Final Term Project on Wind Turbine Power Output Forecasting

The George Washington University
DATS 6313: Time Series Analysis and Model

Ambar Pathak
May 10, 2023

Table of Contents

Abstract 6
Introduction 7
The Dataset 9
Description of the dataset 9
Independent variables 9
Dependent Variable 11
Data Pre-Processing 12
Plot of the dependent variable versus time 13
ACF/PACF of the dependent variable 14
Correlation Matrix 14
Splitting the data 15
Stationarity 16
AD Fuller Test(Raw data) 16
KPSS Test(Raw data) 16
Plot of Rolling mean and variance(Raw data) 17
ACF/PACF Analysis for Stationarity(Raw data) 17
Time Series Decomposition 18
STL decomposition - after recent discovery 20
Rolling Mean and Variance 2.0(Transformed data) 20
ADF and KPSS test (Transformed data) 21
ACF/PACF 2.0 (Transformed data) 21
Holt Winters Method 24
Feature Selection/Elimination 25
Singular values from Singular value decomposition 25
Condition Number 26
Variance Inflation Factor(V.I.F.) 26
First Step(Initial model fitting with all features) 27
Second Step(model fitting with one less feature) 30
Base Models 34
Average Model/Method 34
Naive Model/Method 35
Drift Model/Method 35

Simple Exponential Smoothing Model/Method 36
SARIMA Model H-Step Prediction RMSE Value – 36
BASE Models H-Step Prediction RMSE Value – 36
Multiple Linear Regression 38
Final Model and h-step predictions 38
Hypothesis tests analysis: F-test, t-test 39
T-Test 39
F-test 39
AIC, BIC, RMSE, R-squared and Adjusted R-squared 39
ACF of residuals 40
Q value, Mean and Variance of Residuals 40
SARIMA model Order Determination 41
Preliminary model development procedures and results 41
Discussion of Autocorrelation function and GPAC Table and subsequent plots 42
GPAC Table 44
ARMA Model Parameter Estimation 46
Estimated Parameters , Standard Deviation, and Confidence Intervals for ARMA(0,24) 46
Estimated Parameters , Standard Deviation, and Confidence Intervals for ARMA(24,0) 47
Model Development and Diagnostic Analysis 50
Diagnostic tests (confidence intervals, zero/pole cancellation, chi-square test, Residual Analysis) 51
Estimated Variance and Estimated covariance of the Estimated Parameters 53
Estimator : biased or unbiased 57
Variance of the residual errors versus the variance of the forecast errors 57
Long Short-Term Memory Deep Learning Model(LSTM) 58
Additional Model (Gradient Boosting Regressor) 60
Final Model Selection 61
Forecast Function 65
h-Step Predictions 66
Summary and Conclusion 67
Appendix 68
References 93

Table of Figures and tables

Figure 1(Lecture 1 Notes)....7
Figure 2 11
Figure 3 12
Figure 4 12
Figure 5 12
Figure 6 13
Figure 7 13
Figure 8 14
Figure 9 14
Figure 10 15
Figure 11 16
Figure 12 16
Figure 13 17
Figure 14 17
Figure 15 18
Figure 16 19
Figure 17 19
Figure 18 20
Figure 19 21
Figure 20 21
Figure 21 22
Figure 22 22
Figure 23 23
Figure 24 23
Figure 25 23
Figure 26 24
Figure 27 24
Figure 28 25
Figure 29 26
Figure 30 26
Figure 31 27
Figure 32 28
Figure 33 28
Figure 34 29
Figure 35 29
Figure 36 30
Figure 37 30
Figure 38 33
Figure 39 34
Figure 40 35
Figure 41 35
Figure 42 36
Figure 43 36
Figure 44 37
Figure 45 38
Figure 46 38

Figure 47 39
Figure 48 39
Figure 49 39
Figure 50 39
Figure 51 40
Figure 52 40
Figure 53 41
Figure 54 42
Figure 55 42
Figure 56 43
Figure 57 44
Figure 58 44
Figure 59 45
Figure 60 46
Figure 61 47
Figure 62 47
Figure 63 48
Figure 64 49
Figure 65 50
Figure 66 51
Figure 67 52
Figure 68 52
Figure 69 53
Figure 70 57
Figure 71 58
Figure 72 58
Figure 73 59
Figure 74 60
Figure 75 61
Figure 76 62
Figure 77 63
Figure 78 64
Figure 79 64
Figure 80 65
Figure 81 66
Figure 82 66

Abstract

This project implements and dives deeper into a highly interdisciplinary area of data science called time series forecasting using python programming language. It aims to perform everything right from the data searching to the model building and model finalization for the chosen problem in this project. The project developer starts off by picking the data from Kaggle. For this project, the project developer picked up a dataset for wind turbine power generation. The goal here is to predict the power output of the wind turbine based on the given testing data. Any models built, imported, and used will employ the training data to learn the patterns and use that model to predict future values of Power output of wind turbine in the most accurate manner(relative to other models used for this topic) and compare it with testing data to find root mean square error for prediction which is the basis of comparison for their models. Project team first performed data cleaning and data imputation to make the data void of any errors, missing rows, and column values. Then developer plotted the data to get a bird's eye view of the topic at hand along with basic exploratory data analysis using acf_pacf plot and heatmaps. The data was passed through various custom built and pre-defined functions and logics to check for stationarity of the data and then the developer used those findings to perform differencing on the data make it ready for amazingly taught custom functions called GPAC(General partial Autocorrelation) and custom function based on Levenberg Marquardt(L.M.) Algorithm. The developer also built the base models and did h-step prediction ahead in time with those base models. Since they are base models with linear and simple calculations, predictions were barely accurate, it concluded that those base models are not suitable for the data. The team did backward-stepwise feature reduction of the feature space starting off with checking for Variance Inflation Factor, condition number and singular values to see if the data had multicollinearity which was not the case in their situation. The project developer then continued with feature reduction by building models in a stepwise fashion 69 times to remove those 69 insignificant features with p value higher than 0.05. Then the developer designed an OLS model with the remaining 7 of the total 76 features and found that OLS model was not able to explain the variance in the target variable. Then the developer took the differenced data from earlier, fit it into GPAC to get preliminary order from the data. Then the developer referred to the output of GPAC and used it to build the Parameter estimation function's parameters which seemed very vague and way off the expected parameters that should have come with multiple zero/pole cancellations and output coefficients for the potential forecast function that should not have existed. Then the developer built a SARIMA model for the data based off ACF_PACF plot and the estimated order from GPAC and interpretation of the behavior of LM algorithm. The residual for the SARIMA model was not white meant that there was a missing order not calculated by GPAC, so the developer fed the residuals' acf values to GPAC and it gave an uninterpretable order. Then to test it out the developer tried multiple combinations of potential order and it was still a dead end. Then the developer tried to put the suspicion of data being non-linear(*making unsuitable for the predictions using linear models of time series like ARIMA and SARIMA*) to test by running a deep learning recurrent neural network to perform regression, called Long Short Term Memory(L.S.T.M.) which gave drastically improved accuracy and it was evident from the plot of predicted v/s test data values and the accuracy score of ~ 60%. The developer then ran a gradient boosting regressor which is another machine learning algorithm for nonlinear data, and it gave an accuracy of prediction as 94% which solidified the belief and goal the developer started with, which was to predict the values to the of power in the most accurate manner.

Introduction

On a word-meaning basis, The Time series analysis is an interdisciplinary subject and a niche subset of statistical modelling and data science. It uses a step-by-step methodology to take the data and pass it through those steps to find the information in the data and map it to the built model so that the model can then use that learning(shallow or deep learning) to forecast future values The simplest representation is shown below-

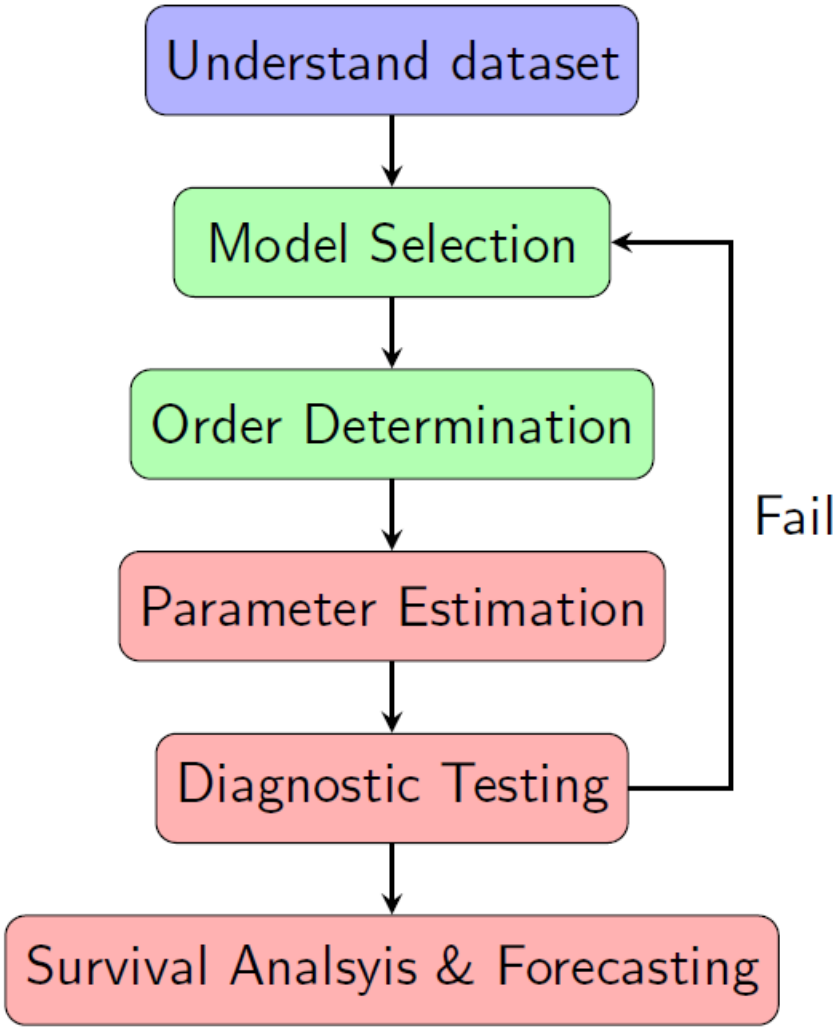


Figure 1(Lecture 1 Notes)

The developer has followed the same top-down approach shown above with the feedback loop to complete this project. Time series analysis allows us to interpret changes in data over time by identifying underlying patterns that are invisible to the naked eye. It has a wide range of applications across industries where data is mapped with time and/or data is generated with time and the data scientists have access to it to make meaningful decisions based on the results of the analysis.

In this report the project developer has gone through the above cycle in a detailed manner. They start off by describing the dataset and getting an overview of the data to get first impressions and

giving them possibility to narrow down the directions of solutions to think into. The developer also cleans the data for the possible hiccups and flaws in it. Then the project developer checks for stationarity and other data analyzing metrics like time series decomposition, seasonal and non-seasonal differencing. Then he checks how the data predicts against base time series models like average, naive, drift, simple exponential smoothing model, and holt winter model. with simpler and fundamental calculation equations. The developer then performs a backward stepwise regression to find the most significant features of the data to predict and forecast the values of dependent variable. Once that is done, the developer then generates metrics for the final model built using backward stepwise regression. Then the project developer takes the differenced data in the data preprocessing and analyzing step and feeds it into GPAC function built through the academic learnings to find the potential order and then feeds those orders into parameter estimation function also built as part of academic learning. This process of picking up generated order from gpac and feeding it into parameter estimation continues until the final understanding of the results from these steps is made. Then the developer runs the in-built function of python for time series analysis called SARIMAX which is used to the seasonal data's prediction, forecasting and diagnostics analysis. Then with the hypothesis of data being nonlinear for any/all linear models of time series forecasting, the developer constructs and runs a deep learning model to check if the patterns in data can be captured by this non linear model and then to further solidify the hypothesis into a fact for this project runs a machine learning nonlinear regression model called gradient boosting regressor to predict the values and compare the accuracy with the previous built primitive and time series and deep learning models. The accuracy from this gradient model is very high and then the project developer is able to confirm that the data is in fact nonlinear and the best model to represent it is the Gradient boosting Regressor model followed by the deep learning model L.S.T.M.

The Dataset

Description of the dataset

The dataset has 78 columns with the first and last column being the timestamp of the data and the regressand variable. The initial data without any pre-processing was split into 2 csv files called features.csv and power.csv. The developer merged those 2 csv files to create a singular data frame with 136730 rows of data and 78 columns.

The data has a lot of features that are metrics tracked in parts for example – Fan blade angle-1, Fan blade angle-2, Fan blade angle-3. Therefore, that kind of metric is merged into one followed by a number in brackets stating how many features with same name are there. Following is the description of the features of the data-

Independent variables

Feature Name	Feature description
Timestamp	The time of recording the data(10-minute intervals)
Gearbox_T1_High_Speed_Shaft_Temperature	High speed shaft's temperature that is in the gearbox T1
Gearbox_T3_High_Speed_Shaft_Temperature	High speed shaft's temperature that is in the gearbox T3
Gearbox_T1_Intermediate_Speed_Shaft_Temperature	Intermediate speed shaft's temperature that is in the gearbox T1
Temperature Gearbox Bearing Hollow Shaft	Hollowshaft contained gearbox's temperature
Tower Acceleration Normal	Normal acceleration of tower
Gearbox_Oil-2_Temperature	2nd slot of oil's temperature in the gearbox
Tower Acceleration Lateral	Lateral acceleration of tower due to wind
Temperature Bearing_A	Ball bearing A temperature
Temperature Trafo-3	The temperature of trafto 1
Gearbox_T3_Intermediate_Speed_Shaft_Temperature	Intermediate speed shaft's temperature that is in the gearbox T3
Gearbox_Oil-1_Temperature	1st slot of oil's temperature in the gearbox
Gearbox_Oil_Temperature	Overall oil temperature
Torque	Torque generated by wind
Converter Control Unit Reactive Power	Reactive power of converter control unit
Temperature Trafo-2	The temperature of trafto 1
Reactive Power	Overall reactive power
Temperature Shaft Bearing-1	temperture for ball bearing in shaft
Gearbox_Distributor_Temperature	the temperature for gearbox in distributor
Moment D Filtered	filtered moment D value.
Moment D Direction	Moment D direction.
N-set 1	N set 1 value
Operating State	State of operation of turbine.

Power Factor	The power factor of turbine.
Temperature Shaft Bearing-2	Second bearings' shaft temperature.
Temperature_Nacelle	Nacelle unit temperature.
Voltage A-N	First, out of three voltages.
Temperature Axis Box-3	Axis box's' 3rd temperature.
Voltage C-N	Third out of three voltages.
Temperature Axis Box-2	Axis box second temperature.
Temperature Axis Box-1	Axis box first temperature.
Voltage B-N	Second out of three voltages.
Nacelle Position_Degree	Nacelle Position in degrees.
Converter Control Unit Voltage	Voltage in the converter control unit.
Temperature Battery Box-3	Third Temperature Value of the battery box.
Temperature Battery Box-2	Second, temperature value of the battery box.
Temperature Battery Box-1	First, temperature value of the battery box.
Hydraulic Prepressure	The prior pressure in terms of hydraulics
Angle Rotor Position	Position of angle rotor in the turbine.
Temperature Tower Base	Temperature at the base of the wind turbine.
Pitch Offset-2 Asymmetric Load Controller	Load controllers value for asymmetric and second pitch offset.
Pitch Offset Tower Feedback	Feedback value of the tower for pitch offset.
Line Frequency	Power line transmission frequency.
Internal Power Limit	Power generation limit internally for the turbine
Circuit Breaker cut ins	Cut-in points installed and active in the turbines for circuit breaking when overload.
Particle Counter	Particle counter for the turbine.
Tower Accelaration Normal Raw	Raw but Normal acceleration of tower.
Torque Offset Tower Feedback	Torque's offset value From Towers feedback.
External Power Limit	External power limit for This turbine.
Blade-2 Actual Value_Angle-B	Angle B. Or second angle. For blade 2.
Blade-1 Actual Value_Angle-B	Angle B. Or second angle. For blade 1
Blade-3 Actual Value_Angle-B	Angle B. Or second angle. For blade 3
Temperature Heat Exchanger Converter Control Unit	Temperature inside heat exchanger converter control unit.
Tower Accelaration Lateral Raw	Lateral and raw. Tower acceleration.
Temperature Ambient	Ambient temperature in the turbine.
Nacelle Revolution	Revolutions made by. nacelle
Pitch Offset-1 Asymmetric Load Controller	Asymmetric load controllers pitch First, offset value.
Tower Deflection	Deflection shown by the tower.

Pitch Offset-3 Asymmetric Load Controller	Asymmetric load controllers pitch third, offset value.
Wind Deviation 1 seconds	When deviation in turbine Tracked every one second.
Wind Deviation 10 seconds	When deviation in turbine Tracked every ten second.
Proxy Sensor_Degree-135	Proxy sensors that are set at 135 degrees.
State and Fault	State and fault metric (numeric) For the turbine.
Proxy Sensor_Degree-225	Proxy sensors that are set at 225 degrees.
Blade-3 Actual Value_Angle-A	Third blade's . A angel's Actual value.
Scope CH 4	Scope value for CH4.
Blade-2 Actual Value_Angle-A	Second blade's . A angel's Actual value.
Blade-1 Actual Value_Angle-A	First blade's . A angel's Actual value.
Blade-2 Set Value_Degree	Blade two's. Set value. By the turbine operator in degrees.
Pitch Demand Baseline_Degree	Expected. Or demanded. Baseline value in degrees for the pitch.
Blade-1 Set Value_Degree	Blade one's. Set value. By the turbine operator in degrees.
Blade-3 Set Value_Degree	Blade three's Set value. By the turbine operator in degrees.
Moment Q Direction	Angular moment Q's direction.
Moment Q Filltered	Angular moment Q's Filtered value.
Proxy Sensor_Degree-45	Proxy sensor. Set at 45 degrees.
Turbine State	State of the turbine.
Proxy Sensor_Degree-315	Proxy sensor set at 315. Out of 360 degrees.

Table 1

Dependent Variable

Feature	Description
Power(kW)	Power output by the turbine in kilowatts

Table 2

```
       Gearbox_T1_High_Speed_Shaft_Temperature  ...  Proxy Sensor_Degree-315
count                             22968.000000  ...             22968.000000
mean                                896.590676  ...               858.780839
std                                6973.493535  ...              6982.781685
min                                -273.000000  ...                -0.238800
25%                                  44.370973  ...                 5.745573
50%                                  57.345000  ...                 5.803725
75%                                  64.513056  ...                 5.885402
max                               99999.000000  ...             99999.000000

[8 rows x 76 columns]
```

Figure 2

```
Out[897]:
count    22968.000000
mean      1137.648328
std       1058.590921
min        -46.188889
25%        102.954028
50%        815.469991
75%       2251.468679
max       2777.044963
Name: Power(kW), dtype: float64
```

Figure 3

Data Pre-Processing

The developer cleaned the data as there was an evident need for it. There were missing rows and few values in a few columns. Soo firstly, the developer ran the below code that gave the output followed by the code snippet.

```
#checking if timestamps are equidistant
time_diff = df['Timestamp'].diff()
if all(time_diff == 600):
    print("The timestamps are equally separated by 10 minutes")
else:
    print("The timestamps are not equally separated by 10 minutes")
```

Figure 4

The output for above code was -

```
The timestamps are not equally separated by 10 minutes
```

Figure 5

Now after that we resampled the data on hourly basis because as stated earlier the data had missing values, so we had to take this measure. We then checked for missing values in hourly data again using below code –

```
# select the index values where the time difference is not 3600 seconds
df['time_diff1'] = (df['Timestamp'] - df['Timestamp'].shift(1)).dt.seconds
mask = df['time_diff1'] != 3600
df['time_diff1'].to_csv("checkDownsampled_Dataset.csv")
result = df.loc[mask, 'Timestamp'].index

print("The non equidistant timestamps after sampling are - ", result)
df = df.drop(columns='time_diff1')
```

Figure 6

Plot of the dependent variable versus time

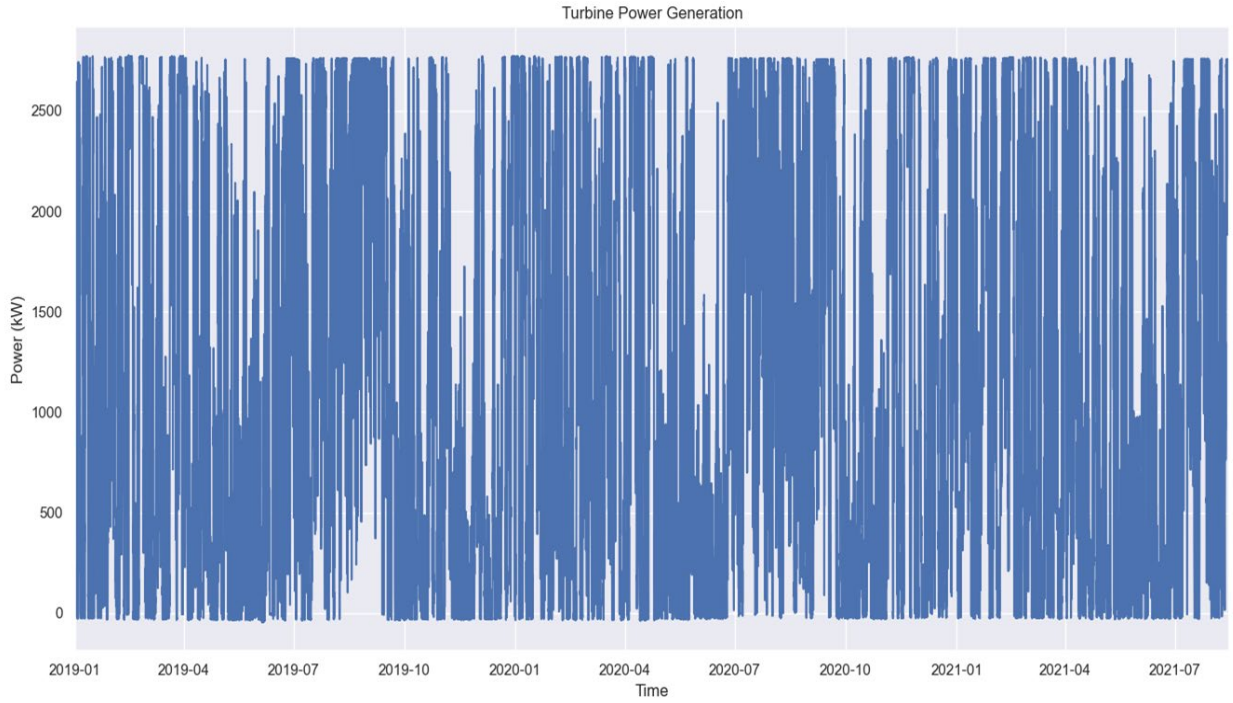


Figure 7

The plot shows no clear trend or seasonality pattern in the data from this first look. The data oscillates between an upper and a lower bound of values.

ACF/PACF of the dependent variable

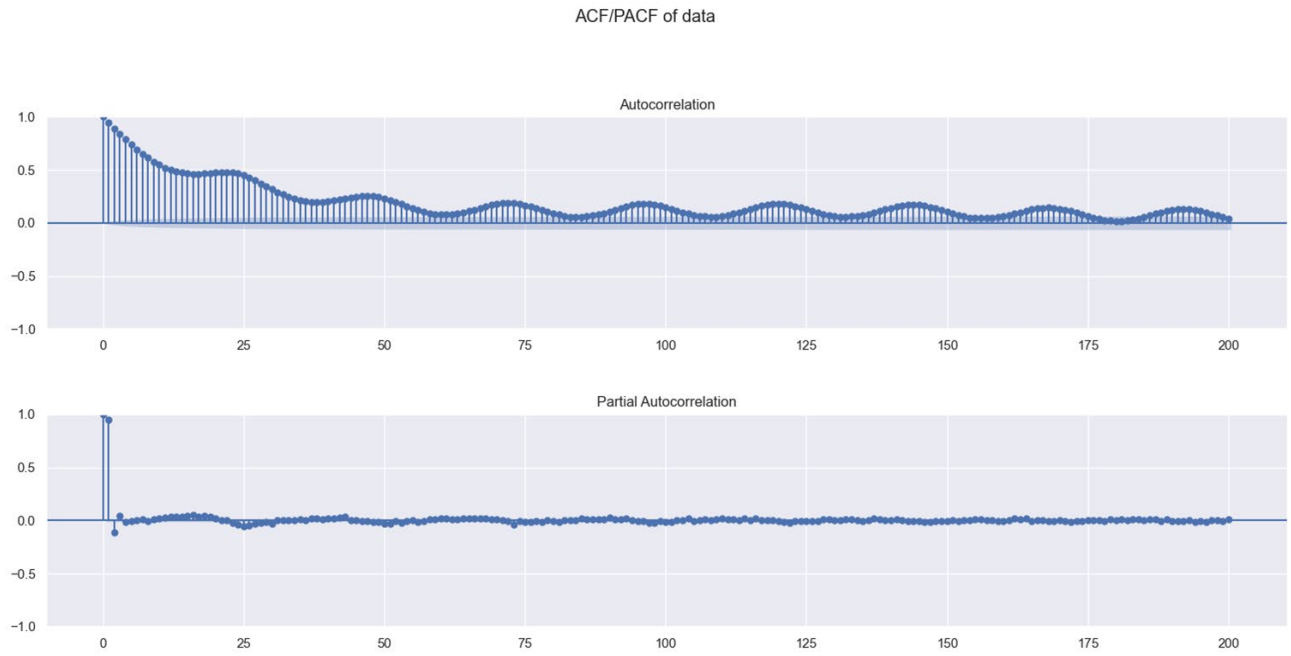


Figure 8

The ACF_PACF plot shows that there is a clear seasonality of 24 in the data with a wavy infinite decay in the ACF Plot and cutoff in PACF plot. The data is plotted for 200 lags.

Correlation Matrix

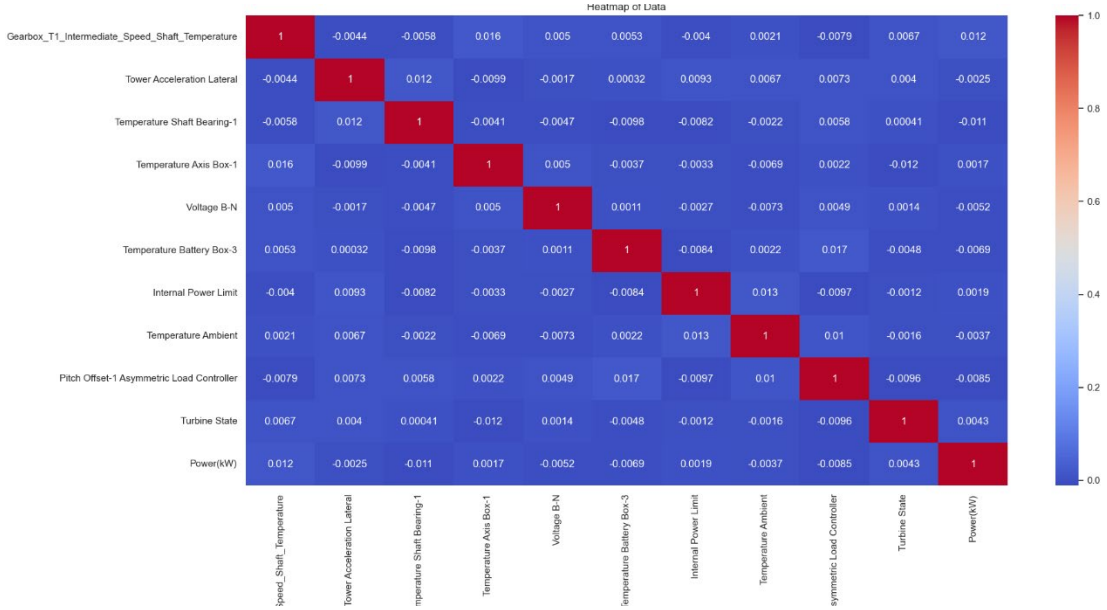


Figure 9

Very weak correlation between the variables and even weaker correlation overall with the regressand variable. The regressand variable power(kW) shows the highest negative correlation with symmetric load controller variable of -0.0085. The reason for only choosing these features is for the

visibility and interpretability of the plot. The module SelectKBest was used to choose the features to be displayed in the report and those K features are the best that represent the data as per the in-built module's function's calculation. We will have a very important use of these set of features during modelling which will be mentioned in the model of gradient boosting regressor.

Splitting the data

```
Size of independent variable's training set is - (18374, 76)
Size of dependent variables's training set is - (18374, 1)
Size of independent variable's test set is - (4594, 76)
Size of dependent variables's test set is - (4594, 1)
```

Figure 10

We split the data by setting shuffle = false to get the time series in the exact and correct order as it is after pre-processing.

Stationarity

To check if there is a need to make the dependent variable(Power(kW)) stationary, we perform two tests and one plot, and their outputs are shown below –

AD Fuller Test(Raw data)

```
ADF Statistic: -12.109727
p-value: 0.000000
Critical Values:
    1%: -3.431
    5%: -2.862
    10%: -2.567
```

Figure 11

These results above show that the data is stationary according to ADF test as the p value is less than 0.05 or 95 percent confidence interval making us reject the null and accept the alternate hypothesis which states that the variable in question or dependent variable is stationary.

KPSS Test(Raw data)

```
Results of KPSS Test:
Test Statistic            0.132786
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
```

Figure 12

These results above show that the data is stationary according to KPSS test as the p value is greater than 0.05 or 95 percent confidence interval making us accept the null and reject the alternate hypothesis. The null hypothesis states that the variable in question or dependent variable is stationary.

