

WIND POWER FORECASTING BASED ON WIND SPEED DATA USING MACHINE LEARNING ALGORITHMS

PROJECT REPORT

submitted by

ARAVIND PRAMOD (TVE16IE017)

DHIRAJ D S (TVE16IE028)

MANU P V (TVE16IE042)

ROHAN DAVID (TVE16IE053)

to

the APJ Abdul Kalam Technological University

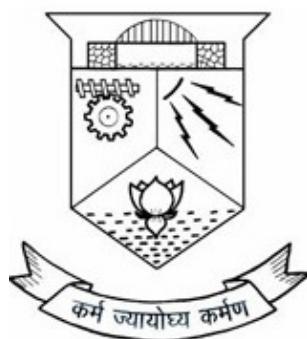
in partial fulfillment of the requirements for the award of the Degree

of

Bachelor of Technology

in

Industrial Engineering



Department of Mechanical Engineering

College of Engineering, Trivandrum

Kerala

July 14, 2020

DECLARATION

We undersigned hereby declare that the project report **Wind Power Forecasting based on Wind Speed Data using Machine Learning Algorithms**, submitted for partial fulfillment of the requirements for the award of degree of Master of Technology of the APJ Abdul Kalam Technological University, Kerala is a bonafide work done by us under supervision of **Dr. Vinod M.** This submission represents our ideas in our own words and where ideas or words of others have been included, we have adequately and accurately cited and referenced the original sources. We also declare that we have adhered to ethics of academic honesty and integrity and have not misrepresented or fabricated any data or idea or fact or source in our submission. We understand that any violation of the above will be a cause for disciplinary action by the institute and/or the University and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been obtained. This report has not been previously formed the basis for the award of any degree, diploma or similar title of any other University.

Place: Trivandrum

Date: July 14, 2020

Aravind Pramod

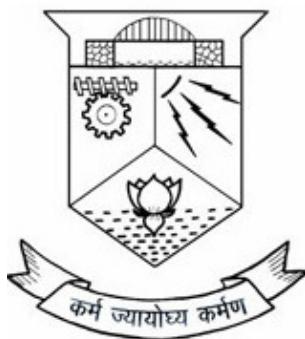
Dhiraj D S

Manu P V

Rohan David

DEPARTMENT OF MECHANICAL ENGINEERING

COLLEGE OF ENGINEERING, TRIVANDRUM



CERTIFICATE

This is to certify that the report entitled "**WIND POWER FORECASTING BASED ON WIND SPEED DATA USING MACHINE LEARNING ALGORITHMS**", submitted by **Aravind Pramod, Dhiraj D S, Manu P V, Rohan David** to the **APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY** in partial fulfilment of the requirements for the award of the degree of Bachelor of Technology in Mechanical Engineering is a bonafide record of the project presented by them under our guidance and supervision. This report in any form has not been submitted to any other University or Institute for any purpose.

Dr. Vinod M

Designation

Dr. Anil Lal

Designation

ACKNOWLEDGEMENT

We express our sense of gratitude to our respected guide Dr. Vinod M for his valuable help, guidance and encouragement he has given us in completing the project.

We would like to thank our Staff Advisor Dr. K Sunil Kumar for his helpful feedback and timely assistance.

We are grateful to our Principal, Dr. Jiji C V for providing congenial environment to work in.

We are also thankful to the Head of the Department of Mechanical Engineering, Dr. Anil Lal for his overwhelming support and advice on the project.

We are also thankful to other faculty members for their support and encouragement.

We would also like to express our gratitude to all our friends who have helped us during the work with their inspiration and cooperation.

Lastly, we would like thank God Almighty for showering abundant blessings upon us

Aravind Pramod

Dhiraj D S

Manu P V

Rohan David

ABSTRACT

Wind Energy is a significant and eligible source that has the potential for producing energy in a continuous and sustainable energy sources. However this has several challenges. The installation cost are high and machinery cant be installed every where. In this work wind power forecasting is performed using previous recorded data. The data set used for this work contains six years hourly data of the features namely wind speed, air temperature, pressure ,wind direction and generated power. The target variable is generated wind power. Making use of above features power generated in the future has to be forecasted. Exploratory data analysis was employed to gain overall insights , null value detection , duplicate value detection and outlier detection. Angle based outlier detection .Feature bagging , Histogram based outlier detection, Isolation forest and K-Nearest Neighbor methods were employed for detecting outliers in the multidimensional data. Throughout the work various analysis were performed in order to obtain conclusive inferences. Feature engineering was done in order find the feature that explains generated power in the best way. Pearson correlation coefficient , joint mutual information and mean square error contribution obtained from regression fit were used to determine the feature that could explain generated wind power. It is wind speed which was highly correlated with generated power , it was proved statistically. Forecasting of wind speed needs to be done using past wind speed data for which the behavior of wind speed is analyzed. Random walk test was performed to check randomness. Visual test, Kwiatkowski Phillips Schmidt Shin test for stationarity and Surrogate test for non linearity was performed to validate the non linear behavior of the wind speed data. Next tests for chaoticity was performed. Optimal lags were found using two methods namely using autocorrelation function and mutual information function. Embedding dimensions were obtained using Caos method. Phase space reconstruction of wind speed data was done. Approximate entropy, correlation dimension and maximum Lyapunov exponent are estimated. From maximum Lyapunov exponent wind speeds chaotic nature was inferred. Long short term memory neural networks , Sequence to sequence unidirectional single layered long short term memory neural network , Sequence to sequence unidirectional double layered long short term memory neural network , Sequence to sequence bidirectional single layered long

short term memory neural network and Sequence to sequence bidirectional double layered long short term memory neural network algorithms were employed to predict future wind speeds. Comparison studies were performed with respect to prediction window as well as varying algorithms. For power prediction a stacked model consisting of Light gradient boosted algorithm, extreme gradient boost algorithm and random forest algorithm was made. Here also performance measures were analyzed by varying the test train split ratios. A new algorithm is proposed as a combination of stacked model and wind speed prediction model. The new algorithm predicts power generated in future. Performance measures of the new algorithm is analysed with respect to prediction window and parent wind speed prediction models.

Keywords : wind power prediction, long short-term memory neural network, chaotic, model stacking.

Contents

List of Figures	x
List of Tables	xvii
Abbreviations	xviii
1 Preamble	1
1.1 Introduction	1
1.2 Problem in hand	2
1.3 Motivation	2
1.4 Methodology	3
1.5 Scope of the work	4
2 Literature Review	6
2.1 Literature Review on Exploratory Data Analysis	6
2.2 Literature Review on Test for Randomness	7
2.3 Literature Review on Test for Non-Linearity	7
2.4 Literature Review on Test for Chaoticity	8
2.5 Literature Review on Chaotic Wind Speed Prediction	11
2.6 Literature Review on Power Prediction using Stacked Model	13
3 Exploratory Data Analysis	14
3.1 Data Preview	14
3.2 Box Plots	14
3.3 Scatter Plots	15
3.4 Duplicate Entries	15
3.5 Null Values	15

3.6	Outlier Detection for Multi Dimensional Data	16
3.6.1	Angle Based Outlier Detection(ABOD)	16
3.6.2	Feature Bagging	17
3.6.3	Histogram-Based Outlier Detection	17
3.6.4	Isolation Forest	18
3.6.5	K-Nearest Neighbors (KNN)	19
3.7	Results and Discussion	20
3.7.1	Preview of Dataset	20
3.7.2	Boxplots for Dataset	20
3.7.3	Presence of Duplicate Entries	22
3.7.4	Presence of Null Values	23
3.7.5	Scatter Plots	24
3.7.6	Outlier Detection for Multi Dimensional Data	28
4	Feature Engineering	54
4.1	Pearson Correlation Coefficient	54
4.2	Joint Mutual Information between Power and Other Input Variables	54
4.3	Mean Square Error Contribution by Variables	55
4.4	Results and Discussion	55
4.4.1	Pearson Correlation Coefficient	55
4.4.2	Joint Mutual Information between Power and Other Input Variables . . .	60
4.4.3	Mean Square Error Contribution by Variables	65
5	Test for Randomness	71
5.1	Random Walk	71
5.2	Differencing of Time Series	72
5.3	Tests for Randomness	72
5.3.1	Visual Test	72
5.3.2	Differencing of Time Series	73
5.3.3	Comparison of Standard Deviations	74

6 Test for Non-linearity	76
6.1 Non-linearity	76
6.2 Stationarity	76
6.3 Visual Test	77
6.4 Autocorrelation Function	77
6.5 Partial Autocorrelation Function	77
6.6 KPSS (Kwiatkowski-Phillips-Schmidt-Shin) test	78
6.7 Surrogate Test	78
6.8 Results and Discussion	79
6.8.1 Visual Test	79
6.8.2 KPSS (Kwiatkowski-Phillips-Schmidt-Shin) test	81
6.8.3 Surrogate Test	82
7 Test for Chaoticity	84
7.1 Chaos Theory	84
7.2 Conditions for determining a dynamical system to be chaotic	85
7.3 Weak and strong sensitive dependence	86
7.4 Attractors	87
7.5 Lyapunov time	88
7.6 Phase Space Reconstruction	88
7.7 Estimation of lag using Autocorrelation function	90
7.8 Estimation of lag using Mutual Information function	90
7.9 Determining embedding dimensions for Phase Space Reconstruction	92
7.10 Approximate Entropy	94
7.11 Correlation Dimension	95
7.12 Lyapunov Exponent	95
7.13 Results and discussion	96
7.13.1 Estimation of lag using Autocorrelation Function	96
7.13.2 Estimation of lag using Mutual information function	97
7.13.3 Estimation of embedding dimension using Cao's Method	97

7.13.4	Phase space reconstruction	98
7.13.5	Estimation of Approximate Entropy	100
7.13.6	Estimation of Correlation Dimension	101
7.13.7	Estimation of Maximal Lyapunov Exponent	102
8	Chaotic Wind Speed Prediction	104
8.1	LSTM	104
8.1.1	Architecture of LSTM	104
8.1.1.1	Forget Gate	105
8.1.1.2	Input Gate	106
8.1.1.3	Cell State	107
8.1.1.4	Output Gate	108
8.1.2	Loss Function	109
8.1.3	Optimizer	110
8.2	Sequence to Sequence Models	111
8.2.1	Activation Functions	113
8.2.1.1	Sigmoid Function	113
8.2.1.2	Tangent Hyperbolic Function	114
8.2.1.3	Softmax Function	114
8.2.2	Model Architecture	114
8.2.2.1	With Unidirectional Long Short Term Memory neural networks as encoder	114
8.2.2.2	With Bidirectional Long Short Term Neural Networks as encoders	115
8.2.2.3	With Stacked Unidirectional Long Short Term Memory Neural Network as encoder	116
8.2.2.4	With Stacked Bidirectional Long Short Term Memory Neural Network as encoder	116
8.3	Results and Discussion	116
8.3.1	LSTM Models	116
8.3.1.1	LSTM Model for 12 Hours	116

8.3.1.2	LSTM Model for 24 Hours	117
8.3.1.3	LSTM Model for 2 Days	118
8.3.1.4	LSTM Model for 1 Week	119
8.3.1.5	LSTM Model for 1 Month	120
8.3.2	Sequence to Sequence Models	121
8.3.2.1	Single Layer Model with Unidirectional LSTM as encoder	122
8.3.2.2	Double Layer Model with Unidirectional LSTM as encoder	124
8.3.2.3	Single Layer Model with Bidirectional LSTM as encoder	125
8.3.2.4	Double Layer Model with Bidirectional LSTM as encoder	127
9	Power Prediction using Stacked Model	129
9.1	Model Stacking	129
9.1.1	Stacking Process	130
9.1.2	Layers of Stacking	131
9.1.3	Stack Of Models	131
9.1.4	Implementation	131
9.1.4.1	Grid Search	131
9.1.4.2	LightGBM	132
9.1.4.3	XGBoost	134
9.1.4.4	Random Forest	136
9.2	Results and Dicussion	137
9.2.1	Stacked Model with 10 percent Test Volume	137
9.2.2	Stacked Model with 20 percent Test Volume	138
9.2.3	Stacked Model with 30 percent Test Volume	140
10	Proposed Algorithm	141
10.1	Proposed Power Prediction Algorithm	141
10.2	Results and Discussion	143
10.2.1	Using Long Short Term Neural Networks Algorithm to predict Speed	143
10.2.1.1	For Prediction Window = 12 Hours	143
10.2.1.2	For Prediction Window = 24 Hours	144

10.2.1.3 For Prediction Window = 2 days	145
10.2.1.4 For Prediction Window = 1 week	147
10.2.1.5 For Prediction Window = 1 month	148
10.2.2 Using Unidirectional Sequence to sequence Long Short Term Neural Network to Predict Power	149
10.2.2.1 Predictions for Unidirectional Single Layer Seqeunce	149
10.2.2.2 Predictions for Unidirectional Double Layer Seqeunce	150
10.2.3 Using Bidirectional Sequence to sequence Long Short Term Neural Network to Predict Power	151
10.2.3.1 Predictions for Bidirectional Single Layer Seqeunce	151
10.2.3.2 Predictions for Bidirectional Double Layer Seqeunce	153
11 Observations and Results	155
11.1 Exploratory Data Analysis	155
11.2 Feature Engineering	156
11.3 Test for Randomness	159
11.4 Test for Non-linearity	159
11.5 Test for Chaoticity	160
11.6 Chaotic Wind Speed Prediction	160
11.6.1 LSTM Models for different Prediction Windows	160
11.6.2 Sequence to Sequence Models	168
11.7 Power Prediction using Stacked Model	174
11.8 Proposed Algorithm	179
11.8.1 Stacked Models with predicted wind speed from LSTM models as Input	180
11.8.2 Stacked Models with predicted wind speed from Sequence to Sequence models as Input	186
12 Conclusion and Scope of Future Works	193
References	195
Appendix	198

