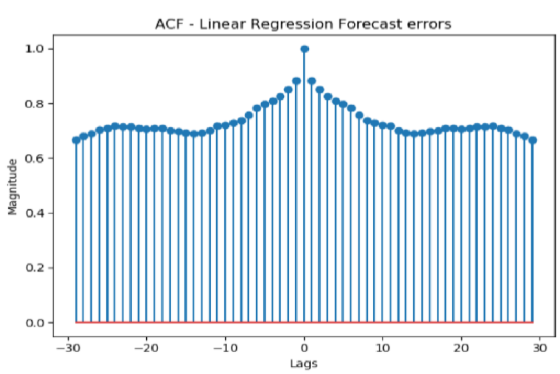
T Test → As mentioned earlier, p value is less than 0.05, rejects the null hypothesis and all variables are significant.

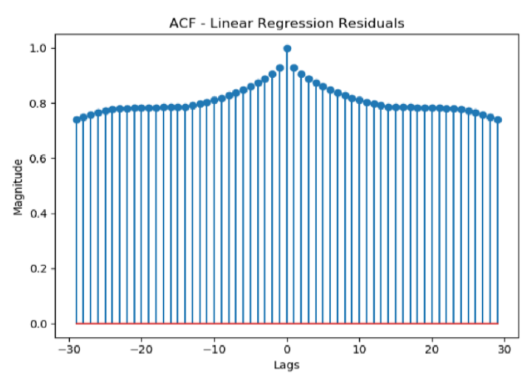
Feature reduction:

Both the correlation matrix and p value suggest values are significant and no multicollinearity exists between them. Thus, reducing the features won't improve the model performance. This full model remains the best model for linear regression approach.









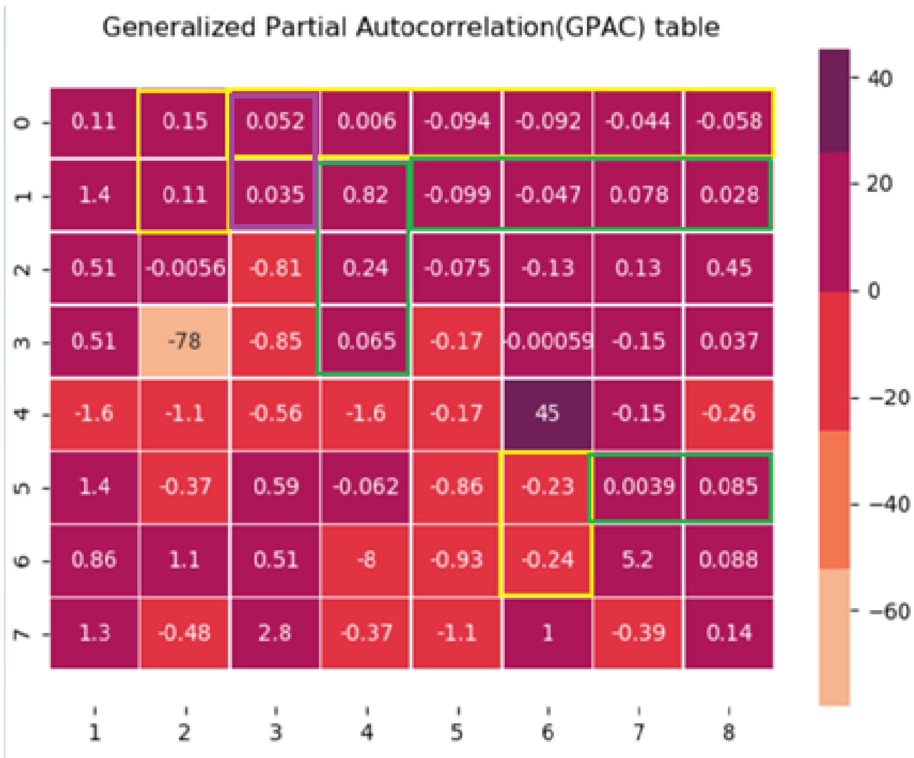
ACF shows that strong linear correlation exists between the residuals lag values, suggesting that the model failed to capture information from the data.

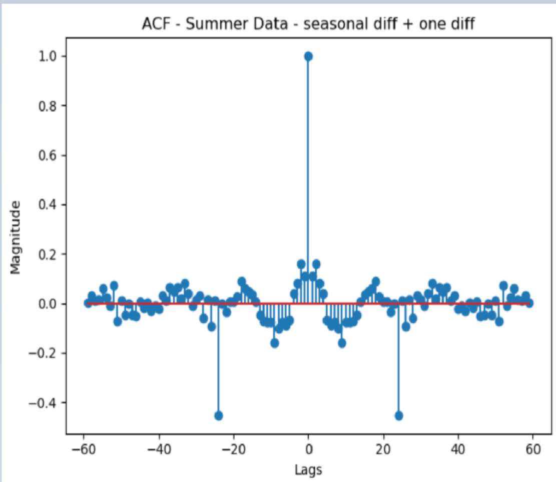
ARMA Model:

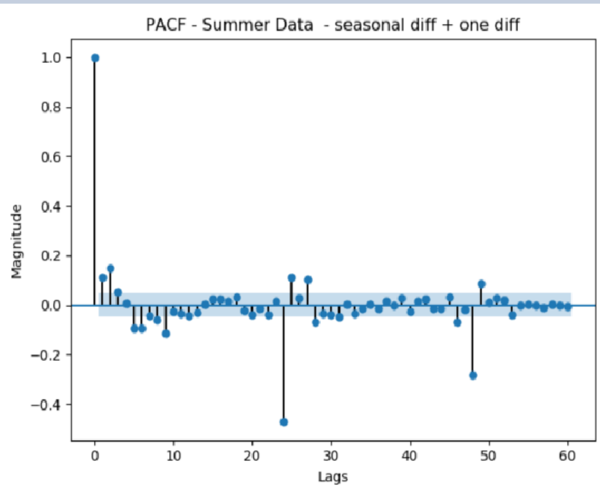
Order Estimation:

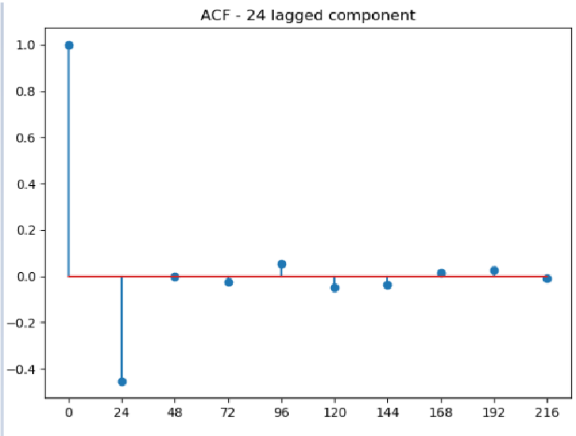
The potential order for the ARMA model can be calculated from the autocorrelation lag behavior present in the data. This checks possible correlation between the values to find best possible correlation value between the y(t) and y(t-h). let's calculate ACF, PACF and GPAC to find the possible order for our data.

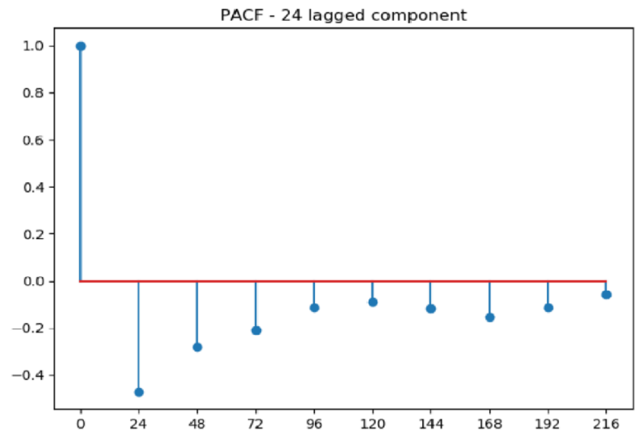
GPAC Table:











From the GPAC, we could see the potential order to be as follows

1. na = 2, nb = 0
2. na = 3, nb = 0
3. na = 4, nb = 1
4. na = 6, nb = 5

PACF plot – two lags have significant value above the blue line – suggesting AR(2) to the model, while ACF plot -> lag value follows the tailing off pattern leaving MA(0) model. Thus, from the ACF & PACF plot – the potential pattern is na = 2 & nb = 0. We were able to see a similar pattern from the GPAC table as well. Let's calculate the estimate for these potential orders and perform ARMA model.

Parameter Estimation:

I have used Lavenberg Marquardt Algorithm to calculate the parameters for the given order.

LM Algorithm Output:

```
LM Algorithm Results
Max Iteration for Convergence:  2

Final Parameters
[-0.09382706 -0.1481415 ]
```

The final parameters for ARMA(2,0) model is given as a1 = -0.0938 and a2 = -0.1481

I have reconfirmed these parameters values from the stats model package ARMA method. The result of the stats model is given as

```
                         - 08-15-2018
=======================================================================================
                        coef    std err          z      P>|z|      [0.025      0.975]
---------------------------------------------------------------------------------------
const             -6.235e-05      0.025     -0.003      0.998      -0.048       0.048
ar.L1.Temperature     0.0938      0.024      3.960      0.000       0.047       0.140
ar.L2.Temperature     0.1480      0.024      6.250      0.000       0.102       0.194
                                    Roots
```

Since Stats model package brings the co-efficient to the right side, the negative sign is neglected. We could see both the results match.

ARMA (2,0) Model is given as

$$\mathbf{Y(t) = 0.0928*y(t-1) + 0.1480 * y(t-2) + e(t)}$$

Diagnostic Analysis on Parameters Estimated

1. Confidence Interval:

```
Confidence Interval
-0.14124783427265453 < a1 < -0.04640628137093169
-0.19556233847916668 < a2 < -0.10072066575308661
```

The a1 and a2 values are well between the confidence interval. They are not passing over the zero value.

2. Zero Pole Cancellation:

The Roots of the numerator and denominator of the shift operator is calculated for given model is

```
The Roots of the numerators are  [ 0.43465362 -0.34082657]
The Roots of the denominators are  []
```

We couldn't find any zero pole cancellation occurring within the roots. They are significant.