Plot of Rolling mean and variance(Raw data)

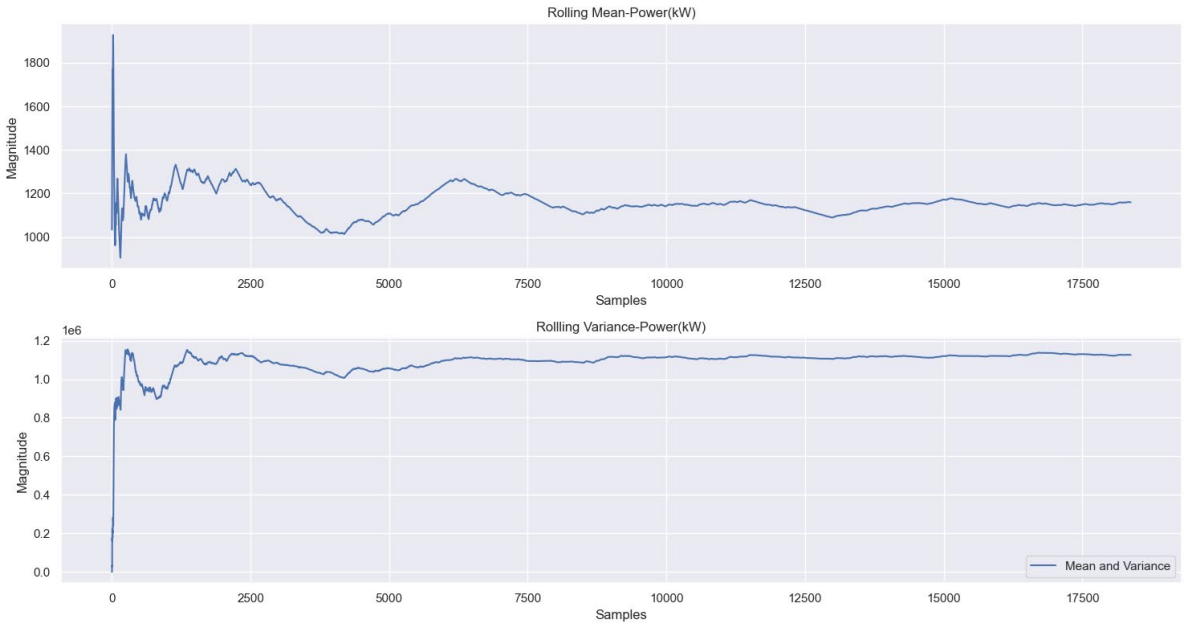


Figure 13

All these test points us in one direction which is that the data is stationary as we see the third test which is a plot that shows the rolling mean and variance of the data is stabilized after around 16000 samples for the regressand variable.

ACF/PACF Analysis for Stationarity(Raw data)

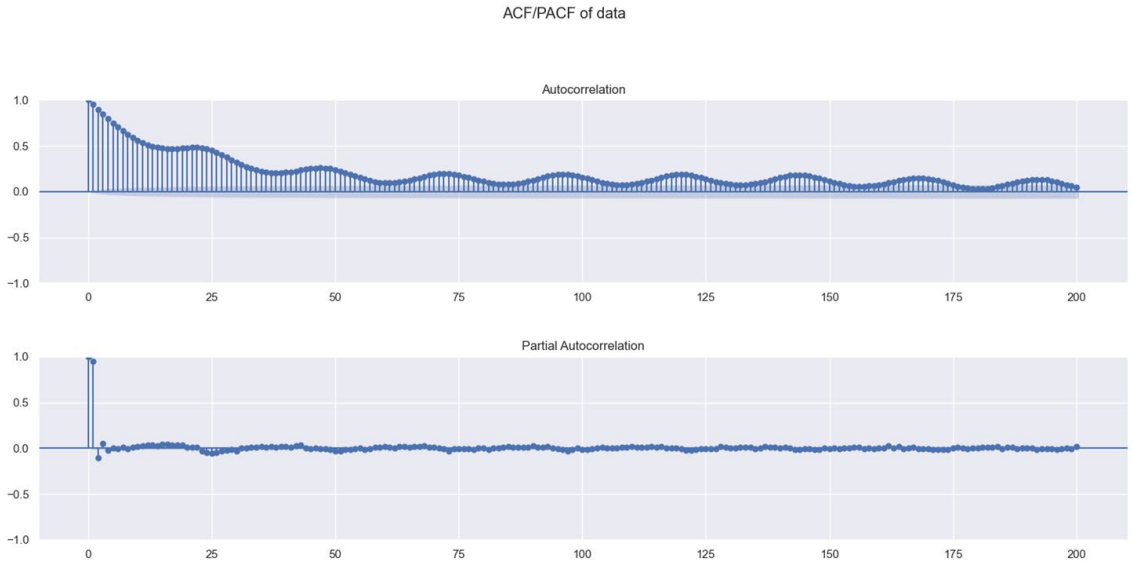


Figure 14

This is where the picture changes a bit, because if we look at ACF PACF from the perspective of stationarity, there looks like a visible need for differencing to make the data void of seasonal non-

stationarity and if we get picky we are also seeing that there is a need for non-seasonal differencing of order 1 as the ACF is a wavy infinite decay and pacf is in cutoff which shows there is still a possibility of trend in the data which was not captured by the ADF and KPSS test and plot of rolling mean and variance.

We have made a discovery in the time series decomposition right down below and that is where we will cover the ADF-test and kpss-test and plotting of the rolling mean and variance for the the transformed data.

Time Series Decomposition

Here we have chosen period = 24 for our data as in ACF/PACF plots above we were able to confirm that our data has decayed peaks in ACF/PACF after every 24 lags. That surely accounts for something and that something here is the seasonal period in our data, and we have done our decomposition based on that. Below is our approximated STL decomposition combination plot-

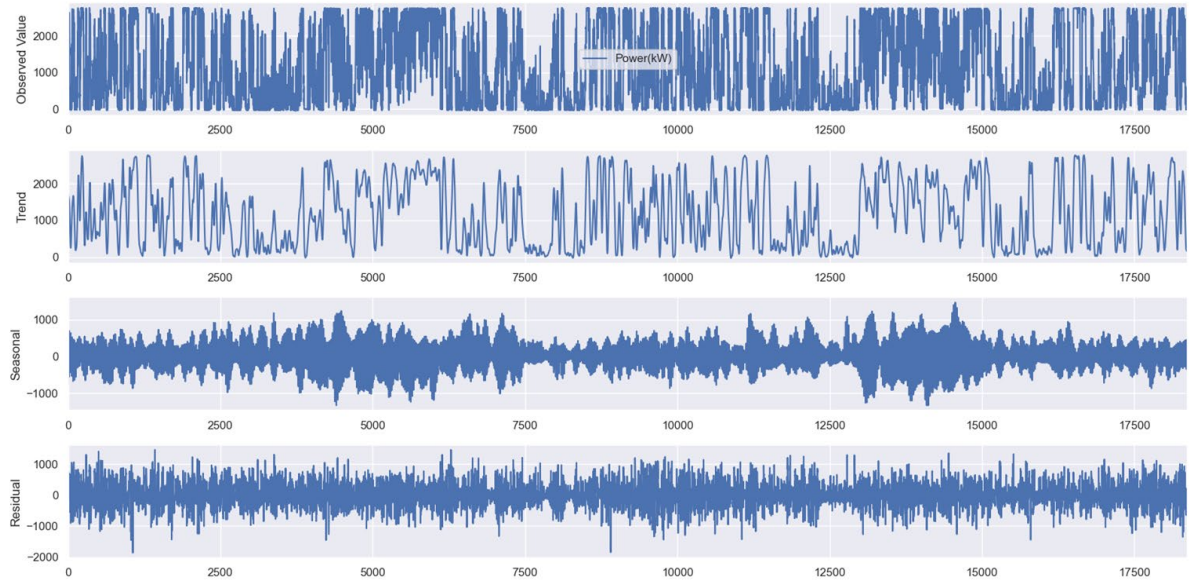


Figure 15

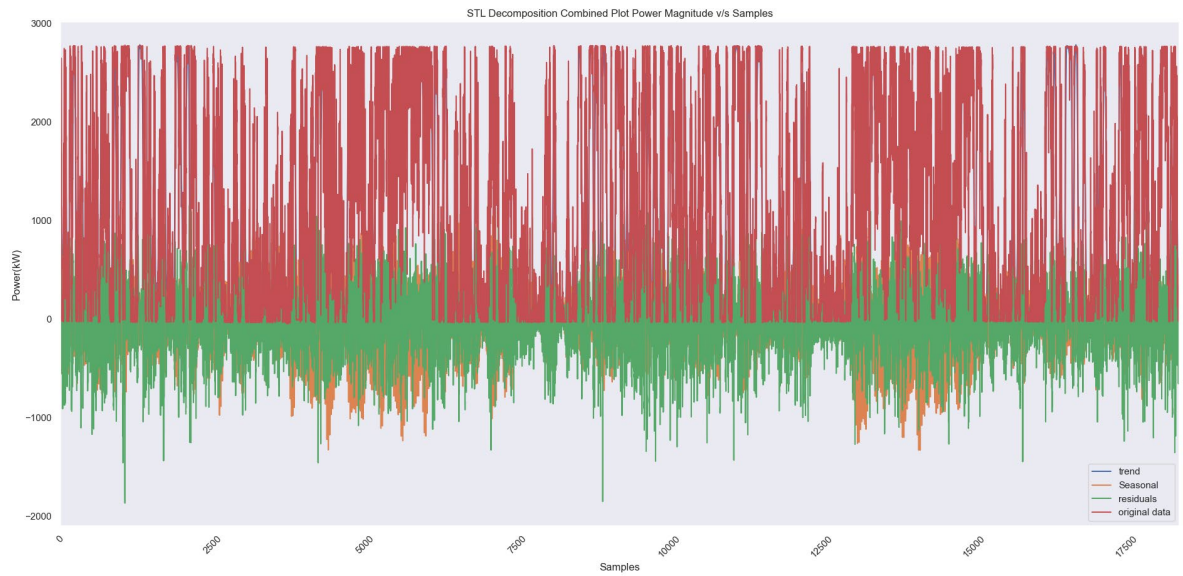


Figure 16

```
The strength of trend for this data set is -  0.8611
The strength of seasonality for this data set is -  0.5224
```

Figure 17

This is where the results are contradictory, and things get weird. Now after proving that the data is stationary earlier, we still see a high trend component in the data. The strength of seasonality shows its moderate to high seasonality which makes sense and adheres to our belief made from the plot of ACF/PACF earlier. The data should not be trended as per the tests for stationarity done before, but the strength of trend and strength of seasonality tests say here otherwise and contradict our belief earlier that data is stationary and not trended.

Thanks to STL decomposition's ability to handle trended data, we got to know this at this stage. Now that calls for performing a non-seasonal differencing of 1 order not to just confirm the strength of trend to become zero but to also check if the results of stationarity test vary or not with the differenced data. Since we would have performed the differencing already we would check the result of the time series decomposition using STL again with the differenced data (as we would have done earlier had we found the trend through adf kpss and rolling mean and variance test) and then we will finally be plotting the detrended and seasonally adjusted data.

STL decomposition - after recent discovery

Rolling Mean and Variance 2.0(Transformed data)

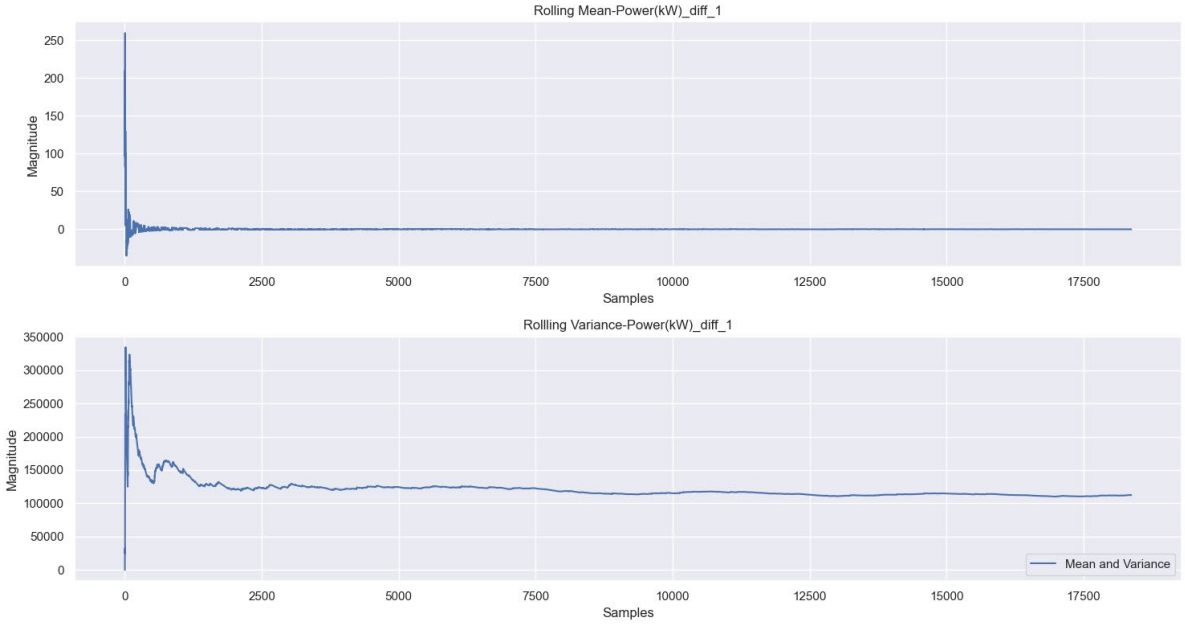


Figure 18

The rolling mean and variance has become even more stabilized so this had no negative effect on the data as it should be.

ADF and KPSS test (Transformed data)

```
ADF Statistic: -29.500953
p-value: 0.000000
Critical Values:
    1%: -3.431
    5%: -2.862
    10%: -2.567
Results of KPSS Test:
C:\Users\patha\AppData\Local\Packages\PythonSo
look-up table. The actual p-value is greater t

  warnings.warn(
Test Statistic            0.002158
p-value                   0.100000
Lags Used                58.000000
Critical Value (10%)      0.347000
Critical Value (5%)       0.463000
Critical Value (2.5%)     0.574000
Critical Value (1%)       0.739000
dtype: float64
```

Figure 19

Both ADF and KPSS tests still reject and accept the null hypothesis respectively which means that the data is stationary according to them. Now we will see ACF/PACF Plot.

ACF/PACF 2.0 (Transformed data)

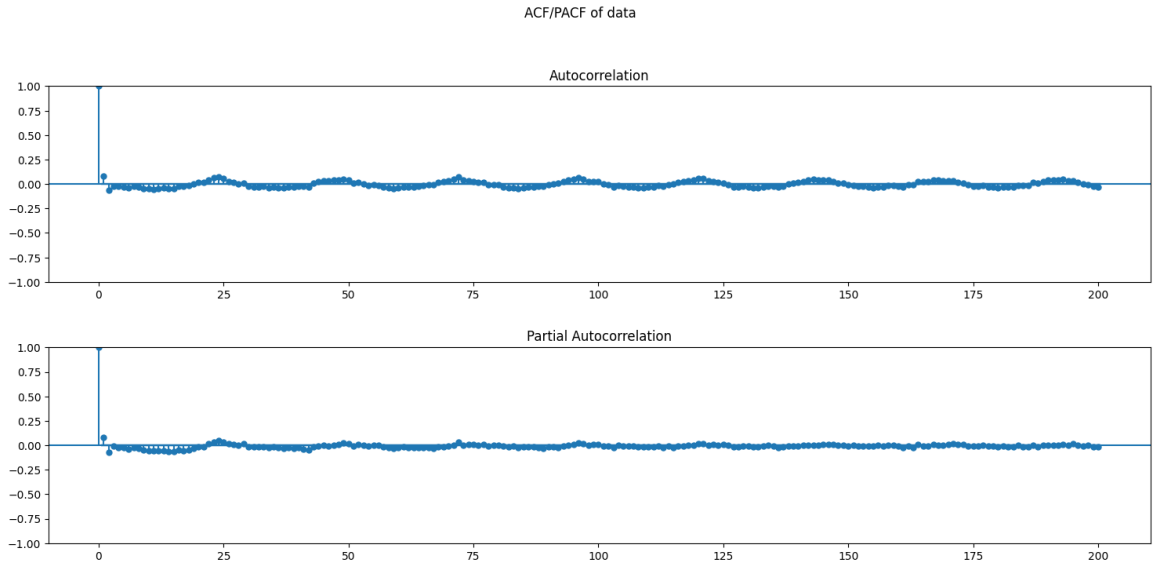


Figure 20

The results of acf/pacf plot have had a major change. We now only see a seasonal component in the data which we will remove for the data work with GPAC and LM Algorithm's parameter estimation and so on.

We will now also resume our STL decomposition from the start with this new data and if our logic is correct the trend component will be totally or close to totally gone. Let us check that below –

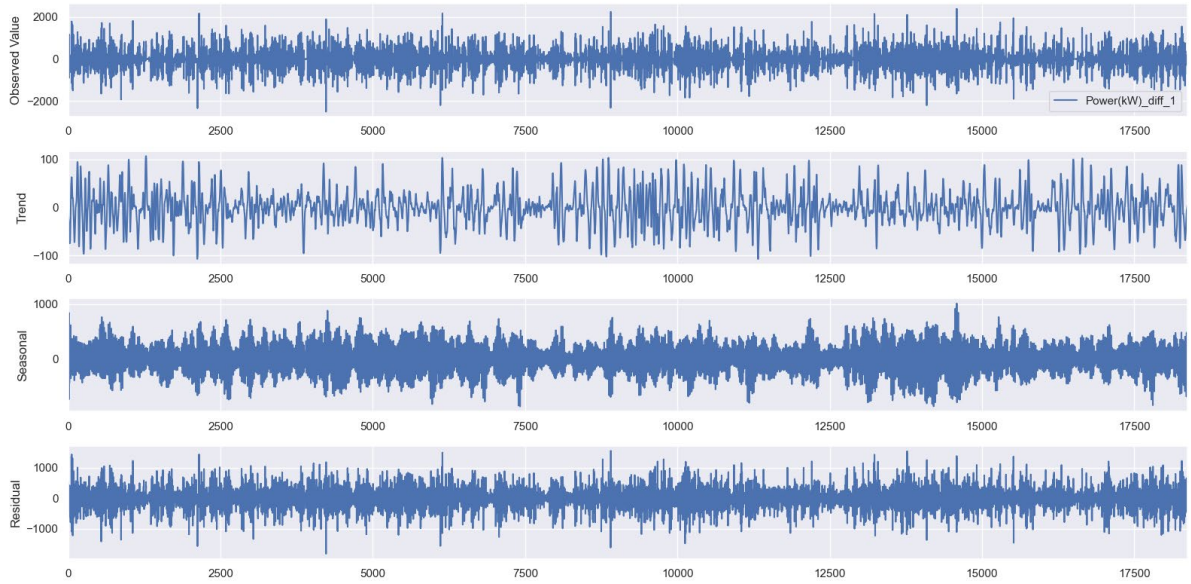


Figure 21

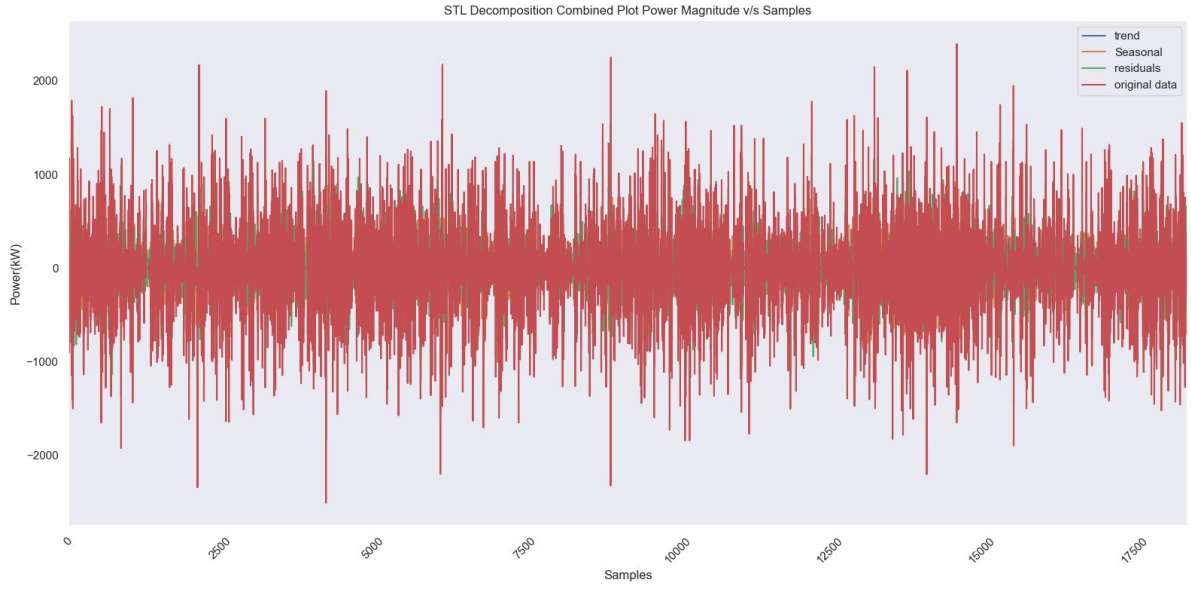


Figure 22

Clearly, we can see the difference between figures of STL decomposition before and after differencing. The plots look oscillatory. Let us see the strength of trend and seasonality.

```
The strength of trend for this data set is -  0.03
The strength of seasonality for this data set is -  0.3848
```

Figure 23

As expected, the strength of trend is almost gone, and we have still a considerable seasonality which makes sense with the kind of data we have so when we run GPAC and that point forward before that we will be doing a seasonal differencing of order 24. We will now plot the detrended and seasonally adjusted data.

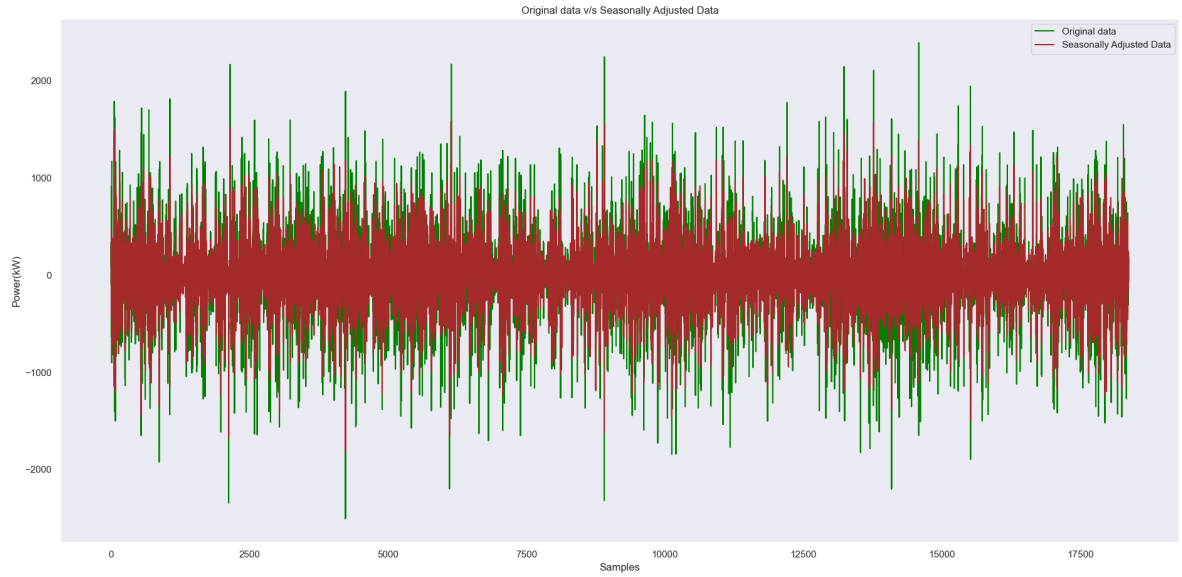


Figure 24

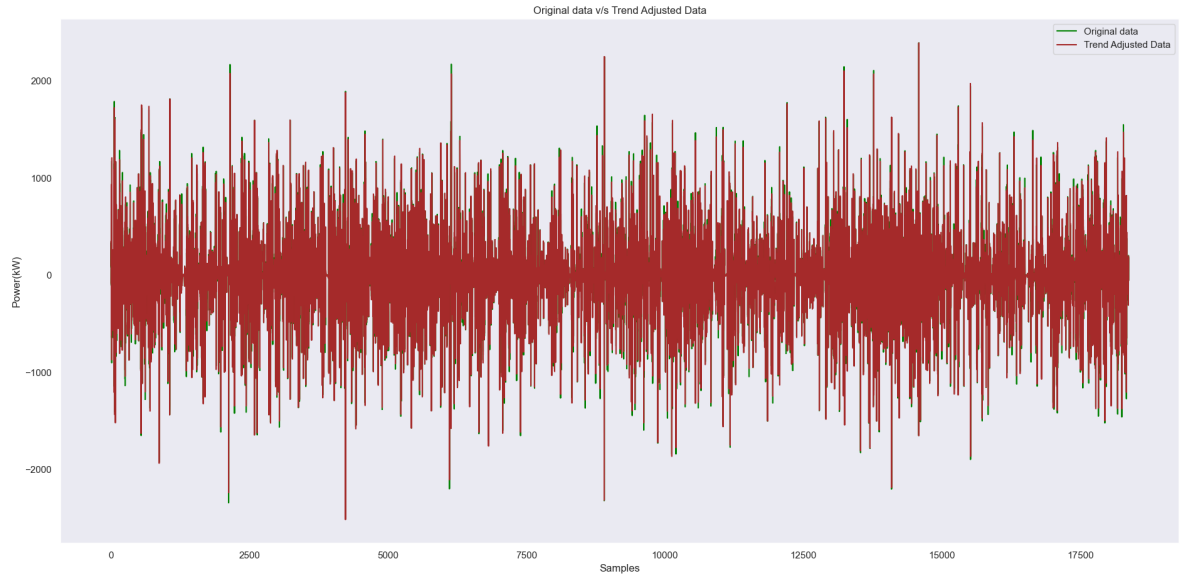


Figure 25

Holt Winters Method

This method is a basic method of time series forecasting where have performed it using the in-built package of python called “statsmodels.tsa.holtwinters”. This method is one of the best in terms of base model performance. It is still not the greatest method for forecasting, but it is a tad bit better than the other methods that we will be discussing below. We used this method to find the best fit using the train dataset and made prediction using test set. The value of accuracy metric that came to us here is as follows -

```
Root Mean square error for Holt-Winter method is 1308.39
```

Figure 26

Below is the prediction made using the test set and compared with the original data and we can see that the data predicted is far from accurate and therefore we also see a high error value as well.

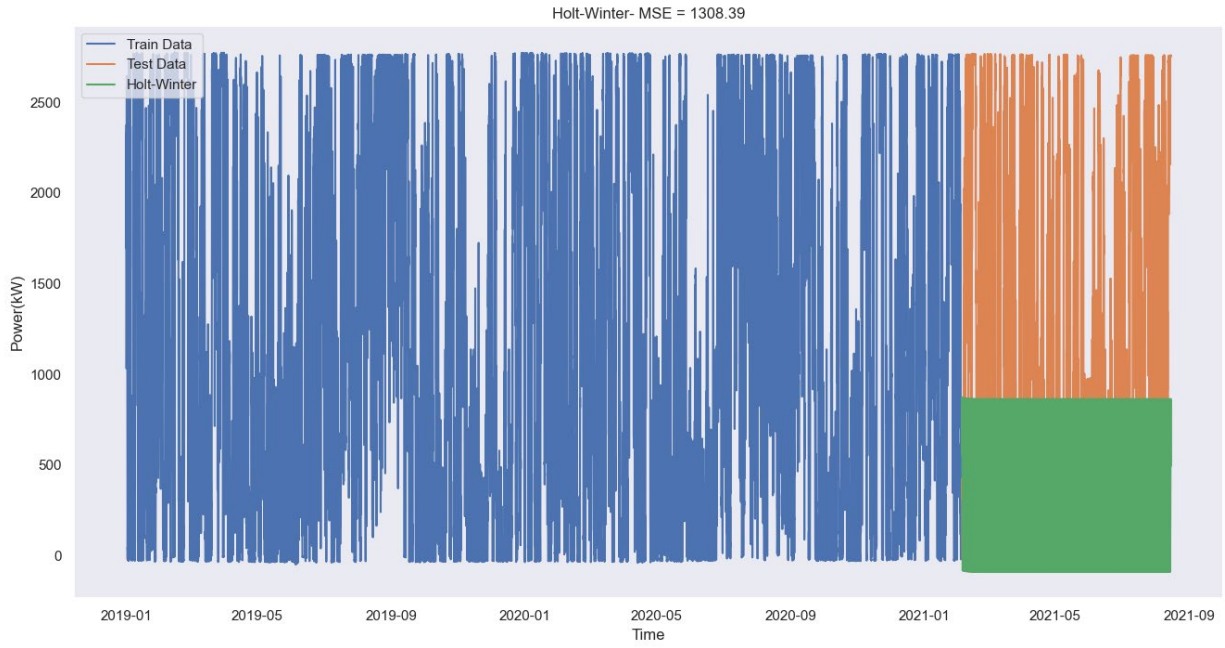


Figure 27

Feature Selection/Elimination

This is a process with clear motive and that is to remove the unnecessary features from our data that do not contribute to explaining the variance of the target variable. The idea behind this is very simple or we can say that the way it is implemented below is straight forward. So, for however many features do you have, you need to first make an initial model and then display the model summary with original data. This model will be built with all the features and whichever feature's p value is the highest makes that feature the most insignificant in the bunch and we need to remove that feature from the feature set and then make the model again and repeat this process until we get highest Adjusted R squared value from the leftover final set of features. The model you make from those features is going to be the final model for Ordinary Least Square Regression method.