List of Figures

3.1	Datset Preview	20
3.2	Boxplots for the dataset	21
3.3	Feature Statistics from Boxplots	21
3.4	Duplicate Entries	23
3.5	Null Values	23
3.6	Air Temperature vs Hour	24
3.7	Pressure v/s Hour	25
3.8	Wind speed v/s Hour	26
3.9	Wind Direction v/s Hour	27
3.10	Power Generated by System v/s Hour	28
3.11	ABOD considering Air temperature v/s Hour	29
3.12	ABOD considering Pressure v/s Hour	30
3.13	ABOD considering Wind speed v/s Hour	31
3.14	ABOD considering Wind direction v/s Hour	32
3.15	ABOD considering Power generated by system v/s Hour	33
3.16	Feature Bagging considering Air temperature v/s Hour	34
3.17	Feature Bagging considering Pressure v/s Hour	35
3.18	Feature Bagging considering Wind speed v/s Hour	36
3.19	Feature Bagging considering Wind direction v/s Hour	37
3.20	Feature Bagging considering Power generated by system v/s Hour	38
3.21	HBOS considering Air temperature v/s Hour	39
3.22	HBOS considering Pressure v/s Hour	40

3.23 HBOS considering Wind speed v/s Hour	41
3.24 HBOS considering Wind direction v/s Hour	42
3.25 HBOS considering Power generated by system v/s Hour v/s Hour	43
3.26 Isolation Forest considering Air temperature v/s Hour	44
3.27 Isolation Forest considering Pressure v/s Hour	45
3.28 Isolation Forest considering Wind speed v/s Hour	46
3.29 Isolation Forest considering Wind direction v/s Hour	47
3.30 Isolation Forest considering Power generated by system v/s Hour	48
3.31 KNN considering Wind direction v/s Hour	49
3.32 KNN considering Pressure v/s Hour	50
3.33 KNN considering Wind speed v/s Hour	51
3.34 KNN considering Wind direction v/s Hour	52
3.35 KNN considering Power generated by system v/s Hour	53
 4.1 Heat Map - 1 year data	55
4.2 Heat Map - 2 years data	56
4.3 Heat Map - 3 years data	57
4.4 Heat Map - 4 years data	58
4.5 Heat Map - 5 years data	59
4.6 Heat Map - 6 years data	60
4.7 Joint Mutual Information - test size 10%	61
4.8 Joint Mutual Information - test size 20%	62
4.9 Joint Mutual Information - test size 30%	63
4.10 Joint Mutual Information - test size 40%	64
4.11 Joint Mutual Information - test size 50%	65
4.12 MSE contribution - test size 10%	66
4.13 MSE contribution - test size 20%	67
4.14 MSE contribution - test size 30%	68
4.15 MSE contribution - test size 40%	69
4.16 MSE contribution - test size 50%	70

5.1	Random walk nature - wind speed vs time(in hour)	73
5.2	Stationary Behaviour of Differenced Series	73
5.3	Comparison of Series Statistics	75
6.1	Autocorrelation vs lag	79
6.2	Partial Autocorrelation vs lag	80
6.3	Logarithmic transformation of Autocorrelation vs lag	80
6.4	Logarithmic transformation of Partial Autocorrelation vs lag	81
6.5	Surrogate data testing	82
7.1	Double pendulum	84
7.2	Billiards game	85
7.3	Attractor	87
7.4	Autocorrelation function vs lag	96
7.5	Average Mutual Information function vs lag	97
7.6	Plot of $E_1(d)$ and $E_2(d)$ vs dimension(d) for lag = 11	98
7.7	Plot of $E_1(d)$ and $E_2(d)$ vs dimension(d) for lag = 1	98
7.8	Phase space reconstruction for lag = 1 and embedding dimension = 11	99
7.9	Phase space reconstruction for lag = 11 and embedding dimension = 11	99
7.10	Approximate Entropy for lag = 1 and embedding dimension = 11	100
7.11	Approximate Entropy for lag = 11 and embedding dimension = 11	100
7.12	Correlation Dimension for lag = 1 and embedding dimension = 11	101
7.13	Correlation Dimension for lag = 11 and embedding dimension = 11	101
7.14	Plot of divergence for lag = 1 and correlation dimension = 11	102
7.15	Plot of divergence on a regression line for lag = 1 and correlation dimension = 11	102
7.16	Plot of divergence for lag = 11 and correlation dimension = 11	103
7.17	Plot of divergence on a regression line for lag = 11 and correlation dimension = 11	103
8.1	LSTMrep	105
8.2	LSTM forget gate	106
8.3	LSTM input gate	107

8.4	LSTM cell state	108
8.5	LSTM output gate	109
8.6	Model Architecture with Unidirectional LSTM Neural Networks as Encoder . .	114
8.7	Model Architecture with Bidirectional LSTM Neural Networks as Encoder . .	115
8.8	Model Architecture with Stacked Unidirectional LSTM Neural Networks as Encoder	116
8.9	Plot between predicted and actual Wind speeds for 12 Hours	117
8.10	Plot between predicted and actual Wind speeds for 24 Hours	118
8.11	Plot between predicted and actual Wind speeds for 2 Days	119
8.12	Plot between predicted and actual Wind speeds for 1 Week	120
8.13	Plot between predicted and actual Wind speeds for 1 Month	121
8.14	Plot between predicted and actual Wind speeds in Unidirectional LSTM . . .	123
8.15	Plot between predicted and actual Wind speeds in Stacked Unidirectional LSTM	124
8.16	Plot between predicted and actual Wind speeds in Bidirectional LSTM	126
8.17	Plot between predicted and actual Wind speeds in Stacked Bidirectional LSTM	127
9.1	Stacking Process	130
9.2	Plot between predicted and actual Power in Stacked Model - 10 percent test volume	138
9.3	Plot between predicted and actual Power in Stacked Model - 20 percent test volume	139
9.4	Plot between predicted and actual Power in Stacked Model - 30 percent test volume	140
10.1	Plot between predicted and actual Power for 12 Hours	143
10.2	Plot between predicted and actual Power for 24 Hours	144
10.3	Plot between predicted and actual Power for 2 days	146
10.4	Plot between predicted and actual Power for 1 week	147
10.5	Plot between predicted and actual Power for 1 month	148
10.6	Predictions from Unidirectional Single Layer Sequence to Sequence Model . . .	149

10.7 Predictions from Unidirectional Double Layer Sequence to Sequence Model . . .	150
10.8 Predictions from Bidirectional Single Layer Sequence to Sequence Model . . .	152
10.9 Predictions from Bdirectional Double Layer Sequence to Sequence Model . . .	153
11.1 Plot between data size and correlation coefficient of independent variables and power	157
11.2 Average Mutual Information value for different Variables	158
11.3 Mean Square Error values for different Variables	158
11.4 Mean Absolute Percent Error values for LSTM Models	161
11.5 Expected Variance values for LSTM Models	162
11.6 Maximum Error values for LSTM Models	162
11.7 Mean Absolute Error values for LSTM Models	163
11.8 Mean Squared Error values for LSTM Models	164
11.9 Root Mean Squared Error values for LSTM Models	164
11.10 Mean Squared Log Error values for LSTM Models	165
11.11 Median Absolute Error values for LSTM Models	165
11.12 R ² score values for LSTM Models	166
11.13 R ² score variance weighted values for LSTM Models	166
11.14 Mean Poisson values for LSTM Models	167
11.15 Mean Gamma values for LSTM Models	167
11.16 Mean Absolute Percent Error values for Sequence Models	168
11.17 Expected Variance values for Sequence Models	169
11.18 Maximum Error values for Sequence Models	169
11.19 Mean Absolute Error values for Sequence Models	170
11.20 Mean Squared Error values for Sequence Models	170
11.21 Root Mean Squared Error values for Sequence Models	171
11.22 Mean Squared Log Error values for Sequence Models	171
11.23 Median Absolute Error values for Sequence Models	172
11.24 R ² score values for Sequence Models	172
11.25 R ² score variance weighted values for Sequence Models	173

11.26	Mean Poisson values for Sequence Models	173
11.27	Expected Variance values for Stacked Models	174
11.28	Maximum Error values for Stacked Models	175
11.29	Mean Absolute Error values for Stacked Models	175
11.30	Mean Squared Error values for Stacked Models	176
11.31	Root Mean Squared Error values for Stacked Models	176
11.32	Mean Squared Log Error values for Stacked Models	177
11.33	Median Absolute Error values for Stacked Models	177
11.34	R ² score values for Stacked Models	178
11.35	R ² score variance weighted values for Stacked Models	178
11.36	Mean Poisson values for Stacked Models	179
11.37	Expected Variance values for Stacked Models with predicted wind speed from LSTM models as Input	180
11.38	Maximum Error values for Stacked Models with predicted wind speed from LSTM models as Input	181
11.39	Mean Absolute Error values for Stacked Models with predicted wind speed from LSTM models as Input	182
11.40	Mean Squared Error values for Stacked Models with predicted wind speed from LSTM models as Input	183
11.41	Root Mean Squared Error values for Stacked Models with predicted wind speed from LSTM models as Input	183
11.42	Median Absolute Error values for Stacked Models with predicted wind speed from LSTM models as Input	184
11.43	R ² score values for Stacked Models with predicted wind speed from LSTM models as Input	184
11.44	R ² score variance weighted values for Stacked Models with predicted wind speed from LSTM models as Input	185
11.45	Mean Poisson values for Stacked Models with predicted wind speed from LSTM models as Input	186

11.46	Expected Variance values for Stacked Models with predicted wind speed from Sequence to Sequence models as Input	187
11.47	Maximum Error values for Stacked Models with predicted wind speed from Sequence to Sequence models as Input	188
11.48	Mean Absolute Error values for Stacked Models with predicted wind speed from Sequence to Sequence models as Input	188
11.49	Mean Squared Error values for Stacked Models with predicted wind speed from Sequence to Sequence models as Input	189
11.50	Root Mean Squared Error values for Stacked Models with predicted wind speed from Sequence to Sequence models as Input	189
11.51	Mean Squared Log Error values for Stacked Models with predicted wind speed from Sequence to Sequence models as Input	190
11.52	Median Absolute Error values for Stacked Models with predicted wind speed from Sequence to Sequence models as Input	190
11.53	R2 score values for Stacked Models with predicted wind speed from Sequence to Sequence models as Input	191
11.54	R2 score variance weighted values for Stacked Models with predicted wind speed from Sequence to Sequence models as Input	191
11.55	Mean Poisson values for Stacked Models with predicted wind speed from Sequence to Sequence models as Input	192

List of Tables

6.1 Critical Values	81
-------------------------------	----

ABBREVIATIONS

- ABOD - Angle based outlier detection
- ACF - Autocorrelation function
- ADF - Augmented Dick Fuller Test
- HBOS - Histogram based outlier detection
- KNN - K-Nearest neighbour
- KPSS - Kwiatkowski Phillips Schimidt Shin
- LSTM - Long Short Term Memory Neural Networks
- MSE - Mean squared error
- PACF - Partial Autocorrelation Function
- R2 Score - R squared score
- XGBoost - Extreme gradient boost

Chapter 1

Preamble

1.1 INTRODUCTION

Wind energy is growing in importance as a renewable energy source. It is clean, sustainable and is now used to power individual homes to entire networks. Moreover, wind farms are fast to build and are relatively cheaper than other forms of renewable energy sources. According to the Indian Wind Turbine Manufacturers' Association, India's current installed capacity is 87027.65 MW and this accounts for 43.4% of the total renewable energy produced in the country. Wind Power will also create employment opportunities with jobs available in manufacturing, installation, maintenance and support services. With more and more countries looking to exploit untapped potential, the relevance of wind energy is only set to increase in the coming years.

Wind is an intermittent energy source, which means that it is not possible to produce or supply it on demand. Wind energy is used in tandem with backups with other more conventional energy sources so that the demand can be met. This calls for extensive planning in every aspect of wind energy. Wind is variable and subject to fluctuations. Variability can range from seasonal variation to changes in pattern in very small time intervals. This leads to the thinking that wind behaviour is completely random in nature. Wind predictions in the long term are often unreliable and inaccurate.

Chaos theory is the branch of mathematics that deals with the study systems that may appear random, but is actually deterministic with high complexity that after a while it is near impossible to predict. Chaos is different from randomness because while it is impossible

to predict the next state in a random behaviour, it is perfectly possible to predict the future states of a chaotic system, given the initial conditions are available and accurate. If wind nature can be shown to be having chaotic behaviour, it is possible that the predictions can improve to a good extent., which will help in better planning and integration of wind energy with other conventional sources of energy and make the best possible use of it.

1.2 PROBLEM IN HAND

Basic problem to be addressed is the prediction of power that could be generated from past recorded data. Data set comprises of six years hourly wind speed, pressure, air temperature , direction and power generated entries. Investigation is to be done to ensure which feature explains power generated to greater extent. After the features are determined a least error affected forecasting of those features need to be done using a suitable method. Using those forecasted results power could be determined.

The main objectives of this work are:- Analyse the various features that suspects relationship with power generated from the data set.

Using the correlated features future power is to be predicted.

1.3 MOTIVATION

India's economy is sixth largest in the world in terms of nominal Gross Domestic Product and Purchasing Power Parity. India is the third largest energy consumer in the world and it's the backbone of Indian economy. Access to cheap energy is necessary for India's accelerated growth. The growing demand of energy in India is related with imports of conventional or non renewable sources of energy which is actually a threat to India's energy security. India has a high potential for generation of renewable energy from various sources. Wind energy in India has emerged as one of the fastest growing energy sources in the world. By this work it would be easier to find potential zones where ample amount of power could be produced there by strengthening nations energy securities.

A proper trade off formulation between various forms of energy in a state is difficult.

Inability of high performance predictions results ends up in loss in revenue as well as energy. A proper prediction of solar as well as wind energy would aid the state in the management of energy policies.

