THE GEORGE WASHINGTON UNIVERSITY

WASHINGTON, DC

DATS 6313_10 Time Series Analysis & Modeling

Instructor: Dr. Reza Jafari

Time Series Final Project

Chiemeziem Oguayo

C.O May 2, 2022

Table of Contents

Abstract

This project explores the use of time series analysis and modeling to identify a model that could predict an individual's household electric power consumption. First the time series data (electric power consumption), was interrogated to determine its various characteristics such, as trend, seasonal effects, stationarity, and descriptive statistics. Second, different time series modeling techniques were surveyed to model and forecast electric power consumption. These models were assessed using a battery of time series techniques. Ultimately, the best model found was to be "".

Introduction

The goal of the project was to apply the learning objectives taught in this course to a real dataset for modeling and prediction. The dataset that was obtained was from UCI machine learning repository. It is the measurement of electric power consumption of a home located in Sceaux, France. It is a multivariate time series data with 2075259 observations, that in turn could be used to model and forecast future household energy consumption. Here the target variable is the active energy, which is the energy consumed by the household.

Descriptive Statistics and Description of Dataset

As mentioned prior, the dataset is a multivariate time series data with 2075259 observations. It is a large dataset, with a huge computational load. So, the data was resampled over hours to reduce the load.

1. (global_active_power*1000/60 - sub_metering_1 - sub_metering_2 - sub_metering_3) represents the active energy consumed every minute (in watt hour) in the household by electrical equipment not measured in sub-meterings 1, 2 and 3.
2. The dataset contains some missing values in the measurements (nearly 1,25% of the rows). All calendar timestamps are present in the dataset but for some timestamps, the measurement values are missing: a missing value is represented by the absence of value between two consecutive semi-colon attribute separators. For instance, the dataset shows missing values on April 28, 2007.

<u>Attribute Information</u>

1.date: Date in format dd/mm/yyyy
2.time: time in format hh:mm:ss
3.global_active_power: household global minute-averaged active power (in kilowatt)
4.global_reactive_power: household global minute-averaged reactive power (in kilowatt)
5.voltage: minute-averaged voltage (in volt)
6.global_intensity: household global minute-averaged current intensity (in ampere)
7.sub_metering_1: energy sub-metering No. 1 (in watt-hour of active energy). It corresponds to the kitchen, containing mainly a dishwasher, an oven and a microwave (hot plates are not

electric but gas powered).
8.sub_metering_2: energy sub-metering No. 2 (in watt-hour of active energy). It corresponds to the laundry room, containing a washing-machine, a tumble-drier, a refrigerator and a light.
9.sub_metering_3: energy sub-metering No. 3 (in watt-hour of active energy). It corresponds to an electric water-heater and an air-conditioner.

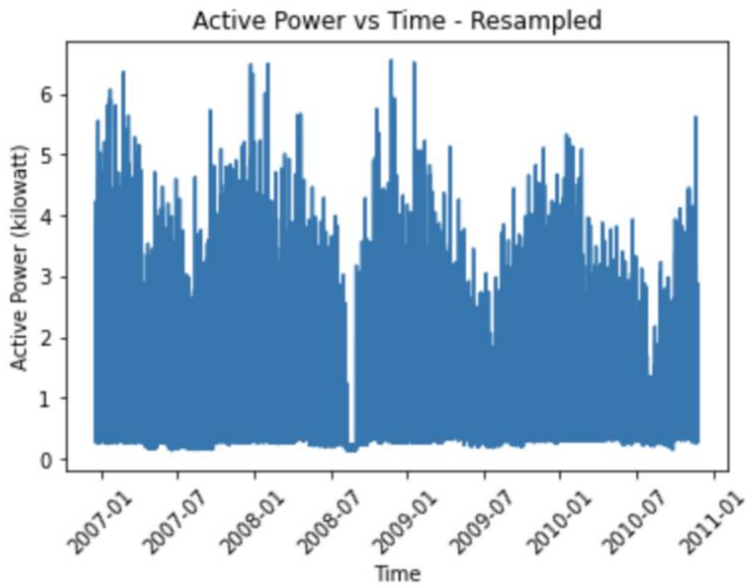

Figure 1. Plot of dependent variable vs time

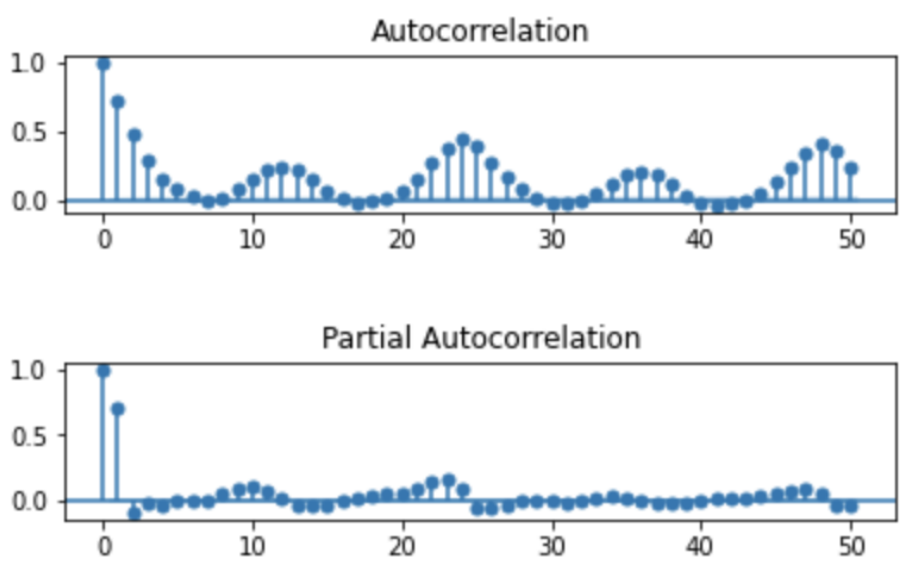

Figure 2. ACF/PACF Plot of variable

From the ACF/PACF plot It seems that there is seasonality as there is as there is a peak at every 12^{th} lag, possibly indicating seasonality. Also, there is a cutoff at the PACF at lag 1, which may indicate a purely AR process.

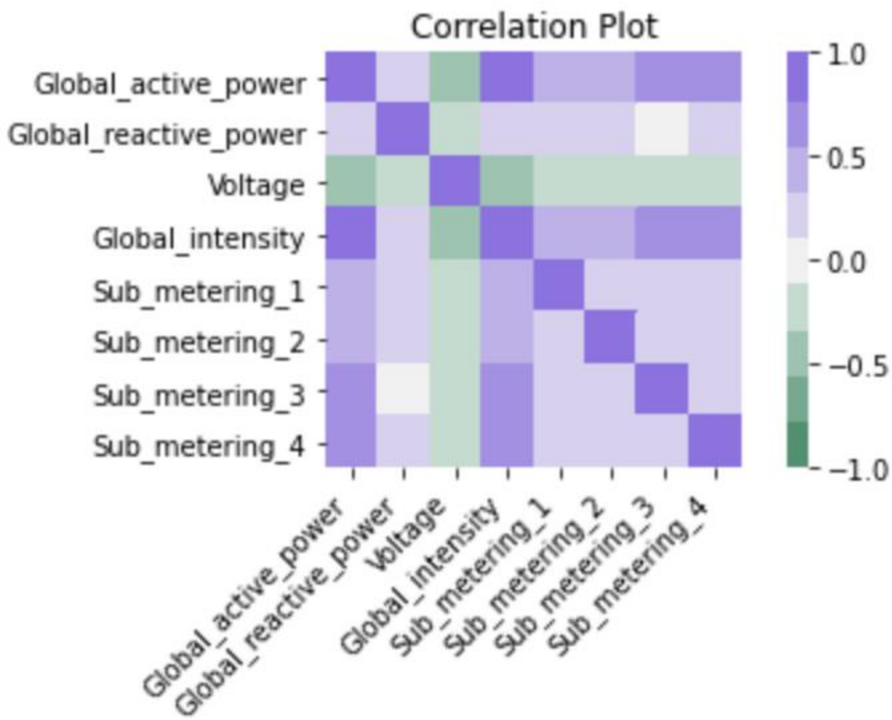

Figure 3. Correlation Plot of features

Global active power is strongly correlated with global intensity. And it seems that Global intensity shows similar correlations with the other variables as active power does with the other variables. In addition, all the variables are negatively correlated with voltage.

Stationarity

```
ADF Statistic: -14.263655
p-value: 0.000000
Critical Values:
        1%: -3.431
        5%: -2.862
        10%: -2.567
p-value is less than 0.05, reject null hypothesis thus time series data is Stationary
```

Figure 4. ADF Test for stationarity

```
(0.6459031030714386,
 0.01,
 52,
 {'10%': 0.119, '5%': 0.146, '2.5%': 0.176, '1%': 0.216})
```

Figure 5. KPSS Test for stationarity

From the ADF and KPSS test we can determine that the time series is indeed stationary. As for the ADF test we rejected the null and for the KPSS test we failed to reject the null.

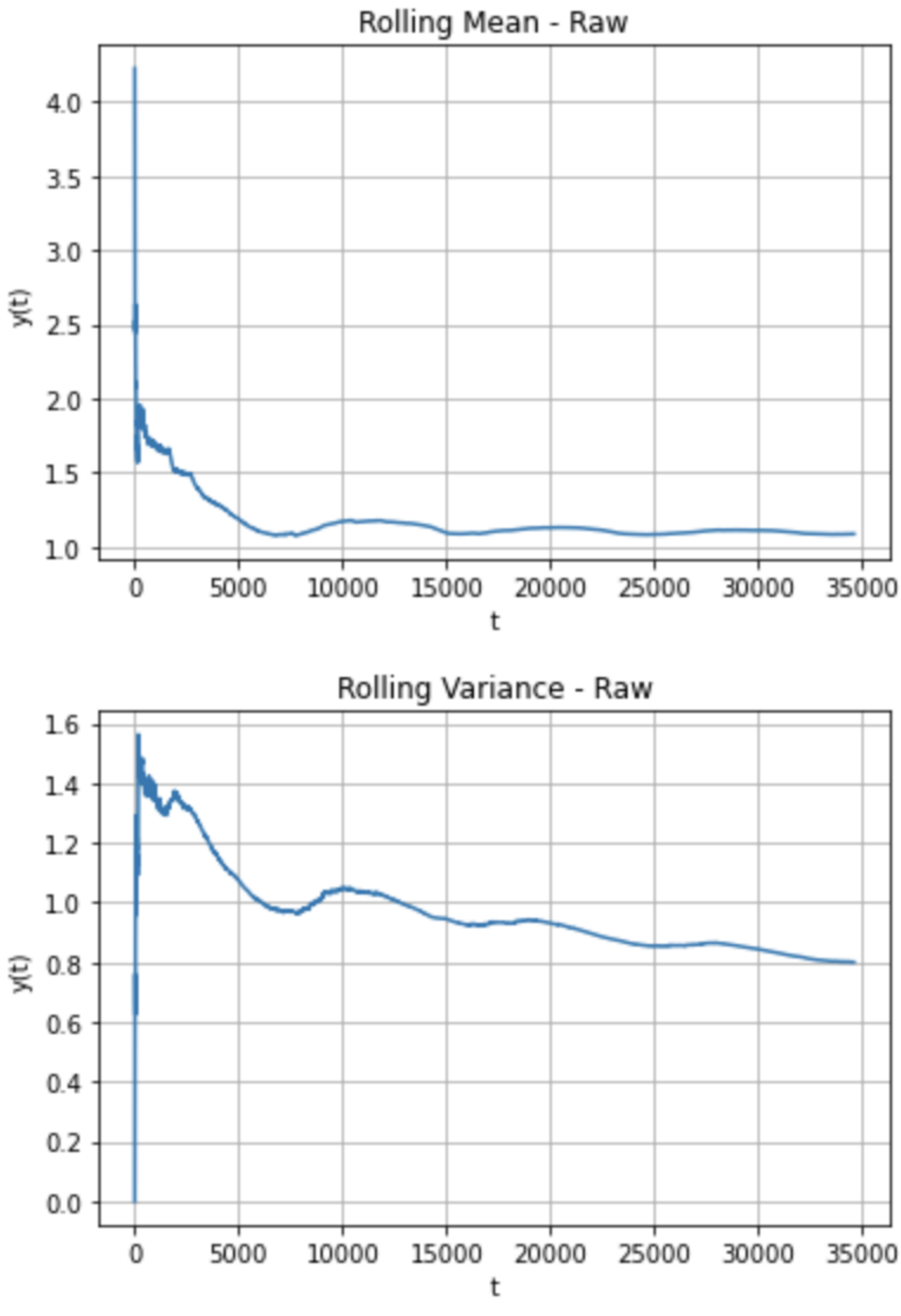

Figure 6. Plot of Rolling Mean and Variance

From the plot of the rolling mean and rolling variance we can see that both begin time stabilize as time progresses, further confirming that the time series is stationary.