Before we begin, we need to point out that it is good to check at least three things and those are values/output of following metrics –

Singular values from Singular value decomposition

```
SingularValues =  [3.73654787e+12 1.34545016e+12 1.33531037e+12 1.31050770e+12
 1.29001397e+12 1.28444726e+12 1.28130029e+12 1.26822280e+12
 1.25764203e+12 1.24918448e+12 1.24500776e+12 1.24237288e+12
 1.23324978e+12 1.23142726e+12 1.22624141e+12 1.22456248e+12
 1.21594527e+12 1.21189866e+12 1.21125910e+12 1.20592624e+12
 1.20018273e+12 1.19768412e+12 1.19432893e+12 1.19136184e+12
 1.18660418e+12 1.18339330e+12 1.17187847e+12 1.16969873e+12
 1.16710248e+12 1.16351184e+12 1.15773696e+12 1.15212596e+12
 1.15066447e+12 1.14394234e+12 1.14139902e+12 1.13846308e+12
 1.13591196e+12 1.13192360e+12 1.12560602e+12 1.12306264e+12
 1.11846308e+12 1.11487502e+12 1.11228751e+12 1.10856693e+12
 1.10657988e+12 1.10225613e+12 1.09810671e+12 1.09579794e+12
 1.09227826e+12 1.08856284e+12 1.08196162e+12 1.07999427e+12
 1.07552836e+12 1.07177740e+12 1.06592888e+12 1.06511779e+12
 1.06266157e+12 1.05871181e+12 1.05542169e+12 1.05437614e+12
 1.04585358e+12 1.04378270e+12 1.03669409e+12 1.03556436e+12
 1.03222176e+12 1.02633692e+12 1.02430360e+12 1.02019804e+12
 1.01029003e+12 1.00688809e+12 1.00399802e+12 1.00256994e+12
 9.93957809e+11 9.85547952e+11 9.60396871e+11 9.51232701e+11]
```

Figure 28

The whole idea behind printing singular values is to find if any values are closer to zero. If they are, we simply remove them as those values lead to one or more features being correlated. We did not have to do it as we can see all our singular values are high in number meaning none of our features are heavily or moderate to heavily correlated. This checks our we saw in our heatmap even for the best features of the dataset as per SelectKBest features package, they had a very little to no correlation amongst themselves.

Condition Number

Using numpy's linalg module we were able to calculate the below condition number which should be lowest as possible to confirm that there is no multicollinearity. That is exactly the case below as our condition number value is too low and we safely prove again that this dataset does not contain multicollinearity.

```
Condition Number =  1.9819
```

Figure 29

Variance Inflation Factor(V.I.F.)

feature	VIF
Gearbox_T1_High_Speed_Shaft_Temperature	1.0154
Gearbox_T3_High_Speed_Shaft_Temperature	1.0174
Gearbox_T1_Intermediate_Speed_Shaft_Temperature	1.0171
Temperature Gearbox Bearing Hollow Shaft	1.0170
Tower Acceleration Normal	1.0159
...	...
Moment Q Direction	1.0144
Moment Q Filltered	1.0142
Proxy Sensor_Degree-45	1.0153
Turbine State	1.0140

Figure 30

This is yet another factor that determines if a particular set of features has multicollinearity or not. By concept, the value for the V.I.F. should be very low and close to zero and should not exceed 5. With the pattern that we see for the multicollinearity metrics above, this metric is no different it also shows that our data has no multicollinearity. It is because the value is close to 0 and does not exceed 1 for the entirety of the list of features.

As we discussed the steps to perform the backward stepwise regression above, we will now start that process – At first we have all the features which we will use to make model. Now since we have different features that measure different metric in a wind turbine, they obviously come with a tricky part where all the SI units or units of measure in general vary a lot and with these many bunch of features it is very evident that all these metrics for example torque, degree angle, temperature for rotor blade etc, they have very different units, torque is a measure of force,angle is a value in degrees and temperature is in degree Celsius.

To make sure our model is not biased with weight of one feature we applied StandardScaler() to scale the data on a rather comparable level. Just because a feature has a bigger value does not mean it should contribute bigger to the model because if we do not scale OLS algorithm might just end up thinking that the bigger valued feature should contribute higher. Also, OLS works with a constant added to the front to account for the intercept and it forces residuals to have zero mean quite crucially.

After these steps we can begin going in a backward stepwise fashion to reduce the feature set. Since our feature space used to predict the target variable is too big, we will display the screenshot of the initial model with all 76 features and the final model with the reduced set of features. It is worth re-instating that we will be removing the features in the decreasing order of the p-value as it appears after performing model.fit().

First Step(Initial model fitting with all features)

OLS Regression Results

Dep. Variable:	Power(kW)	R-squared:	0.008
Model:	OLS	Adj. R-squared:	0.004
Method:	Least Squares	F-statistic:	2.014
Date:	Wed, 10 May 2023	Prob (F-statistic):	4.38e-07
Time:	18:30:06	Log-Likelihood:	-1.5401e+05
No. Observations:	18374	AIC:	3.082e+05
Df Residuals:	18297	BIC:	3.088e+05
Df Model:	76		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Constant	1159.2221	7.813	148.366	0.000	1143.907	1174.537
Gearbox_T1_High_Speed_Shaft_Temperature	-6.6144	7.834	-0.844	0.398	-21.970	8.741
Gearbox_T3_High_Speed_Shaft_Temperature	-3.7273	7.840	-0.475	0.634	-19.094	11.639
Gearbox_T1_Intermediate_Speed_Shaft_Temperature	6.1915	7.831	0.791	0.429	-9.158	21.541
Temperature Gearbox Bearing Hollow Shaft	12.9770	7.829	1.658	0.097	-2.368	28.322
Tower Acceleration Normal	16.3702	7.833	2.090	0.037	1.018	31.723
Gearbox_Oil-2_Temperature	-9.7337	7.829	-1.243	0.214	-25.080	5.612
Tower Acceleration Lateral	-3.8712	7.832	-0.494	0.621	-19.223	11.480

Figure 31

Temperature Bearing_A	-10.0323	7.836	-1.280	0.200	-25.391	5.327
Temperature Trafo-3	5.0035	7.826	0.639	0.523	-10.336	20.343
Gearbox_T3_Intermediate_Speed_Shaft_Temperature	2.6983	7.830	0.345	0.730	-12.649	18.046
Gearbox_Oil-1_Temperature	-3.9861	7.833	-0.509	0.611	-19.340	11.368
Gearbox_Oil_Temperature	-9.1929	7.829	-1.174	0.240	-24.539	6.153
Torque	10.3341	7.834	1.319	0.187	-5.022	25.690
Converter Control Unit Reactive Power	26.5535	7.828	3.392	0.001	11.210	41.897
Temperature Trafo-2	13.9153	7.827	1.778	0.075	-1.427	29.257
Reactive Power	31.6620	7.834	4.042	0.000	16.307	47.017
Temperature Shaft Bearing-1	-11.8693	7.832	-1.516	0.130	-27.220	3.481
Gearbox_Distributor_Temperature	0.5843	7.830	0.075	0.941	-14.763	15.932
Moment D Filtered	10.0866	7.829	1.288	0.198	-5.259	25.432
Moment D Direction	25.2241	7.828	3.222	0.001	9.880	40.569
N-set 1	50.8400	7.835	6.489	0.000	35.482	66.197
Operating State	-7.1093	7.832	-0.908	0.364	-22.461	8.243
Power Factor	0.9296	7.836	0.119	0.906	-14.431	16.290
Temperature Shaft Bearing-2	-11.4557	7.831	-1.463	0.144	-26.805	3.894
Temperature_Nacelle	-4.0616	7.833	-0.519	0.604	-19.415	11.292
Voltage A-N	6.0384	7.827	0.771	0.440	-9.304	21.381
Temperature Axis Box-3	-0.9692	7.831	-0.124	0.902	-16.320	14.381
Voltage C-N	9.9633	7.833	1.272	0.203	-5.389	25.316

Figure 32

Temperature Axis Box-2	-3.5271	7.830	-0.450	0.652	-18.875	11.820
Temperature Axis Box-1	0.5201	7.829	0.066	0.947	-14.826	15.866
Voltage B-N	-4.2117	7.826	-0.538	0.590	-19.551	11.128
Nacelle Position_Degree	-3.7116	7.831	-0.474	0.636	-19.062	11.639
Converter Control Unit Voltage	-0.8713	7.834	-0.111	0.911	-16.226	14.484
Temperature Battery Box-3	-6.5508	7.829	-0.837	0.403	-21.897	8.795
Temperature Battery Box-2	-0.3837	7.833	-0.049	0.961	-15.737	14.969
Temperature Battery Box-1	9.8514	7.828	1.258	0.208	-5.493	25.196
Hydraulic Prepressure	-1.0137	7.837	-0.129	0.897	-16.375	14.348
Angle Rotor Position	-14.0204	7.832	-1.790	0.073	-29.372	1.332
Temperature Tower Base	-2.6744	7.833	-0.341	0.733	-18.028	12.679
Pitch Offset-2 Asymmetric Load Controller	-3.2251	7.829	-0.412	0.680	-18.571	12.120
Pitch Offset Tower Feedback	2.2926	7.833	0.293	0.770	-13.061	17.646
Line Frequency	3.1320	7.831	0.400	0.689	-12.216	18.481
Internal Power Limit	-1.8356	7.833	-0.234	0.815	-17.188	13.517
Circuit Breaker cut-ins	-2.7872	7.830	-0.356	0.722	-18.136	12.561
Particle Counter	22.4023	7.833	2.860	0.004	7.050	37.755
Tower Accelaration Normal Raw	5.6454	7.836	0.720	0.471	-9.714	21.004
Torque Offset Tower Feedback	-3.0856	7.827	-0.394	0.693	-18.428	12.257
External Power Limit	-9.4631	7.831	-1.208	0.227	-24.813	5.887
Blade-2 Actual Value_Angle-B	-4.9069	7.833	-0.626	0.531	-20.259	10.446

Figure 33

Blade-1 Actual Value_Angle-B	14.3623	7.831	1.834	0.067	-0.987	29.711
Blade-3 Actual Value_Angle-B	-3.5780	7.826	-0.457	0.648	-18.918	11.762
Temperature Heat Exchanger Converter Control Unit	3.4375	7.827	0.439	0.661	-11.904	18.779
Tower Accelaration Lateral Raw	11.1307	7.831	1.421	0.155	-4.219	26.480
Temperature Ambient	-0.5848	7.831	-0.075	0.940	-15.935	14.765
Nacelle Revolution	-9.6571	7.831	-1.233	0.218	-25.007	5.692
Pitch Offset-1 Asymmetric Load Controller	-7.5292	7.835	-0.961	0.337	-22.887	7.828
Tower Deflection	2.9884	7.833	0.382	0.703	-12.365	18.342
Pitch Offset-3 Asymmetric Load Controller	-11.9738	7.833	-1.529	0.126	-27.327	3.379
Wind Deviation 1 seconds	15.6866	7.831	2.003	0.045	0.337	31.036
Wind Deviation 10 seconds	-3.4047	7.830	-0.435	0.664	-18.753	11.944
Proxy Sensor_Degree-135	-1.6457	7.828	-0.210	0.833	-16.989	13.697
State and Fault	5.3338	7.835	0.681	0.496	-10.023	20.691
Proxy Sensor_Degree-225	9.8549	7.831	1.258	0.208	-5.494	25.204
Blade-3 Actual Value_Angle-A	-3.0352	7.834	-0.387	0.698	-18.390	12.319
Scope CH 4	-2.8960	7.829	-0.370	0.711	-18.243	12.451
Blade-2 Actual Value_Angle-A	-8.7296	7.835	-1.114	0.265	-24.086	6.627
Blade-1 Actual Value_Angle-A	8.6953	7.830	1.110	0.267	-6.653	24.044
Blade-2 Set Value_Degree	5.7087	7.834	0.729	0.466	-9.648	21.065
Pitch Demand Baseline_Degree	-9.8780	7.829	-1.262	0.207	-25.224	5.468
Blade-1 Set Value_Degree	-2.1385	7.829	-0.273	0.785	-17.485	13.208

Figure 34

Blade-3 Set Value_Degree	0.0577	7.836	0.007	0.994	-15.301	15.416
Moment Q Direction	-2.8775	7.830	-0.368	0.713	-18.224	12.469
Moment Q Filltered	2.4714	7.831	0.316	0.752	-12.878	17.821
Proxy Sensor_Degree-45	9.7425	7.838	1.243	0.214	-5.620	25.105
Turbine State	8.5148	7.831	1.087	0.277	-6.835	23.865
Proxy Sensor_Degree-315	7.4997	7.832	0.958	0.338	-7.852	22.851

Omnibus:	129230.848	Durbin-Watson:	0.112
Prob(Omnibus):	0.000	Jarque-Bera (JB):	1982.682
Skew:	0.344	Prob(JB):	0.00
Kurtosis:	1.546	Cond. No.	1.14

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Figure 35

Now from this we first will remove “Blade-3 Set Value_Degree” as it has the highest p-value of 0.994 and after removing that feature, with the reduced dataframe we will fit the model with modified dataframe and generate the summary again as shown on the next page.

Second Step(model fitting with one less feature)

OLS Regression Results

Dep. Variable:	Power(kW)	R-squared:	0.008
Model:	OLS	Adj. R-squared:	0.004
Method:	Least Squares	F-statistic:	2.041
Date:	Wed, 10 May 2023	Prob (F-statistic):	3.02e-07
Time:	18:55:48	Log-Likelihood:	-1.5401e+05
No. Observations:	18374	AIC:	3.082e+05
Df Residuals:	18298	BIC:	3.088e+05
Df Model:	75		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Constant	1159.2221	7.813	148.370	0.000	1143.908	1174.536
Gearbox_T1_High_Speed_Shaft_Temperature	-6.6146	7.834	-0.844	0.398	-21.969	8.740
Gearbox_T3_High_Speed_Shaft_Temperature	-3.7251	7.834	-0.476	0.634	-19.080	11.630
Gearbox_T1_Intermediate_Speed_Shaft_Temperature	6.1916	7.831	0.791	0.429	-9.157	21.540
Temperature Gearbox Bearing Hollow Shaft	12.9769	7.829	1.658	0.097	-2.368	28.322
Tower Acceleration Normal	16.3695	7.832	2.090	0.037	1.018	31.721
Gearbox_Oil-2_Temperature	-9.7339	7.829	-1.243	0.214	-25.079	5.612
Tower Acceleration Lateral	-3.8710	7.832	-0.494	0.621	-19.222	11.480

Figure 36

.

.

.

.

.

State and Fault	5.3336	7.834	0.681	0.496	-10.023	20.690
Proxy Sensor_Degree-225	9.8548	7.831	1.259	0.208	-5.494	25.203
Blade-3 Actual Value_Angle-A	-3.0347	7.833	-0.387	0.698	-18.388	12.319
Scope CH 4	-2.8960	7.829	-0.370	0.711	-18.242	12.450
Blade-2 Actual Value_Angle-A	-8.7297	7.834	-1.114	0.265	-24.086	6.626
Blade-1 Actual Value_Angle-A	8.6953	7.830	1.110	0.267	-6.653	24.043
Blade-2 Set Value_Degree	5.7099	7.833	0.729	0.466	-9.643	21.062
Pitch Demand Baseline_Degree	-9.8767	7.827	-1.262	0.207	-25.218	5.465
Blade-1 Set Value_Degree	-2.1386	7.829	-0.273	0.785	-17.485	13.207
Moment Q Direction	-2.8779	7.829	-0.368	0.713	-18.224	12.468
Moment Q Filltered	2.4719	7.830	0.316	0.752	-12.876	17.820
Proxy Sensor_Degree-45	9.7433	7.837	1.243	0.214	-5.617	25.104
Turbine State	8.5145	7.831	1.087	0.277	-6.835	23.864
Proxy Sensor_Degree-315	7.4997	7.832	0.958	0.338	-7.851	22.850

Omnibus:	129231.262	Durbin-Watson:	0.112
Prob(Omnibus):	0.000	Jarque-Bera (JB):	1982.681
Skew:	0.344	Prob(JB):	0.00
Kurtosis:	1.546	Cond. No.	1.14

Figure 37

As we can spot in the first part of this shortened image, the Df Model parameter has a value of 75 instead of 76 from the previous run of the model. This parameter shows how many features are in use currently for the result generation/summary generation.

We have curated a list of features that were removed **one by one** from the total features space of size 76. It is shown in the next page -

Feature Removed	Feature description
Gearbox_T1_High_Speed_Shaft_Temperature	High speed shaft's temperature that is in the gearbox T1
Gearbox_T3_High_Speed_Shaft_Temperature	High speed shaft's temperature that is in the gearbox T3
Gearbox_T1_Intermediate_Speed_Shaft_Temperature	Intermediate speed shaft's temperature that is in the gearbox T1
Temperature Gearbox Bearing Hollow Shaft	Hollowshaft contained gearbox's temperature
Gearbox_Oil-2_Temperature	2nd slot of oil's temperature in the gearbox
Tower Acceleration Lateral	Lateral acceleration of tower due to wind
Temperature Bearing_A	Ball bearing A temperature
Temperature Trafo-3	The temperature of trafto 1
Gearbox_T3_Intermediate_Speed_Shaft_Temperature	Intermediate speed shaft's temperature that is in the gearbox T3
Gearbox_Oil-1_Temperature	1st slot of oil's temperature in the gearbox
Gearbox_Oil_Temperature	Overall oil temperature
Torque	Torque generated by wind
Temperature Trafo-2	The temperature of trafto 1
Temperature Shaft Bearing-1	temperture for ball bearing in shaft
Gearbox_Distributor_Temperature	the temperature for gearbox in distributor
Moment D Filtered	filtered moment D value.
Operating State	State of operation of turbine.
Power Factor	The power factor of turbine.
Temperature Shaft Bearing-2	Second bearings' shaft temperature.
Temperature_Nacelle	Nacelle unit temperature.
Voltage A-N	First, out of three voltages.
Temperature Axis Box-3	Axis box's' 3rd temperature.
Voltage C-N	Third out of three voltages.
Temperature Axis Box-2	Axis box second temperature.
Temperature Axis Box-1	Axis box first temperature.
Voltage B-N	Second out of three voltages.
Nacelle Position_Degree	Nacelle Position in degrees.
Converter Control Unit Voltage	Voltage in the converter control unit.
Temperature Battery Box-3	Third Temperature Value of the battery box.
Temperature Battery Box-2	Second, temperature value of the battery box.
Temperature Battery Box-1	First, temperature value of the battery box.
Hydraulic Prepressure	The prior pressure In terms of hydraulics

Angle Rotor Position	Position of angle rotor in the turbine.
Temperature Tower Base	Temperature at the base of the wind turbine.
Pitch Offset-2 Asymmetric Load Controller	Load controllers value for asymmetric and second pitch offset.
Pitch Offset Tower Feedback	Feedback value of the tower for pitch offset.
Line Frequency	Power line transmission frequency.
Internal Power Limit	Power generation limit internally for the turbine
Circuit Breaker cut-ins	Cut-in points installed and active in the turbines for circuit breaking when overload.
Tower Accelaration Normal Raw	Raw but Normal acceleration of tower.
Torque Offset Tower Feedback	Torque's offset value From Towers feedback.
External Power Limit	External power limit for This turbine.
Blade-2 Actual Value_Angle-B	Angle B. Or second angle. For blade 2.
Blade-1 Actual Value_Angle-B	Angle B. Or second angle. For blade 1
Blade-3 Actual Value_Angle-B	Angle B. Or second angle. For blade 3
Temperature Heat Exchanger Converter Control Unit	Temperature inside heat exchanger converter control unit.
Tower Accelaration Lateral Raw	Lateral and raw. Tower acceleration.
Temperature Ambient	Ambient temperature in the turbine.
Nacelle Revolution	Revolutions made by. nacelle
Pitch Offset-1 Asymmetric Load Controller	Asymmetric load controllers pitch First, offset value.
Tower Deflection	Deflection shown by the tower.
Pitch Offset-3 Asymmetric Load Controller	Asymmetric load controllers pitch third, offset value.
Wind Deviation 10 seconds	When deviation in turbine Tracked every ten second.
Proxy Sensor_Degree-135	Proxy sensors that are set at 135 degrees.
State and Fault	State and fault metric (numeric) For the turbine.
Proxy Sensor_Degree-225	Proxy sensors that are set at 225 degrees.
Blade-3 Actual Value_Angle-A	Third blade's . A angel's Actual value.
Scope CH 4	Scope value for CH4.
Blade-2 Actual Value_Angle-A	Second blade's . A angel's Actual value.
Blade-1 Actual Value_Angle-A	First blade's . A angel's Actual value.
Blade-2 Set Value_Degree	Blade two's. Set value. By the turbine operator in degrees.
Pitch Demand Baseline_Degree	Expected. Or demanded. Baseline value

	in degrees for the pitch.
Blade-1 Set Value_Degree	Blade one's. Set value. By the turbine operator in degrees.
Blade-3 Set Value_Degree	Blade three's Set value. By the turbine operator in degrees.
Moment Q Direction	Angular moment Q's direction.
Moment Q Filltered	Angular moment Q's Filtered value.
Proxy Sensor_Degree-45	Proxy sensor. Set at 45 degrees.
Turbine State	State of the turbine.
Proxy Sensor_Degree-315	Proxy sensor set at 315. Out of 360 degrees.

Table 3

After removing above list of features based off their high p-values in a backward stepwise manner, we are left with 7 features which are shown below –

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              Power(kW)   R-squared:                       0.005
Model:                            OLS   Adj. R-squared:                  0.005
Method:                 Least Squares   F-statistic:                     13.89
Date:                Wed, 10 May 2023   Prob (F-statistic):           4.49e-18
Time:                        19:01:19   Log-Likelihood:            -1.5404e+05
No. Observations:               18374   AIC:                         3.081e+05
Df Residuals:                   18366   BIC:                         3.082e+05
Df Model:                           7
Covariance Type:            nonrobust
=============================================================================================================
                                         coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------------------------------------
Constant                                1159.2221      7.810    148.419      0.000    1143.913    1174.531
Tower Acceleration Normal                 15.6123      7.812      1.999      0.046       0.300      30.924
Converter Control Unit Reactive Power     25.9971      7.812      3.328      0.001      10.684      41.310
Reactive Power                            32.5741      7.812      4.170      0.000      17.263      47.885
Moment D Direction                        25.5306      7.812      3.268      0.001      10.218      40.843
N-set 1                                   51.0277      7.812      6.532      0.000      35.715      66.341
Particle Counter                          22.5981      7.811      2.893      0.004       7.287      37.909
Wind Deviation 1 seconds                  15.9186      7.812      2.038      0.042       0.607      31.231
==============================================================================
Omnibus:                   124771.815   Durbin-Watson:                   0.108
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             2004.067
Skew:                           0.345   Prob(JB):                         0.00
```

Figure 38

To state in the report following features are being used by the model –

Tower Acceleration Normal, Converter Control Unit Reactive Power, Reactive Power, Moment D Direction, N-set 1, Particle Counter, Wind Deviation 1 seconds.

Base Models

For base models we will use the 4 iconic base models for time series which are – Average Model, naïve model, drift model and simple exponential smoothing model.

The hstep prediction was achieved using the standard modelling process of each of the models. We have used RMSE(Root mean squared error as a measure to compare each base model with SARIMA model's performance.

Average Model/Method

To make the output visually more understandable, we have plotted the h-step prediction output of values against their original values of the test set as follows –

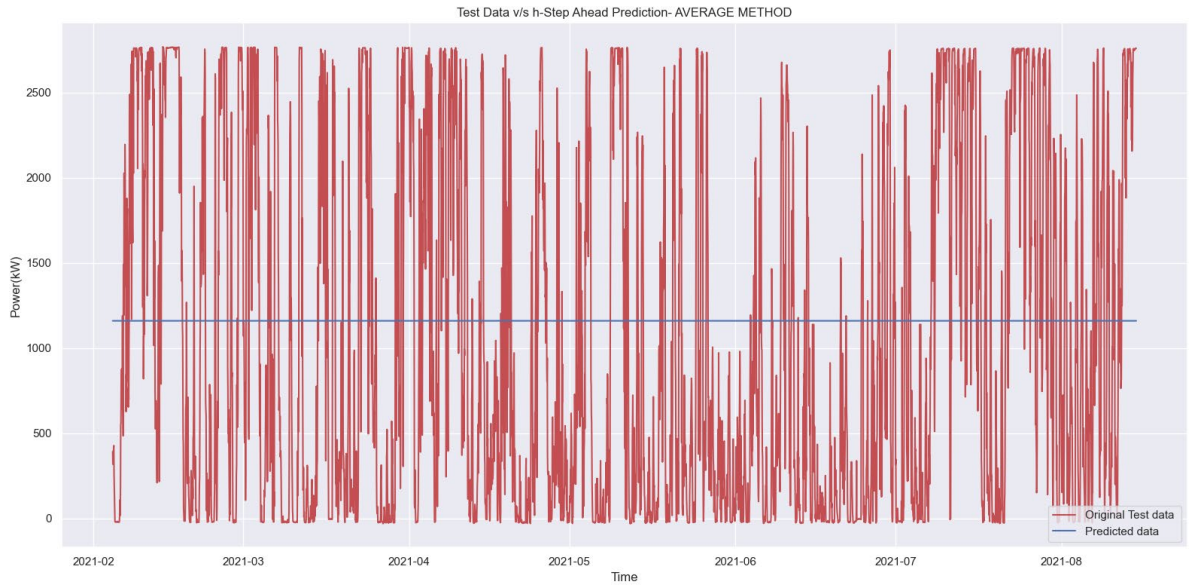


Figure 39

Naive Model/Method

To make the output visually more understandable, we have plotted the h-step prediction output of values against their original values of the test set as follows –

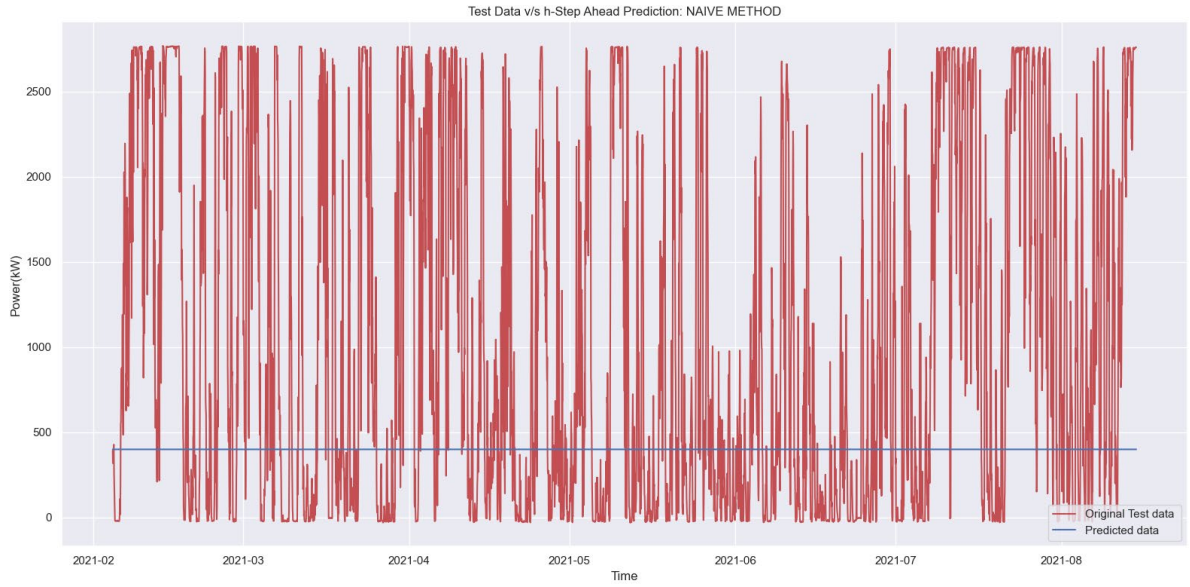


Figure 40

Drift Model/Method

To make the output visually more understandable, we have plotted the h-step prediction output of values against their original values of the test set as follows –

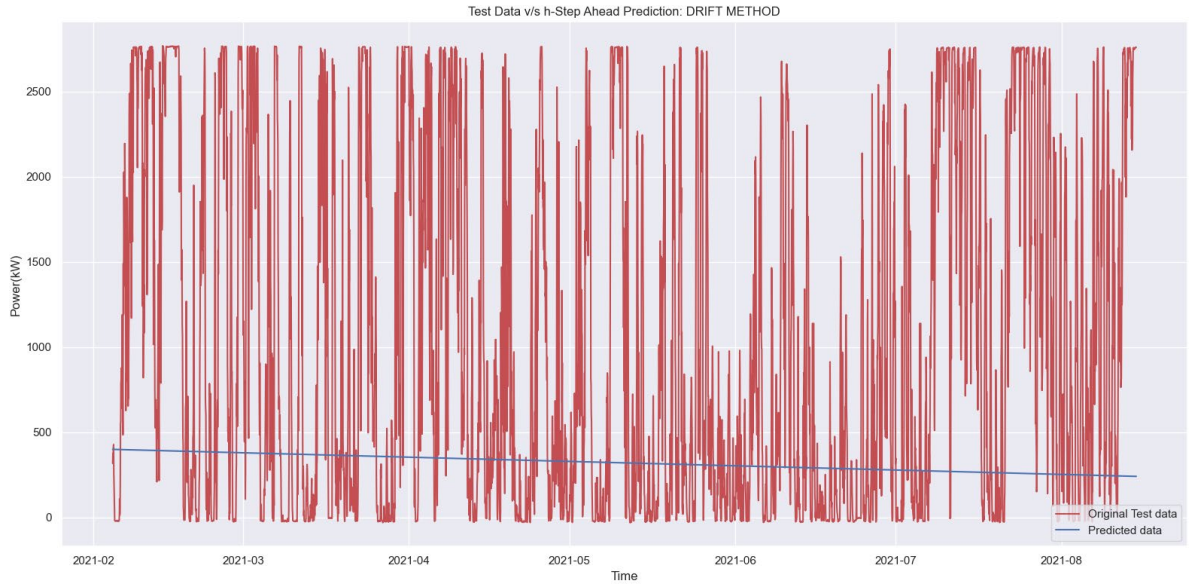


Figure 41

Simple Exponential Smoothing Model/Method

To make the output visually more understandable, we have plotted the h-step prediction output of values against their original values of the test set as follows –

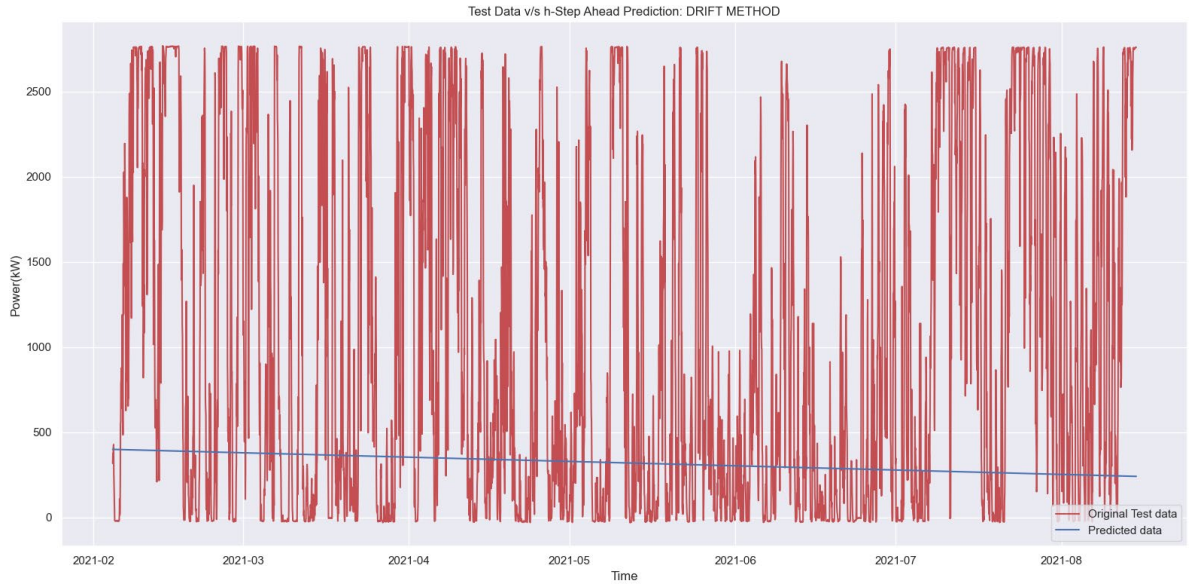


Figure 42

The root mean squared error is the metric we are using to compare values of base models with that of SARIMA for h step predictions they are all listed below –

SARIMA Model H-Step Prediction RMSE Value –