1.4 METHODOLOGY

1. Exploratory data analysis:
 - Overall insights to be obtained from data
 - Null value detection
 - Detection of duplicate entries
 - Outlier detection for multidimensional data using Angle Based Outlier Detection, Feature Bagging, Histogram Based Outlier Detection, Isolation Forest and K-Nearest Neighbours methods.
2. Feature Engineering to be performed in order to find features which best explains power conclusively. Methods to be used are Estimation of Pearson's correlation coefficient, Estimation of joint mutual information between power and rest of the features and Mean square error contribution by variables when power is predicted using regression algorithms.
3. Test for Randomness using random walk test.
4. Test for non linearity.
 - Visual test.
 - Kwiatkowski Phillips Schmidt Shin test for stationarity .
 - Surrogate test for non linearity.
5. Test for chaotic behaviour.
 - Estimation of lag using autocorrelation function.

- Estimation of lag using mutual information function.
- Estimation of embedding dimension using Cao's method.
- Phase space reconstruction.
- Estimation of approximate entropy.
- Estimation of correlation dimension.
- Estimation of maximal Lyapunov exponent.

6. Future chaotic wind speed prediction by following algorithms :

- LSTM with varying prediction window.
- Unidirectional single layered Sequence to Sequence LSTM.
- Unidirectional double layered Sequence to Sequence LSTM.
- Bidirectional single layered Sequence to Sequence LSTM.
- Bidirectional double layered Sequence to Sequence LSTM.

7. Stacked power prediction model comprising of Xgboost, Light GBM and Random forest.

- Performance measures to be analyzed by changing training data volumes.
8. Proposed a new algorithm which combines chaotic wind prediction model and stacked power prediction model in order to predict power generated in future from past wind speeds.

1.5 SCOPE OF THE WORK

This work aims to build more reliable and accurate wind predictions considering the nature of wind is chaotic. Chaotic nature of wind makes it very difficult to make good long term predictions. By using non-linearity and completely understanding the dynamics of wind, it should be possible to improve these forecasts.

Exploratory Data Analysis forms the foundation to understanding the nature of the wind data in hand. A through analysis of data to identify features by various criteria gives the rise to

the understanding that wind speed is the most suitable feature to use. The work calls for tests for checking randomness and then to confirm chaotic nature of wind data. To incorporate the chaotic nature to make predictions, LSTM architecture is used. Stacked model is then used to then make power predictions from wind speed as it is shown to have better accuracy.

Chapter 2

Literature Review

2.1 LITERATURE REVIEW ON EXPLORATORY DATA ANALYSIS

- H.P.Kriegel et al.(2008) proposed a novel-based approach termed Angle-Based Outlier Detection(ABOD) that alleviates the ‘curse of dimensionality’ in outlier detection problems by considering the variance of angles of a point considering a pair of points-compared to normal outlier detection techniques that just compare distances between points. In this paper, a comparison has been made between the ABOD and Local Outlier Factor (LOF) approach and it was found that the ABOD outperforms the LOF in terms of outlier detection effectiveness.
- Aleksandar et al. (2005) proposed a novel feature-bagging approach for outlier detection in high-dimensional data. This approach combines results from different outlier detection techniques that take different features as inputs. The results from different techniques in terms of outlier scores are combined to produce better quality outliers. This paper also stated the applicability of the framework to a set of any outlier detection techniques.
- Goldstein et al. (2012) presented a Histogram based outlier detection algorithm that assumes feature independence making the algorithm more faster than multivariate approaches in terms of precision. This paper also demonstrated that the Histogram based outlier detection algorithm ran faster than standard algorithms used for large datasets.

- F.T.Liu et al. (2012) proposed a different type of a model based technique that helps in explicitly identifying outliers rather than profiling normal data points in the conventional manner. An Isolation Tree structure is constructed to isolate every single instance and the collection of such trees ultimately form an Isolation Forest that detects outliers by considering the average path length from the root node to the termination node of each Isolation Tree involved in the task. In addition, an empirical comparison was done among the proposed method and other outlier detection techniques in terms of efficiency and processing time.
- Goldstein et al. (2012) demonstrated the use of a global-based approach and a local-based approach for outlier detection. It was also found that nearest neighbor based algorithms perform better than clustering based algorithms for most datasets.

2.2 LITERATURE REVIEW ON TEST FOR RANDOMNESS

- The term "random walk" was coined by Karl Pearson in 1905. In a letter to a journal, he wanted to know the probability that a man in a random walk has covered a particular distance in a given number of steps. His question was answered by Lord Rayleigh who had already worked in a similar model. Louis Bachelier had also developed a random walk model around the same time.

2.3 LITERATURE REVIEW ON TEST FOR NON-LINEARITY

- Bent Nielsen in Correlograms for non-stationary autoregressions(2006) used correlograms and partial correlograms as graphical descriptions of temporal dependence to test the stationarity of a given time series. He also provided two methods for computation, one based on autocorrelations and the other based on autocovariances.
- Paul S.P. Cowpertwait in Introductory time series using R(2009) used R programming language to constructing correlograms and partial correlograms of a time series to test stationarity of a process.

- Rob J Hyndman in Forecasting: Principles and Practice(2018) explained about stationarity and differencing of a time series. He also conducted a unit root test using R programming language to test the stationarity of a time series.
- Denis Kwiatkowski et al. in Testing the null hypothesis of stationarity against the alternative of a unit root(1992) proposed the KPSS test with the null hypothesis assuming the stationarity around a mean or a linear trend.
- Bart Hobijn et al. in Generalizations of the KPSS-test for stationarity. Statistica Neerlandica(2004) developed a simple and automatic test for stationarity against random walk alternatives by improving the size properties of the standard KPSS test for stationarity in highly autoregressive cases.
- James Theiler in Surrogate Data to Detect Nonlinearity in Time Series, in Nonlinear Modelling and Forecasting(1992) suggested the surrogate test where the null hypothesis assumes that the time series to be generated from a linear stochastic process. The method involves generating surrogate data sets consistent with this null hypothesis, and finally computing a discriminating statistic for the original and for each of the surrogate data sets to check the non-linearity of the time series.

2.4 LITERATURE REVIEW ON TEST FOR CHAOTICITY

- Robert L Devaney in Introduction to chaotic dynamical systems(1989) formulated the definitions to classify a dynamical system as chaotic.
- Wisdom Jack et al, in Chaotic Evolution of Solar System(1992) suggested the presence of chaotic systems within the nature. He also introduced the term Lyapunov time which shows the time until which the predictions of the system can be considered to be accurate.
- Haselblatt et al, in A First Course in Dynamics(2003) explained about topological entropy which is one of the indicators of the chaotic behaviour of a system. He also

explained about the quantitative measurement of complexity of a system with growth rate of periodic orbits.

- Lorenz Edwards in Deterministic Non-periodic flow(1963) suggested the chaos theory. He discovered the existence of Lorenz attractors and also formulated the 3 partial differential equations used to describe a Lorenz attractor called as Lorenz equations, both of which are later named after him.
- Lorenz Edwards in Atmospheric Predictability as revealed by Naturally Occurring Analogues(1969) suggested the butterfly effect, which proposes that the small changes in a chaotic system can lead to large consequences.
- Steven Strogatz in Nonlinear Dynamics and Chaos(1994) explained about bifurcations, chaos, fractals and their applications. He also explained in detail about attractors including Lorenz attractors and strange attractors.
- Peter Smith in Explaining Chaos(1998) explained the difference between systems with strong and weak sensitive dependence of the system on initial values.
- Geoff Boeing in Visual Analysis of Nonlinear Dynamical Systems: Chaos, Fractals, Self-Similarity and the Limits of Prediction(2016) provided several visualization methods to critically analyze and understand the behavior of nonlinear dynamical systems and to check the limits of the predictions of the behaviour of the system using Python packages.
- Boris P. Bezruchko et al. in Extracting Knowledge from Time Series(2010) explained about the construction of mathematical models for dynamical systems from time series. He explained about the usage of Lyapunov exponents in time series to find the chaotic nature of the time series and calculate the limits of predictions from the system.
- Henry D. I. Abarbanel e al. in Determining embedding dimension for phase-space reconstruction using a geometrical construction(1992) suggested the method of False Nearest Neighbors to estimate the embedding dimension of a given time series.

- Henry D. I. Abarbanel in The analysis of observed chaotic data in physical systems(1996) explained how to reconstruct the phase space of a given time series. He also provided methods to estimate the time delays using autocorrelation function and average mutual information function as well as estimate embedding dimension using false nearest neighbour method.
- Takens (1981) showed that a state space can be reconstructed from a scalar time series when there is no noise from a scalar time series by the methods of delays as the noise levels are constant for each delay component.
- J M. Martinerie et al. in Mutual information, strange attractors, and the optimal estimation of dimension(1992) conducted experiments to identify the optimal window for a given time series using mutual information and autocorrelation function. He also suggested that for noisy time series, mutual information function is better for estimating lag parameter.
- Pincus (1991) developed the idea of Approximate Entropy or ApEn to overcome practical difficulties while using the normal entropy calculations to quantify complexity of the system. This included problems such as requiring vast amounts of data to calculate entropy as well as the effect of noise.
- Kantz and Schreiber (2003) gave the first zero of the autocorrelation function heuristic for establishing time lag. Another heuristic that exists in literature is the first minimum of the auto-mutual information curve suggested by Fraser and Swinney (1986). Both heuristics are fundamentally based on minimizing redundancy.
- Cao (1997) give an algorithm for determining the minimum embedding dimension of a scalar time series based on the choice of optimal delay time.
- Asha Chelani et al. In Nonlinear Dynamical Characterization and Prediction of Ambient Nitrogen Dioxide Concentration. Water, Air, and Soil Pollution(2005) conducted prediction using neural network on a time series containing data regarding nitrogen dioxide concentration in India. The model involves the calculation of lag and embedding

dimension to conduct phase space reconstruction and later computing the correlation integral.

- James Theiler in Efficient algorithm for estimating the correlation dimension from a set of discrete points(1987) provided the algorithm for computing the correlation dimension of a strange attractor from a finite time series.
- Wahyu Caesarendra in An Application of Nonlinear Feature Extraction – A Case Study for Low Speed Slewing Bearing Condition Monitoring and Prognosis(2013) used non linear methods of feature selection on a time series including estimation of largest Lyapunov exponent, estimation of correlation dimension and estimation of approximate entropy.

2.5 LITERATURE REVIEW ON CHAOTIC WIND SPEED PREDICTION

- Hasim Sak et al in Short term memory recurrent neural network architecture for large scale acoustic modeling shows that a two layered deep LSTM RNN can yield higher performances. Here more usage of model parameters are done thereby it out performs a deep feed forward neural networks having an order of magnitude more parameters. Here a computationally efficient method of training is also introduced.
- Chigozie Enyinna Nwankpa et al in Activation functions :Comparison of trends in practice and research for deep learning (2018) presents a survey on existing activation functions used in deep learning applications. It also highlights the recent use of activation functions in a wide range of scenarios.
- Yao Cheng et al in Power LSTM : Power demand forecasting using long short term memory neural networks predicted power demand using long short term memory neural networks. Long short term memory neural networks showed greater accuracy when compared to other algorithms like Support vector machines and Gradient boosted trees. It also states that long short term memory neural networks can well work with typical granularities used in current grid system.

- Iiys Sutskever et al in Sequence to sequence learning with neural networks presents a end to end method to sequence learning that makes minimal assumptions on the structure. Here two long short term memory neural network algorithms are used. One multi layered algorithm is used to map the input sequence of fixed dimensionality and another one is to decode target from vector. By this approach translation from English language to French is performed.
- Hochreiter and Schmidhuber (1997) introduced a special kind of recurrent neural network termed the Long Short Term Neural Network (LSTM). This paper also demonstrated that LSTM's were capable of solving artificial long time lag tasks that have never been solved by previous recurrent network algorithms.
- Klaus Greff et al. (2017) have stated that LSTM's are effective at retaining long-term dependencies and have stated its use for a wide range of complex problems like speech recognition, hand-writing recognition etc.
- Kingma et al. (2014) introduced an adaptive learning rate optimization algorithm termed ADAM for the optimization of objective functions that are stochastic and non-convex in nature. The paper also stated the use of adaptive learning rates by ADAM to optimize the values of the parameters involved in the objective function.
- Y.Bengio et al. (2012) empirically and theoretically demonstrated that random search hyper parameter optimization is more efficient than grid search hyper parameter optimization. The paper also discussed the need to reduce the expected loss over samples with the help of a function. It was also shown that the function that reduces the expected loss involved the use of a hyper parameter which needs to be optimized to yield the best results for a machine learning problem.
- T.Chen et al. (2016) described a scalable end-to-end tree boosting system termed as XGBoost, Also, a weighted quantile algorithm was proposed and a sparsity-aware algorithm was introduced for parallel tree learning.

- G.Ke et al.(2017) proposed a novel Gradient Boosted Decision Tree algorithm termed LightGBM. This also experimentally demonstrated that LightGBM outperforms XGBoost and Stochastic Gradient Boosting (SGB) in terms of computational speed and memory consumption.

2.6 LITERATURE REVIEW ON POWER PREDICTION USING STACKED MODEL

- Gavin C Cawley in On overfitting in model Selection and subsequent selection bias in performance evaluation (2010) demonstrates that a low variance is less important as a non negligible variance causes overfitting while model training.
- Bertrand Clarke in Comparing bayes model averaging and stacking when model approximation error cant be ignored (2003) shows that bayes model averaging is more sensitive to approximation errors. Bayes model averaging was a popular method before the inception of stacking. Overall results in the study shows that stacking is best method and it outperforms bayes model averaging in many cases and robustness.
- David H Wolpert in Stacked Generalization (1992) explains the whole stacking process and the way it should be performed. Various cases and experimental studies are explained in the literature. In this literature cross validation is inferred as multiple generalizers fed back to single one. Many a type like linear combination of generalizers are also explained. It is also said that of all generalizers stacked generalizers reduces the error rates.