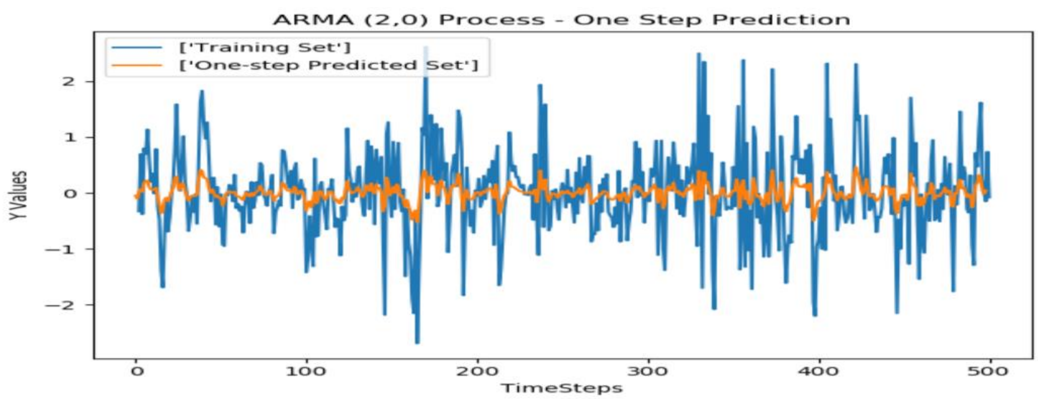
3. Co variance of the Estimated Parameters

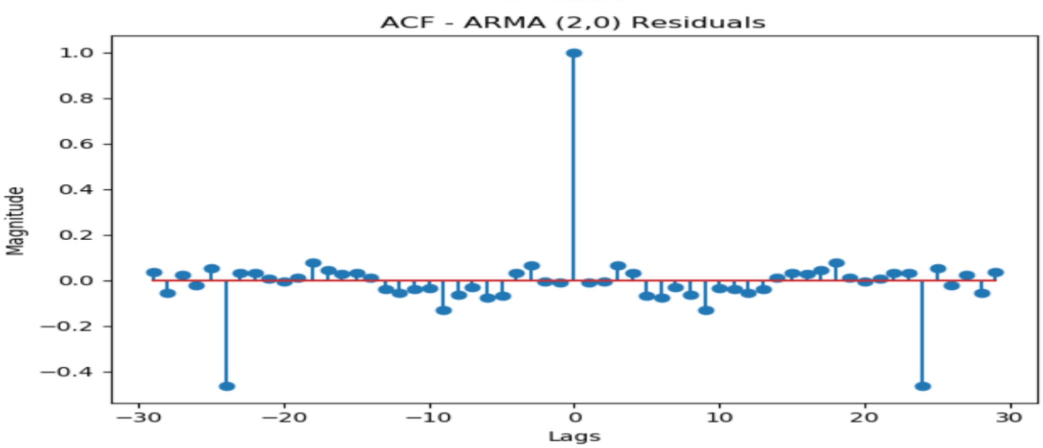
```
array([[ 5.62182510e-04, -6.19334604e-05],
       [-6.19334604e-05,  5.62183930e-04]])
```

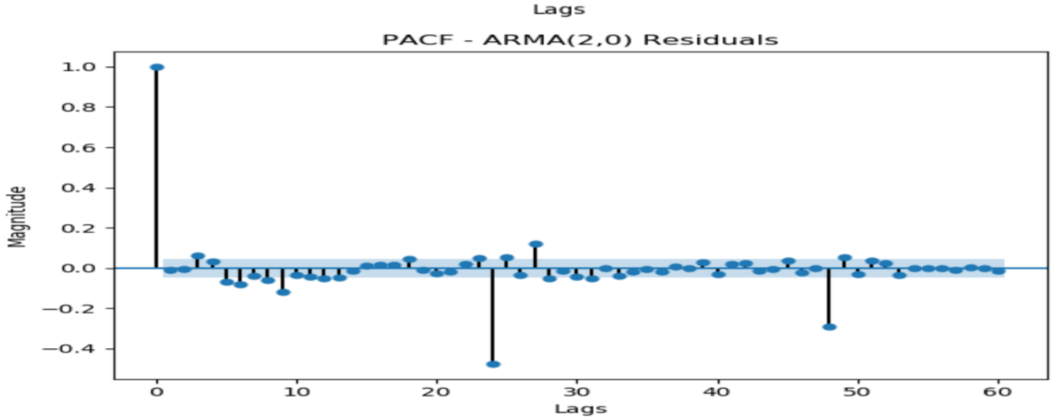
4. One step prediction result:

```
Residual error stats
Mean of the Residual error 4.935565446762119e-05
Variance of the residual error 0.6055519575151731
MSE of the error 0.6055519599511537
```

```
Q Value of the Residuals:  490.45120205594
```







From the above one step prediction stats, we could see the mean and variance is not almost equal to the (0,1), also the Q value is huge. The plots suggest that the model couldn't be predicted properly. Also, the residual plot is not close to white noise as we could see a sharp spike at the interval k = 24,48, etc.

Chi Square test:

From the residual plot, we could say that this model isn't able to capture the entire information present in the data, as residuals do contain some information that are reflected by the large spike in the ACF. And the Q value is also high around 490. Let's compare this value with q_critical and confirm the chi square test results on residual pattern.

```
Q Value of the Residuals:  490.45120205594

Chi-Square Whiteness Test
Q critical value from chi2 table -  41.33713815142739
Residuals are not White
```

Chi square test results suggest that the residual errors are not white. Thus, chi square test failed for this model. This model is not the significant model.

Let's try the same approach for other estimated orders.

ARMA (3,0) Model:

Parameter Estimation:

LM Algorithm Output:

```
LM Algorithm Results
Max Iteration for Convergence:  2

Final Parameters
[-0.08612431 -0.14327064 -0.05192866]
```

The final parameters for ARMA (3,0) model is given as a1 = -0.08612 and a2 = -0.1432 a3 = -0.051

I have reconfirmed these parameters values from the stats model package ARMA method. The result of the stats model is given as

	coef	std err	z	P>|z|	[0.025	0.975]
const	-7.665e-05	0.026	-0.003	0.998	-0.051	0.051
ar.L1.Temperature	0.0861	0.024	3.599	0.000	0.039	0.133
ar.L2.Temperature	0.1432	0.024	6.025	0.000	0.097	0.190
ar.L3.Temperature	0.0518	0.024	2.167	0.030	0.005	0.099

Since Stats model package brings the co-efficient to the right side, the negative sign is neglected. We could see both the results match.

ARMA (3,0) Model is given as

Y(t) = 0.0861*y(t-1) + 0.1432 * y(t-2) + 0.051*y(t-2) + e(t)

Diagnostic Analysis on Parameters Estimated

1. Confidence Interval:

```
Confidence Interval
-0.13402479284288898 < a1 < -0.038223830912148746
-0.19085379213399725 < a2 < -0.0956874780746482
-0.09983536948030486 < a3 < -0.004021960108624288
```

The a1, a2 & a3 values are well between the confidence interval. They are not passing over the zero value.

2. Zero Pole Cancellation:

The Roots of the numerator and denominator of the shift operator is calculated for given model is

```
The Roots of the numerators are  [ 0.5351588 +0.j        -0.22451724+0.21593085j -0.22451724-0.21593085j]
The Roots of the denominators are  []
```

We couldn't find any zero pole cancellation occurring within the roots. They are significant.

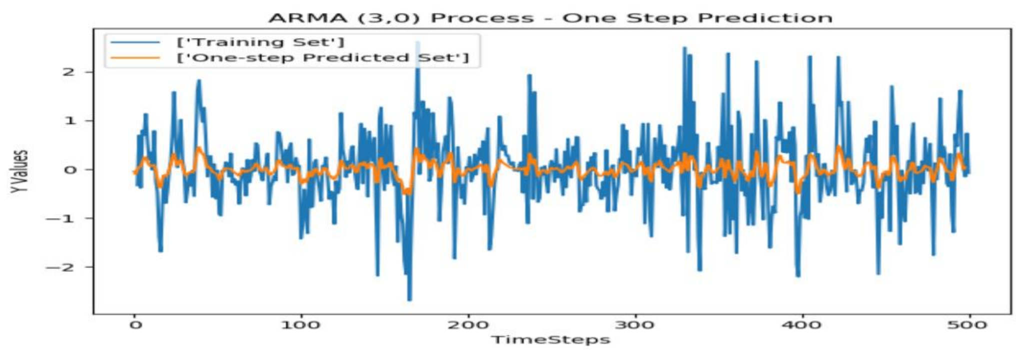
3. Co variance of the Estimated Parameters

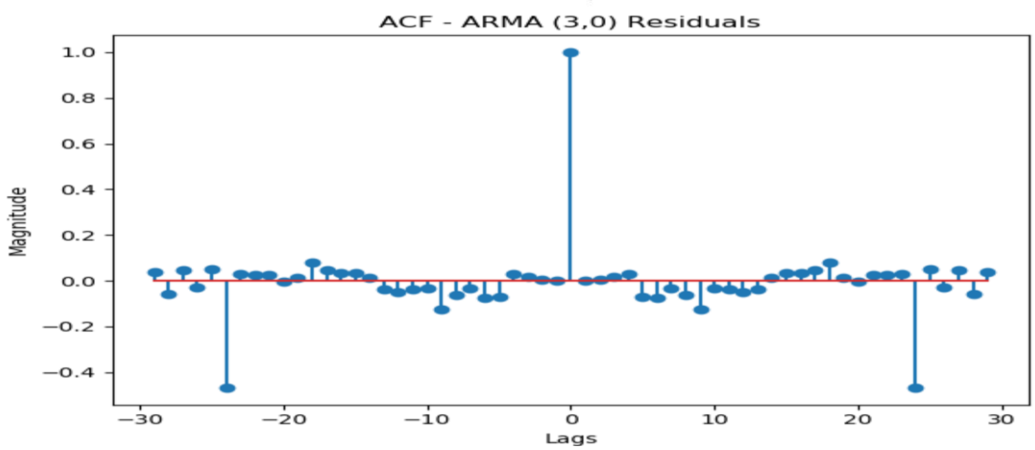
```
The covariance matrix for the estimated parameters :  [[ 5.73614019e-04 -5.38189747e-05 -8.51081252e-05]
 [-5.38189747e-05  5.66039208e-04 -5.38185165e-05]
 [-8.51081252e-05 -5.38185165e-05  5.73763088e-04]]
```

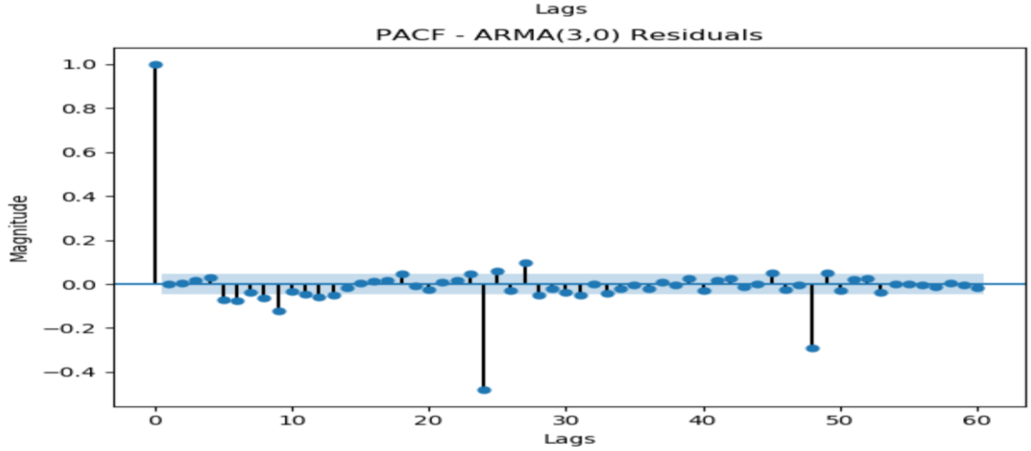
4. One step prediction result:

```
Residual error stats
Mean of the Residual error 4.081647804052109e-05
Variance of the residual error 0.60391980329714
MSE of the error 0.6039198049631249
```

```
Q Value of the Residuals:  491.20120437946156
```







From the above one step prediction stats, we could see the mean and variance is not almost equal to the (0,1), also the Q value is huge. The plots suggest that model couldn't be predicted properly. Also, the residual plot is not close to white noise as we could see a sharp spike at the interval k = 24,48, etc.

