Course: DATS 6450 – Time Series Analysis & Modeling

Instructor: Dr. Reza Jafari

Final Term Project

Topic: Appliance Energy Prediction

Name: Rehapriadarsini Manikandasamy

RM
Initials

12.15.2021
Date

TABLE OF CONTENTS

Abstract

The project work focuses on performing time series analysis and modeling on a time series data. The data used in the project is recorded based on the amount of energy consumed by appliances in a household for 4.5 months. The project applies all the analysis and preprocessing techniques for a time series data to make it prepared for the modelling phase. Different time series linear models are used to find the best suitable model to predict the appliance energy consumed in a household. The project implementation completely relies on python and the best model for the forecast is selected using different performance measures. The best model for the project was Multiple Linear Regression.

INTRODUCTION

Time series analysis is a specific way of analyzing a sequence of data points collected over an interval of time. In time series analysis, analysts record data points at consistent intervals over a set period rather than just recording the data points intermittently or randomly. Time series analysis typically requires many data points to ensure consistency and reliability. An extensive data set ensures you have a representative sample size, and that analysis can cut through noisy data. It also ensures that any trends or patterns discovered are not outliers and can account for seasonal variance. Additionally, time series data can be used for forecasting—predicting future data based on historical data.

To make better analysis, it is always recommended to have processed and prepared data. The data in the initial stage is visualized using basic information and plot to make sure it is ready to be treated for modelling. The stationarity of the data is tested using rolling mean and variances, ADF test and KPSS test. While calculating the rolling mean and variances, the plot is considered to have flat line to say the data is stationary. ADF and KPSS tests follow their own hypothesis (based on p-value and test statistics) to confirm the stationarity in the data. If it is non-stationary, appropriate differencing techniques is performed to make the data stationary. Differencing can also be performed to remove the seasonality in the data.

A time series data has various components. Separation and analysis of those components is needed to proceed further with good modelling. Decomposition can be done using techniques like SVD, STL to understand the patterns in data. Based on the strength of seasonality and trend, the data needs to be seasonally adjusted or detrended.

One of the primary methods of time series analysis is Holts-Winter method. It can be used on the data with both seasonal and trend component. It tries to capture more aspects in the data. A full model with all variables will not be a good model every time. To find a best model, feature selection is performed based on the p-value of OLS model. The variables with high p-values are correlated and unwanted, they are removed to make a better model. The model with best features is developed using multiple linear regression technique. The multiple linear regression model's performance can be tested using AIC/BIC score, R-squared/adjust R-squared values and condition number. More the R-squared value better is the model. The model with high AIC/BIC and condition number is said to be a bad model.

In the same way, the data is modelled using all the base models to understand whether the data can be forecasted using simpler models rather than moving to complex other linear models. The base models are Average, Naïve, Drift and Simple Exponential Smoothening. These models have different properties in forecasting the data. Other more appropriate way of find better model is using the technique of auto-regressive and moving-average. A better model like ARMA, ARIMA and SARIMA are developed based on these aspects using different combination of these two methods. Out of these models, the best model is selected for the final forecasting.

A detailed description about these techniques is discussed in the next chapter.

METHOD, THEORY AND PROCEDURES

Stationary data:
The data can be called stationary when the statistics of the data doesn't change over the period.

Rolling mean and variance:
Rolling in timeseries means creating a rolling window of specified size and perform statistical calculations on data. Rolling mean and variance works in a way where it calculates the mean and variance of the data under a particular window.

Mean = Sum of observations / Total number of observations

$$\text{Variance} = \frac{(\text{Differences between each number in the data set and the mean})^2}{\text{Number of observations in the dataset}}$$

Considering the above formulas, rolling mean and variance are calculated as follows:
During first iteration, the first sample will load and the mean and variance will be calculated. During the second iteration, the first two samples will load, and their necessary mean and variance will be calculated. In this way the iteration will happen till the last observation is loaded and their mean and variances are calculated.

This will result in the generation of two lists rolling mean and variance at every iteration which can be used to plot against time to understand whether the data is stationary or not. Using these plots, we can say a data is stationary if the mean and variance remains constant over the time.

ADF Test:
The Augmented-Dickey Fuller test is a statistical test or unit root test to test whether the data is stationary or not using the degree of hypothesis. The intuition behind this test to predict how strongly a time series defined by trend.

The ADF test has three parameters in test results: test statistic, p-value, critical values

Hypothesis of ADF test:
Null hypothesis (H_0) – If it is failed to be rejected, the times series has unit root which means the data is non-stationary
Alternate hypothesis (H_1) – If the null hypothesis is rejected, the times series has no unit root which means the data is stationary

To reject the null hypothesis, the test statistic should be more negative, and p-value should be less than the threshold (0.05).

KPSS Test:

KPSS test is another test to confirm the data is stationary or not. It is somewhat like ADF test, but the hypothesis is in opposite. KPSS is also a unit root test that tests the stationarity of the data around a deterministic trend.

Hypothesis of KPSS test:
Null hypothesis (H_0) – If it is failed to be rejected, the times series is trend stationary
Alternate hypothesis (H_1) – If the null hypothesis is rejected, the times series has unit root which means the data is not stationary

To reject the null hypothesis, the test statistic should be higher than critical values, and p-value should be less than the threshold (0.05).

Autocorrelation:
Autocorrelation measures the linear relationship between lagged values of time series. Autocorrelation represents the degree of similarity between a given time series and a lagged version of itself over successive time intervals. Autocorrelation measures the relationship between a variable's current value and its past values. An autocorrelation of +1 represents a perfect positive correlation, while an autocorrelation of negative 1 represents a perfect negative correlation.
$R\hat{}\, y\,(\tau)$ is an estimated autocorrelation for stationary time series y(t). It just depends on the lagged values of time series.

$$\hat{R}_y(\tau) = \frac{\sum_{t=\tau+1}^{T}(y_t - \overline{y})(y_{t-\tau} - \overline{y})}{\sum_{t=1}^{T}(y_t - \overline{y})^2}$$

Average method:
The forecast of all future values is equal to the average ("mean") of the historical data. The technique is very useful for forecasting short-term trends. The average method assumes that all observations are of equal importance and gives them equal weights when generating forecasts.

$$\hat{y}_{T+h|T} = \frac{y_1 + y_2 + \ldots + y_T}{T}$$

Naïve method:
For the Naive forecasts, we simply set all forecasts to be the value of the last observation. Hence, the naive method assumes, that the most observation is the only important one, and all previous observations provide no information for the future. This method sometimes works remarkably well for many economic and financial time series data and sometimes it gives a big error.

$$\hat{y}_{T+h|T} = y_T$$

Drift method:
The variation on the naive method is to allow the forecast to increase or decrease over time, where the amount of change over time (called the drift) is set to be the average change seen in the

historical data. This is equivalent to drawing a line between the first and last observations and extrapolating it into the future.

$$\hat{y}_{T+h|T} = y_T + \frac{h}{T-1}\sum_{t=2}^{T}(y_t - y_{t-1}) = y_T + h(\frac{y_T - y_1}{T-1})$$

Simple Exponential Smoothing (SES):

Simple exponential smoothing is calculated using weighted averages where the weights decrease exponentially as observations come from further in the past, the smallest weights are associated with the oldest observations.

$$\hat{y}_{t+1|t} = \alpha y_t + (1-\alpha)\hat{y}_{t|t-1}$$

Mean squared error:

In statistics, the mean squared error or mean squared deviation of an estimator measures the average of the squares of the errors—that is, the average squared difference between the estimated values and the actual value.

$$MSE = mean(e_t^2)$$

Multiple linear regression model:

The multiple linear regression model is,

$yt = \beta 0 + \beta 1x1,t + \beta 2x2,t + \cdots + \beta kxk,t + \varepsilon t$

$\beta 0, \beta 1, \ldots \beta k$ are unknown values which needs to be estimated using LSE using the following equation:

$\beta' = (XT.X)^{-1}\ XT.Y$

where X and Y are given as,

Matrix X has T rows and (k+1) columns where T is the number of samples and k is number of independent variables.

SVD:

Singular Value Decomposition or SVD the popular technique for dimensionality reduction. This is a linear algebra technique to create a projection of a sparse dataset prior to fitting a model.

s, d, v = np.linalg.svd(H)

Backwards stepwise regression:

1. Start with the model containing all potential predictors.
2. Remove one predictor at a time. Keep the model of it improves the measure of predictive accuracy.
3. Iterate until no further improvement.

Estimated Variance:

$$\hat{\sigma}_e = \sqrt{\frac{1}{T-k-1}\sum_{t=1}^{T} e_t^2}$$

STL decomposition method:

STL stands for "Seasonal and Trend decomposition using Loess", while Loess is a method for estimating nonlinear relationships. STL has several advantages over the classical, SEATS and X11 methods: STL can handle all type of seasonality, not only monthly and quarterly data. The seasonal component is allowed to change over time and the rate of change can be controlled by the user. The smoothness of trend-cycle can be controlled by the user.

Seasonally adjusted data:

Additive decomposition: seasonally adjusted data given by: yt − St = Tt + Rt

Multiplicative decomposition: seasonally adjusted data given by: yt/St = Tt × Rt

Strength of trend and seasonality:

The rubric to measure the strength of the trend is given as :

FT = max{0, 1 − Var(Rt) Var(Tt + Rt) }

For strongly trended data, the rubric FT is large.

The rubric to measure the strength of the seasonality is given as

FS = max{0, 1 − Var(Rt) Var(St + Rt) }

FS close to 0 exhibits almost no seasonality, FS close to 1 exhibits strong seasonality

Auto regressive model:

In a multiple regression model, we forecast the variable of interest using linear combination of predictors. In an autoregressive model, we forecast the variable of interest using a linear combination of past values of the variable. An AR process of order na AR(na) can be written as :

y(t) + a1y(t − 1) + a2y(t − 2) + . . . + ana y(t − na) = e(t)

where y(t) is the variable of interest and ǫ(t) is white noise (WsN ∼ (0, σ2e).

Moving average model:

Rather than using past values of the forecast variable in aregression, a moving average model uses past forecast errors in a regression-like model:

y(t) = e(t) + b1e(t − 1) + b2e(t − 2) + . . . + bnb e(t − nb)

where e(t) is white noise WN ∼ (0, σ2e). This is called a moving average model of order nb , MA(nb).

Generalized Partial Auto Correlation:

The generalized partial autocorrelation is used to estimate the order of ARMA model when na != 0 and nb != 0.

$$\phi_{kk}^{j} = \frac{\begin{vmatrix} \hat{R}_y(j) & \hat{R}_y(j-1) & \dots & \hat{R}_y(j+1) \\ \hat{R}_y(j+1) & \hat{R}_y(j) & \dots & \hat{R}_y(j+2) \\ \vdots & \vdots & \vdots & \vdots \\ \hat{R}_y(j+k-1) & \hat{R}_y(j+k-2) & \dots & \hat{R}_y(j+k) \end{vmatrix}}{\begin{vmatrix} \hat{R}_y(j) & \hat{R}_y(j-1) & \dots & \hat{R}_y(j-k+1) \\ \hat{R}_y(j+1) & \hat{R}_y(j) & \dots & \hat{R}_y(j-k+2) \\ \vdots & \vdots & \vdots & \vdots \\ \hat{R}_y(j+k-1) & \hat{R}_y(j+k-2) & \dots & \hat{R}_y(j) \end{vmatrix}}$$

Levenberg-Marquardt Algorithm:
In the gradient decent method, the sum of the squared errors is reduced by updating the parameters in the steepest-decent direction. In the Gauss-Newton method, the sum of the squared errors is reduced by assuming the least square's function is locally quadratic and finding the minimum of the quadratic. The LM method acts like a gradient-decent method when the parameters are far from their optimal value and acts more like the Gauss-Newton method when the parameters are close to their optimal value.

Seasonal ARIMA:
For seasonal time series with period s, observations that are s intervals apart are similar.
SARIMA (Na, D, Nb) with period of seasonality s can be written as:

$$A(q^{-s})\nabla_s^D y_t = B(q^{-s})\epsilon_t$$

where ∇^D is given as

$$\nabla_s^D = (1 - q^{-s})^D$$
$$A(q^{-s}) = 1 - A_1 q^{-s} - A_2 q^{-2s} - \ldots - A_{N_a} q^{-N_a s}$$
$$B(q^{-s}) = 1 - B_1 q^{-s} - B_2 q^{-2s} - \ldots - B_{N_b} q^{-N_b s}$$

Procedure:

1. Load the dataset and provide descriptive statistics about the dependent variable.
2. Check for stationarity (ADF, rolling mean, etc.,) and make the dataset stationary if it is not.
3. Perform Time Series Decomposition
4. Perform Holts-winter method on train and test data
5. Perform feature selection

6. Build multiple linear regression model based on the selected features
7. Perform predictions on base models to find the best fit
8. Develop ARMA, ARIMA and SARIMA models based on orders from GPAC
9. Choose the final model based on the performance measure of each model
10. Develop a forecast function and do h-step prediction based on the selected model

EXPERIMENTAL SETUP

Required libraries:

The following python libraries are required to run the code file seamlessly.

```
import matplotlib.pyplot as plt
import numpy as np
import statsmodels.tsa.arima_model
import statsmodels.tsa.holtwinters as ets
from Toolbox import *
import pandas as pd
from sklearn.model_selection import train_test_split
from statsmodels.tsa.seasonal import STL
from sklearn.metrics import mean_squared_error
from statsmodels.stats.diagnostic import acorr_ljungbox
import statsmodels.api as sm
from numpy import linalg as la
```

DESCRIPTION OF THE DATASET:

The dataset used in the project was taken from UCI Machine Learning Repository it has 19735 rows and 29 columns. The house temperature and humidity conditions were monitored with a ZigBee wireless sensor network hence the dataset is at 10 min for about 4.5 months. Each wireless node transmitted the temperature and humidity conditions around 3.3 min. Then, the wireless data was averaged for 10 minutes periods. The energy data was logged every 10 minutes with m-bus energy meters. Weather from the nearest airport weather station (Chievres Airport, Belgium) was downloaded from a public data set from Reliable Prognosis (rp5.ru) and merged with the experimental datasets using the date and time column. Two random variables have been included in the data set for testing the regression models and to filter out non predictive attributes (parameters).

Attribute Information:

Date	time year
month	day hour:minute:second
Appliances	Energy use in Wh
Lights	Energy use of light fixtures in the house in Wh
T1	Temperature in kitchen area, in Celsius
RH_1	Humidity in kitchen area, in %
T2	Temperature in living room area, in Celsius
RH_2	Humidity in living room area, in %
T3	Temperature in laundry room area
RH_3	Humidity in laundry room area, in %
T4	Temperature in office room, in Celsius
RH_4	Humidity in office room, in %
T5	Temperature in bathroom, in Celsius

RH_5	Humidity in bathroom, in %
T6	Temperature outside the building (north side), in Celsius
RH_6	Humidity outside the building (north side), in %
T7	Temperature in ironing room, in Celsius
RH_7	Humidity in ironing room, in %
T8	Temperature in teenager room 2, in Celsius
RH_8	Humidity in teenager room 2, in %
T9	Temperature in parents' room, in Celsius
RH_9	Humidity in parents' room, in %
To	Temperature outside (from Chievres weather station), in Celsius
Pressure (from Chievres weather station)	in mm Hg
RH_out	Humidity outside (from Chievres weather station), in %
Wind speed (from Chievres weather station)	in m/s
Visibility (from Chievres weather station)	in km
Tdewpoint (from Chievres weather station)	Â°C
rv1	Random variable 1, nondimensional
rv2	Random variable 2, nondimensional

Table1: Attribute Information

Dataset source: https://archive.ics.uci.edu/ml/datasets/Appliances+energy+prediction

Looking at the dataset information, it is clear the data has been recorded from one household and there are two possible dependent variables: Appliances, Lights. Considering the dataset is for one household and attribute values, 'Appliances' is selected as the independent variable.

Pre-processing dataset:

Analyzing the dataset's basic statistics, it is evident that the dataset doesn't have any null values. Hence, the dataset doesn't require much preprocessing and the dataset is also not sliced considering already it has data only for about 4.5 months. The 'date' variable is created as an index for the dataset and the dependent variable is 'Appliances'. The project through out will be using 'Appliances' as the dependent variable with 27 other predictor variables.

Analysis of dependent variable:

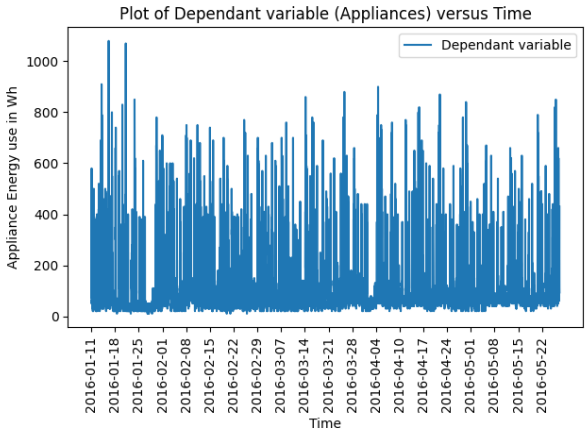


Fig1: Plot of dependent variable vs. Time

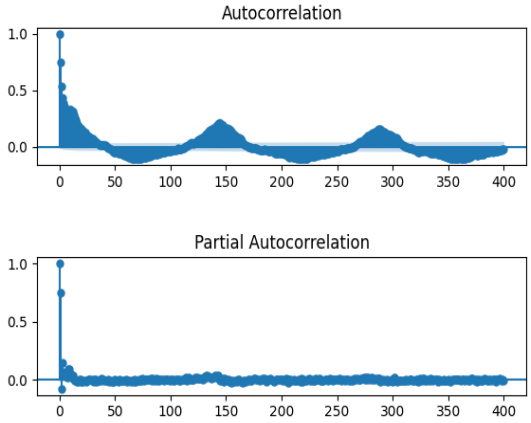


Fig2: ACF/PACF plot (lags=400)

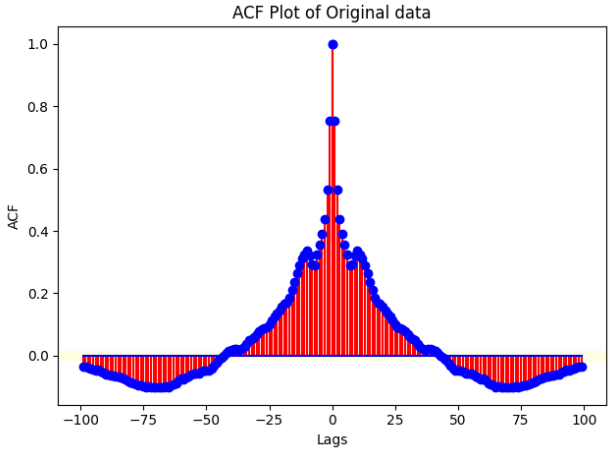


Fig3: ACF plot (lags=100)

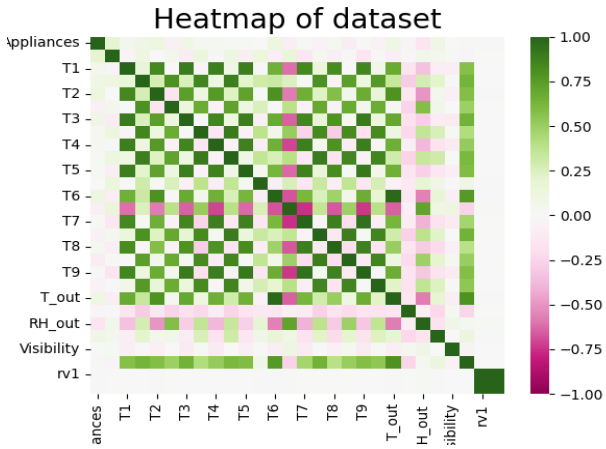


Fig4: Correlation matrix

The plot of dependent variable over the time appears that the data can be stationary but spikes throughout the plot shows there can be some seasonality. To understand better, ACF/PACF plot was generated and as expected the ACF converges over time but there is spike in the ACF values after every144th lag. This shows the data does have seasonality.