```
The root mean squared error of SARIMA Model is =  2623.7224
```

BASE Models H-Step Prediction RMSE Value –

```
The root mean squared error of AVERAGE Model is =  1048.7447
```

```
The root mean squared error of NAIVE Model is =  1230.3982
```

```
The root mean squared error of DRIFT Model is =  1274.0847
```

```
The root mean squared error of SES Model is =  1290.0362
```

Figure 43

Clearly, we are seeing that the base models in all have performed far better than the SARIMA Model. The performance of SARIMA is bad than the base models. To show it visually, on the next page is the plot of h-step prediction of the SARIMA model against the test set –

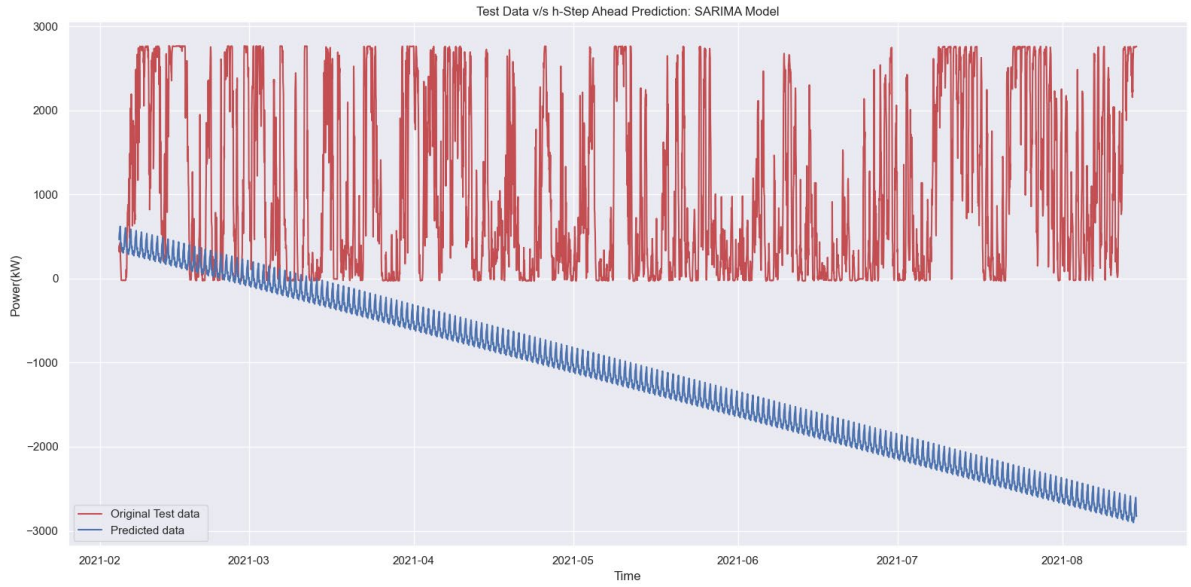


Figure 44

While the base models, due to their simple calculations, have a very stable h-step prediction which in no way is good but it is better than the unrealistic output given by the SARIMA Model.

Multiple Linear Regression

Final Model and h-step predictions

As we had previously designed multiple models for doing the backward stepwise regression, we were left with below model –

```
XtrainScaleddf70 = XtrainScaleddf69.drop(columns=["Blade-1 Actual Value_Angle-B"])
model70 = sm.OLS(y_train,XtrainScaleddf70).fit()
print(model70.summary())
```

Figure 45

```
                            OLS Regression Results
==============================================================================
Dep. Variable:             Power(kW)   R-squared:                       0.005
Model:                           OLS   Adj. R-squared:                  0.005
Method:                Least Squares   F-statistic:                     13.89
Date:               Wed, 10 May 2023   Prob (F-statistic):           4.49e-18
Time:                       19:01:19   Log-Likelihood:            -1.5404e+05
No. Observations:               18374   AIC:                         3.081e+05
Df Residuals:                   18366   BIC:                         3.082e+05
Df Model:                           7
Covariance Type:            nonrobust
=========================================================================================================
                                            coef    std err          t      P>|t|      [0.025      0.975]
---------------------------------------------------------------------------------------------------------
Constant                               1159.2221      7.810    148.419      0.000    1143.913    1174.531
Tower Acceleration Normal                15.6123      7.812      1.999      0.046       0.300      30.924
Converter Control Unit Reactive Power    25.9971      7.812      3.328      0.001      10.684      41.310
Reactive Power                           32.5741      7.812      4.170      0.000      17.263      47.885
Moment D Direction                       25.5306      7.812      3.268      0.001      10.218      40.843
N-set 1                                  51.0277      7.812      6.532      0.000      35.715      66.341
Particle Counter                         22.5981      7.811      2.893      0.004       7.287      37.909
Wind Deviation 1 seconds                 15.9186      7.812      2.038      0.042       0.607      31.231
==============================================================================
Omnibus:                   124771.815   Durbin-Watson:                   0.108
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             2004.067
Skew:                           0.345   Prob(JB):                         0.00
```

Figure 46

This model70 is our final model for backwards stepwise regression and below is our OLS's model70's h-step prediction against test set.

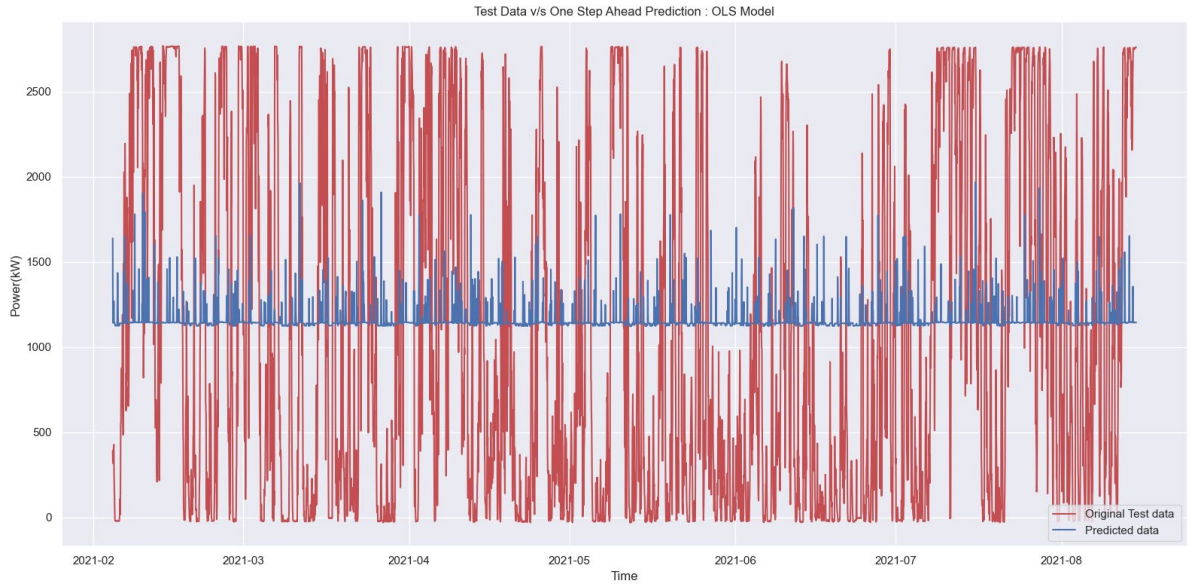


Figure 47

Hypothesis tests analysis: F-test, t-test

T-Test

```
Constant  is statistically significant where p value of the  Constant  is  0.0

Tower Acceleration Normal  is statistically significant where p value of the  Tower Acceleration Normal  is  0.0457

Converter Control Unit Reactive Power  is statistically significant where p value of the  Converter Control Unit Reactive Power  is  0.0009

Reactive Power  is statistically significant where p value of the  Reactive Power  is  0.0

Moment D Direction  is statistically significant where p value of the  Moment D Direction  is  0.0011

N-set 1  is statistically significant where p value of the  N-set 1  is  0.0

Particle Counter  is statistically significant where p value of the  Particle Counter  is  0.0038

Wind Deviation 1 seconds  is statistically significant where p value of the  Wind Deviation 1 seconds  is  0.0416
```

Figure 48

F-test

```
The F Test value for the model 70(Final OLS Model) for
Wind Turbine Power Prediction with features Constant, Tower Acceleration Normal, Converter Control Unit Reactive Power, Reactive Power, Moment D Direction, N-set 1,
 0.0
```

```
Particle Counter, Wind Deviation 1 seconds is-
```

```
0.0
```

Figure 49

AIC, BIC, RMSE, R-squared and Adjusted R-squared

```
The root mean squared error of OLS Model is =  1047.0677
The AIC and BIC scores of the model respectively are =  308094.0636 and  308156.6132
The R-Sqaured and Adjusted R-sqaured values of the model respectively are =  0.0053 and
  0.0049
```

Figure 50

This is where we see the accuracy of the model70 (OLS Regression Model). It may seem OLS has given best RMSE of the bunch but one of the main accuracy metric Adj. R Sqaured = 0.0049 which is better than the what it originally was with all 76 features (0.0040) but it is still not good and it shows that this model is almost incapable of explaining the variance in the dependent variable.

ACF of residuals

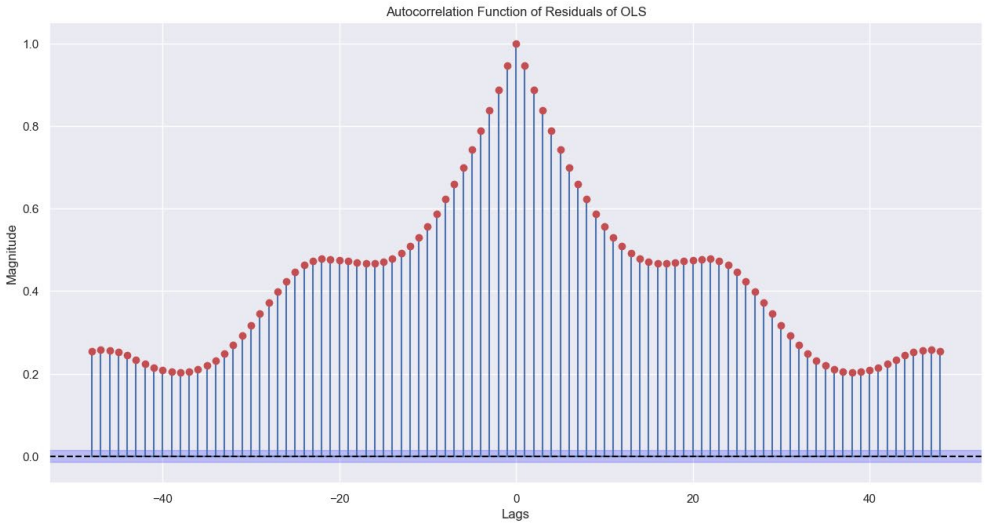


Figure 51

The ACF of residuals above was clearly not a white noise which would have otherwise stated that OLS is the perfect model but it is actually not because acf is not white for residuals.

Q value, Mean and Variance of Residuals

```
The Q value of the OLS Model Residuals is - 245033.2993
Mean of Residuals =  -7.08033754988569e-13 and Variance of residuals = 1120391.2913
```

Figure 52

SARIMA model Order Determination

Preliminary model development procedures and results

This is where we refer to the previous work done in this project. We will now be using that to perform the order estimation using GPAC. Remember we talked about how we will be doing seasonal differencing of order 24 in page 20 last paragraph of this report, well we have performed it now and we will be fitting the differenced data into GPAC as GPAC can only handle the differenced data after all the necessary differencing has been performed.

Following is the output of GPAC from the data that we have after differencing –

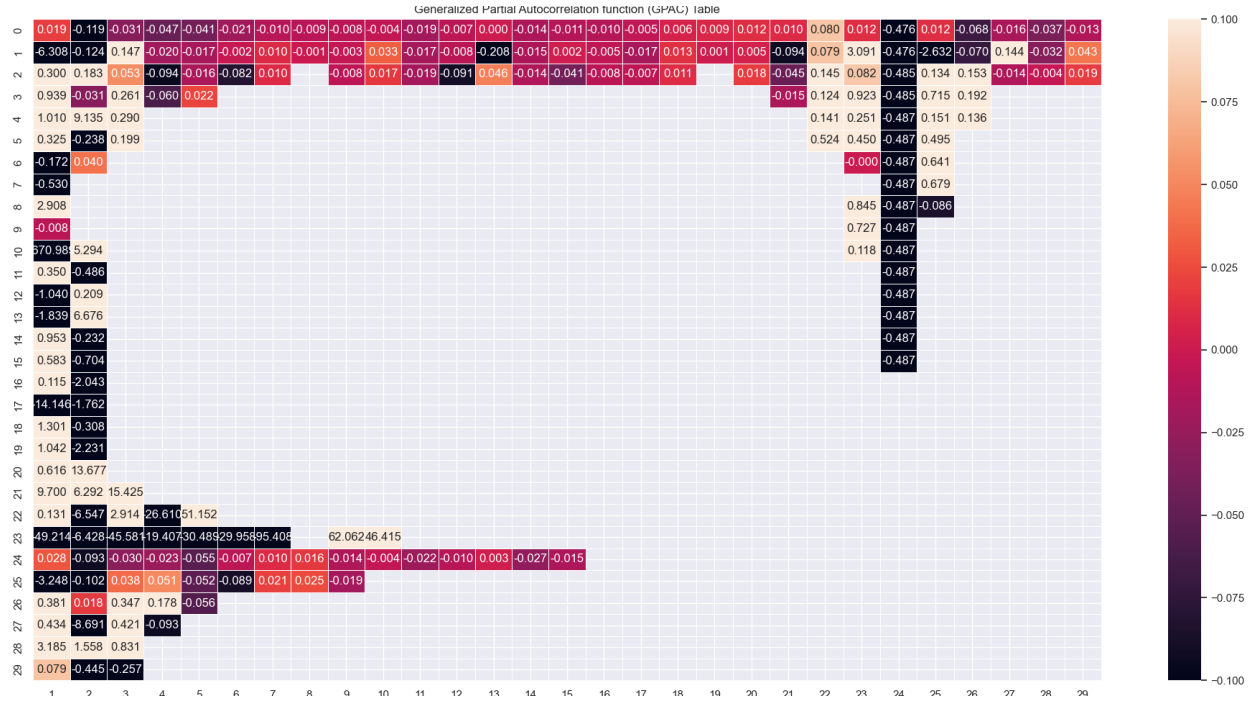
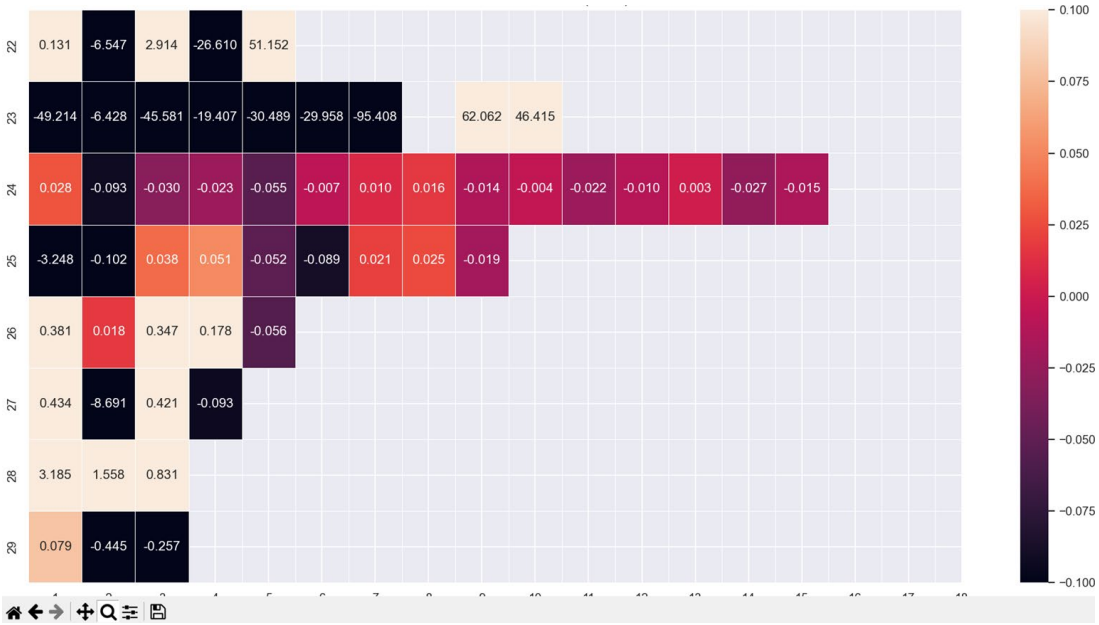


Figure 53



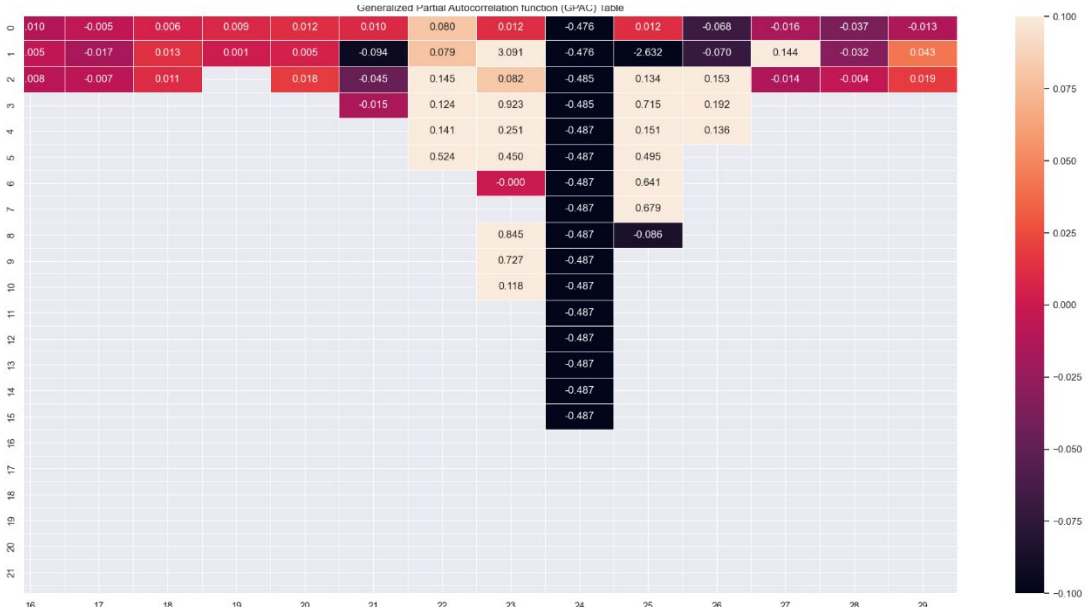


Figure 54

This is a bit cropped version of GPAC as we have zoomed in using interactive plots.

Here we are seeing 2 potential orders – AR(24,0) and MA(0,24)

Discussion of Autocorrelation function and GPAC Table and subsequent plots

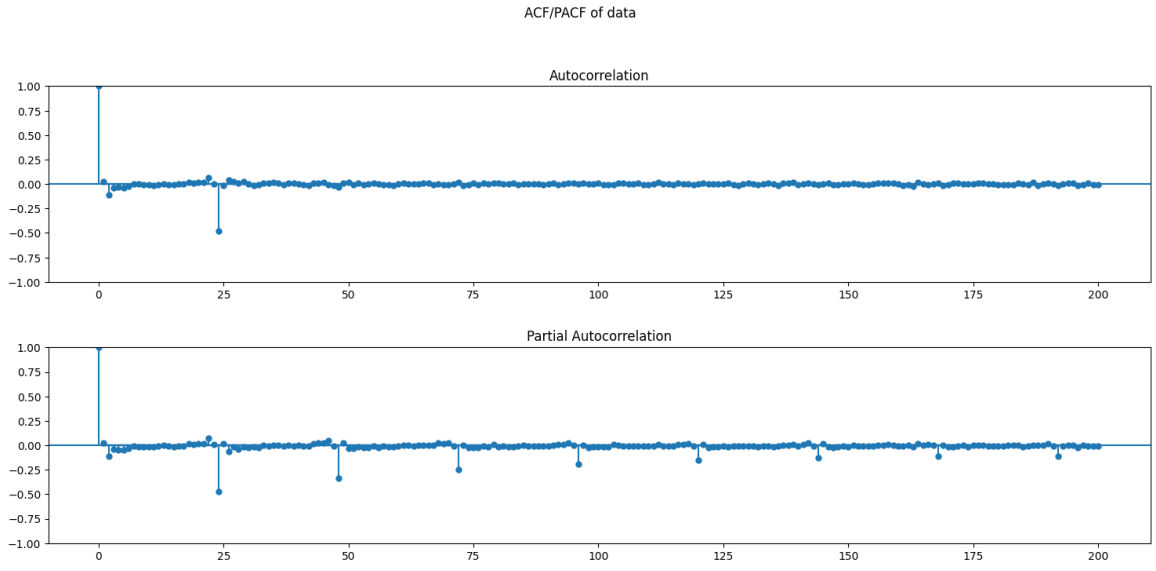


Figure 55

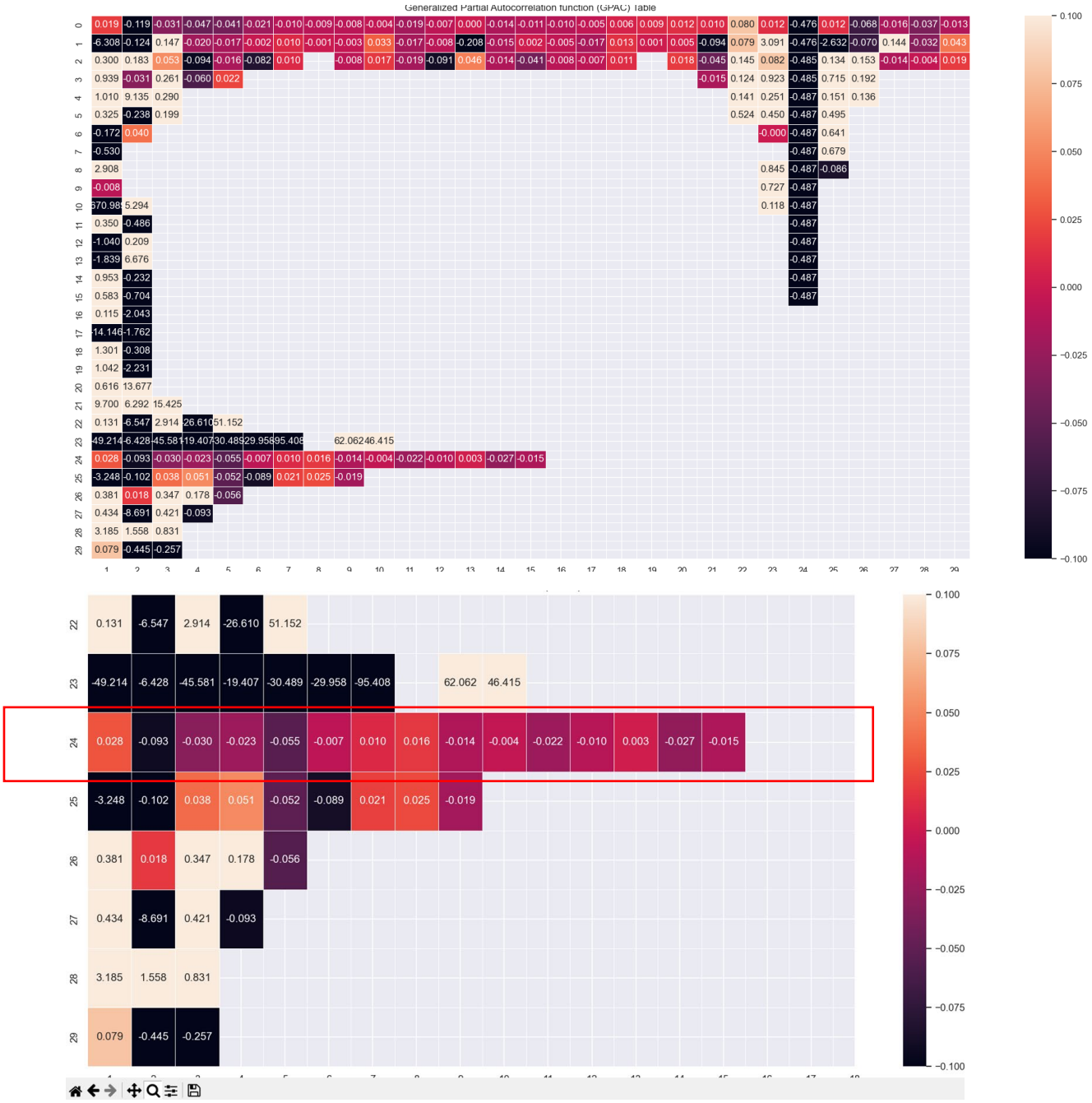

Generalized Partial Autocorrelation function (GPAC) Table
0.100
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
−0.100
0.131 -6.547 2.914 -26.610 51.152
-49.214 -6.428 -45.581 -19.407 -30.489 -29.958 -95.408 62.062 46.415
0.028 -0.093 -0.030 -0.023 -0.055 -0.007 0.010 0.016 -0.014 -0.004 -0.022 -0.010 0.003 -0.027 -0.015
-3.248 -0.102 0.038 0.051 -0.052 -0.089 0.021 0.025 -0.019
0.381 0.018 0.347 0.178 -0.056
0.434 -8.691 0.421 -0.093
3.185 1.558 0.831
0.079 -0.445 -0.257

Figure 56

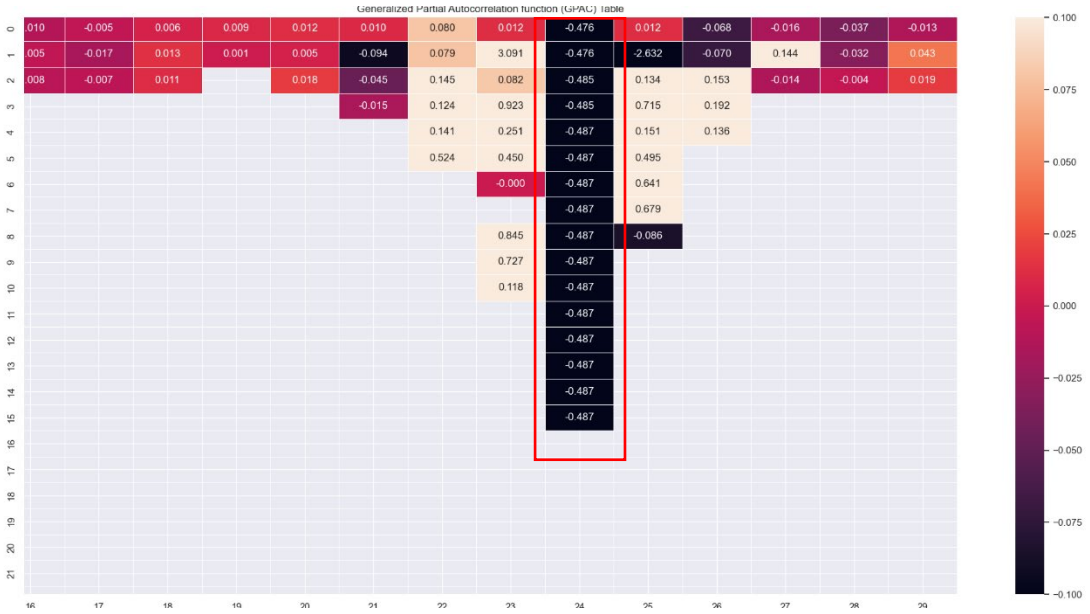


Figure 57

To discuss ACF/PACF Plot and the GPAC table, we are getting an indication that out of 2 orders given by GPAC - AR(24,0) and MA(0,24) and the evident tail-off at seasonal lags in PACF and cutoff after 1 seasonal lag in ACF, it looks like MA(1)xMA(1)$_{24}$ Model. We can conclude so because this model has non-zero ACF at lags k=1 and k=24 for the non-seasonal and seasonal MA models individually. Additionally, it also has nonzero auto correlation at lags 23 and 25 in ACF.

GPAC Table

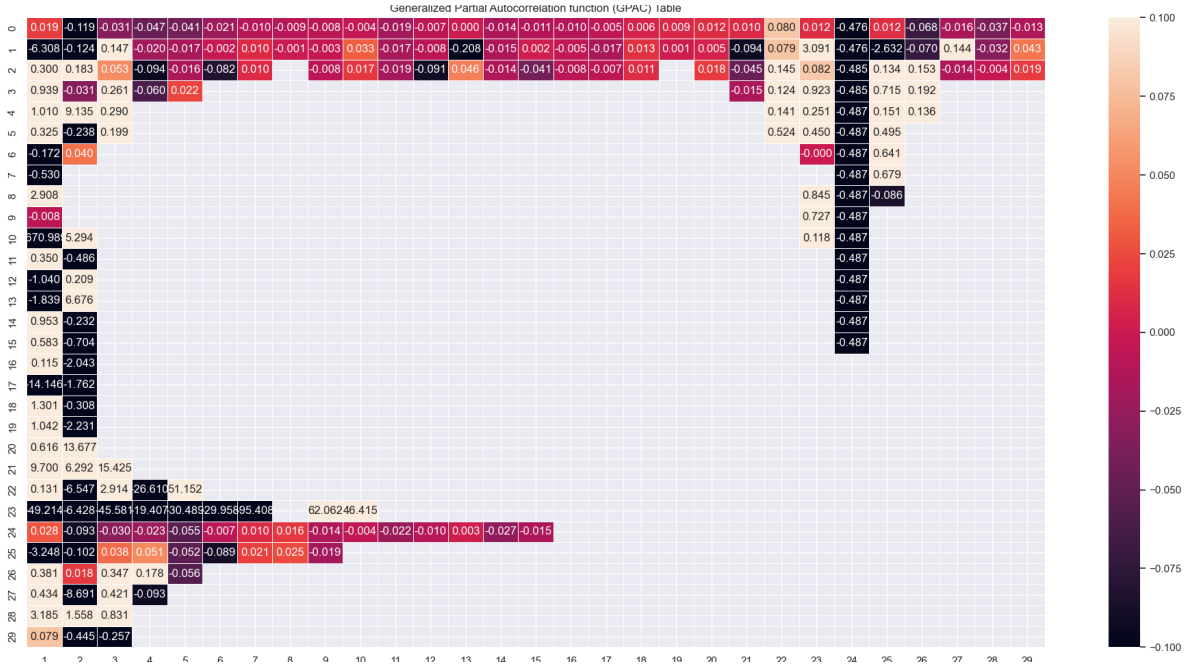


Figure 58

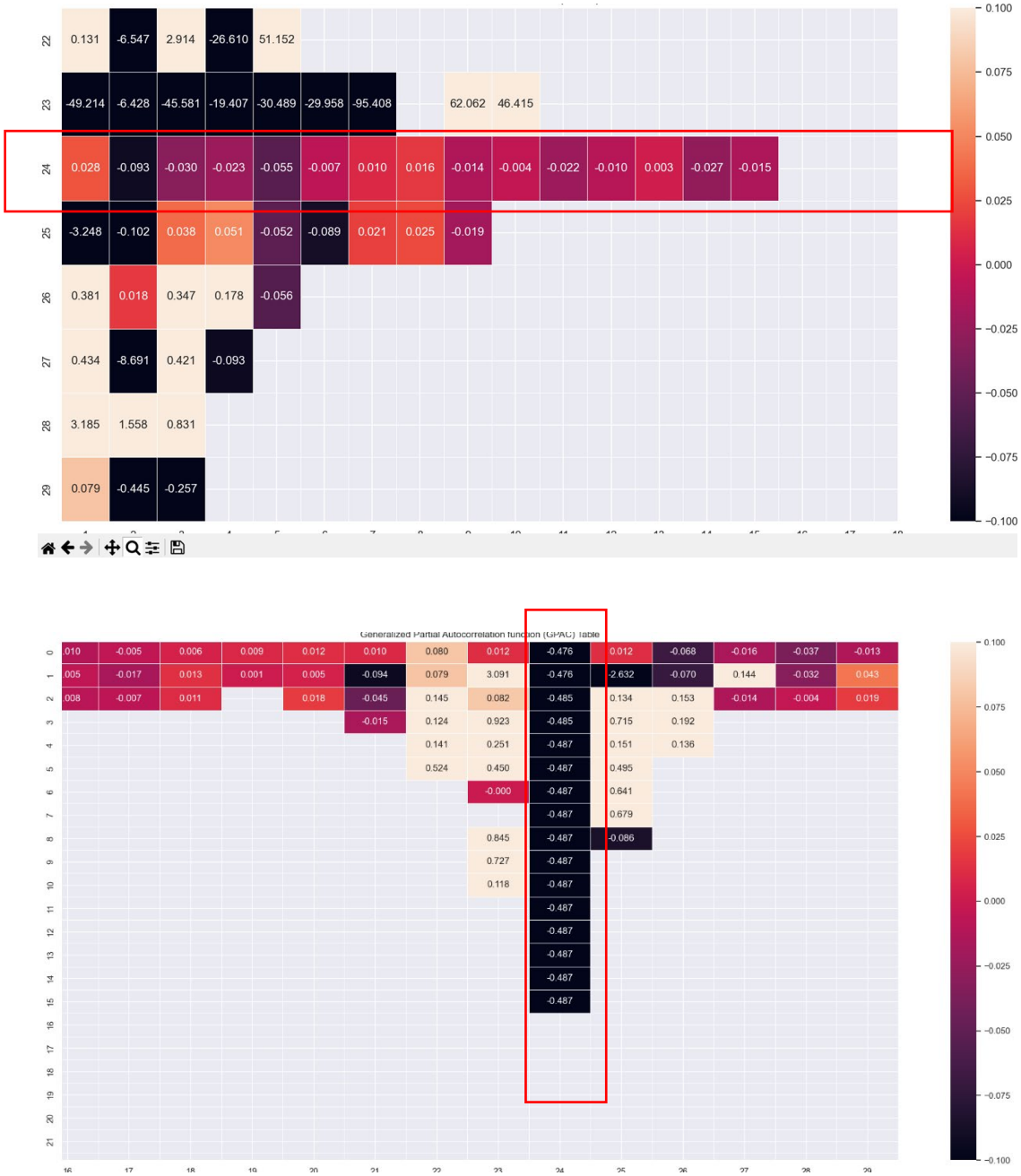

Generalized Partial Autocorrelation function (GPAC) Table

Figure 59

ARMA Model Parameter Estimation

Estimated Parameters , Standard Deviation, and Confidence Intervals for ARMA(0,24)