Chi Square test:

From the residual plot, we could say that this model isn't able to capture the entire information present in the data, as residuals do contain some information that are reflected by the large spike in the ACF. And the Q value is also high around 491. Let's compare this value with q_critical and confirm the chi square test results on the residual pattern.

```
Q Value of the Residuals:  491.20120437946156

Chi-Square Whiteness Test
Q critical value from chi2 table -  40.113272069413625
Residuals are not White
```

Chi square test results suggest that the residual errors are not white. Thus, chi square test failed for this model. This model is not the significant model.

I have also tried other na,nb values and chi square tests but failed for those models as well. One of the possible reasons for the chi square test failure may be non-linearity present in our data. Also, we have calculated the strength of the trend and seasonality in our dataset which is greater than 90%. Thus, remaining residuals dont constitute much to the data series. From the ACF of the residual values, we could see a sharp spike at lags k =24,48, this suggests there is more information left in the seasonal components of the dataset at the interval of 24. Thus, normal ARMA models do not produce good results with our dataset.

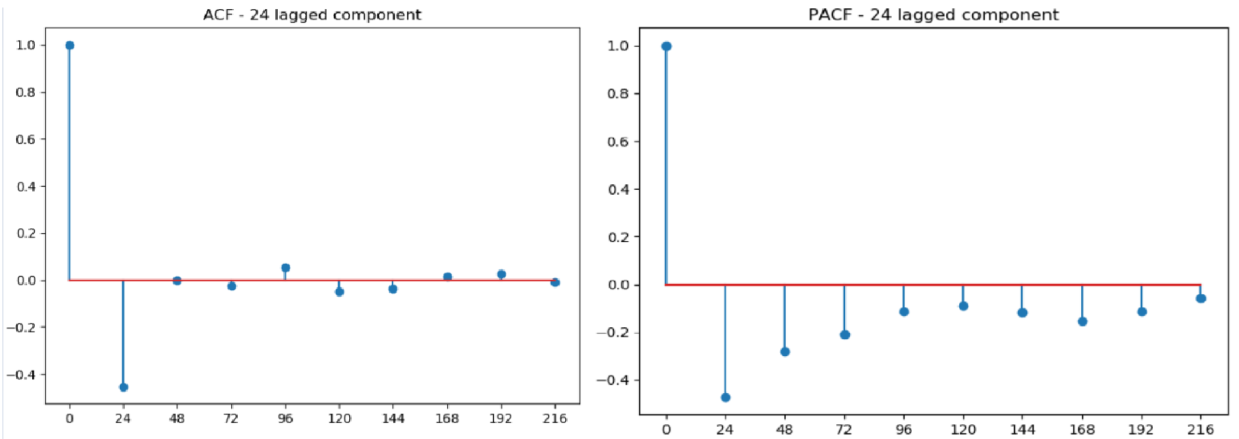
As our data have high seasonality, I planned to experiment with the SARIMA model instead of ARIMA. Since ARIMA model expects the data to be non-seasonal, our earlier research suggests us that more information may be available at seasonal components, so I have skipped the ARIMA part and moved to the the SARIMA.

SARIMA – Seasonal ARIMA:

The SARIMA model takes care of both seasonality and trend present in the data. The input to the model contains both seasonal components and non-seasonal components. And, the integration value in the components takes care of the trend present in the data. One major challenge with SARIMA is to find the order for the seasonal component.

From the earlier analysis, we know that potential non-seasonal components might be (2,0) , (3,0),(4,1), (6,5).

Lets take a closer look at the ACF and PACF values to get seasonal components of the model. To facilitate this, I have graphed the correlation with lags parts k = 24,48, .., etc.



The above plot suggests that ACF is having a sharp spike and cutsoff after that, this gives a MA (1) model while PACF decays through the lags – this constitutes to the AR (0). Hence, potential order for the seasonal components would be (0,1).

Lets built SARIMA model with these values and check for residuals and chi square test values for more information.

Possible model – ARIMA (2,1,0) (0,1,1,24)

SARIMAX function:

```
CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL
                                      SARIMAX Results
============================================================================================
Dep. Variable:                          Temperature   No. Observations:                 1767
Model:             SARIMAX(2, 1, 0)x(0, 1, [1], 24)   Log Likelihood               -1670.918
Date:                              Thu, 17 Dec 2020   AIC                           3349.835
Time:                                      08:28:39   BIC                           3371.686
Sample:                                  06-01-2016   HQIC                          3357.915
                                       - 08-13-2016
Covariance Type:                                opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.1710      0.018      9.537      0.000       0.136       0.206
ar.L2          0.1264      0.019      6.774      0.000       0.090       0.163
ma.S.L24      -0.8355      0.009    -96.505      0.000      -0.852      -0.819
sigma2         0.3922      0.009     44.530      0.000       0.375       0.409
===================================================================================
Ljung-Box (Q):                      290.86   Jarque-Bera (JB):               633.77
Prob(Q):                              0.00   Prob(JB):                         0.00
Heteroskedasticity (H):               1.23   Skew:                             0.06
Prob(H) (two-sided):                  0.01   Kurtosis:                         5.95
===================================================================================
```

From the SARIMA outcome, we could see that Q value is dropped to 290 and all the p values of the coefficients are significant. But the prob of Q values is 0.00 which rejects the chi square test. The one potential reason that these linear methods are failing is due to the extreme seasonality and non-linearity present in the model. Although the Q value is dropped compared to the ARMA model, the MSE values are far greater than other models. Also Variance of 64.43 makes the estimators to be biased.

```
Residual error stats
Mean of the Residual error 0.08705519394616847
Variance of the residual error 63.42347679636747
MSE of the error 63.43105540316048

Q Value of the Residuals:  281.2937205926472

Chi-Square Whiteness Test
Q critical value from chi2 table -  37.65248413348277
Residuals are not White
```

Final Model Selection:

A comparative analysis on the accuracy of the model is performed by comparing the Mean, MSE, Q values from each model.

Base Model Results:

```
                         Average Forecast     Naive     Drift  SES-alpha = 0.5
MSE_pred                            18.82      1.07      1.07             2.94
MSE_Forecast                        36.68     31.44     33.74            35.09
Mean_pred                            0.31     -0.00      0.02            -0.01
Mean_Forecast                        2.69      1.42      2.24             2.38
Variance_pred                       18.72      1.06      1.07             2.94
Variance_Forecast                   29.41     29.41     28.73            29.41
Q Value                          15155.44   9744.10   9687.65         16098.93
Correlation coefficient              1.00      1.00      1.00             1.00
                         Holts_Linear  Holts_winter
MSE_pred                         0.58          0.37
MSE_Forecast                    55.24         18.20
Mean_pred                       -0.00          0.00
Mean_Forecast                   -5.00         -2.36
Variance_pred                    0.58          0.37
Variance_Forecast               30.23         12.62
Q Value                        984.36        356.36
Correlation coefficient          1.00          0.85
```

Linear Regression Model Results:

```
Linear Regression Results
Mean of the Residuals:  2.4132765262666235e-12
Variance of the Residuals:  25.645025486525455
MSE Residuals:  5.823903592229501e-24
Q Value:  254373.16875910715
Mean of the Forecast Error:  2.4354751738285785
Variance of the Forecast Error:  26.955679495909692
MSE Forecast Error:  5.931539322335345
Correlation coefficients predicted value and test set: 0.73
correlation coefficients predicted value and original set: 0.6
```

ARMA Model Results:

	ARMA (2,0)	ARMA (3,0)	SARIMA (2,1,0),(0,1,1)
MSE_pred	0.605552	0.60392	63.4311
Mean_pred	4.93557e-05	4.08165e-05	0.0870552
Variance_pred	0.605552	0.60392	63.4235
Q Value	490.451	491.201	281.294

Considering MSE and variance of the prediction error, we could say Holts winter is working good for the dataset. Also, residual function from the one step prediction is almost equal to white noise pattern. Hence, performance of the holt's winter is best on all the forecast models experimented for the given dataset.