The symmetric ACF plot shows clear seasonality where the plot converges over the positive values of ACF for while and moves into the negative values of ACF. The seasonality at lag 144 was confirmed by the following calculation:

The dataset is recorded for every 10 minutes and there 24 hours in a day, considering 6 records for every one hour it can be concluded that the seasonality is at lag144 (10x6x24).

The PACF plot of the data cuts-off at lag of 12 and this can also be signing that data becomes stationary after lag 12. The correlation matrix has darker patterns which shows that the variables in the dataset are highly correlated with each other. So, proceeding further with model development the correlated variables should be removed to find the best model.

The dataset has been separated into train and test data based on the train test split ratio of 80:20.

STATIONARITY:

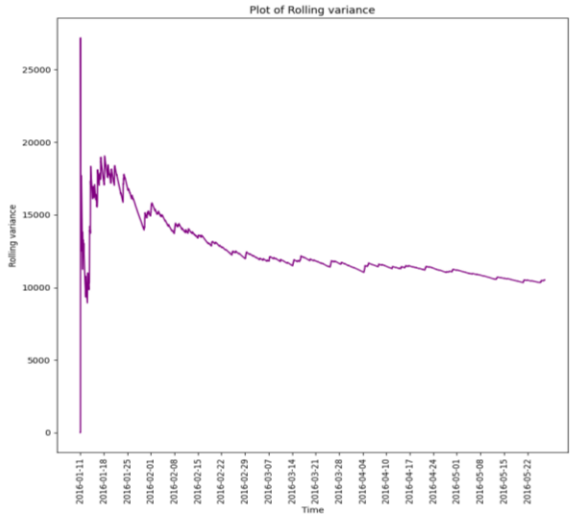


Fig5: Rolling variance (raw data)

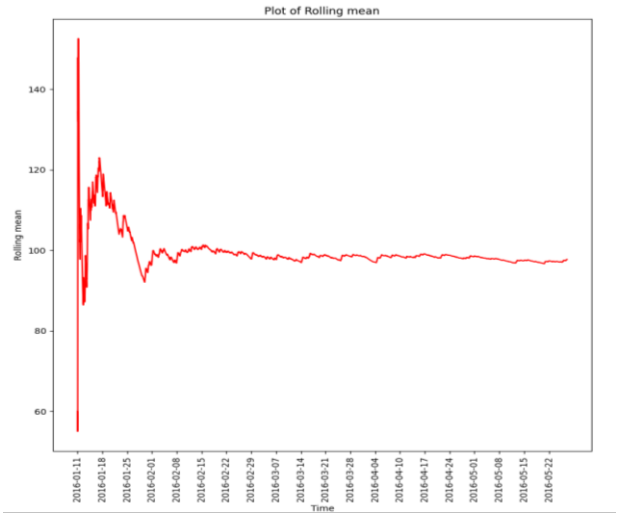


Fig6: Rolling mean (raw data)

The plot of rolling mean and variance shows that the data is stationary because the plot converges into a flat line after few iterations. But the mean of the data is not zero. There is some uncertainty in the first few iterations due to seasonality but the data towards the end shows a flat curve. These minor spikes can be reduced by differencing the data. Though, the plot of rolling mean and variance shows that the data is stationary, ADF and KPSS test has also been performed to confirm the stationarity of the data.

```
ADF Statistic: -21.616378
p-value: 0.000000
Critical Values:
    1%: -3.431
    5%: -2.862
    10%: -2.567
```

Fig7: ADF test of raw data

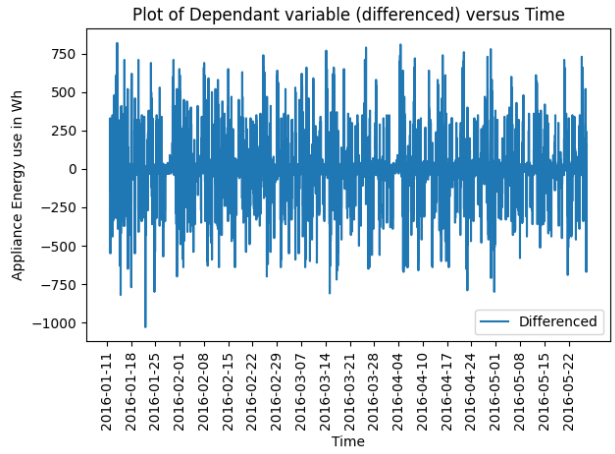
```
Test Statistic            0.036599
p-value                   0.100000
Lags Used                74.000000
Critical Value (10%)      0.347000
Critical Value (5%)       0.463000
Critical Value (2.5%)     0.574000
Critical Value (1%)       0.739000
dtype: float64
```

Fig8: KPSS test (raw)

Looking at the ADF test, the p-value is almost zero and the ADF test statistic is more negative, so we tend to reject the null hypothesis. And say the data is stationary.

The KPSS test has p-value higher than zero and the test statistic is lesser than the any of the critical values. So, we don't reject the null hypothesis and say the data is stationary.

As, discussed earlier the data tends to have some seasonality which resulted in not having mean flattening at zero. So, the data is differenced for interval of 144 and the stationarity of the differenced data is also tested again.



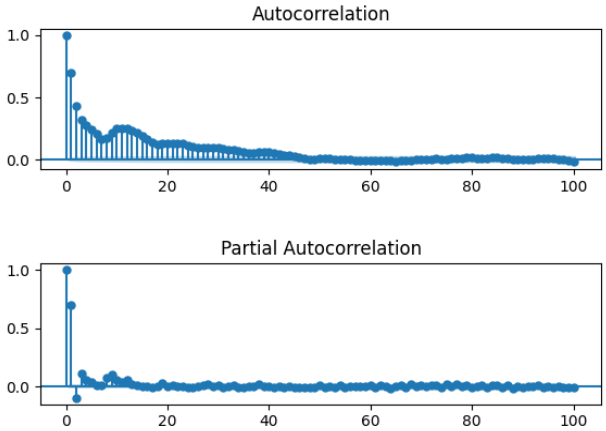
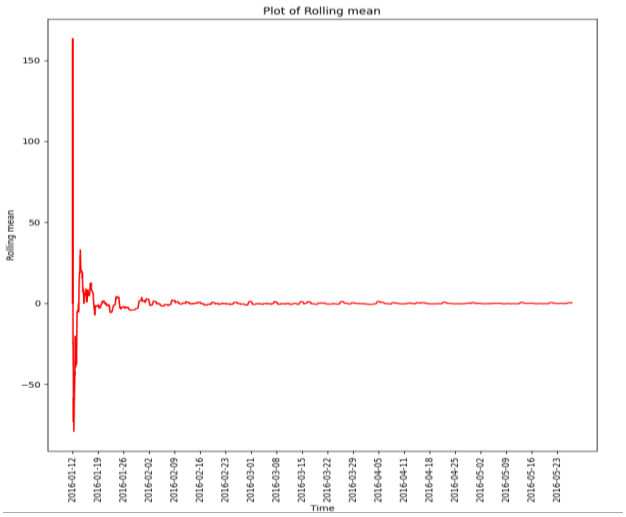


Fig9: Plot of differenced data vs. Time **Fig10: ACF/PACF plot of differenced data**

The differenced data looks more leveled than the original data. The ACF of the differenced data converges after lag 45 and the PACF plot cuts-off after lag 10 or so. The seasonality from the data has been removed using a seasonal differencing of interval 144. Let's analyze the stationarity of the differenced data.



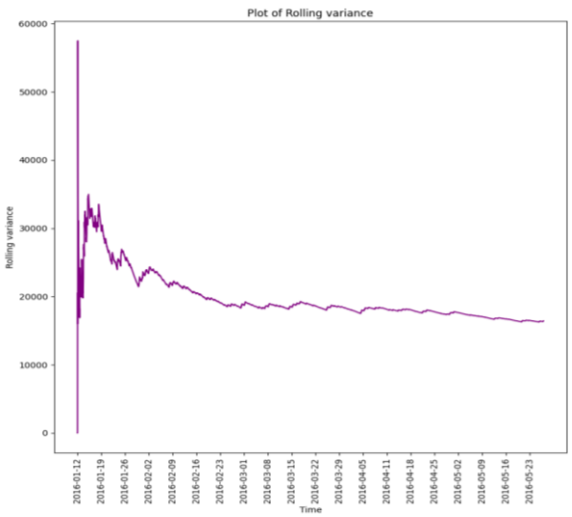


Fig11: Rolling mean (differenced) **Fig12: Rolling variance (differenced)**

The rolling mean and variance of data is more leveled now and the initial spikes due to seasonality in raw data has also been reduced. As expected, the seasonal differencing has made the data to be flat at rolling mean of zero. This shows the differenced data is also perfectly stationary.

```
ADF Statistic: -20.514060
p-value: 0.000000
Critical Values:
    1%: -3.431
    5%: -2.862
    10%: -2.567
```

```
10%. -2.567
Test Statistic                  0.00779
p-value                         0.10000
Lags Used                      62.00000
Critical Value (10%)            0.34700
Critical Value (5%)             0.46300
Critical Value (2.5%)           0.57400
Critical Value (1%)             0.73900
```

Fig13: ADF test (differenced)

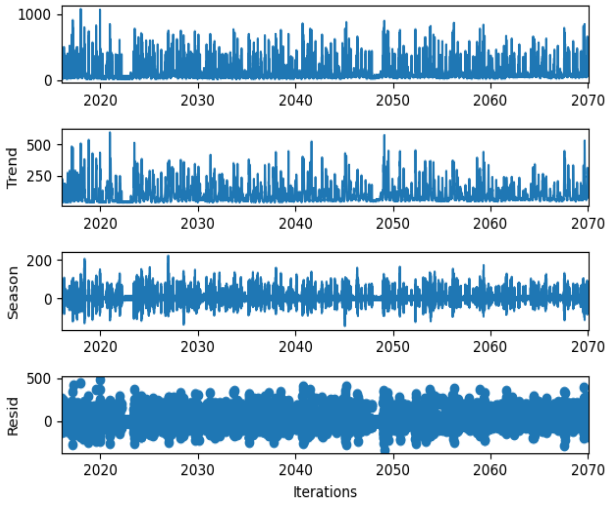
Fig14: KPSS test (differenced)

Looking at the ADF test, the p-value is almost zero and the ADF test statistic is more negative, so we tend to reject the null hypothesis. And say the data is stationary.

The KPSS test has p-value higher than zero and the test statistic is lesser than the any of the critical values. So, we don't reject the null hypothesis and say the data is stationary.

TIME SERIES DECOMPOSITION:

Time series decomposition of the data was performed using STL technique to understand the patterns of trend and seasonality in the data.



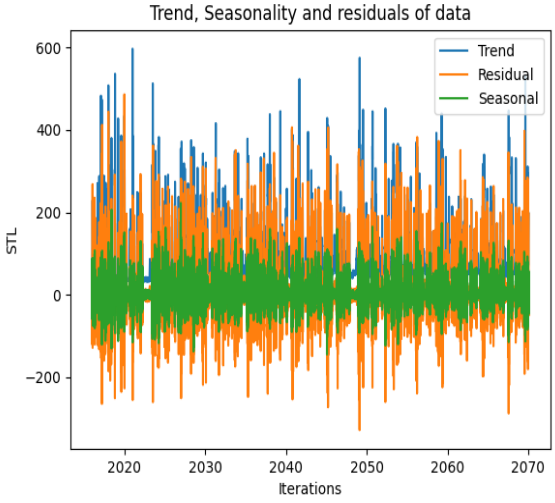


Fig15: Plot of STL decomposition

Fig16: Plot of residuals, trend and seasonal

The STL decomposition techniques helps to analyze the composition of residuals, trend, and seasonal patterns in the data. The plot of STL shows, that the residual values are scattered throughout the plot and there is some spike on the trend and seasonal plot which shows the data might be trended and seasonal. To confirm the percentage of trend and seasonality present in the data the strength of trend and seasonality was calculated.

```
Strength of trend: 0.7279717571942488
Strength of seasonality: 0.2882949243229801
```

Fig17: Strength of seasonality and trend

The strength of seasonality and trend shows the data is highly trended since the value is close to 1 and there exists some minor seasonality in the data as the value is just 0.2 which much away from 1. This shows that the data needs to be detrended and seasonally adjusted.

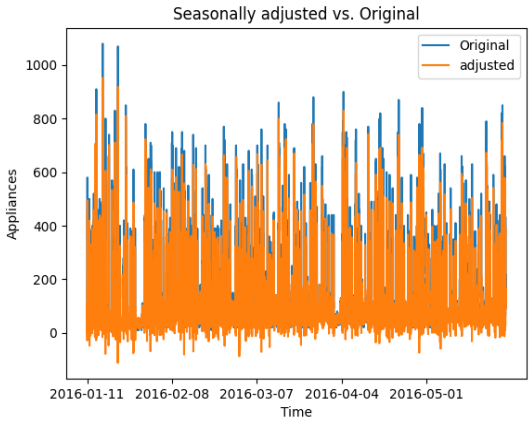


Fig18: Seasonally adjusted vs raw data

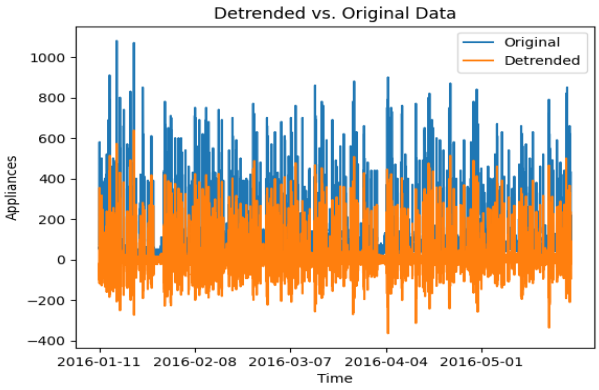


Fig19: Detrended data vs. raw data

The seasonally adjusted data follows the pattern close to the original data and the detrended data also follows the path of original data well. The original data looks like it has more variance than the detrended data.

HOLTS-WINTER METHOD:

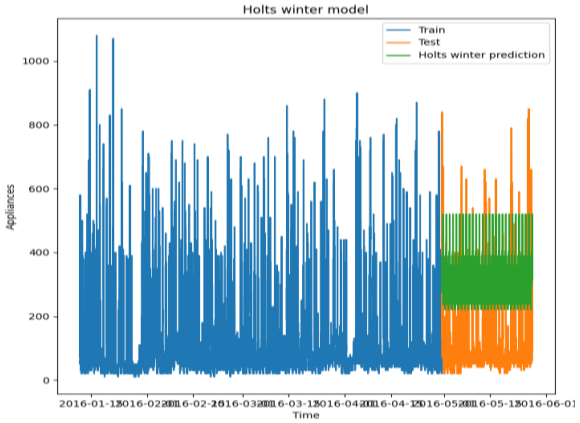


Fig20: Holts-winter method plot

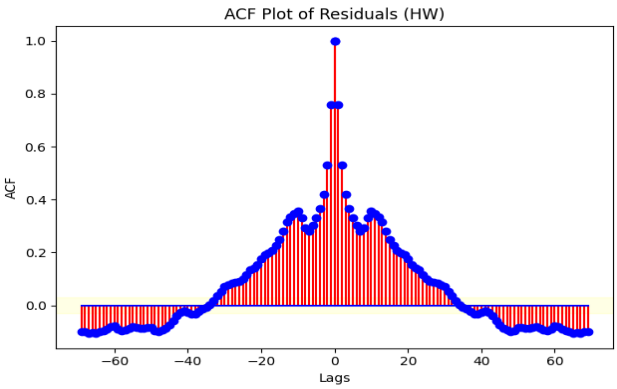


Fig21: ACF plot of residuals

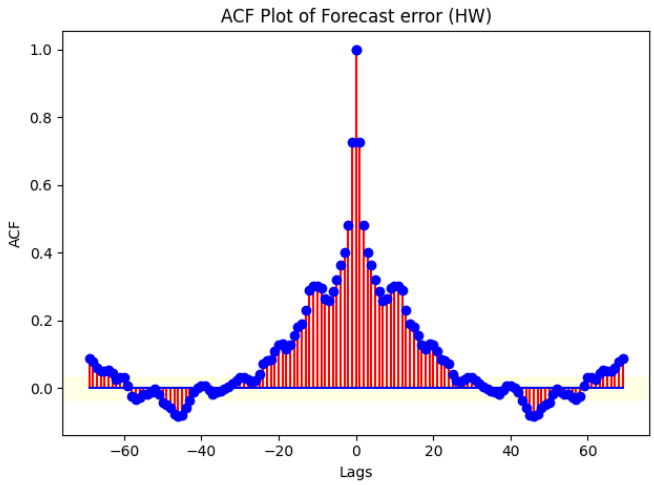


Fig22: ACF of forecast errors

The plot of holts-winter method looks like the model did not perform well on the test data as the path of forecasted follows are different from the pattern of original test values. This can be seen on the ACF plot of the residulas were the residuals are not white. This ideally shows the holts-winter method is not better for forecasting the data.

Performance metrics:

The Q-value of Box-pierce test is high (20191.89) with p-value 0.0. It shows the residual is not white for the holts-winter method. The MSE value of predictions are high (62632.91). The mean of the residual is not close to zero (-223.72) and variance is 12580.21. The RMSE value of predictions is 250.26. All these performance metrics shows the Holts-winter method is not a significant model to forecast the data.

The performance of this method on test data is given below:

MSE: 59846; Q-value: 4571; Mean of forecast: -225.63; Variance: 8937; RMSE:244.63

As expected, the MSE and mean od forecast errors are high which means the method didn't work well with the test data.