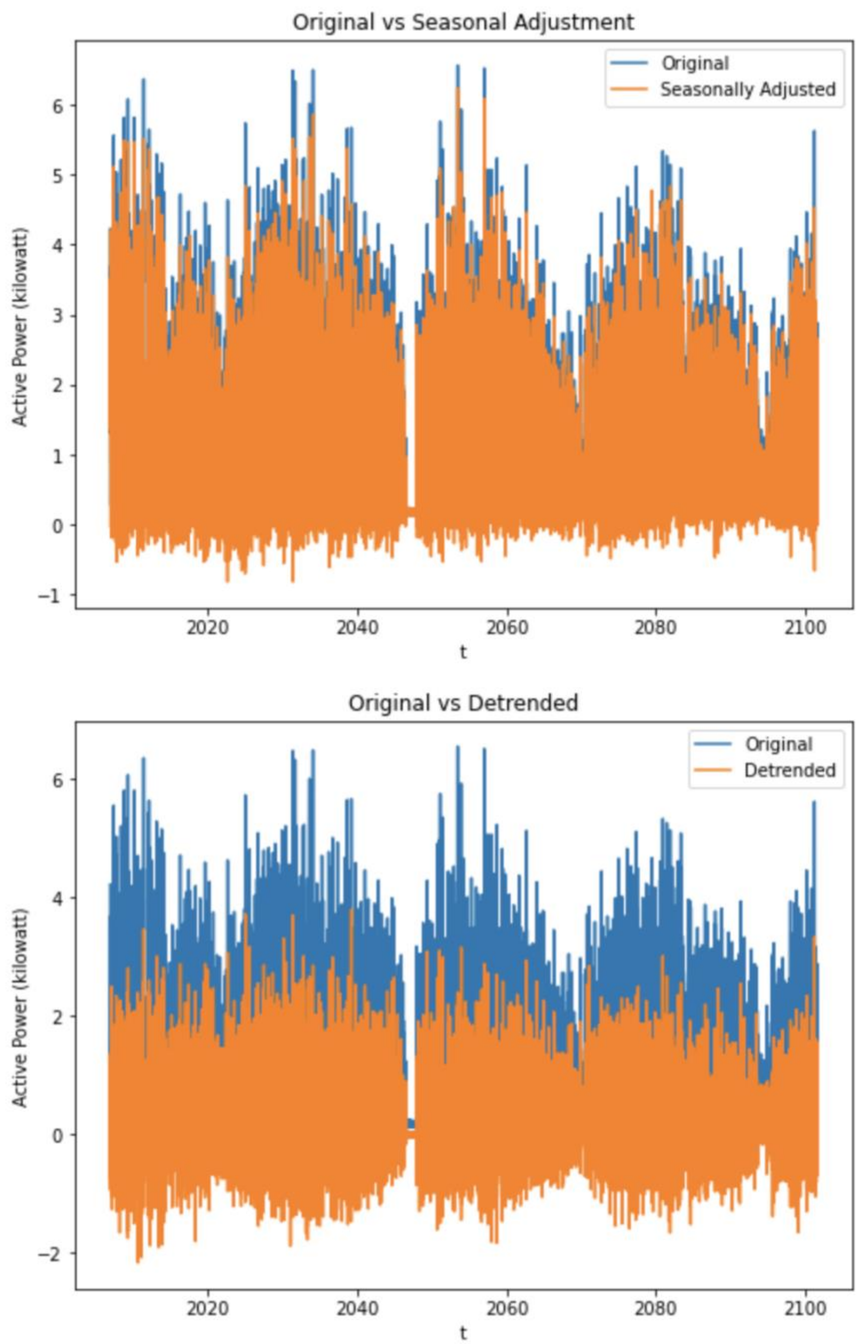

Figure 7. Plot of seasonally adjusted and detrended time series

The strength of trend for this dataset is 0.6678070388597215
The strength of seasonality for this dataset is 0.284315160227935

With the strength of the trend being 0.67 and strength of seasonality being 0.28, we can see that the trend in the data is detectable and it is more trended than it is seasonal.

Holt-Winters Forecast

The Holts-Winters forecasting method applies a triple exponential smoothing for trend, level, and seasonal components. Below are the results.

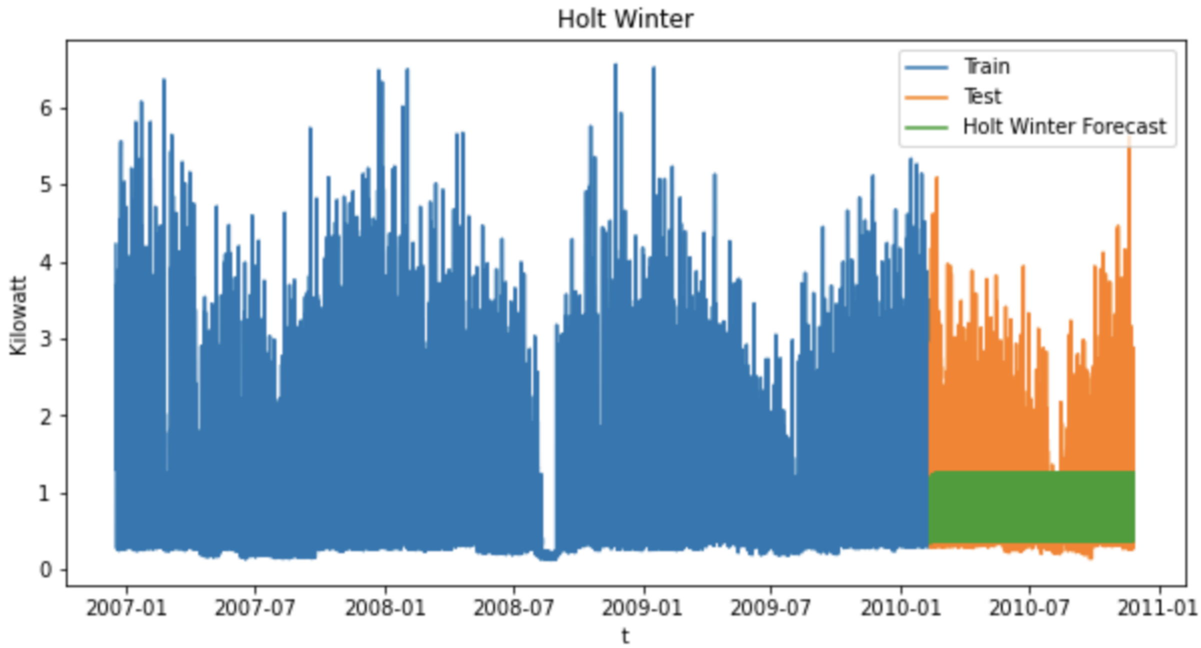

Figure 8. Holt-Winters Forecast

<u>Model Evaluation</u>

Plot of the time series data using Holt-Winters Forecast with seasonality of 12. Depicted is the train, test, and forecast of the test set.

The mean of the error of the HoltWinter Model is 0.2737996715713705
The variance of error of the HoltWinter Model is: 0.47283349535721647
The MSE of the HoltWinter Model is 0.5478
The RMSE of the HoltWinter Model is 0.7401351227985333

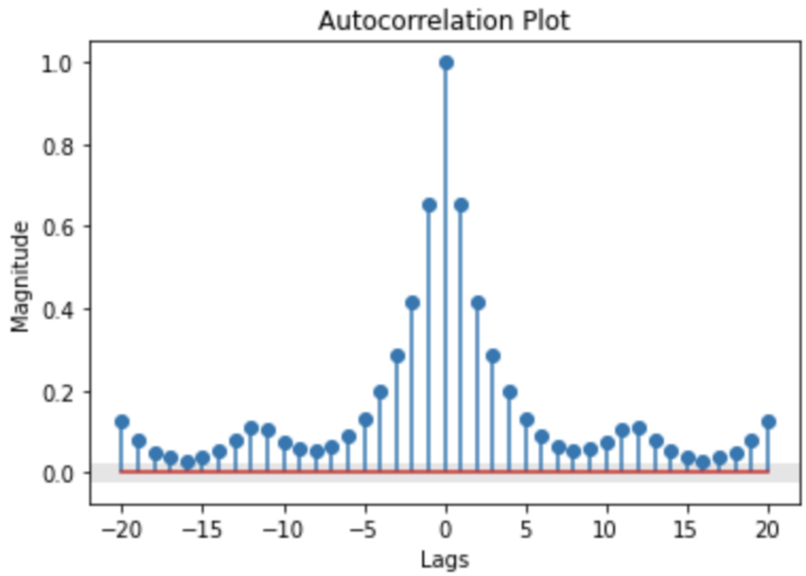

Figure 9. ACF of the residuals of the Holt-Winters Forecast

From the ACF plot we can see that the residuals are not white. Although, it decreases quickly, it seems that there are spikes at every 12^{th} lag.

	lb_stat	lb_pvalue	bp_stat	bp_pvalue
20	5726.243605	0.0	5722.190184	0.0

From the Ljung box statistic (using 20 lags) we can reject the null and determine that model shows the lack of a good fit.

Feature Selection and Backwards Regression

Feature selection is the process of reducing the number of input variables when developing a predictive model. And backwards regression is starting with all the variables and at every step removing variables that do not add to the predictive power of the model.

```
                            OLS Regression Results
==============================================================================
Dep. Variable:     Global_active_power   R-squared:                       1.000
Model:                             OLS   Adj. R-squared:                  1.000
Method:                  Least Squares   F-statistic:                 2.205e+30
Date:                 Mon, 02 May 2022   Prob (F-statistic):               0.00
Time:                         03:18:46   Log-Likelihood:             8.1482e+05
No. Observations:                27671   AIC:                        -1.630e+06
Df Residuals:                    27663   BIC:                        -1.630e+06
Df Model:                            7
Covariance Type:             nonrobust
=========================================================================================
                          coef    std err          t      P>|t|      [0.025      0.975]
-----------------------------------------------------------------------------------------
const               -2.753e-14   2.36e-14     -1.166      0.244   -7.38e-14    1.88e-14
Global_reactive_power -7.55e-15  4.47e-15     -1.691      0.091   -1.63e-14     1.2e-15
Voltage             -3.469e-17   9.75e-17     -0.356      0.722   -2.26e-16    1.56e-16
Global_intensity    -6.661e-16   2.77e-15     -0.240      0.810    -6.1e-15    4.77e-15
Sub_metering_1          0.0600   7.09e-16   8.46e+13      0.000       0.060       0.060
Sub_metering_2          0.0600   7.08e-16   8.48e+13      0.000       0.060       0.060
Sub_metering_3          0.0600   6.75e-16   8.89e+13      0.000       0.060       0.060
Sub_metering_4          0.0600   6.93e-16   8.66e+13      0.000       0.060       0.060
==============================================================================
Omnibus:                     3045.788   Durbin-Watson:                   0.002
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             4143.182
Skew:                           0.927   Prob(JB):                         0.00
Kurtosis:                       3.397   Cond. No.                     2.41e+04
==============================================================================
```

Figure 10. Regression results before selection

```
                                 OLS Regression Results
=======================================================================================
Dep. Variable:     Global_active_power   R-squared (uncentered):                   1.000
Model:                             OLS   Adj. R-squared (uncentered):              1.000
Method:                  Least Squares   F-statistic:                          3.271e+33
Date:                 Mon, 02 May 2022   Prob (F-statistic):                        0.00
Time:                         03:18:52   Log-Likelihood:                      9.0146e+05
No. Observations:                27671   AIC:                                 -1.803e+06
Df Residuals:                    27665   BIC:                                 -1.803e+06
Df Model:                            6
Covariance Type:             nonrobust
=========================================================================================
                          coef    std err          t      P>|t|      [0.025      0.975]
-----------------------------------------------------------------------------------------
Global_reactive_power  1.01e-14    1.7e-16     59.560      0.000    9.77e-15    1.04e-14
Voltage              -8.89e-18      1e-19    -88.899      0.000   -9.09e-18   -8.69e-18
Sub_metering_1          0.0600   3.15e-18   1.91e+16      0.000       0.060       0.060
Sub_metering_2          0.0600   2.49e-18   2.41e+16      0.000       0.060       0.060
Sub_metering_3          0.0600   1.47e-18   4.09e+16      0.000       0.060       0.060
Sub_metering_4          0.0600   1.26e-18   4.76e+16      0.000       0.060       0.060
==============================================================================
Omnibus:                     1248.722   Durbin-Watson:                   0.321
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             4380.678
Skew:                           0.031   Prob(JB):                         0.00
Kurtosis:                       4.948   Cond. No.                     3.95e+03
==============================================================================
```

Figure 11. Regression Results after backwards selection

Backwards stepwise regression was the chosen method of reducing the number of features. At each step a variable was removed if the p-value was not significant or add to the model. In the end, the remaining variables were reactive power, voltage, Sub metering 1-4. All their p values were well under 0.05, so they added to the model. However, the R^2 did not differ between the full and reduced model. It remained at 1, however a model such as that may be meaningless. The AIC and BIC were very low as well, suggesting this model is good a predicting energy consumption.

The condition number for X is: 3954.9106407340573
Singular Values are: [1.60768004e+09 2.37954293e+06 1.29161049e+06 5.05377498e+05 3.21 781057e+05 1.02784178e+02]

The condition number 3954 is large, which may indicate that collinearity exist among the variables. No singular values were close to 0.

Multiple Linear Regression Model

The MLS was built from a continuation of the feature selection and backwards regression.

