Time Series Analysis & Multivariate Modeling

DATS 6450 (Spring 2020)

Instructor: Reza Jafari

Final Project Report

Submitted by: Akhil Kumar Baipaneni

AK Date: Dec 16, 2020

Table of Contents

Abstract 3

Introduction 3

Description of Data 3

Stationarity 5

Time Series Decomposition 5

- Moving Average Method 5
- STL Decomposition 7

Base Models 8

Feature Selection and Multiple Linear Regression 13

ARMA Model 18

Levenberg Marquardt algorithm 19

SARIMA 21

Final Model Selection 23

Forecast Function 23

h-step Prediction 23

Summary and Conclusion 23

Appendix 24

References 42

Abstract:

In this project, we will forecast the temperature data of San Francisco state using different time series models and find the best model by using metrics like mean square error, Q-value, root mean square error. We will observe that mean square error and variance of the predicted and forecast errors are very less for holt's winter seasonal method. But Q-value is very less for SARIMAX model and also the autocorrelation coefficients of the lagged values have decayed to zero.

Introduction:

I have chosen Historical Hourly Weather data 2012-2017 from Kaggle. Here is the link for the dataset https://www.kaggle.com/selfishgene/historical-hourly-weather-data. The aim of the project is to forecast the temperature of San Francisco state. We will be plotting the correlation matrix to check the multicollinearity between the dependent variables. Then, we will perform ADF fuller test to check whether our dependent variable is stationary. Next, we will perform Time series decomposition and calculate the strength of trend and seasonality in the data. Then we will proceed with the modelling on our dataset starting with holt's winter model followed by forward and backward step wise regression, ARMA, SARIMA and other base models like Average, Naïve, Drift, Simple Exponential Smoothing, Holt's Linear method. We will estimate the parameters of the ARMA model using Levenberg Marquardt algorithm. Finally, we will be finding the best model for the dataset by comparing performance of various models using different metrics and perform one-step and h-step predictions on the best model.

Description of Data:

The dataset contains approximately 5 years (October 2012- October 2017) of high temporal resolution (hourly measurements) data of various weather attributes, such as temperature, humidity, air pressure, etc. of San Francisco State. There are 44460 rows in the dataset, and it is a hourly data.

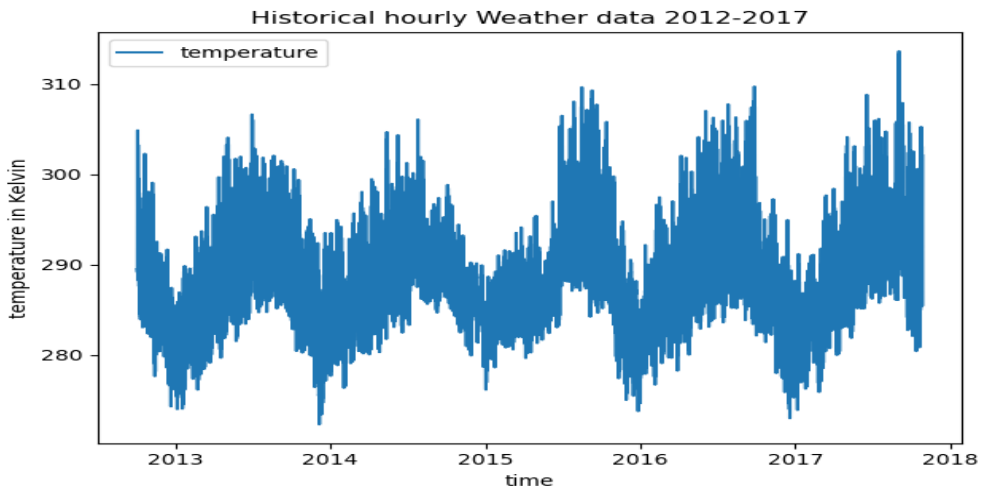
Independent Variables

- datetime - hourly data
- Humidity - amount of water vapor in the air in percentage
- Wind Speed - the rate at which air is moving in a particular area (meters/second).
- Wind Direction - degree of direction
- Pressure - Standard sea level pressure in hectopascals

Dependent Variable

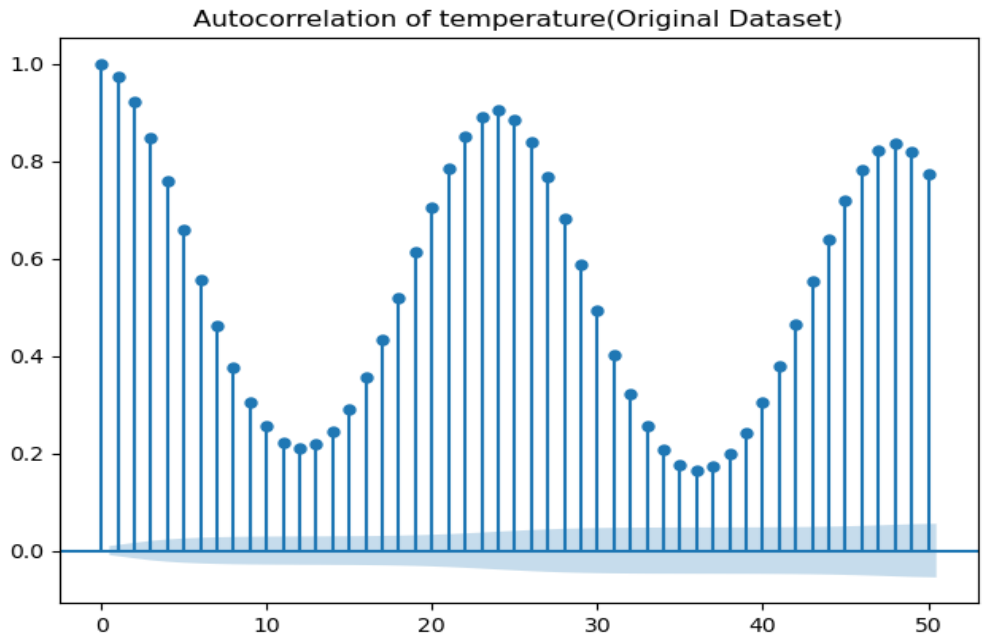
- Temperature - San Francisco temperature in kelvin scale

Below is the plot between dependent variable versus time



From the plot, we can observe that there is a strong seasonality in the dataset with a seasonal period of 24 since it is an hourly data.

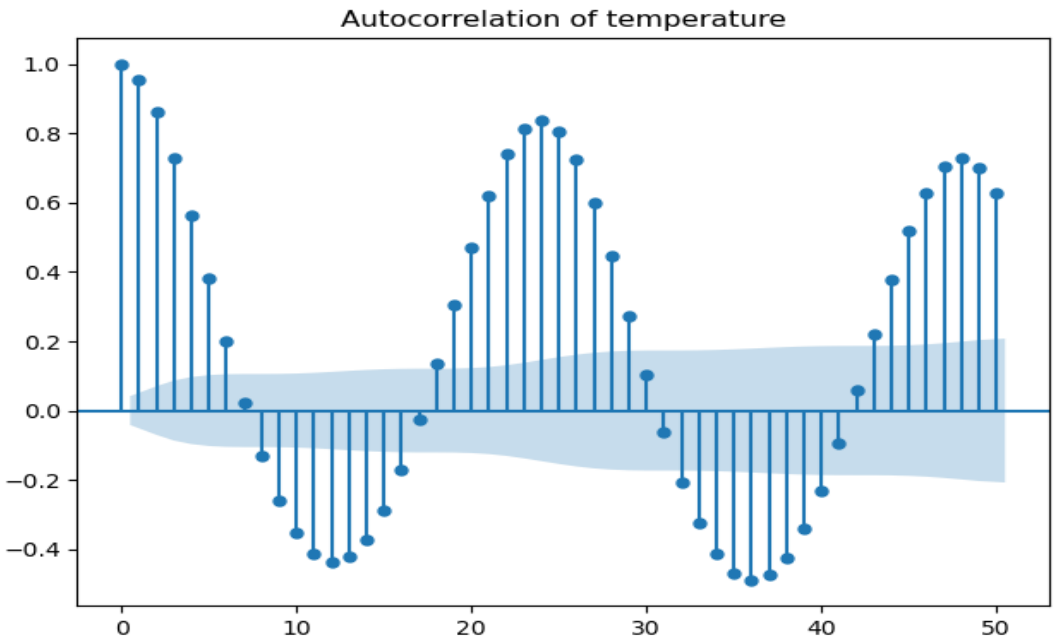
Below is the ACF plot of the dependent variable



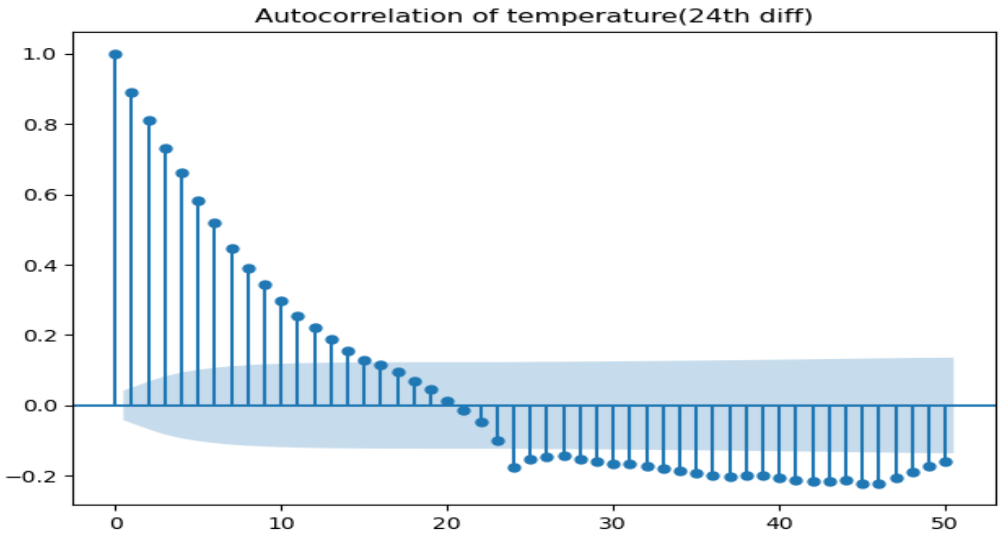
From the plot we can observe that the coefficients of the lagged values are significant and are not decaying to zero.

Based on IEEE paper "THE TIME SERIES APPROACH TO SHORT TERM LOAD FORECASTING" published by Martin T. Hagan and Suzanne M. Behr. If the dataset has a strong seasonal repetitive cycle like weather data, we need to divide the dataset into seasons like summer, spring, winter and fall and then do forecast the temperature on each season data. So, I have divided the dataset into four seasons and then taken summer season data for the time series analysis. After sampling from the original data, there were 1746 rows in the summer season dataset.

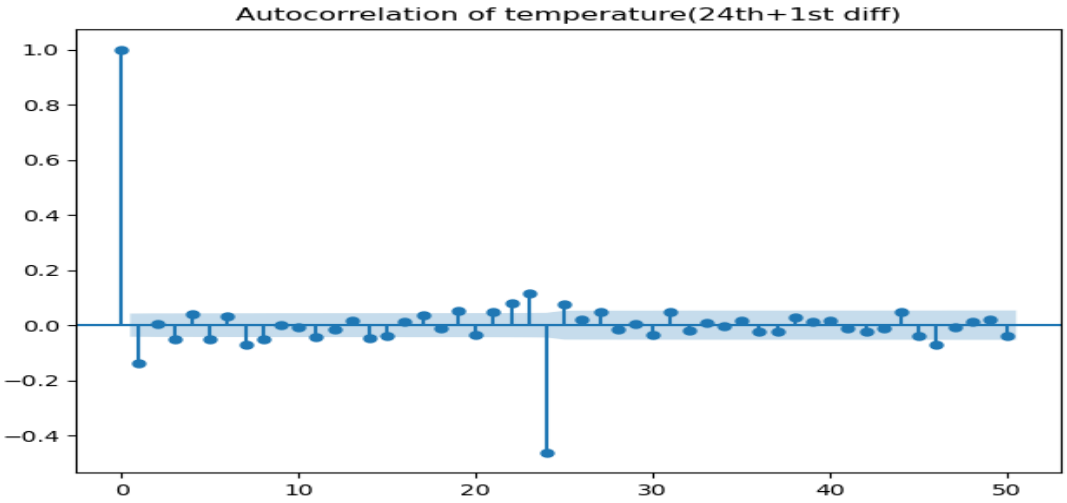
Below is the ACF plot for the sampled data



Since the coefficients of the lagged values are not decaying to zero, we have performed 24^{th} differencing on our data since the seasonal periods are 24 and then plotted the ACF plot.

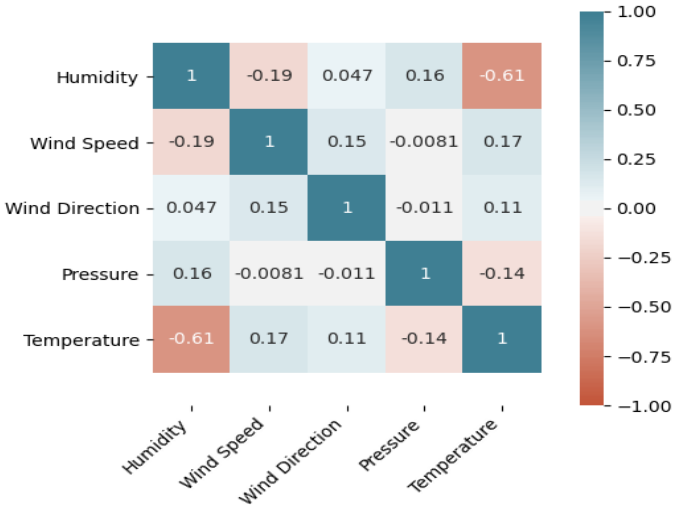


Still, we observe that the coefficients of the lagged values are significant, so we performed another differenced on the data and below is the ACF plot for 24th Differencing plus 1st order differencing with lag value as 50.



From the plot, we can observe that the coefficients are decaying to zero.

Next, we will plot the correlation matrix of the dataset. From the plot, we can observe that there is no multicollinearity exists between the dependent variables Humidity, Wind speed, wind direction, Pressure.



The correlation coefficient between the Humidity and Temperature is -0.609

The correlation coefficient between the Wind Speed and Temperature is 0.168

The correlation coefficient between the Wind Direction and Temperature is 0.111

The correlation coefficient between the Pressure and Temperature is -0.138

From the results of the Pearson Correlation coefficients, we observe that there is a negative correlation between Humidity and dependent variable Temperature.

Coming to the data pre-processing, the I have dropped the weather description from the dataset since it contains categorical data and there are few null values in the dataset. I have replaced the null values with the previous values since there will not be much difference in temperature with the previous hour. I have used forward fill method to fill the null values.

I split the dataset into 80% train and 20% test and trained the model on the train data and forecasted on the test data.

Stationarity:

Next, we will check whether the dataset is stationary by performing the ADF fuller test on the dependent variable and below are the results on the sampled data.

```
ADF Statistic: -5.087012
p-value: 0.000015
Critical Values:
    1%: -3.433
    5%: -2.863
    10%: -2.567
```

As discussed in the previous section, the autocorrelation coefficients of the lagged values are not decaying to zero. So, we have performed 24th differencing plus 1st differencing to make it decay to zero.
Below are the ADF fuller test results for 24th plus 1st differencing data:

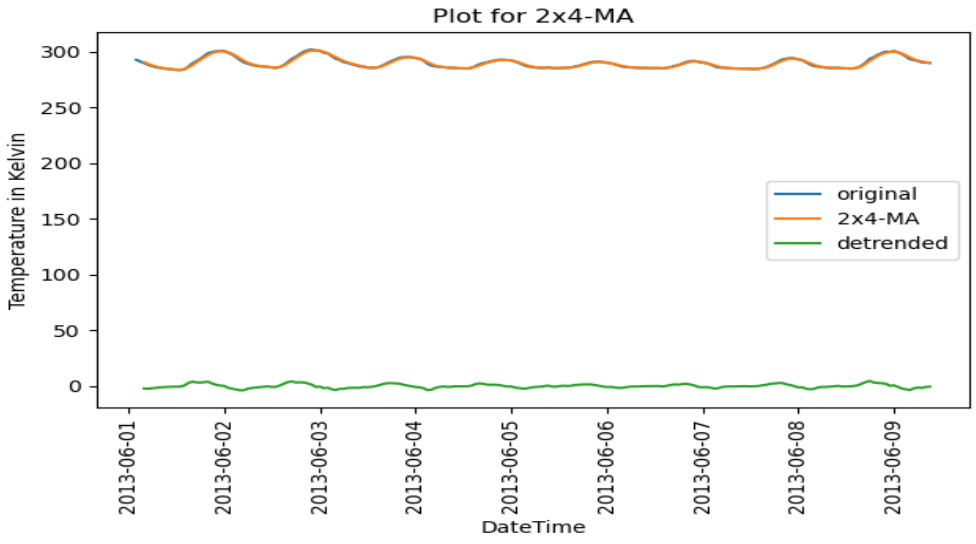
```
ADF Statistic: -12.906413
p-value: 0.000000
Critical Values:
    1%: -3.433
    5%: -2.863
    10%: -2.567
```

Since the ADF test statistic is less than the critical value, we can reject the null hypothesis and say with 99% confidence level that the data is stationary.

Time series Decomposition:

Moving Average method:

We have used 2*4 Moving Average method for approximating the trend component of the data. Below is the plot of the detrended data:



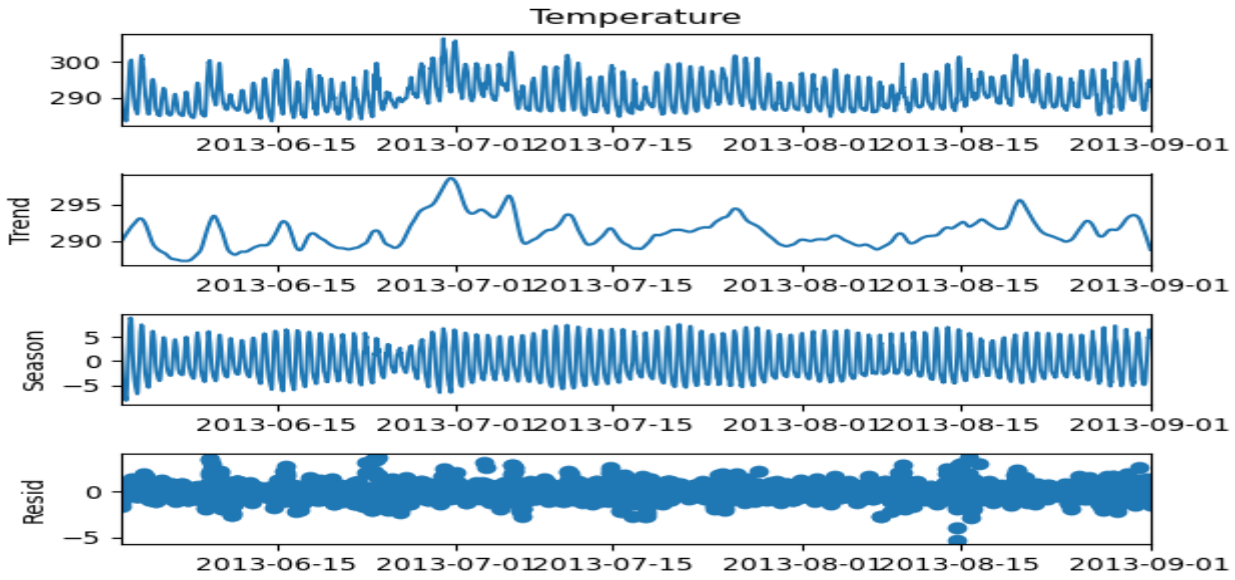
Below are the ADF fuller test on the detrended data:

```
ADF Statistic: -9.730269
p-value: 0.000000
Critical Values:
    1%: -3.433
    5%: -2.863
    10%: -2.567
```

Since the ADF test statistic is less than the critical value, we can reject the null hypothesis and say with 99% confidence level that the detrended data is stationary.