Chapter 3

Exploratory Data Analysis

Exploratory Data Analysis (EDA) is a set of initial investigations performed on the data. It aims to figure out an attitude or philosophy to how data analysis is to be carried out on a dataset. It basically is an iterative process in which questions are asked about the data in hand, answers are found for these questions using various techniques and then refining these answers.

3.1 DATA PREVIEW

Data preview is the first step in Exploratory Data Analysis. A preview of the data gives an idea on the number of entries, columns and data types. Fundamental changes needed are identified and changes are made, such as modifying data types.

3.2 BOX PLOTS

Box plots are useful for visualising range and distribution of data. They split the values into quartiles. The “box” extends from the first quartile (Q1) to the third quartile (Q3) and a horizontal line is used inside the box to show the median of the dataset (Q2). So essentially, they split data into four groups, which are divided by lines which mark the quartiles. Vertical lines called whiskers extend from the top and bottom of the boxes and represent the values outside the middle 50% Outliers in the dataset, if any, are plotted separately in the plot, above or below the whiskers. The maximum and minimum values are also shown by lines. The maximum will be given by $Q3 + 1.5 * IQR$ and the minimum by $Q1 - 1.5 * IQR$ where IQR is the

interquartile range, the difference between upper and lower quartiles.

Box plots help to understand certain characteristics of the data such as range of values, outliers, interquartile range as well as information about how symmetric or skewed the dataset is. If the box has equal proportions around the median, the distribution is symmetric or normal. A positively skewed distribution, the median will be closer to the lower quartile while a negatively skewed distribution will show the median closer to the upper quartile.

3.3 SCATTER PLOTS

Scatter plots display the relationship between any two numerical variables using cartesian coordinates. Each point in the plot represents the values of two variables. The independent variable is usually plotted along the horizontal axis and the dependent variable along the vertical.

Scatter plots are useful in identifying the relationships between variables. It can suggest various correlations between them such as positive and negative, linear and nonlinear as well as the levels of association between the variables are understood from scatter plots. Clusters can also be identified from them, which is often necessary for classification purposes. A trendline can be added to scatter plots and further interpretation of data can be performed.

3.4 DUPLICATE ENTRIES

Data is collected and stored in different ways in systems. It is possible for duplicate entries to appear in the dataset. Dealing with duplicate entries depends on the type of job in hand.

3.5 NULL VALUES

While working with real world datasets, it is possible to have entries with missing values, due to a variety of reasons such as corrupted data or failure in extraction of data. It is important

to identify and make necessary changes in such cases.

Duplicate entries and null values must be dealt with before entering further stages of EDA.

3.6 OUTLIER DETECTION FOR MULTI DIMENSIONAL DATA

Outlier detection in multi dimensional data is much more non-obvious than in the case of 2 dimensional data because the idea of “proximity” gets obscure. Therefore, different algorithms are used to do the same in multi dimensional data, which is usually encountered in real world problems.

As the dimensionality of data increases, approaches like considering the distances between the points for drawing outlier patterns or detecting the presence of outliers is inappropriate.

3.6.1 Angle Based Outlier Detection(ABOD)

ABOD is a commonly used approach for detecting the presence of outliers in high dimensional data. In this technique, the variance of angles is considered as the measure of outlierness for a point in the dataset. This approach is a more accurate approach as it considers the variance of angles of a point measured between a pair of points by considering the angle between the difference vectors with respect to that point as opposed to merely computing the distance between the points for a point in the dataset.

This variance of angles approach is more accurate than a distance based approach since angles are more stable than distances and the ABOD does not substantially deteriorate in high dimensional data. On considering a point in the cluster, it will have the highest variance of angle measure as it is surrounded by points in all possible directions. Whereas, this measure is lesser for a point considered on the border of a cluster but is greater than for a point outside the cluster. This is so for a point outside the cluster because points outside the cluster are

only positioned in particular directions accounting for lesser variance of angles. Thus, lower the variance of angles for a point, more likely it is to be considered as an outlier in a dataset.

This similar behavior of variance of angles for points of the above-mentioned positions can be noticed while considering the spectra of peaks for the variance of angles measured with respect to a point in the dataset. That is, points inside the cluster record the highest peak value followed by points on the border of a cluster followed by points outside the cluster.

The ABOD algorithm computes the variance of angles for each point of the dataset and returns the top m points having the smallest variance of angles measured as outliers.

3.6.2 Feature Bagging

Feature Bagging is an outlier detection technique that involves the use of a meta-estimator that fits a number of base estimators or detectors on various sub-samples of the dataset.

The size of the sub sample is generally the same as that of the input sample size. The process of random sampling of features is carried out ranging from half of the features to all of the features involved in the dataset. Base estimators or detectors such as ABOD or K-Nearest Neighbors are used for the sub samples.

The Feature Bagging algorithm first randomly constructs n -sub samples by randomly selecting a subset of features thereby resulting in a diversity of the base estimators or detectors. Finally, a meta-estimator is created by averaging or taking the maximum of all base estimators or detectors involved to improve the prediction accuracy and to restrict the extent of over-fitting.

3.6.3 Histogram-Based Outlier Detection

The Histogram-Based Outlier Detection technique employs the use of a Histogram-Based Outlier Score (HBOS) to detect outliers in a dataset.

In this technique, for each single feature or dimension, a univariate histogram is first constructed. Where the height of the histogram represents the relative number of data points falling into a particular bin that gives the density estimate for the respective histogram. After individual histograms are generated for each feature (the height of each histogram representing a density estimation), the histograms are normalized so that the maximum height of each histogram is 1.0. This gives an equal weight of each feature to the final outlier score.

The HBOS of every instance ‘p’ or observation is computed using the corresponding height of the bins where the instance is located. The HBOS is a multiplication of the inverse of the estimated densities assuming feature independence. The drawback of assuming feature independence becomes less severe when the dataset has high Number of dimensions due to larger sparsity.

$$\sum_{i=0}^d \log \frac{1}{hist(p)} \quad (3.1)$$

Where d represents the no of dimensions or features, p represents the instance or observation.

3.6.4 Isolation Forest

Isolation Forest is an example of an unsupervised anomaly detection algorithm that detects or identifies outliers instead of highlighting or profiling the normal values or points-the technique followed by conventional outlier detection techniques.

This technique involves the use of splits or partitions to isolate or separate the respective point from the rest of the points. The partitions are generated by randomly selecting an attribute and a split value is then randomly selected between the maximum and minimum values allowed for an attribute. The intuition behind this technique is that further the point is away from points that follow a particular behavior or from points having similar properties, fewer is the number of partitions or splits required to isolate that point from the rest of the points.

This process of partitioning is done by a recursive approach starting from the root node to the terminating node. As this process works in such a manner, it is best represented by a tree structure. Thereby, the partitions required to isolate a point from the rest of the points is equivalent to the path length from the starting node (root node) to the terminating node. As each partition is generated randomly, individual trees are generated with different sets of partitions. The path lengths are then averaged over the path lengths produced by different trees to give the expected path length. Therefore, when a forest of random trees collectively produce shorter paths for some particular points, the points are more likely to be considered as anomalies or outliers.

3.6.5 K-Nearest Neighbors (KNN)

KNN is a neighborhood based anomaly or outlier detection technique. Here, points or outliers are distinguished from the points that follow a particular behavior by measuring the average distance between the respective point and its nearest k neighbors (k is to be specified as per the requirements of the user). The above mentioned approach is the global based approach where an anomaly score is assigned to each instance with respect to the entire dataset. The instance having the highest anomaly score will be considered as the outlier.

For a local based approach, the anomaly score of a point is represented by its outlierness with respect to its nearest neighborhood. The local approach compares the local density of a point to that of its neighbors. The local density of a data point is inversely proportional to the average distance to its k nearest neighbors. In this approach, the Local Outlier Factor (LOF) is considered as the anomaly score for a point. The LOF is defined as the ratio of the local density of the point to the average local density of its neighbors.

Normal points have an LOF score equal to 1.0 whereas for outliers the LOF is greater than 1.0.

The value of k should be given accordingly since higher values of k tend to give a wrong classification and smaller values of k do not give the complete picture. The process of cross-

validation best selects the value of k.

3.7 RESULTS AND DISCUSSION

3.7.1 Preview of Dataset

Before doing an exploratory data analysis on the dataset, it is important to have an idea about the dataset regarding the number of entries (observations), data types, null values present in the dataset as well as the features (columns) in the dataset.

The preview of the dataset is shown below.

	Air temperature	Pressure	Wind speed	Wind direction	Power generated by system
0	10.926	0.979103	9.014	229	33688.1
1	9.919	0.979566	9.428	232	37261.9
2	8.567	0.979937	8.700	236	30502.9


```
Range Index: 52560 entries, 0 to 52559
Data columns (total 5 columns):
 #  Column          Non-Null Count  Dtype  
--- 
 0  Air temperature    52560 non-null float64
 1  Pressure           52560 non-null float64
 2  Wind speed         52560 non-null float64
 3  Wind direction     52560 non-null int64  
 4  Power generated by system 52560 non-null float64
 dtypes: float64(4), int64(1)
```

Figure 3.1: Datset Preview

3.7.2 Boxplots for Dataset

Depicted below are the boxplots for the features involved in the dataset namely air temperature, pressure, wind speed, wind direction and power generated by the system.

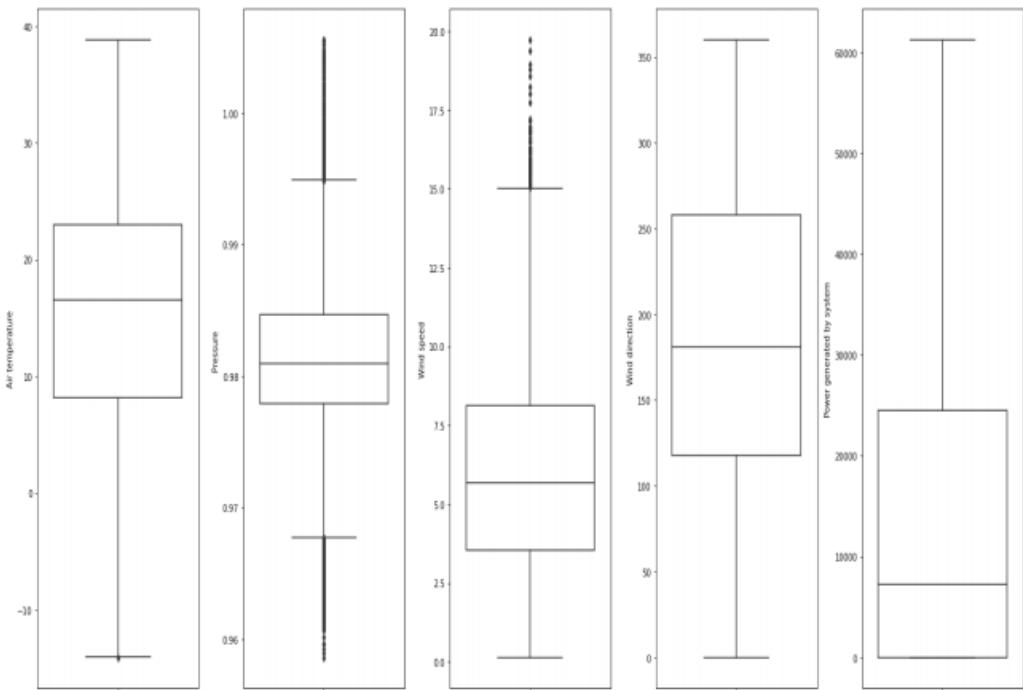


Figure 3.2: Boxplots for the dataset

The detailed statistics concerning the respective boxplots are shown below.

	count	mean	std	min	25%	50%	75%	max
Air temperature	52560.0	15.531589	9.867689	-14.06600	8.180500	16.548000	23.001000	38.80500
Pressure	52560.0	0.981262	0.005393	0.958598	0.977896	0.980961	0.984702	1.00558
Wind speed	52560.0	5.944999	3.009099	0.140000	3.533000	5.678000	8.134000	19.72500
Wind direction	52560.0	181.286187	98.435074	0.000000	118.000000	181.000000	258.000000	360.00000
power generated by system	52560.0	14743.922284	17254.684484	0.000000	0.000000	7286.300000	24537.000000	61245.400000

Figure 3.3: Feature Statistics from Boxplots

From the above boxplots, following are the analysis for each of the boxplots.

The boxplot depicting Air temperature shows a negatively skewed distribution as the length of the lower whisker is more than that of the upper whisker. However, no outliers were observed.

The boxplot depicting Pressure follows a normal distribution as the whiskers on both ends are equal in length and outliers were observable on both sides outside the lower and upper whisker.

The Wind speed boxplot shows a positively skewed distribution as the length of the upper whisker is greater than that of the lower whisker and outliers were observable outside the upper whisker.

The Wind direction boxplot depicts a positively skewed distribution as the length of the upper whisker is slightly greater than that of the lower whisker and no outliers were observable for the boxplot.

For the boxplot depicting the power generated by system, the distribution follows a highly positively skewed distribution, as the length of upper whisker is far greater than that of the lower whisker. In addition, no outliers were observed.

3.7.3 Presence of Duplicate Entries

It was found that there were 8760 entries out of the total 52560 entries that had the same values as that of the observations in the dataset. Shown below is the view depicting some of the duplicate entries.

	Air temperature	Pressure	Wind speed	Wind direction	Power generated by system
3606 0	-7.306	0.995581	5.169	343	5357.22
4482 0	-7.306	0.995581	5.169	343	5357.22
3605 9	-7.095	0.995517	6.638	352	13966.70
4481 9	-7.095	0.995517	6.638	352	13966.70
3510 0	-6.890	0.995736	5.818	338	8639.31

Figure 3.4: Duplicate Entries

3.7.4 Presence of Null Values

No null values were observed in the dataset. The representation is shown below.

```
# Finding the null values.
print(df.isnull().sum())
```

Air temperature	0
Pressure	0
Wind speed	0
Wind direction	0
Power generated by system	0
dtype: int64	

Figure 3.5: Null Values

3.7.5 Scatter Plots

Scatter plots are used to depict the relation between the variables involved in a plot and are used to depict outliers in a dataset. The following scatter plots were considered.