Forecast Function:

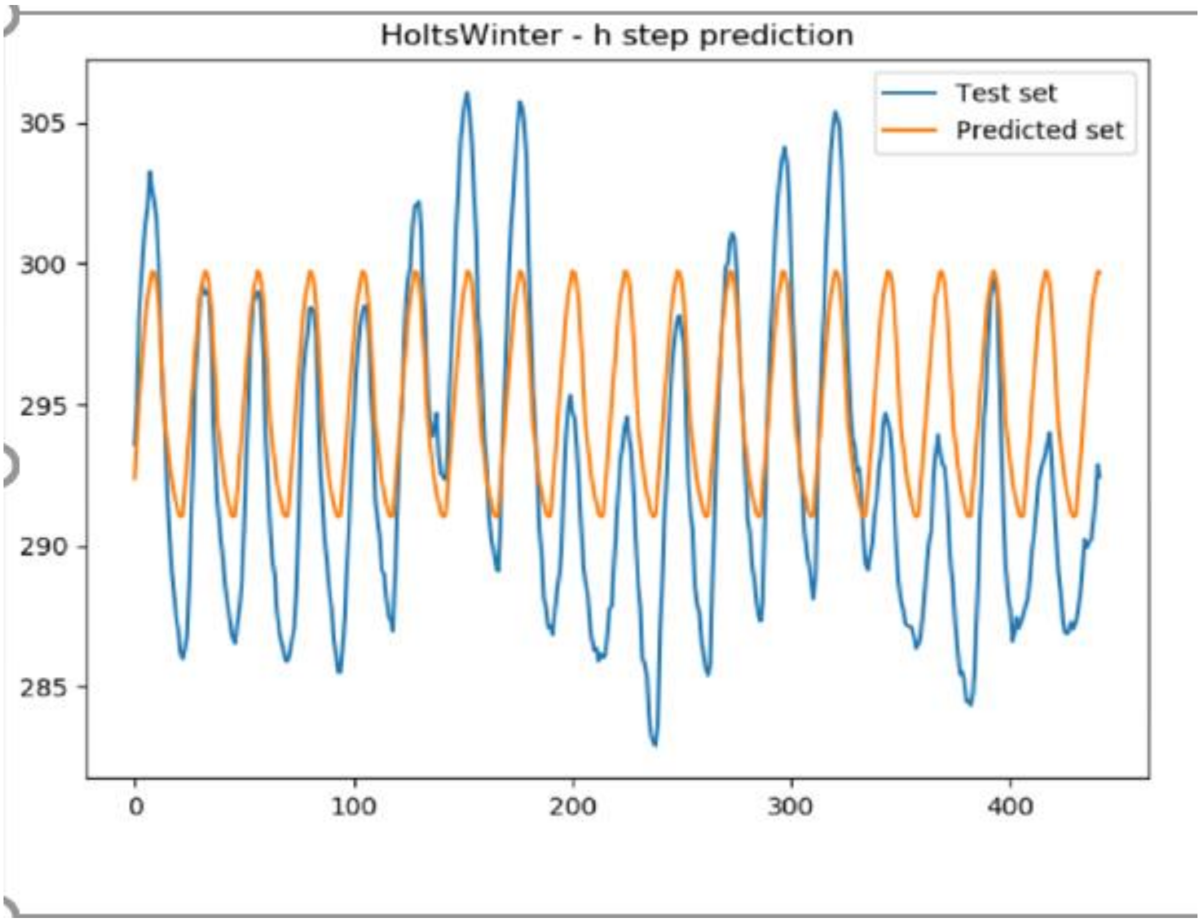
Holts Winter Method:

Below forecast function computes both one step prediction on the train dataset and h step prediction on the test dataset.

```
def holtwinter(train_y, test_y,error,lag,freq,trend,season):
    train_label = train_y.index
    test_label = test_y.index
    train_y_predict = ets.ExponentialSmoothing(train_y,trend=trend,
damped=True, seasonal=season, seasonal_periods=freq).fit()
    test_y_predict = train_y_predict.forecast(len(test_y))
    x_val = np.arange(0,len(test_y))
    plt.figure()
    plt.plot(x_val[:720], test_y[:720], label="Test set")
    plt.plot(x_val[:720], test_y_predict[:720], label="Predicted set")
    plt.title("HoltsWinter - h step prediction")
    plt.legend()
    plt.show()
    forecast_error = np.array(test_y) - np.array(test_y_predict)
    predicted_error = np.array(train_y) -
np.array(train_y_predict.fittedvalues)
    variance_predicted_error = np.var(predicted_error)
    MSE_predicted_error = np.mean(predicted_error ** 2)
    MSE_forecast_error = np.mean(forecast_error ** 2)
    variance_forecast_error = np.var(forecast_error)
    # Correlation Co efficient
    cc = correlation_coefficient_cal(forecast_error, np.array(test_y))
    # Autocorrelation on prediction error
    if error == "Pred_error":
        error_val = predicted_error
    else:
        error_val = forecast_error
    q = ACF(train_y, error_val, lag, "Holt Winter Residuals")
    stats = [round(MSE_predicted_error,2), round(MSE_forecast_error,2),
round(np.mean(predicted_error),2), round(np.mean(forecast_error),2),
             round(variance_predicted_error,2),
```

```
round(variance_forecast_error,2), round(q,2), cc]
    return stats, test_y_predict
```

H step prediction result on the first 500 test samples using Holts winter method is given below,



Summary and Conclusion:

Thus, we could say that Holt winter method fits our problem with least MSE values and variance on the prediction error, making it best among other models. The same procedure must be followed for other seasonal data splits. The major reason for drop in the performance by linear methods like ARMA, ARIMA, SARIMA is because of the non-linearity present in the model along with multiple seasonal patterns and cyclic behavior of the dataset. This makes it difficult for the linear methods. For future scope, we may want to explore other non-linear models like transfer function or neural nets to improve the performances.