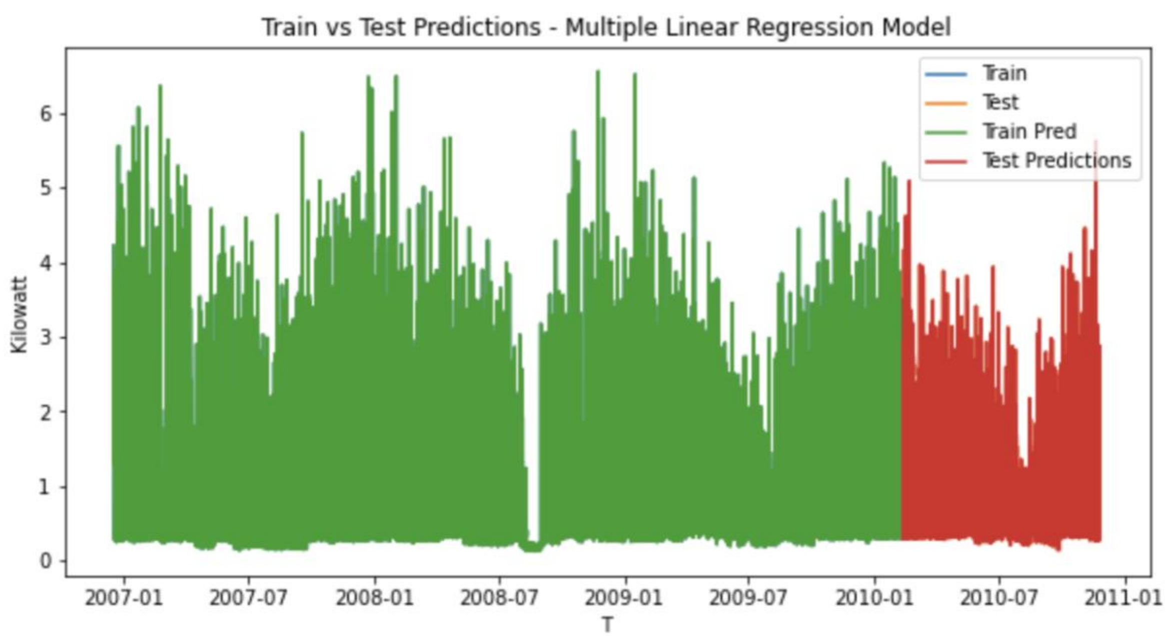

Figure 12. Multiple Linear Regression Model. The predictions/forecast overlaps very well with the train and test data

Model evaluation

The p-value of t-test is: Global_reactive_power 0.0
Voltage 0.0
Sub_metering_1 0.0
Sub_metering_2 0.0
Sub_metering_3 0.0
Sub_metering_4 0.0
The p-value of f-test is: 0.0

As this model uses the variables from the previous step, the variables in this model are all significant.

The AIC value of the model is: -1802906.2892009038
The BIC value of the model is: -1802856.9203596297
The R-squared value of the model is: 1.0
The Adjusted R-squared value of the model is: 1.0

The AIC and BIC are very small indicating that the model performs well at predicting energy Consumption. The R^2 and Adjusted R^2 are both 1 indicating a perfect fit, and has immense accuracy.

The variance of Prediction error is: 7.681051435276483e-31
The mean of Prediction error is: 1.479475695740853e-15
The variance of Forecast error is: 7.272602573587606e-31
The mean of Forecast error is: 1.3813634634340346e-15
The MSE of the residual is : 0.0
The Q value is: 4607.635496166898

We can see that the error for this is very small, with the MSE of this model being 0. This is a good model

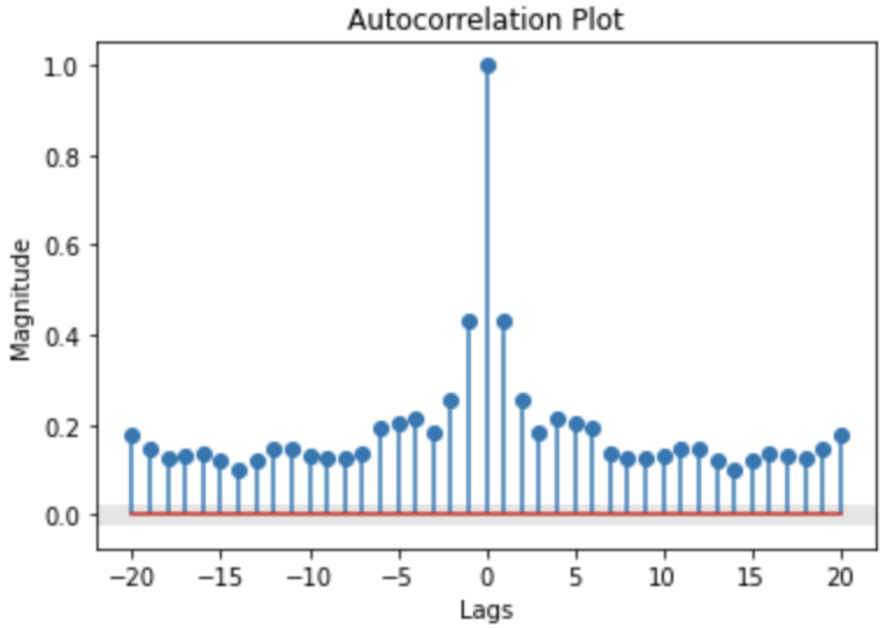

Figure 13. ACF of MLR Forecast

We can say the error of the multiple linear regression model are white, as the ACF decays quickly.

Average Method

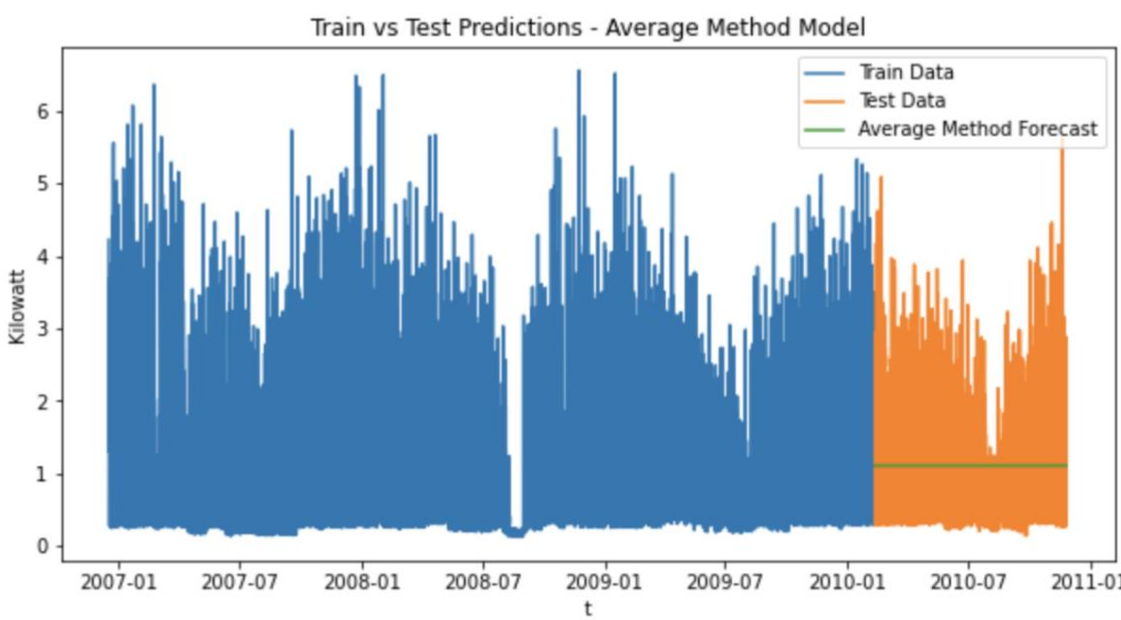

Figure 14. Average Method Forecast

<u>Model evaluation</u>

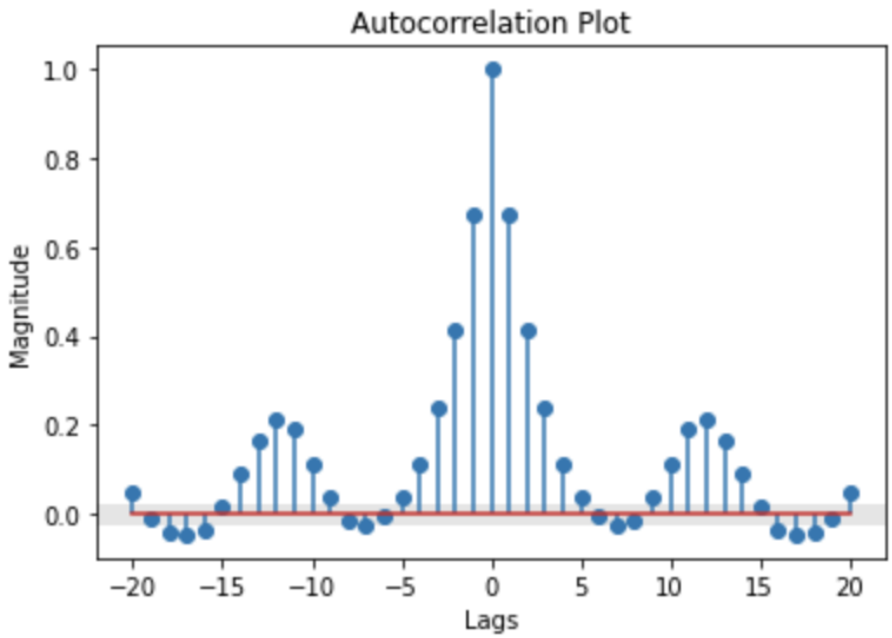

Figure 15. ACF of residuals of the residuals of Average Method Forecast

From the ACF we can see that the residuals are not white.

The MSE of the residual for the Average Method is: 0.55
The RMSE of the residual for the Average Method is: 0.7416198487095663
The variance of Forecast error is: 0.5340719263053189
The mean of Forecast error is: -0.12052942911981225
The Q value is: 5787.213651928438

This model is not good at capturing the variance in the data. Given the Q value the model also does not show a good fit.

Naïve Method

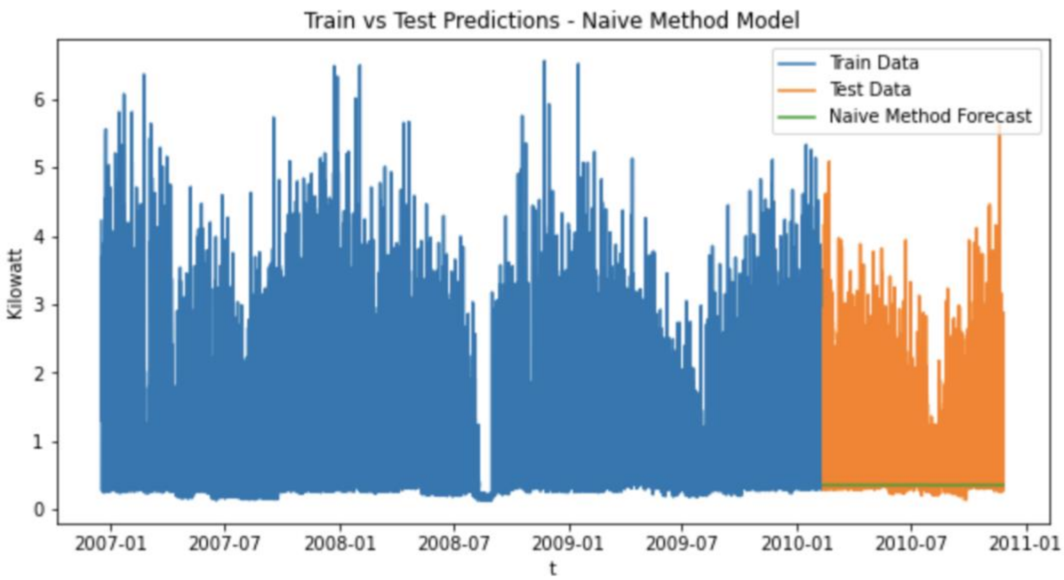

Figure 16. Naïve Method Forecast

<u>Model evaluation</u>

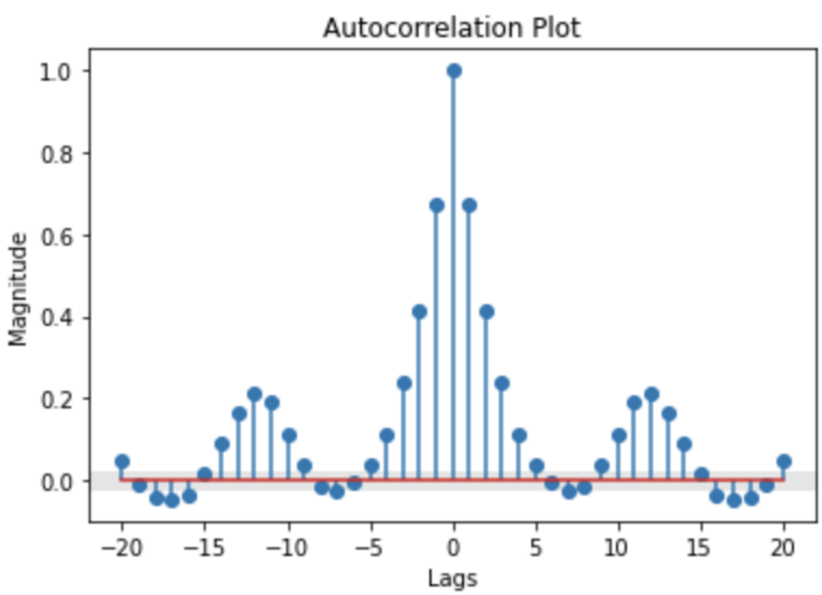

Figure 17. ACF of the residuals of Naïve Method Forecast

The MSE of the residual for the Average Method is: 0.93

The RMSE of the residual for the Average Method is: 0.9643650760992956
The variance of Forecast error is: 0.5340719263053189
The mean of Forecast error is: 0.6326776187722848
The Q value is: 5787.213651928438

We can see the error is higher in this model when compared to the previous models.