1. Air Temperature v/s Hour

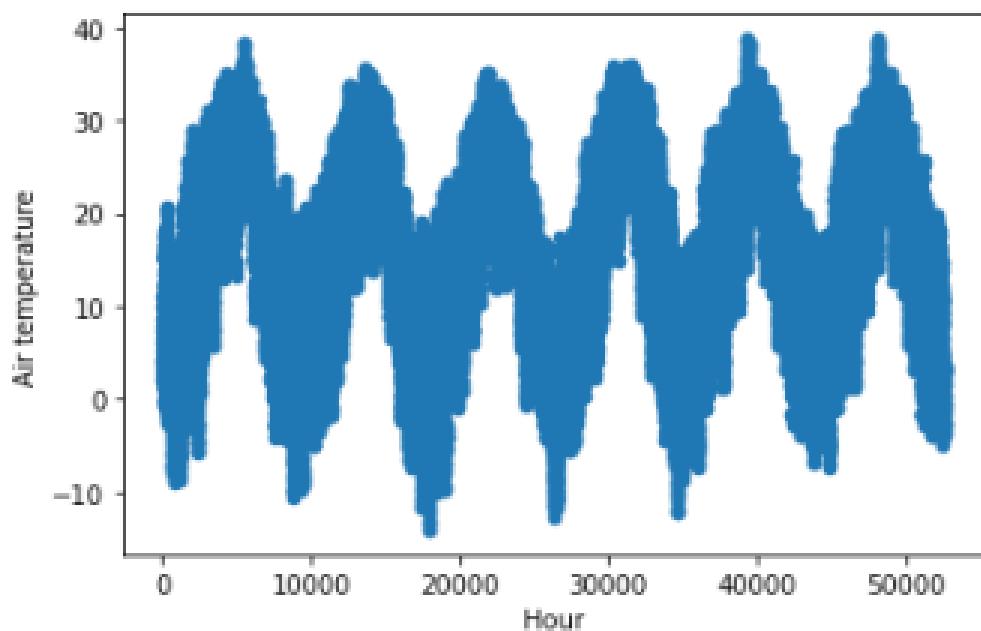


Figure 3.6: Air Temperature vs Hour

The Air temperature v/s Hour plot shows a similar pattern and no observable outliers were detected from the scatterplot.

2. Pressure v/s Hour

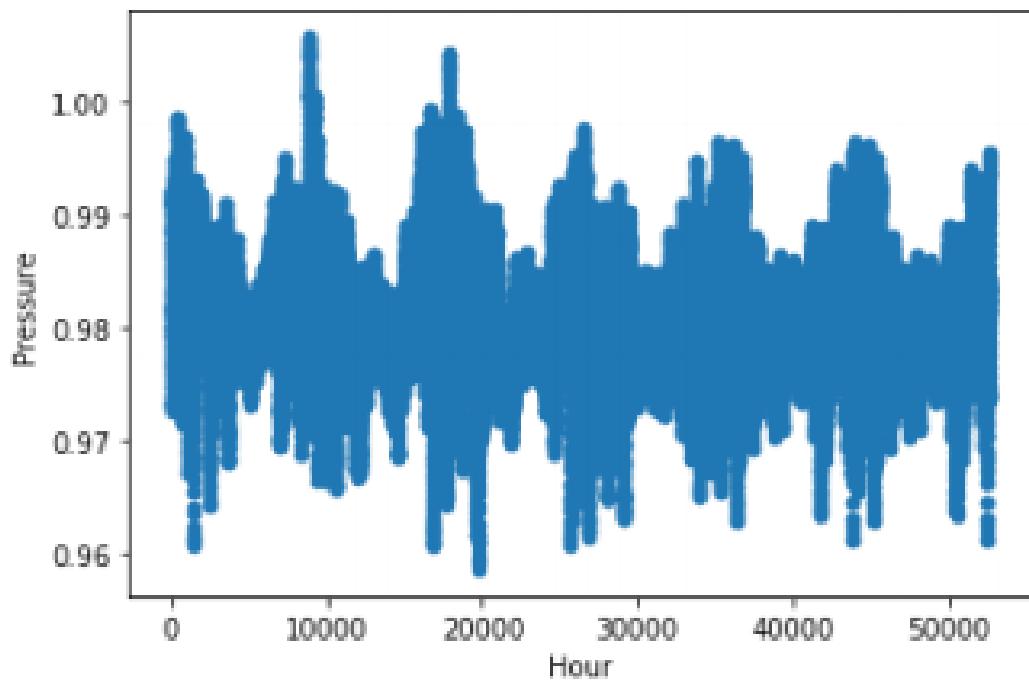


Figure 3.7: Pressure v/s Hour

Unlike the previous scatter plot apart from the similar pattern, outliers can be observed.

3. Wind speed v/s Hour

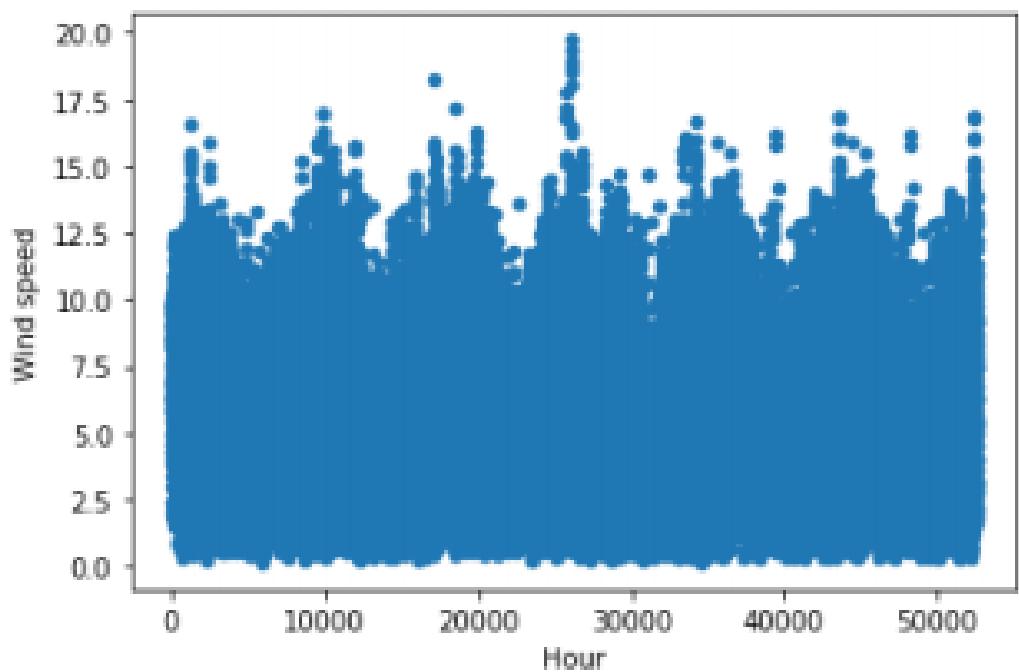


Figure 3.8: Wind speed v/s Hour

In this scatter plot, outliers could be observed.

4. Wind Direction v/s Hour

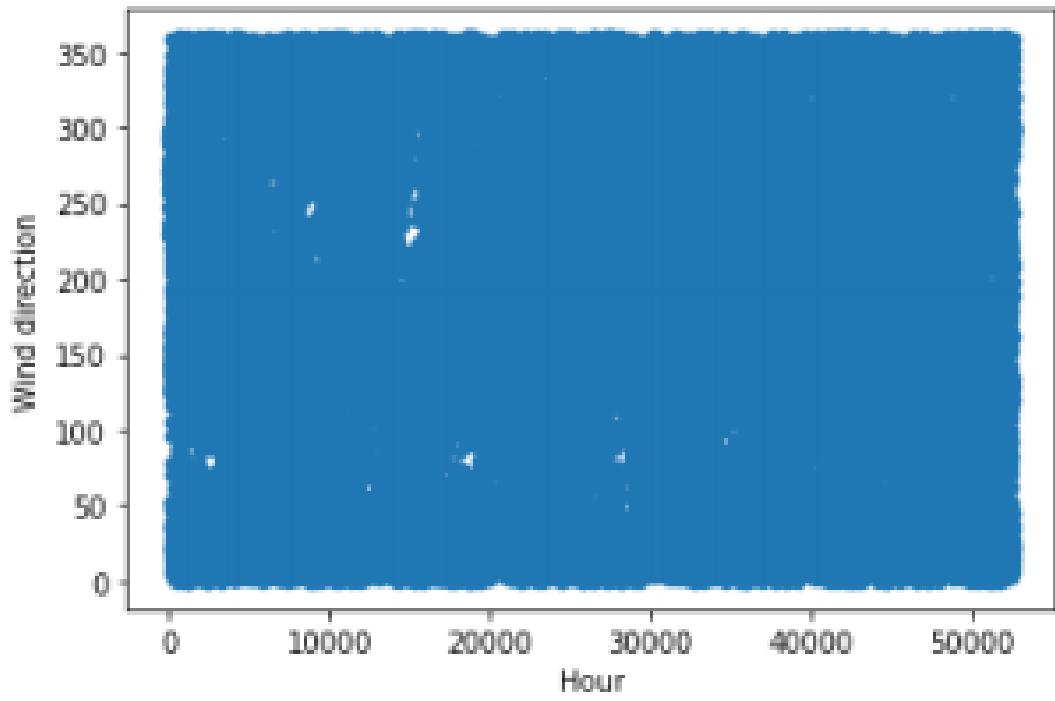


Figure 3.9: Wind Direction v/s Hour

In this scatter plot, outliers could be observed.

5. Power Generated by System v/s Hour

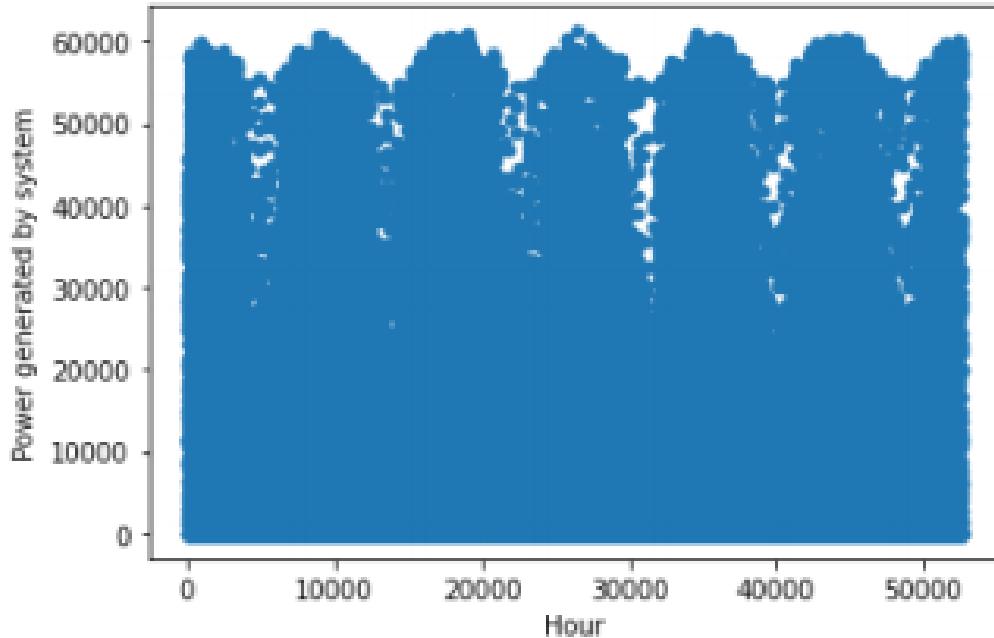


Figure 3.10: Power Generated by System v/s Hour

No outliers could be observed, similar to the previous scatter plot.

3.7.6 Outlier Detection for Multi Dimensional Data

The previously used techniques for detecting outliers in data give only a simplistic picture regarding the nature of outliers as the dataset considered is a multi dimensional dataset. Therefore, there arises the need to identify outliers for multi dimensional data using other outlier detection techniques. For this dataset, the outlier detection techniques were performed using the pyOD API in python. The techniques used were.