```
Estimated params are:
 [[ 6.08363644e-04]
 [-1.32311705e-02]
 [-1.30018633e-03]
 [-5.25915364e-03]
 [ 1.90942789e-03]
 [-1.28532291e-02]
 [-4.89716598e-03]
 [-1.59017658e-03]
 [-8.22480725e-03]
 [ 3.14813057e-03]
 [-4.06207708e-03]
 [-8.84437258e-05]
 [ 1.49986697e-03]
 [ 3.61115737e-03]
 [-6.85251039e-03]
 [-5.83525823e-03]
 [-1.56738762e-03]
 [-1.34124312e-02]
 [ 3.60712366e-03]
 [ 4.92406197e-03]
 [ 5.28956414e-03]
 [ 3.49197204e-03]
 [ 2.08966591e-03]
 [-9.40684311e-01]]
```

```
Standard deviation of the Error is : [[335.0604275]]
```

```
The Confidence interval for b1 is - [[-0.0044309]] < b1 < [[0.00564762]]
The Confidence interval for b2 is - [[-0.01827055]] < b2 < [[-0.00819179]]
The Confidence interval for b3 is - [[-0.00634335]] < b3 < [[0.00374298]]
The Confidence interval for b4 is - [[-0.01030196]] < b4 < [[-0.00021635]]
The Confidence interval for b5 is - [[-0.0031335]] < b5 < [[0.00695236]]
The Confidence interval for b6 is - [[-0.01789594]] < b6 < [[-0.00781052]]
The Confidence interval for b7 is - [[-0.00993982]] < b7 < [[0.00014549]]
The Confidence interval for b8 is - [[-0.0066334]] < b8 < [[0.00345305]]
The Confidence interval for b9 is - [[-0.01326733]] < b9 < [[-0.00318229]]
The Confidence interval for b10 is - [[-0.00189473]] < b10 < [[0.00819099]]
The Confidence interval for b11 is - [[-0.00910492]] < b11 < [[0.00098076]]
The Confidence interval for b12 is - [[-0.00513156]] < b12 < [[0.00495468]]
The Confidence interval for b13 is - [[-0.00354329]] < b13 < [[0.00654302]]
The Confidence interval for b14 is - [[-0.00143172]] < b14 < [[0.00865403]]
The Confidence interval for b15 is - [[-0.01189551]] < b15 < [[-0.00180951]]
The Confidence interval for b16 is - [[-0.01087807]] < b16 < [[-0.00079244]]
```

Figure 60

```
The Confidence interval for b17 is - [[-0.00661117]] < b17 < [[0.00347639]]
The Confidence interval for b18 is - [[-0.01845583]] < b18 < [[-0.00836903]]
The Confidence interval for b19 is - [[-0.00143669]] < b19 < [[0.00865094]]
The Confidence interval for b20 is - [[-0.00012]] < b20 < [[0.00996812]]
The Confidence interval for b21 is - [[0.00024558]] < b21 < [[0.01033355]]
The Confidence interval for b22 is - [[-0.00155277]] < b22 < [[0.00853671]]
The Confidence interval for b23 is - [[-0.00295125]] < b23 < [[0.00713058]]
The Confidence interval for b24 is - [[-0.94572541]] < b24 < [[-0.93564321]]
```

Figure 61

Estimated Parameters , Standard Deviation, and Confidence Intervals for ARMA(24,0)

```
Estimated params are:
 [[-0.01602564]
 [ 0.08983799]
 [ 0.03417755]
 [ 0.04026448]
 [ 0.0368752 ]
 [ 0.01726033]
 [ 0.01086365]
 [ 0.01275656]
 [ 0.01532822]
 [ 0.01145155]
```

Figure 62

```
[ 0.01816194]
[ 0.0127605 ]
[ 0.0085079 ]
[ 0.01586475]
[ 0.01438562]
[ 0.01084567]
[ 0.00446011]
[-0.00094173]
[ 0.00722127]
[ 0.00173065]
[ 0.00837619]
[-0.01941758]
[-0.01724067]
[ 0.47702787]]
```

```
Standard Deviation of the Error is : [[397.73558996]]
```

Figure 63

```
The Confidence interval for a1 is - [[-0.02901355]] < a1 < [[-0.00303772]]
The Confidence interval for a2 is - [[0.07685041]] < a2 < [[0.10282556]]
The Confidence interval for a3 is - [[0.02112539]] < a3 < [[0.04722971]]
The Confidence interval for a4 is - [[0.02720315]] < a4 < [[0.05332582]]
The Confidence interval for a5 is - [[0.02380024]] < a5 < [[0.04995016]]
The Confidence interval for a6 is - [[0.00417441]] < a6 < [[0.03034625]]
The Confidence interval for a7 is - [[-0.00222481]] < a7 < [[0.0239521]]
The Confidence interval for a8 is - [[-0.00033286]] < a8 < [[0.02584598]]
The Confidence interval for a9 is - [[0.00223834]] < a9 < [[0.02841811]]
The Confidence interval for a10 is - [[-0.00163861]] < a10 < [[0.0245417]]
The Confidence interval for a11 is - [[0.00507267]] < a11 < [[0.03125122]]
The Confidence interval for a12 is - [[-0.0003318]] < a12 < [[0.0258528]]
The Confidence interval for a13 is - [[-0.00458521]] < a13 < [[0.02160101]]
The Confidence interval for a14 is - [[0.00277366]] < a14 < [[0.02895584]]
The Confidence interval for a15 is - [[0.00129319]] < a15 < [[0.02747805]]
The Confidence interval for a16 is - [[-0.00224676]] < a16 < [[0.0239381]]
The Confidence interval for a17 is - [[-0.00863413]] < a17 < [[0.01755435]]
The Confidence interval for a18 is - [[-0.01403568]] < a18 < [[0.01215223]]
The Confidence interval for a19 is - [[-0.00587066]] < a19 < [[0.0203132]]
The Confidence interval for a20 is - [[-0.01135028]] < a20 < [[0.01481159]]
The Confidence interval for a21 is - [[-0.00469154]] < a21 < [[0.02144392]]
The Confidence interval for a22 is - [[-0.03247624]] < a22 < [[-0.00635893]]
The Confidence interval for a23 is - [[-0.03023608]] < a23 < [[-0.00424526]]
The Confidence interval for a24 is - [[0.46403201]] < a24 < [[0.49002373]]
```

```
mu greater than mumax
Out[97]: 'Mu >Mu_Max'
```

Figure 64

The reason we are showing this for ARMA(24,0) process's parameter estimation part is that it is here to show that this order is not the right order as the output also shows that mu>muMax so we will not consider this order for SARIMA modelling.

Model Development and Diagnostic Analysis

In this part after, estimating orders using GPAC and generating parameters, We will build a SARIMA model for our data as we had previously realized while plotting ACF PACF along with GPAC on page 42 that our model might just be a seasonal MA model of following order - $MA(1)_{24}$.

We have now built following SARIMAX function based on the model estimated from ACF,PACF and GPAC Table –

```
SARIMAmodel = sm.tsa.SARIMAX(y_train, order=(0,1,0),seasonal_order=(0,1,1,24), trend='n').fit()
```

And following was the output given by our model –

```
                                SARIMAX Results
==========================================================================================
Dep. Variable:                          Power(kW)   No. Observations:                18374
Model:             SARIMAX(0, 1, 0)x(0, 1, [1], 24)   Log Likelihood           -132492.917
Date:                              Wed, 10 May 2023   AIC                         264989.834
Time:                                      19:42:30   BIC                         265005.469
Sample:                                           0   HQIC                        264994.971
                                              - 18374
Covariance Type:                                opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ma.S.L24      -0.9802      0.001   -673.651      0.000      -0.983      -0.977
sigma2      1.093e+05    582.170    187.807      0.000    1.08e+05     1.1e+05
===================================================================================
Ljung-Box (L1) (Q):                  29.69   Jarque-Bera (JB):             25125.48
Prob(Q):                              0.00   Prob(JB):                         0.00
Heteroskedasticity (H):               0.88   Skew:                            -0.04
Prob(H) (two-sided):                  0.00   Kurtosis:                         8.73
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

Figure 65

This is the sole SARIMA model that we will be building because the parameter estimations for the other model ARMA(24,0) is not significant and correct.

Diagnostic tests (confidence intervals, zero/pole cancellation, chi-square test, Residual Analysis)

For our SARIMA model with we see that the output above only shows one seasonal parameter of order 24 but our parameter estimation showed that there are 24 parameters

Another thing worth pointing out is that for confidence intervals for estimated parameters of the model ARMA(0,24) has zero pole cancellation in following coefficients as shown below in the image as well –

b1,b3,b5,b7,b8,b10,b11,b12,b13,b14, b17, b19,b20,b22,b23

```
The Confidence interval for b1 is - [[-0.0044309]] < b1 < [[0.00564762]]
The Confidence interval for b2 is - [[-0.01827055]] < b2 < [[-0.00819179]]
The Confidence interval for b3 is - [[-0.00634335]] < b3 < [[0.00374298]]
The Confidence interval for b4 is - [[-0.01030196]] < b4 < [[-0.00021635]]
The Confidence interval for b5 is - [[-0.0031335]] < b5 < [[0.00695236]]
The Confidence interval for b6 is - [[-0.01789594]] < b6 < [[-0.00781052]]
The Confidence interval for b7 is - [[-0.00993982]] < b7 < [[0.00014549]]
The Confidence interval for b8 is - [[-0.0066334]] < b8 < [[0.00345305]]
The Confidence interval for b9 is - [[-0.01326733]] < b9 < [[-0.00318229]]
The Confidence interval for b10 is - [[-0.00189473]] < b10 < [[0.00819099]]
The Confidence interval for b11 is - [[-0.00910492]] < b11 < [[0.00098076]]
The Confidence interval for b12 is - [[-0.00513156]] < b12 < [[0.00495468]]
The Confidence interval for b13 is - [[-0.00354329]] < b13 < [[0.00654302]]
The Confidence interval for b14 is - [[-0.00143172]] < b14 < [[0.00865403]]
The Confidence interval for b15 is - [[-0.01189551]] < b15 < [[-0.00180951]]
The Confidence interval for b16 is - [[-0.01087807]] < b16 < [[-0.00079244]]
```

```
The Confidence interval for b17 is - [[-0.00661117]] < b17 < [[0.00347639]]
The Confidence interval for b18 is - [[-0.01845583]] < b18 < [[-0.00836903]]
The Confidence interval for b19 is - [[-0.00143669]] < b19 < [[0.00865094]]
The Confidence interval for b20 is - [[-0.00012]] < b20 < [[0.00996812]]
The Confidence interval for b21 is - [[0.00024558]] < b21 < [[0.01033355]]
The Confidence interval for b22 is - [[-0.00155277]] < b22 < [[0.00853671]]
The Confidence interval for b23 is - [[-0.00295125]] < b23 < [[0.00713058]]
The Confidence interval for b24 is - [[-0.94572541]] < b24 < [[-0.93564321]]
```

Figure 66

```
Chi critical: 72.44330737654823
Q Value: 813.1656963144826
Alfa value for 99% accuracy: 0.01
The residual is NOT white
```

We will see the residuals later in this section below and they are not white.

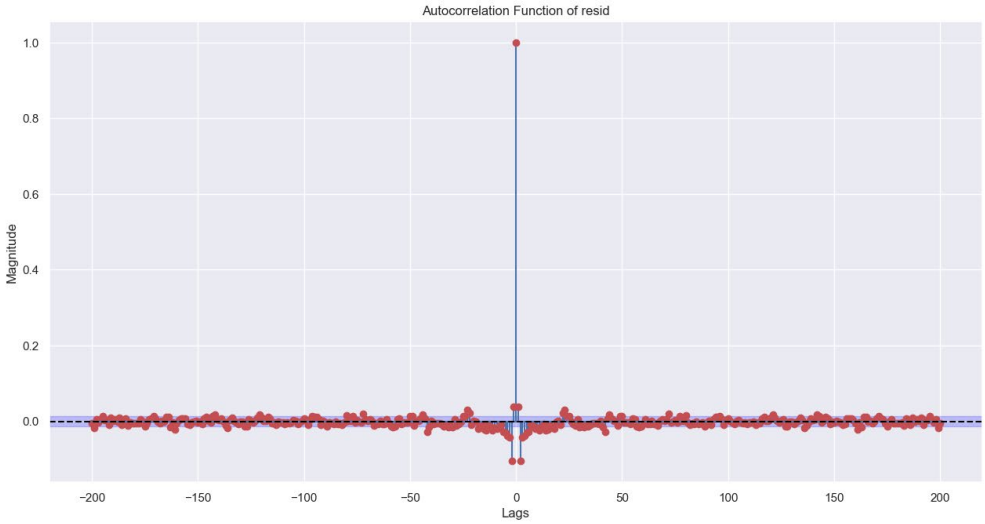


Figure 67

Also, here we see that ACF of residuals is not white so we fit the data back to GPAC and we get even more uninterpretable results.

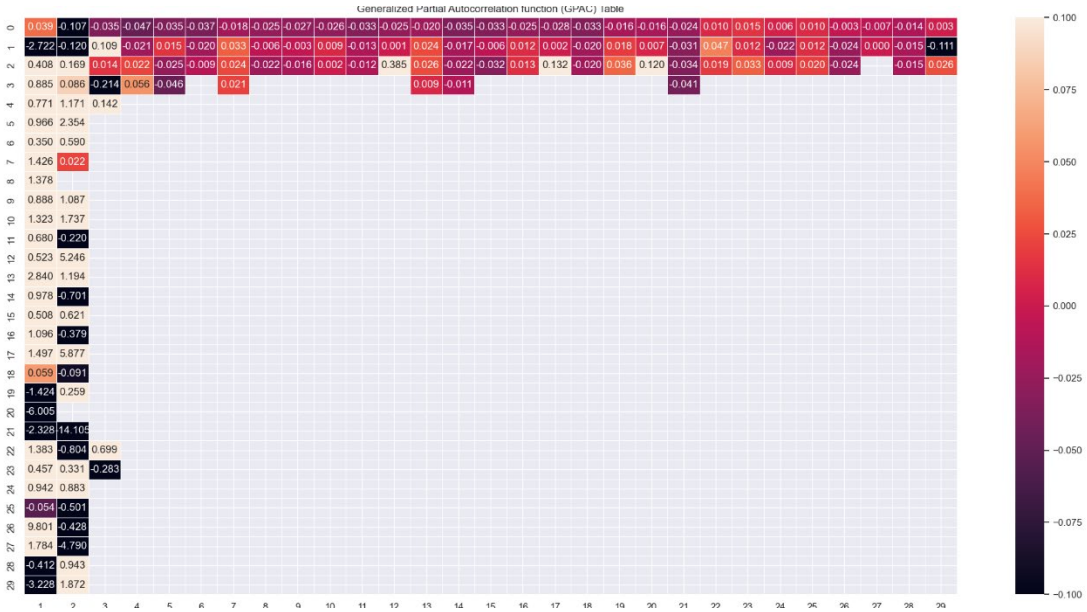


Figure 68

Estimated Variance and Estimated covariance of the Estimated Parameters

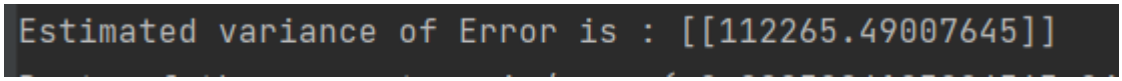


Figure 69

The Covariance Matrix is:

[[6.34853592e-06 1.61324480e-07 -5.59568997e-07 1.87646621e-07
 -4.69503005e-08 2.64933885e-07 -1.42875303e-06 -3.83902875e-07
 -4.04630320e-07 -7.92371568e-07 3.69641913e-07 -1.28941494e-07
  1.39713465e-08 -1.10389814e-07 3.69692036e-07 -7.90002222e-07
 -4.18510185e-07 -3.72642875e-07 -1.43145104e-06 2.71871833e-07
 -9.96478258e-09 2.12242923e-07 -5.03469765e-07 1.67346079e-07]
[1.61324480e-07 6.34882676e-06 1.60003561e-07 -5.60873567e-07
  1.86616698e-07 -4.72181430e-08 2.66849024e-07 -1.42900289e-06
 -3.82868205e-07 -4.03947866e-07 -7.92807166e-07 3.69594573e-07
 -1.28756568e-07 1.45741800e-08 -1.11086801e-07 3.70878744e-07
 -7.89426763e-07 -4.18347744e-07 -3.71924160e-07 -1.43140148e-06
  2.73070092e-07 -9.92353838e-09 2.13950436e-07 -5.03330498e-07]
[-5.59568997e-07 1.60003561e-07 6.35838097e-06 1.60582126e-07
 -5.57584474e-07 1.84848077e-07 -3.49584369e-08 2.71331637e-07
 -1.42673030e-06 -3.75550520e-07 -4.07136955e-07 -7.90476754e-07
  3.69362482e-07 -1.29295995e-07 1.15213643e-08 -1.04535247e-07
  3.75484295e-07 -7.86754012e-07 -4.05202773e-07 -3.74953584e-07
 -1.43424397e-06 2.69446698e-07 -9.92069542e-09 2.11927939e-07]
[1.87646621e-07 -5.60873567e-07 1.60582126e-07 6.35746356e-06
  1.59627227e-07 -5.59027318e-07 1.89673567e-07 -3.34933521e-08
  2.73052578e-07 -1.42387666e-06 -3.76921382e-07 -4.07500309e-07
 -7.90575430e-07 3.70418457e-07 -1.30272450e-07 1.42155309e-08
 -1.03943891e-07 3.77666124e-07 -7.80707556e-07 -4.06247067e-07
 -3.73996751e-07 -1.43383752e-06 2.73630525e-07 -1.04718243e-08]
[-4.69503005e-08 1.86616698e-07 -5.57584474e-07 1.59627227e-07
  6.35779263e-06 1.58095719e-07 -5.50922483e-07 1.91990895e-07
 -3.12736543e-08 2.77705506e-07 -1.42598529e-06 -3.76189123e-07

-4.07607593e-07 -7.89984394e-07 3.68281056e-07 -1.25835838e-07
 1.65575142e-08 -1.01765472e-07 3.86037807e-07 -7.82309020e-07
-4.06215990e-07 -3.75199086e-07 -1.43095709e-06 2.72605299e-07]
[2.64933885e-07 -4.72181430e-08 1.84848077e-07 -5.59027318e-07
 1.58095719e-07 6.35723877e-06 1.60054184e-07 -5.51044938e-07
 1.93152855e-07 -3.04617608e-08 2.77107883e-07 -1.42640585e-06
-3.76154129e-07 -4.06663263e-07 -7.90506550e-07 3.69509111e-07
-1.25767670e-07 1.73178543e-08 -1.00286716e-07 3.86015876e-07
-7.80684252e-07 -4.05582553e-07 -3.72173591e-07 -1.43090826e-06]
[-1.42875303e-06 2.66849024e-07 -3.49584369e-08 1.89673567e-07
-5.50922483e-07 1.60054184e-07 6.35709148e-06 1.62062176e-07
-5.54238492e-07 1.93862650e-07 -3.02543781e-08 2.81248463e-07
-1.42659145e-06 -3.80132032e-07 -4.06951658e-07 -7.90993562e-07
 3.73182794e-07 -1.28680687e-07 1.65696982e-08 -1.01607050e-07
 3.78264143e-07 -7.86645307e-07 -4.19107523e-07 -3.73397791e-07]
[-3.83902875e-07 -1.42900289e-06 2.71331637e-07 -3.34933521e-08
 1.91990895e-07 -5.51044938e-07 1.62062176e-07 6.35852974e-06
 1.61673900e-07 -5.52750173e-07 1.93310574e-07 -2.92043037e-08
 2.81097566e-07 -1.42744565e-06 -3.80575178e-07 -4.05875367e-07
-7.89743540e-07 3.73451834e-07 -1.25882728e-07 1.56205409e-08
-1.03874904e-07 3.76631668e-07 -7.89065384e-07 -4.19846093e-07]
[-4.04630320e-07 -3.82868205e-07 -1.42673030e-06 2.73052578e-07
-3.12736543e-08 1.93152855e-07 -5.54238492e-07 1.61673900e-07
 6.35675071e-06 1.60071609e-07 -5.51819309e-07 1.94319122e-07
-2.91797800e-08 2.79662282e-07 -1.42678765e-06 -3.82421996e-07
-4.05774560e-07 -7.91492915e-07 3.69804107e-07 -1.25595148e-07
 1.33787278e-08 -1.05210608e-07 3.71456924e-07 -7.88996416e-07]
[-7.92371568e-07 -4.03947866e-07 -3.75550520e-07 -1.42387666e-06
 2.77705506e-07 -3.04617608e-08 1.93862650e-07 -5.52750173e-07
 1.60071609e-07 6.35760835e-06 1.59933124e-07 -5.49625398e-07
 1.94197749e-07 -3.13380740e-08 2.79436415e-07 -1.42662803e-06