FEATURE SELECTION:

```
Singular Values:  [9.49180454e+09 1.43321499e+07 6.64639821e+06 2.41644450e+06
 1.71132439e+06 1.32612807e+06 1.14252067e+06 9.44023889e+05
 1.65167716e+05 1.12684780e+05 9.14268363e+04 6.75353423e+04
 4.80002467e+04 4.32048592e+04 4.09875639e+04 2.03833935e+04
 1.26346857e+04 1.07403810e+04 9.83140093e+03 7.27434807e+03
 5.34565853e+03 3.98959028e+03 3.07348563e+03 1.96329401e+03
 9.85144085e+02 7.32816280e+02 7.12515963e-01 9.48420991e-07]
The condition number is  6.800018126641549e+16
```

Fig23: Singular values and condition number

The SVD analysis shows that the full model has singular values which converges to zero over the end and the condition number is relatively high. This shows the model has features which are highly correlated and they need to be removed based on the OLS summary.

```
                            OLS Regression Results
==============================================================================
Dep. Variable:             Appliances   R-squared:                       0.173
Model:                            OLS   Adj. R-squared:                  0.172
Method:                 Least Squares   F-statistic:                     126.7
Date:                Wed, 08 Dec 2021   Prob (F-statistic):               0.00
Time:                        14:08:22   Log-Likelihood:                -94410.
No. Observations:               15788   AIC:                         1.889e+05
Df Residuals:                   15761   BIC:                         1.891e+05
Df Model:                          26
Covariance Type:            nonrobust
------------------------------------------------------------------------------
```

Fig24: Full model metrics

Looking at the adjusted R squared and R-squared value the model with all the variables included doesn't have better peformance and the AIC,BIC are relatively high. This shows the correlated features need to be removed to find the best fit.

Using, the OLS package the p-value of the models are estimated and the features with high p-values are removed using backward stepwise regression method.

The following features are removed based on backward regression technqiue: rv1, rv2, const, T7, RH_7, RH_5, T5, RH_out, Tdewpoint, RH_9, T1, Press_mm_hg

After removing the above defined features, the condition number of the model dropped below 1000 which shows the model is the best fit with all the correlated features removed.

```
The best features are Index(['lights', 'RH_1', 'T2', 'RH_2', 'T3', 'RH_3', 'T4', 'RH_4', 'T6',
       'RH_6', 'T8', 'RH_8', 'T9', 'T_out', 'Windspeed', 'Visibility'],
      dtype='object')
```

Fig25: Best features

Using the above defined features, a better multiple linear regression model can be developed to forecast the data.

MULTIPLE LINEAR REGRESSION:

	coef	std err	t	P>|t|	[0.025	0.975]
lights	2.1396	0.103	20.785	0.000	1.938	2.341
RH_1	13.8562	0.633	21.887	0.000	12.615	15.097
T2	-17.1101	1.402	-12.204	0.000	-19.858	-14.362
RH_2	-13.2627	0.739	-17.935	0.000	-14.712	-11.813
T3	25.5010	1.076	23.699	0.000	23.392	27.610
RH_3	8.6120	0.687	12.534	0.000	7.265	9.959
T4	-3.9367	1.052	-3.743	0.000	-5.998	-1.875
RH_4	-2.8629	0.677	-4.231	0.000	-4.189	-1.536
T6	8.3984	0.755	11.126	0.000	6.919	9.878
RH_6	0.3553	0.070	5.085	0.000	0.218	0.492
T8	8.5673	0.886	9.666	0.000	6.830	10.305
RH_8	-6.3745	0.321	-19.843	0.000	-7.004	-5.745
T9	-13.4323	1.560	-8.612	0.000	-16.490	-10.375
T_out	-6.1772	0.825	-7.484	0.000	-7.795	-4.559
Windspeed	1.0866	0.360	3.021	0.003	0.381	1.792
Visibility	0.2045	0.062	3.305	0.001	0.083	0.326

Fig26: Multiple Linear Regression model features

R-squared (uncentered):	0.557
Adj. R-squared (uncentered):	0.556
F-statistic:	1237.
Prob (F-statistic):	0.00
Log-Likelihood:	-94423.
AIC:	1.889e+05
BIC:	1.890e+05

Fig27: Performance metrics

Looking at the final model, the adjusted R-squared and R-squared values have increased better than the full model and the model's accuracy 0.557 which comparatively better than the previous model.

The predictions using the developed model are performed and the results of the performance of the model on train data are shown below.

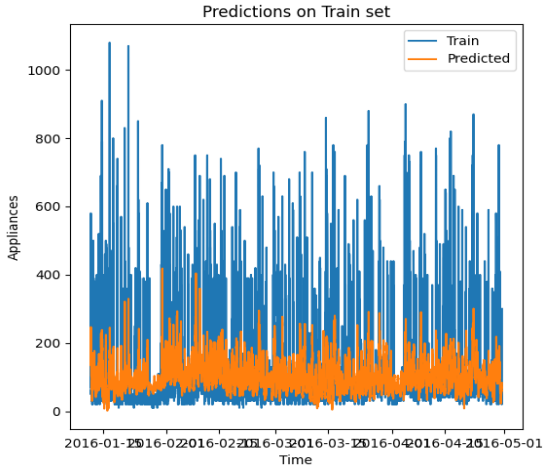


Fig28: Predictions on train data

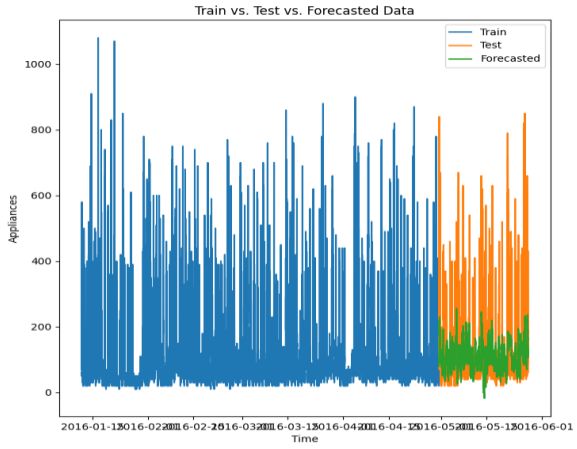


Fig29: Plot of train vs. test vs. Forecasted

The predicted data and forecasted data closely follows the path of the original data but it doesn't completely track the drastic increase in the data.

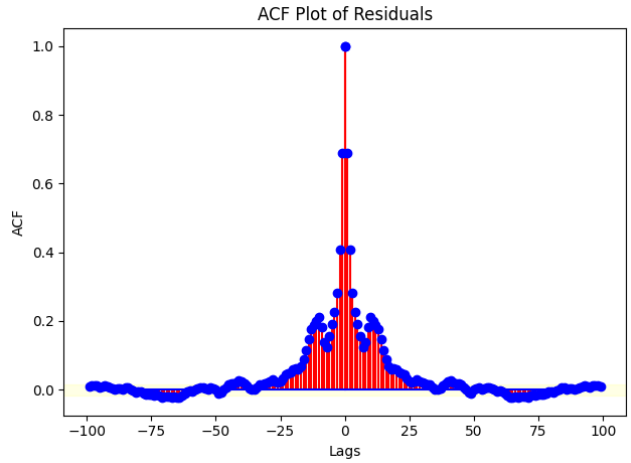


Fig30: ACF plot of residuals

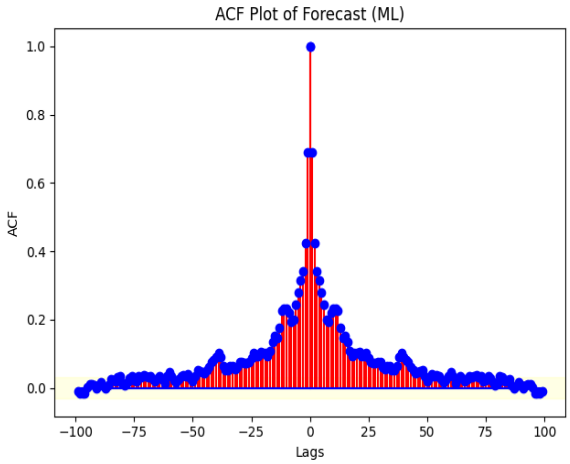


Fig31: ACF plot of forecast errors

The ACF plot of residuals shows the data converges after lag 25 and in bigger picture of lags the ACF of the residual may look like a white noise. To confirm whether the model is a better for forecasting or not let's comapre the performance metrics.

Performance metrics:

The MSE value of predictions are less comparatively (9168.34). The Q-value of Box-pierce test is comparatively less (12733.0) with p-value 0.0. It shows the residual can be white for the method. The mean of the residual is close to zero (0.053) and variance is 9168.34. The RMSE value of predictions is 95.75. All these performance metrics shows the Multiple Linear regression could be a better model which can be used for forecasting the data.

The performance of this method on test data is given below:

MSE: 7565.88; Q-value: 3753.47; Mean of forecast: -8.80; Variance: 7488.36; RMSE:86.98

As expected, the MSE and mean of forecast errors are less which means the method did work well with the test data.

BASE MODELS:

Average Method:

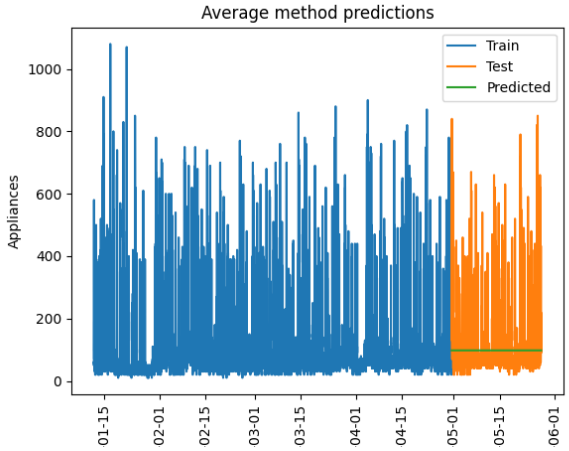


Fig32: Average method plot

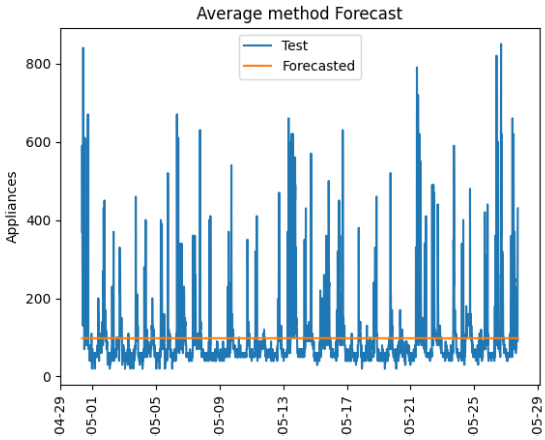


Fig33: AM: Test vs. Forecasted

Looking at the plot of average method, the method is not better fit for forecasting the values for the data because it predicts the values as flat curve. The value remains throughout the test data while the original data has more spikes than the forecasted.

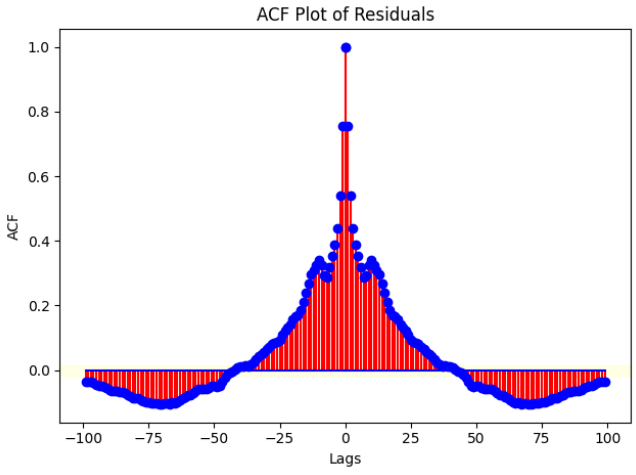


Fig34: ACF plot of residuals (Average)

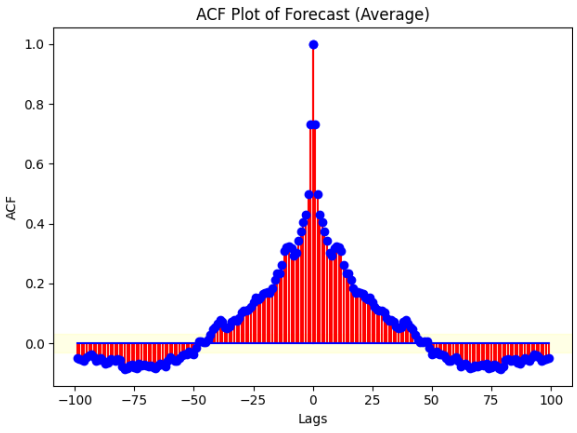


Fig35: ACF plot of forecast errors

The ACF plot of the residuals clearly shows that the ACF values are not converging throughout the given lags. And the ACF of forecast errors are also not a white noise. So, the model can't be a better fit for the dataset.

Performance metrics:

The Q-value of Box-pierce test is high (21037.19) with p-value 0.0. It shows the residual is not white for the Average method. The MSE value of predictions are high (11074.34). The mean of the residual is not close to zero (-2.008) and variance is 11070.31. The RMSE value of predictions is 95.750. All these performance metrics shows the average method is not a significant model to forecast the data.

The performance of this method on test data is given below:

MSE: 8290.72; Q-value: 4991.83; Mean of forecast: -1.64; Variance: 8288; RMSE: 86.98

As expected, the MSE and mean od forecast errors are high which means the method didn't work well with the test data.

Naive Method:

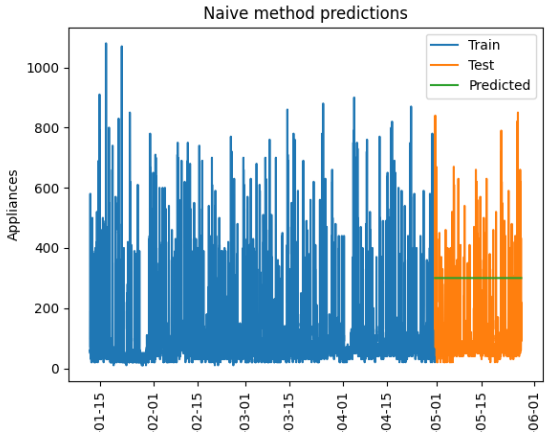


Fig36: Plot of Naïve method

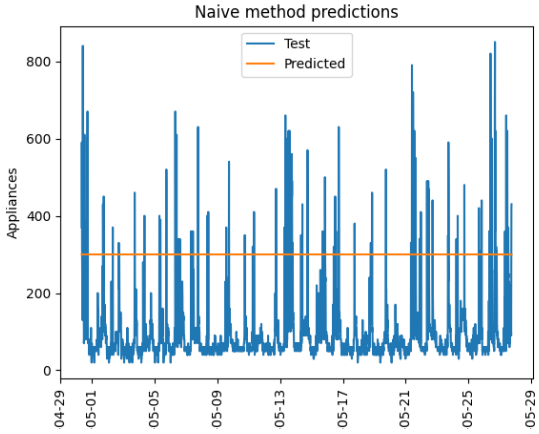
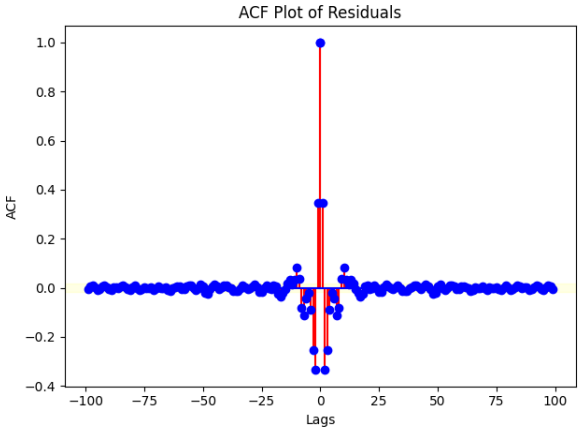


Fig37: Plot of test vs. Forecasted

Looking at the plot of naive method, the method may not better fit for forecasting the values for the data because it predicts the values as flat curve. The value remains throughout the test data while the original data has more spikes than the forecasted.



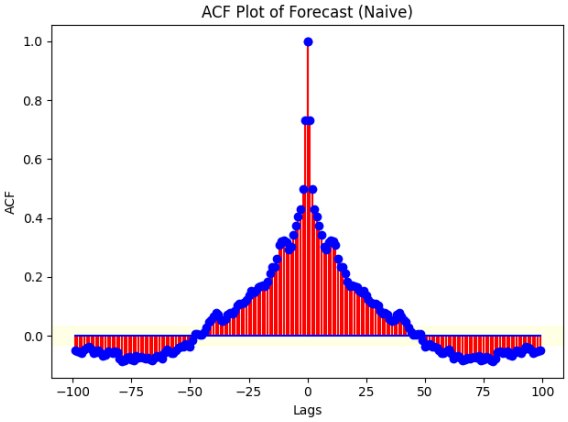


Fig38: ACF plot of residuals **Fig39: ACF plot of forecast**

The ACF plot of the residuals clearly shows that the ACF values are converging throughout the given lags. This can also be a better model, let's dicuss the performance metrics below.

Performance metrics:

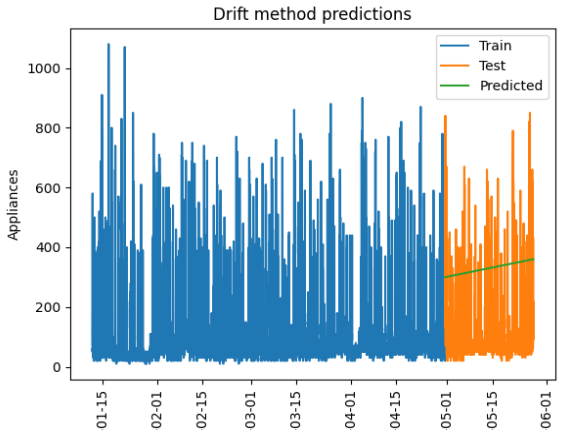
The Q-value of Box-pierce test is less comapred to average method (4799.30) with p-value 0.0. It shows the residual can be white for the naive method. The MSE value of predictions are high (10183.72). The mean of the residual is close to zero (0.0006) and variance is 10183.72. The RMSE value of predictions is 100.91. All these performance metrics shows the Naive method is can be a significant model to forecast the data.

The performance of this method on test data is given below:

MSE: 49750.34; Q-value 5014.17; Mean of forecast: -1.64; Variance: 8288; RMSE: 223.04

As expected, the MSE and mean of forecast errors are less compared to average method which means the method may work well with the test data (extended).