Drift Method

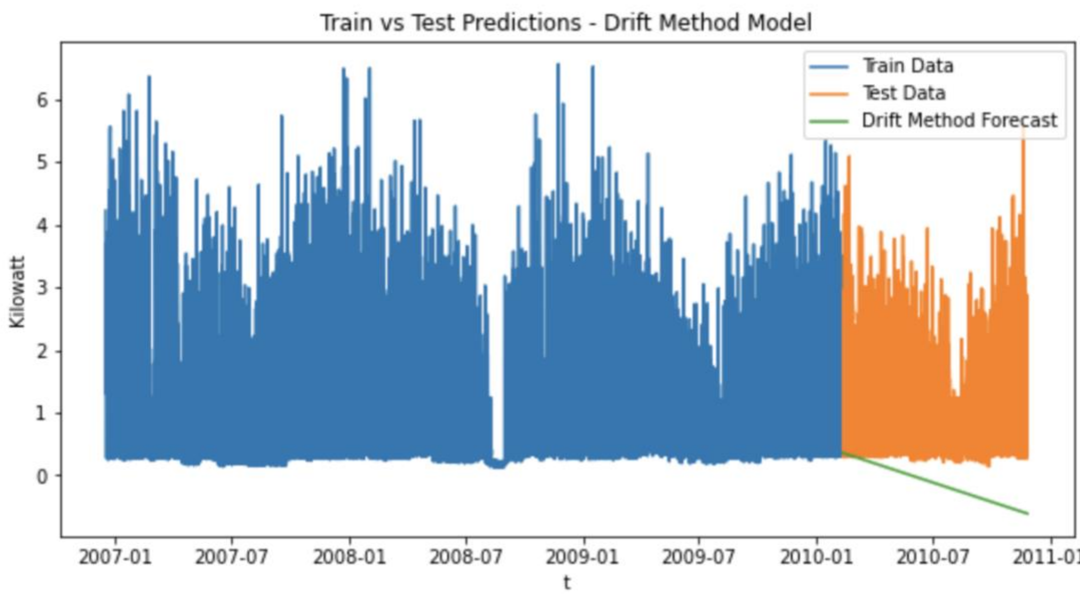

Figure 18. Drift Method Forecast

Model Evaluation

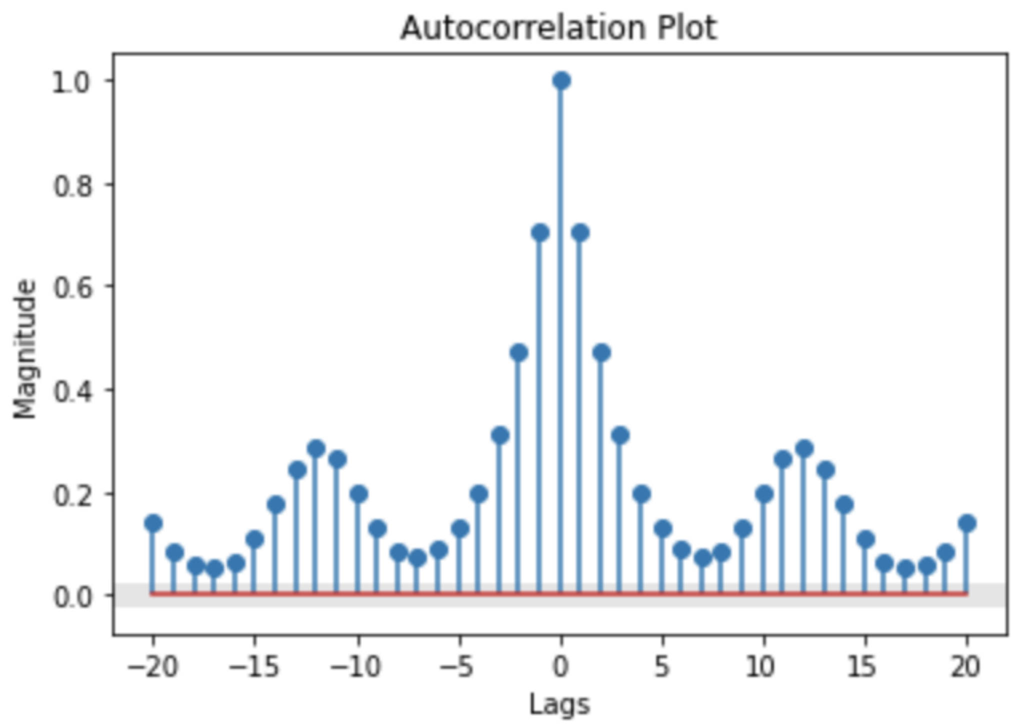

Figure 19. ACF of the residuals of the Drift Method Forecast

The MSE of the residual for the Drift Method is: 1.84
The RMSE of the residual for the Drift Method is: 1.3564659966250536
The variance of Forecast error is: 0.5910623550324018
The mean of Forecast error is: 1.1160060127408833
The Q value is: 8587.958908264609

This model is not good at capturing the variance in the data. Given the Q value the model also does not show a good fit. As with some of the previous models, there still appears to be a seasonal trend (except MLR).

Simple Exponential Smoothing

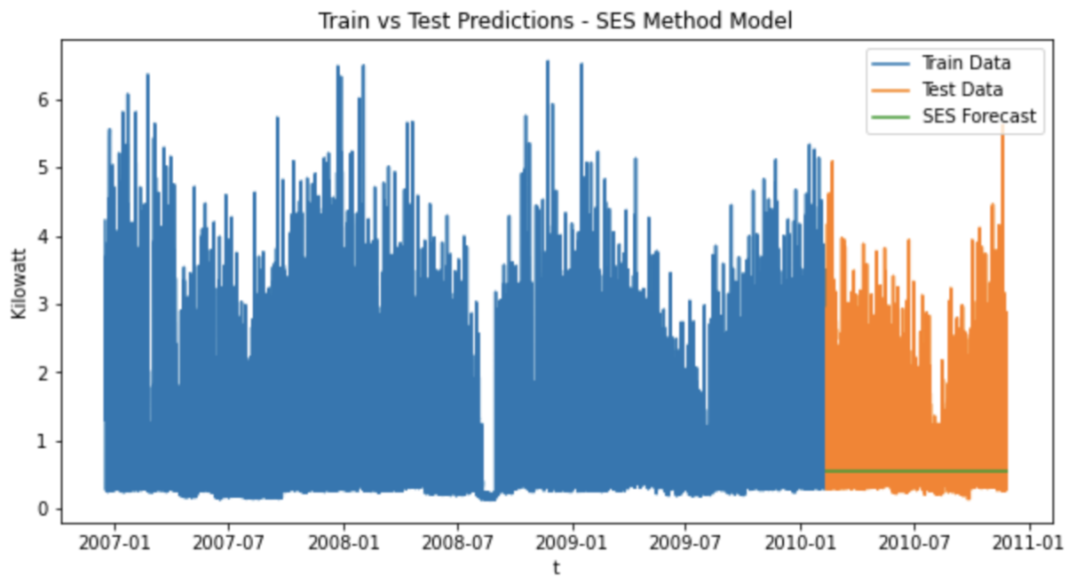

Figure 20. SES Method Forecast

<u>Model Evaluation</u>

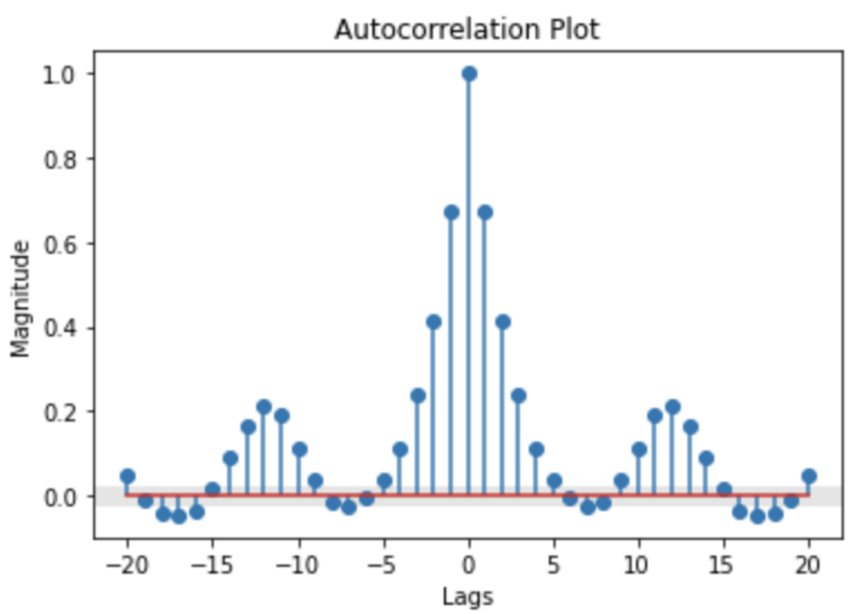

Figure 21. ACF of the residuals of SES Forecast

The MSE of the residual for the Average Method is: 0.73
The RMSE of the residual for the Average Method is: 0.8544003745317531
The variance of Forecast error is: 0.5340719263053189
The mean of Forecast error is: 0.4377268455074007
The Q value is: 5787.213651928447

This model is not good at capturing the variance in the data. Given the Q value the model also does not show a good fit. As with some of the previous models, there still appears to be a seasonal trend (except MLR).

Arima Model (1,0)

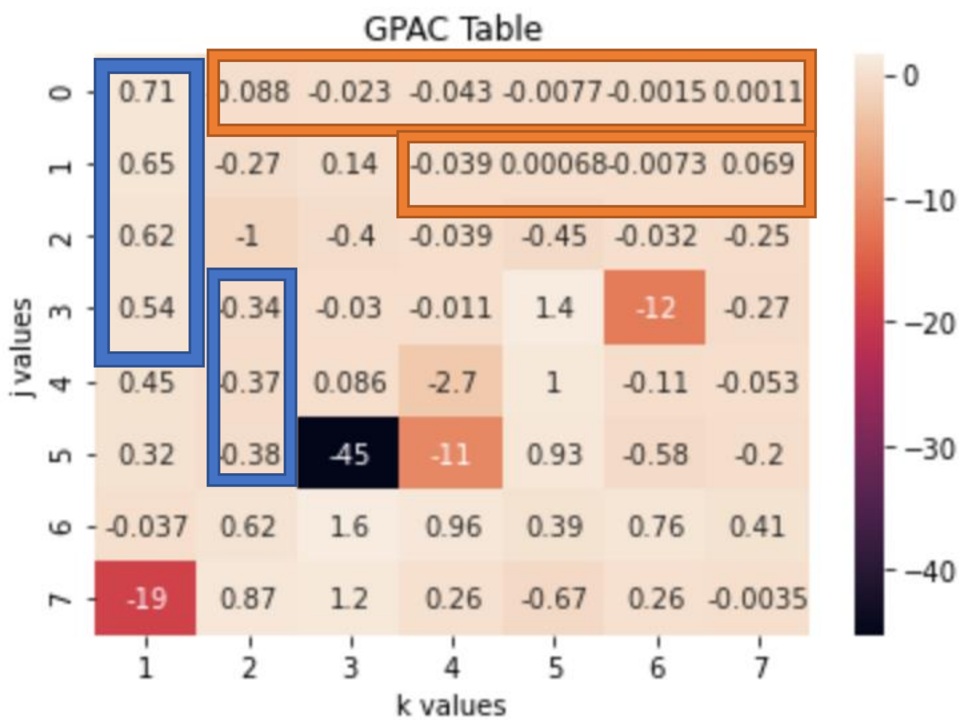

Figure 22. GPAC used to determine order of ARMA

Potential orders of (1,0), (2,1), (2,0)

```
                              ARMA Model Results
==============================================================================
Dep. Variable:     Global_active_power   No. Observations:                34589
Model:                      ARMA(1, 0)   Log Likelihood              -34488.616
Method:                        css-mle   S.D. of innovations              0.656
Date:                 Mon, 02 May 2022   AIC                          68981.233
Time:                         03:21:13   BIC                          68998.135
Sample:                     12-16-2006   HQIC                         68986.620
                          - 11-26-2010
==============================================================================================
                              coef    std err          z      P>|z|      [0.025      0.975]
ar.L1.Global_active_power   0.8849      0.003    353.162      0.000       0.880       0.890
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            1.1301           +0.0000j            1.1301            0.0000
-----------------------------------------------------------------------------
```

Figure 23. Arima Model (1,0) Summary

The parameters(coef) and confidence interval is highlighted in the box above.
The AR estimated coefficient a0 is: 0.8848861970081253
The confidence interval for estimated coefficient a0 is:
ar.L1.Global_active_power 0.879975 0.88979
The confidence interval for the estimated parameter does not include 0 so they are significant.