STL Decomposition:

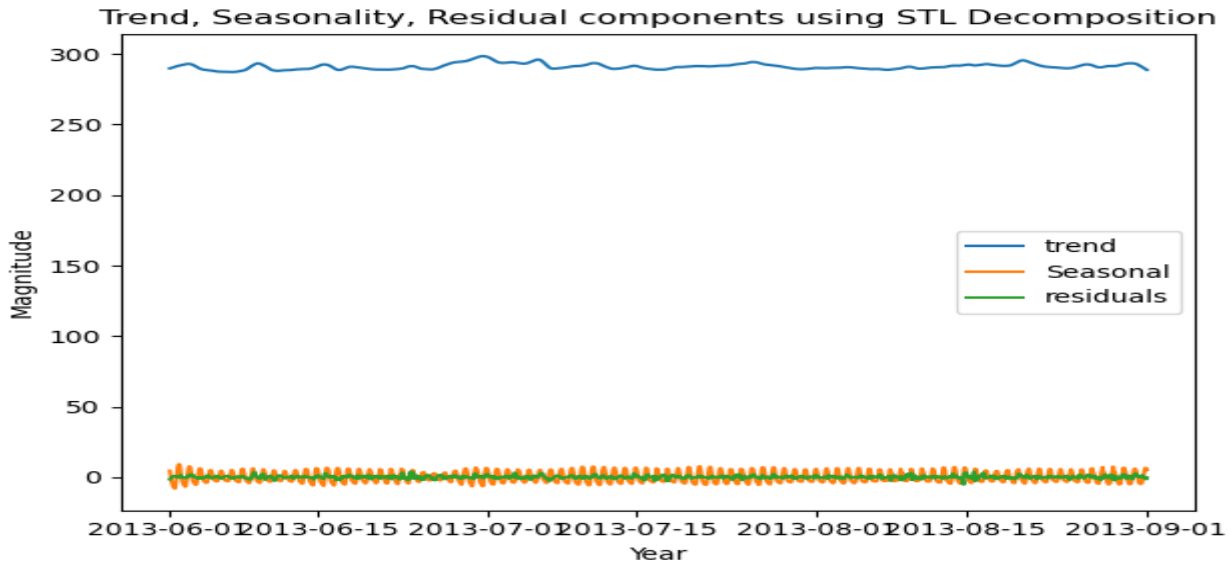
Next, we will use the STL decomposition to approximate the trend and seasonality and residual components of the dataset.



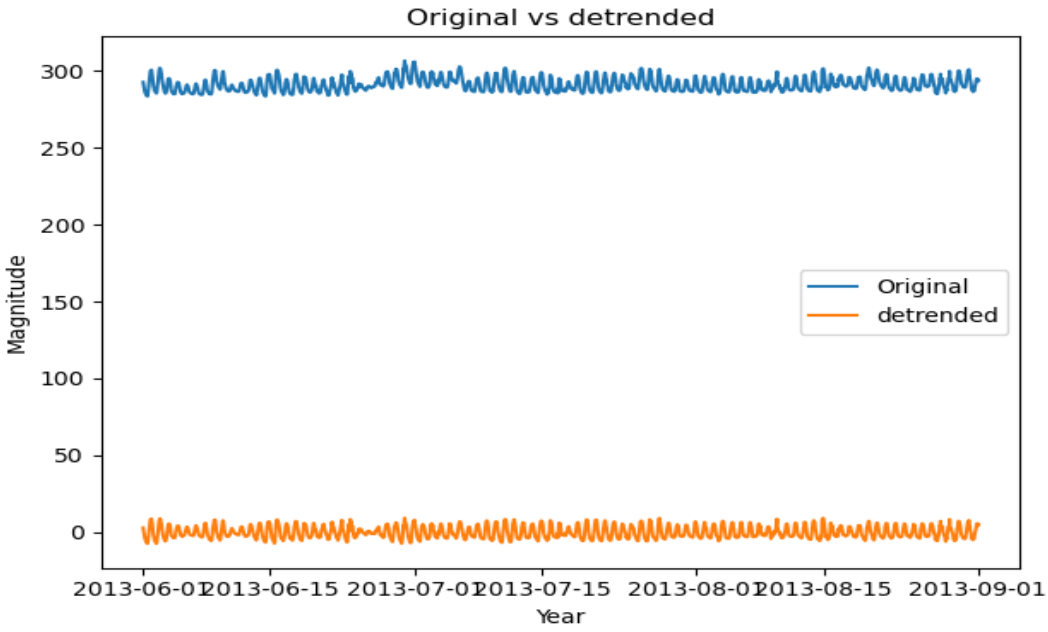
From the above plot, we observe that there is a strong seasonal component and trend involved in our dataset. We will calculate the strength of trend and seasonality in our data.

Strength of trend Hourly weather dataset is 0.863

Strength of seasonality for Hourly weather dataset is 0.950

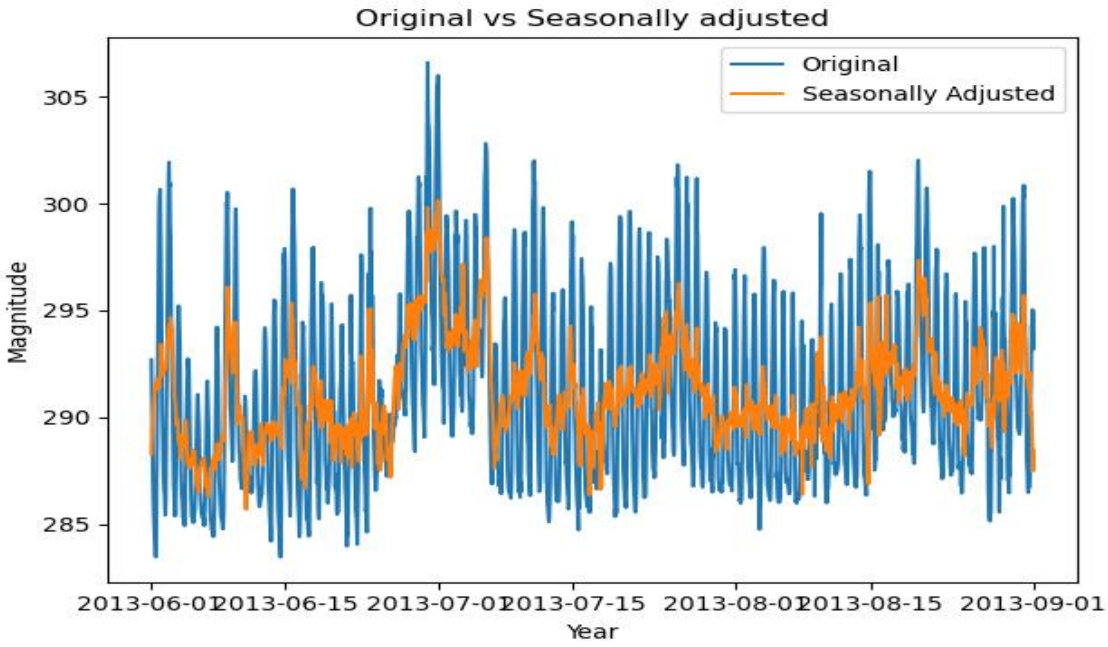


Below is the plot between the original data vs the detrended data:



From the above plot, we can observe that the detrended is stationary.

Below is the plot between Original vs Seasonally Adjusted data:

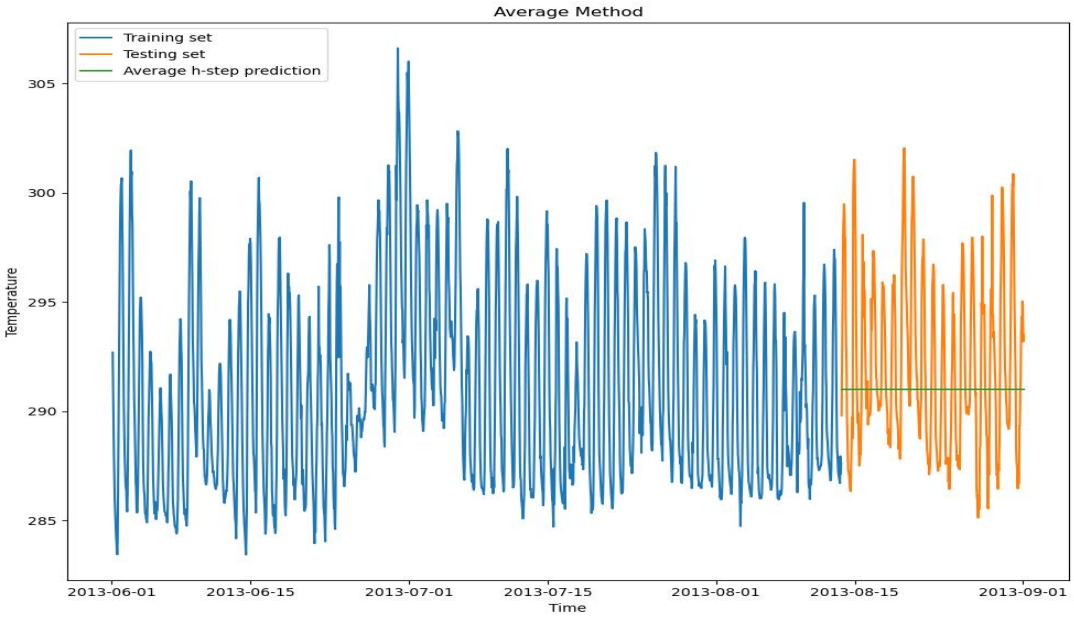


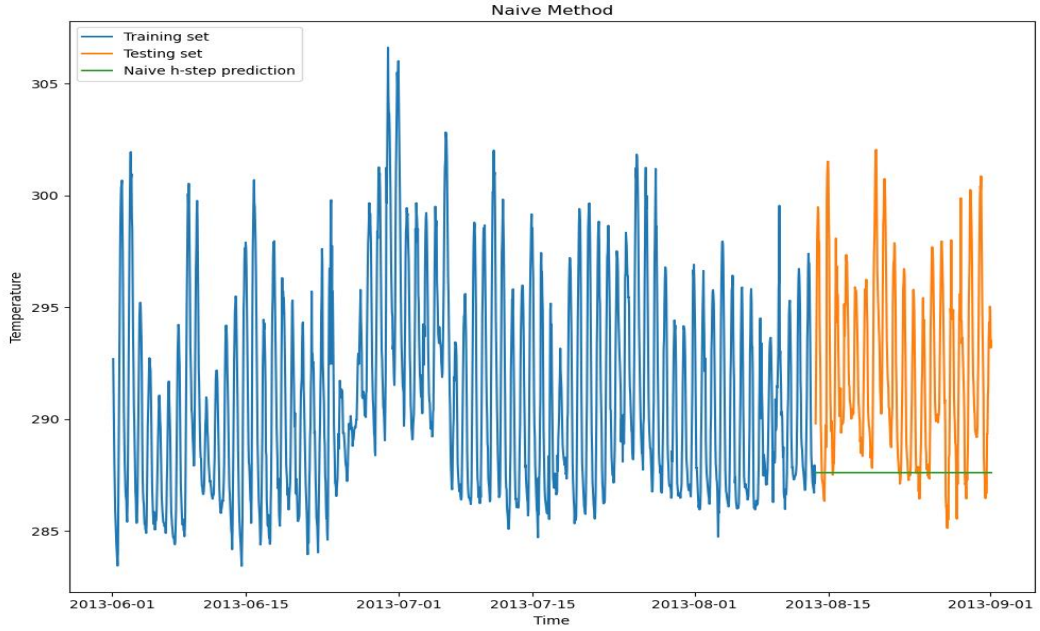
From the plot, we can observe that the seasonal component is removed from the data and we can only the trend and residual component.

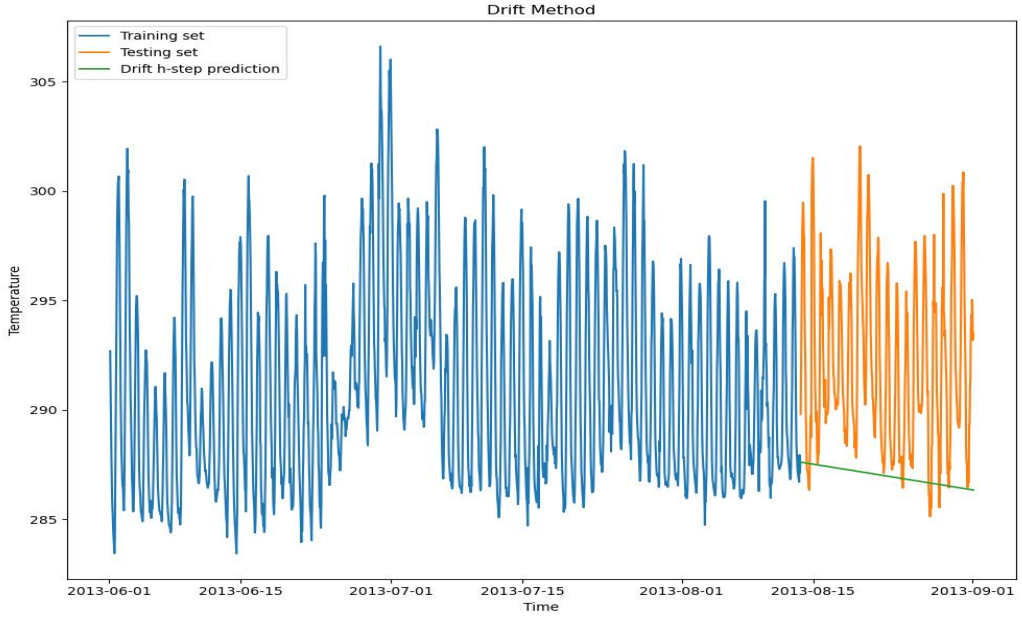
Next, we will be using the base models to find the best fit on our data and predict using the test dataset.

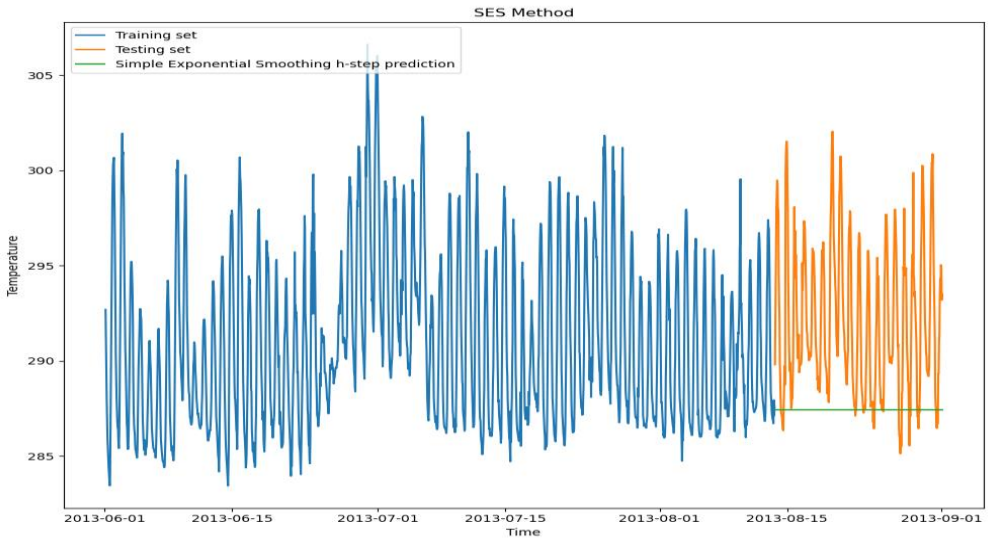
Base Models:

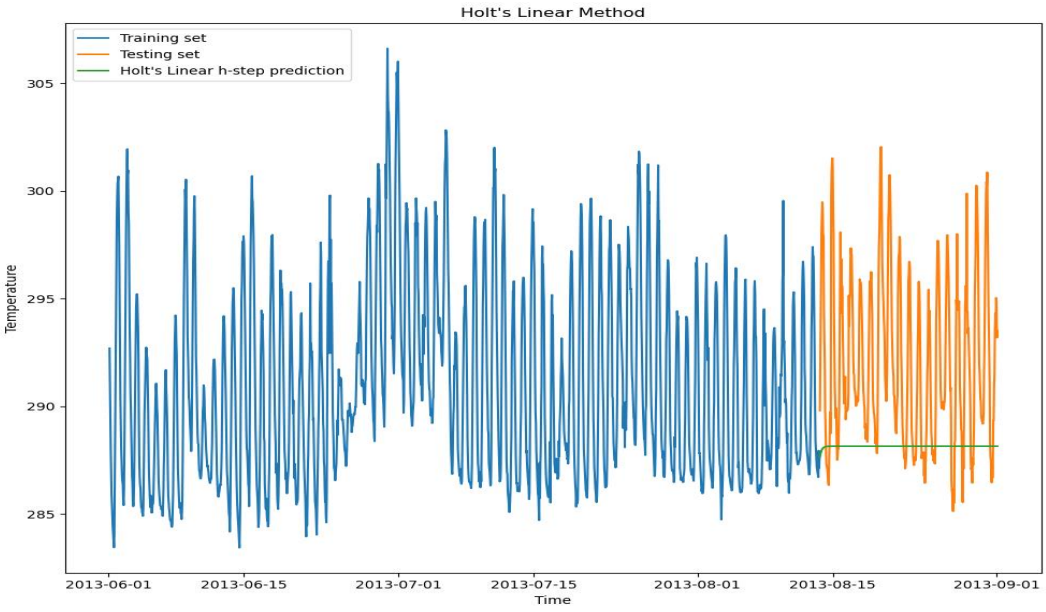
Below are the plots that show us the train data, test data and h-step predictions made using Average, Naïve, Drift, Simple Exponential smoothing, Holt's Linear, Holt's winter seasonal methods. From the plots, we can observe the Holt's winter seasonal is the best estimator of future values among base models.