Drift Method:



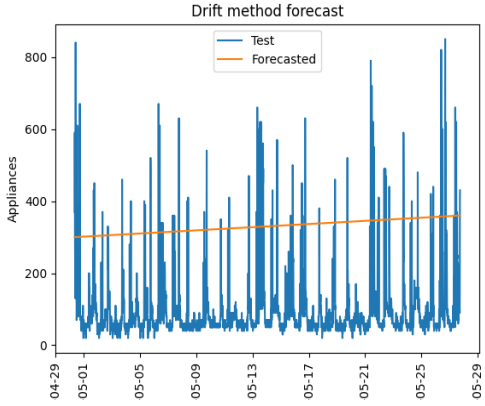
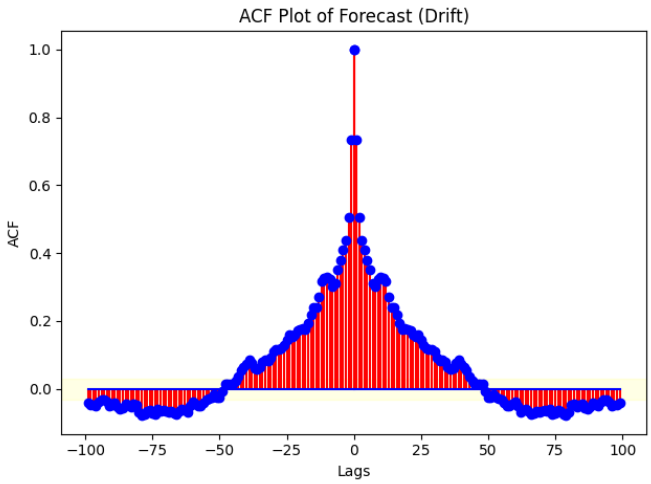


Fig40: Plot of drift method **Fig41: Test vs. Forecasted**



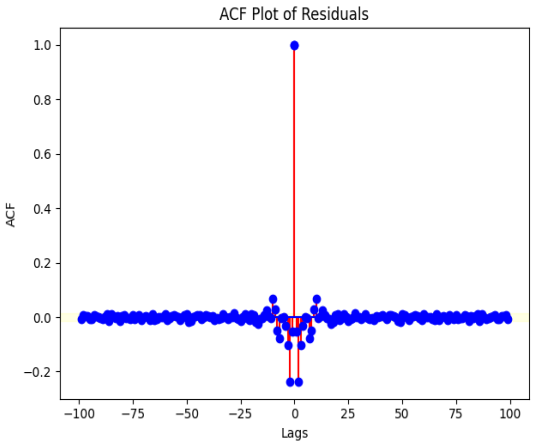


Fig41: ACF plot of forecast errors

Fig42: ACF plot of Residuals

The plot of drift method shows the forecasted data didn't trace the exact path of provided test data. But contradicting to the plot, ACF of residuals shows a white noise. While the forecast error's ACF doesn't converge over the lags. Let's discuss the performance measures to get a better understanding.

Performance metrics:

The Q-value of Box-pierce test is less comapred to average method (1128.89) with p-value 0.0. It shows the residual can be white for the drift method. The MSE value of predictions are less (5385.52). The mean of the residual is close to zero (-0.010) and variance is 5385.52. The RMSE value of predictions is 73.386. All these performance metrics shows the Naive method is can be a significant model to forecast the data.

The performance of this method on test data is given below:

MSE: 62980.33; Q-value: 5129.25; Mean of forecast: -233; Variance: 8396.18; RMSE: 250.958

As expected, the MSE and mean of forecast errors are less compared to average method which means the method may work well with the test data (extended).

Simple Exponential Smoothening Method:

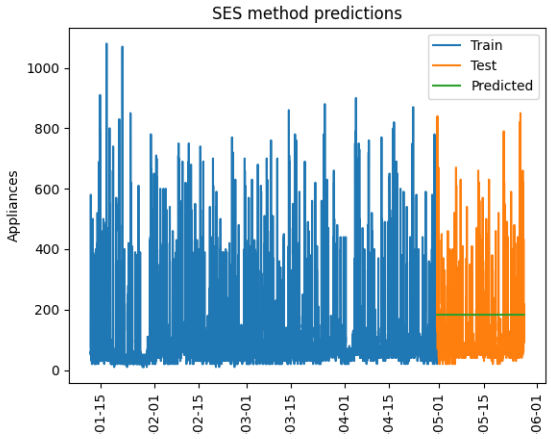


Fig43: Plot of SES method

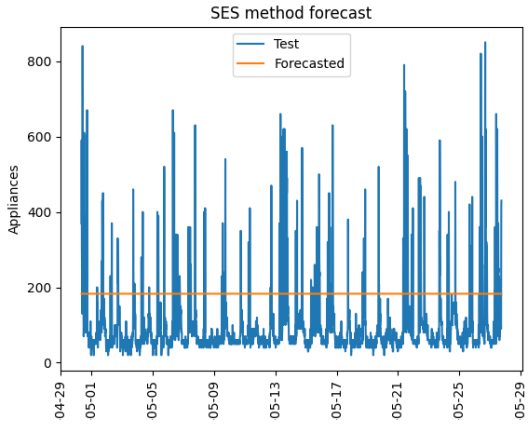


Fig44: Test vs. Forecasted

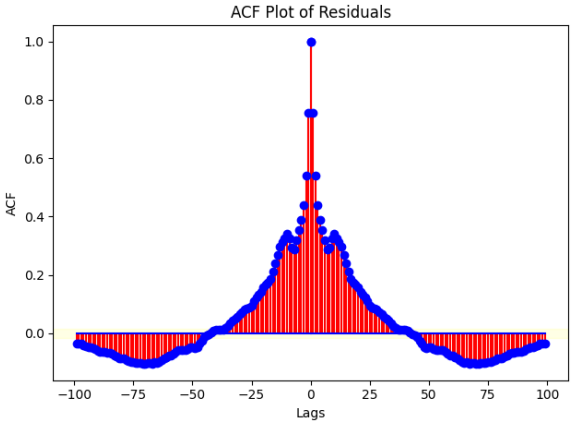


Fig45: ACF plot of residuals

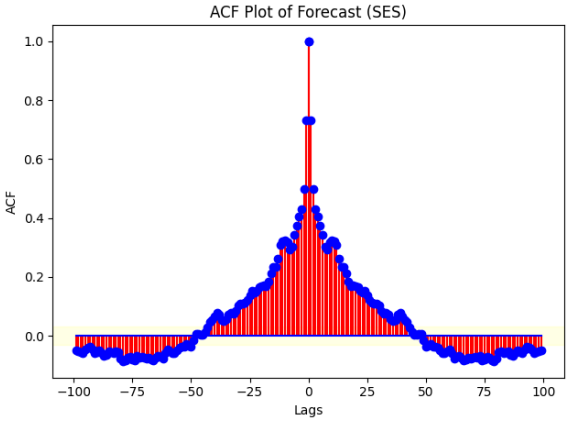


Fig46: ACF plot of forecast error

Looking at the plot of SES method, the method may not better fit for forecasting the values for the data because it predicts the values as flat curve. The value remains throughout the test data while the original data has more spikes than the forecasted.

The ACF plot of the residuals clearly shows that the ACF values are not converging throughout the given lags. And the ACF of forecast errors are also not a white noise. So, the model can't be a better fit for the dataset.

Performance metrics:

The Q-value of Box-pierce test is high (21049.19) with p-value 0.0. It shows the residual is not white for the SES method. The MSE value of predictions are high (18280.09). The mean of the residual is not close to zero (-84.93) and variance is 11067.19. The RMSE value of predictions is 135.20. All these performance metrics shows the SES method is not a significant model to forecast the data.

The performance of this method on test data is given below:

MSE: 15784.70; Q-value: 5014.17; Mean of forecast: -86.58; Variance: 8288.01; RMSE: 125.63

As expected, the MSE and mean od forecast errors are high which means the method didn't work well with the test data

ARMA, ARIMA AND SARIMA MODELS:

The order determination for the model can be done visualizing the GPAC table.

GPAC:

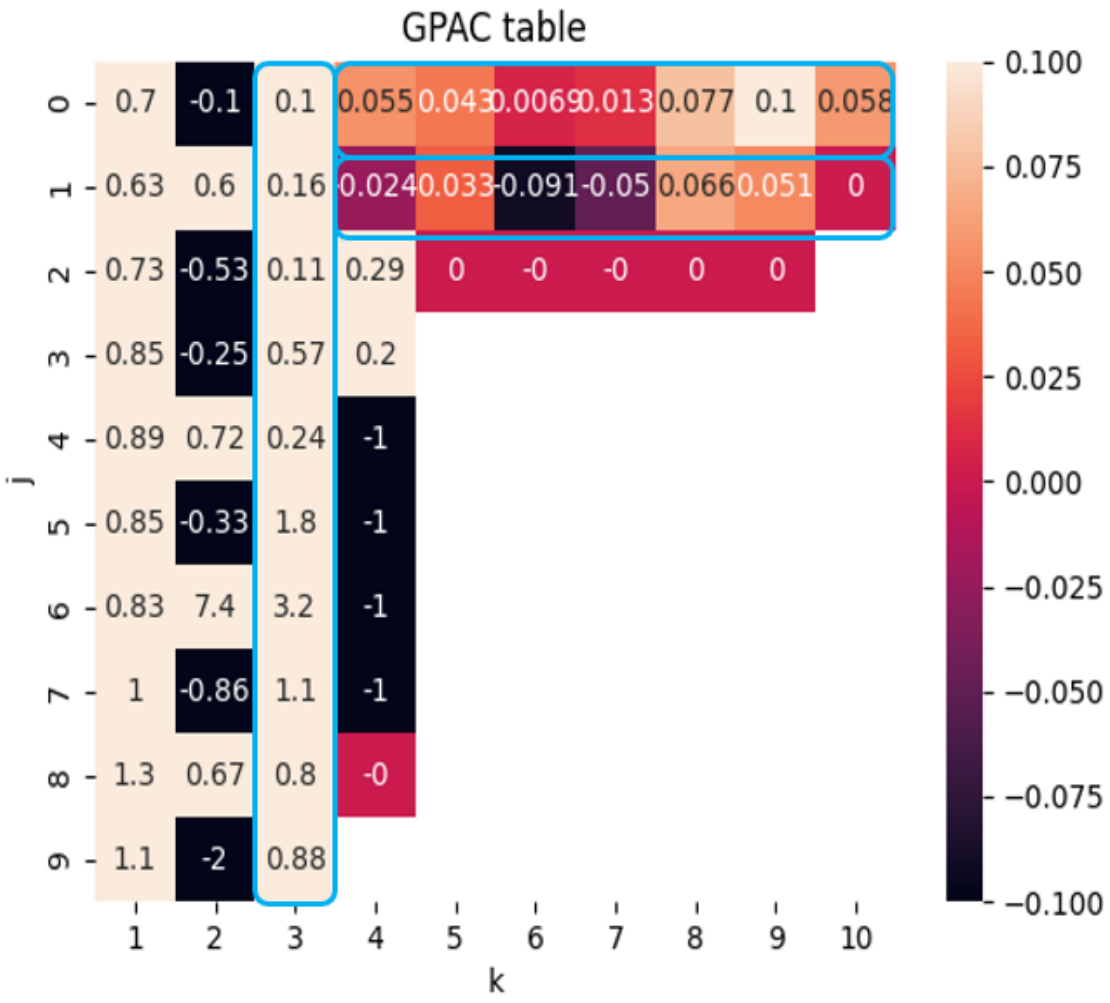


Fig47: GPAC

The GPAC table shows an order of (3,0) and (3,1) based on the column of constants and row of zeros.

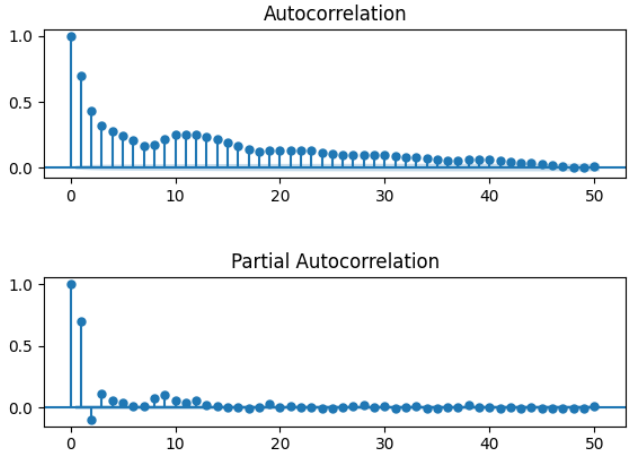


Fig48: ACF/PACF plot of GPAC

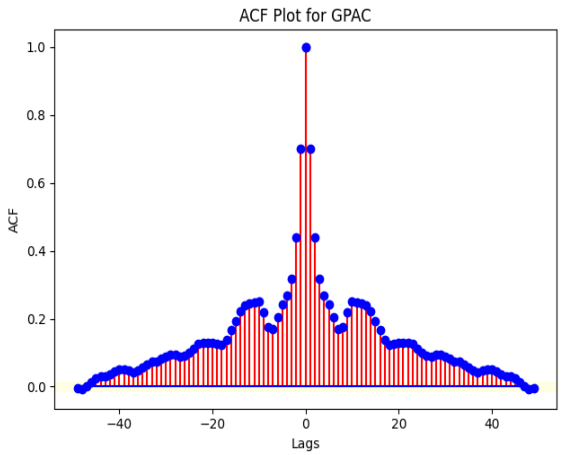


Fig49: ACF plot of GPAC

Looking at the plots of ACF/PACF generated for GPAC, we can see the ACF values converges over the lags.

ARMA (3,0):

Parameter estimation for the ARMA (3,0) is performed using the LMA algorithm.

```
The AR coefficient a0 is: 0.8826251619682011
The AR coefficient a1 is: -0.18060249327214198
The AR coefficient a2 is: 0.19154822230647062
```

Fig50: ARMA (3,0) parameter estimation

```
Confidence interval:
                          0         1
ar.L1.Appliances   0.867311  0.897940
ar.L2.Appliances -0.201005 -0.160200
ar.L3.Appliances   0.176234  0.206862
```

Fig51: Confidence Interval of ARMA (3,0)

The parameters estimated for the ARMA process doesn't include zero in their confidence interval. It shows the estimated parameters are statistically significant.

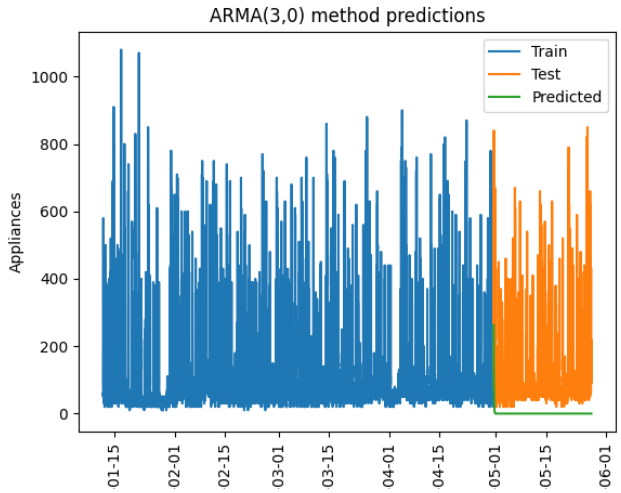


Fig52: Plot of ARMA (3,0)

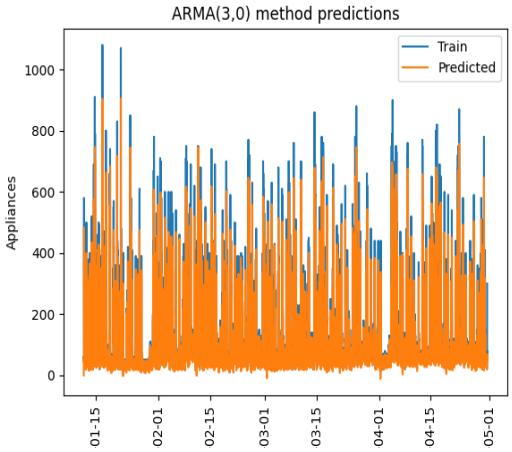


Fig53: Train vs. Predicted

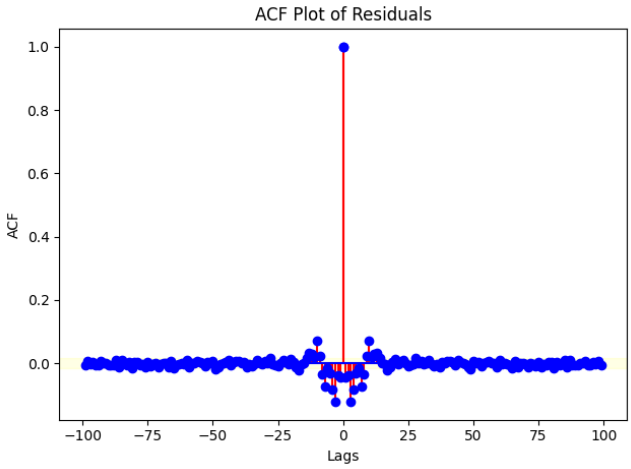


Fig54: ACF plot of residuals (ARMA)

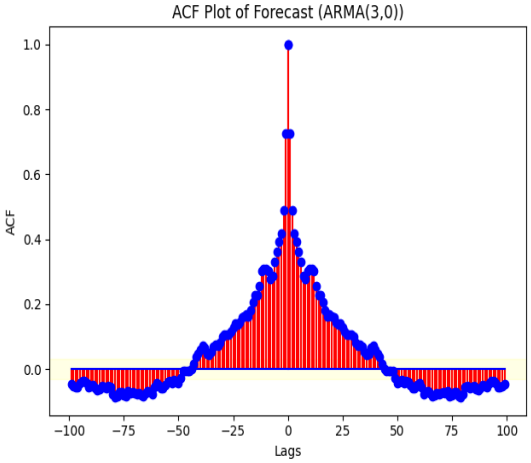


Fig55: ACF plot of forecast

```
Covariance matrix
                    ar.L1.Appliances  ar.L2.Appliances  ar.L3.Appliances
ar.L1.Appliances       6.105410e-05         -0.000054      7.277838e-07
ar.L2.Appliances      -5.374990e-05          0.000108     -5.374481e-05
ar.L3.Appliances       7.277838e-07         -0.000054      6.104928e-05
Standard error: ar.L1.Appliances    0.007814
ar.L2.Appliances    0.010410
ar.L3.Appliances    0.007813
```

Fig56: Covariance matrix and standard error

Looking at the ACF plot of residuals of ARMA(3,0) process, the ACF values are white which can be considered as a better model for forecast. But the ACF values after forecasting on test data doesn't look like white. So, it shows the model won't perform well on test data.

The plot of train versus predicted data shows the model has captured the data of train data well but looking at the forecasted values, the model didn't perform well due to the bias in the estimators.

Performance metrics:

The Q-value of Box-pierce test is high (414.215047) with p-value close to zero. The MSE value of predictions are less than our few other models (4846.33). The mean of the residual is not close to zero (10.449) and variance is 4737.13. The RMSE value of predictions is 69.615. All these performance metrics shows the model cannot be a significant model to forecast the data.