1. Angle Based Outlier Detection (ABOD) Considering Air temperature v/s Hour, the results were OUTLIERS : 2734 INLIERS : 49826 Angle-based Outlier Detector (ABOD)

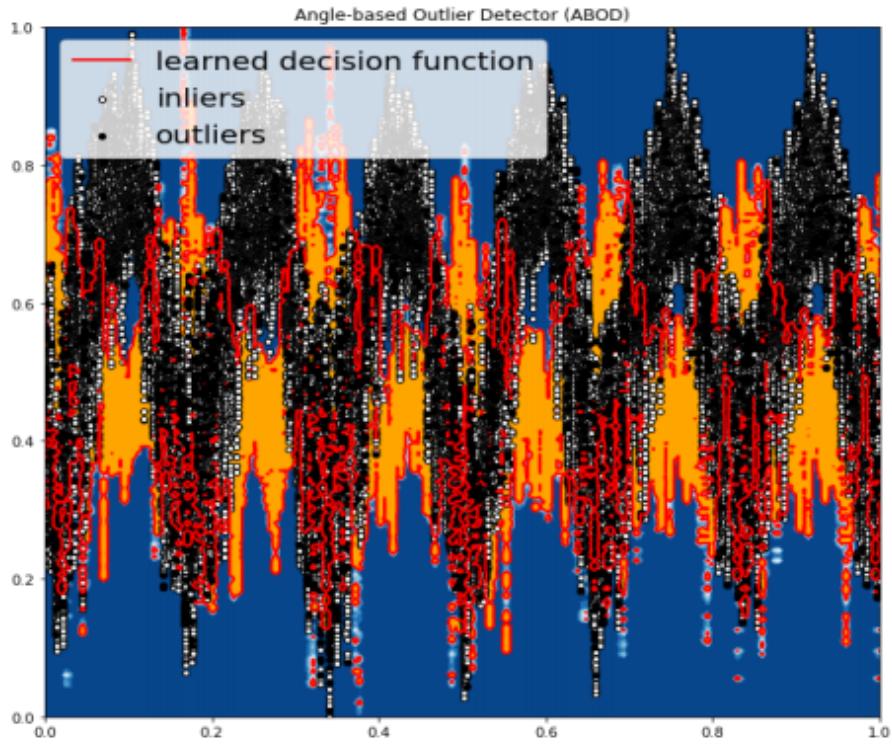


Figure 3.11: ABOD considering Air temperature v/s Hour

Considering Pressure v/s Hour, the results were: OUTLIERS : 2734 INLIERS : 49826
Angle-based Outlier Detector (ABOD)

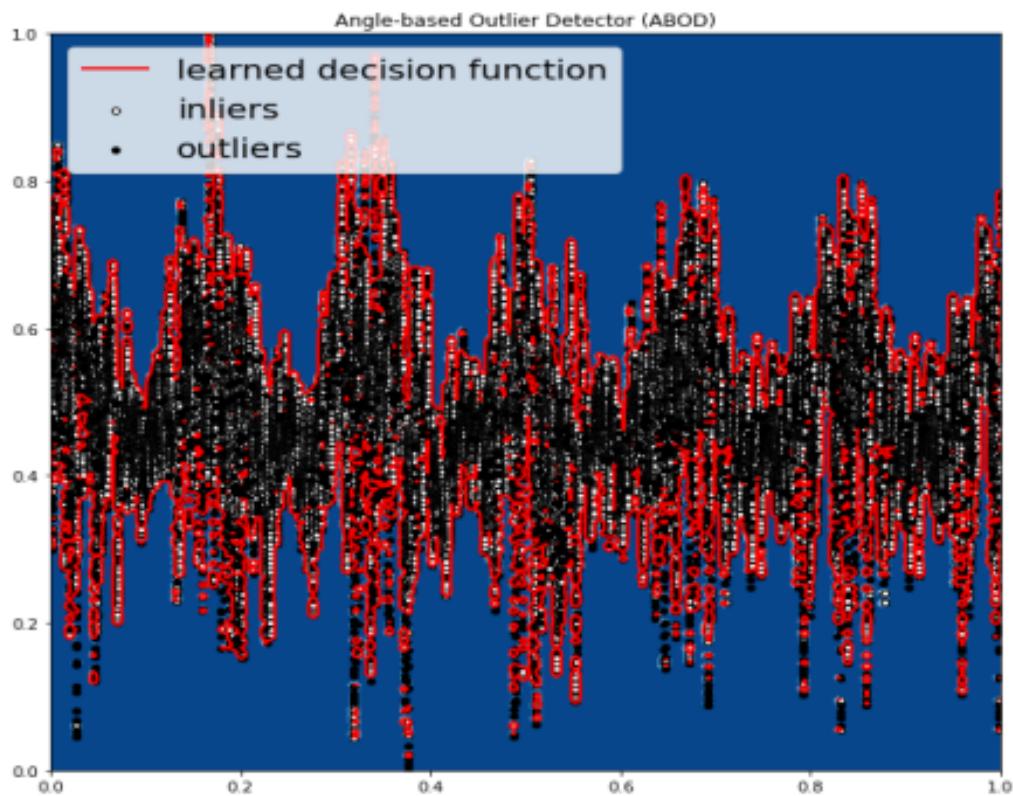


Figure 3.12: ABOD considering Pressure v/s Hour

Considering Wind speed v/s Hour, the results were: OUTLIERS : 2877 INLIERS : 49683
Angle-based Outlier Detector (ABOD)

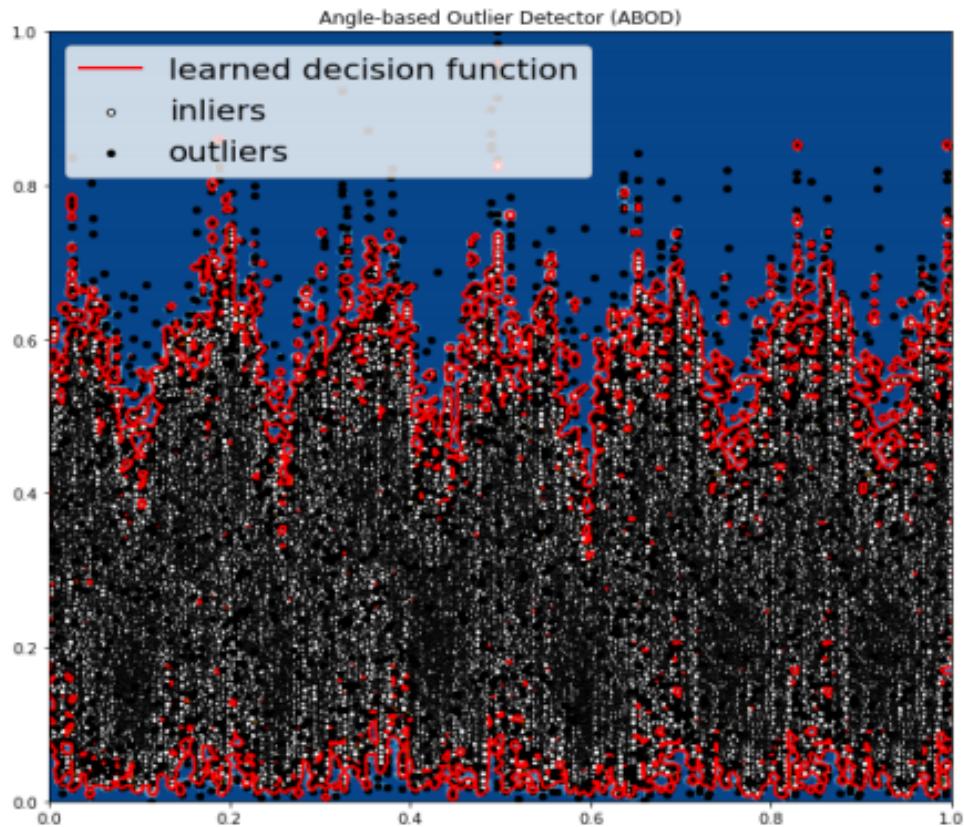


Figure 3.13: ABOD considering Wind speed v/s Hour

Considering Wind direction v/s Hour, the results were: OUTLIERS : 3025 INLIERS : 49535 Angle-based Outlier Detector (ABOD)

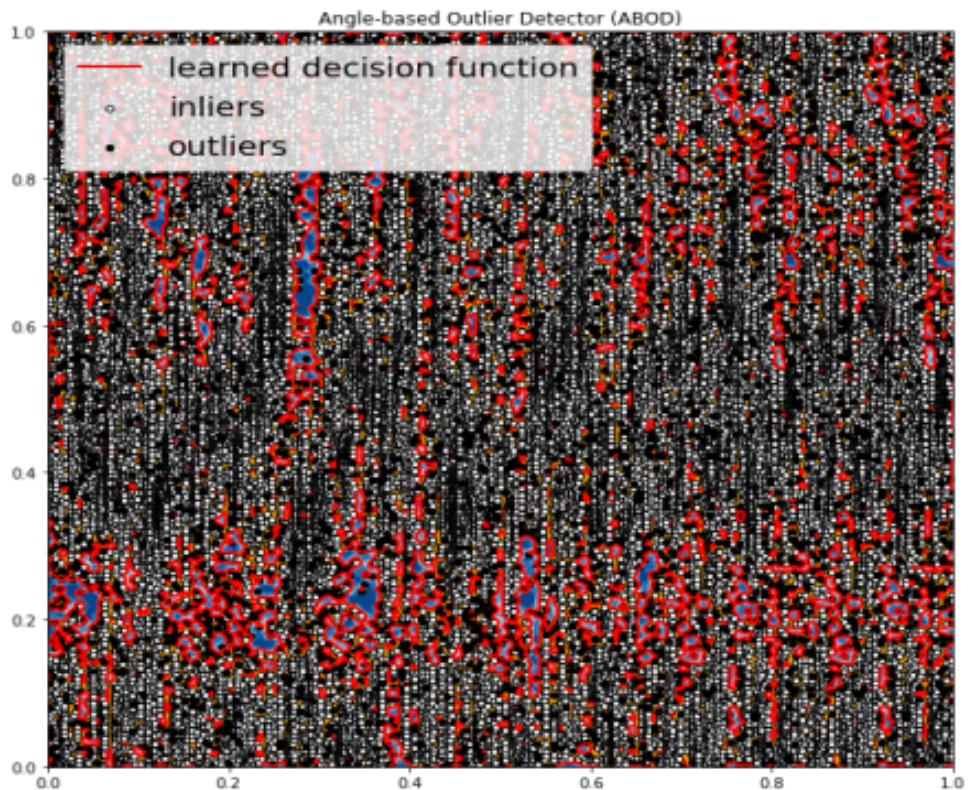


Figure 3.14: ABOD considering Wind direction v/s Hour

Considering Power generated by system v/s Hour, the results were: OUTLIERS : 2699
INLIERS : 49861 Angle-based Outlier Detector (ABOD)

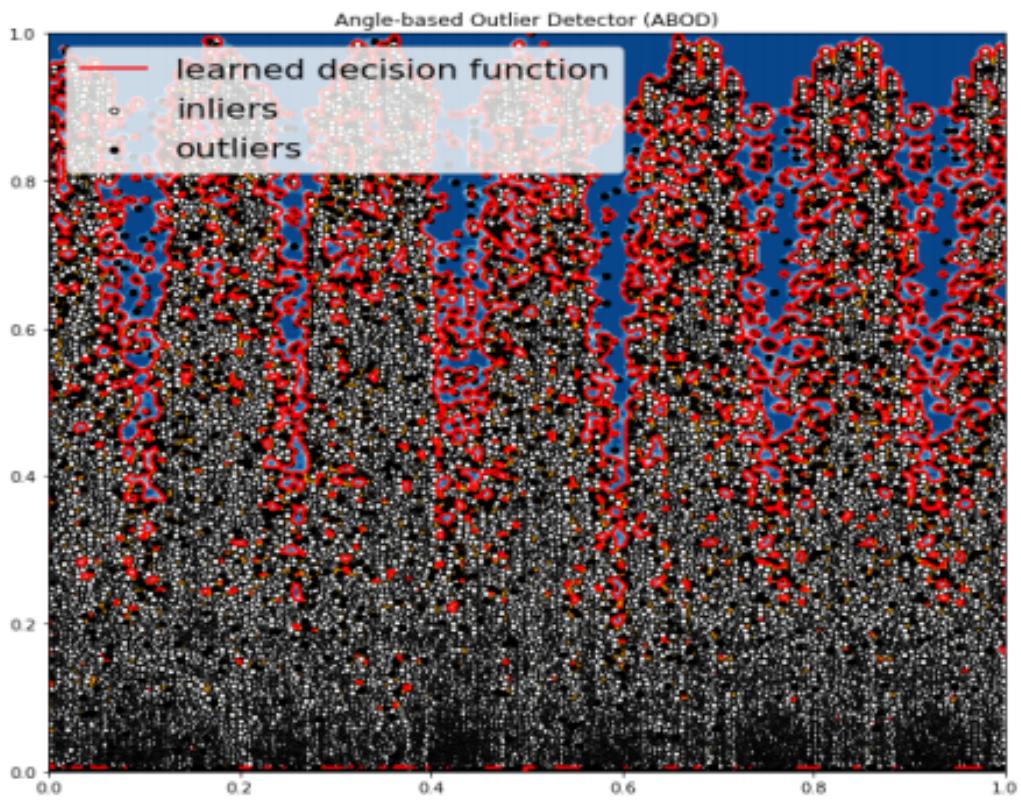


Figure 3.15: ABOD considering Power generated by system v/s Hour

2. Feature Bagging

Considering Air temperature v/s Hour, the results were: OUTLIERS : 2262 INLIERS : 50298 Feature Bagging

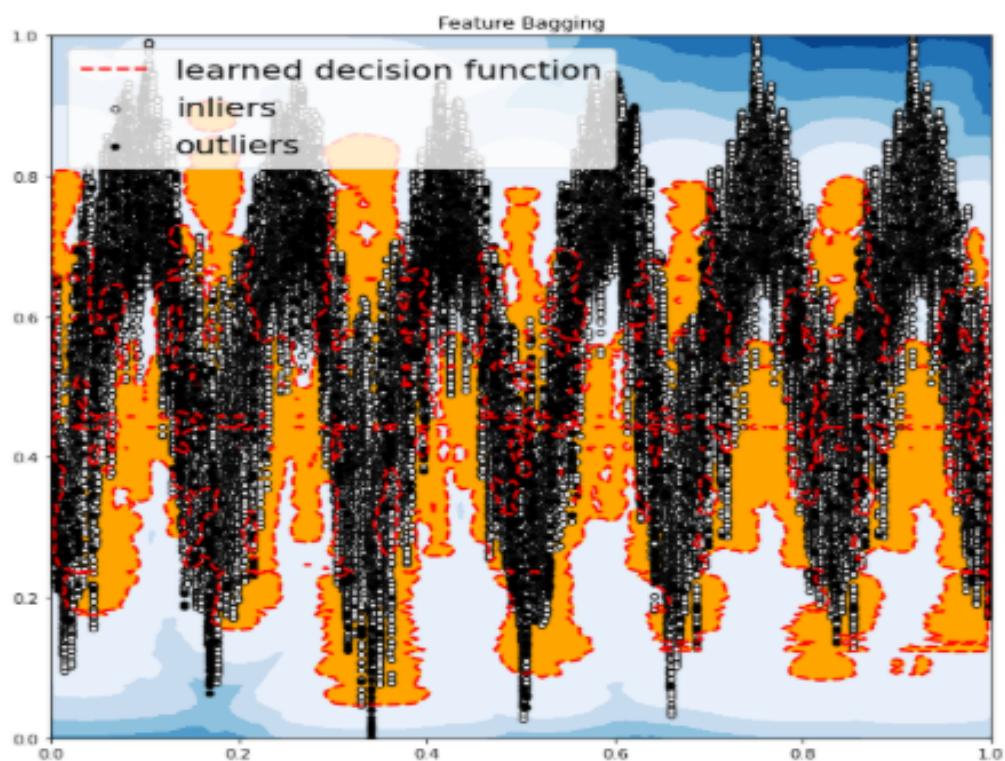


Figure 3.16: Feature Bagging considering Air temperature v/s Hour

Considering Pressure v/s Hour, the results were: OUTLIERS : 2262 INLIERS : 50298
Feature Bagging