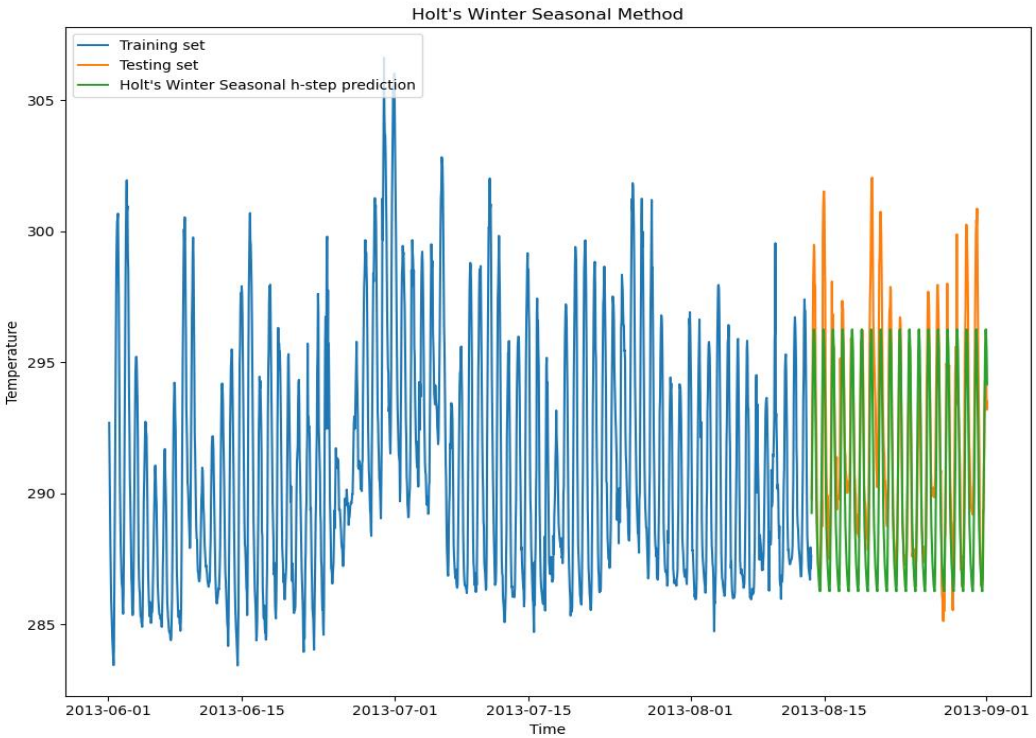

Average Method
Training set
Testing set
Average h-step prediction
Temperature
305
300
295
290
285
2013-06-01
2013-06-15
2013-07-01
2013-07-15
2013-08-01
2013-08-15
2013-09-01
Time


Naive Method
Training set
Testing set
Naive h-step prediction
Temperature
305
300
295
290
285
2013-06-01
2013-06-15
2013-07-01
2013-07-15
2013-08-01
2013-08-15
2013-09-01
Time

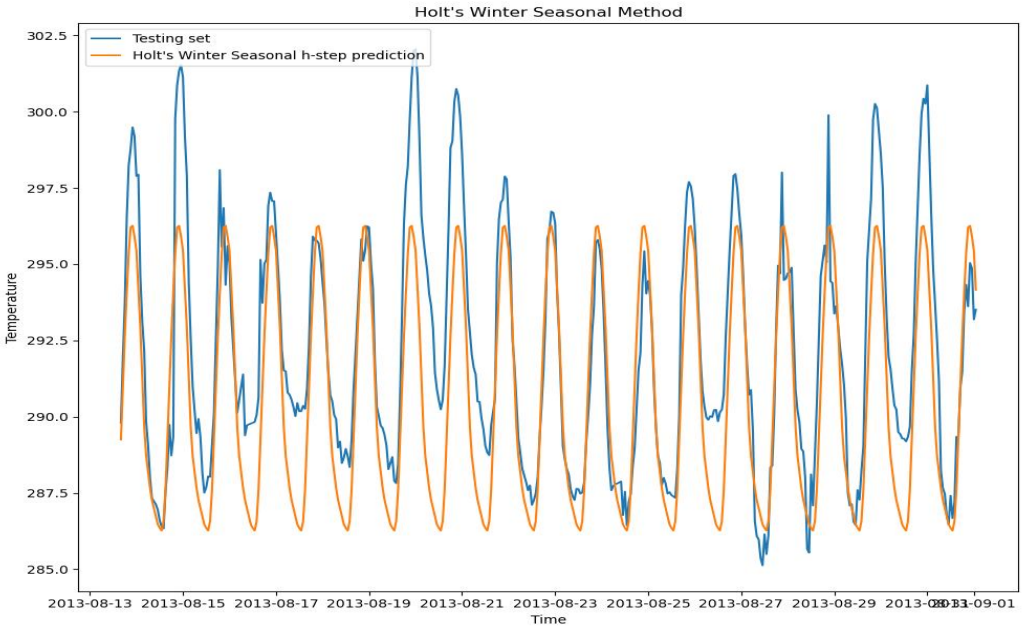

Drift Method
Training set
Testing set
Drift h-step prediction
Temperature
305
300
295
290
285
2013-06-01
2013-06-15
2013-07-01
2013-07-15
2013-08-01
2013-08-15
2013-09-01
Time


SES Method
Training set
Testing set
Simple Exponential Smoothing h-step prediction
Temperature
305
300
295
290
285
2013-06-01
2013-06-15
2013-07-01
2013-07-15
2013-08-01
2013-08-15
2013-09-01
Time

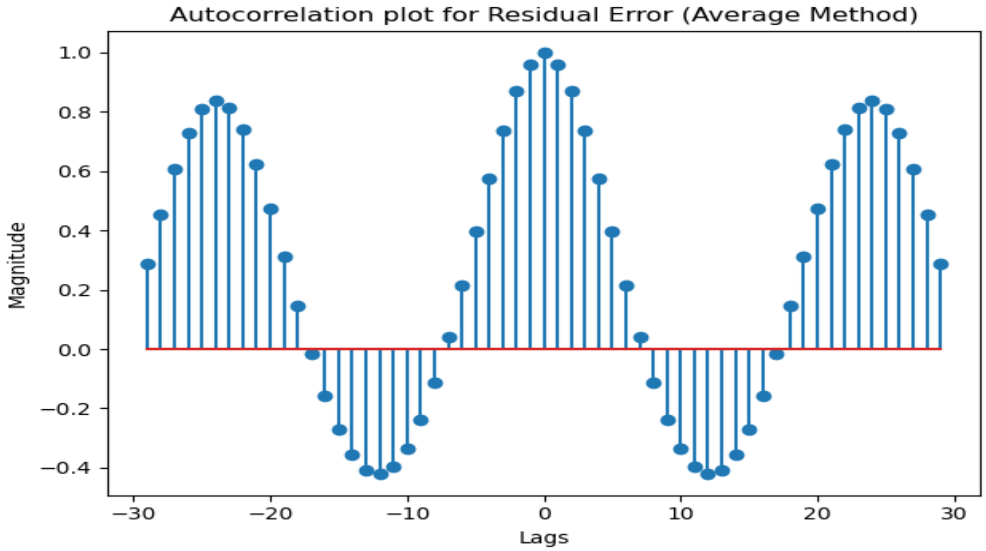

Holt's Linear Method
Training set
Testing set
Holt's Linear h-step prediction
Temperature
305
300
295
290
285
2013-06-01
2013-06-15
2013-07-01
2013-07-15
2013-08-01
2013-08-15
2013-09-01
Time

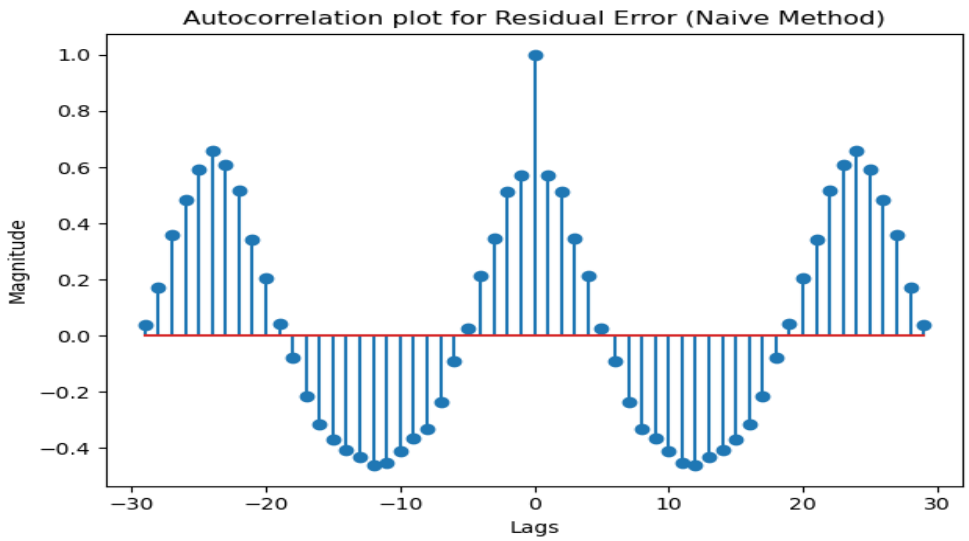

Holt's Winter Seasonal Method
Training set
Testing set
Holt's Winter Seasonal h-step prediction
Temperature
305
300
295
290
285
2013-06-01
2013-06-15
2013-07-01
2013-07-15
2013-08-01
2013-08-15
2013-09-01
Time

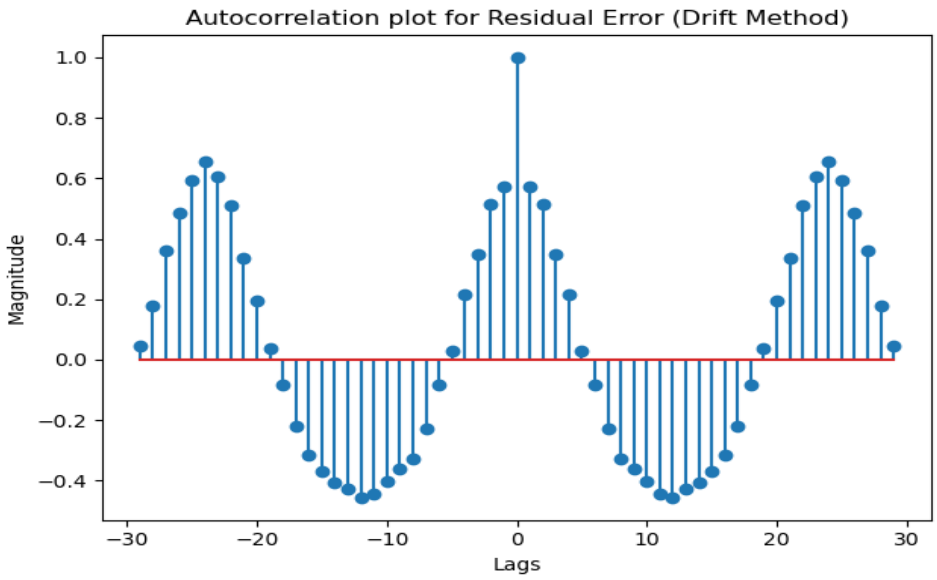
Below is the plot between testing set versus the h-step predicted values of the Holt's winter seasonal model. From the plot, we observe that the h-step predicted forecast values are almost fitting the test dataset.

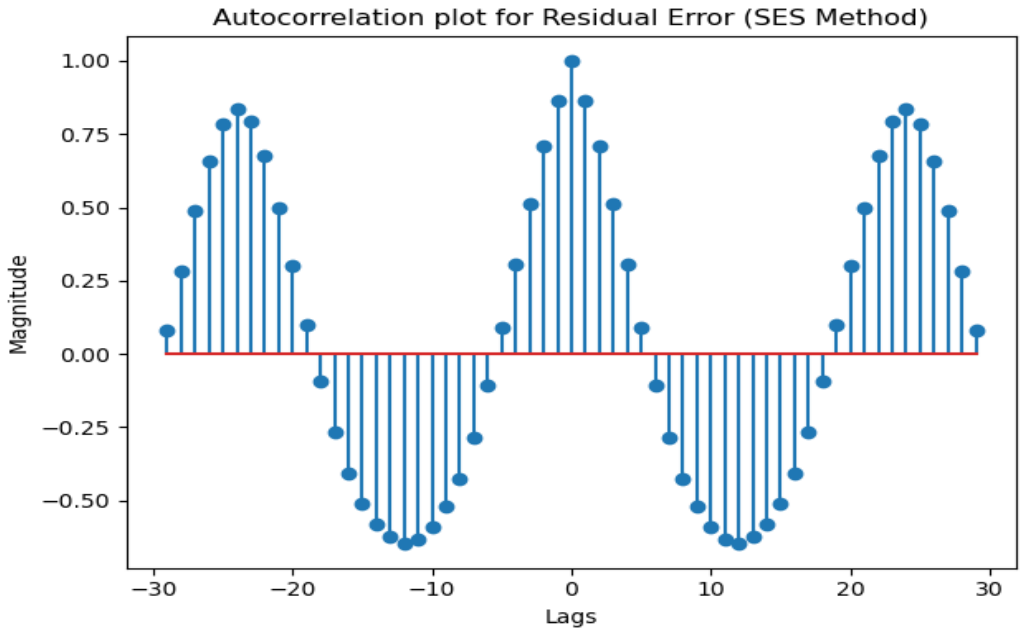


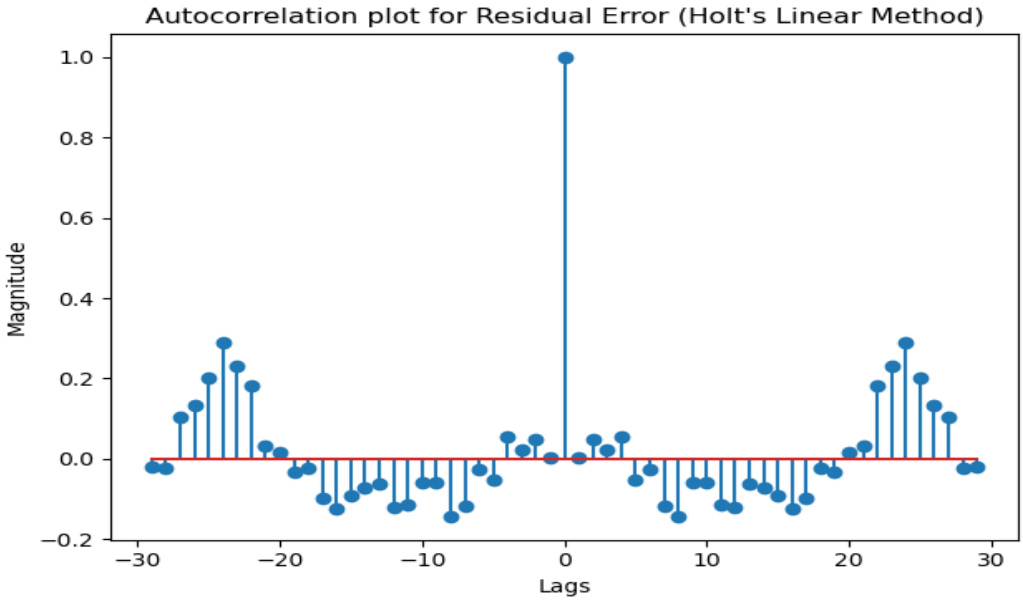
Below are the AutoCorrelation (ACF) plots for the Residual errors. From the plots, we can observe that Holt's winter seasonal plot has all of the coefficients close to zero compared to the other plots and also, we can observe the residuals error is white. Other models have not adequately captured the information in the data.

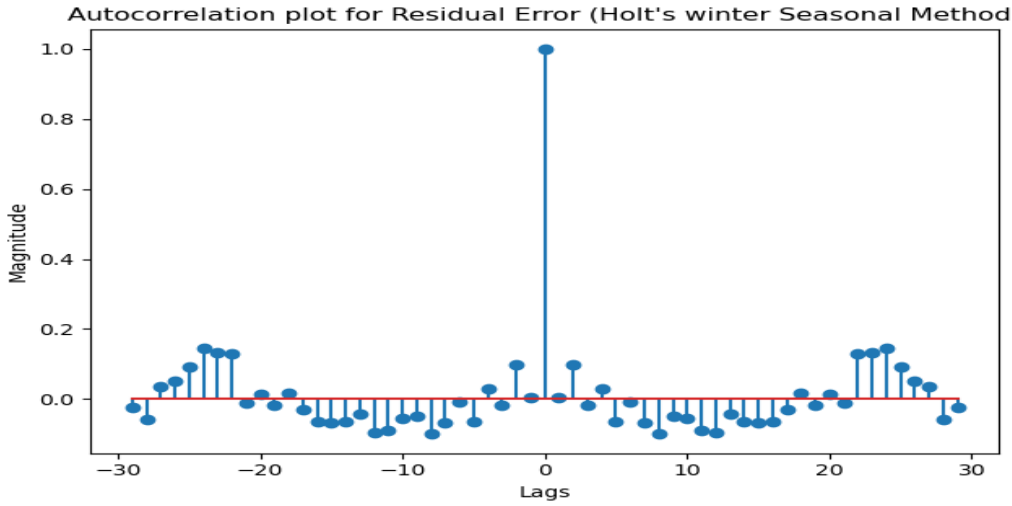


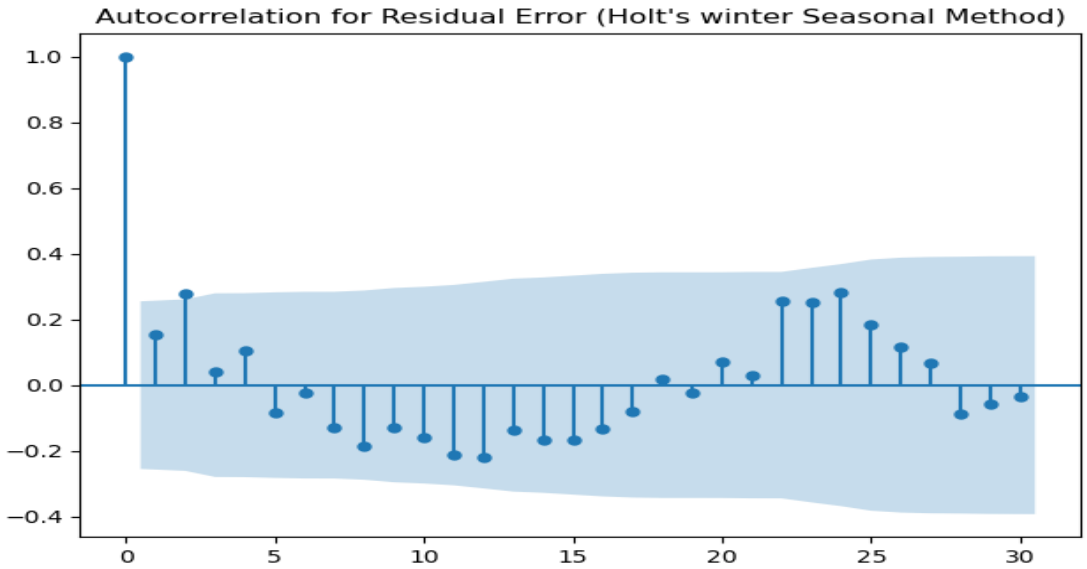



Autocorrelation plot for Residual Error (Drift Method)
Magnitude
Lags
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−30
−20
−10
0
10
20
30


Autocorrelation plot for Residual Error (SES Method)
Magnitude
Lags
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−30
−20
−10
0
10
20
30


Autocorrelation plot for Residual Error (Holt's Linear Method)
Magnitude
Lags
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−30
−20
−10
0
10
20
30





From the above plot, we can observe that the residuals are white for Holt's winter seasonal method.