No zero/pole cancellations

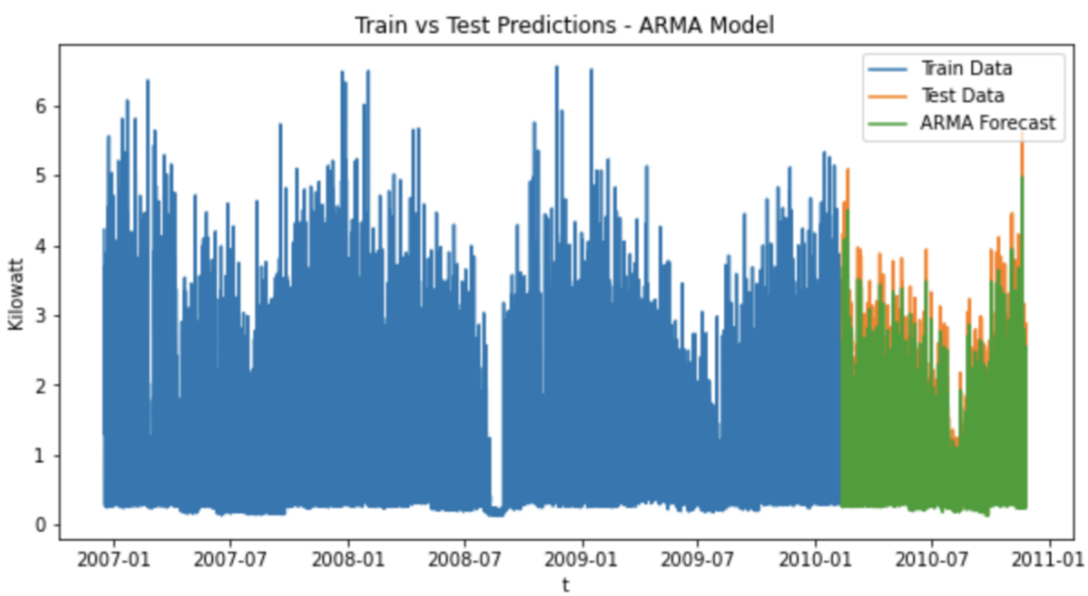

Figure 23. ARMA Model (1,0)

It seems that a good amount of the test data is captured by the model

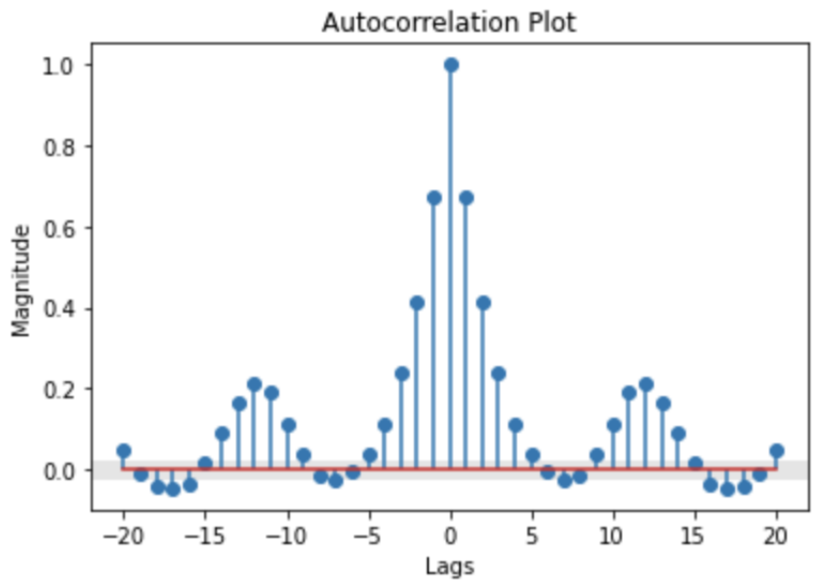

Figure 24. ACF of residuals of ARMA (1,0)

Model evaluation

The standard deviation of the parameter estimates is: 0.1849904247716646
The MSE of the residual for the Average Method is: 0.02
The RMSE of the residual for the Average Method is: 0.1414213562373095

The variance of Prediction error is: 0.4397077869371758
The mean of Prediction error is: 0.12782308641163162
The variance of Forecast error is: 0.007077087308328619
The mean of Forecast error is: 0.11393706581311966
The variance of prediction vs forecast error is: 62.1311802130389

The Q value is: 5787.213651928447

The covariance of estimated parameters is:
ar.L1.Global_active_power   0.000006

This model has a low MSE compared to other models and seems to perform well. However, the ACF of the residuals are note white. The seasonal trends seems to be apparent in the lags. This is not the best model.

ARMA (2,1)

```
                              ARMA Model Results
==============================================================================
Dep. Variable:     Global_active_power   No. Observations:              34589
Model:                      ARMA(2, 1)   Log Likelihood            -32586.234
Method:                        css-mle   S.D. of innovations            0.621
Date:                 Tue, 03 May 2022   AIC                        65180.468
Time:                         03:48:47   BIC                        65214.273
Sample:                     12-16-2006   HQIC                       65191.242
                          - 11-26-2010
==============================================================================================
                              coef    std err          z      P>|z|      [0.025      0.975]
----------------------------------------------------------------------------------------------
ar.L1.Global_active_power     1.6807   1.77e-05   9.51e+04      0.000       1.681       1.681
ar.L2.Global_active_power    -0.6807   1.34e-05  -5.08e+04      0.000      -0.681      -0.681
ma.L1.Global_active_power    -0.9954      0.001  -1902.583      0.000      -0.996      -0.994
==============================================================================
                  Real          Imaginary           Modulus         Frequency
------------------------------------------------------------------------------
AR.1            1.0000           +0.0000j            1.0000            0.0000
AR.2            1.4691           +0.0000j            1.4691            0.0000
MA.1            1.0047           +0.0000j            1.0047            0.0000
------------------------------------------------------------------------------
```

Figure 25. ARMA (2,1) Model Results

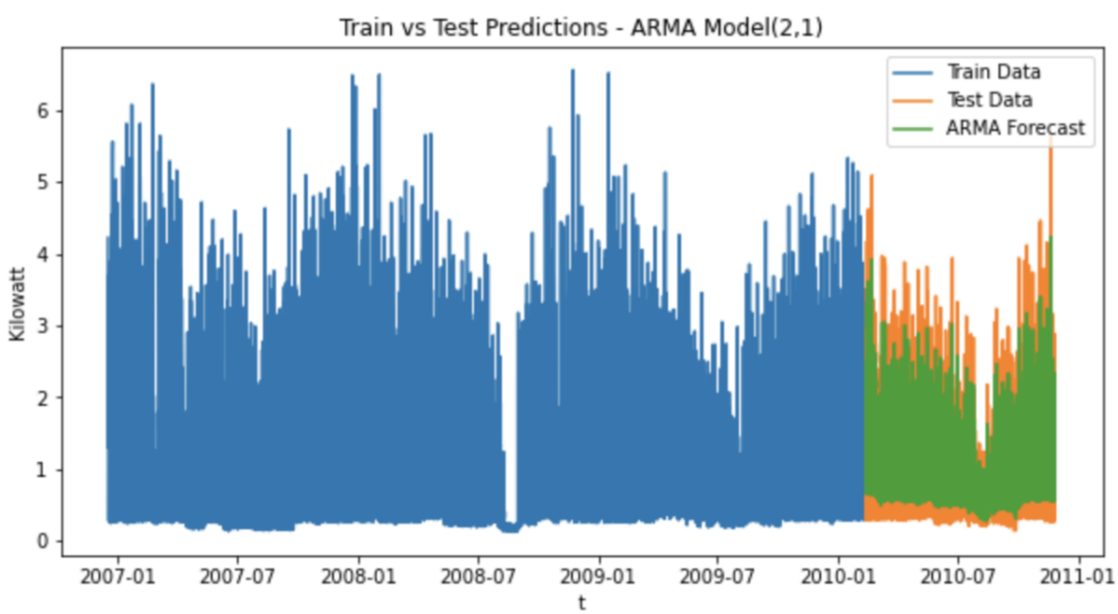

Figure 26. ARMA (2,1) Forecast

The AR coefficient a0 is: 1.680670545689072
The AR coefficient a1 is: -0.680677527114449
The MA coefficient a0 is: -0.9953711208984621

The confidence interval for estimated coefficient is:
ar.L1.Global_active_power 1.680636 1.680705
ar.L2.Global_active_power -0.680704 -0.680651
ma.L1.Global_active_power -0.996397 -0.994346

the covariance Matrix for the data is

```
                          ar.L1.Global_active_power  \
ar.L1.Global_active_power              3.124721e-10
ar.L2.Global_active_power             -1.810637e-10
ma.L1.Global_active_power             -1.752500e-09

                          ar.L2.Global_active_power  \
ar.L1.Global_active_power             -1.810637e-10
ar.L2.Global_active_power              1.793209e-10
ma.L1.Global_active_power              2.802235e-11

                          ma.L1.Global_active_power
ar.L1.Global_active_power             -1.752500e-09
ar.L2.Global_active_power              2.802235e-11
ma.L1.Global_active_power              2.737050e-07
```

Model Evaluation

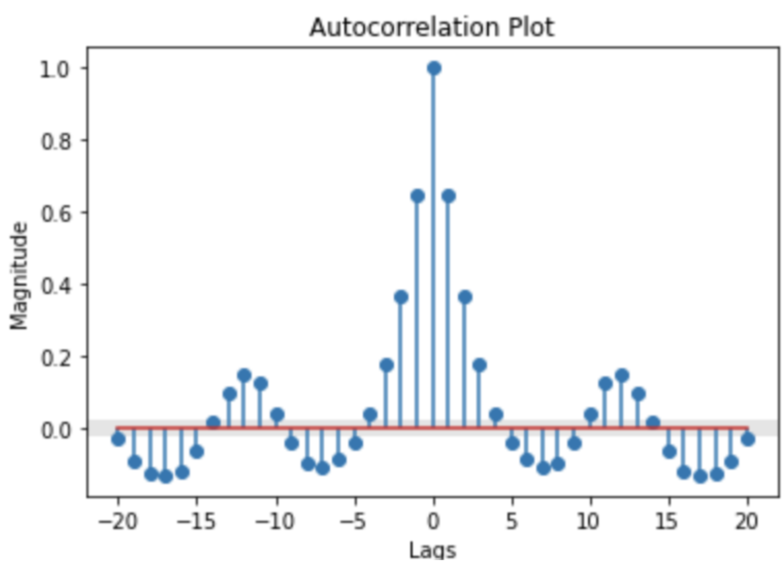

Figure 27. ACF of residuals of ARMA(2,1)

The MSE of the residual for the Average Method is: 0.05

The RMSE of the residual for the Average Method is: 0.22360679774997896

The variance of Prediction error is: 0.4099483755224675
The mean of Prediction error is: 0.0006077393701143024
The variance of Forecast error is: 0.04926784273669454
The mean of Forecast error is: -0.0018049658070193994
The variance of prediction vs forecast error is: 8.320810345063865
The Q value is: 5032.4482524419
The standard deviation of the parameter estimates is: 0.14843194803134716

This model has a low MSE compared to other models and seems to perform well. However, the ACF of the residuals are note white. The seasonal trends seem to be apparent in the lags. This is not the best model. And the ARMA (1,0) performs better than with order (2,1)