The performance of this method on test data is given below:

MSE: 17223.51; Q-value: 4824.96; Mean of forecast: 95.71; Variance: 8062; RMSE: 125.63

As expected, the MSE and mean of forecast errors are high which means the method didn't work well with the test data

ARMA (3,1):

Parameter estimation for the ARMA (3,1) is performed using the LMA algorithm.

```
    ...:
The AR coefficient a0 is: 1.6812121990548763
The AR coefficient a1 is: -0.8575867746710102
The AR coefficient a2 is: 0.17117671394561582
The MA coefficient a0 is: -0.8748667168727922
```

Fig57: ARMA (3,1) parameter estimation

```
Confidence interval:
                             0         1
ar.L1.Appliances  1.660670  1.701755
ar.L2.Appliances -0.885630 -0.829544
ar.L3.Appliances  0.154574  0.187779
ma.L1.Appliances -0.889873 -0.859860
```

Fig58: Confidence Interval of ARMA (3,1)

The parameters estimated for the ARMA process doesn't include zero in their confidence interval. It shows the estimated parameters are statistically significant.

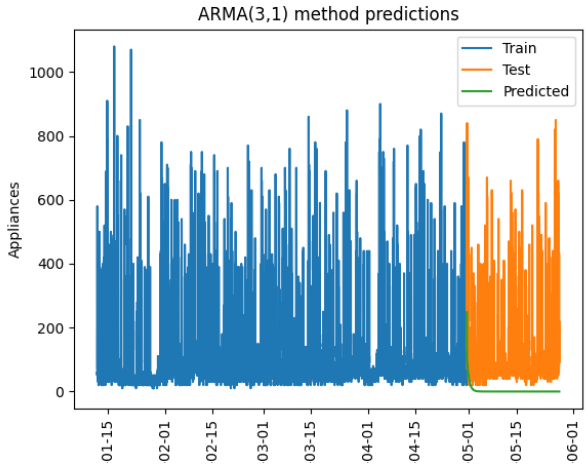


Fig59: Plot of ARMA (3,1)

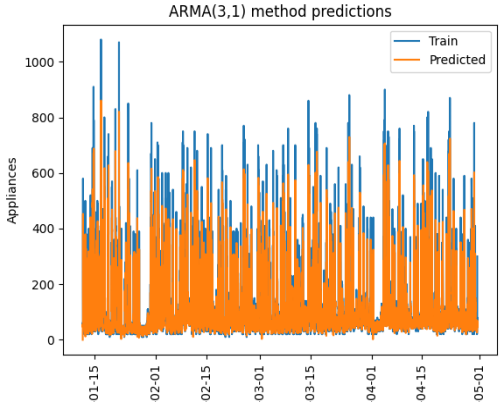


Fig60: Train vs. Predicted

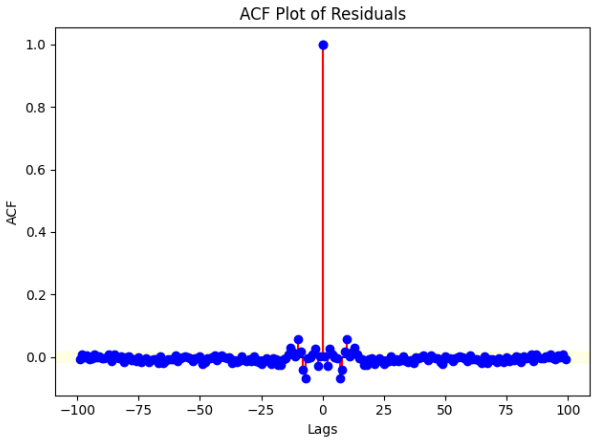


Fig61: ACF plot of residuals (ARMA)

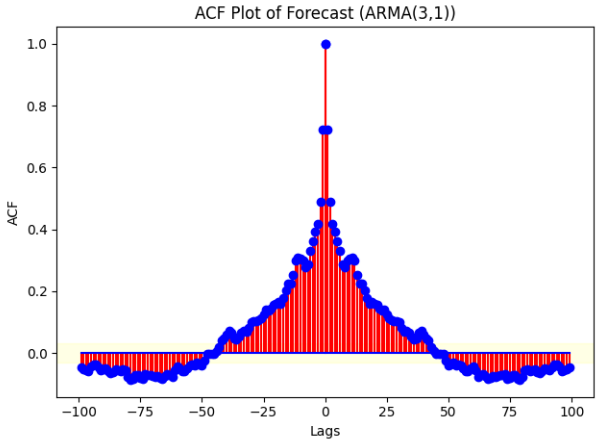


Fig62: ACF plot of forecast

```
                    ar.L1.Appliances  ...  ma.L1.Appliances
ar.L1.Appliances            0.000110  ...         -0.000053
ar.L2.Appliances           -0.000120  ...          0.000025
ar.L3.Appliances            0.000014  ...          0.000024
ma.L1.Appliances           -0.000053  ...          0.000059

[4 rows x 4 columns]
Standard error: ar.L1.Appliances    0.010481
ar.L2.Appliances    0.014308
ar.L3.Appliances    0.008471
ma.L1.Appliances    0.007657
```

Fig63: Covariance matrix and standard error

Looking at the ACF plot of residuals of ARMA(3,1) process, the ACF values are white which can be considered as a better model for forecast. But the ACF values after forecasting on test data doesn't look like white. So, it shows the model won't perform well on test data.

The plot of train versus predicted data shows the model has captured the data of train data well but looking at the forecasted values, the model didn't perform well due to the bias in the estimators.

Performance metrics:

The Q-value of Box-pierce test is less (24.179) with p-value close to zero. The MSE value of predictions are less than our few other models (4594.58). The mean of the residual is not close to zero (4.090) and variance is 4577.84. The RMSE value of predictions is 67.78. All these performance metrics shows the model can be a significant model to forecast the data.

The performance of this method on test data is given below:

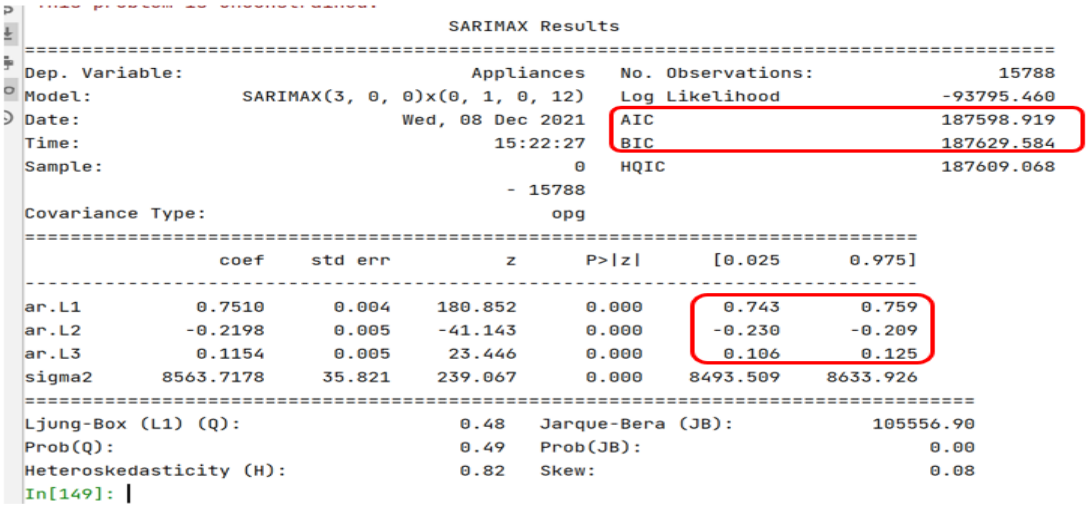
MSE: 16830.1964; Q-value: 4824.96; Mean of forecast: 93.73; Variance: 8043; RMSE: 129.73

As expected, the MSE and mean of forecast errors are high which means the method didn't work well with the test data

The ARMA models developed didn't perform well on the test data. Let's try a SARIMA model to check whether we could find a better model or not.

SARIMA Model:

Tried constructing a SARIMA model of (3,0,0)X(0,1,144) but the model was not computationally easy to work on the system. Hence, the seasonality period was reduced to 12 a model of SARIMA(3,0,0)X(0,1,0,12) was constructed.



SARIMAX Results

Dep. Variable:	Appliances	No. Observations:	15788
Model:	SARIMAX(3, 0, 0)x(0, 1, 0, 12)	Log Likelihood	-93795.460
Date:	Wed, 08 Dec 2021	AIC	187598.919
Time:	15:22:27	BIC	187629.584
Sample:	0	HQIC	187609.068
	- 15788		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.7510	0.004	180.852	0.000	0.743	0.759
ar.L2	-0.2198	0.005	-41.143	0.000	-0.230	-0.209
ar.L3	0.1154	0.005	23.446	0.000	0.106	0.125
sigma2	8563.7178	35.821	239.067	0.000	8493.509	8633.926

Ljung-Box (L1) (Q):	0.48	Jarque-Bera (JB):	105556.90
Prob(Q):	0.49	Prob(JB):	0.00
Heteroskedasticity (H):	0.82	Skew:	0.08

In[149]:

Fig64: SARIMA model

Looking at the result of SARIMA model constructed, the estimated parameters are under the confidence interval not inclusive of zero. Hence, the estimated parameters are significant but the AIC, BIC values are high which shows the model may not be a good one.

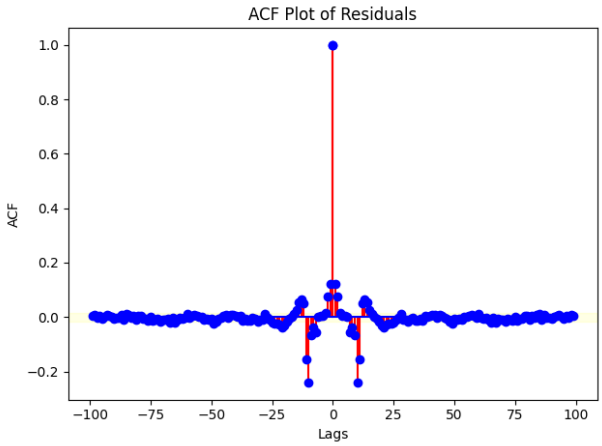


Fig65: ACF plot of residuals

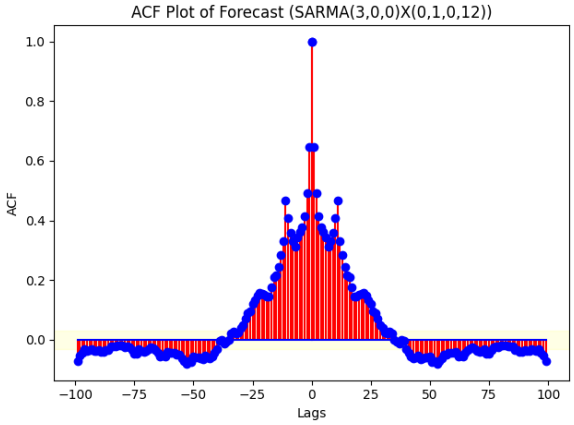


Fig66: ACF plot of forecast

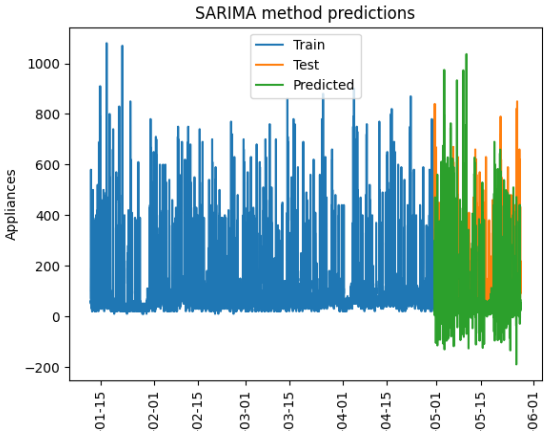


Fig67: Plot of the SARIMA

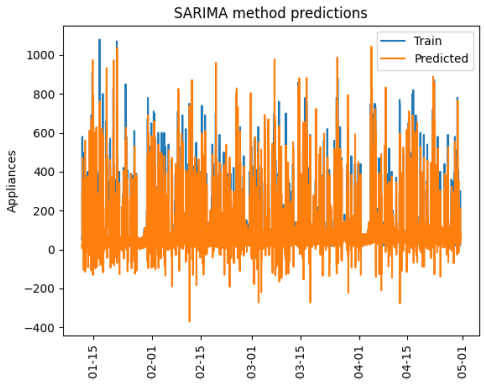


Fig68: Plot of train vs. predicted

Looking at the ACF plot of residuals, the ACF values are converging more similar to white noise. And the ACF of forecast errors also converge more to zero than the other models. The plot of test versus forecasted values shows that the model has tried capturing the values almost similarly. This shows that the SARIMA model could work better than other models if the computer was computationally high to do seasonal ARIMA of 144.

Performance metrics:

The Q-value of Box-pierce test is less (332.953388) with p-value close to zero. The MSE value of predictions are less than our few other models (5232.14). The mean of the residual is close to zero (0.065) and variance is 5232.14. The RMSE value of predictions is 72.33. All these performance metrics shows the model can be a significant model to forecast the data.

The performance of this method on test data is given below:

MSE: 21737.43; Q-value: 4351.759; Mean of forecast: -0.67; Variance: 5232.14; RMSE: 147.43

As expected, the MSE of forecast errors are high which means the method didn't work well with the test data

The SARIMA model has more chances to out perform the other models. Unfortuantely, seaonality of 12 wasn't effective much to forecast the data.

FINAL MODEL SELECTION:

Comparing all the models to select the best model.

METRICS	**HOLTS WINTER**	**MLR**	**AVG.**	**NAÏVE**	**DRIFT**	**SES**	**ARMA (3,0)**	**ARMA (3,1)**	**SARIMA**
MSE TRAIN	62632.91	9168.34	11074.34	10183.72	5385.52	18280.09	4846.33	4594.581	5232.14
Q-VALUE	20191.89	12733.0	21037.19	4799.30	1128.89	21049.19	414.215047	24.179	332.88
MEAN RESID	-223.72	0.053	-2.008	0.0006	-0.010	-84.93	10.449	4.0905	0.06
VAR. RESID	12580.21	9168.34	11070.31	10183.72	5385.52	11067.19	4737.134	4577.84	-0.67
RMSE TRAIN	250.26	95.75	95.750	100.91	73.386	135.20	69.615	67.78	72.33
MSE TEST	59846.6	7565.88	8290.72	49750.34	62980.3	15784.7	17223	16830	21737.4
RMSE TEST	244.63	86.98	86.98	223.04	250.958	125.63	125.63	129.73	147.43

Considering the previous discussions, the ACF plot of residuals being white and less Q-values the models multiple linear regression, drift, ARMA and SARIMA seems to be better. Out of these models, considering the mean and MSE of residuals multiple linear regression, drift, ARMA and SARIMA has lesser values. It means the models predicted close to the actual values. Out of these four models, while looking at their performance on test data multiple linear regression seems to have lesser MSE value and when compared to drift and naïve method it traced the path of test data more closely.

Hence, the best model for the data forecasting is multiple linear regression.

FORECAST FUNCTION:

The forecast function for multiple linear regression can be written using the final 15 variables selected using OLS model summary.

y_hat_t = 2.1396*lights + 13.8562*RH_1 + (-17.11)*T2 + (-13.2627)*RH_2 + 25.5010*T3+8.6120*RH_3 + (-3.9367)*T4 + (-2.8629)*RH_4 + (8.3984)*T6 + (0.3553)*RH_6+ 8.5673*T8 + (-6.3745)*RH_8 + (-13.4323)*T9 + (-6.1772)*T_out+ 1.0866 *Windspeed + 0.2045*Visibility

H-STEP AHEAD PREDICTION:

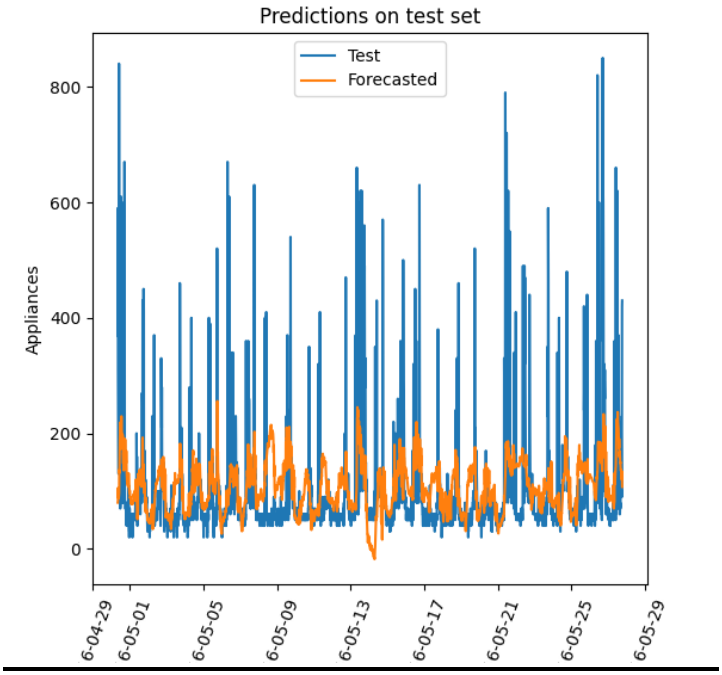


Fig69: Plot of test data

The h-step prediction on test data was conducted based on the forecast function developed. The forecast function has more closely followed the path of the test data and forecast on test data haven't covered the drastic increase or decrease of the values in the data, but it has captured better than the other models.

CONCLUSION

The Multiple Linear Regression model has been selected as the best model to predict the energy consumed in the household. The model was chosen based on the consideration of its performance on both the test and train data. Considering, other models in the competition with this model ARMA (3,1) had lower MSE on train data but on the test data the model didn't capture the exact data. While the SARIMA was limited due to computational complexities with the seasonality of 12 it performed well on train and test data but compared to the MSE on test data the Multiple Linear Regression model outperformed it too. The adjust R-squared was good (0.577) compared to the other models which were performing lesser than it.

The limitation of the selected model would be the model didn't capture well the drastic increase or decrease in the data values. To overcome this limitation SARIMA model with seasonality of 144 can be used on high computational systems and forecast more accurate results for the dataset. The future enhancement of the project would be implementing a SARIMA model with exact seasonal order or LSTM – a neural network-based model to approximate the energy consumption in households. For future modelling purposes, the data should be increased at least till one year and including more households to understand the variation in energy consumption. And variable to notify number of people in the house would also help to estimate the energy requirement for house with different dimensions.

APPENDIX

Steps to run the project file:

Run the Project_RM.py file to implement the project end-to-end.