Appendix:

```
# importing necessary packages
import pandas as pd
import warnings
import matplotlib.pyplot as plt
import statsmodels.api as sm
from scipy import signal
from scipy.stats import chi2
import numpy as np
from statsmodels.tsa.stattools import pacf
from statsmodels.tsa.stattools import adfuller
from sklearn.model_selection import train_test_split
from statsmodels.graphics.tsaplots import plot_pacf
import seaborn as sns
from statsmodels.tsa.seasonal import STL
import statsmodels.tsa.holtwinters as ets
import math
from statsmodels.tsa.stattools import acf
warnings.filterwarnings("ignore")
# Loading the Seattle dataset
Seattle_data = pd.read_csv("./Historical Hourly Weather Data 2012-2017 -
Seattle.csv", header=0, index_col=[0], parse_dates=[0])
print("First five observations from dataset \n", Seattle_data.head())
#Full dataset
print("Full data")
Seattle_data.info()
Seattle_data = Seattle_data.loc['2016-01-01 00:00:00':]
#Dataset description
print("\nData structure - Seattle Dataset")
Seattle_data.info()


#Data Preprocessing
#Proportion of missing values in each columnsS
col = Seattle_data.columns
missing_values = dict()
for i in range(0,len(col)):
    missing_values[col[i]] =
(Seattle_data[col[i]].isnull().sum()/len(Seattle_data))*100
missing_values = pd.DataFrame.from_dict(missing_values,orient='index')
print(missing_values)
#Dependent Variable Missing Variable Imputation
#As the missing values are less than 0.0001% - Using Naive method - missing
values are imputed with previous value
missing_values_time =
Seattle_data[Seattle_data['Temperature'].isnull()].index.to_list()
pos = [Seattle_data.index.get_loc(i) for i in missing_values_time]
for i in pos:
    new_val = Seattle_data['Temperature'].iloc[i-1]
    Seattle_data['Temperature'].iloc[i] = new_val
print("\nAfter interpolation - Number of missing values in dependent variable
are")
print(Seattle_data["Temperature"].isnull().sum())

#Independent Variable Missing Variable Imputation
#Humidity
missing_value = Seattle_data['Humidity'].isnull().sum()
```

```
while (missing_value != 0):
    missing_values_time =
Seattle_data[Seattle_data['Humidity'].isnull()].index.to_list()
    pos = [Seattle_data.index.get_loc(i) for i in missing_values_time]
    for i in pos:
        new_val = Seattle_data['Humidity'].iloc[i - 1]
        if new_val != "":
            Seattle_data['Humidity'].iloc[i] = new_val
            missing_value = missing_value - 1
print("\nAfter interpolation - Number of missing values in independent
variable - Humidity are")
print(Seattle_data["Humidity"].isnull().sum())

#Pressure
missing_value = Seattle_data['Pressure'].isnull().sum()
while (missing_value != 0):
    missing_values_time =
Seattle_data[Seattle_data['Pressure'].isnull()].index.to_list()
    pos = [Seattle_data.index.get_loc(i) for i in missing_values_time]
    for i in pos:
        new_val = Seattle_data['Pressure'].iloc[i - 1]
        if not np.isnan(new_val):
            Seattle_data['Pressure'].iloc[i] = new_val
            missing_value = missing_value - 1
print("\nAfter interpolation - Number of missing values in independent
variable - Humidity are")
print(Seattle_data["Pressure"].isnull().sum())

# Temperature Pattern over the years
# Plot dependent variable versus time
plt.figure()
plt.plot(Seattle_data["Temperature"])
plt.xlabel("Date Time")
plt.ylabel("Temperature")
plt.title("Hourly Weather - Seattle")
plt.show()

# 7 Day pattern over the time
# This helps to understand the seasonal nature of the data in more detail

sample_day = Seattle_data["Temperature"].loc['2016-07-28 00:00:00':'2016-08-
05 00:00:00']
y_index = np.arange(0,len(sample_day))
plt.figure()
plt.plot(y_index, sample_day)
plt.xlabel("Date Time")
plt.ylabel("Temperature")
plt.title("Hourly Weather - 7 day pattern")
plt.show()


# ***********************************************
# *********** Correlation matrix ******************
# ***********************************************
corr_func_sea = Seattle_data[['Humidity', 'Wind Speed', 'Wind Direction',
'Pressure','Temperature']]
correlation = corr_func_sea.corr()
ax = sns.heatmap(correlation, vmin=-
```

```
1,vmax=1,center=0,cmap=sns.diverging_palette(20,220,n=200),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='righ
t')
ax.set_title("Correlation matrix")
plt.show()

#dependent and independent variable
sea_x = Seattle_data[['Humidity', 'Wind Speed', 'Wind Direction',
'Pressure']]
sea_y = Seattle_data["Temperature"]

# ************************************************
# *********** Seasonal Sub samples ****************
# ************************************************

#Seasonal split
spring_data_y = sea_y.loc['2016-03-01 00:00:00':'2016-06-01 00:00:00']
summer_data_y = sea_y.loc['2016-06-01 00:00:00':'2016-09-01 00:00:00']
fall_data_y = sea_y.loc['2016-09-01 00:00:00':'2016-12-01 00:00:00']
winter_data_y = sea_y.loc['2016-12-01 00:00:00':'2017-03-01 00:00:00']

#Seasonal split
spring_data_x = sea_x.loc['2016-03-01 00:00:00':'2016-06-01 00:00:00']
summer_data_x = sea_x.loc['2016-06-01 00:00:00':'2016-09-01 00:00:00']
fall_data_x = sea_x.loc['2016-09-01 00:00:00':'2016-12-01 00:00:00']
winter_data_x = sea_x.loc['2016-12-01 00:00:00':'2017-03-01 00:00:00']

# ************************************************
# *********** Train and Test Split ****************
# ************************************************
#Linear Regression Train & Test Split
#Adding an Intercept Term
Intercept_term = pd.DataFrame(np.ones((summer_data_x.shape[0],1)),
columns=['Intercepts'], index=summer_data_x.index)
summer_data_x = pd.concat([Intercept_term,summer_data_x],axis=1,sort=False)
X_train, X_test, Y_train, Y_test =
train_test_split(summer_data_x,summer_data_y,shuffle=False,test_size=0.2)

#Other Model Split
Y_train_spr, Y_test_spr =
train_test_split(spring_data_y,shuffle=False,test_size=0.2)
Y_train_sum, Y_test_sum =
train_test_split(summer_data_y,shuffle=False,test_size=0.2)
Y_train_fal, Y_test_fal =
train_test_split(fall_data_y,shuffle=False,test_size=0.2)
Y_train_win, Y_test_win =
train_test_split(winter_data_y,shuffle=False,test_size=0.2)


# ************************************************
# *********** Stationaity Checks ******************
# ************************************************

def ACF_func(sample,lags,label):
    sample = np.array(sample)
    mean_sam = np.mean(sample)
```

```
    y_mean = sample - mean_sam
    T = [(np.sum([y_mean[t + 1] * y_mean[t + 1 - k] for t in range(k - 1,
len(sample) - 1)]) / (np.sum(y_mean ** 2))) for k in range(0, lags)]
    #print("ACF Values are \n",T)
    #As ACF is symmetric function t[k] = t[-k]
    T_new = np.append(np.flip(T), T[1:])
    plt.figure()
    plt.stem(np.arange(-(lags - 1), lags), T_new, use_line_collection=True)
    plt.xlabel("Lags")
    plt.ylabel("Magnitude")
    plt.title("ACF - {}".format(label))
    plt.show()
    return T
def pcf_plot(x,lags,label):
    sm.graphics.tsa.plot_pacf(x, lags=lags, title="PACF - {}".format(label))
    plt.xlabel("Lags")
    plt.ylabel("Magnitude")
    plt.show()

#ACF & PCF  plots
ACT_t = ACF_func(Y_train_sum,30,"Seattle Hourly Weather Summer Data")
pcf_plot(Y_train_sum,30, "Seattle Hourly Summer Weather Data")

#ADF Test Results
def ADF_test(x):
    test_res = adfuller(x)
    print("ADF Statistics: %f" %test_res[0])
    print("p value: %f" % test_res[1])
    print("Critical Values:")
    for CI,value in test_res[4].items():
        print("\t %s : %0.3f" %(CI,value))
print("\nADF Test Results")
print("\nHourly Weather Analysis - Summer Data")
ADF_test(Y_train_sum)

# *****************************************************
# *********** Seasonal Differencing ******************
# *****************************************************
# 24 differencing
sum_diff24 = Y_train_sum.diff(periods=24)
ACT_t = ACF_func(sum_diff24[24:],60,"Summer Data - seasonal diff 24")
pcf_plot(sum_diff24[24:],60, "Summer Data  - seasonal diff 24")

#24 + 1 differencing
sum_diff_24_1 = sum_diff24[24:].diff()
ACT_t = ACF_func(sum_diff_24_1[1:],60,"Summer Data - seasonal diff + one
diff")
pcf_plot(sum_diff_24_1[1:],60, "Summer Data  - seasonal diff + one diff")

print("\nHourly Weather Analysis - Summer Data - Difference Transformed")
ADF_test(sum_diff_24_1[1:])

# For calculating the seasonal components order - ACF & PACF with larger lags
is prefered



def STL_decomposition(y,xlab,y_lab):
```

```
    #STL_1 = STL(y.iloc[:,0])
    STL_1 = STL(y)
    result = STL_1.fit()
    plt.figure()
    result.plot()
    plt.show()
    T = result.trend
    S = result.seasonal
    R = result.resid
    plt.figure()
    x_ax = np.arange(0,len(y))
    plt.plot(x_ax[:240],T[:240],label="Trend")
    plt.plot(x_ax[:240], S[:240], label="Seasonality")
    plt.plot(x_ax[:240], R[:240], label="residuals")
    plt.title("STL decomposition Plot comparison -
Trend,seasonality,Residual")
    plt.xlabel(xlab)
    plt.ylabel(y_lab)
    plt.legend()
    plt.show()
    return T,S,R

T,S,R = STL_decomposition(Y_train_sum,"Series value t","Magnitude")


#Seasonal Adjusted data
Adjusted_seasonal = Y_train_sum - S
x_ax = np.arange(0,len(Y_train_sum))
plt.figure()
plt.plot(x_ax[:240], Y_train_sum[:240], label="Orginal Data")
plt.plot(x_ax[:240], Adjusted_seasonal[:240], label="Seasonal Adjusted")
plt.title("Seasonaly Adjusted plot - STL decomposition")
plt.xlabel("Series t")
plt.ylabel("Magnitude")
plt.legend()
plt.show()

#Detrended data
Adjusted_detrended = Y_train_sum - T
plt.figure()
plt.plot(x_ax[:240], Y_train_sum[:240], label="Orginal Data")
plt.plot(x_ax[:240], Adjusted_detrended[:240], label="detrended")
plt.title("detrended plot - STL decomposition")
plt.xlabel("Series t")
plt.ylabel("Magnitude")
plt.legend()
plt.show()

#Detrended data
detrended_seasonaly_Adj = Y_train_sum - T - S
plt.figure()
#plt.plot(Y_train, label="Orginal Data")
plt.plot(x_ax[:240], detrended_seasonaly_Adj[:240], label="detrended &
Seasonaly Adjusted")
plt.title("detrended & Seasonally Adjusted plot - STL decomposition")
plt.xlabel("Series t")
plt.ylabel("Magnitude")
plt.legend()
```

```
plt.show()

#Strength of the trend
trend_streng = np.max([0, 1 - (np.var(R)/np.var(T+R))])
print("\n The strength of trend for this data set is ",trend_streng)

#Strength of the Seasonality
seasonal_streng = np.max([0, 1 - ( np.var(R)/np.var(S+R))])
print("\n The strength of seasonality for this data set is",seasonal_streng)



#Helper functions
#Q Value
def ACF(y,error,lag,error_dat):
    ACF_Residuals = ACF_func(error,lag,error_dat)
    q = len(y) * np.sum(np.array(ACF_Residuals[1:]) ** 2)
    return q

#Correlation Coefficient
def correlation_coefficient_cal(X,Y):
    mean_x = np.mean(X)
    mean_y = np.mean(Y)
    sq_var_x = []
    sq_var_y = []
    for i in range(0,len(X)):
        sq_var_x.append(X[i]-mean_x)
        sq_var_y.append(Y[i] - mean_y)