-3.80299377e-07 -4.06864760e-07 -7.90666489e-07 3.68890039e-07
-1.30025833e-07 1.01231256e-08 -1.12245457e-07 3.70597541e-07]
[3.69641913e-07 -7.92807166e-07 -4.07136955e-07 -3.76921382e-07
-1.42598529e-06 2.77107883e-07 -3.02543781e-08 1.93310574e-07
-5.51819309e-07 1.59933124e-07 6.35755507e-06 1.58926142e-07
-5.49545802e-07 1.95263401e-07 -3.13482379e-08 2.79654753e-07
-1.42751544e-06 -3.79631615e-07 -4.06695063e-07 -7.90309320e-07
 3.70867564e-07 -1.28454158e-07 1.36429139e-08 -1.11871353e-07]
[-1.28941494e-07 3.69594573e-07 -7.90476754e-07 -4.07500309e-07
-3.76189123e-07 -1.42640585e-06 2.81248463e-07 -2.92043037e-08
 1.94319122e-07 -5.49625398e-07 1.58926142e-07 6.35826561e-06
 1.58911255e-07 -5.49597105e-07 1.94079318e-07 -2.90760050e-08
 2.81333703e-07 -1.42683259e-06 -3.76045166e-07 -4.07452744e-07
-7.90790819e-07 3.69663333e-07 -1.28378931e-07 1.31719697e-08]
[1.39713465e-08 -1.28756568e-07 3.69362482e-07 -7.90575430e-07
-4.07607593e-07 -3.76154129e-07 -1.42659145e-06 2.81097566e-07
-2.91797800e-08 1.94197749e-07 -5.49545802e-07 1.58911255e-07
 6.35835685e-06 1.58926080e-07 -5.49572688e-07 1.94002564e-07
-2.90745294e-08 2.81228022e-07 -1.42717319e-06 -3.75986204e-07
-4.07359983e-07 -7.90758792e-07 3.69665544e-07 -1.28286905e-07]
[-1.10389814e-07 1.45741800e-08 -1.29295995e-07 3.70418457e-07
-7.89984394e-07 -4.06663263e-07 -3.80132032e-07 -1.42744565e-06
 2.79662282e-07 -3.13380740e-08 1.95263401e-07 -5.49597105e-07
 1.58926080e-07 6.35764527e-06 1.59977790e-07 -5.51893824e-07
 1.93154700e-07 -3.03643865e-08 2.77268396e-07 -1.42652097e-06
-3.76714988e-07 -4.07256220e-07 -7.93385420e-07 3.70002780e-07]
[3.69692036e-07 -1.11086801e-07 1.15213643e-08 -1.30272450e-07
 3.68281056e-07 -7.90506550e-07 -4.06951658e-07 -3.80575178e-07
-1.42678765e-06 2.79436415e-07 -3.13482379e-08 1.94079318e-07
-5.49572688e-07 1.59977790e-07 6.35795735e-06 1.59884120e-07
-5.53078693e-07 1.94029544e-07 -2.99634447e-08 2.77459920e-07

-1.42463492e-06 -3.74992595e-07 -4.03625378e-07 -7.93083999e-07]
[-7.90002222e-07 3.70878744e-07 -1.04535247e-07 1.42155309e-08
-1.25835838e-07 3.69509111e-07 -7.90993562e-07 -4.05875367e-07
-3.82421996e-07 -1.42662803e-06 2.79654753e-07 -2.90760050e-08
1.94002564e-07 -5.51893824e-07 1.59884120e-07 6.35749448e-06
1.61909845e-07 -5.54879501e-07 1.93156850e-07 -3.05668523e-08
2.73222527e-07 -1.42796133e-06 -3.82691291e-07 -4.04246854e-07]
[-4.18510185e-07 -7.89426763e-07 3.75484295e-07 -1.03943891e-07
1.65575142e-08 -1.25767670e-07 3.73182794e-07 -7.89743540e-07
-4.05774560e-07 -3.80299377e-07 -1.42751544e-06 2.81333703e-07
-2.90745294e-08 1.93154700e-07 -5.53078693e-07 1.61909845e-07
6.35993513e-06 1.61642206e-07 -5.52116035e-07 1.92216858e-07
-3.25883396e-08 2.70677619e-07 -1.43081863e-06 -3.83374323e-07]
[-3.72642875e-07 -4.18347744e-07 -7.86754012e-07 3.77666124e-07
-1.01765472e-07 1.73178543e-08 -1.28680687e-07 3.73451834e-07
-7.91492915e-07 -4.06864760e-07 -3.79631615e-07 -1.42683259e-06
2.81228022e-07 -3.03643865e-08 1.94029544e-07 -5.54879501e-07
1.61642206e-07 6.35897275e-06 1.59773248e-07 -5.52192636e-07
1.89942982e-07 -3.34182823e-08 2.66589336e-07 -1.43102019e-06]
[-1.43145104e-06 -3.71924160e-07 -4.05202773e-07 -7.80707556e-07
3.86037807e-07 -1.00286716e-07 1.65696982e-08 -1.25882728e-07
3.69804107e-07 -7.90666489e-07 -4.06695063e-07 -3.76045166e-07
-1.42717319e-06 2.77268396e-07 -2.99634447e-08 1.93156850e-07
-5.52116035e-07 1.59773248e-07 6.36001136e-06 1.58006820e-07
-5.60517909e-07 1.84651990e-07 -4.61691524e-08 2.65046612e-07]
[2.71871833e-07 -1.43140148e-06 -3.74953584e-07 -4.06247067e-07
-7.82309020e-07 3.86015876e-07 -1.01607050e-07 1.56205409e-08
-1.25595148e-07 3.68890039e-07 -7.90309320e-07 -4.07452744e-07
-3.75986204e-07 -1.42652097e-06 2.77459920e-07 -3.05668523e-08
1.92216858e-07 -5.52192636e-07 1.58006820e-07 6.36063989e-06
1.59568147e-07 -5.59303647e-07 1.86500131e-07 -4.57143417e-08]

```
[-9.96478258e-09  2.73070092e-07 -1.43424397e-06 -3.73996751e-07
 -4.06215990e-07 -7.80684252e-07  3.78264143e-07 -1.03874904e-07
  1.33787278e-08 -1.30025833e-07  3.70867564e-07 -7.90790819e-07
 -4.07359983e-07 -3.76714988e-07 -1.42463492e-06  2.73222527e-07
 -3.25883396e-08  1.89942982e-07 -5.60517909e-07  1.59568147e-07
  6.36043941e-06  1.60374651e-07 -5.62777990e-07  1.87415079e-07]
[ 2.12242923e-07 -9.92353838e-09  2.69446698e-07 -1.43383752e-06
 -3.75199086e-07 -4.05582553e-07 -7.86645307e-07  3.76631668e-07
 -1.05210608e-07  1.01231256e-08 -1.28454158e-07  3.69663333e-07
 -7.90758792e-07 -4.07256220e-07 -3.74992595e-07 -1.42796133e-06
  2.70677619e-07 -3.34182823e-08  1.84651990e-07 -5.59303647e-07
  1.60374651e-07  6.36235568e-06  1.60410336e-07 -5.61996737e-07]
[-5.03469765e-07  2.13950436e-07 -9.92069542e-09  2.73630525e-07
 -1.43095709e-06 -3.72173591e-07 -4.19107523e-07 -7.89065384e-07
  3.71456924e-07 -1.12245457e-07  1.36429139e-08 -1.28378931e-07
  3.69665544e-07 -7.93385420e-07 -4.03625378e-07 -3.82691291e-07
 -1.43081863e-06  2.66589336e-07 -4.61691524e-08  1.86500131e-07
 -5.62777990e-07  1.60410336e-07  6.35269840e-06  1.61298907e-07]
[ 1.67346079e-07 -5.03330498e-07  2.11927939e-07 -1.04718243e-08
  2.72605299e-07 -1.43090826e-06 -3.73397791e-07 -4.19846093e-07
 -7.88996416e-07  3.70597541e-07 -1.11871353e-07  1.31719697e-08
 -1.28286905e-07  3.70002780e-07 -7.93083999e-07 -4.04246854e-07
 -3.83374323e-07 -1.43102019e-06  2.65046612e-07 -4.57143417e-08
  1.87415079e-07 -5.61996737e-07  1.61298907e-07  6.35317270e-06]]
```

Estimator : biased or unbiased

The derived model is a biased estimator as we can see below that the variance of the residual and forecast errors do not match and are not even close to each other.

Variance of the residual errors versus the variance of the forecast errors

```
variance of residual error is = 110173.2541
variance of forecast error is = 1860258.6518
```

Figure 70

Long Short-Term Memory Deep Learning Model(LSTM)

This is a Recurrent Neural network capable of long-term reference to the learnings in the past or early layers. Can remember in long terms due to the gate mechanism very similar to gates in electrical engineering. Whenever past reference is requested or needed, the gates to that point in the past(older layers) sort of connects to fetch the information from there and utilize it in further learning and/or output generation. This worked greatly for us because we have a lot of past information that we want to refer to and the linear models cannot fathom that. Like for some power output pattern that happened for a period in the past can be captured by LSTM to make current predictions.

```
model = Sequential()
model.add(LSTM(64, activation="relu", input_shape=(train_X.shape[1], train_X.shape[2]), return_sequences=True))
model.add(LSTM(50, return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
model.summary()
history = model.fit(train_X, train_y,  batch_size=32, validation_split=.1, epochs=30, verbose=1)
```

Figure 71

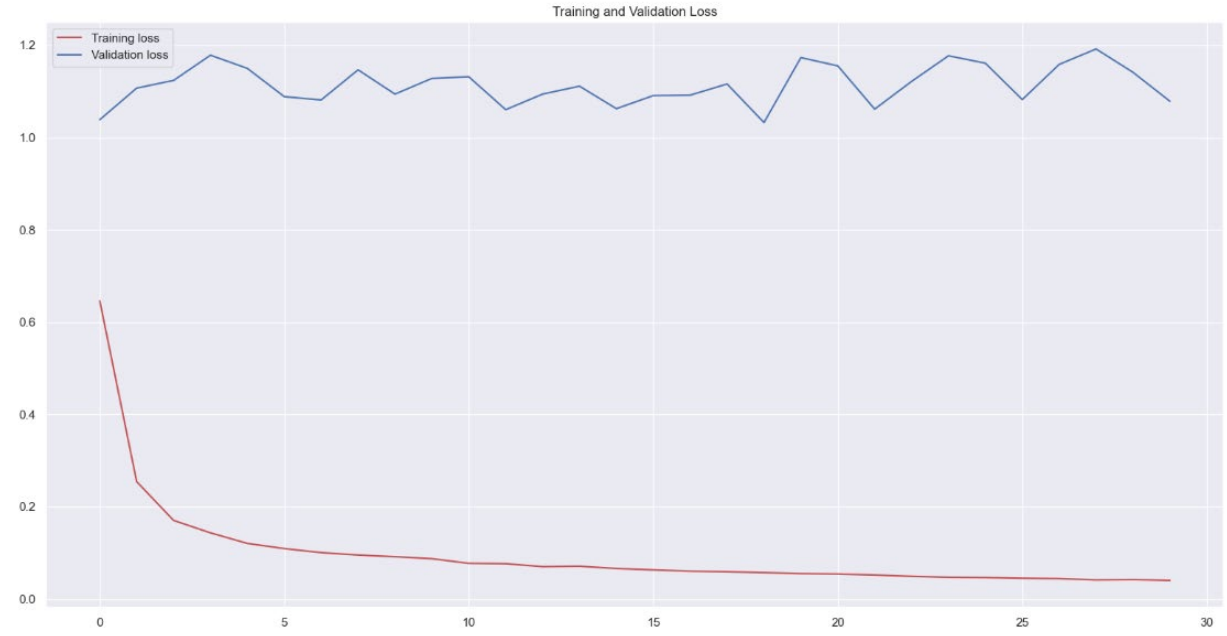
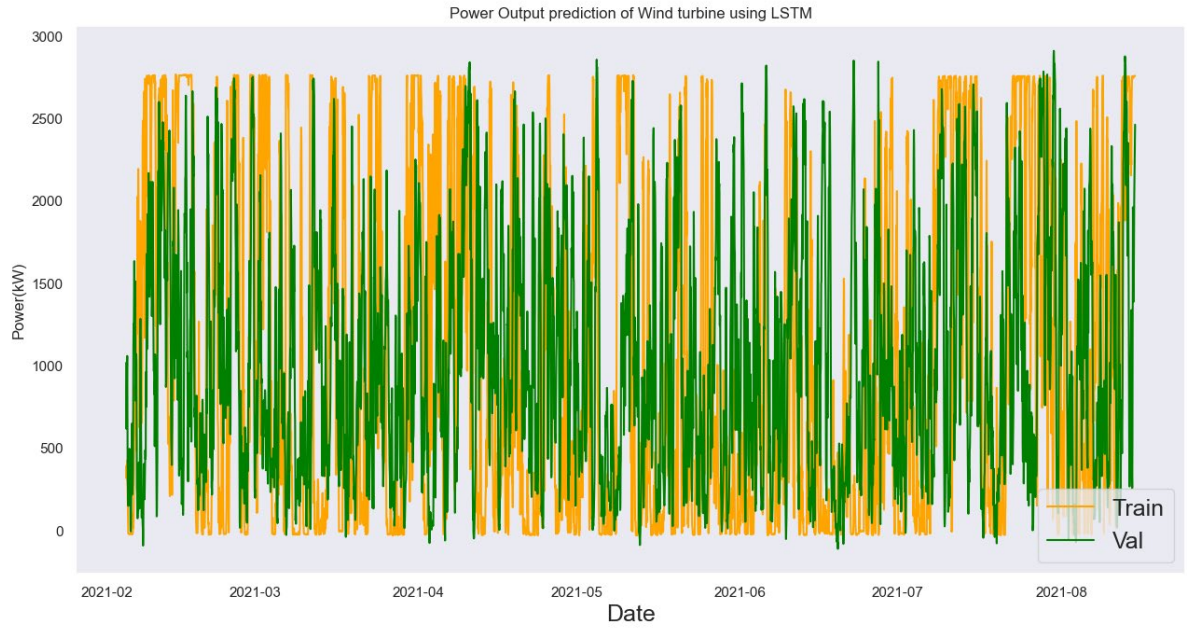


Figure 72




```
The root mean squared error of long Short Term Memory Neural Network is =
  1189.0
The accuracy of long Short Term Memory Neural Network is =
  57.5204
```

Figure 73

Additional Model (Gradient Boosting Regressor)

At this We used this model to additionally test the hypothesis of *"whether the reason for bad predictions using time series models is because of the non-linearity of data"* and this model's result are best of the results hence it helps us prove that hypothesis as shown by the results -

```
Gradient Boosting Regression Model Test Score/R2 value is =  0.9436
The Adjusted R2 value for Gradient Boosting Regression is =  0.9436
```

```
The RMSE for Gradient Boosting Regression Model is = 247.75681113242075
```

Figure 74

In the final model picking section below we will be covering the reason and detailed building and proving of the above hypothesis after covering the analysis of requested models in this report.

Final Model Selection

For the final model selection, we have come to a conclusion that **our final model is going to be Gradient Boosting Regressor**.

Firstly, let's compare the accuracy metrics(Root Mean Square Error) for all the models developed so far –

1. Holt Winter Method

```
Root Mean square error for Holt-Winter method is 1308.39
```

2. Base Models

```
The root mean squared error of AVERAGE Model is =  1048.7447
```

```
The root mean squared error of NAIVE Model is =  1230.3982
```

```
The root mean squared error of DRIFT Model is =  1274.0847
```

```
The root mean squared error of SES Model is =  1290.0362
```

3. OLS Model

```
The root mean squared error of OLS Model is =  1047.0677
```

4. SARIMA Model

```
The root mean squared error of SARIMA Model is =  2623.7224
```

5. LSTM

```
The root mean squared error of long Short Term Memory Neural Network is =
  1189.0
```

6. Gradient Boosting Regressor

```
The RMSE for Gradient Boosting Regression Model is = 247.75681113242075
```

Figure 75

On a pure merit basis of Root mean squared Error (R.M.S.E.) for all the models we have developed we see that the R.M.S.E. for Gradient boosting Regressor algorithm is the smallest hence it is already turning out to be a better model but there is reason to choose this model out of all the models.

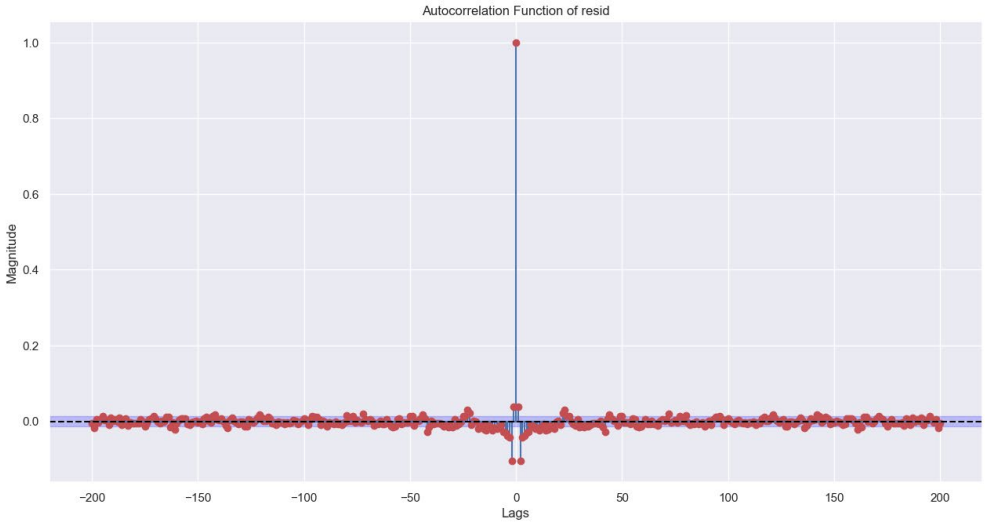
To explain why this model was chosen , there is a need to get to the early stages of the project when we had just plotted the data against time. That was the first time the data gave us the feeling that there is no clear trend or seasonality that is visible at first glance which normally is the case for most of the regular datasets when we plot them against time. But it was too immature to decide there. As we moved on with where we saw the first instance of abrupt values when we were calculating parameters using Levenberg Marquardt(L.M.) Algorithm. That is when we saw that the L.M. algorithm behaving erratically by having a lot of parameters in the output with some in the random pole cancellation interval as shown below –

```
The Confidence interval for b1 is - [[-0.0044309]] < b1 < [[0.00564762]]
The Confidence interval for b2 is - [[-0.01827055]] < b2 < [[-0.00819179]]
The Confidence interval for b3 is - [[-0.00634335]] < b3 < [[0.00374298]]
The Confidence interval for b4 is - [[-0.01030196]] < b4 < [[-0.00021635]]
The Confidence interval for b5 is - [[-0.0031335]] < b5 < [[0.00695236]]
The Confidence interval for b6 is - [[-0.01789594]] < b6 < [[-0.00781052]]
The Confidence interval for b7 is - [[-0.00993982]] < b7 < [[0.00014549]]
The Confidence interval for b8 is - [[-0.0066334]] < b8 < [[0.00345305]]
The Confidence interval for b9 is - [[-0.01326733]] < b9 < [[-0.00318229]]
The Confidence interval for b10 is - [[-0.00189473]] < b10 < [[0.00819099]]
The Confidence interval for b11 is - [[-0.00910492]] < b11 < [[0.00098076]]
The Confidence interval for b12 is - [[-0.00513156]] < b12 < [[0.00495468]]
The Confidence interval for b13 is - [[-0.00354329]] < b13 < [[0.00654302]]
The Confidence interval for b14 is - [[-0.00143172]] < b14 < [[0.00865403]]
The Confidence interval for b15 is - [[-0.01189551]] < b15 < [[-0.00180951]]
The Confidence interval for b16 is - [[-0.01087807]] < b16 < [[-0.00079244]]
```

```
The Confidence interval for b17 is - [[-0.00661117]] < b17 < [[0.00347639]]
The Confidence interval for b18 is - [[-0.01845583]] < b18 < [[-0.00836903]]
The Confidence interval for b19 is - [[-0.00143669]] < b19 < [[0.00865094]]
The Confidence interval for b20 is - [[-0.00012]] < b20 < [[0.00996812]]
The Confidence interval for b21 is - [[0.00024558]] < b21 < [[0.01033355]]
The Confidence interval for b22 is - [[-0.00155277]] < b22 < [[0.00853671]]
The Confidence interval for b23 is - [[-0.00295125]] < b23 < [[0.00713058]]
The Confidence interval for b24 is - [[-0.94572541]] < b24 < [[-0.93564321]]
```

Figure 76

Not only did these results did not align with SARIMA model output (as they should have because LM algorithm out should show same parameters), but the standalone result of SARIMA model also showed that the residual generated are not white –



When we tried to fit the residual to GPAC there was no relevant pattern of row of zeroes or col to find the remaining order.

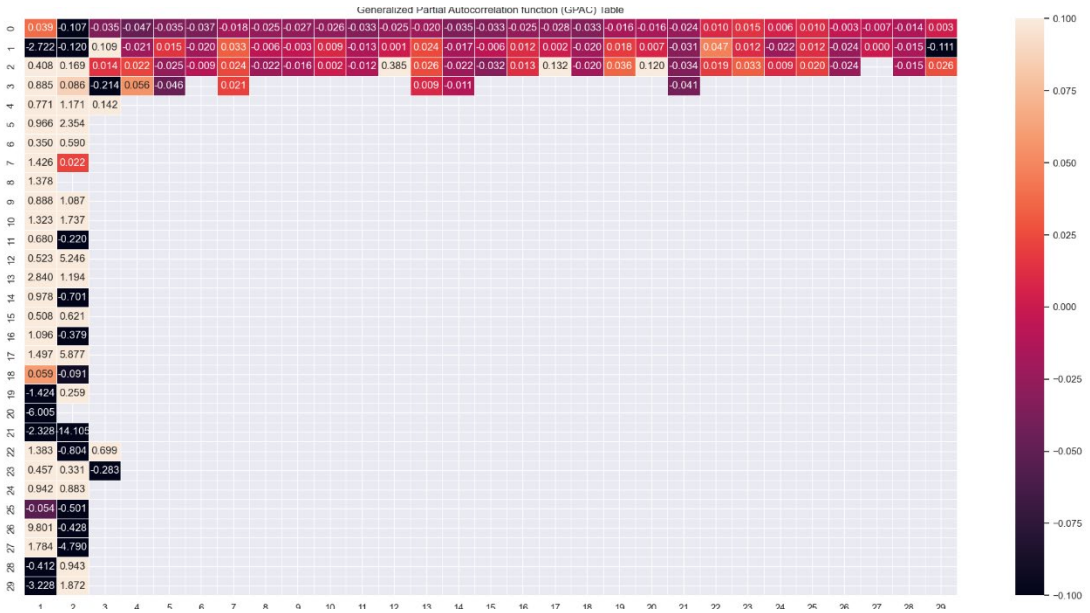


Figure 77

As for the base models, although there SSE is lower compared to the SARIMA model but it is not lower than the Gradient boosting Regressor for that matter. After this we were going to implement LSTM regardless and that is when it clicked, what is the data is non linear. All the results shown and generated above had one thing in common they all used some for of linear model building and the rule number one for linear model is that it cannot tackle non linear data.

LSTM was our first nonlinear model and it gave us great results straightaway as it is designed to look in to non linear way and find patterns from the past values which is what our data demanded(as explained above in LSTM modelling section). There was more thing that itched, which was that despite LSTM being able to cover the variance in the data to a moderate extent, it was still someway off from the actual values of power generated by turbine. It can be figured out by looking at this plot again –

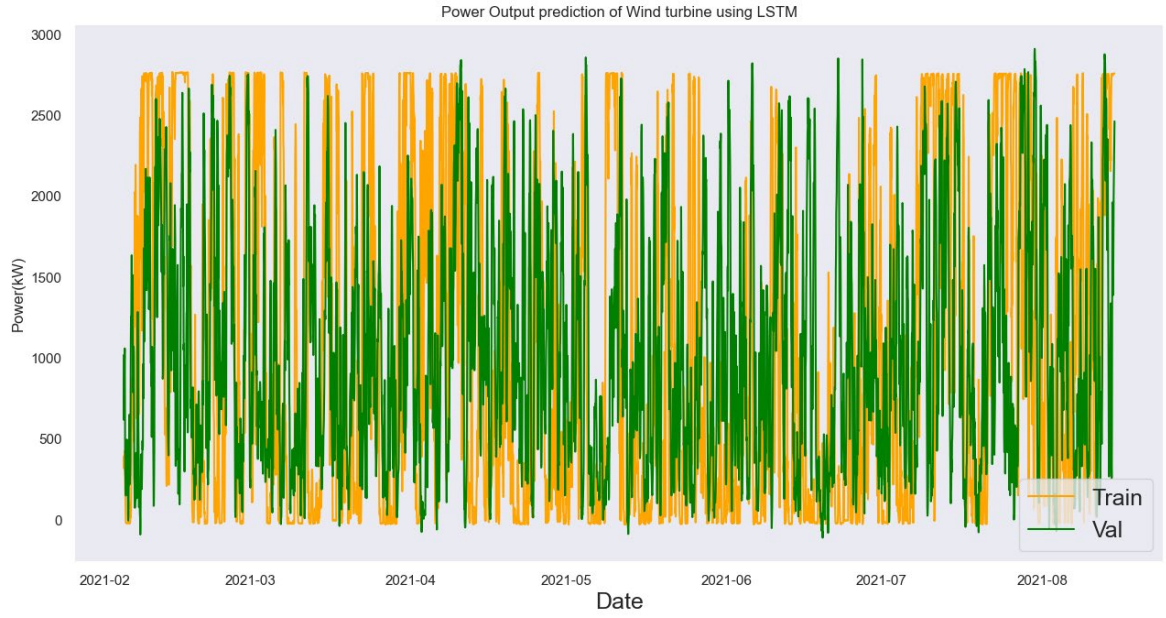


Figure 78

LSTM was able to move with the data synonymously meaning wherever there is a spike or downfall in the power output generation. It was able to imitate the behavior of the data in the very next or further coming steps. But it was not accurate with the actual values when checked against RMSE and accuracy metric as below –

```
The root mean squared error of long Short Term Memory Neural Network is =
  1189.0
The accuracy of long Short Term Memory Neural Network is =
  57.5204
```

Figure 79

The accuracy for LSTM was just alright and moderate and not the best we had seen so far until came Gradient Boosting Regression. Gradient Boosting Regression was able to imitate the behavior of the data almost accurately and the error and prediction accuracy metrics RMSE and Overall score of the model was very high and miles ahead of the rest of the models as shown below –

```
Gradient Boosting Regression Model Test Score/R2 value is =  0.9436
The Adjusted R2 value for Gradient Boosting Regression is =  0.9436
The RMSE for Gradient Boosting Regression Model is = 247.7568
```

Figure 80

Gradient boosting is one of the most popular machine learning algorithms for tabular datasets. It is powerful enough to find any nonlinear relationship between your model target and features and has great usability that can deal with missing values, outliers, and high cardinality categorical values on your features without any special treatment. While you can build barebone gradient boosting trees using some popular libraries such as XGBoost or LightGBM without knowing any details of the algorithm, you still want to know how it works when you start tuning hyper-parameters, customizing the loss functions, etc., to get better quality on your model.(towards data science). This is the core reason why it made sense to choose gradient boost model.

Forecast Function

Since SARIMA model was a linear model, it was unable to predict all the variations in the data and the resulting forecast function would not have made sense because the residuals were unable to convert to white noise.

h-Step Predictions

With our final model of Gradient boosting regression, following were our preditions –

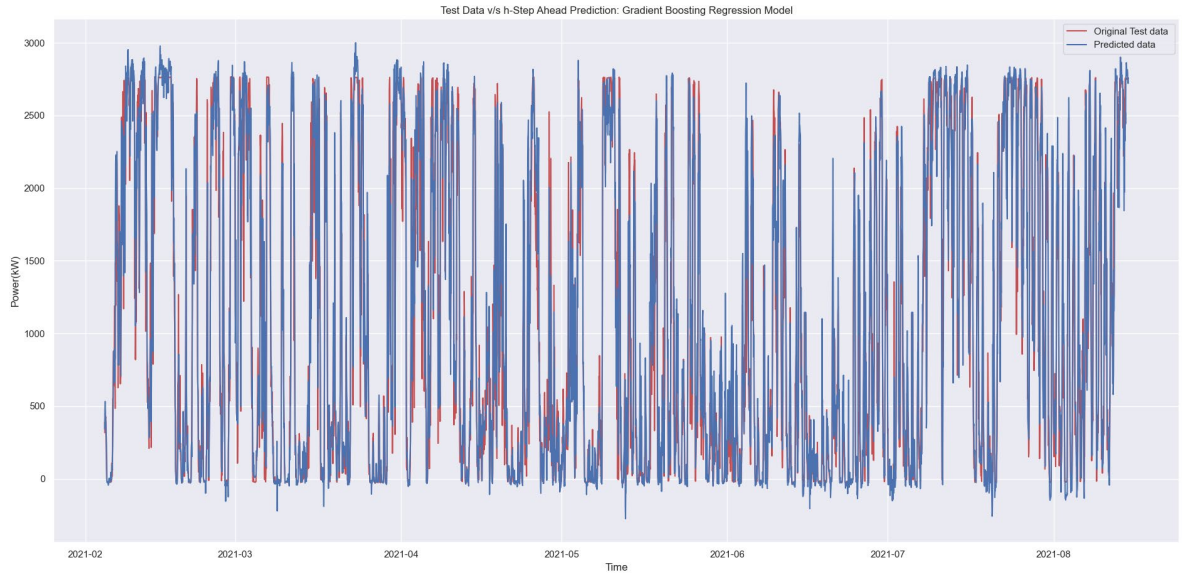


Figure 81

The algorithm most accurately was able to imitate the behaviour of the predictor and was able to explain to the best way possible.

```
Gradient Boosting Regression Model Test Score/R2 value is =  0.9436
The Adjusted R2 value for Gradient Boosting Regression is =  0.9436
The RMSE for Gradient Boosting Regression Model is = 247.7568
```

Figure 82

Summary and Conclusion

1. We were able to find the best machine learning model to predict our regressand variable and that model was Gradient Boosting Regressor.
2. It predicted the data with high accuracy and low RMSE value.\
3. The second-best model that performed decently well was LSTM model with about 57-58 percent of accuracy

To conclude the project, it is worth pointing out that the LSTM, even though it was deep neural network model, fell short in terms of predicting the values of the dependent variable more accurately then a nonlinear data centric machine learning model. This can be viewed from another perspective that now we have a certain reason to keep pursuing this idea of Wind Turbine Power forecasting using LSTM. Only this time what we can do is analyze and research the hyperparameter tuning for the LSTM Model and then try to predict the values using the tuned model. There is a very real chance that the LSTM outperforms the Gradient Boosting Regressor.