Toolbox.py

```
import numpy as np
import pandas as pd
from pandas import Series
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import adfuller,kpss,pacf
import statsmodels.api as sm
import seaborn as sns
from scipy import signal
np.random.seed(12345)
#Function to Calculate rolling mean and variance
def cal_rolling_mean_var(x,y):
    #calculating rolling mean and variance
    rolling_mean=[]
    rolling_variance=[]
    for i in range(1,len(x)+1):
        result=np.mean(x[:i])
        result_var=np.var(x[:i])
        rolling_mean.append(result)
        rolling_variance.append(result_var)
    print("Rolling mean:",rolling_mean)
    print("Rolling variance:",rolling_variance)
    print("*"*100)

    #Plotting mean
    plt.figure(figsize=(10,10))
    plt.plot(y,rolling_mean, color='red')
    plt.ylabel("Rolling mean")
    plt.xlabel('Time')
    plt.xticks(y[::1000],rotation='vertical')
    plt.title('Plot of Rolling mean')
    plt.show()
    #Plotting variance
    plt.figure(figsize=(10,10))
    plt.plot(y,rolling_variance, color="Purple")
    plt.xticks(y[::1000], rotation='vertical')
    plt.ylabel(f"Rolling variance")
    plt.xlabel('Time')
    plt.title('Plot of Rolling variance')
    plt.show()


#Function to perform ADF test
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))

#Function to perform KPSS test.
def kpss_test(z):
    kpsstest = kpss(z, regression='c', nlags="auto")
    kpss_output = pd.Series(kpsstest[0:3], index=['Test Statistic','p-
value','Lags Used'])
    for key,value in kpsstest[3].items():
        kpss_output['Critical Value (%s)'%key] = value
    print (kpss_output)

#First order differencing
def first_order_diff(l):
    diff=[]
    diff[0]=diff.append([np.nan])
    for j in range(1,len(l)):
        diff_val=l[j]-l[j-1]
        diff.append(diff_val)
    return Series(diff)

#Second order differencing
def second_order_diff(l):
    diff=[]
    diff[0]=diff.append([np.nan])
    diff[1]=diff.append([np.nan])
    for j in range(2,len(l)):
        diff_val=l[j]-2*l[j-1]+l[j-2]
        diff.append(diff_val)
    return Series(diff)

#Third order differencing
def third_order_diff(l):
    diff=[]
    diff[0]=diff.append([np.nan])
    diff[1]=diff.append([np.nan])
    diff[2]=diff.append([np.nan])
    for j in range(3,len(l)+2):
        diff_val=l[j]-l[j-1]
        diff.append((diff_val))
    return Series(diff)

#Function to calculate Corrrelation coefficient
def correlation_coefficient_cal(x,y):
    n=len(x)
    sum1=0
    sum2=0
    for i in x:
        sum1+=i

    meanx=sum1/n

    for j in y:
        sum2+=j
```

```
    meany=sum2/n
    diff=0
    deno1=0
    deno2=0
    for i,j in zip(x,y):
        diff+=(i-meanx)*(j-meany)
        deno1+=(i-meanx)**2
        deno2+=(j-meany)**2

    den=(deno1)**0.5
    den1=(deno2)**0.5
    r=diff/(den*den1)
    return r

#Auto correlation Function
def ACF(x,lags):
    acfmean=np.mean(x)
    T=len(x)
    list1=[]
    prod1=0
    deno1=0
    for t in range(0, T):
        deno1 += (x[t] - acfmean) ** 2

    for l in range(0,lags):
        for t in range(l,T):
            prod1+=(x[t]-acfmean)*(x[t-l]-acfmean)
        acf=float(prod1/deno1)
        prod1=0
        list1.append(acf)
    print("ACF:",list1)
    return list1


#Function to estimate variance
def estimated_var(x,k):
    sum=0
    for i in range(0,len(x)):
        sum+=i**2
    deno=1/(len(x)-k-1)
    var=np.sqrt((deno*sum))
    return var

#Function to calculate moving average
def movingaverage(data):
    m=int(input("Enter m:"))
    if m%2!=0:
       j = 0
       ma_list =[]
       while (j+m)!=len(data)+1:
           sum = 0
           mean=0
           for i in range(j,j+m):
               sum+=data[i]
           mean=sum/m
           ma_list.append(mean)
```

```
            j+=1
            k = int((m - 1) / 2)
        return ma_list,k

    elif m%2==0:
        fold=int(input("Folding order:"))
        if fold%2!=0:
            print("Invalid fold value(Fold should be even)")
        else:
            j=0
            ma_list = []
            while (j + m) != len(data) + 1:
                sum = 0
                mean = 0
                for i in range(j, j + m):
                    sum += data[i]
                mean = sum / m
                ma_list.append(mean)
                j += 1
            j=0
            final=[]
            while (j + fold) != len(ma_list) + 1:
                sum = 0
                mean = 0
                for i in range(j, j + fold):
                    sum += ma_list[i]
                mean = sum / fold
                final.append(mean)
                j += 1
            k=int(m/2)
            return final,k


#Function to generate arma process
def armaprocess_GPAC():
    T=int(input("Enter number of samples"))
    mean=int(input("Enter the mean of white noise"))
    var=int(input("Enter variance of white noise"))
    na=int(input("Enter AR process order"))
    nb=int(input("Enter MA process order"))
    naparam=[0]*na
    nbparam=[0]*nb
    for i in range(0,na):
        naparam[i]=float(input(f"Enter the coefficient of AR:a{i+1}"))
    for i in range(0,nb):
        nbparam[i]=float(input(f"Enter the coefficient of MA:b{i+1}"))
    ar=np.r_[1,naparam]
    ma=np.r_[1,nbparam]
    arma_process = sm.tsa.ArmaProcess(ar, ma)
    if mean==0:
        y=arma_process.generate_sample(T)
    else:
        mean_y = mean* (1 + np.sum(nbparam)) / (1 + np.sum(naparam))
        y = arma_process.generate_sample(T, scale=np.sqrt(var) + mean_y)
    return y

#Calculate Qvalue in the GPAC
```

```
def qvalue_cal(y,an,bn):
    deno=[]
    k=an
    j=bn
    for a in range(k):
        deno.append([])
        for b in range(k):
            deno[a].append(y[np.abs(j+b)])
        j=j-1
    ddeno=round(np.linalg.det(deno),5)
    j=bn
    num=deno[:k-1]
    num.append([])
    for a in range(k):
        num[k-1].append(y[j+a+1])
    dnum=round(np.linalg.det(num),5)
    if ddeno==0:
        return float('inf')
    else:
        qval=dnum/ddeno
        return round(qval,4)

#Function to align Q-values in GPAC
def GPAC(y,k,j):
    q=[]
    for b in range(j):
        q.append([])
        for a in range(1,k+1):
            q[b].append(qvalue_cal(y,a,b))
    gpac=np.array(q).reshape(j,k)
    gpactable=pd.DataFrame(gpac)
    c=np.arange(1,k+1)
    gpactable.columns=c
    print(gpactable)
    sns.heatmap(gpactable,annot=True)
    plt.xlabel('k')
    plt.ylabel('j')
    plt.title('GPAC table')
    plt.show()

#ACF plot
def acf_plot(lags,acf,samples):
    x = np.arange(0,lags)
    m = 1.96 / np.sqrt(len(samples))
    plt.stem(x,acf, linefmt='r-', markerfmt='bo', basefmt='b-')
    plt.stem(-1 *x,acf,linefmt='r-',markerfmt='bo', basefmt='b-')
    plt.title("ACF")
    plt.axhspan(-m,m,alpha=.2,color='yellow')
    plt.xlabel('Lags')
    plt.ylabel('ACF')
    plt.show()

#Professor's ACF_PACF plot
from statsmodels.graphics.tsaplots import plot_acf , plot_pacf
def ACF_PACF_Plot(y,lags):
 acf = sm.tsa.stattools.acf(y, nlags=lags)
 pacf = sm.tsa.stattools.pacf(y, nlags=lags)
```

```
 fig = plt.figure()
 plt.subplot(211)
 plt.title('ACF/PACF of the raw data')
 plot_acf(y, ax=plt.gca(), lags=lags)
 plt.subplot(212)
 plot_pacf(y, ax=plt.gca(), lags=lags)
 fig.tight_layout(pad=3)
 plt.show()

#LM algorithm
def WN(teta,na,y):
    numerator=[1]+list(teta[na:])
    denominator=[1]+list(teta[:na])
    if len(numerator)!=len(denominator):
        while len(numerator)<len(denominator):
            numerator.append(0)
        while len(denominator)<len(numerator):
            denominator.append(0)
    system=(denominator,numerator,1)
    t,e=signal.dlsim(system,y)
    e=[i[0] for i in e]
    return np.array(e)

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

def step1(delta,na,nb,teta,y):
    e_teta=WN(teta,na,y)
    SSE_O=np.dot(e_teta.T,e_teta)
    X=[]
    for i in range(na+nb):
        teta_delta = teta.copy()
        teta_delta[i]=teta[i]+delta
        en=WN(teta_delta,na,y)
        Xi=(e_teta-en)/delta
        X.append(Xi)
    Xfinal=np.transpose(X)
    A=np.dot(Xfinal.T,Xfinal)
    G=np.dot(Xfinal.T,e_teta)
    return A,G,SSE_O

def step2(A,G,mu,na,nb,teta,y):
    n=na+nb
    I=np.identity(n)
    dteta1=A+(mu*I)
    dteta_inv=np.linalg.inv(dteta1)
    delta_teta=np.dot(dteta_inv,G)
    teta_new=teta+delta_teta
    e=WN(teta_new,na,y)
    SSE_new=np.dot(e.T,e)
    if np.isnan(SSE_new):
        SSE_new=10**10
    return SSE_new,delta_teta,teta_new

def step3(max_iter,mu,delta,epsilon,mu_max,na,nb,y):
    num_iter=0
```

```
    teta=step0(na,nb)
    SSE=[]
    while num_iter<max_iter:
        A,G,SSE_O=step1(delta,na,nb,teta,y)
        if num_iter == 0:
            SSE.append(SSE_O)
        SSE_new,delta_teta,teta_new=step2(A,G,mu,na,nb,teta,y)
        SSE.append(SSE_new)
        if SSE_new<SSE_O:
            if np.linalg.norm(delta_teta)<epsilon:
                teta_hat=teta_new
                var=SSE_new/(len(y)-A.shape[0])
                A_inv=np.linalg.inv(A)
                cov=var*A_inv
                return SSE,cov,teta_hat,var
            else:
                teta=teta_new
                mu=mu/10
        while SSE_new>=SSE_O:
            mu=mu*10
            if mu>mu_max:
               print('Mu\'s maximum limit is exceeded')
               return None,None,None,None
            SSE_new, delta_teta, teta_new = step2(A, G, mu, na,nb, teta, y)
        num_iter+=1
        teta = teta_new
        if num_iter>max_iter:
            print('Maximum iterations exceeded')
            return None,None,None,None


with np.errstate(divide='ignore'):
    np.float64(1.0) / 0.0
#Confidence interval
def conf_int(cov,params,na,nb):
    print("Confidence Interval:")
    for i in range(na):
        pos=params[i]+2*np.sqrt(cov[i][i])
        neg=params[i]-2*np.sqrt(cov[i][i])
        print(neg,f'<a{i+1}<',pos)
    for i in range(nb):
        pos=params[na+i]+2*np.sqrt(cov[na+i][na+i])
        neg=params[na+i]-2*np.sqrt(cov[na+i][na+i])
        print(neg,f'<b{i+1}<',pos)

#zero-poles cancellation
def zero_poles(params,na):
    y_den=[1]+list(params[:na])
    e_num=[1]+list(params[na:])
    zeros=np.roots(e_num)
    poles=np.roots(y_den)
    print("The roots of numerator are",zeros)
    print("The roots of denominator are",poles)

#Plot of SSE
def plotSSE(SSE):
    iter=np.arange(0,len(SSE))
```

```
    plt.plot(iter,SSE,label='SSE')
    plt.xlabel('Number of iterations')
    plt.ylabel('SSE')
    plt.title('SSE vs. #of Iterations')
    plt.legend()
    plt.show()

#Plot one step ahead prediction plot
def onestepplot(y,y_hat):
    plt.plot(y,label='Actual/Train')
    plt.plot(y_hat,label='one step predictions')
    plt.title('Plot of Actual/Train vs. One step prediction')
    plt.legend()
    plt.xlabel('Samples')
    plt.ylabel('Value')
    plt.show()

#Chi-square test
from scipy.stats import chi2
def chi_test(na,nb,lags,Q,e):
    DOF=lags-na-nb
    alpha=0.01
    chi_critical=chi2.ppf(1-alpha,DOF)
    if Q<chi_critical:
        print('The residuals are white')
    else:
        print('The residual is not white')
    lbvalue,pvalue=sm.stats.acorr_ljungbox(e,lags=[lags])
    print('From acorr_ljungbox test')
    print(lbvalue)
    print(pvalue)

#differencing Professor's code
def difference(y,interval=1):
    diff=[]
    for i in range(interval,len(y)):
        value=y[i]-y[i-interval]
        diff.append(value)
    return diff

#SARIMA simulation
def sarima_model():
    T=int(input('Enter number of samples'))
    mean=eval(input('Enter mean of white nosie'))
    var=eval(input('Enter variance of white noise'))
    na = int(input("Enter AR process order"))
    nb = int(input("Enter MA process order"))
    naparam = [0] * na
    nbparam = [0] * nb
    for i in range(0, na):
        naparam[i] = float(input(f"Enter the coefficient of AR:a{i + 1}"))
    for i in range(0, nb):
        nbparam[i] = float(input(f"Enter the coefficient of MA:b{i + 1}"))
    while len(naparam) < len(nbparam):
        naparam.append(0)
    while len(nbparam) < len(naparam):
        nbparam.append(0)
```

```
    ar = np.r_[1, naparam]
    ma = np.r_[1, nbparam]
    e=np.random.normal(mean,np.sqrt(var),T)
    system=(ma,ar,1)
    t,process=signal.dlsim(system,e)
    return process

#Base models
#average
def avg_one(x):
    train=[]
    for i in range(0,len(x)):
        mean=np.mean(x[0:i])
        train.append(mean)
    return train
def avg_hstep(train,test):
    forecast=np.mean(train)
    pred=[]
    for i in range(len(test)):
        pred.append(forecast)
    return pred
```