Below are the consolidated results of the Base models:

Methods	Q_val	MSE(P)	MSE(F)	var(P)	var(F)	corrcoeff
Average	3597.95	20.09	16.64	19.96	15.35	1.0
Naive	1885.77	1.67	35.68	1.67	15.35	1.0
Drift	1867.08	1.67	41.76	1.67	15.28	1.0
SES	3560.33	4.29	37.29	4.29	15.35	1.0
HoltL	161.88	1.03	31.22	1.03	15.37	1.0
HoltW	246.27	0.64	5.91	0.64	3.65	0.4

From the above results, we can observe that Holt's winter seasonal method has low Mean square error value for predicted and forecast errors. Also, there is a very less difference between variance of predicted and forecast values compared to other methods. Also, from the ACF plot, we observed that the residual errors are white. So, we can conclude that Holt's winter seasonal method is the best method to fit the train data and predict the forecast values.

Feature Selection and Multiple Linear Regression:

Feature selection for the multiple Linear Regression was done using the Forward and Backward step Regression.

In the Forward step, we keep on adding the independent variables and fit the data to the ordinary Least Squares model (OLS) and check the p-value of the T-test whether they are significant or not. If they are not significant, we drop that variable from the model. Also, we will observe the Adjusted R-sqaured value whether it is increasing or decreasing after a new variable to the model.

Added Humidity variable to the model, we observe that all the coefficients of the model are significant from p-value of T-test and F-test which are less than 0.05. So, we can say that our model is better than the constant only model. Adjusted R2 and R-sqaured values are equal to 0.39. From this, we can say that this model explains 39% variability in our data. AIC and BIC values are 9432 and 9443 respectively.

OLS Regression Results

Dep. Variable:	Temperature	R-squared:	0.390
Model:	OLS	Adj. R-squared:	0.390
Method:	Least Squares	F-statistic:	1129.
Date:	Thu, 17 Dec 2020	Prob (F-statistic):	1.00e-191
Time:	15:47:41	Log-Likelihood:	-4713.9
No. Observations:	1766	AIC:	9432.
Df Residuals:	1764	BIC:	9443.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	306.4382	0.467	655.956	0.000	305.522	307.354
Humidity	-0.1874	0.006	-33.599	0.000	-0.198	-0.176

Omnibus:	248.096	Durbin-Watson:	0.294
Prob(Omnibus):	0.000	Jarque-Bera (JB):	360.577
Skew:	1.039	Prob(JB):	5.03e-79
Kurtosis:	3.762	Cond. No.	471.

Next, we have added the Wind Speed variable to our model and below are the results. We observe that all the coefficients of the model are significant from p-value of T-test and F-test which are less than 0.05. So, we can say that our model is better than the constant only model. Adjusted R2 and R-sqaured values have increased to 0.394. From this, we can say that this model explains 39.4% variability in our data. The values of AIC and BIC values have decreased to 9422 and 9438 respectively which is good.

OLS Regression Results

Dep. Variable:	Temperature	R-squared:	0.394
Model:	OLS	Adj. R-squared:	0.394
Method:	Least Squares	F-statistic:	573.9
Date:	Thu, 17 Dec 2020	Prob (F-statistic):	1.09e-192
Time:	15:47:41	Log-Likelihood:	-4707.9
No. Observations:	1766	AIC:	9422.
Df Residuals:	1763	BIC:	9438.
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	305.8302	0.498	614.356	0.000	304.854	306.807
Humidity	-0.1843	0.006	-32.728	0.000	-0.195	-0.173
Wind Speed	0.2093	0.061	3.457	0.001	0.091	0.328

Omnibus:	255.762	Durbin-Watson:	0.289
Prob(Omnibus):	0.000	Jarque-Bera (JB):	376.542
Skew:	1.056	Prob(JB):	1.72e-82
Kurtosis:	3.811	Cond. No.	504.

Next, we have added the Wind Direction variable to our model and below are the results. We observe that all the coefficients of the model are significant from p-value of T-test and F-test which are less than 0.05. So, we can say that our model is better than the constant only model. Adjusted R2 value and R-sqaured value have increased to 0.405 and 0.406 respectively. From this, we can say that this model explains 40.5% variability in our data. The values of AIC and BIC values have decreased to 9389 and 9411 respectively which is good.

OLS Regression Results

Dep. Variable:	Temperature	R-squared:	0.406
Model:	OLS	Adj. R-squared:	0.405
Method:	Least Squares	F-statistic:	401.8
Date:	Thu, 17 Dec 2020	Prob (F-statistic):	7.73e-199
Time:	15:47:41	Log-Likelihood:	-4690.4
No. Observations:	1766	AIC:	9389.
Df Residuals:	1762	BIC:	9411.
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	304.1083	0.572	531.738	0.000	302.987	305.230
Humidity	-0.1869	0.006	-33.403	0.000	-0.198	-0.176
Wind Speed	0.1517	0.061	2.497	0.013	0.033	0.271
Wind Direction	0.0088	0.001	5.941	0.000	0.006	0.012

Omnibus:	236.343	Durbin-Watson:	0.318
Prob(Omnibus):	0.000	Jarque-Bera (JB):	337.412
Skew:	1.009	Prob(JB):	5.39e-74
Kurtosis:	3.716	Cond. No.	1.74e+03

Next, we have added the Pressure variable to our model and below are the results. We observe that pressure coefficient is not significant from p-value of T-test since it is greater than 0.05. P-value of F-test is less than 0.05. So, we drop the pressure variable from the model. Adjusted R2 values and R-sqaured values did not change. The values of AIC and BIC values have increased to 9391 and 9418 respectively which is not good.

OLS Regression Results

Dep. Variable:	Temperature	R-squared:	0.406
Model:	OLS	Adj. R-squared:	0.405
Method:	Least Squares	F-statistic:	301.2
Date:	Thu, 17 Dec 2020	Prob (F-statistic):	1.64e-197
Time:	15:47:41	Log-Likelihood:	-4690.3
No. Observations:	1766	AIC:	9391.
Df Residuals:	1761	BIC:	9418.
Df Model:	4		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	298.7416	22.358	13.362	0.000	254.890	342.593
Humidity	-0.1871	0.006	-32.758	0.000	-0.198	-0.176
Wind Speed	0.1510	0.061	2.484	0.013	0.032	0.270
Wind Direction	0.0088	0.001	5.942	0.000	0.006	0.012
Pressure	0.0053	0.022	0.240	0.810	-0.038	0.049

Omnibus:	236.132	Durbin-Watson:	0.318
Prob(Omnibus):	0.000	Jarque-Bera (JB):	337.032
Skew:	1.008	Prob(JB):	6.52e-74
Kurtosis:	3.717	Cond. No.	2.84e+05

After dropping the pressure variable, the final results of the model is shown below:

OLS Regression Results

Dep. Variable:	Temperature	R-squared:	0.406
Model:	OLS	Adj. R-squared:	0.405
Method:	Least Squares	F-statistic:	401.8
Date:	Thu, 17 Dec 2020	Prob (F-statistic):	7.73e-199
Time:	15:47:41	Log-Likelihood:	-4690.4
No. Observations:	1766	AIC:	9389.
Df Residuals:	1762	BIC:	9411.
Df Model:	3		
Covariance Type:	nonrobust		

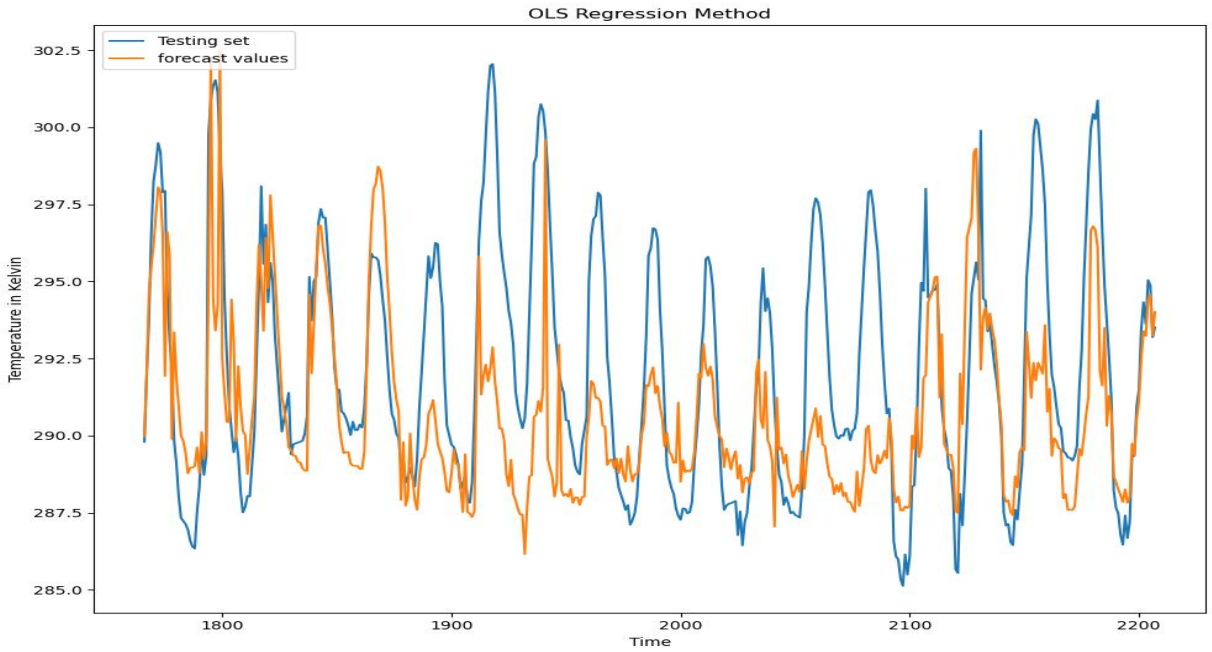
	coef	std err	t	P>|t|	[0.025	0.975]
const	304.1083	0.572	531.738	0.000	302.987	305.230
Humidity	-0.1869	0.006	-33.403	0.000	-0.198	-0.176
Wind Speed	0.1517	0.061	2.497	0.013	0.033	0.271
Wind Direction	0.0088	0.001	5.941	0.000	0.006	0.012

Omnibus:	236.343	Durbin-Watson:	0.318
Prob(Omnibus):	0.000	Jarque-Bera (JB):	337.412
Skew:	1.009	Prob(JB):	5.39e-74
Kurtosis:	3.716	Cond. No.	1.74e+03

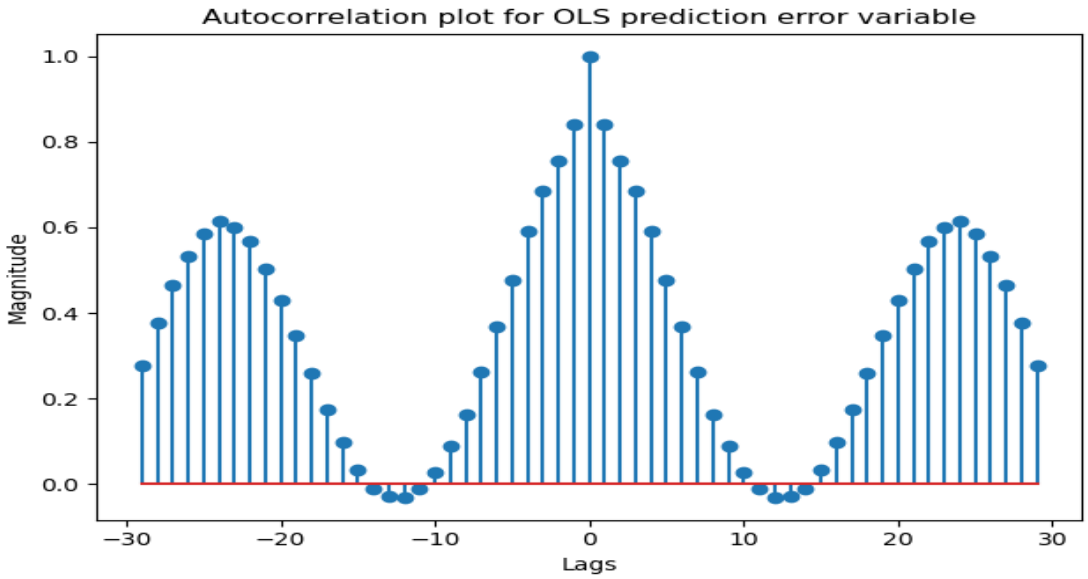
From the above results, we can observe that all the coefficients are significant since the p-value of T-test and F-test are less than 0.05. Based on Adjusted R-sqaured value, we can say that 40.6% of the variability in the data is explained by our model.

In Backward step regression, we consider all the variables and fit the data on the model. If all the coefficients are significant from T-test and F-test, we can say that model is the good model for prediction. If the coefficients are not significant and if AIC and BIC value is not decreasing and Adjusted R-sqaured is not increasing, then we will remove the variable from the model and continue the analysis. We got the same results as Forward step for the Backward step regression.

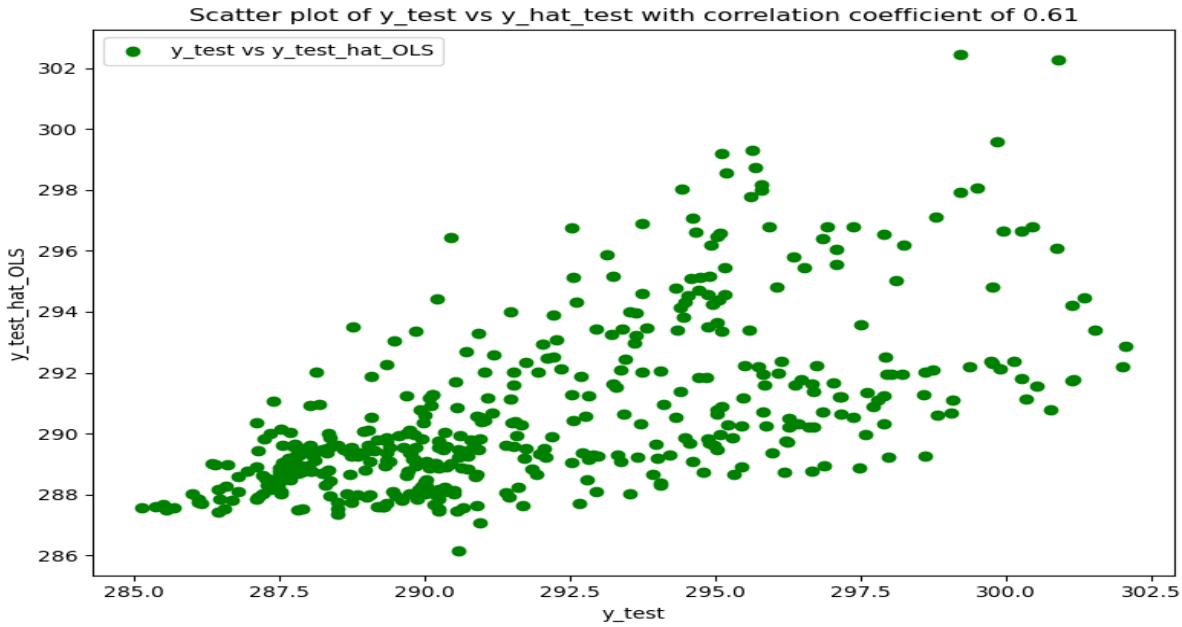
Below is the plot between one-step prediction versus the test set. From the plot, we can observe that prediction values did not fit the test data.



From the ACF plot of the residual errors, we observe that the coefficients of the lagged values are not decaying to zero. This model has not adequately captured the information in the dataset.



Below is the scatter plot between the test data and predicted values. We observe that they are positively correlated with a correlation coefficient of 0.61.



Below are the results of Q-value, mean, variance and RMSE of prediction and forecast errors.