LSTM

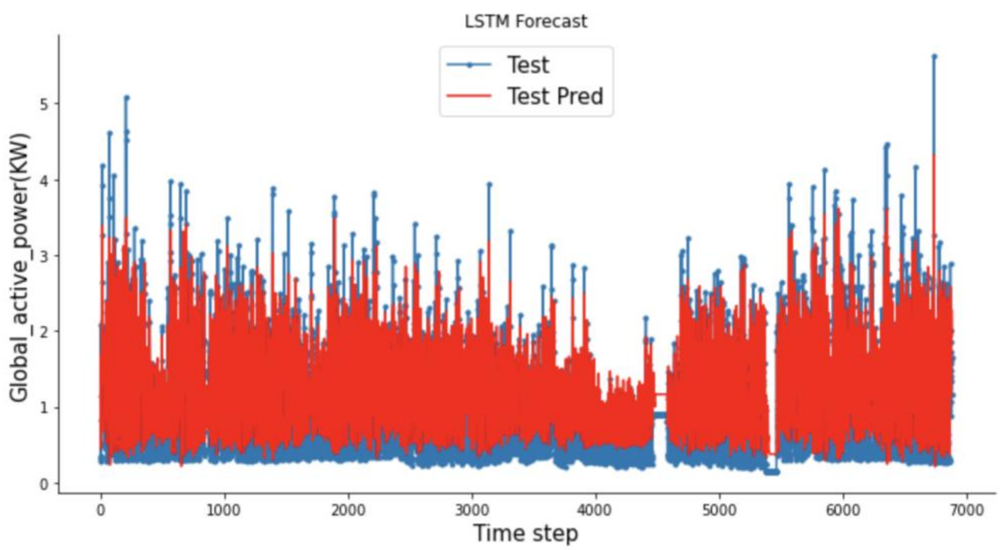

Figure 28. LSTM Forecast

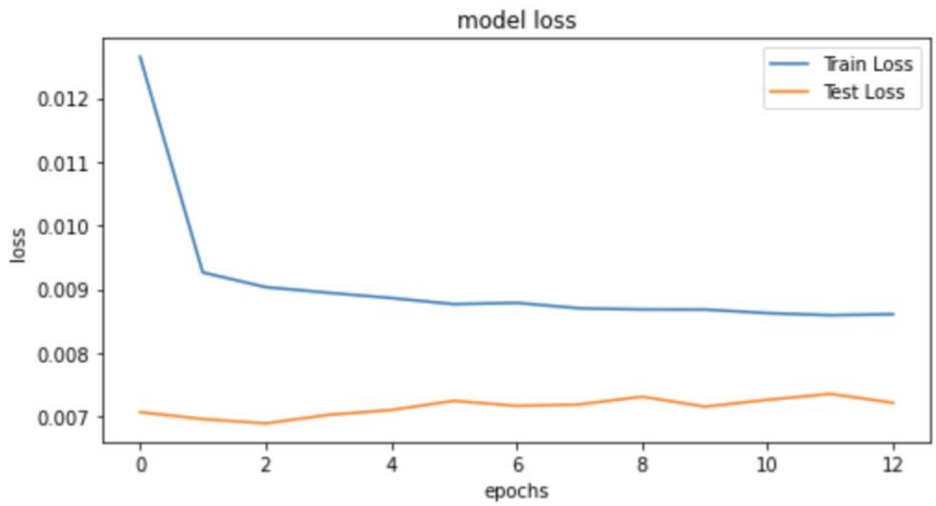

Figure 29. Model loss. The loss of the model is very low for the test set

Train Mean Absolute Error: 0.4644396976856607
Train Root Mean Squared Error: 0.6176469288002417
Test Mean Absolute Error: 0.4240039845240055
Test Root Mean Squared Error: 0.5469491639580324
The Q value is: 79.6109408324576

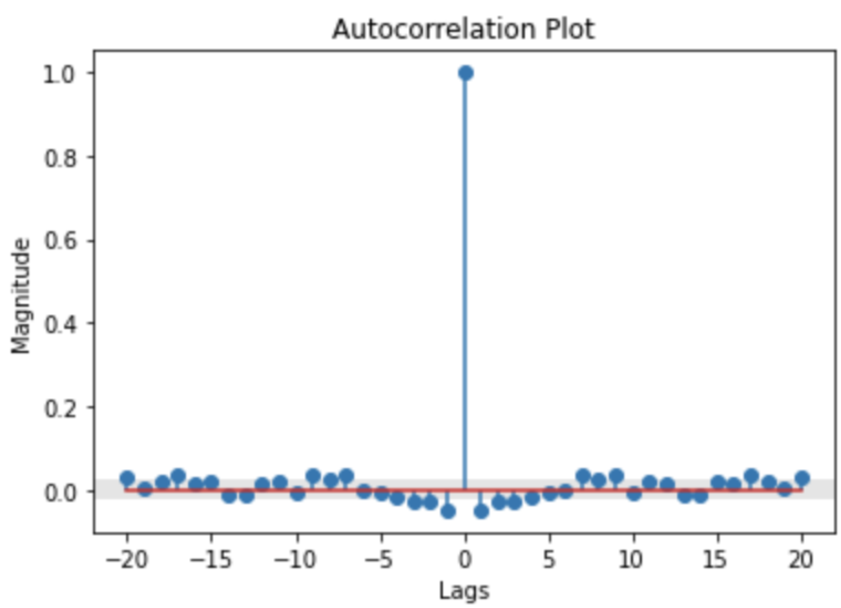

Figure 30. ACF of residuals of LSTM forecast

The ACF exhibits a spike at 0, with most of the other spikes in the insignificant region.

```
   lb_stat    lb_pvalue    bp_stat     bp_pvalue
20  79.73596  4.352548e-09  79.610941  4.570273e-09
```

When interpreting the Q value and the results of the Ljung box test, we can see that the model shows a good fit.

Conclusion

Model	MSE
Holt-Winter's Model	**0.74**
Multiple Linear Regression Model	**0.0**
Average	**0.55**
Naïve	**0.93**
Drift	**1.84**
SES	**0.73**
ARMA (0,1)	**0.02**
ARMA (2,1)	**0.05**
LSTM	**0.55**

The final model that will the chosen is the Multiple Linear Regression Model, as it had the lowest MSE and was the only model with a perfect fit. This model was better able to capture the data. Although it should be noted that the ARMA models performed just as well, with MSE's almost comparable to that of the Multiple Linear Regression model. The difference is MSE's could be said to be negligible. Also, interesting to note, is that the LSTM model, was the only model whose Q value was significant. The final forecast function can be written using the weights present in Multiple Linear Regression summary.

Y = (1.01e-14) * (global_reactive_power) – (8.98e-18) * (Voltage) + (0.06) * (Sum (Sub meterings 1-4))

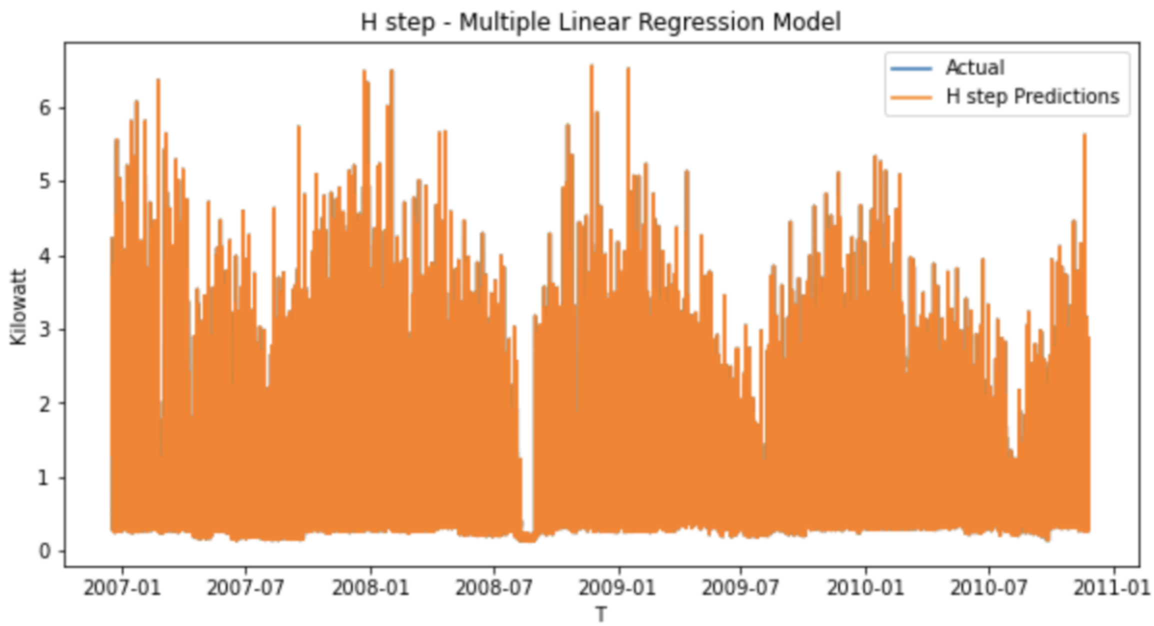

Figure 31. H step prediction of best model – Multiple Linear Regression Model

Reference:

Dataset source:

https://archive.ics.uci.edu/ml/datasets/Individual+household+electric+power+consumption#

Li, Susan. “Time Series Analysis, Visualization & Forecasting with LSTM.” *Medium*, Towards Data Science, 17 May 2019, https://towardsdatascience.com/time-series-analysis-visualization-forecasting-with-lstm-77a905180eba.