Project_RM.py:

```
import matplotlib.pyplot as plt
import numpy as np
import statsmodels.tsa.arima_model
import statsmodels.tsa.holtwinters as ets
from Toolbox import *
import pandas as pd
from sklearn.model_selection import train_test_split
from statsmodels.tsa.seasonal import STL
from sklearn.metrics import mean_squared_error
from statsmodels.stats.diagnostic import acorr_ljungbox
import statsmodels.api as sm
from numpy import linalg as la
import warnings
warnings.filterwarnings('ignore')

#6.a: Preprocessing dataset
data=pd.read_csv('energydata_complete.csv')
print('Dataset:\n',data.head())
print('Dataset information:\n',data.info())
print('Dataset don\'t have null values')
data.date=pd.to_datetime(data.date)
data.set_index('date',inplace=True)
print('Let\'s look at the end of the data:\n',data.tail())

#6.b: dependant variable vs. time
print('Dependant variable:Appliances')
plt.plot(data.index,data.Appliances,label='Dependant variable')
plt.xlabel('Time')
plt.ylabel('Appliance Energy use in Wh')
plt.title('Plot of Dependant variable (Appliances) versus Time')
plt.xticks(data.index[::1000],rotation='vertical')
```

```
plt.tight_layout()
plt.legend()
plt.show()

#6c: ACF/PACF of dependent variable
ACF_PACF_Plot(data.Appliances,400)

print('ACF plot of dependent variable (symmetric)')
acf_data=ACF(data.Appliances,100)
x=np.arange(0,100)
m=1.96/np.sqrt(len(data.Appliances))
plt.stem(x,acf_data,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.stem(-1*x,acf_data,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.xlabel('Lags')
plt.ylabel('ACF')
plt.title('ACF Plot of Original data')
plt.axhspan(-m,m,alpha = .1, color = 'yellow')
plt.tight_layout()
plt.show()

#6d:Correlation Matrix with seaborn heatmap with the Pearson's correlation
coefficient.
cor=data.corr()
sns.heatmap(cor,vmin=-1,vmax=1,center=0,cmap='PiYG')
plt.title("Heatmap of dataset", fontsize =20)
plt.show()

#6e:split the dataset into train, test set (80:20)
#split the entire dataset into train and test set
train,test=train_test_split(data,test_size=0.2,shuffle=False)

#7: stationarity test

#Check rolling mean and variance
cal_rolling_mean_var(data.Appliances,data.index)
#The mean and variance are having a flat plot but the mean is not at 0.
#It looks like data need to be differenced

#ADF test
ADF_Cal(data.Appliances)
#p-value is 0.000, the data is stationary

#kpss test
kpss_test(data.Appliances)
#Data is stationary

#seaonal Differencing is required on data to make the rolling mean at 0
#interval of 144 is used since data has spike at every 144 lags of ACF/PACF
plot
diff=difference(data.Appliances,interval=144)
diff_df=pd.DataFrame(diff,index=data.index[144:])
diff_df.rename(columns={0:'Appliances'},inplace=True)
plt.plot(diff_df.index,diff_df.Appliances,label='Differenced')
plt.xlabel('Time')
plt.ylabel('Appliance Energy use in Wh')
plt.title('Plot of Dependant variable (differenced) versus Time')
plt.xticks(data.index[::1000],rotation='vertical')
```

```
plt.tight_layout()
plt.legend()
plt.show()

#ACF/PACF
ACF_PACF_Plot(diff_df,100)

#stationarity
cal_rolling_mean_var(diff_df,diff_df.index)
ADF_Cal(diff_df)
kpss_test(diff_df)
#The data is stationary

#8:Time series Decomposition
Appliances=data['Appliances']
Appliances=pd.Series(np.array(data['Appliances']),index =
pd.date_range('2016-01-11 17:00:00',periods= len(Appliances)))
STL=STL(Appliances)
res=STL.fit()
fig=res.plot()
plt.xlabel("Iterations")
plt.tight_layout()
plt.show()

T=res.trend
S=res.seasonal
R=res.resid
plt.plot(T,label="Trend")
plt.plot(R,label="Residual")
plt.plot(S,label="Seasonal")
plt.xlabel("Iterations")
plt.ylabel("STL")
plt.legend()
plt.title("Trend, Seasonality and residuals of data")
plt.show()

#Strength of trend
var=1-(np.var(R)/np.var(T+R))
Ft=np.max([0,var])
print("Strength of trend:",Ft)

#Strength of seasonality
var1=1-(np.var(R)/np.var(S+R))
Fs=np.max([0,var1])
print("Strength of seasonality:",Fs)

#seasonally adjusted data
seasonally_adj=Appliances-S
plt.plot(data.index,data.Appliances,label="Original")
plt.plot(data.index,seasonally_adj,label="adjusted")
plt.xlabel("Time")
plt.xticks(data.index[::4000])
plt.ylabel("Appliances")
plt.title("Seasonally adjusted vs. Original")
plt.legend()
plt.show()
```

```
#detrended data
detrended=Appliances-T
plt.plot(data.index,data.Appliances,label="Original")
plt.plot(data.index,detrended,label="Detrended")
plt.xlabel("Time")
plt.xticks(data.index[::4000])
plt.ylabel("Appliances")
plt.title("Detrended vs. Original Data")
plt.legend()
plt.show()

# Holt-Winters method
print(data.columns)
X=data[['lights', 'T1', 'RH_1', 'T2', 'RH_2', 'T3', 'RH_3', 'T4',
       'RH_4', 'T5', 'RH_5', 'T6', 'RH_6', 'T7', 'RH_7', 'T8', 'RH_8', 'T9',
       'RH_9', 'T_out', 'Press_mm_hg', 'RH_out', 'Windspeed', 'Visibility',
       'Tdewpoint', 'rv1', 'rv2']]
y=data['Appliances']
x_train,x_test,y_train,y_test=train_test_split(X,y,test_size=0.2,shuffle=Fals
e)
holtw= ets.ExponentialSmoothing(y_train, damped_trend= True,trend='add',
seasonal='add',seasonal_periods=144).fit()

#HW prediction on train set
holtw_predt=holtw.forecast(steps=len(y_train))
holtw_dft=pd.DataFrame(holtw_predt,columns=['Appliances']).set_index(y_train.
index)

#HW prediction on test set
holtw_pred=holtw.forecast(steps=len(y_test))
holtw_df=pd.DataFrame(holtw_pred,columns=['Appliances']).set_index(y_test.ind
ex)

#plot of HW model
plt.figure(figsize=(8,8))
plt.plot(y_train.index,y_train, label='Train')
plt.plot(y_test.index,y_test, label='Test')
plt.plot(holtw_df.index,holtw_df.Appliances,label='Holts winter prediction')
plt.legend()
plt.xlabel("Time")
plt.ylabel("Appliances")
plt.title("Holts winter model")
plt.show()

#Model performance on train and test data

#MSE
HW_train_mse=mean_squared_error(y_train,holtw_dft.Appliances)
print('MSE of Holts Winter method on train data:',HW_train_mse)
HW_test_mse=mean_squared_error(y_test,holtw_df.Appliances)
print('MSE of Holts Winter method on test data:',HW_test_mse)

#residual error
HW_reserror=y_train-holtw_dft.Appliances

#Forecast error
HW_foerror=y_test-holtw_df.Appliances
```

```
#ACF
acf_hw_res=ACF(HW_reserror.values,70)
x=np.arange(0,70)
m=1.96/np.sqrt(len(y_test))
plt.stem(x,acf_hw_res,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.stem(-1*x,acf_hw_res,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.xlabel('Lags')
plt.ylabel('ACF')
plt.title('ACF Plot of Residuals (HW)')
plt.axhspan(-m,m,alpha = .1, color = 'yellow')
plt.tight_layout()
plt.show()

acf_hw_fore=ACF(HW_foerror.values,70)
x=np.arange(0,70)
m=1.96/np.sqrt(len(y_test))
plt.stem(x,acf_hw_fore,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.stem(-1*x,acf_hw_fore,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.xlabel('Lags')
plt.ylabel('ACF')
plt.title('ACF Plot of Forecast error (HW)')
plt.axhspan(-m,m,alpha = .1, color = 'yellow')
plt.tight_layout()
plt.show()

#Q-value
hotl_q_t=sm.stats.acorr_ljungbox(HW_reserror, lags=5,return_df=True)
print('Q-value (residual):',hotl_q_t)
lbvalue=sm.stats.acorr_ljungbox(HW_foerror,lags=5,return_df=True)
print('Q-value (Forecast):\n',lbvalue)

#Error mean and variance
print('Holts winter: Mean of residual error is',np.mean(HW_reserror),'and
Forecast error is',np.mean(HW_foerror))
print('Holts winter: Variance of residual error is',np.var(HW_reserror),'and
Forecast error is',np.var(HW_foerror))

#RMSE
HW_train_rmse=mean_squared_error(y_train,holtw_dft.Appliances,squared=False)
print('RMSE of Holts Winter method on train data:',HW_train_rmse)
HW_test_rmse=mean_squared_error(y_test,holtw_df.Appliances,squared=False)
print('RMSE of Holts Winter method on test data:',HW_test_rmse)

#10: Feature selection and collinearity
X_mat=x_train.values
Y=y_train.values
X_svd=sm.add_constant(X_mat)
H=np.matmul(X_svd.T,X_svd)
s,d,v=np.linalg.svd(H)
print('Singular Values: ',d)

#Condition number
print("The condition number is ",la.cond(X_svd))
#the condion number is high. Collinearity exists, so we need to remove the
features using backward regression
```

```
#Feature selection
x_train_ols=sm.add_constant(x_train)
model=sm.OLS(y_train,x_train_ols).fit()
print(model.summary())

#Remove rv1 and rv2 - high p-value
x_train_ols.drop(['rv1','rv2'],axis=1,inplace=True)
model=sm.OLS(y_train,x_train_ols).fit()
print(model.summary())

#Remove const - high p-value
x_train_ols.drop(['const'],axis=1,inplace=True)
model=sm.OLS(y_train,x_train_ols).fit()
print(model.summary())

#Remove T7 - high p-value
x_train_ols.drop(['T7'],axis=1,inplace=True)
model=sm.OLS(y_train,x_train_ols).fit()
print(model.summary())

#Remove RH_7 - high p-value
x_train_ols.drop(['RH_7'],axis=1,inplace=True)
model=sm.OLS(y_train,x_train_ols).fit()
print(model.summary())


#Remove RH_5 - high p-value
x_train_ols.drop(['RH_5'],axis=1,inplace=True)
model=sm.OLS(y_train,x_train_ols).fit()
print(model.summary())

#Remove T5 - high p-value
x_train_ols.drop(['T5'],axis=1,inplace=True)
model=sm.OLS(y_train,x_train_ols).fit()
print(model.summary())

#Remove RH_out  - high p-value
x_train_ols.drop(['RH_out'],axis=1,inplace=True)
model=sm.OLS(y_train,x_train_ols).fit()
print(model.summary())

#Remove Tdewpoint  - high p-value
x_train_ols.drop(['Tdewpoint'],axis=1,inplace=True)
model=sm.OLS(y_train,x_train_ols).fit()
print(model.summary())

#Remove RH_9  - high p-value
x_train_ols.drop(['RH_9'],axis=1,inplace=True)
model=sm.OLS(y_train,x_train_ols).fit()
print(model.summary())

#Remove T1-high std. error
x_train_ols.drop(['T1'],axis=1,inplace=True)
model=sm.OLS(y_train,x_train_ols).fit()
print(model.summary())

#Remove Press_mm_hg- high p-value
```

```
x_train_ols.drop(['Press_mm_hg'],axis=1,inplace=True)
model=sm.OLS(y_train,x_train_ols).fit()
print(model.summary())

#The collinearity is removed and the best model is defined above

#12. Multiple Linear Regression
print('The best features are',x_train_ols.columns)

#Prediction on train data
pred_train=model.predict(x_train[['lights', 'RH_1', 'T2', 'RH_2', 'T3',
'RH_3', 'T4', 'RH_4', 'T6',
       'RH_6', 'T8', 'RH_8', 'T9', 'T_out', 'Windspeed', 'Visibility']])

#Residual error
ml_res=y_train-pred_train

#Prediction on test data
pred_test=model.predict(x_test[['lights', 'RH_1', 'T2', 'RH_2', 'T3', 'RH_3',
'T4', 'RH_4', 'T6',
       'RH_6', 'T8', 'RH_8', 'T9', 'T_out', 'Windspeed', 'Visibility']])

#Forecast error
ml_fore=y_test-pred_test

#Plot of train data
plt.figure(figsize=(6,6))
plt.plot(y_train.index,y_train, label='Train')
plt.plot(pred_train.index,pred_train, label='Predicted')
plt.legend()
plt.xlabel("Time")
plt.ylabel("Appliances")
plt.title("Predictions on Train set")
plt.show()

#Plot of test data
plt.figure(figsize=(6,6))
plt.plot(y_test.index,y_test, label='Test')
plt.plot(pred_test.index,pred_test, label='Forecasted')
plt.legend()
plt.xlabel("Time")
plt.xticks(rotation=70)
plt.ylabel("Appliances")
plt.title("Predictions on test set")
plt.show()

#Plot of train and test data
plt.figure(figsize=(8,8))
plt.plot(y_train.index,y_train, label='Train')
plt.plot(y_test.index,y_test, label='Test')
plt.plot(pred_test.index,pred_test, label='Forecasted')
plt.legend()
plt.xlabel("Time")
plt.ylabel("Appliances")
plt.title("Train vs. Test vs. Forecasted Data")
plt.show()
```

```
#ACF
acf_ml_train=ACF(ml_res,100)
x=np.arange(0,100)
m=1.96/np.sqrt(len(y_train))
plt.stem(x,acf_ml_train,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.stem(-1*x,acf_ml_train,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.xlabel('Lags')
plt.ylabel('ACF')
plt.title('ACF Plot of Residuals')
plt.axhspan(-m,m,alpha = .1, color = 'yellow')
plt.tight_layout()
plt.show()

acf_ml_test=ACF(ml_fore.values,100)
x=np.arange(0,100)
m=1.96/np.sqrt(len(y_test))
plt.stem(x,acf_ml_test,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.stem(-1*x,acf_ml_test,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.xlabel('Lags')
plt.ylabel('ACF')
plt.title('ACF Plot of Forecast (ML)')
plt.axhspan(-m,m,alpha = .1, color = 'yellow')
plt.tight_layout()
plt.show()

#Model performance on train and test data

#MSE
ML_train_mse=mean_squared_error(y_train,pred_train)
print('MSE of MLR on train data:',ML_train_mse)
ML_test_mse=mean_squared_error(y_test,pred_test)
print('MSE of MLR on test data:',ML_test_mse)

#Q-value
q_ml_train=sm.stats.acorr_ljungbox(ml_res, lags=5, return_df=True)
print('Q-value (residual):',q_ml_train)
q_mltest=sm.stats.acorr_ljungbox(ml_fore,lags=5,return_df=True)
print('Q-value (Forecast):\n',q_mltest)

#Error mean and variance
print('MLR: Mean of residual error is',np.mean(ml_res),'and Forecast error
is',np.mean(ml_fore))
print('MLR: Variance of residual error is',np.var(ml_res),'and Forecast error
is',np.var(ml_fore))

#RMSE
ml_train_rmse=mean_squared_error(y_train,pred_train,squared=False)
print('RMSE of MLR method on train data:',ml_train_rmse)
ml_test_rmse=mean_squared_error(y_test,pred_test,squared=False)
print('RMSE of MLR method on test data:',ml_test_rmse)

#Base models
#Average method
train_pred_avg=avg_one(y_train)
test_pred_avg=avg_hstep(y_train,y_test)

#Plot of average method
```

```
plt.plot(y_train.index,y_train,label='Train')
plt.plot(y_test.index,y_test,label='Test')
plt.plot(y_test.index,test_pred_avg,label='Predicted')
plt.xlabel('Time')
plt.xticks(rotation='vertical')
plt.ylabel('Appliances')
plt.title('Average method predictions')
plt.legend()
plt.show()

#Plot of test vs predicted
plt.plot(y_test.index,y_test,label='Test')
plt.plot(y_test.index,test_pred_avg,label='Forecasted')
plt.xlabel('Time')
plt.xticks(rotation='vertical')
plt.ylabel('Appliances')
plt.title('Average method Forecast')
plt.legend()
plt.show()

#residual and forecast error
avg_res=y_train-train_pred_avg
avg_fore=y_test-test_pred_avg

#ACF
acf_avg_train=ACF(avg_res.values[1:],100)
x=np.arange(0,100)
m=1.96/np.sqrt(len(y_train))
plt.stem(x,acf_avg_train,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.stem(-1*x,acf_avg_train,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.xlabel('Lags')
plt.ylabel('ACF')
plt.title('ACF Plot of Residuals')
plt.axhspan(-m,m,alpha = .1, color = 'yellow')
plt.tight_layout()
plt.show()

acf_avg_test=ACF(avg_fore.values,100)
x=np.arange(0,100)
m=1.96/np.sqrt(len(y_test))
plt.stem(x,acf_avg_test,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.stem(-1*x,acf_avg_test,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.xlabel('Lags')
plt.ylabel('ACF')
plt.title('ACF Plot of Forecast (Average)')
plt.axhspan(-m,m,alpha = .1, color = 'yellow')
plt.tight_layout()
plt.show()

#Model performance on train and test data

#MSE
avg_train_mse=mean_squared_error(y_train[1:],train_pred_avg[1:])
print('MSE of Average on train data:',avg_train_mse)
avg_test_mse=mean_squared_error(y_test,test_pred_avg)
print('MSE of Average on test data:',avg_test_mse)
```

```
#Q-value
q_avg_train=acorr_ljungbox(avg_res.values[1:], lags=5,
boxpierce=True,return_df=True)
print('Q-value (residual):',q_avg_train)
q_avgtest=sm.stats.acorr_ljungbox(avg_fore.values[1:],lags=5,boxpierce=True,r
eturn_df=True)
print('Q-value (Forecast):\n',q_avgtest)

#Error mean and variance
print('Average: Mean of residual error is',np.mean(avg_res),'and Forecast
error is',np.mean(avg_fore))
print('Average: Variance of residual error is',np.var(avg_res),'and Forecast
error is',np.var(avg_fore))

#RMSE
avg_train_rmse=mean_squared_error(y_train[1:],pred_train[1:],squared=False)
print('RMSE of Average method on train data:',avg_train_rmse)
avg_test_rmse=mean_squared_error(y_test,pred_test,squared=False)
print('RMSE of Average method on test data:',avg_test_rmse)


#Naive method
train_naive=[]
for i in range(len(y_train[1:])):
    train_naive.append(y_train.values[i-1])

test_naive=[y_train.values[-1] for i in y_test]
naive_fore= pd.DataFrame(test_naive).set_index(y_test.index)

#Plot of naive method
plt.plot(y_train.index,y_train,label='Train')
plt.plot(y_test.index,y_test,label='Test')
plt.plot(y_test.index,test_naive,label='Predicted')
plt.xlabel('Time')
plt.xticks(rotation='vertical')
plt.ylabel('Appliances')
plt.title('Naive method predictions')
plt.legend()
plt.show()

#Plot of test vs predicted
plt.plot(y_test.index,y_test,label='Test')
plt.plot(y_test.index,test_naive,label='Predicted')
plt.xlabel('Time')
plt.xticks(rotation='vertical')
plt.ylabel('Appliances')
plt.title('Naive method predictions')
plt.legend()
plt.show()

#residual and forecast error
naive_res=y_train[1:]-train_naive
naive_fore=y_test-test_naive

#ACF
acf_naive_train=ACF(naive_res.values,100)
x=np.arange(0,100)
```

```
m=1.96/np.sqrt(len(y_train))
plt.stem(x,acf_naive_train,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.stem(-1*x,acf_naive_train,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.xlabel('Lags')
plt.ylabel('ACF')
plt.title('ACF Plot of Residuals')
plt.axhspan(-m,m,alpha = .1, color = 'yellow')
plt.tight_layout()
plt.show()

acf_naive_test=ACF(naive_fore.values,100)
x=np.arange(0,100)
m=1.96/np.sqrt(len(y_test))
plt.stem(x,acf_naive_test,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.stem(-1*x,acf_naive_test,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.xlabel('Lags')
plt.ylabel('ACF')
plt.title('ACF Plot of Forecast (Naive)')
plt.axhspan(-m,m,alpha = .1, color = 'yellow')
plt.tight_layout()
plt.show()

#Model performance on train and test data

#MSE
naive_train_mse=mean_squared_error(y_train[1:],train_naive)
print('MSE of Naive on train data:',naive_train_mse)
naive_test_mse=mean_squared_error(y_test,test_naive)
print('MSE of Naive on test data:',naive_test_mse)

#Q-value
q_naive_train=acorr_ljungbox(naive_res, lags=5, boxpierce=True,
return_df=True)
print('Q-value (residual):',q_naive_train)
q_naivetest=acorr_ljungbox(naive_fore,lags=5,boxpierce=True,return_df=True)
print('Q-value (Forecast):\n',q_naivetest)

#Error mean and variance
print('Naive: Mean of residual error is',np.mean(naive_res),'and Forecast
error is',np.mean(naive_fore))
print('Naive: Variance of residual error is',np.var(naive_res),'and Forecast
error is',np.var(naive_fore))

#RMSE
naive_train_rmse=mean_squared_error(y_train[1:],train_naive,squared=False)
print('RMSE of Naive method on train data:',naive_train_rmse)