    cov_xy = sum([ x*y for x,y in zip(sq_var_x,sq_var_y)])
    cov_x = round(np.sqrt(sum(np.square(sq_var_x))),2)
    cov_y = round(np.sqrt(sum(np.square(sq_var_y))),2)
    r = cov_xy/(cov_x * cov_y)
    return round(r,2)

#HoltsWinter
def holtwinter(train_y, test_y,error,lag,freq,trend,season):
    train_label = train_y.index
    test_label = test_y.index
    train_y_predict = ets.ExponentialSmoothing(train_y,trend=trend,
damped=True, seasonal=season, seasonal_periods=freq).fit()
    test_y_predict = train_y_predict.forecast(len(test_y))
    x_val = np.arange(0,len(test_y))
    plt.figure()
    plt.plot(x_val[:720], test_y[:720], label="Test set")
    plt.plot(x_val[:720], test_y_predict[:720], label="Predicted set")
    plt.title("HoltsWinter - h step prediction")
    plt.legend()
    plt.show()
    forecast_error = np.array(test_y) - np.array(test_y_predict)
    predicted_error = np.array(train_y) -
np.array(train_y_predict.fittedvalues)
    variance_predicted_error = np.var(predicted_error)
    MSE_predicted_error = np.mean(predicted_error ** 2)
    MSE_forecast_error = np.mean(forecast_error ** 2)
    variance_forecast_error = np.var(forecast_error)
    # Correlation Co efficient
    cc = correlation_coefficient_cal(forecast_error, np.array(test_y))
    # Autocorrelation on prediction error
```

```
    if error == "Pred_error":
        error_val = predicted_error
    else:
        error_val = forecast_error
    q = ACF(train_y, error_val, lag, "Holt Winter Residuals")
    stats = [round(MSE_predicted_error,2), round(MSE_forecast_error,2),
round(np.mean(predicted_error),2), round(np.mean(forecast_error),2),
             round(variance_predicted_error,2),
round(variance_forecast_error,2), round(q,2), cc]
    return stats, test_y_predict

# Average Method
def average_forecast(train_y, test_y,error,lag):
    train_y = np.array(train_y)
    test_y = np.array(test_y)
    train_y_predict = [np.mean(train_y[:i]) for i in range(1, len(train_y))]
    test_y_predict = [np.mean(train_y[:len(train_y)]) for i in
range(len(test_y))]
    predicted_error = np.array(train_y[1:]) - np.array(train_y_predict)
    forecast_error = np.array(test_y) - np.array(test_y_predict)
    MSE_predicted_error = np.mean(predicted_error ** 2)
    MSE_forecast_error = np.mean(forecast_error ** 2)
    variance_predicted_error = np.var(predicted_error)
    variance_forecast_error = np.var(forecast_error)
    cc = correlation_coefficient_cal(forecast_error,test_y)
    #Autocorrelation on prediction error
    if error == "Pred_error":
        error_val = predicted_error
    else:
        error_val = forecast_error
    q = ACF(train_y,error_val,lag,"Average Method Residuals")
    stats = [round(MSE_predicted_error,2), round(MSE_forecast_error,2),
round(np.mean(predicted_error),2), round(np.mean(forecast_error),2),
round(variance_predicted_error,2), round(variance_forecast_error,2),
round(q,2), cc]
    return stats, test_y_predict

# Naive Method
def naive_forecast(train_y, test_y,error,lag):
    train_y = np.array(train_y)
    test_y = np.array(test_y)
    train_y_predict = train_y[0:len(train_y)-1]
    test_y_predict = [train_y[len(train_y)-1] for i in range(len(test_y))]
    predicted_error = np.array(train_y[1:]) - np.array(train_y_predict)
    forecast_error = np.array(test_y) - np.array(test_y_predict)
    MSE_predicted_error = np.mean(predicted_error ** 2)
    MSE_forecast_error = np.mean(forecast_error ** 2)
    variance_predicted_error = np.var(predicted_error)
    variance_forecast_error = np.var(forecast_error)
    #Correlation Co efficient
    cc = correlation_coefficient_cal(forecast_error, test_y)
    # Autocorrelation on prediction error
    if error == "Pred_error":
        error_val = predicted_error
    else:
        error_val = forecast_error
    q = ACF(train_y,error_val, lag, "Naive Method Residuals")
    stats = [round(MSE_predicted_error,2), round(MSE_forecast_error,2),
```

```
round(np.mean(predicted_error),2), round(np.mean(forecast_error),2),
             round(variance_predicted_error,2),
round(variance_forecast_error,2), round(q,2), cc]
    return stats, test_y_predict

#Drift Method
def drift_forecast(train_y, test_y,error,lag):
    train_y = np.array(train_y)
    test_y = np.array(test_y)
    train_y_predict = []
    train_y_predict.extend([train_y[0]])
    train_y_predict.extend([train_y[i] + ((train_y[i]-train_y[0])/i) for i in
range(1, len(train_y)-1)])
    test_y_predict = [train_y[len(train_y)-1] + (i*((train_y[len(train_y)-1]
- train_y[0])/(len(train_y)-1))) for i in range(1, len(test_y)+1)]
    predicted_error = np.array(train_y[1:]) - np.array(train_y_predict)
    forecast_error = np.array(test_y) - np.array(test_y_predict)
    MSE_predicted_error = np.mean(predicted_error ** 2)
    MSE_forecast_error = np.mean(forecast_error ** 2)
    variance_predicted_error = np.var(predicted_error)
    variance_forecast_error = np.var(forecast_error)
    # Correlation Co efficient
    cc = correlation_coefficient_cal(forecast_error, test_y)
    # Autocorrelation on prediction error
    if error == "Pred_error":
        error_val = predicted_error
    else:
        error_val = forecast_error
    q = ACF(train_y,error_val, lag, "Drift Residuals")
    stats = [round(MSE_predicted_error,2), round(MSE_forecast_error,2),
round(np.mean(predicted_error),2), round(np.mean(forecast_error),2),
             round(variance_predicted_error,2),
round(variance_forecast_error,2), round(q,2), cc]
    return stats, test_y_predict

#SES Method
def ses_forecast(train_y, test_y, a, error,lag):
    train_y = np.array(train_y)
    test_y = np.array(test_y)
    train_y_predict = []
    train_y_predict.extend([train_y[0]])
    for i in range(0, len(train_y)-1):
        train_y_predict.extend([((a * train_y[i]) + ((1 - a) *
train_y_predict[i]))])
    test_y_predict = [ ((a*train_y[len(train_y)-1]) + ((1-
a)*(train_y_predict[len(train_y_predict)-1]))) for i in range(0,
len(test_y))]
    predicted_error = np.array(train_y) - np.array(train_y_predict)
    forecast_error = np.array(test_y) - np.array(test_y_predict)
    MSE_predicted_error = np.mean(predicted_error[1:] ** 2)
    MSE_forecast_error = np.mean(forecast_error ** 2)
    variance_predicted_error = np.var(predicted_error[1:])
    variance_forecast_error = np.var(forecast_error)
    # Correlation Co efficient
    cc = correlation_coefficient_cal(forecast_error, test_y)
    # Autocorrelation on prediction error
    if error == "Pred_error":
        error_val = predicted_error[1:]
```

```python
    else:
        error_val = forecast_error
    q = ACF(train_y,error_val, lag, "SES Residuals")
    stats = [round(MSE_predicted_error,2), round(MSE_forecast_error,2),
round(np.mean(predicted_error[1:]),2), round(np.mean(forecast_error),2),
             round(variance_predicted_error,2),
round(variance_forecast_error,2), round(q,2), cc]
    return stats, test_y_predict

def holtlinear(train_y, test_y, error,lag,freq, trend):
    train_y_predict = ets.ExponentialSmoothing(train_y,trend=trend,
damped=True,seasonal=None,seasonal_periods=freq).fit()
    test_y_predict = train_y_predict.forecast(steps=len(test_y))
    forecast_error = np.array(test_y) - np.array(test_y_predict)
    predicted_error = np.array(train_y) -
np.array(train_y_predict.fittedvalues)
    variance_predicted_error = np.var(predicted_error)
    MSE_predicted_error = np.mean(predicted_error ** 2)
    MSE_forecast_error = np.mean(forecast_error ** 2)
    variance_forecast_error = np.var(forecast_error)
    x_val = np.arange(0, len(test_y))
    plt.figure()
    plt.plot(x_val[:720], test_y[:720], label="Test set")
    plt.plot(x_val[:720], test_y_predict[:720], label="Predicted set")
    plt.title("HoltsLinear - h step prediction")
    plt.legend()
    plt.show()
    # Correlation Co efficient
    cc = correlation_coefficient_cal(forecast_error, np.array(test_y))
    # Autocorrelation on prediction error
    if error == "Pred_error":
        error_val = predicted_error
    else:
        error_val = forecast_error
    q = ACF(train_y, error_val, lag, "Holt Linear Residuals")
    stats = [round(MSE_predicted_error,2), round(MSE_forecast_error,2),
round(np.mean(predicted_error),2), round(np.mean(forecast_error),2),
             round(variance_predicted_error,2),
round(variance_forecast_error,2), round(q,2), cc]
    return stats, test_y_predict

#Plotting Forecast Model - Function
def sub_plt(rows, cols, title, sub_title, y_train, y_test, forecast_val,
x_label, y_label, freq):
    fig, axes = plt.subplots(nrows=rows, ncols=cols, constrained_layout=True,
figsize=(15,7))
    fig.subplots_adjust(hspace=0.8, wspace=0.5, top=0.85)
    fig.suptitle(title)
    for ax, y, sub_tit in zip(axes.flatten(), forecast_val, sub_title):
        ax.plot(y_train.index, y_train, label="Training Set")
        ax.plot(y_test.index, y_test, color='orange', label="Testing Set")
        ax.plot(y_test.index, y, color='green', label="h-step forecast")
        ax.set(title=sub_tit, xlabel=x_label, ylabel=y_label)
        ax.xaxis.label.set_size(10)
        ax.title.set_size(10)
        ax.legend(fontsize=10)
        fig.autofmt_xdate()
        if freq != 1:
```

```
            ax.get_xticklabels()
            ax.set_xticks(ax.get_xticks()[::freq])
    fig.tight_layout()
    plt.show()

#Base Model Evaluation
Forecast_Model_Sea = pd.DataFrame(columns=['Average Forecast', 'Naive',
'Drift', 'SES-alpha = 0.5','Holts_Linear','Holts_winter'])
Forecast_Model_Sea['Average Forecast'], forecast_val_avg =
average_forecast(Y_train_sum, Y_test_sum, "Pred_error",30)
Forecast_Model_Sea['Naive'], forecast_val_naive = naive_forecast(Y_train_sum,
Y_test_sum, "Pred_error",30)
Forecast_Model_Sea['Drift'],forecast_val_drift = drift_forecast(Y_train_sum,
Y_test_sum, "Pred_error",30)
Forecast_Model_Sea['SES-alpha = 0.5'], forecast_val_ses =
ses_forecast(Y_train_sum, Y_test_sum, 0.5, "Pred_error",30)
Forecast_Model_Sea['Holts_Linear'],forecast_val_holtlin =
holtlinear(Y_train_sum, Y_test_sum, "Pred_error",30,24,'add')
Forecast_Model_Sea['Holts_winter'],forecast_val_holtwin =
holtwinter(Y_train_sum, Y_test_sum, "Pred_error",30,24,'add','mul')
Forecast_Model_Sea =
Forecast_Model_Sea.set_index(pd.Series(['MSE_pred','MSE_Forecast','Mean_pred'
,'Mean_Forecast','Variance_pred','Variance_Forecast','Q Value', 'Correlation
coefficient']))

forecast_val = [list(forecast_val_avg), list(forecast_val_naive),
list(forecast_val_drift), list(forecast_val_ses), list(forecast_val_holtlin),
list(forecast_val_holtwin)]
sub_title = ['Average Forecast Model', 'Naive Forecast Model', 'Drift
Forecast Model', 'Ses-alpha=0.5 Forecast Model', 'Holt Linear Forecast
Model', 'Holt Winter Forecast Model']

sub_plt(3,2,"Forecast Model Result", sub_title, Y_train_sum, Y_test_sum,
forecast_val, 'Date(Year)', 'Temperature', 1 )