Appendix

***See pynb file attached if you want to run the code.**

```
#!/usr/bin/env python

# coding: utf-8


# In[164]:



import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

import seaborn as sns

import statsmodels.api as sm

import statsmodels.tsa.holtwinters as ets

from statsmodels.tsa.api import SimpleExpSmoothing

from scipy import signal

import math

import seaborn as sns

from statsmodels.graphics.tsaplots import plot_acf , plot_pacf

from statsmodels.tsa.stattools import adfuller
```

```
from statsmodels.tsa.stattools import acf

from sklearn.model_selection import train_test_split

from numpy import linalg as LA

from datetime import datetime



# # READING DATA AND CREATING SEVERAL TIME SERIES FUNCTIONS


# In[165]:



df = pd.read_csv('household_power_consumption.txt', sep=';',

        parse_dates={'dt': ['Date', 'Time']}, infer_datetime_format=True,

        low_memory=False, na_values=['nan', '?'], index_col='dt')



# In[166]:



# Creating a new columns for all the engery in the rest of the house that doesn't include
submeterings 1,2,&3.

# We are only given sub_metering1,sub_metering2,sub_metering3.
```

```
# (global_active_power*1000/60 - sub_metering_1 - sub_metering_2 - sub_metering_3)
represents the active energy consumed
# every minute (in watt hour) in the household by electrical equipment not measured in sub-
meterings 1, 2 and 3.
df['Sub_metering_4'] = (
    df['Global_active_power'] * 1000 / 60 - df['Sub_metering_1'] - df['Sub_metering_2'] -
df['Sub_metering_3'])



# In[167]:



def difference(dataset, interval):
  diff = []
  for i in range(interval, len(dataset)):
    value = dataset[i] - dataset[i - interval]
    diff.append(value)
  return diff
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))
    if result[1] < 0.05:
        print("p-value is less than 0.05, reject null hypothesis thus time series data is Stationary")
    else:
        print("p-value is greater than 0.05, we failed to reject null hypothesis thus time series data
is "
              "Non-Stationary")


from statsmodels.tsa.stattools import kpss



def kpss_test(timeseries):
    print('Results of KPSS Test:')
    kpsstest = kpss(timeseries, 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)
def rolling_mean(x):
```

```
    roll = []
    for i in range(1, len(x) + 1):
        rolling = np.mean(x[:i])
        roll.append(rolling)
    return roll



def rolling_var(x):
    roll = []
    for i in range(1, len(x) + 1):
        rolling = np.var(x[:i])
        roll.append(rolling)
    return roll
def acf_cal(x, k):
    acf_values = []
    mn = np.mean(x)
    for k in range(0, k + 1):
        acf_values.append(sum((x - mn).iloc[k:] * (x.shift(k) - mn).iloc[k:]) / sum((x - mn) ** 2))
    return acf_values
def GPAC(ry, j0, k0):
    # get phi
    def phi(ry, j, k):
```

```
    # FIRST STEP: GET phi
    # creating 0 for placeholders for denominator
    denominator = np.zeros(shape=(k, k))
    # replacing denom matrix with ry(j) values
    for a in range(k):
        for b in range(k):
            denominator[a][b] = ry[abs(j + a - b)]
    # making a copy of denom for numerator
    numerator = denominator.copy()
    # creating last column for numerator
    numL = np.array(ry[j + 1:j + k + 1])
    numerator[:, -1] = numL
    phi = np.linalg.det(numerator) / np.linalg.det(denominator)
    return phi


table0 = [[0 for i in range(1, k0)] for i in range(j0)]


for c in range(j0):
    for d in range(1, k0):
        table0[c][d - 1] = phi(ry, c, d)


pac = pd.DataFrame(np.array(table0), index=np.arange(j0), columns=np.arange(1, k0))
```

```
    return pac



def GPAC_plot(acf, j, k):
    gpac = GPAC(acf, j, k)
    plt.figure()
    sns.heatmap(gpac, annot=True)
    plt.title('GPAC Table')
    plt.xlabel('k values')
    plt.ylabel('j values')
    plt.show()
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()
```

```
def plotacf(s, k, n):

    # n = len(y)

    # k = number of lags

    t = pd.Series(s)

    acf = acf_cal(t, k)

    acf1 = acf[::-1][:-1]

    acf2 = acf1 + acf

    x = np.arange(-k, k + 1)

    insig = 1.96 / np.sqrt(len(np.arange(n)))

    plt.stem(x, acf2, markerfmt='o')

    plt.axhspan(-insig, insig, alpha=0.2, facecolor='0.5')

    plt.ylabel('Magnitude')

    plt.xlabel('Lags')

    plt.title('Autocorrelation Plot')

    plt.show()



# # CLEANING THE DATA


# In[168]:
```

```
# Checking the amount of nan values in the data

df.isnull().sum()


# In[169]:


# Using forward fill to handle nan values

df.ffill(axis='rows', inplace=True)


# In[170]:


df.isnull().sum()


# In[171]:


# Resample the data, because of computational time.

# Reduces the number of observations of 2075259 to 34589, structure is still retained
```

```
df_resample = df.resample('H').mean()

# In[172]:

# Plot of the dependent variable against time
y = df_resample['Global_active_power']
plt.figure()
plt.plot(y,)
plt.title('Active Power vs Time - Resampled')
plt.xlabel('Time')
plt.ylabel('Active Power (kilowatt)')
plt.xticks(rotation=45)
plt.show()

# # STATIONARITY CHECK

# In[173]:
```

```
# ACF/PACF Plot of the dependent variable

ACF_PACF_Plot(y, 50)



# In[174]:



# Correlation matrix
plt.figure()
corr = df_resample.corr()
ax = sns.heatmap(corr, vmin=-1, vmax=1, center=0, cmap=sns.diverging_palette(150, 275, s=80,
l=55, n=9), square=True)
ax.set_xticklabels(ax.get_xticklabels(), rotation=45, horizontalalignment='right')
plt.title('Correlation Plot')
plt.tight_layout()
plt.show()



# In[175]:
```

```
# Stationary Check on raw data

# the null for ADF is that the time series non-stationary

ADF_Cal(y)



# In[176]:



# the null for KPSS is that the time series is stationary

sm.tsa.stattools.kpss(y, regression='ct')



# In[177]:



# plot of rolling mean and rolling variance of raw data


Rmean = rolling_mean(y)

Rvar = rolling_var(y)


plt.figure()

plt.plot(Rmean)
```

```
plt.title('Rolling Mean - Raw')

plt.xlabel('t')

plt.ylabel('y(t)')

plt.grid()

plt.show()


plt.figure()

plt.plot(Rvar)

plt.title('Rolling Variance - Raw')

plt.xlabel('t')

plt.ylabel('y(t)')

plt.grid()

plt.show()



# Stationary so we can continue


# # SPLITTING THE DATA


# In[178]:
```

```
# Splitting the data into Train and Test

# Split the dataset into train set 80% and test set 20%

y_train, y_test = train_test_split(df_resample, shuffle=False, test_size=0.2)

#X =

df[['Global_reactive_power','Voltage','Global_intensity','Sub_metering_1','Sub_metering_2','Su

b_metering_3','Sub_metering_4']]

#x_train, y_train = train_test_split()




# # TIME SERIES DECOMPOSITION


# In[179]:




# Time Series Decomposition

ActivePower = df_resample['Global_active_power']

ActivePower = pd.Series(np.array(df_resample['Global_active_power']),

                 index=pd.date_range('2006-12-16 17:00:00', periods=len(ActivePower)),

                 name='Activee Power (kilowatt)')




# In[180]:
```

```
from statsmodels.tsa.seasonal import STL

STL = STL(ActivePower)

res = STL.fit()


T = res.trend

S = res.seasonal

R = res.resid


# Seasonally adjusted data and plot it vs the original

sadjusted = ActivePower - S

detrended = ActivePower - T


plt.figure(figsize=(8,6))

plt.plot(ActivePower, label= 'Original')

plt.plot(sadjusted, label='Seasonally Adjusted')

plt.title("Original vs Seasonal Adjustment")

plt.xlabel("t")

plt.ylabel("Active Power (kilowatt)")

plt.legend()

plt.show()
```

```
plt.figure(figsize=(8,6))

plt.plot(ActivePower, label= 'Original')

plt.plot(detrended, label='Detrended')

plt.title("Original vs Detrended")

plt.xlabel("t")

plt.ylabel("Active Power (kilowatt)")

plt.legend()

plt.show()



# In[181]:



Ft = np.max([0,1 - np.var(R)/np.var(T+R)])

Fs = np.max([0,1 - np.var(R)/np.var(S+R)])

print("The strength of trend for this dataset is ", Ft)

print("The strength of seasonality for this dataset is", Fs)



# # HOLTS WINTER FORECAST
```

```
# In[182]:



# Holts winter method

# use training data to fit model

model = ets.ExponentialSmoothing(y_train['Global_active_power'], damped_trend= True,

                         seasonal_periods=12, trend='mul', seasonal='mul').fit()

# prediction on train set

HW_train = model.forecast(steps=len(y_train['Global_active_power']))

HW_train = pd.DataFrame(HW_train,

columns=['Global_active_power']).set_index(y_train.index)


# prediction on test set

HW_test = model.forecast(steps=len(y_test['Global_active_power']))

HW_test = pd.DataFrame(HW_test, columns=['Global_active_power']).set_index(y_test.index)



# In[183]:



# forecast error
```

```
HW_FE = (y_test['Global_active_power'].values).flatten() -
HW_test['Global_active_power'].values.flatten()
# forecast MSE

HW_MSE = np.round(np.mean(np.square(np.subtract(HW_test['Global_active_power'].values,
y_test['Global_active_power'].values))), 4)



# In[184]:



# MODEL ASSESSMENT



# In[185]:



print("The mean of the error of the HoltWinter Model is", np.mean(HW_FE))
print("The variance of error of the HoltWinter Model is :", np.var(HW_FE))
print("The MSE of the HoltWinter Model is", HW_MSE)
print("The RMSE of the HoltWinter Model is", np.sqrt(HW_MSE))
```

```
# In[186]:



plt.figure(figsize=(10, 5))

plt.plot(y_train['Global_active_power'], label='Train')

plt.plot(y_test['Global_active_power'], label='Test')

plt.plot(HW_test, label='Holt Winter Forecast')

plt.title('Holt Winter')

plt.xlabel('t')

plt.ylabel('Kilowatt')

plt.legend()

plt.show()



# In[187]:



plotacf(HW_FE, 20, len(HW_FE))



# In[188]:
```

```
print(sm.stats.acorr_ljungbox(HW_FE, lags=[20], boxpierce=True, return_df=True))

# In[189]:

# Residuals are not white, model is not a good fit

# # FEATURE SELECTION

# In[303]:

X =
df_resample[['Global_reactive_power','Voltage','Global_intensity','Sub_metering_1','Sub_mete
ring_2','Sub_metering_3','Sub_metering_4']]
X = sm.add_constant(X)
Y = df_resample[['Global_active_power']]
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, shuffle = False, test_size = 0.2)
```

```
# OLS Model

model = sm.OLS(Y_train, X_train).fit()

print(model.summary())




# In[304]:




# Removing Global intensity P-value: 0.810

X.drop('Global_intensity', axis=1, inplace=True)

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, shuffle=False, test_size=0.2)

model = sm.OLS(Y_train, X_train).fit()

print(model.summary())




# In[305]:




# Removing const - P-value:0.486

X.drop('const', axis=1, inplace=True)

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, shuffle=False, test_size=0.2)

model = sm.OLS(Y_train, X_train).fit()
```

```
print(model.summary())



# # MULTIPLE LINEAR REGRESSION MODEL

# In[306]:



model = sm.OLS(Y_train, X_train).fit()
Tr_pred = model.predict(X_train)
T_pred = model.predict(X_test)
print(model.summary())



# In[307]:



X = X_train.values
print("The condition number for X is", LA.cond(X))
H = np.matmul(X.T,X)
s,d,v = np.linalg.svd(H)
print("SingularValues are", d)
```

```
# In[308]:

plt.figure(figsize=(10, 5))
plt.plot(Y_train, label='Train')  # Y_train Values
plt.plot(Y_test, label='Test')  # Y_test Values
plt.plot(Tr_pred, label='Train Pred')  # Train Predictions
plt.plot(T_pred, label='Test Predictions')  # Test Predictions using X_test
plt.legend(loc='best')
plt.xlabel('T')
plt.ylabel('Kilowatt')
plt.title('Train vs Test Predictions - Multiple Linear Regression Model')
plt.show()

# In[314]:

H_pred =
model.predict(df_resample.drop(columns=['Global_active_power','Global_intensity']))
```

```
# In[316]:

plt.figure(figsize=(10, 5))
plt.plot(df_resample['Global_active_power'], label='Actual')
plt.plot(H_pred, label='H step Predictions')
plt.legend(loc='best')
plt.xlabel('T')
plt.ylabel('Kilowatt')
plt.title('H step - Multiple Linear Regression Model')
plt.show()

# In[313]:

df_resample

# In[196]:
```

```
#t-test

print("The p-value of t-test is: ",model.pvalues)


#f-test

print("The p-value of f-test is: ",model.f_pvalue)


# In[197]:


print("The AIC value of the model is :",model.aic)

print("The BIC value of the model is :",model.bic)

print("The R-squared value of the model is: ",model.rsquared)

print("The Adjusted R-squared value of the model is: ",model.rsquared_adj)


# In[198]:


OLS_PE = Y_train.values.flatten() - Tr_pred.values
```

```
OLS_FE = Y_test.values.flatten() - T_pred.values

OLS_MSE = round(np.square(OLS_FE).mean(),2)

print("The MSE of the residual is :", OLS_MSE)



# In[199]:



print(sm.stats.acorr_ljungbox(OLS_FE, lags=[20], boxpierce=True, return_df=True))



# In[200]:



OLS_acf = acf_cal(pd.Series(OLS_FE),20)



# In[201]:



OLS_acf2 = acf_cal(pd.Series(OLS_PE), 20)
```

```
# In[202]:

plotacf(OLS_FE, 20, len(OLS_FE))

# In[203]:

plotacf(OLS_PE, 20, len(OLS_PE))

# In[204]:

# Q value
Q_ML_FE = len(OLS_FE)*sum(np.square(OLS_acf[1:]))
print("The Q value is:", Q_ML_FE)

# In[205]:
```

```
print("The variance of Prediction error is:", np.var(OLS_PE))

print("The mean of Prediction error is:", np.mean(OLS_PE))

print("The variance of Forecast error is:", np.var(OLS_FE))

print("The mean of Forecast error is:", np.mean(OLS_FE))



# ###### BASE MODELS #######


# # AVERAGE METHOD


# In[206]:




#y_train1 = y_train.values

#y_test1 = y_test.values

h_step = []


for i in range(len(y_test)):

    h_step.append(np.mean(y_train['Global_active_power'].values))
```

```
# In[207]:

AV_FE = y_test['Global_active_power'].values.flatten() - np.array(h_step).flatten()

AV_MSE = round(np.square(AV_FE).mean(),2)