naive_test_rmse=mean_squared_error(y_test,test_naive,squared=False)
print('RMSE of Naive method on test data:',naive_test_rmse)

#Drift method
train_drift = []
value = 0
for i in range(len(y_train)):
    if i > 1:
        slope_val = (y_train[i - 1]-y_train[0]) / (i-1)
        y_predict = (slope_val * i) + y_train[0]
        train_drift.append(y_predict)
```

```
    else:
        continue

test_drift= []
for h in range(len(y_test)):
    slope_val = (y_train.values[-1] - y_train.values[0] ) /( len(y_train) - 1
)
    y_predict= y_train.values[-1] + ((h +1) * slope_val)
    test_drift.append(y_predict)

#Plot of drift method
plt.plot(y_train.index,y_train,label='Train')
plt.plot(y_test.index,y_test,label='Test')
plt.plot(y_test.index,test_drift,label='Predicted')
plt.xlabel('Time')
plt.xticks(rotation='vertical')
plt.ylabel('Appliances')
plt.title('Drift method predictions')
plt.legend()
plt.show()

#Plot of test vs predicted
plt.plot(y_test.index,y_test,label='Test')
plt.plot(y_test.index,test_drift,label='Forecasted')
plt.xlabel('Time')
plt.xticks(rotation='vertical')
plt.ylabel('Appliances')
plt.title('Drift method forecast')
plt.legend()
plt.show()

#residual and forecast error
drift_res=y_train[2:]-train_drift
drift_fore=y_test-test_drift

#ACF
acf_drift_train=ACF(drift_res.values,100)
x=np.arange(0,100)
m=1.96/np.sqrt(len(y_train))
plt.stem(x,acf_drift_train,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.stem(-1*x,acf_drift_train,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.xlabel('Lags')
plt.ylabel('ACF')
plt.title('ACF Plot of Residuals')
plt.axhspan(-m,m,alpha = .1, color = 'yellow')
plt.tight_layout()
plt.show()

acf_drift_test=ACF(drift_fore.values,100)
x=np.arange(0,100)
m=1.96/np.sqrt(len(y_test))
plt.stem(x,acf_drift_test,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.stem(-1*x,acf_drift_test,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.xlabel('Lags')
plt.ylabel('ACF')
plt.title('ACF Plot of Forecast (Drift)')
plt.axhspan(-m,m,alpha = .1, color = 'yellow')
```

```
plt.tight_layout()
plt.show()

#Model performance on train and test data

#MSE
drift_train_mse=mean_squared_error(y_train[2:],train_drift)
print('MSE of Drift on train data:',drift_train_mse)
drift_test_mse=mean_squared_error(y_test,test_drift)
print('MSE of Drift on test data:',drift_test_mse)

#Q-value
q_drift_train=acorr_ljungbox(drift_res, lags=5, boxpierce=True,
return_df=True)
print('Q-value (residual):',q_drift_train)
q_drifttest=acorr_ljungbox(drift_fore,lags=5,boxpierce=True,return_df=True)
print('Q-value (Forecast):\n',q_drifttest)

#Error mean and variance
print('Drift: Mean of residual error is',np.mean(drift_res),'and Forecast
error is',np.mean(drift_fore))
print('Drift: Variance of residual error is',np.var(drift_res),'and Forecast
error is',np.var(drift_fore))

#RMSE
drift_train_rmse=mean_squared_error(y_train[2:],train_drift,squared=False)
print('RMSE of Drift method on train data:',drift_train_rmse)
drift_test_rmse=mean_squared_error(y_test,test_drift,squared=False)
print('RMSE of Drift method on test data:',drift_test_rmse)

#SES
ses=
ets.ExponentialSmoothing(y_train,trend=None,damped_trend=False,seasonal=None)
.fit(smoothing_level=0.5)
train_ses= ses.forecast(steps=len(y_train))
train_ses=pd.DataFrame(train_ses).set_index(y_train.index)

test_ses= ses.forecast(steps=len(y_test))
test_ses=pd.DataFrame(test_ses).set_index(y_test.index)

#Plot of SES method
plt.plot(y_train.index,y_train,label='Train')
plt.plot(y_test.index,y_test,label='Test')
plt.plot(y_test.index,test_ses[0],label='Predicted')
plt.xlabel('Time')
plt.xticks(rotation='vertical')
plt.ylabel('Appliances')
plt.title('SES method predictions')
plt.legend()
plt.show()

#Plot of test vs predicted
plt.plot(y_test.index,y_test,label='Test')
plt.plot(y_test.index,test_ses[0],label='Forecasted')
plt.xlabel('Time')
plt.xticks(rotation='vertical')
plt.ylabel('Appliances')
```

```
plt.title('SES method forecast')
plt.legend()
plt.show()

#residual and forecast error
ses_res=y_train[2:]-train_ses[0]
ses_fore=y_test-test_ses[0]

#ACF
acf_ses_train=ACF(ses_res.values[2:],100)
x=np.arange(0,100)
m=1.96/np.sqrt(len(y_train))
plt.stem(x,acf_ses_train,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.stem(-1*x,acf_ses_train,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.xlabel('Lags')
plt.ylabel('ACF')
plt.title('ACF Plot of Residuals')
plt.axhspan(-m,m,alpha = .1, color = 'yellow')
plt.tight_layout()
plt.show()

acf_ses_test=ACF(ses_fore.values,100)
x=np.arange(0,100)
m=1.96/np.sqrt(len(y_test))
plt.stem(x,acf_ses_test,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.stem(-1*x,acf_ses_test,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.xlabel('Lags')
plt.ylabel('ACF')
plt.title('ACF Plot of Forecast (SES)')
plt.axhspan(-m,m,alpha = .1, color = 'yellow')
plt.tight_layout()
plt.show()

#Model performance on train and test data

#MSE
ses_train_mse=mean_squared_error(y_train,train_ses)
print('MSE of SES on train data:',ses_train_mse)
ses_test_mse=mean_squared_error(y_test,test_ses)
print('MSE of SES on test data:',ses_test_mse)

#Q-value
q_ses_train=acorr_ljungbox(ses_res[2:], lags=5, boxpierce=True,
return_df=True)
print('Q-value (residual):',q_ses_train)
q_sestest=acorr_ljungbox(ses_fore,lags=5,boxpierce=True,return_df=True)
print('Q-value (Forecast):\n',q_sestest)

#Error mean and variance
print('SES: Mean of residual error is',np.mean(ses_res),'and Forecast error
is',np.mean(ses_fore))
print('SES: Variance of residual error is',np.var(ses_res),'and Forecast
error is',np.var(ses_fore))

#RMSE
ses_train_rmse=mean_squared_error(y_train,train_ses,squared=False)
print('RMSE of SES method on train data:',ses_train_rmse)
```

```
ses_test_rmse=mean_squared_error(y_test,test_ses,squared=False)
print('RMSE of SES method on test data:',ses_test_rmse)

#13: ARMA models

#Order determination
diff_train,diff_test=train_test_split(diff_df,test_size=0.2,shuffle=False)

#ACF
ACF_PACF_Plot(diff_df,50)
acf_gpac=ACF(diff_train.Appliances.values,50)
x=np.arange(0,50)
m=1.96/np.sqrt(len(diff_df))
plt.stem(x,acf_gpac,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.stem(-1*x,acf_gpac,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.xlabel('Lags')
plt.ylabel('ACF')
plt.title('ACF Plot for GPAC')
plt.axhspan(-m,m,alpha = .1, color = 'yellow')
plt.tight_layout()
plt.show()

#GPAC
GPAC(acf_gpac,10,10)

#order ARMA(3,0) and ARMA(3,1)

#14: LMA algorithm
na=3
nb=0
#ARMA model
arma_30=sm.tsa.ARMA(y_train,(na,nb)).fit(trend='nc',disp=0)

#Estimated parameters
for i in range(na):
    print(f"The AR coefficient a{i} is:",arma_30.params[i])
for i in range(nb):
    print(f"The MA coefficient a{i} is:",arma_30.params[i + na])
print(arma_30.summary())

#confidence interval
print('Confidence interval:\n',arma_30.conf_int())
#Estimators are significant does not include zero in them

#Prediction on train set
arma30_train=arma_30.predict(start=0,end=15787)
arma30_res=y_train-arma30_train

#Forecast
arma30_test=arma_30.predict(start=15788, end=19734)
arma30_fore=y_test-arma30_test


#ACF
acf_arma30_train=ACF(arma30_res,100)
x=np.arange(0,100)
m=1.96/np.sqrt(len(y_train))
```

```
plt.stem(x,acf_arma30_train,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.stem(-1*x,acf_arma30_train,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.xlabel('Lags')
plt.ylabel('ACF')
plt.title('ACF Plot of Residuals')
plt.axhspan(-m,m,alpha = .1, color = 'yellow')
plt.tight_layout()
plt.show()

acf_arma30_test=ACF(arma30_fore.values,100)
x=np.arange(0,100)
m=1.96/np.sqrt(len(y_test))
plt.stem(x,acf_arma30_test,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.stem(-1*x,acf_arma30_test,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.xlabel('Lags')
plt.ylabel('ACF')
plt.title('ACF Plot of Forecast (ARMA(3,0))')
plt.axhspan(-m,m,alpha = .1, color = 'yellow')
plt.tight_layout()
plt.show()

#Model performance on train and test data

#MSE
arma30_train_mse=mean_squared_error(y_train,arma30_train)
print('MSE of ARMA(3,0) on train data:',arma30_train_mse)
arma30_test_mse=mean_squared_error(y_test,arma30_test)
print('MSE of ARMA(3,0) on test data:',arma30_test_mse)

#Q-value
q_arma30_train=acorr_ljungbox(arma30_res, lags=5, boxpierce=True,
return_df=True)
print('Q-value (residual):',q_arma30_train)
q_ar30test=acorr_ljungbox(arma30_fore,lags=5,boxpierce=True,return_df=True)
print('Q-value (Forecast):\n',q_ar30test)

#Error mean and variance
print('ARMA(3,0): Mean of residual error is',np.mean(arma30_res),'and
Forecast error is',np.mean(arma30_fore))
print('ARMA(3,0): Variance of residual error is',np.var(arma30_res),'and
Forecast error is',np.var(arma30_fore))

#Covariance matrix
print('Covariance matrix\n',arma_30.cov_params())

#standard error
print('Standard error:',arma_30.bse)

#RMSE
ar30_train_rmse=mean_squared_error(y_train,arma30_train,squared=False)
print('RMSE of ARMA(3,0) method on train data:',ar30_train_rmse)
ar30_test_rmse=mean_squared_error(y_test,arma30_test,squared=False)
print('RMSE of ARMA(3,0) method on test data:',ar30_test_rmse)


#Plot of ARMA(3,0) method
plt.plot(y_train.index,y_train,label='Train')
```

```
plt.plot(y_test.index,y_test,label='Test')
plt.plot(y_test.index,arma30_test,label='Predicted')
plt.xlabel('Time')
plt.xticks(rotation='vertical')
plt.ylabel('Appliances')
plt.title('ARMA(3,0) method predictions')
plt.legend()
plt.show()

#Plot of test vs forecasted
plt.plot(y_test.index,y_test,label='Test')
plt.plot(y_test.index,arma30_test,label='Forecasted')
plt.xlabel('Time')
plt.xticks(rotation='vertical')
plt.ylabel('Appliances')
plt.title('ARMA(3,0) method forecast')
plt.legend()
plt.show()

#Plot of train vs predicted
plt.plot(y_train.index,y_train,label='Train')
plt.plot(y_train.index,arma30_train,label='Predicted')
plt.xlabel('Time')
plt.xticks(rotation='vertical')
plt.ylabel('Appliances')
plt.title('ARMA(3,0) method predictions')
plt.legend()
plt.show()

#The model didn't work well on test data let's try other model.
na = 3
nb = 1
# ARMA model
arma31= sm.tsa.ARMA(y_train,(na, nb)).fit(trend='nc',disp=0)
print(arma31.summary())

#Estimated parameters
for i in range(na):
    print(f"The AR coefficient a{i} is:",arma31.params[i])
for i in range(nb):
    print(f"The MA coefficient a{i} is:",arma31.params[i + na])


#confidence interval
print('Confidence interval:\n',arma31.conf_int())
#Estimators are significant does not include zero in them

#Prediction on train set
arma31_train=arma31.predict(start=0,end=15787)
arma31_res=y_train-arma31_train

#Forecast
arma31_test=arma31.predict(start=15788, end=19734)
arma31_fore=y_test-arma31_test


#ACF
```

```
acf_arma31_train=ACF(arma31_res,100)
x=np.arange(0,100)
m=1.96/np.sqrt(len(y_train))
plt.stem(x,acf_arma31_train,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.stem(-1*x,acf_arma31_train,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.xlabel('Lags')
plt.ylabel('ACF')
plt.title('ACF Plot of Residuals')
plt.axhspan(-m,m,alpha = .1, color = 'yellow')
plt.tight_layout()
plt.show()

acf_arma31_test=ACF(arma31_fore.values,100)
x=np.arange(0,100)
m=1.96/np.sqrt(len(y_test))
plt.stem(x,acf_arma31_test,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.stem(-1*x,acf_arma31_test,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.xlabel('Lags')
plt.ylabel('ACF')
plt.title('ACF Plot of Forecast (ARMA(3,1))')
plt.axhspan(-m,m,alpha = .1, color = 'yellow')
plt.tight_layout()
plt.show()

#Model performance on train and test data

#MSE
arma31_train_mse=mean_squared_error(y_train,arma31_train)
print('MSE of ARMA(3,1) on train data:',arma31_train_mse)
arma31_test_mse=mean_squared_error(y_test,arma31_test)
print('MSE of ARMA(3,1) on test data:',arma31_test_mse)

#Q-value
q_arma31_train=acorr_ljungbox(arma31_res, lags=5, boxpierce=True,
return_df=True)
print('Q-value (residual):',q_arma31_train)
q_ar31test=acorr_ljungbox(arma30_fore,lags=5,boxpierce=True,return_df=True)
print('Q-value (Forecast):\n',q_ar31test)

#Error mean and variance
print('ARMA(3,1): Mean of residual error is',np.mean(arma31_res),'and
Forecast error is',np.mean(arma31_fore))
print('ARMA(3,1): Variance of residual error is',np.var(arma31_res),'and
Forecast error is',np.var(arma31_fore))

#Covariance matrix
print('Covariance matrix\n',arma31.cov_params())

#standard error
print('Standard error:',arma31.bse)

#RMSE
ar31_train_rmse=mean_squared_error(y_train,arma31_train,squared=False)
print('RMSE of ARMA(3,1) method on train data:',ar31_train_rmse)
ar31_test_rmse=mean_squared_error(y_test,arma31_test,squared=False)
print('RMSE of ARMA(3,1) method on test data:',ar31_test_rmse)
```

```
#Plot of ARMA(3,1) method
plt.plot(y_train.index,y_train,label='Train')
plt.plot(y_test.index,y_test,label='Test')
plt.plot(y_test.index,arma31_test,label='Predicted')
plt.xlabel('Time')
plt.xticks(rotation='vertical')
plt.ylabel('Appliances')
plt.title('ARMA(3,1) method predictions')
plt.legend()
plt.show()

#Plot of test vs forecasted
plt.plot(y_test.index,y_test,label='Test')
plt.plot(y_test.index,arma31_test,label='Forecasted')
plt.xlabel('Time')
plt.xticks(rotation='vertical')
plt.ylabel('Appliances')
plt.title('ARMA(3,1) method forecast')
plt.legend()
plt.show()

#Plot of train vs predicted
plt.plot(y_train.index,y_train,label='Train')
plt.plot(y_train.index,arma31_train,label='Predicted')
plt.xlabel('Time')
plt.xticks(rotation='vertical')
plt.ylabel('Appliances')
plt.title('ARMA(3,1) method predictions')
plt.legend()
plt.show()


#Let's try SARIMA

#SARIMA= (3,0,0)X(0,1,0,12)
#Seasonality of 144 is not computational there exists a memory error so
reduce it to 12
sarima=
sm.tsa.statespace.SARIMAX(y_train,order=(3,0,0),seasonal_order=(0,1,0,12),
                                    enforce_stationarity=False,
                                    enforce_invertibility=False)
results=sarima.fit()
print(results.summary())

#predictions on train data
sarima_train=results.get_prediction(start=0, end=len(y_train), dynamic=False)
Sarima_pred=sarima_train.predicted_mean
Sarima_res= y_train-Sarima_pred.values[1:]

#forecast
sarima_test=results.predict(start=0, end=(len(y_test)))
sarima_fore=y_test-sarima_test.values[1:]

#ACF

#ACF
```

```
acf_sarima_train=ACF(Sarima_res.values,100)
x=np.arange(0,100)
m=1.96/np.sqrt(len(y_train))
plt.stem(x,acf_sarima_train,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.stem(-1*x,acf_sarima_train,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.xlabel('Lags')
plt.ylabel('ACF')
plt.title('ACF Plot of Residuals')
plt.axhspan(-m,m,alpha = .1, color = 'yellow')
plt.tight_layout()
plt.show()

acf_sarima_test=ACF(sarima_fore.values,100)
x=np.arange(0,100)
m=1.96/np.sqrt(len(y_test))
plt.stem(x,acf_sarima_test,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.stem(-1*x,acf_sarima_test,linefmt='r-', markerfmt='bo', basefmt='b-')
plt.xlabel('Lags')
plt.ylabel('ACF')
plt.title('ACF Plot of Forecast (SARMA(3,0,0)X(0,1,0,12))')
plt.axhspan(-m,m,alpha = .1, color = 'yellow')
plt.tight_layout()
plt.show()

#Model performance on train and test data

#MSE
sarima_train_mse=mean_squared_error(y_train,Sarima_pred[1:])
print('MSE of SARIMA on train data:',sarima_train_mse)
sarima_test_mse=mean_squared_error(y_test,sarima_test[1:])
print('MSE of SARIMA on test data:',sarima_test_mse)

#Q-value
q_sarima_train=acorr_ljungbox(Sarima_res, lags=5, boxpierce=True,
return_df=True)
print('Q-value (residual):',q_sarima_train)
q_sarimatest=acorr_ljungbox(sarima_fore,lags=5,boxpierce=True,return_df=True)
print('Q-value (Forecast):\n',q_sarimatest)


#Error mean and variance
print('SARIMA: Mean of residual error is',np.mean(Sarima_res),'and Forecast
error is',np.mean(sarima_fore))
print('SARIMA: Variance of residual error is',np.var(Sarima_res),'and
Forecast error is',np.var(sarima_fore))

#Covariance matrix
print('Covariance matrix\n',results.cov_params())

#standard error
print('Standard error:',results.bse)

#RMSE
sarima_train_rmse=mean_squared_error(y_train,Sarima_pred[1:],squared=False)
print('RMSE of SARIMA method on train data:',sarima_train_rmse)
sarima_test_rmse=mean_squared_error(y_test,sarima_test[1:],squared=False)
print('RMSE of SARIMA method on test data:',sarima_test_rmse)
```

```
#Plot of SARIMA method
plt.plot(y_train.index,y_train,label='Train')
plt.plot(y_test.index,y_test,label='Test')
plt.plot(y_test.index,sarima_test[1:],label='Predicted')
plt.xlabel('Time')
plt.xticks(rotation='vertical')
plt.ylabel('Appliances')
plt.title('SARIMA method predictions')
plt.legend()
plt.show()

#Plot of test vs forecasted
plt.plot(y_test.index,y_test,label='Test')
plt.plot(y_test.index,sarima_test[1:],label='Forecasted')
plt.xlabel('Time')
plt.xticks(rotation='vertical')
plt.ylabel('Appliances')
plt.title('SARIMA method forecast')
plt.legend()
plt.show()

#Plot of train vs predicted
plt.plot(y_train.index,y_train,label='Train')
plt.plot(y_train.index,Sarima_pred[1:],label='Predicted')
plt.xlabel('Time')
plt.xticks(rotation='vertical')
plt.ylabel('Appliances')
plt.title('SARIMA method predictions')
plt.legend()
plt.show()


#16:The best model with less MSE is Multiple Linear Regression

#17: Forecast function can be written as
print(model.summary())
#y_hat_t= 2.1396*lights + 13.8562*RH_1 + (-17.11)*T2 + (-13.2627)*RH_2 +
25.5010*T3
#         +8.6120*RH_3 + (-3.9367)*T4 + (-2.8629)*RH_4 + (8.3984)*T6 +
(0.3553)*RH_6
#         + 8.5673*T8 + (-6.3745)*RH_8 + (-13.4323)*T9 + (-6.1772)*T_out
#         + 1.0866 *Windspeed + 0.2045*Visibility

#18: H-step ahead prediction

#predictions of test data for MLR was performed previously (see at MLR part
of code)
#Let's plot the test vs forecasted values of MLR
plt.figure(figsize=(6,6))
plt.plot(y_test.index,y_test, label='Test')
plt.plot(pred_test.index,pred_test, label='Forecasted')
plt.legend()
plt.xlabel("Time")
plt.xticks(rotation=70)
plt.ylabel("Appliances")
```

```
plt.title("Predictions on test set")
plt.show()
```

REFERENCES

- Luis M. Candanedo, Véronique Feldheim, Dominique Deramaix. Energy and Buildings, Volume 140, 1 April 2017, Pages 81-97, ISSN 0378-7788, http://dx.doi.org/10.1016/j.enbuild.2017.01.083
- https://github.com/LuisM78/Appliances-energy-prediction-data