Q value of the residual error: 9448.77

mean of prediction error: 0.00

variance of prediction error: 11.903354510025538

standard deviation of prediction error: 3.450123839810035

RMSE of prediction error: 3.445236284105255

mean of forecast error: 1.39

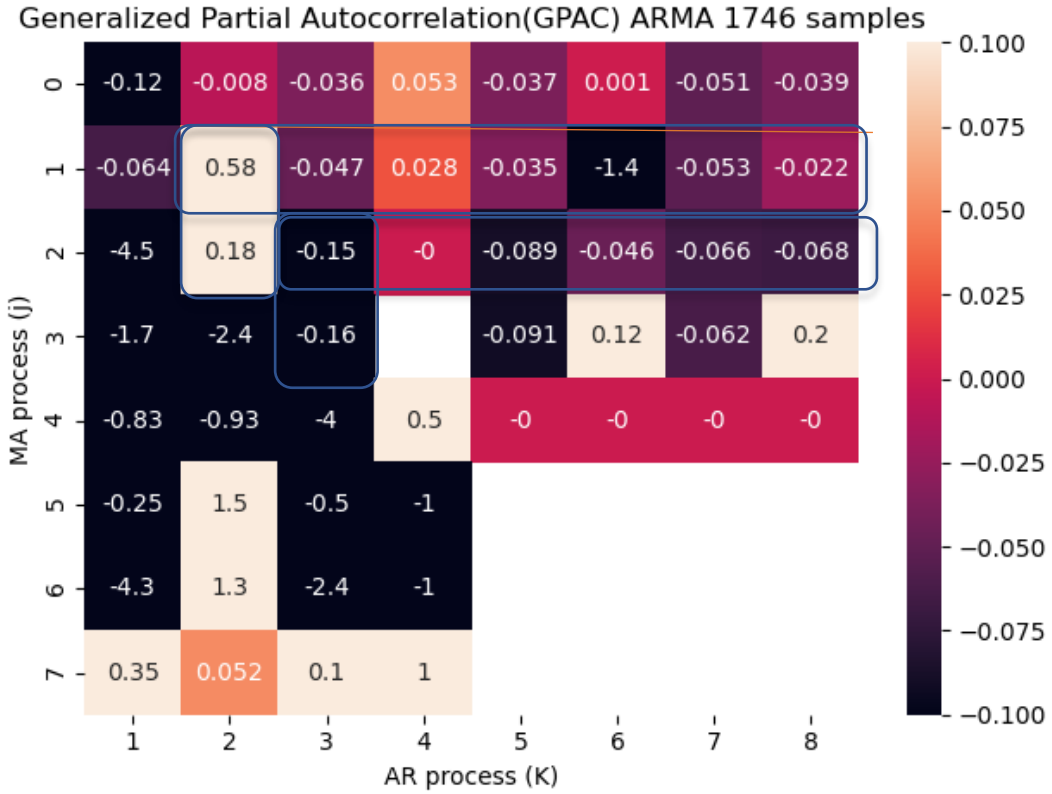
variance of forecast error: 11.779565046058144

standard deviation of forecast error: 3.4321370960464477

RMSE of forecast error: 3.4126693434611517

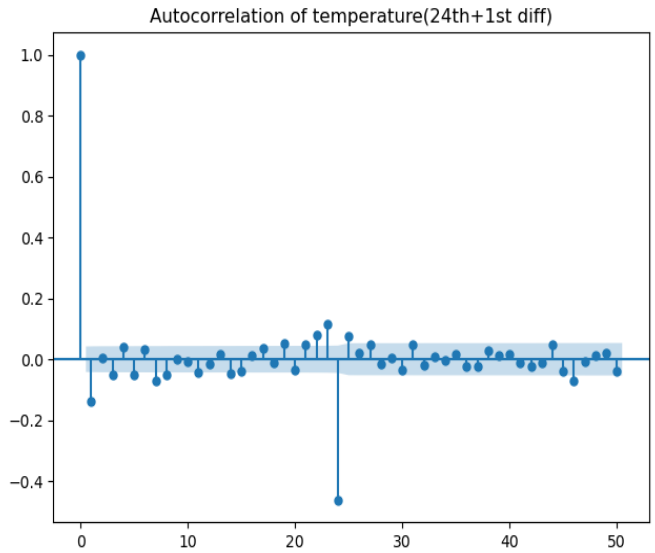
From the results, we observe that Q-value is too large. But there is very less difference between the mean of residual error and forecast error. Also, there is very less difference between the variance of residual error and forecast error. RMSE values of the residual and forecast errors are low for Multiple Linear Regression model.

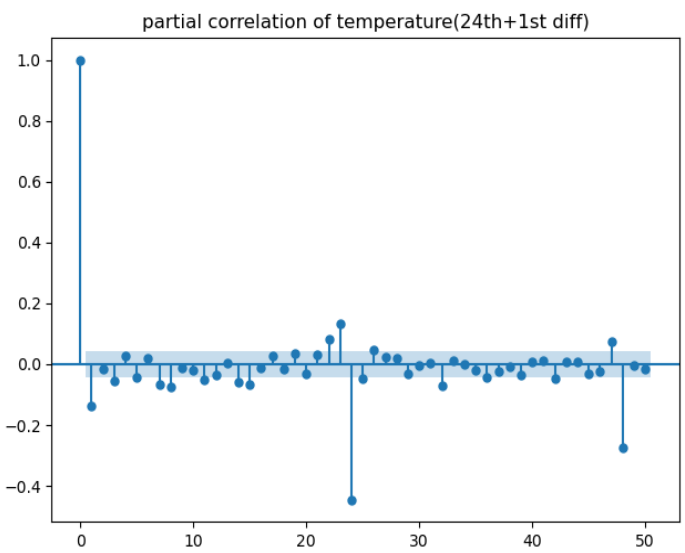
ARMA Model:



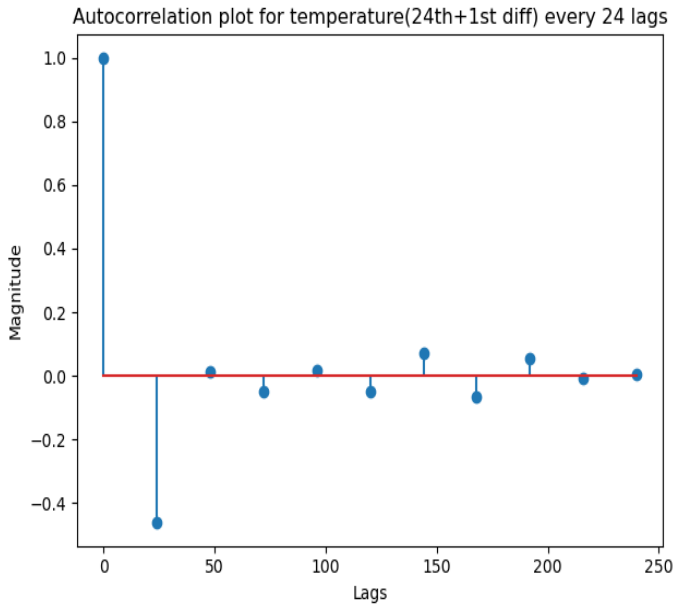
From the GPAC table, we don't see any clear pattern for identifying the order of ARMA model. But we can pick (2,1) and (3,2).

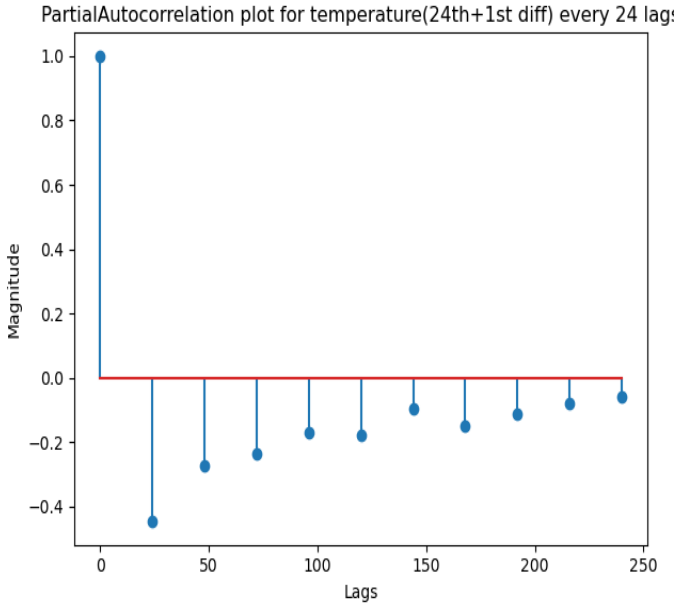
Below are the ACF and PACF plots for 24th differencing plus 1st differencing. From the plots, we observe that there is a negative spike in both the ACF and PACF plots that are significant and then the values are decayed to almost zero. So, we can say that the non-seasonal order can be 1,1 with a differencing of 1.





Below we are plots of ACF and PACF that show the only the 24th, 48th, 72nd etc. lag values of 24th differencing plus 1st differencing data. From the below plots, we observe that the values are decaying slowly in the PACF plot and negative significant spike in the ACF plot. So, we can say that the seasonal order can be (0,1,1,24).





Levenberg Marquardt algorithm:

We have estimated the coefficients of the ARMA model using this algorithm with 2, 1 as AR and MA order and below are the results.

The AR coefficient a0 is: -0.838954678478241

The AR coefficient a1 is: -0.08763433884391858

The MA coefficient a0 is: -0.9932258638734274

Final Parameters are: [-0.83895468 -0.08763434 -0.99322586]

Standard deviation of the parameter estimates: [5.83035770e-04 5.79595597e-04 1.15803663e-05]

Covariance Matrix: [[5.83035770e-04, -5.12390906e-04 , 1.24511161e-05]

[-5.12390906e-04 5.79595597e-04 1.05254365e-05]

[1.24511161e-05 1.05254365e-05 1.15803663e-05]]

estimated variance of error: 1.0541016873838474

Confidence Interval for Estimated parameters

-0.8872469457515866 < a1 < -0.7906624112048954

-0.13578392232901404 < a2 < -0.03948475535882311

-1.0000318513145885 < b1 < -0.9864198764322663

There is no zero/pole cancellation.
Zeros : [0.99322586]

Poles : [0.93289294 -0.09393826]

The derived model is an unbiased estimator since the mean of the residual errors is zero and variance is 1 which represents a normal distribution.

mean of residual errors: -0.0269442522417055

variance of residual error: 1.0518099047554024

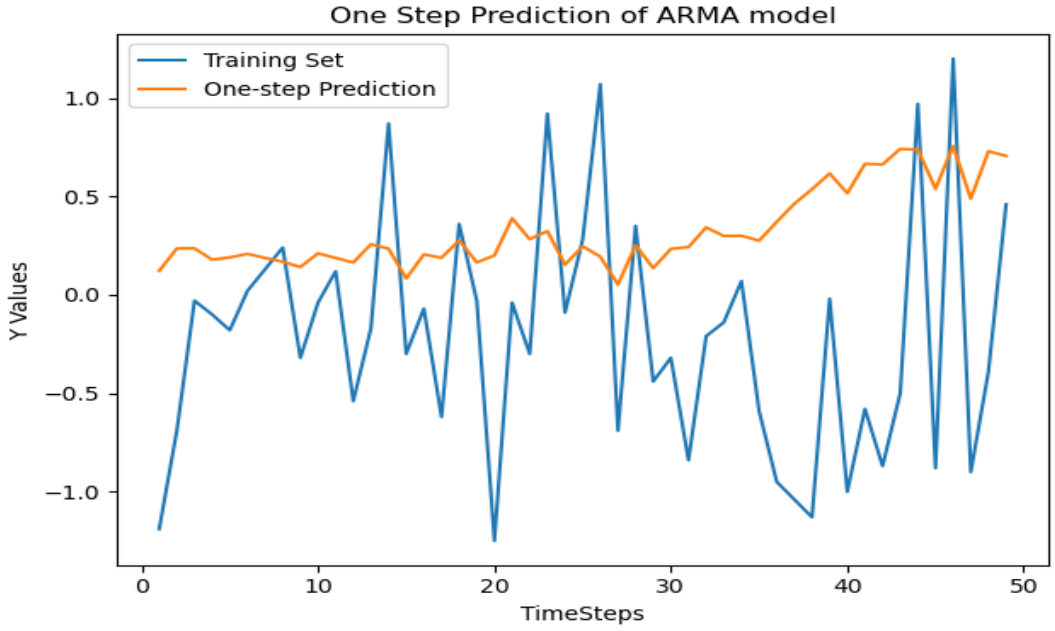
Then we calculated the calculated Q value on the residual errors and performed chi-square test and observed that the Q-value is greater than the critical value, so the residuals are not white.
Q-value: 408.9083372774238

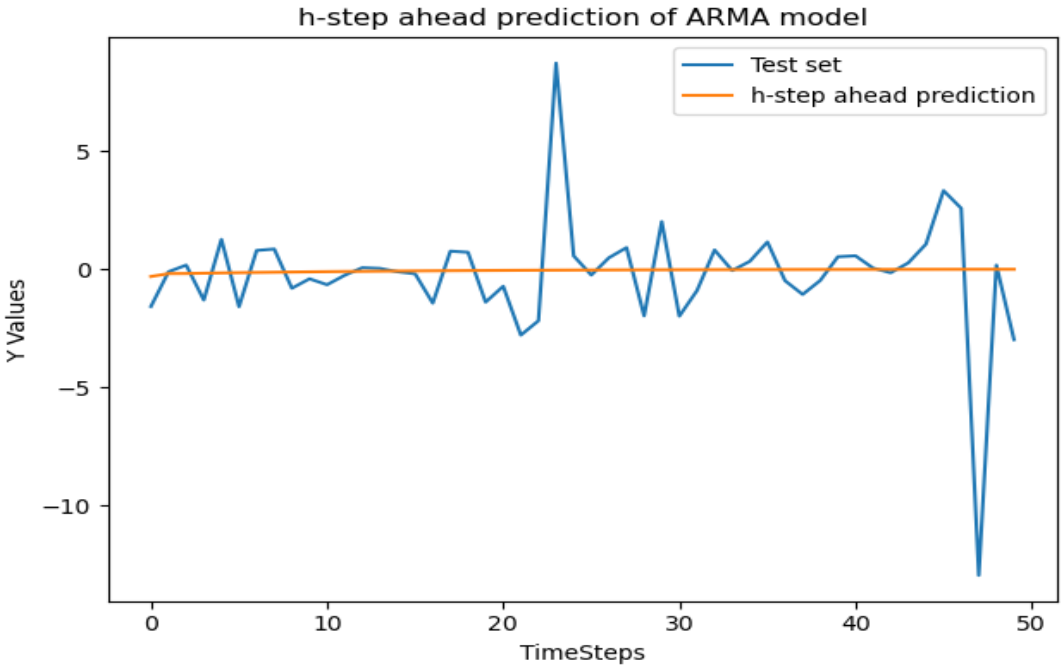
Chi critical: 46.962942124751436

Variance of forecast error: 1.8429835931063132

We can observe that there is very less difference between the variance of the residual and forecast errors.

Below we can observe from the plot of one-step and h-step prediction that ARMA model is not the best fit for the data. I tried different orders of ARMA model, but it did not pass the chi-square test. Also, the data is more non-linear, so ARMA model is not a good fit for our data. Next, we will look into SARIMA model.





SARIMA:

Below are the SARIMA model with a non-seasonal order of (2,1,1) and seasonal order of (0,1,1,24). I have also tried the non-seasonal order of (1,1,1) and seasonal order of (0,1,1,24) that was observed from the ACF and PACF plots. But the prior one gave better AIC and BIC, MSE values. We can observe from the below results that p-value of all the coefficients are significant. Also, the residuals are white which can be observed from the probability value of Q of Ljung Box test which is greater than 0.05 which means that the residual errors are not autocorrelated.

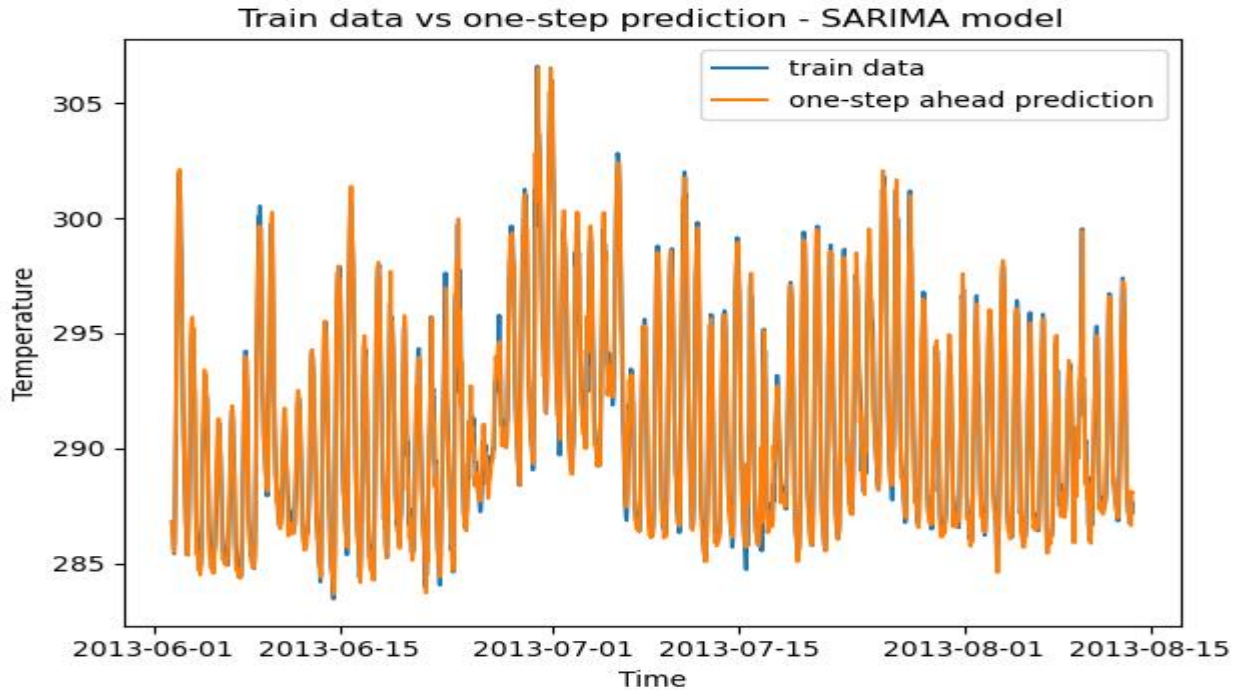
SARIMAX Results

Dep. Variable:	Temperature	No. Observations:	1766
Model:	SARIMAX(2, 1, 1)x(0, 1, 1, 24)	Log Likelihood	-2129.949
Date:	Thu, 17 Dec 2020	AIC	4269.898
Time:	17:38:09	BIC	4297.210
Sample:	06-01-2013 - 08-13-2013	HQIC	4279.997
Covariance Type:	opg		

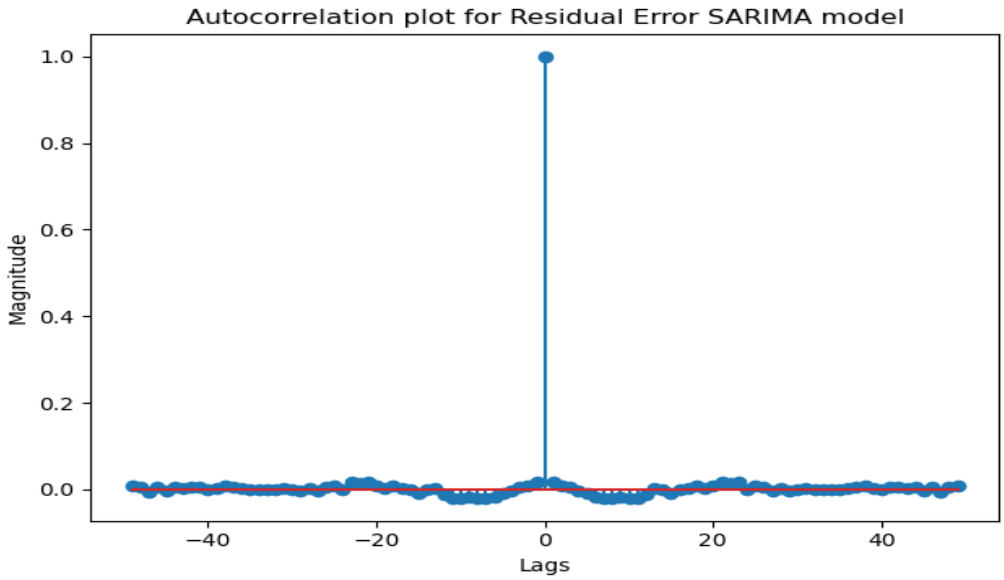
	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-0.4862	0.124	-3.934	0.000	-0.728	-0.244
ar.L2	0.0883	0.017	5.264	0.000	0.055	0.121
ma.L1	0.4889	0.124	3.935	0.000	0.245	0.732
ma.S.L24	-0.9473	0.010	-91.159	0.000	-0.968	-0.927
sigma2	0.6554	0.011	61.925	0.000	0.635	0.676