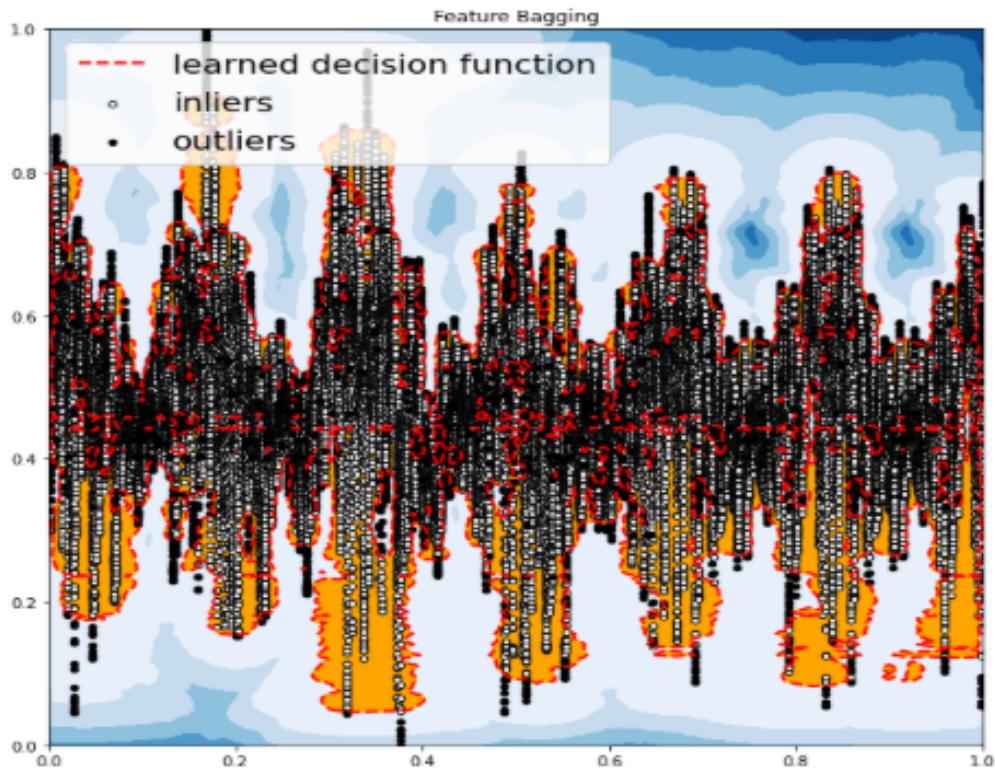


Figure 3.17: Feature Bagging considering Pressure v/s Hour

Considering Wind speed v/s Hour, the results were: OUTLIERS : 2192 INLIERS : 50368
Feature Bagging

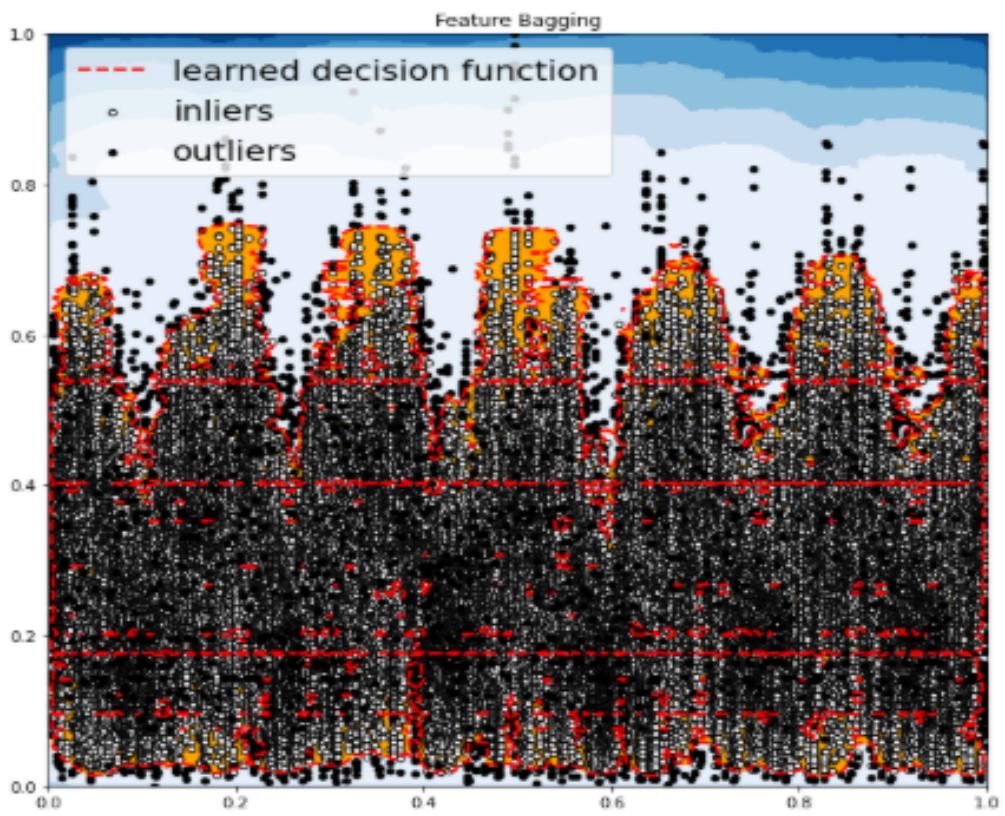


Figure 3.18: Feature Bagging considering Wind speed v/s Hour

Considering Wind direction v/s Hour, the results were: OUTLIERS : 2347 INLIERS : 50213 Feature Bagging

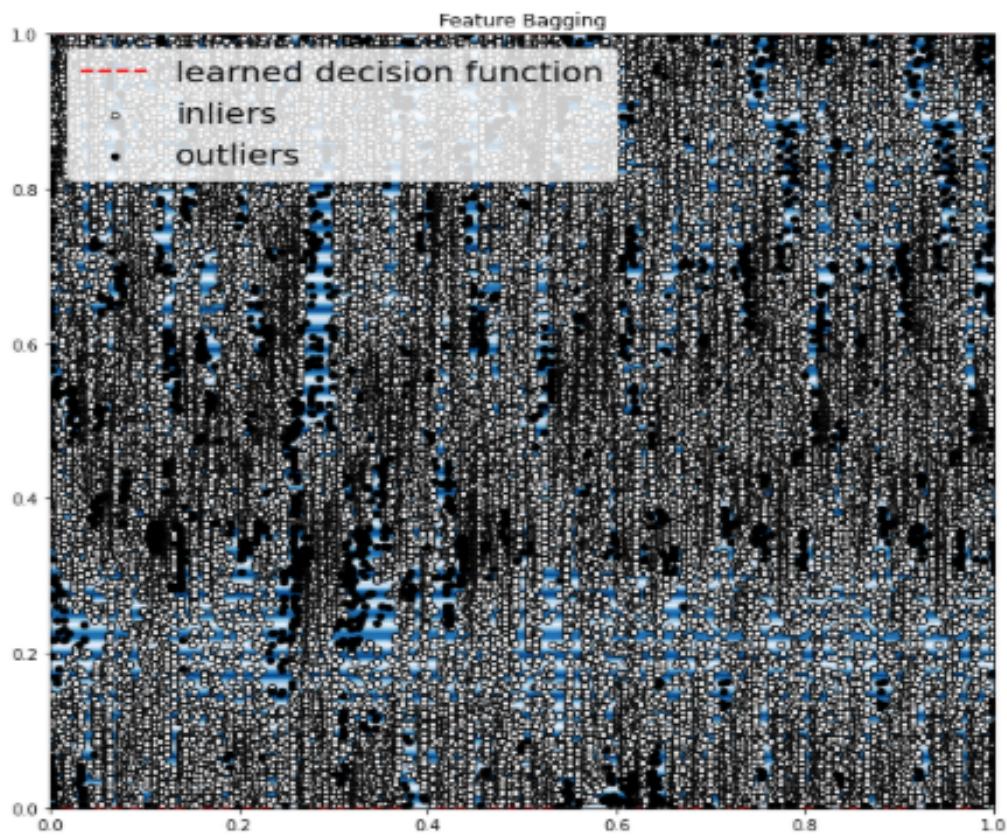


Figure 3.19: Feature Bagging considering Wind direction v/s Hour

Considering Power generated by system v/s Hour, the results were: OUTLIERS : 2391
INLIERS : 50169 Feature Bagging

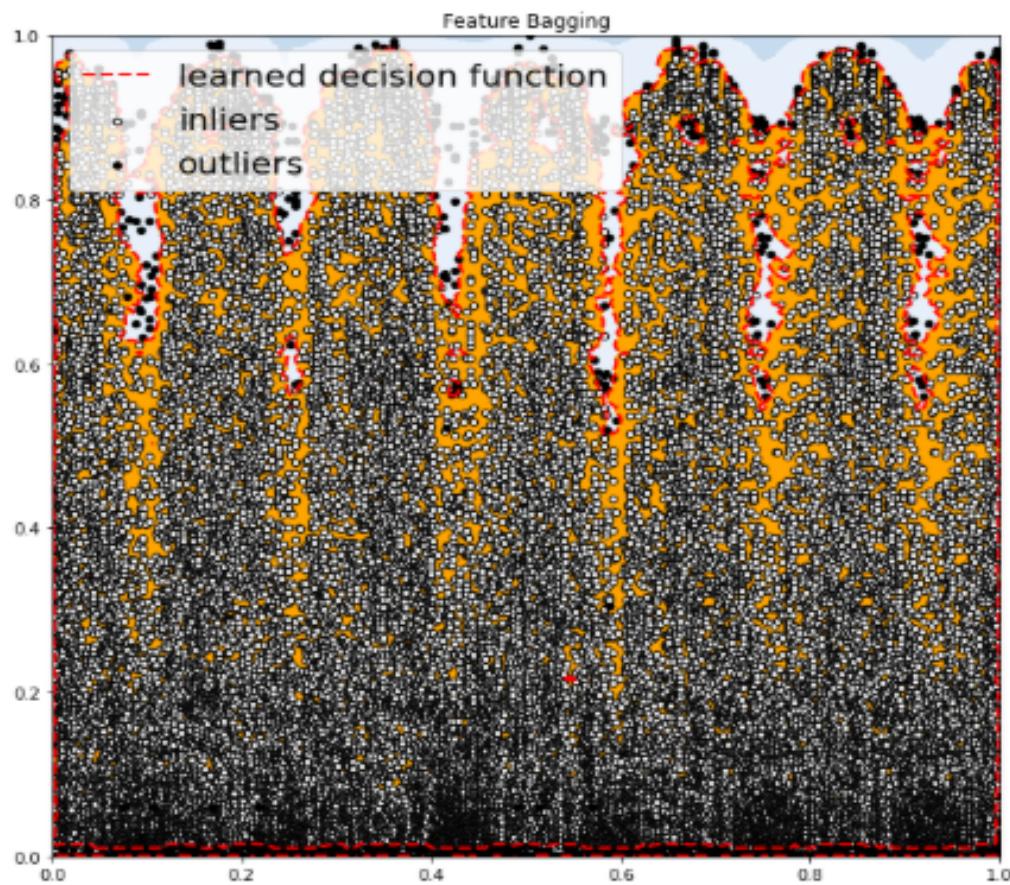


Figure 3.20: Feature Bagging considering Power generated by system v/s Hour

3. Histogram-base Outlier Detection(HBOS) Considering Air temperature v/s Hour, the results were: OUTLIERS : 2582 INLIERS : 49978 Histogram-base Outlier Detection (HBOS)