As for the other machine learning models any models in the gradient boosting and even forest family of models can be implemented to compare the accuracies that won't just be limited to one metric and multiple metrics plot together on Dash and made into a web-app deployed on cloud platform.

Appendix

```
import numpy as np
import pandas as pd
import statsmodels.api as sm
from scipy import signal
import scipy
import math
from scipy.stats import chi2
import numpy.linalg as LA
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.colors import LogNorm
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.feature_selection import SelectKBest,f_classif
from sklearn.metrics import mean_squared_error
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.seasonal import STL
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.stattools import kpss
from sklearn.model_selection import train_test_split
import pmdarima as pm
import statsmodels.tsa.holtwinters as ets
from sklearn.preprocessing import StandardScaler
from xgboost import XGBRegressor
from sklearn.decomposition import PCA
import plotly.express as px
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error
# import matplotlib
# matplotlib.use('TkAgg')
from sklearn.metrics import r2_score


features = pd.read_csv('features.csv')
power = pd.read_csv('power.csv')
df = pd.merge(features,power, how='inner', on = 'Timestamp')
df['Timestamp'] = pd.to_datetime(df.iloc[:,0])

#checking if timestamps are equidistant
time_diff = df['Timestamp'].diff()
if all(time_diff == 600):
    print("The timestamps are equally separated by 10 minutes")
else:
    print("The timestamps are not equally separated by 10 minutes")

#we will now down sample the data
df.set_index('Timestamp', inplace=True)
df = df.resample('H').mean()
df.fillna(df.mean(),inplace=True) #or use interpolate on particular column
or try using it on a dataframe
df.to_csv("Downsampled_Dataset.csv")
df = pd.read_csv("Downsampled_Dataset.csv",parse_dates=True)
df['Timestamp'] = pd.to_datetime(df['Timestamp'])
print("Timestamps have been set to hourly basis and data is now ready to be
used")

# select the index values where the time difference is not 3600 seconds
df['time_diff1'] = (df['Timestamp'] - df['Timestamp'].shift(1)).dt.seconds
mask = df['time_diff1'] != 3600
```

```
df['time_diff1'].to_csv("checkDownsampled_Dataset.csv")
result = df.loc[mask, 'Timestamp'].index

print("The non equidistant timestamps after sampling are - ", result)
df = df.drop(columns='time_diff1')


date = pd.date_range(start = '1/1/2019',
                     end = '08/14/2021',
                     periods = len(df))


fig,ax = plt.subplots(figsize = (16,8))
ax.plot(df['Timestamp'], df['Power(kW)'])
ax.set_title('Turbine Power Generation')
ax.set_xlabel('Time')
ax.set_ylabel('Power (kW)')
ax.margins(x=0)
plt.show()

def Cal_rolling_mean_var(sliced_df):
    rollingMean = []
    rollingVariance = []
    for i in range(len(sliced_df)):
        mean = (sliced_df.iloc[0:i]).mean()
        rollingMean.append(mean)
        variance = np.var(sliced_df.iloc[0:i])
        rollingVariance.append(variance)

    rollingVariance = pd.Series(rollingVariance)
    rollingVariancenotna = rollingVariance.notna()
    newrollingVariance = rollingVariance[rollingVariancenotna].tolist()


    fig, axes = plt.subplots(2, 1, figsize=(15,8))

    axes[0].plot(rollingMean)
    axes[0].set(xlabel='Samples',
            ylabel='Magnitude')
    axes[0].set_title('Rolling Mean-'+sliced_df.columns[0])

    axes[1].plot(newrollingVariance)
    axes[1].set(xlabel='Samples',
            ylabel='Magnitude')
    axes[1].set_title('Rollling Variance-'+sliced_df.columns[0])
    plt.legend(['Mean and Variance'],loc="lower right")
    plt.tight_layout()
    plt.show()
def ADF_Cal(x):
    result = adfuller(x)
    print("ADF Statistic: %f" %result[0])
    print('p-value: %f' % result[1])
    print('Critical Values:')
    for key, value in result[4].items():
        print('\t%s: %.3f' % (key, value))
def 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 ACF_PACF_Plot(y, lags):
    acf = sm.tsa.stattools.acf(y, nlags=lags)
    pacf = sm.tsa.stattools.pacf(y, nlags=lags)
    fig = plt.figure(figsize=(16,8))
    plt.subplot(211)
    plt.suptitle('ACF/PACF of 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 norderdiff(DF, col_name, order):
    # name = col_name + '_' + str(order) + '_Diff'
    # DF[name] = 0
    # temp1 = DF[col_name][::-1]
    # temp2 = temp1[0:-order] - temp1.values[order:]
    # DF[name] = temp2[::-1]
    DF[f'{col_name}_diff_{order}'] = DF[col_name].diff(order)
    return DF
def calGPAC(ry3,j,k):
    df = pd.DataFrame(0, index=range(j), columns=range(1, k))
    for a in range(1,k):
        for b in range(0,j):
            ws = (len(ry3) // 2) + a
            we = (len(ry3) // 2) + b
            num = np.zeros((a,a))
            den = np.zeros((a,a))
            for c in range(0,a+1): #loop for stacking vertical slices one
step ahead to make the a*a matrix
                if c>0 and c!=a: #if a = 6 for ex this will run for c=1 2 3
4 5

                    column = np.array(ry3[ws-a+b-c:we+a-c].values)
                    den[:,c] = column
                    #b=b-1
                elif c==0: #only for 0th column
                    column=0
                    column = np.array(ry3[ws-a+b:we+a].values)
                    den[:,c] = column
                    #b=b-1
                elif c == a:  #we allow for c in range(0,a+1) for this elif
c==a to run so that we can do num den thing
                    num = den.copy()
                    finalcol = np.array(ry3[ws-a+b+1:we+a+1].values) #add
one to those indexes to make your last column
                    num[:,-1] = finalcol
                    determnum = np.linalg.det(num)
                    determden = np.linalg.det(den)

                    if abs(determden) < 1e-5:
                        phi = np.inf
                        df.loc[b, a] = phi
                    else:
                        phi = determnum / determden
                        df.loc[b, a] = phi
                else:
                    print("Code has conceptual issues")
    #arrfinal = df.to_numpy()
    sns.heatmap(df,annot=True,fmt='.3f',linewidths=0.5)
```

```
    plt.title("Generalized Partial Autocorrelation function (GPAC) Table")
    plt.tight_layout()
    plt.show()
    return df
def paramEst(y, theta, N,na,nb):
    theta = theta.reshape(len(theta),1)
    mu = 0.01
    mu_factor = 10
    delta = 0.000001
    epsilon = 0.001
    mu_max = 1e20
    MAX = 100
    #Step 1
    n=na+nb
    numIter = 0
    SSEcompiled = []
    while numIter< MAX:
        MAPARAMLOOPNUM = [0] * max(na, nb)
        ARPARAMLOOPDEN = [0] * max(na, nb)

        MAPARAMLOOPNUM[:nb] = theta[na:]
        ARPARAMLOOPDEN[:na] = theta[:na]

        num = np.insert(MAPARAMLOOPNUM, 0, 1).tolist()
        den1 = np.insert(ARPARAMLOOPDEN, 0, 1).tolist()
        den = [valchk if not isinstance(valchk,np.ndarray) else
valchk.flatten()[0] for valchk in den1]

        system = (den, num, 1)  # den,num,1 because we are generating
synthetic error
        _, eNew = signal.dlsim(system, y)
        #eNew = eNew[:, 0]####
        SSE = np.dot(eNew.T , eNew)  # SSE = e^T.e
        SSEcompiled.append(SSE.ravel().flatten())

        X = np.empty((N, n)).reshape(N, n)
        for i in range(0, n):  # X construction loop
            thetatemp = theta.copy()
            thetatemp[i] = thetatemp[i] + delta

            MAPARAMLOOPNUM = [0] * max(na, nb)
            ARPARAMLOOPDEN = [0] * max(na, nb)

            MAPARAMLOOPNUM[:nb] = thetatemp[na:]
            ARPARAMLOOPDEN[:na] = thetatemp[:na]

            num = np.insert(MAPARAMLOOPNUM, 0, 1).tolist()
            den1 = np.insert(ARPARAMLOOPDEN, 0, 1).tolist()
            den = [valchk if not isinstance(valchk, np.ndarray) else
valchk.flatten()[0] for valchk in den1]

            system = (den, num, 1)  # den,num,1 because we are generating
synthetic error
            _, eNewi = signal.dlsim(system, y)

            X[:, i] = ((eNew.ravel() - eNewi.ravel()) / delta)  # array
shape is flat

        A = np.dot(X.T, X)
        g = np.dot(X.T, eNew)
```

```
        # Step2
        delta_theta = np.dot(LA.inv(A + (mu * np.identity(n))) , g)
        thetaNew = theta + delta_theta
        MAPARAMNUMthetaNew = [0] * max(na, nb)
        ARPARAMDENthetaNew = [0] * max(na, nb)

        ARPARAMDENthetaNew[:na] = thetaNew[:na]
        MAPARAMNUMthetaNew[:nb] = thetaNew[na:]

        num = np.insert(MAPARAMNUMthetaNew, 0, 1).tolist()
        den1 = np.insert(ARPARAMDENthetaNew, 0, 1).tolist()
        den = [valchk if not isinstance(valchk, np.ndarray) else
valchk.flatten()[0] for valchk in den1]

        system = (den, num, 1)  # den,num,1 because we are generating
synthetic error
        _, eNewthetaNew = signal.dlsim(system, y)
        SSENew = np.dot(eNewthetaNew.T,eNewthetaNew)

        if SSENew<SSE:
            if LA.norm(delta_theta)<epsilon:
                thetacap = thetaNew.copy()
                variancecapofError = (SSENew/(N-n))
                sdval = np.sqrt(variancecapofError)
                covariancethetacap = variancecapofError*(LA.inv(A))
                covDiag = np.diag(covariancethetacap)
                print(f"Estimated params are:\n {thetacap}")
                # print(f"True params are: {AN[1:na+1] + BN[1:nb+1]}")
                print(f"Standard deviation of the Error is : {sdval}")
                print(f"Estimated variance of Error is :
{variancecapofError}")
                numMARoots = np.roots(num)
                denARRoots = np.roots(den)

                for b in range(0,len(numMARoots)):
                    print(f"Roots of the numerators is/are-
{numMARoots[b]}")

                for a in range(0,len(denARRoots)):
                    print(f"Roots of the denominator is/are-
{denARRoots[a]}")

                confIntna = thetacap[:na]
                confIntnb = thetacap[na:]
                for l in range(0,na):
                    print(f"The Confidence interval for a{l+1} is -
{confIntna[[l]] - 2*(np.sqrt(covDiag[[l]]))} < a{l+1} < {confIntna[[l]] +
2*(np.sqrt(covDiag[[l]]))} ")

                for m in range(0,nb):
                    print(f"The Confidence interval for b{m+1} is -
{confIntnb[[m]] - 2*(np.sqrt(covDiag[[m]]))} < b{m+1} < {confIntnb[[m]] +
2*(np.sqrt(covDiag[[m]]))} ")

                print(f"The Covariance Matrix is:\n {covariancethetacap}")
                plt.plot(SSEcompiled, label='S.S.E.', color='purple')
                plt.grid()
                plt.legend()
                plt.margins(x=0)
                plt.xticks(np.arange(0, numIter + 1, step=1))
                plt.title('Sum of Squared Errors ')
```

```
                plt.xlabel('Iterations')
                plt.ylabel('Magnitude of Error')
                plt.figure(figsize=(16, 8))
                plt.show()
                return "Correct"
            else:
                theta = thetaNew.copy()
                mu = mu/10

        while SSENew>=SSE:
            mu = mu*10
            if mu>mu_max:
                thetacap = thetaNew.copy()
                variancecapofError = (SSENew/(N-n))
                sdval = np.sqrt(variancecapofError)
                covariancethetacap = variancecapofError*(LA.inv(A))
                covDiag = np.diag(covariancethetacap)
                print(f"Estimated params are:\n {thetacap}")
                # print(f"True params are: {AN[1:na] + BN[1:nb]}")
                print(f"Standard Deviation of the Error is : {sdval}")
                numMARoots = np.roots(thetacap[na:].ravel())
                denARRoots = np.roots(thetacap[:na].ravel())

                for b in range(0,len(numMARoots)):
                    print(f"Roots of the numerators is/are -
{numMARoots[b]}")

                for a in range(0,len(denARRoots)):
                    print(f"Roots of the denominator is/are -
{denARRoots[a]}")

                confIntna = thetacap[:na]
                confIntnb = thetacap[na:]
                for l in range(0,na):
                    print(f"The Confidence interval for a{l+1} is -
{confIntna[[l]] - 2*(np.sqrt(covDiag[[l]]))} < a{l+1} < {confIntna[[l]] +
2*(np.sqrt(covDiag[[l]]))} ")

                for m in range(0,nb):
                    print(f"The Confidence interval for b{m+1} is -
{confIntnb[[m]] - 2*(np.sqrt(covDiag[[m]]))} < b{m+1} < {confIntnb[[m]] +
2*(np.sqrt(covDiag[[m]]))} ")

                print(f"The Covariance Matrix is:\n {covariancethetacap}")

                print(f"mu greater than mumax")
                plt.plot(SSEcompiled, label='S.S.E.', color='purple')
                plt.grid()
                plt.legend()
                plt.margins(x=0)
                plt.xticks(np.arange(0, numIter + 1, step=1))
                plt.title('Sum of Squared Errors ')
                plt.xlabel('Iterations')
                plt.ylabel('Magnitude of Error')
                plt.figure(figsize=(16, 8))
                plt.show()
                return "Mu >Mu_Max"

            delta_theta = np.dot(LA.inv(A + (mu * np.identity(n))), g)
            thetaNew = theta + delta_theta
```

```
            MAPARAMNUMthetaNew = [0] * max(na, nb)
            ARPARAMDENthetaNew = [0] * max(na, nb)

            ARPARAMDENthetaNew[:na] = thetaNew[:na]
            MAPARAMNUMthetaNew[:nb] = thetaNew[na:]

            num = np.insert(MAPARAMNUMthetaNew, 0, 1).tolist()
            den = np.insert(ARPARAMDENthetaNew, 0, 1).tolist()

            system = (den, num, 1)  # den,num,1 because we are generating
synthetic error
            _, eNewthetaNew = signal.dlsim(system, y)

            SSENew = np.dot(eNewthetaNew.T, eNewthetaNew)

        numIter+=1
        if numIter>MAX:
            thetacap = thetaNew.copy()
            variancecapofError = (SSENew / (N - n))
            covariancethetacap = variancecapofError * (LA.inv(A))
            covDiag = np.diag(covariancethetacap)
            print(f"Estimated params are:\n {thetacap}")
            # print(f"True params are: {AN[1:na] + BN[1:nb]}")#------------
-----
            print(f"Variance of the Error is : {variancecapofError}")
            numMARoots = np.roots(thetacap[na:].ravel())
            denARRoots = np.roots(thetacap[:na].ravel())

            for b in range(0, len(numMARoots)):
                print(f"Roots of the numerator is/are -{numMARoots[b]}")

            for a in range(0, len(denARRoots)):
                print(f"Roots of the denominator is/ are -{denARRoots[a]}")

            confIntna = thetacap[:na]
            confIntnb = thetacap[na:]
            for l in range(0, na):
                print(
                    f"The Confidence interval for a{l + 1} is -
{confIntna[[l]] - 2 * (np.sqrt(covDiag[[l]]))} < a{l + 1} < {confIntna[[l]]
+ 2 * (np.sqrt(covDiag[[l]]))} ")

            for m in range(0, nb):
                print(
                    f"The Confidence interval for b{m + 1} is -
{confIntnb[[m]] - 2 * (np.sqrt(covDiag[[m]]))} < b{m + 1} < {confIntnb[m] +
2 * (np.sqrt(covDiag[[m]]))} ")

            print(f"The Covariance Matrix is:\n {covariancethetacap}")
            print(f"Iteration higher than MAX")

            plt.plot(SSEcompiled, label='S.S.E.', color='purple')
            plt.grid()
            plt.legend()
            plt.margins(x=0)
            plt.xticks(np.arange(0, numIter + 1, step=1))
            plt.title('Sum of Squared Errors ')
            plt.xlabel('Iterations')
            plt.ylabel('Magnitude of Error')
            plt.figure(figsize=(16, 8))
            plt.show()
```

```
            return "NumIter > MAX"

        theta = thetaNew.copy()
def ACFPlot(array, lag, title):
    lag = list(range(-lag, lag + 1))
    numerator = 0
    denominator = 0
    mean = np.mean(array)

    for i in range(len(array)):
        denominator = (array[i] - mean) ** 2 + denominator  # cumalative
sum

    totalacf = []
    for j in lag:
        j = abs(j)
        numerator = 0
        for k in range(j, len(array)):
            numerator = (array[k] - mean) * (array[k - j] - mean) +
numerator  # num cumalative sum
        acfonelag = numerator / denominator
        totalacf.append(acfonelag)  # appending those sums in array

    plt.stem(lag, totalacf, markerfmt='ro', basefmt='b-',
use_line_collection=True)
    plt.axhspan(1.96 / len(array) ** 0.5, -1.96 / len(array) ** 0.5,
alpha=0.2, color='blue')
    plt.axhline(y=0, color='black', linestyle='--')
    plt.title('Autocorrelation Function of ' + title)
    plt.xlabel('Lags')
    plt.ylabel('Magnitude')
    plt.figure(figsize=(16, 8))
    plt.show()
    return totalacf


df.iloc[:,0:-1].describe()
df.iloc[:,-1].describe()


#Raw Data
subsetdf = df[['Power(kW)']]
ACF_PACF_Plot(subsetdf,200)

#Corr plot

X,y = df.iloc[:,1:-1],df[['Power(kW)']]

featureforcorr = SelectKBest(score_func = f_classif, k=10)
X1 = featureforcorr.fit_transform(X,y)
listfeatureforcorr = []
for columnname, true in zip(df.iloc[:,1:-1].columns,
featureforcorr.get_support()):
    if true:
        listfeatureforcorr.append(columnname)


print("The chosen features for plotting heatmap are-\n ",
listfeatureforcorr)
```

```
corrdf = df.loc[:,listfeatureforcorr]
corrdf = pd.concat([corrdf,subsetdf],axis=1)
correlationplot = corrdf.corr(method='pearson', numeric_only=True)
sns.set(font_scale=5)
sns.set(rc={'figure.figsize':(20,20)})
sns.heatmap(correlationplot, cmap='coolwarm',annot=True)
plt.title("Heatmap of Data")
plt.tight_layout()
plt.show()


# model=pm.auto_arima(subsetdf[:5000],start_p=0,d=0,start_q=0,
#           max_p=15,max_q=5,
#           start_P=0,D=0, start_Q=0,
#           max_P=8, max_Q=8,
#           m=24, seasonal=True, stationary=True,
#           error_action='warn',trace=True,
#           supress_warnings=True,stepwise=True,
#           random_state=20,n_fits=100)

#Train test split
X_train, X_test, y_train, y_test = train_test_split(X,y,
                                    random_state=6313,
                                    test_size=0.20,
                                    shuffle=False)

print("Size of independent variable's training set is -", X_train.shape)
print("Size of dependent variables's training set is -", y_train.shape)
print("Size of independent variable's test set is -", X_test.shape)
print("Size of dependent variables's test set is -", y_test.shape)

subsetdf = y_train.copy(deep = true)
#Stationarity test
Cal_rolling_mean_var(subsetdf)
ADF_Cal(subsetdf)
kpss_test(subsetdf)
ACF_PACF_Plot(subsetdf,200)

#STL decomposition complete process
from statsmodels.tsa.seasonal import STL
#RAW DATA
STL = STL(subsetdf, period=24)
res = STL.fit()
fig,ax = plt.subplots(nrows=4,ncols=1,figsize = (16,8))
res.observed.plot(ax =ax[0])
ax[0].set_ylabel('Observed Value')
ax[0].margins(x=0)
res.trend.plot(ax =ax[1])
ax[1].set_ylabel('Trend')
ax[1].margins(x=0)
res.seasonal.plot(ax =ax[2])
ax[2].set_ylabel('Seasonal')
ax[2].margins(x=0)
res.resid.plot(ax =ax[3])
ax[3].set_ylabel('Residual')
ax[3].margins(x=0)
plt.tight_layout()
plt.show()

T = res.trend
S = res.seasonal
```

```
R = res.resid
plt.figure(figsize=(16,8))
plt.plot(T,label = 'trend')
plt.plot(S,label = 'Seasonal')
plt.plot(R,label = 'residuals')
plt.plot(subsetdf, label = 'original data')
plt.legend()
plt.title("STL Decomposition Combined Plot Power Magnitude v/s Samples")
plt.xticks(rotation = 45)
plt.grid()
plt.xlabel("Samples")
plt.ylabel("Power(kW)")
plt.tight_layout()
plt.margins(x=0)
plt.show()

num = np.var(R)
deno = np.var(T+R)
strengthofTrendFt =max(0,(1- (num/deno)))
print("The strength of trend for this data set is - ",
np.round(strengthofTrendFt,4))
num = np.var(R)
deno = np.var(R+S)
strengthofSeasonalityFs =max(0,(1- (num/deno)))
print("The strength of seasonality for this data set is - ",
np.round(strengthofSeasonalityFs,4))


# #We find SOT, SOS high even though the adf kpss and rolling mean var say
otherwise so we perform one order non seasonal diff
#TRANSFORMED DATA 1
subsetdf = norderdiff(subsetdf,'Power(kW)',1)
subsetdf = subsetdf.dropna()
subsetdf = subsetdf.reset_index()
subsetdf = subsetdf.iloc[:,-1]
subsetdf = pd.DataFrame(subsetdf)
Cal_rolling_mean_var(subsetdf)
ADF_Cal(subsetdf)
kpss_test(subsetdf)
ACF_PACF_Plot(subsetdf,200)

#Then we check STL decomposition again
from statsmodels.tsa.seasonal import STL
STL = STL(subsetdf, period=24)
res = STL.fit()
fig,ax = plt.subplots(nrows=4,ncols=1,figsize = (16,8))
res.observed.plot(ax =ax[0])
ax[0].set_ylabel('Observed Value')
ax[0].margins(x=0)
res.trend.plot(ax =ax[1])
ax[1].set_ylabel('Trend')
ax[1].margins(x=0)
res.seasonal.plot(ax =ax[2])
ax[2].set_ylabel('Seasonal')
ax[2].margins(x=0)
res.resid.plot(ax =ax[3])
ax[3].set_ylabel('Residual')
ax[3].margins(x=0)
plt.tight_layout()
plt.show()
```

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

plt.figure(figsize=(16,8))
plt.plot(T,label = 'trend')
plt.plot(S,label = 'Seasonal')
plt.plot(R,label = 'residuals')
plt.plot(subsetdf, label = 'original data')
plt.legend()
plt.xticks(rotation = 45)
plt.grid()
plt.title("STL Decomposition Combined Plot Power Magnitude v/s Samples")
plt.xlabel("Samples")
plt.ylabel("Power(kW)")
plt.tight_layout()
plt.margins(x=0)
plt.show()

num = np.var(R)
deno = np.var(T+R)
strengthofTrendFt =max(0,(1- (num/deno)))
print("The strength of trend for this data set is - ",
np.round(strengthofTrendFt,4))
num = np.var(R)
deno = np.var(R+S)
strengthofSeasonalityFs =max(0,(1- (num/deno)))
print("The strength of seasonality for this data set is - ",
np.round(strengthofSeasonalityFs,4))
seasonAdj = subsetdf['Power(kW)_diff_1']-S
plt.figure(figsize=(16,8))
plt.plot(subsetdf, label = 'Original data', color = 'green')
plt.plot(seasonAdj, label = 'Seasonally Adjusted Data', color = 'brown')
plt.grid()
plt.legend()
plt.xlabel("Samples")
plt.ylabel("Power(kW)")
plt.title("Original data v/s Seasonally Adjusted Data")
plt.tight_layout()
plt.show()
trendAdj = subsetdf['Power(kW)_diff_1']-T
plt.figure(figsize=(16,8))
plt.plot(subsetdf, label = 'Original data', color = 'green')
plt.plot(trendAdj, label = 'Trend Adjusted Data', color = 'brown')
plt.grid()
plt.legend()
plt.xlabel("Samples")
plt.ylabel("Power(kW)")
plt.title("Original data v/s Trend Adjusted Data")
plt.tight_layout()
plt.show()


#Holt Winters
holtwintersubset = df[['Power(kW)']]
holtwintersubset.index = date
yt, yf = train_test_split(holtwintersubset, shuffle= False, test_size=0.2)
holtt =
ets.ExponentialSmoothing(yt,trend='add',damped_trend=True,seasonal='add',
seasonal_periods=24).fit()
holtf = holtt.forecast(steps=len(yf))
```

```
holtf = pd.DataFrame(holtf).set_index(yf.index)
holtwinterRMSE =
np.sqrt(np.square(np.subtract(yf.values,np.ndarray.flatten(holtf.values))).
mean())
print(f'Root Mean square error for Holt-Winter method is
{holtwinterRMSE:.2f}')
fig, ax = plt.subplots(figsize = (16,8))
ax.plot(df['Timestamp'][:len(yt)],yt,label= "Train Data")
ax.plot(df['Timestamp'][len(yt):],yf,label= "Test Data")
ax.plot(df['Timestamp'][len(yt):],holtf,label= "Holt-Winter")
plt.legend(loc='upper left')
plt.title(f'Holt-Winter- MSE = {holtwinterRMSE:.2f}')
plt.xlabel('Time')
plt.ylabel('Power(kW)')
plt.grid()
plt.show()


#Feature Selection/Elimination
X,y = df.iloc[:,1:-1],df[['Power(kW)']]
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X.values, i)
                   for i in range(len(X.columns))]

print(np.round(vif_data,4))
#https://www.geeksforgeeks.org/detecting-multicollinearity-with-vif-python/

matH = (X.transpose())@X
s, d, v = np.linalg.svd(matH)
print("SingularValues = ", np.round(d,4))

condNo = LA.cond(X)
print("Condition Number = ", np.round(condNo,4) )
#Scaler Transformation
scaler = StandardScaler()
XtrainScaled = scaler.fit_transform(X_train)
XtestScaled = scaler.transform(X_test)
print("Transformation successfull, fit transform for train sets and then
transform for test set")
XtrainScaled = sm.add_constant(XtrainScaled)
xcols = X_test.columns.tolist()
xcols = ['Constant']+xcols
XtrainScaleddf = pd.DataFrame(XtrainScaled, columns=xcols)
#https://statisticsbyjim.com/regression/interpret-constant-y-intercept-
regression/#:~:text=The%20reason%20I%20just%20discussed,the%20constant%20to
%20equal%20zero.



model = sm.OLS(y_train,XtrainScaleddf).fit()
print(model.params)
print(model.summary())

#dropping features
XtrainScaleddf2 = XtrainScaleddf.drop(columns=["Blade-3 Set Value_Degree"])
model2 = sm.OLS(y_train,XtrainScaleddf2).fit()
print(model2.summary())

XtrainScaleddf3 = XtrainScaleddf2.drop(columns=["Temperature Battery Box-
2"])
```

```
model3 = sm.OLS(y_train,XtrainScaleddf3).fit()
print(model3.summary())


XtrainScaleddf4 = XtrainScaleddf3.drop(columns=["Temperature Axis Box-1"])
model4 = sm.OLS(y_train,XtrainScaleddf4).fit()
print(model4.summary())

XtrainScaleddf5 = XtrainScaleddf4.drop(columns=["Temperature Ambient"])
model5 = sm.OLS(y_train,XtrainScaleddf5).fit()
print(model5.summary())


XtrainScaleddf6 =
XtrainScaleddf5.drop(columns=["Gearbox_Distributor_Temperature"])
model6 = sm.OLS(y_train,XtrainScaleddf6).fit()
print(model6.summary())

XtrainScaleddf7 = XtrainScaleddf6.drop(columns=["Converter Control Unit
Voltage"])
model7 = sm.OLS(y_train,XtrainScaleddf7).fit()
print(model7.summary())

XtrainScaleddf8 = XtrainScaleddf7.drop(columns=["Power Factor"])
model8 = sm.OLS(y_train,XtrainScaleddf8).fit()
print(model8.summary())

XtrainScaleddf9 = XtrainScaleddf8.drop(columns=["Temperature Axis Box-3"])
model9 = sm.OLS(y_train,XtrainScaleddf9).fit()
print(model9.summary())

XtrainScaleddf10 = XtrainScaleddf9.drop(columns=["Hydraulic Prepressure"])
model10 = sm.OLS(y_train,XtrainScaleddf10).fit()
print(model10.summary())

XtrainScaleddf11 = XtrainScaleddf10.drop(columns=["Proxy Sensor_Degree-
135"])
model11 = sm.OLS(y_train,XtrainScaleddf11).fit()
print(model11.summary())