Ljung-Box (L1) (Q):	0.01	Jarque-Bera (JB):	6504.04
Prob(Q):	0.94	Prob(JB):	0.00
Heteroskedasticity (H):	0.88	Skew:	-0.38
Prob(H) (two-sided):	0.14	Kurtosis:	12.44

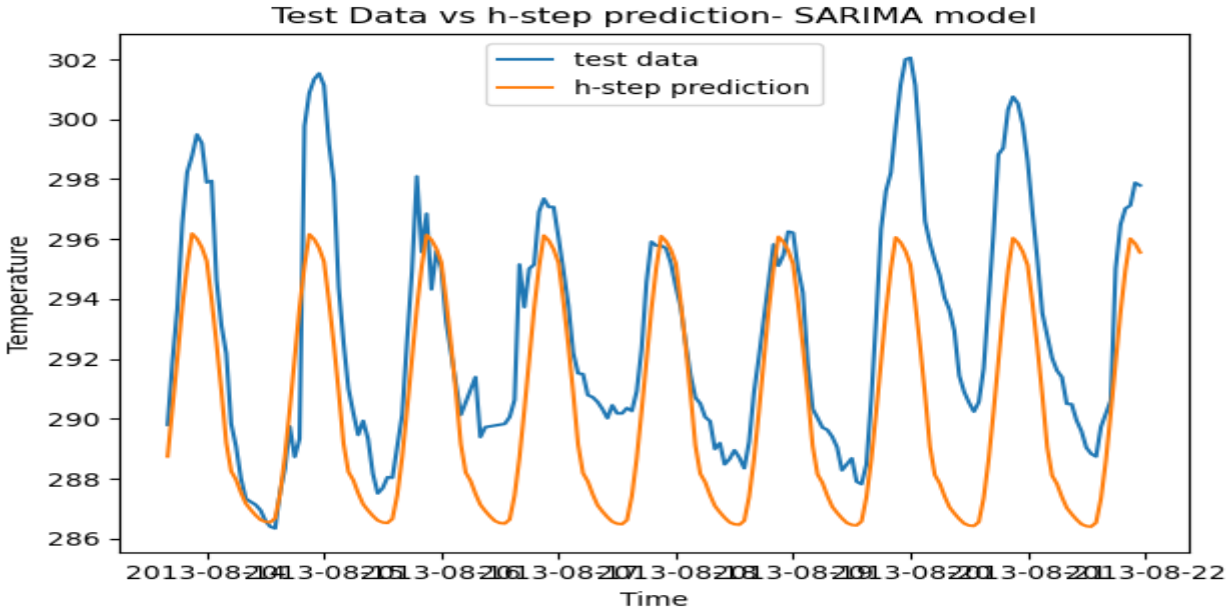
Below is the plot of one step prediction versus the train data. From the plot, we observe that the model is fitting the train data really well.



Next, we have plotted the ACF plot for residual errors which shows that the residual errors are white and all the coefficients of the lagged values are decaying to zero.



Next, we have done the h-step prediction and plotted it versus the test data, and we can see the model is a good fit on the test data as well.



Below are the results of MSE, variance of residual error and forecast error and Q-value of the residual error using box-pierce test.

The variance of the forecast error is less compared to the variance of residual error. But these values are not less than Holt's winter seasonal method.

variance of residual error: 13.058152039196232

variance of forecast error: 3.8233122233559063

We can observe that Q-value: 7.6976091952148655 is less than the chi-critical value 72.44330737654823, so we can say that residuals are white. The Q-value is much better than the Q-value of Holt's winter seasonal method.

We can observe that the MSE values of residual errors and forecast errors are less but not less than the Holt's winter seasonal method.

SARIMA(2,1,1)(0,1,1,24) MSE Residual Error: 13.065533104769958

SARIMA(2,1,1)(0,1,1,24) MSE forecast Error: 10.105503214153503

SARIMA(2,1,1)(0,1,1,24) RMSE forecast Error: 3.1789154147528844

Final Model Selection:

Overall, from the above analysis, we can conclude that Holt's winter seasonal model is the better model by looking the mean square error of residual errors and forecast errors compared to SARIMA and other models. Also, the difference between the variance of the residual errors and the forecast errors is less for the Holt's winter seasonal compared to SARIMA and other models. But the limitation of Holt's winter model is that the Q-value is larger compared to the SARIMA. But by looking at the Autocorrelation plots of the residual errors, we can observe that the residual errors are white for both Holt's winter seasonal and the SARIMA models and for the other models, residuals are not white. But by looking at the plots between one-step prediction versus the train set and h-step prediction and the test set, we can say that we can use both Holt's winter seasonal and SARIMA models for forecasting the hourly temperature values of Dan Francisco state.

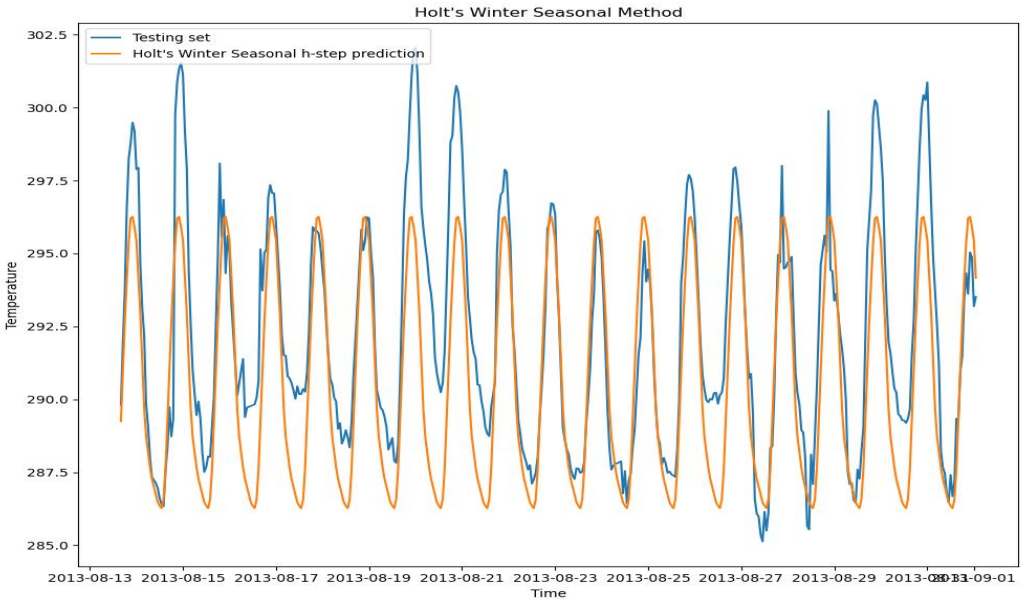
Forecast function:

Holt's winter seasonal method:

```
import statsmodels.tsa.holtwinters as ets
holtw_fitted_model = ets.ExponentialSmoothing(train, trend='add', damped=True, seasonal='mul',
seasonal_periods=24).fit()
holtw_train_pred = holtw_fitted_model.fittedvalues
holtw_test_forecast = holtw_fitted_model.forecast(steps=len(test))
holtw_test_forecast = pd.DataFrame(holtw_test_forecast).set_index(test.index)
```

h-step predictions:

Holt's winter Seasonal Method Plot



From the above h-step prediction versus the test data plot, we observe that Holt's winter seasonal is doing better on the test data.

Summary and Conclusion:

Overall, we concluded that Holt's winter seasonal is performing better compared to other models. I have taken an extra step by looking at the SARIMA model which also gave good results. Only limitation with the Holt's winter seasonal is the Q-value is very large compared to SARIMA. But the ACF plot of the residual errors showed that they are white for both Holt's winter and SARIMA. I think we can use transfer functions to improve the performance of the model.

Appendix:

Code:

Temperature_Prediction.py

```
import numpy as np
import pandas as pd
import seaborn as sns
from scipy.stats import chi2
from sklearn.model_selection import train_test_split
from Final_Project.Toolbox import avg_method, naive_method, drift_method, ses, plot_func,
plot_acf, cal_auto_corr, adf_cal, correlation_coefficent_cal, cal_moving_average, LR_plot_fun,
cal_Q_value, Cal_GPAC, plot_gpac, step3, confidence_interval, zeros_poles, chi_square_test,
sse_plot
from statsmodels.tsa.seasonal import STL
import matplotlib.pyplot as plt
import statsmodels.tsa.holtwinters as ets
import statsmodels.api as sm
from statsmodels.tsa.api import SimpleExpSmoothing
from statsmodels.tsa.statespace.sarimax import SARIMAX
from scipy.stats import chi2
from sklearn.metrics import mean_squared_error
from statsmodels.tools.eval_measures import rmse
import warnings
warnings.filterwarnings("ignore")

# The dataset contains ~5 years(October 2012- October 2017) of high temporal resolution (hourly
measurements) data of various weather attributes,
# such as temperature, humidity, air pressure, etc of San Francisco State.
# Independent Variables - datetime, Humidity, Wind Speed, Wind Direction, Pressure, Weather
Description
# datetime - hourly data
# Humidity - amount of water vapor in the air in percentage
# Wind Speed - the rate at which air is moving in a particular area (meters/second).
# Wind Direction - degree of direction
# Pressure - Standard sea level pressure in hectopascals
# Weather Description

# Dependent Variables - Temperature
# Temperature - San Francisco temperature in kelvin scale

# Reading the data
df = pd.read_csv('San Francisco Weather Data Oct2012-Oct2017.csv', index_col='datetime',
parse_dates=True)

# Setting the frequency to hourly
df.index.freq = '1H'

# Let's see the first few rows in the dataset
pd.set_option('display.max_columns', None)
print(df.head())

# More information about the dataset
print(df.info())

# Replacing the Missing values with the previous values
df = df.fillna(method='ffill')
print(df.info())
df = df.drop(columns='Weather Description')
plot_func(df['Temperature'], 'temperature', 'time', 'temperature in Kelvin', 'Historical hourly
Weather data 2012-2017')
lags = 50
sm.graphics.tsa.plot_acf(df['Temperature'], lags=lags, title='Autocorrelation of
temperature(Original Dataset)')
plt.show()
```

```
def temp_pred(df, season):
    print("************************************ {} Results
************************************".format(season))
    temp = df['Temperature']
    print("Number of missing values in temperature variable: {}".format(temp.isna().sum()))
    temp_1 = temp.diff(periods=24)
    temp_1 = temp_1[24:]
    temp_2 = temp_1.diff()
    temp_2 = temp_2[1:]

    # dependent variable versus time
    plot_func(temp, 'temperature', 'time', 'temperature in Kelvin', 'Historical hourly Weather
data 2012-2013')
    plot_func(temp_1, 'temperature', 'time', 'Magnitude', 'Historical hourly Weather data 2012-
2013 (24th diff)')
    plot_func(temp_2, 'temperature', 'time', 'Magnitude', 'Historical hourly Weather data 2012-
2013 (24th+1st diff)')

    # ACF of the dependent variable
    lags = 50
    sm.graphics.tsa.plot_acf(temp, lags=lags, title='Autocorrelation of temperature')
    plt.show()
    sm.graphics.tsa.plot_acf(temp_1, lags=lags, title='Autocorrelation of temperature(24th
diff)')
    plt.show()
    sm.graphics.tsa.plot_acf(temp_2, lags=lags, title='Autocorrelation of temperature(24th+1st
diff)')
    plt.show()
    sm.graphics.tsa.plot_pacf(temp, lags=lags, title='partial correlation of temperature')
    plt.show()
    sm.graphics.tsa.plot_pacf(temp_1, lags=lags, title='partial correlation of temperature(24th
diff)')
    plt.show()
    sm.graphics.tsa.plot_pacf(temp_2, lags=lags, title='partial correlation of
temperature(24th+1st diff)')
    plt.show()

    lags = 240
    acf_1 = sm.graphics.tsa.acf(temp_1, nlags=lags)
    plt.figure()
    plt.stem(range(0,lags+1)[::24], acf_1[::24], use_line_collection=True)
    plt.xlabel('Lags')
    plt.ylabel('Magnitude')
    plt.title('Autocorrelation plot for {} every 24 lags'.format('temperature(24th diff)'))
    plt.show()

    acf_2 = sm.graphics.tsa.acf(temp_2, nlags=lags)
    plt.figure()
    plt.stem(range(0, lags+1)[::24], acf_2[::24], use_line_collection=True)
    plt.xlabel('Lags')
    plt.ylabel('Magnitude')
    plt.title('Autocorrelation plot for {} every 24 lags'.format('temperature(24th+1st diff)'))
    plt.show()

    pacf_1 = sm.graphics.tsa.pacf(temp_1, nlags=lags)
    plt.figure()
    plt.stem(range(0, lags+1)[::24], pacf_1[::24], use_line_collection=True)
    plt.xlabel('Lags')
    plt.ylabel('Magnitude')
    plt.title('PartialAutocorrelation plot for {} every 24 lags'.format('temperature(24th
diff)'))
    plt.show()

    pacf_2 = sm.graphics.tsa.pacf(temp_2, nlags=lags)
```

```
    plt.figure()
    plt.stem(range(0,lags+1)[::24], pacf_2[::24], use_line_collection=True)
    plt.xlabel('Lags')
    plt.ylabel('Magnitude')
    plt.title('PartialAutocorrelation plot for {} every 24 lags'.format('temperature(24th+1st
diff)'))
    plt.show()

    # Correlation Matrix with seaborn heatmap and Pearson's correlation coefficent
    corrMatrix = df.corr()
    ax = sns.heatmap(corrMatrix, vmin=-1, vmax=1, center=0, cmap=sns.diverging_palette(20, 220,
n=200), square=True, annot=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')
    plt.show()

    r_ht = correlation_coefficent_cal(df['Humidity'], df.Temperature)
    print("The correlation coefficient between the Humidity and Temperature is
{:.3f}".format(r_ht))
    r_wst = correlation_coefficent_cal(df['Wind Speed'], df.Temperature)
    print("The correlation coefficient between the Wind Speed and Temperature is
{:.3f}".format(r_wst))
    r_wdt = correlation_coefficent_cal(df['Wind Direction'], df.Temperature)
    print("The correlation coefficient between the Wind Direction and Temperature is
{:.3f}".format(r_wdt))
    r_pt = correlation_coefficent_cal(df['Pressure'], df.Temperature)
    print("The correlation coefficient between the Pressure and Temperature is
{:.3f}".format(r_pt))

    df['Temperature'].plot.hist(bins=20, grid=True, edgecolor='k').autoscale(enable=True,
axis='both', tight=True)
    plt.xlabel('Temperature in Kelvin')
    plt.ylabel('Frequency')
    plt.title('Histogram plot of Temperature distribution')
    plt.show()

    adf_cal(temp)
    adf_cal(temp_1)
    adf_cal(temp_2)

    # Detrending the data using Moving Average method
    detrended_2x4, ma_2x4 = cal_moving_average(temp, ma_order=4, folding_order=2)
    adf_cal(detrended_2x4)

    # Time series decomposition using STL(Seasonal and Trend decomposition using Loess) method
    STL1 = STL(temp)
    res = STL1.fit()
    fig = res.plot()
    plt.show()

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

    plt.figure()
    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()
```

```
    detrended = temp-T
    plt.figure()
    plt.plot(temp, label='Original')
    plt.plot(detrended, label='detrended')
    plt.xlabel('Year')
    plt.ylabel('Magnitude')
    plt.title('Original vs detrended')
    plt.legend()
    plt.show()

    adjusted_seasonal = temp-S
    plt.figure()
    plt.plot(temp, label='Original')
    plt.plot(adjusted_seasonal, label='Seasonally Adjusted')
    plt.xlabel('Year')
    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 Hourly weather dataset is {:.3f}'.format(F))

    FS = np.max([0, 1-np.var(np.array(R))/np.var(np.array(S+R))])
    print('Strength of seasonality for Hourly weather dataset is {:.3f}'.format(FS))


    # Average, Naive, Drift, Simple Exponential Smoothing, Holt's Linear and Holt's winter
Seasonal Methods
    print("--------------Average, Naive, Drift, Simple Exponential Smoothing, Holt's Linear and
Holt's winter Seasonal Methods-----------------")
    train, test = train_test_split(temp, shuffle=False, test_size=0.2)
    train.index.freq = '1H'
    test.index.freq = '1H'
    h = len(test)

    train_pred_avg = []
    for i in range(1,len(train)):
        res = avg_method(train.iloc[0:i])
        train_pred_avg.append(res)

    test_forecast_avg1 = np.ones(len(test)) * avg_method(train)
    test_forecast_avg = pd.DataFrame(test_forecast_avg1).set_index(test.index)
    residual_error_avg = np.array(train[1:]) - np.array(train_pred_avg)
    forecast_error_avg = test - test_forecast_avg1
    MSE_train_avg = np.mean((residual_error_avg)**2)
    MSE_test_avg = np.mean((forecast_error_avg)**2)
    mean_pred_avg = np.mean(residual_error_avg)
    mean_forecast_avg = np.mean(forecast_error_avg)
    print('Mean of prediction errors for Average method: ', mean_pred_avg)
    print('Mean of forecast errors for Average method: ', mean_forecast_avg)
    var_pred_avg = np.var(residual_error_avg)
    var_forecast_avg = np.var(forecast_error_avg)

    naive_train_pred = []
    for i in range(0, len(train)-1):
        res = naive_method(train[i])
        naive_train_pred.append(res)

    res = np.ones(len(test)) * train[-1]
    naive_test_forecast1 = np.ones(len(test)) * res
    naive_test_forecast = pd.DataFrame(naive_test_forecast1).set_index(test.index)
    residual_error_naive = np.array(train[1:]) - np.array(naive_train_pred)
    forecast_error_naive = test - naive_test_forecast1
```

```
    MSE_train_naive = np.mean((residual_error_naive)**2)
    MSE_test_naive = np.mean((forecast_error_naive)**2)
    mean_pred_naive = np.mean(residual_error_naive)
    mean_forecast_naive = np.mean(forecast_error_naive)
    print('Mean of prediction errors for Naive method: ', mean_pred_naive)
    print('Mean of forecast errors for Naive method: ', mean_forecast_naive)
    var_pred_naive = np.var(residual_error_naive)
    var_forecast_naive = np.var(forecast_error_naive)

    drift_train_forecast=[]
    for i in range(1, len(train)):
        if i == 1:
            drift_train_forecast.append(train[0])
        else:
            h = 1
            res = drift_method(train[0:i], h)
            drift_train_forecast.append(res)

    drift_test_forecast1=[]
    for h in range(1, len(test)+1):
        res = drift_method(train, h)
        drift_test_forecast1.append(res)

    drift_test_forecast = pd.DataFrame(drift_test_forecast1).set_index(test.index)
    residual_error_drift = np.array(train[1:]) - np.array(drift_train_forecast)
    forecast_error_drift = np.array(test) - np.array(drift_test_forecast1)
    MSE_train_drift = np.mean((residual_error_drift)**2)
    MSE_test_drift = np.mean((forecast_error_drift)**2)
    mean_pred_drift = np.mean(residual_error_drift)
    mean_forecast_drift = np.mean(forecast_error_drift)
    print('Mean of prediction errors for Drift method: ', mean_pred_drift)
    print('Mean of forecast errors for Drift method: ', mean_forecast_drift)
    var_pred_drift = np.var(residual_error_drift)
    var_forecast_drift = np.var(forecast_error_drift)

    l0 = train[0]
    ses_train_pred = ses(train, 0.50, l0)
    ses_test_forecast1 = np.ones(len(test)) * (0.5*(train[-1]) + (1-0.5)*(ses_train_pred[-1]))
    ses_test_forecast = pd.DataFrame(ses_test_forecast1).set_index(test.index)
    residual_error_ses = np.array(train[1:]) - np.array(ses_train_pred)
    forecast_error_ses = np.array(test) - np.array(ses_test_forecast1)
    MSE_train_SES = np.mean((residual_error_ses)**2)
    MSE_test_SES = np.mean((forecast_error_ses)**2)
    mean_pred_SES = np.mean(residual_error_ses)
    mean_forecast_SES = np.mean(forecast_error_ses)
    print('Mean of prediction errors for SES method: ', mean_pred_SES)
    print('Mean of forecast errors for SES method: ', mean_forecast_SES)
    var_pred_SES = np.var(residual_error_ses)
    var_forecast_SES = np.var(forecast_error_ses)

    # SES Method using statsmodels for alpha=0.5
    # ses_train = train.ewm(alpha=0.5, adjust=False).mean()  # Another way of doing it
    ses_model1 = SimpleExpSmoothing(train)
    ses_fitted_model1 = ses_model1.fit(smoothing_level=0.5, optimized=False)
    ses_train_pred1 = ses_fitted_model1.fittedvalues.shift(-1)
    ses_test_forecast1 = ses_fitted_model1.forecast(steps=len(test))
    ses_test_forecast1 = pd.DataFrame(ses_test_forecast1).set_index(test.index)
    MSE_test_SES1 = np.square(np.subtract(test.values,
np.ndarray.flatten(ses_test_forecast1.values))).mean()

    # Holt's Linear Trend
    holtl_fitted_model = ets.ExponentialSmoothing(train, trend='additive', damped=True,
seasonal=None).fit()
    holtl_train_pred = holtl_fitted_model.fittedvalues
    holtl_test_forecast = holtl_fitted_model.forecast(steps=len(test))
```

```
    holtl_test_forecast = pd.DataFrame(holtl_test_forecast).set_index(test.index)
    residual_error_holtl = np.subtract(train.values,
np.ndarray.flatten(holtl_train_pred.values))
    forecast_error_holtl = np.subtract(test.values,
np.ndarray.flatten(holtl_test_forecast.values))
    MSE_train_holtl = np.mean((residual_error_holtl)**2)
    MSE_test_holtl = np.mean((forecast_error_holtl)**2)
    mean_pred_holtl = np.mean(residual_error_holtl)
    mean_forecast_holtl = np.mean(forecast_error_holtl)
    print("Mean of prediction errors for Holt's Linear method: ", mean_pred_holtl)
    print("Mean of forecast errors for Holt's Linear method: ", mean_forecast_holtl)
    var_pred_holtl = np.var(residual_error_holtl)
    var_forecast_holtl = np.var(forecast_error_holtl)

    # Holt's Winter Seasonal Trend
    holtw_fitted_model = ets.ExponentialSmoothing(train, trend='add', damped=True,
seasonal='mul', seasonal_periods=24).fit()
    holtw_train_pred = holtw_fitted_model.fittedvalues
    holtw_test_forecast = holtw_fitted_model.forecast(steps=len(test))
    holtw_test_forecast = pd.DataFrame(holtw_test_forecast).set_index(test.index)
    residual_error_holtw = np.subtract(train.values,
np.ndarray.flatten(holtw_train_pred.values))
    forecast_error_holtw = np.subtract(test.values,
np.ndarray.flatten(holtw_test_forecast.values))
    MSE_train_holtw = np.mean((residual_error_holtw)**2)
    MSE_test_holtw = np.mean((forecast_error_holtw)**2)
    mean_pred_holtw = np.mean(residual_error_holtw)
    mean_forecast_holtw = np.mean(forecast_error_holtw)
    print("Mean of prediction errors for Holt's Winter Seasonal method: ", mean_pred_holtw)
    print("Mean of forecast errors for Holt's Winter Seasonal method: ", mean_forecast_holtw)
    var_pred_holtw = np.var(residual_error_holtw)
    var_forecast_holtw = np.var(forecast_error_holtw)

    fig, ax = plt.subplots(figsize=(10,8))
    ax.plot(train, label='Training set')
    ax.plot(test, label='Testing set')
    ax.plot(test_forecast_avg, label='Average h-step prediction')
    plt.xlabel('Time')
    plt.ylabel('Temperature')
    plt.title('Average Method')
    plt.legend(loc='upper left')
    plt.show()

    fig, ax = plt.subplots(figsize=(10,8))
    ax.plot(train, label='Training set')
    ax.plot(test, label='Testing set')
    ax.plot(naive_test_forecast, label='Naive h-step prediction')
    plt.xlabel('Time')
    plt.ylabel('Temperature')
    plt.title('Naive Method')
    plt.legend(loc='upper left')
    plt.show()

    fig, ax = plt.subplots(figsize=(10,8))
    ax.plot(train, label='Training set')
    ax.plot(test, label='Testing set')
    ax.plot(drift_test_forecast, label='Drift h-step prediction')
    plt.xlabel('Time')
    plt.ylabel('Temperature')
    plt.title('Drift Method')
    plt.legend(loc='upper left')
    plt.show()

    fig, ax = plt.subplots(figsize=(10,8))
    ax.plot(train, label='Training set')
```

```
ax.plot(test, label='Testing set')
ax.plot(ses_test_forecast, label='Simple Exponential Smoothing h-step prediction')
plt.xlabel('Time')
plt.ylabel('Temperature')
plt.title('SES Method')
plt.legend(loc='upper left')
plt.show()

fig, ax = plt.subplots(figsize=(10,8))
ax.plot(train, label='Training set')
ax.plot(test, label='Testing set')
ax.plot(holtl_test_forecast, label="Holt's Linear h-step prediction")
plt.xlabel('Time')
plt.ylabel('Temperature')
plt.title("Holt's Linear Method")
plt.legend(loc='upper left')
plt.show()

fig, ax = plt.subplots(figsize=(10, 8))
ax.plot(train, label='Training set')
ax.plot(test, label='Testing set')
ax.plot(holtw_test_forecast, label="Holt's Winter Seasonal h-step prediction")
plt.xlabel('Time')
plt.ylabel('Temperature')
plt.title("Holt's Winter Seasonal Method")
plt.legend(loc='upper left')
plt.show()

fig, ax = plt.subplots(figsize=(10,8))
# ax.plot(train, label='Training set')
ax.plot(test, label='Testing set')
ax.plot(holtw_test_forecast, label="Holt's Winter Seasonal h-step prediction")
plt.xlabel('Time')
plt.ylabel('Temperature')
plt.title("Holt's Winter Seasonal Method")
plt.legend(loc='upper left')
plt.show()

# Auto_correlation for Residual errors and Q value for Residual errors
#Average Method
k = len(test)
lags = 30
avg_residual_acf = cal_auto_corr(residual_error_avg, lags)
Q_residual_avg = k * np.sum(np.array(avg_residual_acf[lags:])**2)
plt.figure()
plt.stem(range(-(lags-1),lags), avg_residual_acf, use_line_collection=True)
plt.xlabel('Lags')
plt.ylabel('Magnitude')
plt.title('Autocorrelation plot for Residual Error (Average Method)')
plt.show()

# Naive method
k = len(test)
lags = 30
naive_residual_acf = cal_auto_corr(residual_error_naive, lags)
Q_residual_naive = k * np.sum(np.array(naive_residual_acf[lags:])**2)
plt.figure()
plt.stem(range(-(lags-1),lags), naive_residual_acf, use_line_collection=True)
plt.xlabel('Lags')
plt.ylabel('Magnitude')
plt.title('Autocorrelation plot for Residual Error (Naive Method)')
plt.show()

# Drift Method
k = len(test)
```

```
    lags = 30
    drift_residual_acf = cal_auto_corr(residual_error_drift, lags)
    Q_residual_drift = k * np.sum(np.array(drift_residual_acf[lags:])**2)
    plt.figure()
    plt.stem(range(-(lags-1),lags), drift_residual_acf, use_line_collection=True)
    plt.xlabel('Lags')
    plt.ylabel('Magnitude')
    plt.title('Autocorrelation plot for Residual Error (Drift Method)')
    plt.show()

    # SES method
    k = len(test)
    lags = 30
    ses_residual_acf = cal_auto_corr(residual_error_ses, lags)
    Q_residual_SES = k * np.sum(np.array(ses_residual_acf[lags:])**2)
    plt.figure()
    plt.stem(range(-(lags-1), lags), ses_residual_acf, use_line_collection=True)
    plt.xlabel('Lags')
    plt.ylabel('Magnitude')
    plt.title('Autocorrelation plot for Residual Error (SES Method)')
    plt.show()

    # holt's linear method
    k = len(test)
    lags = 30
    holtl_residual_acf = cal_auto_corr(residual_error_holtl, lags)
    Q_residual_holtl = k * np.sum(np.array(holtl_residual_acf[lags:])**2)
    plt.figure()
    plt.stem(range(-(lags-1), lags), holtl_residual_acf, use_line_collection=True)
    plt.xlabel('Lags')
    plt.ylabel('Magnitude')
    plt.title("Autocorrelation plot for Residual Error (Holt's Linear Method)")
    plt.show()

    # holt's Winter Seasonal method
    k = len(train)
    lags = 30
    holtw_residual_acf = cal_auto_corr(residual_error_holtw, lags)
    Q_residual_holtw = k * np.sum(np.array(holtw_residual_acf[lags:])**2)
    print("Q-value of Residual error for Holts winter method: {}".format(Q_residual_holtw))
    plt.figure()
    plt.stem(range(-(lags-1), lags), holtw_residual_acf, use_line_collection=True)
    plt.xlabel('Lags')
    plt.ylabel('Magnitude')
    plt.title("Autocorrelation plot for Residual Error (Holt's winter Seasonal Method)")
    plt.show()

    sm.graphics.tsa.plot_acf(holtw_residual_acf, lags=lags, title="Autocorrelation for Residual
Error (Holt's winter Seasonal Method)")
    plt.show()

    corr_avg = correlation_coefficent_cal(forecast_error_avg, test)
    corr_naive = correlation_coefficent_cal(forecast_error_naive, test)
    corr_drift = correlation_coefficent_cal(forecast_error_drift, test)
    corr_ses = correlation_coefficent_cal(forecast_error_ses, test)
    corr_holtl = correlation_coefficent_cal(forecast_error_holtl, test)
    corr_holtw = correlation_coefficent_cal(forecast_error_holtw, test)
    d = {'Methods':['Average', 'Naive', 'Drift', 'SES', "HoltL", "HoltW"],
         'Q_val': [round(Q_residual_avg, 2), round(Q_residual_naive, 2),
round(Q_residual_drift,2), round(Q_residual_SES,2), round(Q_residual_holtl,2),
round(Q_residual_holtw,2)],
         'MSE(P)': [round(MSE_train_avg,2), round(MSE_train_naive,2), round(MSE_train_drift,2),
round(MSE_train_SES,2), round(MSE_train_holtl,2), round(MSE_train_holtw,2)],
         'MSE(F)': [round(MSE_test_avg,2), round(MSE_test_naive,2), round(MSE_test_drift,2),
round(MSE_test_SES,2), round(MSE_test_holtl,2), round(MSE_test_holtw,2)],
```

```
        'var(P)': [round(var_pred_avg,2), round(var_pred_naive,2), round(var_pred_drift,2),
round(var_pred_SES,2), round(var_pred_holtl,2), round(var_pred_holtw,2)],
        'var(F)':[round(var_forecast_avg,2), round(var_forecast_naive,2),
round(var_forecast_drift,2), round(var_forecast_SES,2), round(var_forecast_holtl,2),
round(var_forecast_holtw,2)],
        'corrcoeff':[round(corr_avg,2), round(corr_naive,2), round(corr_drift,2),
round(corr_ses,2), round(corr_holtl,2), round(corr_holtw,2)]}
    df1 = pd.DataFrame(data=d)
    df1 = df1.set_index('Methods')
    pd.set_option('display.max_columns', None)
    print(df1)


    # Forward step regression
    df2 = df[['Humidity', 'Temperature']]
    features = df2.drop(columns='Temperature')
    target = df2['Temperature']
    features = sm.add_constant(features)
    x_train, x_test, y_train, y_test = train_test_split(features, target, shuffle=False,
test_size=0.2)
    model = sm.OLS(y_train, x_train).fit()
    print(model.summary())

    df2 = df[['Humidity', 'Wind Speed', 'Temperature']]
    features = df2.drop(columns='Temperature')
    target = df2['Temperature']
    features = sm.add_constant(features)
    x_train, x_test, y_train, y_test = train_test_split(features, target, shuffle=False,
test_size=0.2)
    model = sm.OLS(y_train, x_train).fit()
    print(model.summary())

    df2 = df[['Humidity', 'Wind Speed', 'Wind Direction', 'Temperature']]
    features = df2.drop(columns='Temperature')
    target = df2['Temperature']
    features = sm.add_constant(features)
    x_train, x_test, y_train, y_test = train_test_split(features, target, shuffle=False,
test_size=0.2)
    model = sm.OLS(y_train, x_train).fit()
    print(model.summary())

    df2 = df[['Humidity', 'Wind Speed', 'Wind Direction', 'Pressure', 'Temperature']]
    features = df2.drop(columns='Temperature')
    target = df2['Temperature']
    features = sm.add_constant(features)
    x_train, x_test, y_train, y_test = train_test_split(features, target, shuffle=False,
test_size=0.2)
    model = sm.OLS(y_train, x_train).fit()
    print(model.summary())

    df2 = df[['Humidity', 'Wind Speed', 'Wind Direction', 'Temperature']]
    features = df2.drop(columns='Temperature')
    target = df2['Temperature']
    features = sm.add_constant(features)
    x_train, x_test, y_train, y_test = train_test_split(features, target, shuffle=False,
test_size=0.2)
    model = sm.OLS(y_train, x_train).fit()
    print(model.summary())

    # Backward step regression
    df2 = df[['Humidity', 'Wind Speed', 'Wind Direction', 'Pressure', 'Temperature']]
    features = df2.drop(columns='Temperature')
    target = df2['Temperature']
    features = sm.add_constant(features)
    x_train, x_test, y_train, y_test = train_test_split(features, target, shuffle=False,
```

```
test_size=0.2)
    model = sm.OLS(y_train, x_train).fit()
    print(model.summary())

    df2 = df[['Humidity', 'Wind Speed', 'Wind Direction', 'Temperature']]
    features = df2.drop(columns='Temperature')
    target = df2['Temperature']
    features = sm.add_constant(features)
    x_train, x_test, y_train, y_test = train_test_split(features, target, shuffle=False,
test_size=0.2)
    model = sm.OLS(y_train, x_train).fit()
    print(model.summary())

    # 1-step ahead prediction
    y_hat_OLS = model.predict(x_train)
    y_test_hat_OLS = model.predict(x_test)
    LR_plot_fun(y_train, y_test, y_hat_OLS, y_test_hat_OLS, 'OLS Regression Method')

    prediction_error = y_train - y_hat_OLS
    forecast_error = y_test - y_test_hat_OLS
    lags = 30
    prediction_error_acf = cal_auto_corr(prediction_error, lags)
    forecast_error_acf = cal_auto_corr(forecast_error, lags)
    plot_acf(prediction_error_acf, lags=lags, var_name='OLS prediction error')
    plot_acf(forecast_error_acf, lags=lags, var_name='OLS forecast error')
    Q_value = cal_Q_value(prediction_error, prediction_error_acf, lags)

    T = len(x_train)
    K = len(x_train.columns)
    pred_var = (1/(T-K-1)) * (np.sum((prediction_error)**2))
    pred_std = np.sqrt((1/(T-K-1)) * (np.sum((prediction_error)**2)))
    print("Q value of the residual error: {:.2f}".format(Q_value))
    print("mean of prediction error: {:.2f}".format(np.mean(prediction_error)))
    print("variance of prediction error: ", pred_var)
    print("standard deviation of prediction error: ", pred_std)
    print("RMSE of prediction error: ", np.sqrt(np.mean(prediction_error**2)))

    T = len(x_test)
    K = len(x_test.columns)
    forecast_var = (1/(T-K-1)) * (np.sum((forecast_error)**2))
    forecast_std = np.sqrt((1/(T-K-1)) * (np.sum((forecast_error)**2)))
    print("mean of forecast error: {:.2f}".format(np.mean(forecast_error)))
    print("variance of forecast error: ", forecast_var)
    print("standard deviation of forecast error: ", forecast_std)
    print("RMSE of forecast error: ", np.sqrt(np.mean(forecast_error**2)))

    corr_coeff = round(correlation_coefficent_cal(y_test, y_test_hat_OLS),2)
    plt.figure(figsize=(8, 6))
    plt.scatter(y_test, y_test_hat_OLS, c='green', alpha=1, label='y_test vs y_test_hat_OLS')
    plt.xlabel('y_test')
    plt.ylabel('y_test_hat_OLS')
    plt.title("Scatter plot of y_test vs y_hat_test with correlation coefficient of
{}".format(corr_coeff))
    plt.legend()
    plt.show()

    corr_coeff1 = round(correlation_coefficent_cal(y_train, y_hat_OLS),2)
    plt.figure(figsize=(8, 6))
    plt.scatter(y_train, y_hat_OLS, c='green', alpha=1, label='y_test vs y_test_hat_OLS')
    plt.xlabel('y_train')
    plt.ylabel('y_hat_OLS')
    plt.title("Scatter plot of y_train vs y_hat_OLS with correlation coefficient of
{}".format(corr_coeff1))
    plt.legend()
    plt.show()
```

```
    return temp, temp_1, temp_2


winter_data = df[1453:3613]
spring_data = df[3613:5821]
summer_data = df[5821:8029]
fall_data = df[8029:10213]

df = summer_data
temp, temp_1, temp_2 = temp_pred(df, "summer")

# GPAC
y = temp_2.copy()
y_train, y_test = train_test_split(y, shuffle=False, test_size=0.2)
j=8; k=8
lags = 30
T = len(y_train)
ry = cal_auto_corr(y_train, lags)
plot_acf(ry, lags=lags, var_name='temperature(2nd Diff)')
gpac_table = Cal_GPAC(ry[lags-1:], j=8, k=8)
data_frame = pd.DataFrame(gpac_table)
data_frame.columns = np.arange(1, k+1)
plot_gpac(data_frame, T=T)

print("potential order of AR and MA is 2 and 1 respectively")

# LM Algorithm
na = 2
nb = 1

# ARMA parameter Estimation
theta_hat, var_error, covariance_theta, sse_list = step3(y_train, na, nb)
an = []
bn = []
for i in range(na):
    an.append(theta_hat[i])
    print('The AR coefficient a{}'.format(i), 'is: ', theta_hat[i])
for i in range(nb):
    bn.append(theta_hat[i+na])
    print('The MA coefficient a{}'.format(i), 'is: ', theta_hat[i+na])

print("Final Parameters are: {}".format(theta_hat))
print("Standard deviation of the parameter estimates:{}".format(covariance_theta.diagonal()))
print("Covariance Matrix: {}".format(covariance_theta))
print("estimated variance of error: {}".format(var_error))
confidence_interval(covariance_theta, na, nb, theta_hat)
zeros, poles = zeros_poles(theta_hat, na, nb)
sse_plot(sse_list, title='y(t) - 0.83y(t - 1) -0.08y(t-1) = e(t) -0.99 (t-1) SSE Plot')

# 1- step prediction
y_hat_t_1 = []
for i in range(0, len(y_train)):
    if i == 0:
        y_hat_t_1.append(-theta_hat[0] * y_train[i] + theta_hat[2]*y_train[i])
    else:
        y_hat_t_1.append(-theta_hat[0] * y_train[i] - theta_hat[1] * y_train[i-1] +
theta_hat[2]*(y_train[i] - y_hat_t_1[i - 1]))

x = [i for i in range(50)]
plt.figure()
plt.plot(x[1:], y_train[1:50], label="Training Set")
plt.plot(x[1:], y_hat_t_1[:49], label="One-step Prediction")
plt.xlabel("TimeSteps")
plt.ylabel("Y Values")
plt.legend()
```

```
plt.title("One Step Prediction of ARMA model")
plt.show()

residual_errors = np.subtract(y_train[1:], y_hat_t_1[:-1])
print('mean of residual errors: ', np.mean(residual_errors))
print('variance of residual error : ', np.var(residual_errors))
k = len(y_train)-1
residual_error_acf = cal_auto_corr(residual_errors, lags)
Q_value = k * np.sum(np.array(residual_error_acf[lags:])**2)
print("Q-value: {}".format(Q_value))
chi_square_test(Q_value, lags, na, nb)

# h-step ahead prediction
y_hat_t_h = []
for h in range(len(y_test)):
    if h == 0:
        y_hat_t_h.append(-theta_hat[0] * y_train[h-1] - theta_hat[1] * y_train[h-2]+
theta_hat[2]*(y_train[h-1] - y_hat_t_1[h-2]))
    elif h==1:
        y_hat_t_h.append(-theta_hat[0] * y_hat_t_h[h-1] - theta_hat[1] * y_train[h-2])
    else:
        y_hat_t_h.append(-theta_hat[0] * y_hat_t_h[h - 1] - theta_hat[1] * y_hat_t_h[h-2])

forecast_error = np.subtract(y_test, y_hat_t_h)
print('Variance of forecast error : ', np.var(forecast_error))

plt.figure()
plt.plot(x, y_test[:50], label='Test set')
plt.plot(x, y_hat_t_h[:50], label='h-step ahead prediction')
plt.xlabel("TimeSteps")
plt.ylabel("Y Values")
plt.title('h-step ahead prediction of ARMA model')
plt.legend()
plt.show()

# SARIMA Model

train, test = train_test_split(temp, shuffle=False, test_size=0.2)
train.index.freq = '1H'
test.index.freq = '1H'
model = SARIMAX(train, order=(2,1,1), seasonal_order=(0,1,1,24))
results = model.fit()
print(results.summary())
# one-step prediction
one_step = results.fittedvalues[1:]
sarima_residual_errors = np.subtract(train[1:], one_step)
sarima_residual_mse = mean_squared_error(train[1:], one_step)
print('SARIMA(2,1,1)(0,1,1,24) MSE Residual Error: {}'.format(sarima_residual_mse))
sarima_mean_residual_error = np.mean(sarima_residual_errors)
sarima_var_residual_error = np.mean(sarima_residual_errors)
print('variance of residual error : ', np.var(sarima_residual_errors))

# Plot of one-step prediction against train values
title = 'Train data vs one-step prediction - SARIMA model'
ylabel='Temperature'
xlabel='Time'
plt.figure()
plt.plot(train[31:], label='train data')
plt.plot(one_step[30:], label='one-step ahead prediction')
plt.xlabel(xlabel)
plt.ylabel(ylabel)
plt.title(title)
plt.legend()
plt.show()
```

```
def chi_square_test1(Q_value, lags, na, nb, Na, Nb,alpha=0.01):
    DOF = lags - na - nb-Na-Nb
    chi_critical = chi2.ppf(1-alpha, DOF)
    print("chi_critical: {}".format(chi_critical))
    if Q_value < chi_critical:
        print("The Residuals are White")
    else:
        print("Residuals are not White")

k = len(train)-1
lags = 50
na = 1
nb = 1
Na = 0
Nb = 1
residual_error_acf = cal_auto_corr(sarima_residual_errors, lags)
Q_value = k * np.sum(np.array(residual_error_acf[lags:])**2)
print("Q-value: {}".format(Q_value))
chi_square_test1(Q_value, lags, na, nb, Na, Nb)
plt.figure()
plt.stem(range(-(lags - 1), lags), residual_error_acf, use_line_collection=True)
plt.xlabel('Lags')
plt.ylabel('Magnitude')
plt.title('Autocorrelation plot for Residual Error SARIMA model')
plt.show()

# Obtain predicted values
start=len(train)
end=len(train)+len(test[:200])-1
predictions = results.predict(start=start, end=end, dynamic=False,
typ='levels').rename('SARIMA(2,1,1)(0,1,1,24) Predictions')
sarima_forecast_errors = np.subtract(test, predictions)
sarima_mean_forecast_error = np.mean(sarima_forecast_errors)
sarima_var_forecast_error = np.var(sarima_forecast_errors)
print('variance of forecast error : ', np.var(sarima_forecast_errors))

# h-step predictions against known values
title = 'Test Data vs h-step prediction- SARIMA model'
ylabel='Temperature'
xlabel='Time'
plt.figure()
plt.plot(test[:200], label='test data')
plt.plot(predictions, label='h-step prediction')
plt.xlabel(xlabel)
plt.ylabel(ylabel)
plt.title(title)
plt.legend()
plt.show()

sarima_forecast_mse = mean_squared_error(test[:200], predictions)
print('SARIMA(2,1,1)(0,1,1,24) MSE forecast Error: {}'.format(sarima_forecast_mse))
sarima_rmse = rmse(test[:200], predictions)
print('SARIMA(2,1,1)(0,1,1,24) RMSE forecast Error: {}'.format(sarima_rmse))
corr_sarima = correlation_coefficent_cal(test[:200], predictions)
```