print("The MSE of the residual for the Average Method is :", AV_MSE)

print("THe RMSE of the residual for the Average Method is:", np.sqrt(AV_MSE))

# In[208]:

print(sm.stats.acorr_ljungbox(AV_FE, lags=[20], boxpierce=True, return_df=True))

# In[209]:

AV_acf = acf_cal(pd.Series(AV_FE),20)
```

```
# In[210]:

plotacf(AV_FE, 20, len(AV_FE))

# In[211]:

# Q value
Q_AV_FE = len(AV_FE)*sum(np.square(AV_acf[1:]))
print("The Q value is:", Q_AV_FE)

# In[212]:

print("The variance of Forecast error is:", np.var(AV_FE))
print("The mean of Forecast error is:", np.mean(AV_FE))

# In[213]:
```

```
plt.figure(figsize=(10, 5))
plt.plot(Y_train,label= "Train Data")
plt.plot(Y_test,label= "Test Data")
plt.plot(y_test.index,h_step, label= "Average Method Forecast")
plt.legend(loc='best')
plt.title('Train vs Test Predictions - Average Method Model')
plt.xlabel("t")
plt.ylabel("Kilowatt")
plt.show()


# # NAIVE METHOD


# In[214]:


h_stepN = []
for i in range(len(y_test)):
    h_stepN.append(y_train['Global_active_power'][-1])
```

```
# In[215]:


N_FE = y_test['Global_active_power'].values.flatten() - np.array(h_stepN).flatten()
N_MSE = round(np.square(N_FE).mean(),2)
print("The MSE of the residual for the Average Method is :", N_MSE)
print("THe RMSE of the residual for the Average Method is:", np.sqrt(N_MSE))


# In[216]:


print(sm.stats.acorr_ljungbox(N_FE, lags=[20], boxpierce=True, return_df=True))


# In[217]:


N_acf = acf_cal(pd.Series(N_FE),20)
```

```
# In[218]:


plotacf(N_FE, 20, len(N_FE))


# In[219]:


# Q value
Q_N_FE = len(N_FE)*sum(np.square(N_acf[1:]))
print("The Q value is:", Q_N_FE)


# In[220]:


print("The variance of Forecast error is:", np.var(N_FE))
print("The mean of Forecast error is:", np.mean(N_FE))


# In[221]:
```

```
plt.figure(figsize=(10, 5))

plt.plot(Y_train,label= "Train Data")

plt.plot(Y_test,label= "Test Data")

plt.plot(y_test.index,h_stepN, label= "Naive Method Forecast")

plt.legend(loc='best')

plt.title('Train vs Test Predictions - Naive Method Model')

plt.xlabel("t")

plt.ylabel("Kilowatt")

plt.show()



# # DRIFT METHOD


# In[222]:




h_stepDR = []

for i in range(1,len(y_test) + 1):

   slope = (y_train['Global_active_power'][-1] - y_train['Global_active_power'][0]) /
(len(y_train)-1)
```

```
    h_stepDR.append(y_train['Global_active_power'][-1]+ i*slope)


# In[223]:


DR_FE = y_test['Global_active_power'].values.flatten() - np.array(h_stepDR).flatten()
DR_MSE = round(np.square(DR_FE).mean(),2)
print("The MSE of the residual for the Drift Method is :", DR_MSE)
print("THe RMSE of the residual for the Drift Method is:", np.sqrt(DR_MSE))


# In[224]:


print(sm.stats.acorr_ljungbox(DR_FE, lags=[20], boxpierce=True, return_df=True))


# In[225]:


DR_acf = acf_cal(pd.Series(DR_FE),20)
```

```
# In[226]:


plotacf(DR_FE, 20, len(DR_FE))


# In[227]:



# Q value
Q_DR_FE = len(DR_FE)*sum(np.square(DR_acf[1:]))
print("The Q value is:", Q_DR_FE)


# In[228]:



print("The variance of Forecast error is:", np.var(DR_FE))
print("The mean of Forecast error is:", np.mean(DR_FE))
```

```
# In[229]:



plt.figure(figsize=(10, 5))

plt.plot(Y_train,label= "Train Data")

plt.plot(Y_test,label= "Test Data")

plt.plot(y_test.index,h_stepDR, label= "Drift Method Forecast")

plt.legend(loc='best')

plt.title('Train vs Test Predictions - Drift Method Model')

plt.xlabel("t")

plt.ylabel("Kilowatt")

plt.show()



# # SES METHOD

# In[230]:



fit = SimpleExpSmoothing(np.asarray(y_train['Global_active_power'])).fit(smoothing_level=0.5,
optimized=False)
```

```
h_stepSES = fit.forecast(len(y_test))


# In[231]:


SES_FE = y_test['Global_active_power'].values.flatten() - np.array(h_stepSES).flatten()
SES_MSE = round(np.square(SES_FE).mean(),2)
print("The MSE of the residual for the Average Method is :", SES_MSE)
print("THe RMSE of the residual for the Average Method is:", np.sqrt(SES_MSE))


# In[232]:


print(sm.stats.acorr_ljungbox(SES_FE, lags=[20], boxpierce=True, return_df=True))


# In[233]:


SES_acf = acf_cal(pd.Series(SES_FE),20)
```

```
# In[234]:

plotacf(SES_FE, 20, len(SES_FE))

# In[235]:

# Q value
Q_SES_FE = len(SES_FE)*sum(np.square(SES_acf[1:]))
print("The Q value is:", Q_SES_FE)

# In[236]:

print("The variance of Forecast error is:", np.var(SES_FE))
print("The mean of Forecast error is:", np.mean(SES_FE))
```

```
# In[237]:



plt.figure(figsize=(10, 5))

plt.plot(Y_train,label= "Train Data")

plt.plot(Y_test,label= "Test Data")

plt.plot(y_test.index,h_stepSES, label= "SES Forecast")

plt.legend(loc='best')

plt.title('Train vs Test Predictions - SES Method Model')

plt.xlabel("t")

plt.ylabel("Kilowatt")

plt.show()



# # ARMA MODEL


# In[238]:



y_acf = acf_cal(pd.Series(y.values),20)
```

```
# In[239]:

GPAC_plot(y_acf, 8, 8)

# In[240]:

# potentially (2,0) or (1,0) or (2,1)

# In[242]:

model = sm.tsa.ARMA(y,(1,0)).fit(trend='nc',disp=0)

# In[243]:
```

```
print(model.summary())


# In[244]:


for i in range(1):
    print("The AR estimated coefficient a{}".format(i), "is:", model.params[i])

for i in range(1):
    print("The confidence interval for estimated coefficient a{}".format(i), "is:", model.conf_int())


# In[245]:


print("The standard deviation of the parameter estimates is: ",model.summary().tables[1])


# In[255]:
```

```
print("The corvariance of estimated parameters is:",model.cov_params())

# In[246]:

print("The standard deviation of the parameter estimate is: ",np.square(model.sigma2))

# In[247]:

# Prediction with ARMA
onestep_ARMA = model.predict(start=0, end=len(y_train)-1)
hstep_ARMA = model.predict(start=len(y_train)-1, end=len(df_resample))

# In[248]:

ARMA_PE = y_train['Global_active_power'].values.flatten() - np.array(onestep_ARMA).flatten()
ARMA_FE = y_test['Global_active_power'].values.flatten() - np.array(hstep_ARMA[2:]).flatten()
```

```
ARMA_MSE = round(np.square(ARMA_FE).mean(),2)

print("The MSE of the residual for the Average Method is :", ARMA_MSE)

print("THe RMSE of the residual for the Average Method is:", np.sqrt(ARMA_MSE))



# In[249]:



print(sm.stats.acorr_ljungbox(ARMA_FE, lags=[20], boxpierce=True, return_df=True))



# In[250]:



ARMA_acf = acf_cal(pd.Series(ARMA_FE),20)



# In[251]:



plotacf(ARMA_FE, 20, len(ARMA_FE))
```

```
# In[252]:


# Q value
Q_ARMA_FE = len(ARMA_FE)*sum(np.square(ARMA_acf[1:]))
print("The Q value is:", Q_ARMA_FE)


# In[253]:


print("The variance of Prediction error is:", np.var(ARMA_PE))
print("The mean of Prediction error is:", np.mean(ARMA_PE))
print("The variance of Forecast error is:", np.var(ARMA_FE))
print("The mean of Forecast error is:", np.mean(ARMA_FE))
print("The variance of prediction vs forecast error is:",ARMA_PE.var()/ARMA_FE.var())


# In[254]:
```

```
plt.figure(figsize=(10, 5))

plt.plot(Y_train,label= "Train Data")

plt.plot(Y_test,label= "Test Data")

plt.plot(y_test.index,hstep_ARMA[2:], label= "ARMA Forecast")

plt.legend(loc='best')

plt.title('Train vs Test Predictions - ARMA Model')

plt.xlabel("t")

plt.ylabel("Kilowatt")

plt.show()



# In[256]:



model2 = sm.tsa.ARMA(y,(2,1)).fit(trend='nc',disp=0)



# In[257]:



print(model2.summary())
```

```
# In[147]:


for i in range(2):
    print("The AR coefficient a{}".format(i), "is:", model2.params[i])
for i in range(1):
    print("The MA coefficient a{}".format(i), "is:", model2.params[i + 2])


# In[258]:


for i in range(1):
    print("The confidence interval for estimated coefficient is:", model2.conf_int())


# In[150]:


print("The standard deviation of the parameter estimates is: ",np.square(model2.sigma2))
```

```
# In[260]:

print("the covariance Matrix for the data is", model2.cov_params())

# In[261]:

# Prediction with ARMA(2,1)
onestep_ARMA2 = model2.predict(start=0, end=len(y_train)-1)
hstep_ARMA2 = model2.predict(start=len(y_train)-1, end=len(df_resample))

# In[262]:

ARMA_PE2 = y_train['Global_active_power'].values.flatten() -
np.array(onestep_ARMA2).flatten()
ARMA_FE2 = y_test['Global_active_power'].values.flatten() -
np.array(hstep_ARMA2[2:]).flatten()
```

```
ARMA_MSE2 = round(np.square(ARMA_FE2).mean(),2)

print("The MSE of the residual for the Average Method is :", ARMA_MSE2)

print("THe RMSE of the residual for the Average Method is:", np.sqrt(ARMA_MSE2))



# In[153]:



print(sm.stats.acorr_ljungbox(ARMA_FE2, lags=[20], boxpierce=True, return_df=True))



# In[154]:



ARMA_acf2 = acf_cal(pd.Series(ARMA_FE2),20)



# In[155]:



plotacf(ARMA_FE2, 20, len(ARMA_FE2))
```

```
# In[263]:


# Q value
Q_ARMA_FE2 = len(ARMA_FE2)*sum(np.square(ARMA_acf2[1:]))
print("The Q value is:", Q_ARMA_FE2)


# In[264]:


print("The variance of Prediction error is:", np.var(ARMA_PE2))
print("The mean of Prediction error is:", np.mean(ARMA_PE2))
print("The variance of Forecast error is:", np.var(ARMA_FE2))
print("The mean of Forecast error is:", np.mean(ARMA_FE2))
print("The variance of prediction vs forecast error is:",ARMA_PE2.var()/ARMA_FE2.var())


# In[266]:
```

```
plt.figure(figsize=(10, 5))

plt.plot(Y_train,label= "Train Data")

plt.plot(Y_test,label= "Test Data")

plt.plot(y_test.index,hstep_ARMA2[2:], label= "ARMA Forecast")

plt.legend(loc='best')

plt.title('Train vs Test Predictions - ARMA Model(2,1)')

plt.xlabel("t")

plt.ylabel("Kilowatt")

plt.show()



# # LSTM


# In[ ]:




# code for LSTM does not belong to me, it was adopted from Susan Li fro towardsdatascience

# https://towardsdatascience.com/time-series-analysis-visualization-forecasting-with-lstm-

77a905180eba




# In[112]:
```

```
get_ipython().system('pip install tensorflow')


# In[121]:


import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
from keras.layers import *
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from keras.callbacks import EarlyStopping


# In[275]:
```

```
data = df_resample.Global_active_power.values #numpy.ndarray
data = data.astype('float32')
data = np.reshape(data, (-1, 1))
scaler = MinMaxScaler(feature_range=(0, 1))
data = scaler.fit_transform(data)
train_size = int(len(data) * 0.80)
test_size = len(data) - train_size
train, test = data[0:train_size,:], data[train_size:len(data),:]


# In[276]:



# convert an array of values into a dataset matrix
def create_dataset(dataset, look_back=1):
   X, Y = [], []
   for i in range(len(dataset)-look_back-1):
       a = dataset[i:(i+look_back), 0]
       X.append(a)
       Y.append(dataset[i + look_back, 0])
   return np.array(X), np.array(Y)
```

```
# In[277]:


# reshape into X=t and Y=t+1
look_back = 30
X_train, Y_train = create_dataset(train, look_back)
X_test, Y_test = create_dataset(test, look_back)


# In[278]:


X_train.shape


# In[279]:


Y_train.shape
```

```
# In[280]:


# reshape input to be [samples, time steps, features]
X_train = np.reshape(X_train, (X_train.shape[0], 1, X_train.shape[1]))
X_test = np.reshape(X_test, (X_test.shape[0], 1, X_test.shape[1]))


# In[281]:


model = Sequential()
model.add(LSTM(100, input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(Dropout(0.2))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')

history = model.fit(X_train, Y_train, epochs=20, batch_size=70, validation_data=(X_test, Y_test),
                    callbacks=[EarlyStopping(monitor='val_loss', patience=10)], verbose=1,
shuffle=False)
```

```
# Training Phase
model.summary()



# In[282]:




# make predictions
train_predict = model.predict(X_train)
test_predict = model.predict(X_test)
# invert predictions
train_predict = scaler.inverse_transform(train_predict)
Y_train = scaler.inverse_transform([Y_train])
test_predict = scaler.inverse_transform(test_predict)
Y_test = scaler.inverse_transform([Y_test])

print('Train Mean Absolute Error:', mean_absolute_error(Y_train[0], train_predict[:,0]))
print('Train Root Mean Squared Error:',np.sqrt(mean_squared_error(Y_train[0],
train_predict[:,0])))
print('Test Mean Absolute Error:', mean_absolute_error(Y_test[0], test_predict[:,0]))
print('Test Root Mean Squared Error:',np.sqrt(mean_squared_error(Y_test[0],
test_predict[:,0])))
```

```
# In[283]:


plt.figure(figsize=(8,4))
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Test Loss')
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epochs')
plt.legend(loc='upper right')
plt.show();


# In[285]:


aa=[x for x in range(200)]
plt.figure(figsize=(10,5))
plt.plot(Y_test[0], marker='.', label="Test")
plt.plot(test_predict[:,0], 'r', label="Test Pred")
```

```
#plt.plot(Y_train[0], marker='.', label="Train")

#plt.plot(train_predict[:,0],'g',label="Train Pred")

# plt.tick_params(left=False, labelleft=True) #remove ticks

plt.tight_layout()

sns.despine(top=True)

plt.subplots_adjust(left=0.07)

plt.ylabel('Global_active_power(KW)', size=15)

plt.xlabel('Time step', size=15)

plt.legend(fontsize=15)

plt.title('LSTM Forecast')

plt.show();



# In[273]:



len(y_train['Global_active_power'].values.flatten())



# In[294]:
```

```
LSTM_PE = np.subtract(Y_train[0], train_predict[:,0])

LSTM_PMSE = np.sqrt(mean_squared_error(Y_train[0], train_predict[:,0]))

LSTM_FE = np.subtract(Y_test[0], test_predict[:,0])

LSTM_MSE = np.sqrt(mean_squared_error(Y_test[0], test_predict[:,0]))



# In[297]:



LSTM_acf = acf_cal(pd.Series(LSTM_FE),20)



# In[298]:



# Q value

Q_LSTM_FE = len(LSTM_FE)*sum(np.square(LSTM_acf[1:]))

print("The Q value is:", Q_LSTM_FE)



# In[299]:
```

```
plotacf(LSTM_FE, 20, len(LSTM_FE))


# In[300]:


print(sm.stats.acorr_ljungbox(LSTM_FE, lags=[20], boxpierce=True, return_df=True))


# In[ ]:
```