print(Forecast_Model_Sea[['Average Forecast', 'Naive', 'Drift', 'SES-alpha =
0.5']])
print(Forecast_Model_Sea[['Holts_Linear','Holts_winter']])



#Linear Regression Train & Test Split
#Adding an Intercept Term
Intercept_term = pd.DataFrame(np.ones((sea_x.shape[0],1)),
columns=['Intercepts'], index=sea_x.index)
sea_x = pd.concat([Intercept_term,sea_x],axis=1,sort=False)
X_train, X_test, Y_train, Y_test =
train_test_split(sea_x,sea_y,shuffle=False,test_size=0.2)

# *************************************************************
# *********** Linear Regression ********************************
# *************************************************************
model = sm.OLS(Y_train,X_train).fit()
print("\nModel Summary")
print(model.summary())

# ***************************************************************************
# *********** Linear Regression Prediction & Plot **************************
```

```
# ****************************************************************************
predicts = model.predict(X_test)
plt.figure()
plt.title("Linear Regression Weather Dataset - Prediction")
plt.plot(Y_train, label="Train_Dataset")
plt.plot(Y_test, label="test")
plt.plot(predicts,label="Predictions")
plt.xlabel("Time")
plt.ylabel("Magnitude")
plt.legend()
plt.show()

# ****************************************************************************
# ************* Forecast Errors & ACF ****************************************
# ****************************************************************************
forecast_error = Y_test - predicts
ACT_t = ACF_func(forecast_error, 30, "Linear Regression Forecast errors")

af = correlation_coefficient_cal(Y_test,predicts)

# ****************************************************************************
# ************* Estimated Variance - Forecast Errors *************************
# ****************************************************************************
T = X_test.shape[0]
K = X_test.shape[1]-1
const = (1 / (T-K-1))
SSE = np.sum(np.square(forecast_error))
variance_forec = const*SSE
SD_forec = np.sqrt(variance_forec)
print("Variance of the Forecast error is given as",variance_forec)
print("SD of the Forecast error is given as",SD_forec)

# ****************************************************************************
# ************* Predicted Errors & ACF
***************************************
# ****************************************************************************
pred_values = model.predict(X_train)
predict_error = Y_train - pred_values
ACT_t = ACF_func(predict_error, 30, "Linear Regression Residuals")

a = correlation_coefficient_cal(Y_train,pred_values)
print("\ncorrelation coefficients predicted value and original set:", a)

# ****************************************************************************
# ************* Estimated Variance - Residuals ************************
# ****************************************************************************
T = X_train.shape[0]
K = X_train.shape[1]-1
const = (1 / (T-K-1))
SSE = np.sum(np.square(predict_error))
variance_pred = const*SSE
SD_pred = np.sqrt(variance_pred)
print("Variance of the Residuals is given as",variance_pred)
print("SD of the Residuals is given as",SD_pred)

# ****************************************************************************
# ************* Q Value - Residuals ************************
# ****************************************************************************
```

```
q = ACF(X_train, predict_error, 30, "Linear Regression Residuals")
print("Q value of Linear Regression Model Residuals: ", q)

# **************************************************************************
# *********** Linear Regression Prediction & Plot **************************
# **************************************************************************
predicts = model.predict(X_test)
x_ind = np.arange(0,len(Y_test))
plt.figure()
plt.title("Linear Regression Weather Dataset - Prediction")
plt.plot(x_ind[:720],Y_test[:720], label="test")
plt.plot(x_ind[:720],predicts[:720],label="Predictions")
plt.xlabel("Time")
plt.ylabel("Magnitude")
plt.legend()
plt.show()


print("\nLinear Regression Results")
print("Mean of the Residuals: ", np.mean(predict_error))
print("Variance of the Residuals: ",variance_pred)
print("MSE Residuals: ", np.square(np.mean(predict_error)))
print("Q Value: ", q)
print("Mean of the Forecast Error: ", np.mean(forecast_error))
print("Variance of the Forecast Error: ",variance_forec)
print("MSE Forecast Error: ", np.square(np.mean(forecast_error)))
print("Correlation coefficients predicted value and test set:", af)
print("correlation coefficients predicted value and original set:", a)



#Helper functions
#Gpac Function
def GPAC(ry,a,b):
    gpac_mat = []
    for j in range(0,b):
        row = []
        for k in range(1,a+1):
            mat_ele = [[ry[np.abs(n)] for n in range(j-m,k+j-m)] for m in
range(0,k-1)]
            last_num = [ry[np.abs(n)] for n in range(j+1,k+j+1)]
            last_den = [ry[np.abs(n)] for n in range(j-k+1,j+1)]
            num = mat_ele.copy()
            den = mat_ele.copy()
            num.append(last_num)
            den.append(last_den)
            num = np.array(num).transpose()
            den = np.array(den).transpose()
            #gpac = np.round((np.linalg.det(num) / np.linalg.det(den)),3)
            det_num = np.linalg.det(num)
            det_den = np.linalg.det(den)
            if det_den == 0.0:
                gpac = math.inf
            else:
                gpac = det_num/det_den
            row.append(gpac)
        gpac_mat.append(row)
```

```python
    GPAC_tab = pd.DataFrame(gpac_mat, columns=list(np.arange(1, a+1)))
    return GPAC_tab

#mean subtraction
y = sum_diff_24_1[1:] - np.mean(sum_diff_24_1[1:])

#Find ry - autocorrelation values with lags of 50
ry = ACF_func(y, 60,'Seattle Weather data after detrended and seasonaility adj')
G_pac = GPAC(ry,8,8)

print("\nGPAC Results\n")
print(G_pac)
G_pac.replace(np.inf, np.nan, inplace=True)
# Heat map
ax = sns.heatmap(G_pac, center=0, cmap=sns.color_palette('rocket_r'),
annot=True, linewidths=.5)
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
ax.set_xticklabels(ax.get_xticklabels())
plt.title("Generalized Partial Autocorrelation(GPAC) table")
plt.show()

#PACF Plots
pcf_plot(sum_diff_24_1[1:],60, "Summer Data  - seasonal diff + one diff")

# Slicing 24 lagged component values
acf_24 = acf(y,nlags = 216)
pacf_24 = pacf(y,nlags = 216)
acf_seas = acf_24[::24]
pacf_seas = pacf_24[::24]

#plotting
lag_com = np.arange(24,240,24)
x = [0]
x.extend(lag_com)
plt.figure()
plt.stem(x,acf_seas)
plt.title("ACF - 24 lagged component")
plt.xticks(x)
plt.show()

plt.figure()
plt.stem(x,pacf_seas)
plt.title("PACF - 24 lagged component")
plt.xticks(x)
plt.show()


#Parameter Estimation
#Helper Functions
#Step 1 - Lavenberg Marquardf Algorithm
def gardient_Cal(y,theta,na,nb,sigma):
    den = [1]
    den.extend(theta[:na])
    num = [1]
    num.extend(theta[na:])
    if len(num) != len(den):
```

```
        diff = np.abs(len(num) - len(den))
        diff = list(np.zeros(diff, dtype=int))
        if len(num) < len(den):
            num.extend(diff)
        else:
            den.extend(diff)
    sys = (den, num, 1)
    _, e = signal.dlsim(sys, y)
    n = na+nb
    X = []
    for i in range(0,na):
        den_new = den.copy()
        den_new[i+1] = den[i+1]+sigma
        num_new = num
        sys = (den_new, num_new, 1)
        _, e_theta = signal.dlsim(sys, y)
        x = (e - e_theta)/sigma
        x = x.flatten()
        X.append(x.transpose())
    for i in range(0,n-na):
        num_new = num.copy()
        num_new[i+1] = num[i+1]+sigma
        den_new = den
        sys = (den_new, num_new, 1)
        _, e_theta = signal.dlsim(sys, y)
        x = (e - e_theta)/sigma
        x = x.flatten()
        X.append(x.transpose())
    X = np.array(X).transpose()
    A = np.matmul(X.transpose(), X)
    g = np.matmul(X.transpose(),e)
    SSE = np.matmul(e.transpose(), e)
    return A,SSE,g

#Step 2 - Lavenberg Marquardf Algorithm
def max_prob_lm_alg(A,g,y,u,na,nb, theta):
    n = na+nb
    I = np.identity(n)
    det_theta = np.matmul(np.linalg.inv((A + (u*I))),g)
    det_theta = det_theta.flatten()
    theta_new = theta+det_theta
    den = [1]
    den.extend(theta_new[:na])
    num = [1]
    num.extend(theta_new[na:])
    if len(num) != len(den):
        diff = np.abs(len(num) - len(den))
        diff = list(np.zeros(diff, dtype=int))
        if len(num) < len(den):
            num.extend(diff)
        else:
            den.extend(diff)
    sys = (den, num, 1)
    _, e = signal.dlsim(sys, y)
    SSE_new = np.matmul(e.transpose(), e)
    return SSE_new,theta_new,det_theta

#Step 3 - Lavenberg Marquardf Algorithm
```

```
def
lm_alg(iter_max,SSE_new,SSE,det_theta,theta_new,na,nb,A,u,u_max,y,sigma,g):
    N = len(y)
    n = na+nb
    SSE_plot = [SSE[0][0]]
    for i in range(1,iter_max):
        SSE_plot.append(SSE_new[0][0])
        if SSE_new < SSE:
            if np.linalg.norm(det_theta,2) < (10**-4):
                theta_hat = theta_new
                sigma_sq = (SSE_new)/(N-n)
                cov = sigma_sq * np.linalg.inv(A)
                print("Max Iteration for Convergence: ", i)
                return theta_hat, cov, SSE_plot
            else:
                theta=theta_new
                u=u/10
                A,SSE,g = gardient_Cal(y, theta, na, nb, sigma)
                SSE_new, theta_new, det_theta = max_prob_lm_alg(A, g,y, u,
na, nb, theta)
        while(SSE_new >= SSE):
            u = u*10
            if u > u_max:
                print("u is greater than max of u - 10^10")
                return 0,0,0
            SSE_new, theta_new, det_theta = max_prob_lm_alg(A,g,
y,u,na,nb,theta)
        if i == iter_max-1:
            print("Maximum Iteration reached")
            return 0, 0,0
#Confidence Interval
def confidence_interval(theta,covar,na,nb):
    upper_interval = [theta[i]+(2*np.sqrt(covar[i][i])) for i in
range(0,len(theta))]
    lower_interval = [theta[i] - (2 * np.sqrt(covar[i][i])) for i in range(0,
len(theta))]
    print("\nConfidence Interval")
    for i in range(0,na):
        print("{} < a{} <
{}".format(lower_interval[i],i+1,upper_interval[i]))
    for i in range(0,nb):
        print("{} < b{} <
{}".format(lower_interval[i+na],i+1,upper_interval[i+na]))

#One Step Prediction
#ALl the initial conditions are assumed as zero
def one_step_predictions(train,y_para,e_para):
    predicts = [0]
    train = list(train)
    y = np.array(train).reshape(len(train),)
    na_len = len(y_para)
    nb_len = len(e_para)
    #Y parameter values
    pred_y_term = [np.sum([y[i-j-1]*(-y_para[j]) for j in range(0, na_len) if
i-j > 0]) for i in range(1, len(y))]
    if nb_len !=0:
        pred_y_term[0] = pred_y_term[0] + (e_para[0]*y[0])
        predicts.append(pred_y_term[0])
```

```
        prediction_error = list(y[:2] - predicts)
        #e coeeficients
        for i in range(2,len(y)):
            pred_e_terms = np.sum([((y[i-j-1] * e_para[j]) - predicts[i-j-
1]*(e_para[j])) for j in range(0, nb_len) if i-j >= 1])
            predicts.append(pred_y_term[i-1]+pred_e_terms)
            preds = y[i] - predicts[i]
            prediction_error.append(preds)
    else:
        predicts.extend(pred_y_term)
        prediction_error = list(y - predicts)
    return predicts,prediction_error

#Correlation co-efficient
def correlation_coefficient_cal(X,Y):
    mean_x = np.mean(X)
    mean_y = np.mean(Y)
    sq_var_x = []
    sq_var_y = []
    for i in range(0,len(X)):
        sq_var_x.append(X[i]-mean_x)
        sq_var_y.append(Y[i] - mean_y)
    cov_xy = sum([ x*y for x,y in zip(sq_var_x,sq_var_y)])
    cov_x = round(np.sqrt(sum(np.square(sq_var_x))),2)
    cov_y = round(np.sqrt(sum(np.square(sq_var_y))),2)
    r = cov_xy/(cov_x * cov_y)
    return round(r,2)

#From the GPAC Results
#ARMA (2,0) Process
na = 2
nb = 0


#Initializing the Paramters for the LM

sigma = 10**-6
iter_max = 100
u_max = 10**10
u = 0.01

#Parameter Estimation
# Step 0 - LM Algorithm
theta = list(np.zeros(na + nb, dtype=int))
#Step 1 - LM Algorithm
A, SSE_old, g = gardient_Cal(y, theta, na, nb, sigma)
#Step 2- LM ALgorithm
SSE_new, theta_new, det_theta = max_prob_lm_alg(A, g, y, u, na, nb, theta)
print("\nLM Algorithm Results")
#Step 3 - LM Algorthm
theta_hat, cov, SSE_vals = lm_alg(iter_max, SSE_new, SSE_old, det_theta,
theta_new, na, nb, A, u, u_max, y, sigma, g)
if np.all(theta_hat) == True:
    print("\nFinal Parameters")
    print(theta_hat)
    confidence_interval(theta_hat, cov, na, nb)
    #One step prediction
    y_para = theta_hat[:na]
```

```
    e_para = theta_hat[na:]
    predict_val, pred_error = one_step_predictions(y, y_para, e_para)
    y_ax = np.arange(1, len(y) + 1)
    plt.figure()
    plt.plot(y_ax[1:500], y[1:500], label=["Training Set"])
    plt.plot(predict_val[1:500], label=["One-step Predicted Set"])
    plt.xlabel("TimeSteps")
    plt.ylabel("Y Values")
    plt.legend()
    plt.title("ARMA ({},{}) Process - One Step Prediction".format(na,nb))
    plt.show()
    #Residual Errors
    ACF_Residuals = ACF_func(pred_error, 30, "ARMA ({},{})
Residuals".format(na,nb))
    mean_error = np.mean(pred_error)
    var_error = np.var(pred_error)
    MSE_predicted_error = np.mean(np.array(pred_error) ** 2)
    q = len(y) * np.sum(np.array(ACF_Residuals[1:]) ** 2)
    print("\nQ Value of the Residuals: ", q)
    print("\nChi-Square Whiteness Test")
    DOF = 30 - na - nb
    alfa = 0.05
    chi_critical = chi2.ppf(1 - alfa, DOF)
    print("Q critical value from chi2 table - ", chi_critical)
    if q < chi_critical:
        print("The Residuals are White")
    else:
        print("Residuals are not White")
    #Residual Error stats
    mean_error = np.mean(pred_error)
    var_error = np.var(pred_error)
    MSE_predicted_error = np.mean(np.array(pred_error) ** 2)
    print("\nResidual error stats")
    print("Mean of the Residual error", mean_error)
    print("Variance of the residual error", var_error)
    print("MSE of the error", MSE_predicted_error)
    num_val = [1]
    num_val.extend(y_para)
    den_val = [1]
    den_val.extend(e_para)
    num_roots = np.roots(num_val)
    den_roots = np.roots(den_val)
    print("The Roots of the numerators are ", num_roots)
    print("The Roots of the denominators are ", den_roots)
    print("The covariance matrix for the estimated parameters : ", cov)
    # PACF Plots
    pcf_plot(pred_error, 60, "ARMA({},{}) Residuals".format(na,nb))

#ARMA Model from the package to check the co-efficient results
from statsmodels.tsa.arima_model import ARIMA

# 1
model = ARIMA(y, order=(na,0,nb))
model_fit = model.fit(disp=0)
print(model_fit.summary())

ARMA_Model_Sea = pd.DataFrame(columns=['ARMA (2,0)'])
ARMA_Model_Sea['ARMA (2,0)'] = [MSE_predicted_error, mean_error, var_error,
```

```
q,"No" ]
ARMA_Model_Sea =
ARMA_Model_Sea.set_index(pd.Series(['MSE_pred','Mean_pred','Variance_pred','Q
Value', 'Chi square test passed']))

#ALl initial conditions are set to zero
def multi_step_predict(train,test,y_para,e_para):
    predicts = []
    train = np.array(train).reshape(len(train),)
    test = np.array(test).reshape(len(test),)
    train_len = len(train)
    na_len = len(y_para)
    nb_len = len(e_para)
    if nb_len != 0:
        if na_len != 0:
            for i in range(0,na_len):
                y_term = np.sum([-train[train_len-j+i]*y_para[j-1] for j in
range(1, na_len+1) if i-j < 0])
                y_term_fut = np.sum([-predicts[i-j]*y_para[j-1] for j in
range(1, na_len+1) if i-j >= 0])
                y_term = y_term+y_term_fut
                if i < nb_len:
                    e_term = np.sum([train[train_len-j+i]*e_para[j-1] for j
in range(1, nb_len+1) if i-j < 0])
                else:
                    e_term = 0.0
                preds = y_term+e_term
                predicts.append(preds)
            for i in range(na_len, len(test)):
                y_term = np.sum([-predicts[i - j] * y_para[j - 1] for j in
range(1, na_len + 1)])
                if nb_len <= i:
                    predicts.append(y_term)
                else:
                    e_term = np.sum([train[train_len - j] * e_para[j - 1] for
j in range(1, nb_len + 1)])
                    preds = y_term + e_term
                    predicts.append(preds)
        else:
            for i in range(0,nb_len):
                e_term = np.sum([train[train_len - j + i] * e_para[j - 1] for
j in range(1, nb_len + 1) if i - j < 0])
                predicts.append(e_term)
            for i in range(nb_len,len(test)):
                e_t = 0
                predicts.append(e_t)
    else:
        for i in range(0,na_len):
            y_term = np.sum([-train[train_len-j+i]*y_para[j-1] for j in
range(1, na_len+1) if i-j < 0])
            y_term_fut = np.sum([-predicts[i-j]*y_para[j-1] for j in range(1,
na_len+1) if i-j >= 0])
            y_term = y_term+y_term_fut
            predicts.append(y_term)
        for i in range(na_len, len(test)):
            y_term = np.sum([-predicts[i - j] * y_para[j - 1] for j in
range(1, na_len + 1)])
            predicts.append(y_term)
```

```
    return predicts



#ARMA (3,0) Model

na = 3
nb = 0

#Parameter Estimation
# Step 0 - LM Algorithm
theta = list(np.zeros(na + nb, dtype=int))
#Step 1 - LM Algorithm
A, SSE_old, g = gardient_Cal(y, theta, na, nb, sigma)
#Step 2- LM ALgorithm
SSE_new, theta_new, det_theta = max_prob_lm_alg(A, g, y, u, na, nb, theta)
print("\nLM Algorithm Results")
#Step 3 - LM Algorthm
theta_hat, cov, SSE_vals = lm_alg(iter_max, SSE_new, SSE_old, det_theta,
theta_new, na, nb, A, u, u_max, y, sigma, g)
if np.all(theta_hat) == True:
    print("\nFinal Parameters")
    print(theta_hat)
    confidence_interval(theta_hat, cov, na, nb)
    #One step prediction
    y_para = theta_hat[:na]
    e_para = theta_hat[na:]
    predict_val, pred_error = one_step_predictions(y, y_para, e_para)
    y_ax = np.arange(1, len(y) + 1)
    plt.figure()
    plt.plot(y_ax[1:500], y[1:500], label=["Training Set"])
    plt.plot(predict_val[1:500], label=["One-step Predicted Set"])
    plt.xlabel("TimeSteps")
    plt.ylabel("Y Values")
    plt.legend()
    plt.title("ARMA ({},{}) Process - One Step Prediction".format(na,nb))
    plt.show()
    #Residual Errors
    ACF_Residuals = ACF_func(pred_error, 30, "ARMA ({},{})
Residuals".format(na,nb))
    mean_error = np.mean(pred_error)
    var_error = np.var(pred_error)
    MSE_predicted_error = np.mean(np.array(pred_error) ** 2)
    q = len(y) * np.sum(np.array(ACF_Residuals[1:]) ** 2)
    print("\nQ Value of the Residuals: ", q)
    print("\nChi-Square Whiteness Test")
    DOF = 30 - na - nb
    alfa = 0.05
    chi_critical = chi2.ppf(1 - alfa, DOF)
    print("Q critical value from chi2 table - ", chi_critical)
    if q < chi_critical:
        print("The Residuals are White")
    else:
        print("Residuals are not White")
    #Residual Error stats
    mean_error = np.mean(pred_error)
    var_error = np.var(pred_error)
    MSE_predicted_error = np.mean(np.array(pred_error) ** 2)
```

```
    print("\nResidual error stats")
    print("Mean of the Residual error", mean_error)
    print("Variance of the residual error", var_error)
    print("MSE of the error", MSE_predicted_error)
    num_val = [1]
    num_val.extend(y_para)
    den_val = [1]
    den_val.extend(e_para)
    num_roots = np.roots(num_val)
    den_roots = np.roots(den_val)
    print("The Roots of the numerators are ", num_roots)
    print("The Roots of the denominators are ", den_roots)
    print("The covariance matrix for the estimated parameters : ", cov)
    # PACF Plots
    pcf_plot(pred_error, 60, "ARMA({},{}) Residuals".format(na,nb))

#ARMA Model from the package to check the co-efficient results
from statsmodels.tsa.arima_model import ARIMA

# 1
model = ARIMA(y, order=(na,0,nb))
model_fit = model.fit(disp=0)
print(model_fit.summary())


ARMA_Model_Sea['ARMA (3,0)'] = [MSE_predicted_error, mean_error, var_error,
q,"No" ]


#SARIMAX - Input is direct value without differencing values as package takes
care of differencing part

smodel = sm.tsa.statespace.SARIMAX(Y_train_sum, order=(2,1,0),
seasonal_order=(0,1,1,24)).fit()
print(smodel.summary())

preds = smodel.fittedvalues
pred_error = np.array(Y_train_sum) - np.array(preds)

ACF_Residuals = ACF_func(pred_error, 30, "SARiMA (2,1,0)(0,1,1,24)
Residuals")

mean_error = np.mean(pred_error)
var_error = np.var(pred_error)
MSE_predicted_error = np.mean(np.array(pred_error) ** 2)
print("\nResidual error stats")
print("Mean of the Residual error", mean_error)
print("Variance of the residual error", var_error)
print("MSE of the error", MSE_predicted_error)
y_ax = np.arange(1,len(Y_train_sum)+1)
plt.figure()
plt.plot(np.array(y_ax), np.array(Y_train_sum), label=["Training Set"])
plt.plot(np.array(y_ax), np.array(preds), label=["One-step Predicted Set"])
plt.xlabel("TimeSteps")
plt.ylabel("Y Values")
plt.legend()
plt.title("SARIMA (2,1,0)(2,1,1,24) Process - One Step Prediction")
plt.show()
```

```
#Q value caluclation
q = len(y) * np.sum(np.array(ACF_Residuals[1:]) ** 2)
print("\nQ Value of the Residuals: ", q)
print("\nChi-Square Whiteness Test")
DOF = 30 - 2 - 0 - 2 - 1
alfa = 0.05
chi_critical = chi2.ppf(1 - alfa, DOF)
print("Q critical value from chi2 table - ", chi_critical)
if q < chi_critical:
    print("The Residuals are White")
else:
    print("Residuals are not White")


ARMA_Model_Sea['SARIMA (2,1,0),(0,1,1)'] = [MSE_predicted_error, mean_error,
var_error, q,"No" ]

print("Model Performances on the ARMA Models")
print(ARMA_Model_Sea)
```