Toolbox.py

```
'''
Title: Helper Functions
Author: Akhil Kumar Baipaneni
Created on: 12/06/2020
Description: This file contains the helper functions required for the final project
'''

import numpy as np
import pandas as pd
```

```
import seaborn as sns
import matplotlib.pyplot as plt
import math
from statsmodels.tsa.stattools import adfuller
from scipy.stats import chi2
from scipy import signal


#Average Method
def avg_method(train):
    y_hat_avg = np.mean(train)
    return y_hat_avg


# Naive Method
def naive_method(t):
    return t


# Drift method
def drift_method(t, h):
    y_hat_drift = t[len(t)-1] + h*((t[len(t)-1]-t[0])/(len(t) - 1))
    return y_hat_drift


#SES Method
def ses(t, damping_factor, l0):
    yhat4 = []
    yhat4.append(l0)
    for i in range(1, len(t)-1):
        res = damping_factor*(t[i]) + (1-damping_factor)*(yhat4[i-1])
        yhat4.append(res)
    return yhat4


def plot_func(var, label, x_label, y_label, title):
    plt.figure()
    plt.plot(var, label=label)
    plt.xlabel(x_label)
    plt.ylabel(y_label)
    plt.title(title)
    plt.legend()
    plt.show()


def plot_acf(var_acf, lags, var_name):
    plt.figure()
    plt.stem(range(-(lags-1),lags), var_acf, use_line_collection=True)
    plt.xlabel('Lags')
    plt.ylabel('Magnitude')
    plt.title('Autocorrelation plot for {} variable'.format(var_name))
    plt.show()


def adf_cal(x):
    result = adfuller(x)
    print('ADF Statistic: %f' %result[0])
    print('p-value: %f' %result[1])
    print('Critical Values:')
    for key, value in result[4].items():
        print('\t%s: %.3f' % (key, value))


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
    result = res_num/res_den
    return result


def cal_auto_corr(y, k):
    res = []
    res1 = []
    for t in range(0, k):
        result = auto_corr(y, t)
        res.append(result)
    for t in range(k-1, 0, -1):
        res1.append(res[t])
    res1.extend(res)
    return res1


def correlation_coefficent_cal(x, y):
    result = 0
    cov_res = 0
    var_res1 = 0
    var_res2 = 0
    mean_x = np.mean(x)
    mean_y = np.mean(y)
    if len(x) == len(y):
        for i in range(0, len(x)):
            cov_res += ((x[i]-mean_x)*(y[i]-mean_y))
            var_res1 += (x[i]-mean_x)**2
            var_res2 += (y[i]-mean_y)**2
    result += cov_res/(np.sqrt(var_res1)*np.sqrt(var_res2))
    return result


def plot_ma(y, k, trend, detrend, ma_order, folding_order):
    plt.figure()
    plt.plot(np.array(y.index[:200]), np.array(y[:200]), label='original')
    if ma_order%2 != 0:
        plt.plot(np.array(y.index[k:200]), np.array(trend[:200-k]), label='{}-
MA'.format(ma_order))
        plt.title('Plot for {}-MA'.format(ma_order))
    else:
        plt.plot(np.array(y.index[k:200]), np.array(trend[:200 - k]), label='{}x{}-
MA'.format(folding_order, ma_order))
        plt.title('Plot for {}x{}-MA'.format(folding_order, ma_order))
    plt.plot(np.array(y.index[k:200]), np.array(detrend[:200-k]), label='detrended')
    plt.xlabel('DateTime')
    plt.ylabel('Temperature in Kelvin')
    plt.xticks(rotation=90)
    plt.legend()
    plt.show()


def cal_moving_average(y, ma_order, folding_order):
    ma = []
    k = int(np.ceil((ma_order - 1) / 2))
    for t in range(0, len(y) - ma_order + 1):
        temp = np.sum(y[t:ma_order + t])
        ma.append(temp / ma_order)
```

```
    if folding_order > len(ma):
        print("Invalid Folding order. Moving Average cannot be calculated if folding order is
greater than the length of first moving average result")
    # passing folding order as zero for odd order of moving average
    elif folding_order != 0:
        k1 = int(np.ceil((ma_order - 1) / 2) + ((folding_order - 1) / 2))
        folding_ma = []
        for t in range(0, len(ma) - folding_order + 1):
            a = np.sum(y[t:folding_order + t])
            folding_ma.append(a / folding_order)
        print("Result of {}x{}-MA is: {}".format(folding_order, ma_order, folding_ma))
        detrended = np.subtract(list(y.iloc[k1:-k1]), folding_ma)
        plot_ma(y, k1, ma, detrended, ma_order, folding_order)
        return detrended, folding_ma
    else:
        print("Result of {}-MA is: {}".format(ma_order, ma))
        detrended = np.subtract(list(y.iloc[k:-k]), ma)
        plot_ma(y, k, ma, detrended, ma_order, folding_order)
        return detrended, ma

# Function to plot train, test and prediction values
def LR_plot_fun(train, test, y_hat_OLS, y_test_hat_OLS, title):
    year = pd.date_range(start='2012-10-01 13:00:00', end='2017-10-28 00:00:00', freq='1H')
    plt.figure(figsize=(10,8))
    # plt.plot(year[0:len(train)], train, label='Training set')
    # plt.plot(year[0:len(train)], y_hat_OLS, label='Prediction values')
    plt.plot(range(len(train), len(train)+len(test)), test, label='Testing set')
    plt.plot(range(len(train), len(train)+len(y_test_hat_OLS)), y_test_hat_OLS, label='forecast
values')
    plt.xlabel('Time')
    plt.ylabel('Temperature in Kelvin')
    plt.title(title)
    plt.legend(loc='upper left')
    plt.show()


def cal_Q_value(residual_error, residual_error_acf, lags):
    k = len(residual_error)
    return k * np.sum(np.array(residual_error_acf[lags:]) ** 2)


def step0(na, nb):
    theta = np.zeros(shape=(na+nb,1))
    return theta.flatten()


def cal_wn(na, theta, y):
    num = [1] + list(theta[na:])
    den = [1] + list(theta[:na])
    if len(den) < len(num):
        den.extend([0 for i in range(len(num) - len(den))])
    elif len(num) < len(den):
        num.extend([0 for i in range(len(den) - len(num))])
    sys = (den, num, 1)
    _, e = signal.dlsim(sys, y)
    e = [item[0] for item in e]
    return np.array(e)


def step1(na, nb, theta, delta, y):
    X_i = []
    e = cal_wn(na, theta, y)
    sse_old = np.matmul(e.T, e)
    for i in range(na + nb):
        updated_theta = theta.copy()
```

```
        updated_theta[i] = theta[i] + delta
        e_new = cal_wn(na, updated_theta, y)
        x_i = (e - e_new) / delta
        X_i.append(x_i)
    X = np.column_stack(X_i)
    A = np.matmul(X.T, X)
    g = np.matmul(X.T, e)
    return sse_old, A, g


def step2(A, g, mu, na, nb, theta, y):
    mu_I = mu * np.identity(na + nb)
    delta_theta = np.matmul(np.linalg.inv(A + (mu_I)), g)
    theta_new = theta + delta_theta
    e_new = cal_wn(na, theta_new, y)
    sse_new = np.matmul(e_new.T, e_new)
    if np.isnan(sse_new):
        sse_new = 10 ** 10
    return delta_theta, theta_new, sse_new


def step3(y, na, nb):
    mu_factor = 10
    max_iterations = 100
    mu_max = 1e10
    delta = 1e-6
    mu = 0.01
    iterations = 0
    sse_list = []
    epsilon = 1e-3
    theta = step0(na, nb)
    while iterations < max_iterations:
        print("Iteration number: {}".format(iterations))
        sse_old, A, g = step1(na, nb, theta, delta, y)
        print("SSE old: {}".format(sse_old))
        if iterations == 0:
            sse_list.append(sse_old)
        delta_theta, theta_new, sse_new = step2(A, g, mu, na, nb, theta, y)
        print("SSE new: {}".format(sse_new))
        sse_list.append(sse_new)
        if sse_new < sse_old:
            if np.linalg.norm(delta_theta) < epsilon:
                theta_hat = theta_new
                var_error = sse_new/(len(y) - (na+nb))
                covariance_theta = var_error * np.linalg.inv(A)
                print("Algorithm converged")
                return theta_hat, var_error, covariance_theta, sse_list
            else:
                theta = theta_new
                mu /= mu_factor
        while sse_new >= sse_old:
            mu = mu * mu_factor
            if mu > mu_max:
                print("Error. mu has reached it's max value")
                return None, None, None, None
            delta_theta, theta_new, sse_new = step2(A, g, mu, na, nb, theta, y)
        theta = theta_new
        iterations += 1
        if iterations > max_iterations:
            print("Error. Reached maximum iterations")
            return None, None, None, None


def confidence_interval(covariance_theta, na, nb, theta_hat):
    print("Confidence Interval for Estimated parameters")
```

```
    for i in range(na):
        lower = theta_hat[i] - 2 * np.sqrt(covariance_theta[i][i])
        upper = theta_hat[i] + 2 * np.sqrt(covariance_theta[i][i])
        print('{} < a{} < {}'.format(lower, i+1, upper))

    for j in range(nb):
        lower = theta_hat[na+j] - 2 * np.sqrt(covariance_theta[na+j][na+j])
        upper = theta_hat[na+j] + 2 * np.sqrt(covariance_theta[na+j][na+j])
        print('{} < b{} < {}'.format(lower, j+1, upper))


def zeros_poles(theta_hat, na, nb):
    p = [1] + list(theta_hat[:na])
    z = [1] + list(theta_hat[na:])
    poles = np.roots(p)
    zeros = np.roots(z)
    print('Zeros : {}'.format(zeros))
    print('Poles : {}'.format(poles))
    return zeros, poles


def sse_plot(sse_list, title):
    plt.figure()
    plt.plot(sse_list, label = 'sum square error')
    plt.xlabel('# of Iterations')
    plt.ylabel('sum square error')
    plt.title(title + ': Sum Square Error')
    plt.legend()
    plt.show()


def chi_square_test(Q_value, lags, na, nb, alpha=0.01):
    DOF = lags - na - nb
    chi_critical = chi2.ppf(1-alpha, DOF)
    print("chi_critical: {}".format(chi_critical))
    if Q_value < chi_critical:
        print("The Residuals are White")
    else:
        print("Residuals are not White")


def phi_kk(ry, j, k):
    num = np.zeros((k, k), dtype=np.float64)
    den = np.zeros((k, k), dtype=np.float64)
    for b in range(k):
        for a in range(k):
            if b != k-1:
                num[a][b] = ry[np.abs(j + a - b)]
                den[a][b] = ry[np.abs(j + a - b)]
            else:
                num[a][b] = ry[np.abs(j + a + 1)]
                den[a][b] = ry[np.abs(j - k + a + 1)]
    numerator_det = np.round(np.linalg.det(num), 5)
    denominator_det = np.round(np.linalg.det(den), 5)
    phi = math.inf if denominator_det == 0.0 else np.round(np.divide(numerator_det,
denominator_det), 3)
    return phi


def plot_gpac(df, T):
    sns.heatmap(df, annot=True)
    plt.xlabel('AR process (K)')
    plt.ylabel('MA process (j)')
    plt.title('Generalized Partial Autocorrelation(GPAC) ARMA {} samples'.format(T))
    plt.show()
```

```
def Cal_GPAC(ry, j, k):
    gpac_table = np.zeros(shape=(j, k), dtype=np.float64)
    for a in range(j):
        for b in range(1, k+1):
            gpac_table[a][b-1] = phi_kk(ry, a, b)
    return gpac_table
```

References:

IEEE Paper provided by the professor

https://blackboard.gwu.edu/bbcswebdav/pid-1268053-dt-announcement-rid-81612352_2/courses/56367_202003/HaganLoadForecast87.pdf