XtrainScaleddf12 = XtrainScaleddf11.drop(columns=["Internal Power Limit"])
model12 = sm.OLS(y_train,XtrainScaleddf12).fit()
print(model12.summary())

XtrainScaleddf13 = XtrainScaleddf12.drop(columns=["Blade-1 Set
Value_Degree"])
model13 = sm.OLS(y_train,XtrainScaleddf13).fit()
print(model13.summary())

XtrainScaleddf14 = XtrainScaleddf13.drop(columns=["Pitch Offset Tower
Feedback"])
model14 = sm.OLS(y_train,XtrainScaleddf14).fit()
print(model14.summary())

XtrainScaleddf15 = XtrainScaleddf14.drop(columns=["Moment Q Filltered"])
model15 = sm.OLS(y_train,XtrainScaleddf15).fit()
print(model15.summary())

XtrainScaleddf16 = XtrainScaleddf15.drop(columns=["Temperature Tower
Base"])
model16 = sm.OLS(y_train,XtrainScaleddf16).fit()
```

```
print(model16.summary())

XtrainScaleddf17 =
XtrainScaleddf16.drop(columns=["Gearbox_T3_Intermediate_Speed_Shaft_Tempera
ture"])
model17 = sm.OLS(y_train,XtrainScaleddf17).fit()
print(model17.summary())

XtrainScaleddf18 = XtrainScaleddf17.drop(columns=["Circuit Breaker cut-
ins"])
model18 = sm.OLS(y_train,XtrainScaleddf18).fit()
print(model18.summary())

XtrainScaleddf19 = XtrainScaleddf18.drop(columns=["Scope CH 4"])
model19 = sm.OLS(y_train,XtrainScaleddf19).fit()
print(model19.summary())


XtrainScaleddf20 = XtrainScaleddf19.drop(columns=["Moment Q Direction"])
model20 = sm.OLS(y_train,XtrainScaleddf20).fit()
print(model20.summary())

XtrainScaleddf21 = XtrainScaleddf20.drop(columns=["Tower Deflection"])
model21 = sm.OLS(y_train,XtrainScaleddf21).fit()
print(model21.summary())


XtrainScaleddf22 = XtrainScaleddf21.drop(columns=["Blade-3 Actual
Value_Angle-A"])
model22 = sm.OLS(y_train,XtrainScaleddf22).fit()
print(model22.summary())

XtrainScaleddf23 = XtrainScaleddf22.drop(columns=["Torque Offset Tower
Feedback"])
model23 = sm.OLS(y_train,XtrainScaleddf23).fit()
print(model23.summary())


XtrainScaleddf24 = XtrainScaleddf23.drop(columns=["Pitch Offset-2
Asymmetric Load Controller"])
model24 = sm.OLS(y_train,XtrainScaleddf24).fit()
print(model24.summary())

XtrainScaleddf25 = XtrainScaleddf24.drop(columns=["Line Frequency"])
model25 = sm.OLS(y_train,XtrainScaleddf25).fit()
print(model25.summary())

XtrainScaleddf26 = XtrainScaleddf25.drop(columns=["Wind Deviation 10
seconds"])
model26 = sm.OLS(y_train,XtrainScaleddf26).fit()
print(model26.summary())

XtrainScaleddf27 = XtrainScaleddf26.drop(columns=["Nacelle
Position_Degree"])
model27 = sm.OLS(y_train,XtrainScaleddf27).fit()
print(model27.summary())

XtrainScaleddf28 = XtrainScaleddf27.drop(columns=["Temperature Heat
Exchanger Converter Control Unit"])
model28 = sm.OLS(y_train,XtrainScaleddf28).fit()
print(model28.summary())
```

```
XtrainScaleddf29 = XtrainScaleddf28.drop(columns=["Blade-3 Actual
Value_Angle-B"])
model29 = sm.OLS(y_train,XtrainScaleddf29).fit()
print(model29.summary())


XtrainScaleddf30 = XtrainScaleddf29.drop(columns=["Temperature Axis Box-
2"])
model30 = sm.OLS(y_train,XtrainScaleddf30).fit()
print(model30.summary())

XtrainScaleddf31 =
XtrainScaleddf30.drop(columns=["Gearbox_T3_High_Speed_Shaft_Temperature"])
model31 = sm.OLS(y_train,XtrainScaleddf31).fit()
print(model31.summary())

XtrainScaleddf32 = XtrainScaleddf31.drop(columns=["Tower Acceleration
Lateral"])
model32 = sm.OLS(y_train,XtrainScaleddf32).fit()
print(model32.summary())


XtrainScaleddf33 = XtrainScaleddf32.drop(columns=["Gearbox_Oil-
1_Temperature"])
model33 = sm.OLS(y_train,XtrainScaleddf33).fit()
print(model33.summary())


XtrainScaleddf34 = XtrainScaleddf33.drop(columns=["Voltage B-N"])
model34 = sm.OLS(y_train,XtrainScaleddf34).fit()
print(model34.summary())


XtrainScaleddf35 = XtrainScaleddf34.drop(columns=["Temperature_Nacelle"])
model35 = sm.OLS(y_train,XtrainScaleddf35).fit()
print(model35.summary())


XtrainScaleddf36 = XtrainScaleddf35.drop(columns=["Blade-2 Actual
Value_Angle-B"])
model36 = sm.OLS(y_train,XtrainScaleddf36).fit()
print(model36.summary())


XtrainScaleddf37 = XtrainScaleddf36.drop(columns=["State and Fault"])
model37 = sm.OLS(y_train,XtrainScaleddf37).fit()
print(model37.summary())

XtrainScaleddf38 = XtrainScaleddf37.drop(columns=["Temperature Trafo-3"])
model38 = sm.OLS(y_train,XtrainScaleddf38).fit()
print(model38.summary())


XtrainScaleddf39 = XtrainScaleddf38.drop(columns=["Tower Accelaration
Normal Raw"])
model39 = sm.OLS(y_train,XtrainScaleddf39).fit()
print(model39.summary())

XtrainScaleddf40 = XtrainScaleddf39.drop(columns=["Blade-2 Set
Value_Degree"])
```

```
model40 = sm.OLS(y_train,XtrainScaleddf40).fit()
print(model40.summary())


XtrainScaleddf41 =
XtrainScaleddf40.drop(columns=["Gearbox_T1_Intermediate_Speed_Shaft_Tempera
ture"])
model41 = sm.OLS(y_train,XtrainScaleddf41).fit()
print(model41.summary())


XtrainScaleddf42 = XtrainScaleddf41.drop(columns=["Voltage A-N"])
model42 = sm.OLS(y_train,XtrainScaleddf42).fit()
print(model42.summary())


XtrainScaleddf43 = XtrainScaleddf42.drop(columns=["Temperature Battery Box-
3"])
model43 = sm.OLS(y_train,XtrainScaleddf43).fit()
print(model43.summary())

XtrainScaleddf44 =
XtrainScaleddf43.drop(columns=["Gearbox_T1_High_Speed_Shaft_Temperature"])
model44 = sm.OLS(y_train,XtrainScaleddf44).fit()
print(model44.summary())


XtrainScaleddf45 = XtrainScaleddf44.drop(columns=["Operating State"])
model45 = sm.OLS(y_train,XtrainScaleddf45).fit()
print(model45.summary())

XtrainScaleddf46 = XtrainScaleddf45.drop(columns=["Proxy Sensor_Degree-
315"])
model46 = sm.OLS(y_train,XtrainScaleddf46).fit()
print(model46.summary())


XtrainScaleddf47 = XtrainScaleddf46.drop(columns=["Pitch Offset-1
Asymmetric Load Controller"])
model47 = sm.OLS(y_train,XtrainScaleddf47).fit()
print(model47.summary())


XtrainScaleddf48 = XtrainScaleddf47.drop(columns=["Blade-1 Actual
Value_Angle-A"])
model48 = sm.OLS(y_train,XtrainScaleddf48).fit()
print(model48.summary())

XtrainScaleddf49 = XtrainScaleddf48.drop(columns=["Turbine State"])
model49 = sm.OLS(y_train,XtrainScaleddf49).fit()
print(model49.summary())


XtrainScaleddf50 = XtrainScaleddf49.drop(columns=["Blade-2 Actual
Value_Angle-A"])
model50 = sm.OLS(y_train,XtrainScaleddf50).fit()
print(model50.summary())

XtrainScaleddf51 = XtrainScaleddf50.drop(columns=["External Power Limit"])
model51 = sm.OLS(y_train,XtrainScaleddf51).fit()
print(model51.summary())
```

```
XtrainScaleddf52 =
XtrainScaleddf51.drop(columns=["Gearbox_Oil_Temperature"])
model52 = sm.OLS(y_train,XtrainScaleddf52).fit()
print(model52.summary())


XtrainScaleddf53 = XtrainScaleddf52.drop(columns=["Voltage C-N"])
model53 = sm.OLS(y_train,XtrainScaleddf53).fit()
print(model53.summary())

XtrainScaleddf54 = XtrainScaleddf53.drop(columns=["Proxy Sensor_Degree-
45"])
model54 = sm.OLS(y_train,XtrainScaleddf54).fit()
print(model54.summary())


XtrainScaleddf55 = XtrainScaleddf54.drop(columns=["Temperature Bearing_A"])
model55 = sm.OLS(y_train,XtrainScaleddf55).fit()
print(model55.summary())

XtrainScaleddf56 = XtrainScaleddf55.drop(columns=["Pitch Demand
Baseline_Degree"])
model56 = sm.OLS(y_train,XtrainScaleddf56).fit()
print(model56.summary())


XtrainScaleddf57 = XtrainScaleddf56.drop(columns=["Gearbox_Oil-
2_Temperature"])
model57 = sm.OLS(y_train,XtrainScaleddf57).fit()
print(model57.summary())


XtrainScaleddf58 = XtrainScaleddf57.drop(columns=["Temperature Battery Box-
1"])
model58 = sm.OLS(y_train,XtrainScaleddf58).fit()
print(model58.summary())

XtrainScaleddf59 = XtrainScaleddf58.drop(columns=["Proxy Sensor_Degree-
225"])
model59 = sm.OLS(y_train,XtrainScaleddf59).fit()
print(model59.summary())
#------------------------------------------------------------------------
--- removed torque by mistake added back and
# then removed moment d filtered now seeing double torque but removed in
next line

XtrainScaleddf60 = XtrainScaleddf59.drop(columns=["Moment D Filtered"])
model60 = sm.OLS(y_train,XtrainScaleddf60).fit()
print(model60.summary())

XtrainScaleddf61 = XtrainScaleddf60.drop(columns=["Torque"])
model61 = sm.OLS(y_train,XtrainScaleddf61).fit()
print(model61.summary())


XtrainScaleddf62 = XtrainScaleddf61.drop(columns=["Nacelle Revolution"])
model62 = sm.OLS(y_train,XtrainScaleddf62).fit()
print(model62.summary())
```

```
XtrainScaleddf63 = XtrainScaleddf62.drop(columns=["Tower Accelaration
Lateral Raw"])
model63 = sm.OLS(y_train,XtrainScaleddf63).fit()
print(model63.summary())



XtrainScaleddf64 = XtrainScaleddf63.drop(columns=["Pitch Offset-3
Asymmetric Load Controller"])
model64 = sm.OLS(y_train,XtrainScaleddf64).fit()
print(model64.summary())

XtrainScaleddf65 = XtrainScaleddf64.drop(columns=["Temperature Shaft
Bearing-2"])
model65 = sm.OLS(y_train,XtrainScaleddf65).fit()
print(model65.summary())


XtrainScaleddf66 = XtrainScaleddf65.drop(columns=["Temperature Shaft
Bearing-1"])
model66 = sm.OLS(y_train,XtrainScaleddf66).fit()
print(model66.summary())

XtrainScaleddf67 = XtrainScaleddf66.drop(columns=["Temperature Gearbox
Bearing Hollow Shaft"])
model67 = sm.OLS(y_train,XtrainScaleddf67).fit()
print(model67.summary())

XtrainScaleddf68 = XtrainScaleddf67.drop(columns=["Temperature Trafo-2"])
model68 = sm.OLS(y_train,XtrainScaleddf68).fit()
print(model68.summary())


XtrainScaleddf69 = XtrainScaleddf68.drop(columns=["Angle Rotor Position"])
model69 = sm.OLS(y_train,XtrainScaleddf69).fit()
print(model69.summary())

XtrainScaleddf70 = XtrainScaleddf69.drop(columns=["Blade-1 Actual
Value Angle-B"])
model70 = sm.OLS(y_train,XtrainScaleddf70).fit()
print(model70.summary())

#model70 is final model now we do all other analysis and prediction

XtestScaled = sm.add_constant(XtestScaled)
XtestScaleddf = pd.DataFrame(XtestScaled,index=X_test.index, columns=xcols)

XtestScaleddf = XtestScaleddf[['Constant','Tower Acceleration
Normal','Converter Control Unit Reactive Power',
                               'Reactive Power','Moment D Direction','N-set
1','Particle Counter','Wind Deviation 1 seconds']]

OLSPredict = model70.predict(XtestScaleddf)

plt.figure(figsize=(16,8))
plt.plot(df['Timestamp'][len(y_train):],y_test['Power(kW)'], 'r',
label="Original Test data ")
plt.plot(df['Timestamp'][len(y_train):], OLSPredict, 'b', label="Predicted
data")
plt.xlabel("Time")
```

```
plt.ylabel("Power(kW)")
plt.legend(loc = 'lower right')
plt.title("Test Data v/s One Step Ahead Prediction : OLS Model")
plt.tight_layout()
plt.show()

rmseOLS = np.sqrt(mean_squared_error(y_test, OLSPredict))
print("The root mean squared error of OLS Model is = ",
np.round(rmseOLS,4))
print("The AIC and BIC scores of the model respectively are =
",np.round(model70.aic,4),"and ",np.round(model70.bic,4))
print("The R-Sqaured and Adjusted R-sqaured values of the model
respectively are = ",np.round(model70.rsquared,4),"and\n
",np.round(model70.rsquared_adj,4))
#COMPLETE T test
p_val_in_T_test = model70.pvalues
conf95 = 0.05
for i,var in enumerate(XtrainScaleddf70.columns):
      if p_val_in_T_test[i] < conf95:
            print(var," is statistically significant where p value of the "
, var ," is ",np.round(p_val_in_T_test[i],4),"\n")
      else:
            print(var + " is not statistically significant\n")
#COMPLETE F TEST
F_Test_Stat = model70.f_pvalue
columns = ', '.join(XtrainScaleddf70.columns.tolist())
print("The F Test value for the model 70(Final OLS Model) for \nWind
Turbine Power Prediction with features", columns,"is-\n",
np.round(F_Test_Stat,4))

acf = ACFPlot(model70.resid,48,"Residuals of OLS")
re = acf[(len(acf)//2):]
Q = len(y) * np.sum(np.square(re[1:]))
print("The Q value of the OLS Model Residuals is -", np.round(Q,4))

print("Mean of Residuals = ",np.mean(model70.resid),"and Variance of
residuals =",np.round(np.var(model70.resid),4))




#BASE MODELS
#AVERAGE METHOD
averagesubset = df[['Power(kW)']]
ytavg, yfavg = train_test_split(averagesubset, shuffle= False,
test_size=0.2)
yhstep = np.mean(ytavg.values)
yhstep = len(yfavg)*[yhstep]
print("Average Method - h Step prediction =", yhstep)
yhstep = pd.DataFrame(yhstep, columns = ['Power(kW)'], index=yfavg.index)


plt.figure(figsize=(16,8))
plt.plot(df['Timestamp'][len(ytavg):],yfavg['Power(kW)'], 'r',
label="Original Test data ")
plt.plot(df['Timestamp'][len(ytavg):], yhstep['Power(kW)'], 'b',
label="Predicted data")
plt.xlabel("Time")
plt.ylabel("Power(kW)")
plt.legend(loc = 'lower right')
plt.title("Test Data v/s h-Step Ahead Prediction- AVERAGE METHOD")
```

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

rmseAVG = np.sqrt(mean_squared_error(yfavg, yhstep))
print("The root mean squared error of AVERAGE Model is = ",
np.round(rmseAVG,4))



#NAIVE METHOD
naivesubset = df[['Power(kW)']]
ytnaive, yfnaive = train_test_split(naivesubset, shuffle= False,
test_size=0.2)
ynaivehstep = ytnaive.values[-1]
ynaivehstep = len(yfnaive)*[ynaivehstep[0]]
print("Naive Method - h Step prediction =", ynaivehstep)
ynaivehstep = pd.DataFrame(ynaivehstep, columns = ['Power(kW)'],
index=yfnaive.index)


plt.figure(figsize=(16,8))
plt.plot(df['Timestamp'][len(ytnaive):],yfnaive['Power(kW)'], 'r',
label="Original Test data ")
plt.plot(df['Timestamp'][len(ytnaive):], ynaivehstep['Power(kW)'], 'b',
label="Predicted data")
plt.xlabel("Time")
plt.ylabel("Power(kW)")
plt.legend(loc = 'lower right')
plt.title("Test Data v/s h-Step Ahead Prediction: NAIVE METHOD")
plt.tight_layout()
plt.show()

rmseNAIVE = np.sqrt(mean_squared_error(yfnaive, ynaivehstep))
print("The root mean squared error of NAIVE Model is = ",
np.round(rmseNAIVE,4))




#DRIFT METHOD
driftsubset = df[['Power(kW)']]
ytdrift, yfdrift = train_test_split(driftsubset, shuffle= False,
test_size=0.2)
ydrifthstep =[]
for i in range(1,len(yfdrift)):
    ydrifthstep.append((ytdrift.values[-1]+i*((ytdrift.values[-1]-
ytdrift.values[0])/(len(ytdrift.values)-1))))

ydrifthstep = [i[0] for i in ydrifthstep]

print("Drift Method - h Step prediction =", ydrifthstep)

yfdrift = yfdrift.iloc[1:,]
ydrifthstep = pd.DataFrame(ydrifthstep, columns = ['Power(kW)'],
index=yfdrift.index)


plt.figure(figsize=(16,8))
plt.plot(df['Timestamp'][len(ytdrift)+1:],yfdrift['Power(kW)'], 'r',
label="Original Test data")
```

```
plt.plot(df['Timestamp'][len(ytdrift)+1:], ydrifthstep['Power(kW)'], 'b',
label="Predicted data")
plt.xlabel("Time")
plt.ylabel("Power(kW)")
plt.legend(loc = 'lower right')
plt.title("Test Data v/s h-Step Ahead Prediction: DRIFT METHOD")
plt.tight_layout()
plt.show()

rmseDRIFT = np.sqrt(mean_squared_error(yfdrift, ydrifthstep))
print("The root mean squared error of DRIFT Model is = ",
np.round(rmseDRIFT,4))



#SES METHOD
SeSsubset = df[['Power(kW)']]
ytses, yfses = train_test_split(SeSsubset, shuffle= False, test_size=0.2)
alpha = 0.5
yses1step = [ytses.iloc[0].values.tolist()[0]]
for i in range(len(ytses)-1):
    yses1step.append(ytses.iloc[i].values.tolist()[0]*alpha+ (1-
alpha)*yses1step[i])

yseshstep = ytses.iloc[-1].values.tolist()[0]*alpha + (1-alpha)*yses1step[-
1]
yseshstep = len(yfses)*[yseshstep]
print("Simple Exponential Smoothing - h Step prediction =", yseshstep)


yseshstep = pd.DataFrame(yseshstep, columns = ['Power(kW)'],
index=yfses.index)


plt.figure(figsize=(16,8))
plt.plot(df['Timestamp'][len(ytses):],yfses['Power(kW)'], 'r',
label="Original Test data ")
plt.plot(df['Timestamp'][len(ytses):],yseshstep['Power(kW)'], 'b',
label="Predicted data")
plt.xlabel("Time")
plt.ylabel("Power(kW)")
plt.legend(loc = 'lower right')
plt.title("Test Data v/s h-Step Ahead Prediction: Simple Exponential
Smoothing Method")
plt.tight_layout()
plt.show()

rmseSES = np.sqrt(mean_squared_error(yfses, yseshstep))
print("The root mean squared error of SES Model is = ",
np.round(rmseSES,4))



#TRANSFORMED DATA 2 - 24 diff seasonal on top of 1 diff for GPAC and LM
Algorithm use later
#subsetdf = norderdiff(subsetdf,'Power(kW)',24)
subsetdf = norderdiff(subsetdf,'Power(kW)_diff_1',24)
subsetdf = subsetdf.dropna()
subsetdf = subsetdf.reset_index()
subsetdf = subsetdf.iloc[:,-1]
subsetdf = pd.DataFrame(subsetdf)
```

```
#GPAC
ry = sm.tsa.stattools.acf(subsetdf, nlags=100, fft=False)
ry1 = ry[::-1]
ry2 = np.concatenate((np.reshape(ry1, 101), ry[1:]))
ry3 = pd.Series(ry2)
calGPAC(ry3,30,30)

ACF_PACF_Plot(subsetdf,200)

theta = np.zeros((24))

paramEst(subsetdf,theta,len(subsetdf),0,24)
paramEst(subsetdf,theta,len(subsetdf),24,0)


#ARIMAmodel = ARIMA(subsetdftrain, order=(0, 0, 1), trend='n').fit() # d
and D not to be filled when your data is differenced
SARIMAmodel = sm.tsa.SARIMAX(y_train,
order=(0,1,1),seasonal_order=(0,1,1,24), trend='n').fit()
print(SARIMAmodel.summary())
y_model_hat = SARIMAmodel.predict(start = 1, end = len(y_train)-1)#--------
----param match but value shit in prediction
# y_train_forresid = y_train["Power(kW)"].iloc[1:].squeeze()
# y_model_hat.index = y_train_forresid.index
res_eSARIMAX = (y_train['Power(kW)'].iloc[1:])- y_model_hat
acf = ACFPlot(res_eSARIMAX.values,200,'residuals')
acf = pd.Series(acf)
calGPAC(acf,30,30)

y_hat_h_step = SARIMAmodel.forecast(steps=(len(y_test)))
y_hat_h_step.index = y_test.index

fore_errorSARIMAX = y_test['Power(kW)'] - y_hat_h_step

varResidSARIMAX = SARIMAmodel.resid.var()
varforecastSARIMAX = np.var(fore_errorSARIMAX)

print("variance of residual error is =",np.round(varResidSARIMAX,4))
print("variance of forecast error is =",np.round(varforecastSARIMAX,4))
re = acf[(len(acf)//2):]
Q = len(y) * np.sum(np.square(re[1:]))
DOF = 48 - 0 - 1
alfa = 0.01
chi_critical = chi2.ppf(1 - alfa, DOF)
print('Chi critical:', chi_critical)
print('Q Value:', Q)
print('Alfa value for 99% accuracy:', alfa)
if Q < chi_critical:
    print("The residual is white ")
else:
    print("The residual is NOT white ")

plt.figure(figsize=(16,8))
plt.plot(df['Timestamp'][:len(y_train)],y_train, 'r', label="Original Train
data ")
plt.plot(df['Timestamp'][:len(y_train)-1],y_model_hat, 'b', label="Model
Fit data")
plt.xlabel("Time")
plt.ylabel("Power(kW)")
```

```
plt.legend()
plt.title(" Train versus One Step Prediction")
plt.tight_layout()
plt.show()

plt.figure(figsize=(16,8))
plt.plot(df['Timestamp'][len(y_train):],y_test['Power(kW)'], 'r',
label="Original Test data ")
plt.plot(df['Timestamp'][len(y_train):], y_hat_h_step, 'b',
label="Predicted data")
plt.xlabel("Time")
plt.ylabel("Power(kW)")
plt.legend()
plt.title("Test Data v/s h-Step Ahead Prediction: SARIMA Model")
plt.tight_layout()
plt.show()

rmseSARIMAX = np.sqrt(mean_squared_error(y_test, y_hat_h_step))
print("The root mean squared error of SARIMA Model is = ",
np.round(rmseSARIMAX,4))


#LSTM
import tensorflow as tf
physical_devices = tf.config.list_physical_devices('GPU')
#tf.config.experimental.set_memory_growth(physical_devices[0],True)
CUDA_VISIBLE_DEVICES = '0,1'
from tensorflow.keras import Sequential
# from keras.layers import CuDNNLSTM
from tensorflow.keras.layers import Dense, LSTM, Dropout  # ,CuDNNLSTM
from tensorflow.keras.preprocessing.sequence import TimeseriesGenerator
from tensorflow.keras.optimizers import Adam


scaler = StandardScaler()
scaler = scaler.fit(df.iloc[:,1:])
dfscaled = scaler.transform(df.iloc[:,1:])

trainsize = int(len(df)*0.80)
testsize = int(len(df)*0.20)

traindata = dfscaled[:trainsize,:]
subsetdflen = len(df[['Power(kW)']])
npast = subsetdflen-trainsize
time_step = 24 # because of seasonality pattern in my data
Xtrain, ytrain = [],[]

for i in range(npast, len(traindata)):
    Xtrain.append(traindata[i - time_step:i, 0:traindata.shape[1]-1])
    ytrain.append(traindata[i, traindata.shape[1]-1])

train_X = np.array(Xtrain)
train_y = np.array(ytrain)


model = Sequential()
model.add(LSTM(64, activation="relu", input_shape=(train_X.shape[1],
train_X.shape[2]), return_sequences=True))
model.add(LSTM(50, return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(1))
```

```
model.compile(optimizer='adam', loss='mse')
model.summary()
history = model.fit(train_X, train_y,  batch_size=32, validation_split=.1,
epochs=30, verbose=1)
plt.figure()
plt.plot(history.history['loss'], 'r', label='Training loss')
plt.plot(history.history['val_loss'], 'b', label='Validation loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.show()

testdata = dfscaled[trainsize - time_step:,:]
Xtest = []
ytest = df.iloc[trainsize:,-1]


for i in range(time_step, len(testdata)):
    Xtest.append(testdata[i-time_step:i, 0:traindata.shape[1]-1])

Xtest = np.array(Xtest)
predictions = model.predict(Xtest)
forecast_copies = np.repeat(predictions, traindata.shape[1], axis=-1)
predictions = scaler.inverse_transform(forecast_copies)[:, -1]


train = df.iloc[:trainsize]
valid = df.iloc[trainsize:]
valid['predictions'] = predictions

fig = plt.figure(figsize=(16, 8))
ax = fig.add_subplot(111)
ax.set_title('Power Output prediction of Wind turbine using LSTM')
ax.set_xlabel("Date", fontsize=18)
ax.set_ylabel('Power(kW)')
#ax.plot(df['Timestamp'][:len(train)],train['Power(kW)'], 'blue')
ax.plot(df['Timestamp'][len(train):], valid['Power(kW)'],  'orange')
ax.plot(df['Timestamp'][len(train):],valid['predictions'], 'Green')
ax.legend(["Train", "Val", "predictions"], loc="lower right", fontsize=18)
ax.grid()
plt.show()

rmseLSTM = np.sqrt(mean_squared_error(ytest, valid[['predictions']]))
print("The root mean squared error of long Short Term Memory Neural Network
is =\n ", np.round(rmseLSTM,4))
diffminmax = np.max(ytest) - np.min(ytest)
accuracy = (1-(rmseLSTM/diffminmax))*100
print("The accuracy of long Short Term Memory Neural Network is =\n ",
np.round(accuracy,4))


flist = ['Gearbox_T1_Intermediate_Speed_Shaft_Temperature',
 'Tower Acceleration Lateral',
 'Temperature Shaft Bearing-1',
 'Temperature Axis Box-1',
 'Voltage B-N',
 'Temperature Battery Box-3',
 'Internal Power Limit',
 'Temperature Ambient',
 'Pitch Offset-1 Asymmetric Load Controller',
 'Turbine State']
```

```
Xhailmarydf = X_train.loc[:,flist]
Xhailmarydftestingforaccuracy = X_test.loc[:,flist]
ytestingforRMSE = y_test.loc[:,'Power(kW)']



GBRModel =
GradientBoostingRegressor(n_estimators=150,max_depth=4,learning_rate = 0.2
,random_state=44)
GBRModel.fit(Xhailmarydf, y_train)
ypredgbr =GBRModel.predict(Xhailmarydftestingforaccuracy)
print('Gradient Boosting Regression Model Test Score/R2 value is = ' ,
np.round(GBRModel.score(Xhailmarydftestingforaccuracy, y_test),4))
print('The Adjusted R2 value for Gradient Boosting Regression is = ',
np.round(r2_score(ytestingforRMSE, ypredgbr),4))
RMSEGBR = np.sqrt(mean_squared_error(y_test, ypredgbr))
print('The RMSE for Gradient Boosting Regression Model is =',
np.round(RMSEGBR,4))

plt.figure(figsize=(16,8))
plt.plot(df['Timestamp'][len(y_train):],ytestingforRMSE, 'r',
label="Original Test data ")
plt.plot(df['Timestamp'][len(y_train):], ypredgbr, 'b', label="Predicted
data")
plt.xlabel("Time")
plt.ylabel("Power(kW)")
plt.legend()
plt.title("Test Data v/s h-Step Ahead Prediction: Gradient Boosting
Regression Model")
plt.tight_layout()
plt.show()



#https://www.makeareadme.com/
#https://towardsdatascience.com/all-you-need-to-know-about-gradient-
boosting-algorithm-part-1-regression-2520a34a502
```

References

1. Lecture pdf, Handwritten notes and in class recordings for DATS-6313 Spring 2023 Semester (blackboard)

2. https://www.makeareadme.com/

3. https://towardsdatascience.com/all-you-need-to-know-about-gradient-boosting-algorithm-part-1-regression-2520a34a502

4. https://statisticsbyjim.com/regression/interpret-constant-y-intercept-regression/#:~:text=The%20reason%20I%20just%20discussed,the%20constant%20to%20equal%20zero.