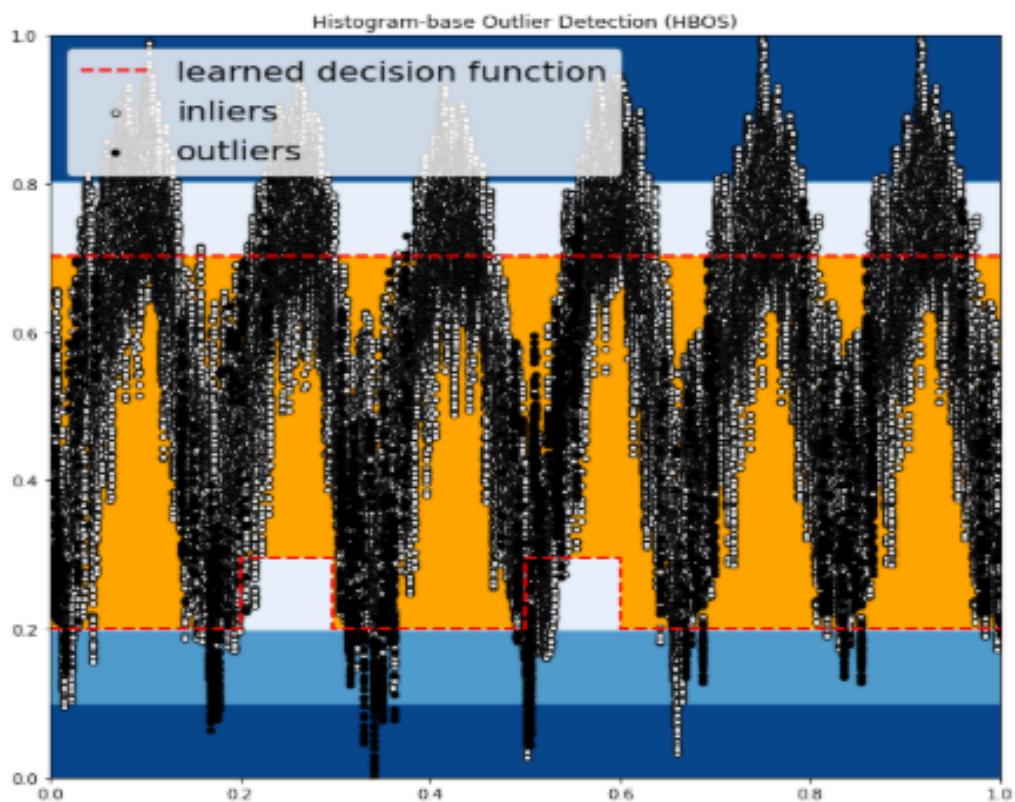


Figure 3.21: HBOS considering Air temperature v/s Hour

Considering Pressure v/s Hour, the results were: OUTLIERS : 2582 INLIERS : 49978

Histogram-base Outlier Detection (HBOS)

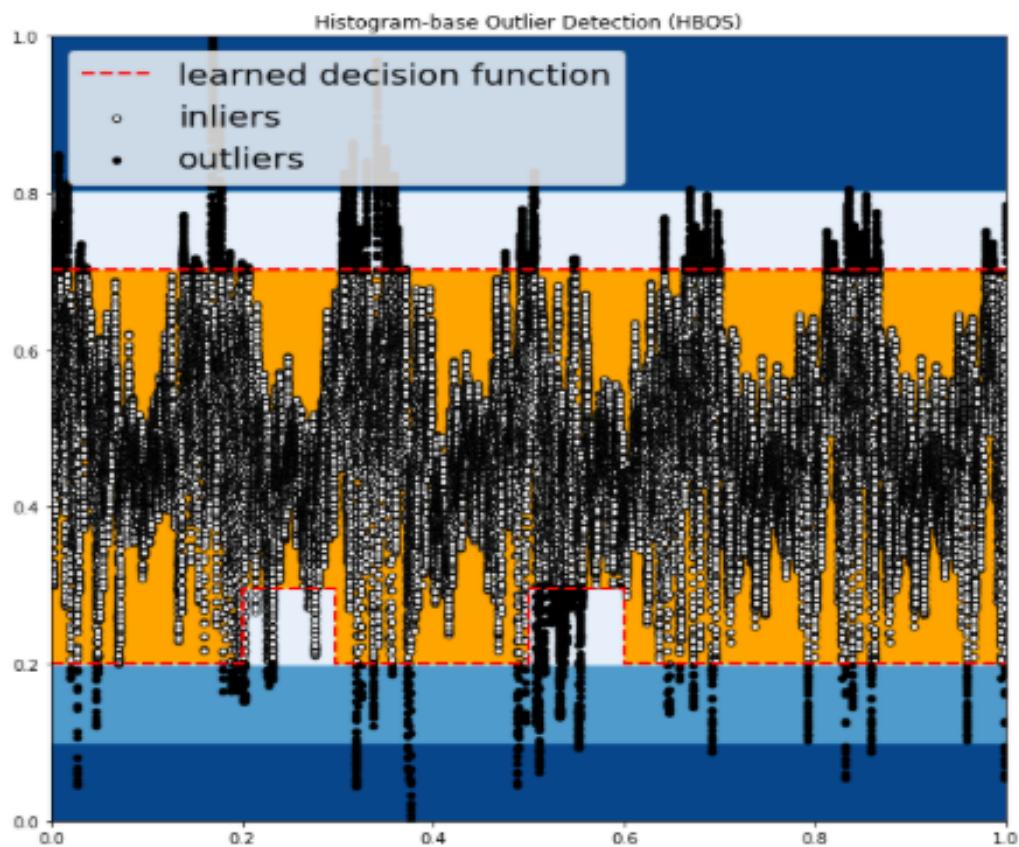


Figure 3.22: HBOS considering Pressure v/s Hour

Considering Wind speed v/s Hour, the results were: OUTLIERS : 2298 INLIERS : 50262

Histogram-base Outlier Detection (HBOS)

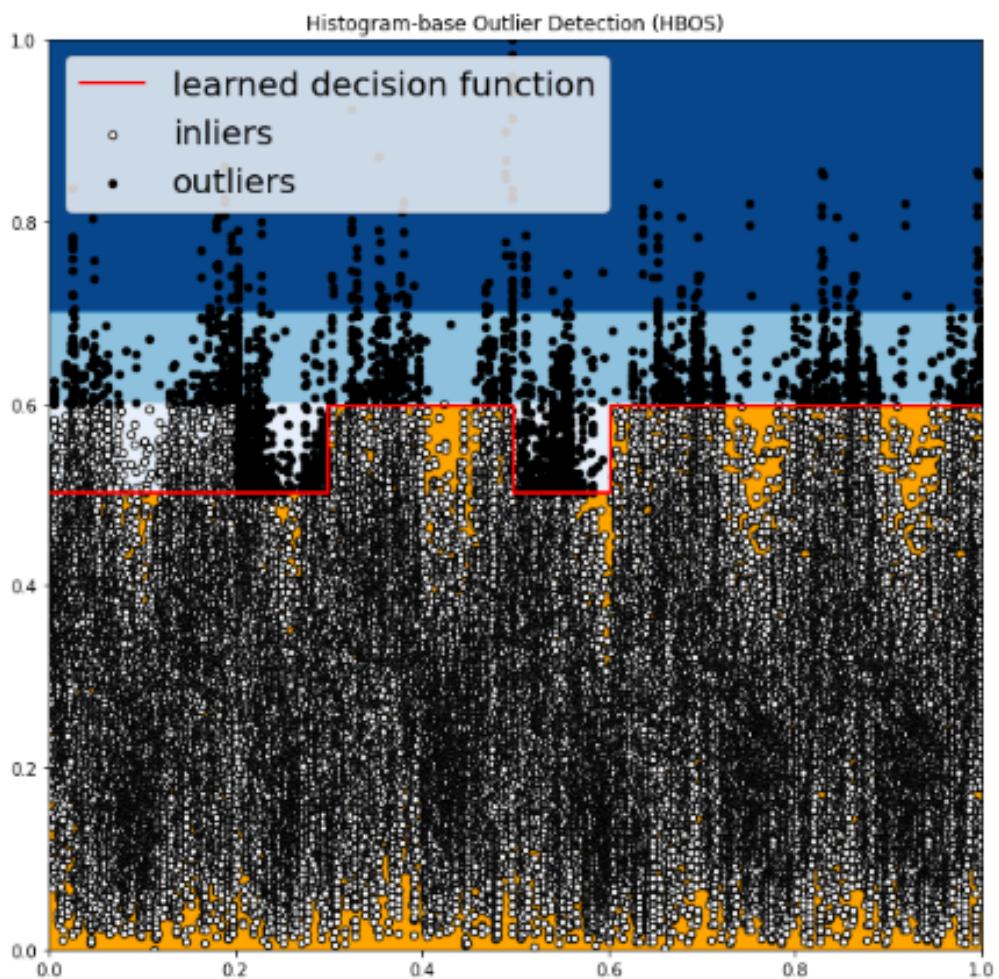


Figure 3.23: HBOS considering Wind speed v/s Hour

Considering Wind direction v/s Hour, the results were: OUTLIERS : 1091 INLIERS : 51469 Histogram-base Outlier Detection (HBOS)

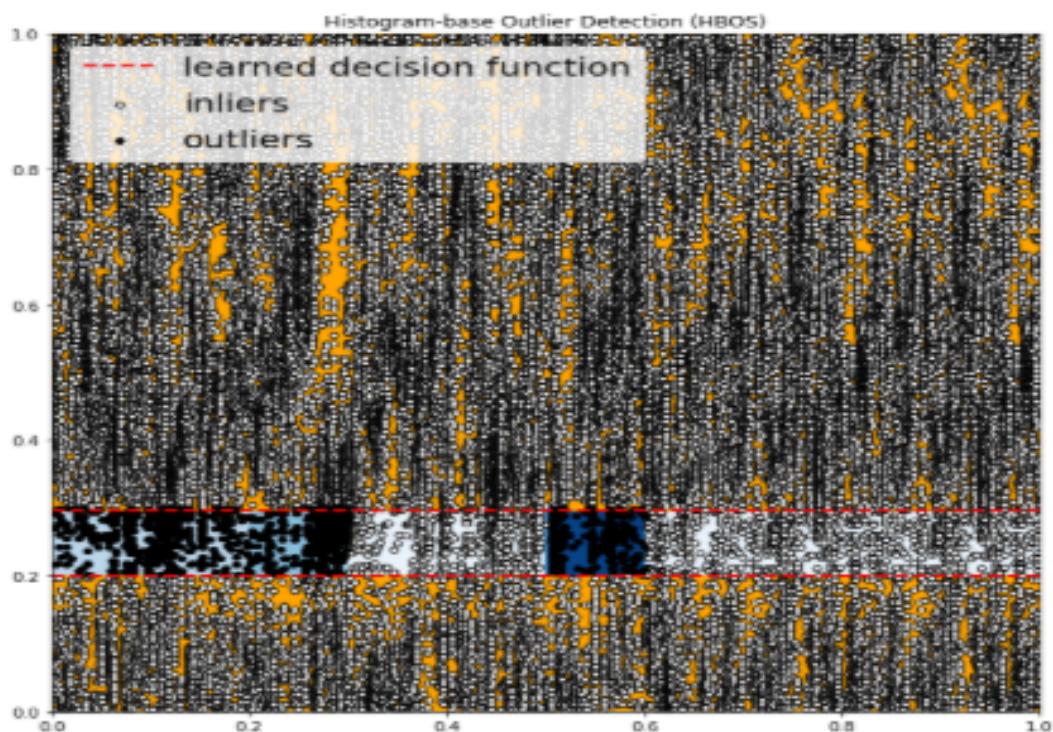


Figure 3.24: HBOS considering Wind direction v/s Hour

Considering Power generated by system v/s Hour, the results were: OUTLIERS : 2475
INLIERS : 50085 Histogram-base Outlier Detection (HBOS)

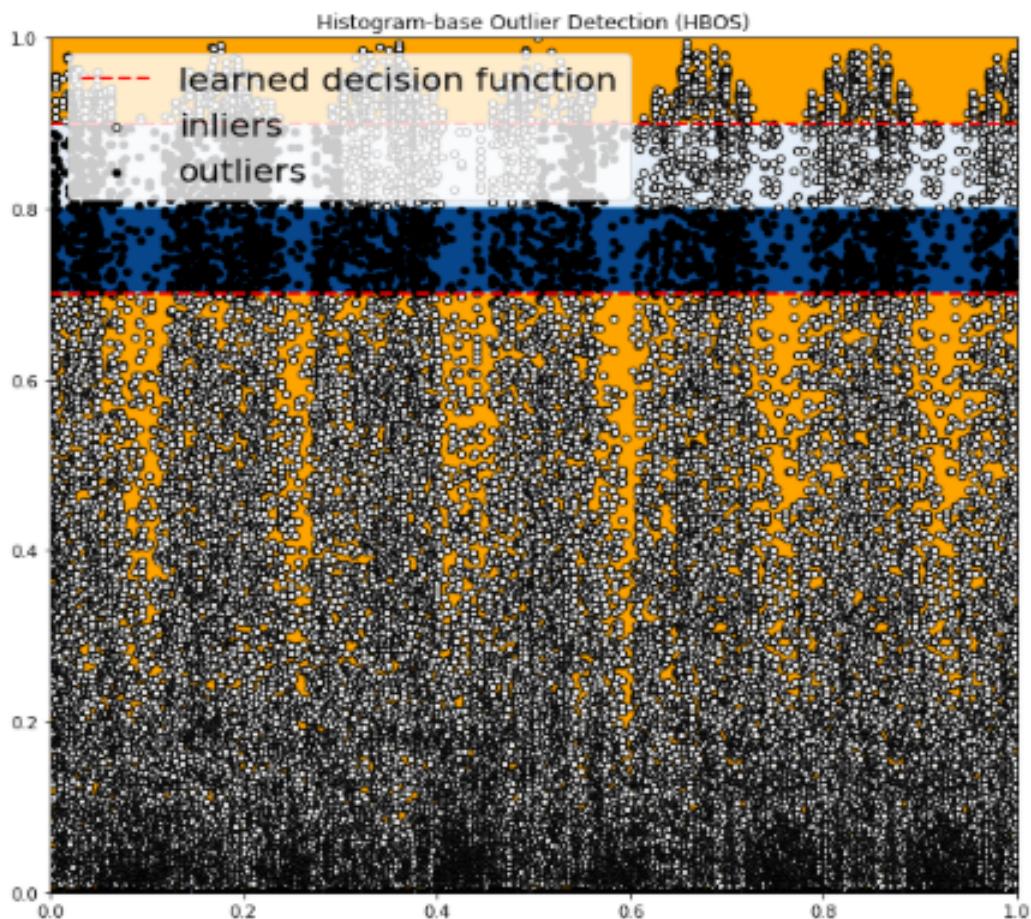


Figure 3.25: HBOS considering Power generated by system v/s Hour v/s Hour

4. Isolation Forest

Considering Air temperature v/s Hour, the results were: OUTLIERS : 2628 INLIERS : 49932 Isolation Forest

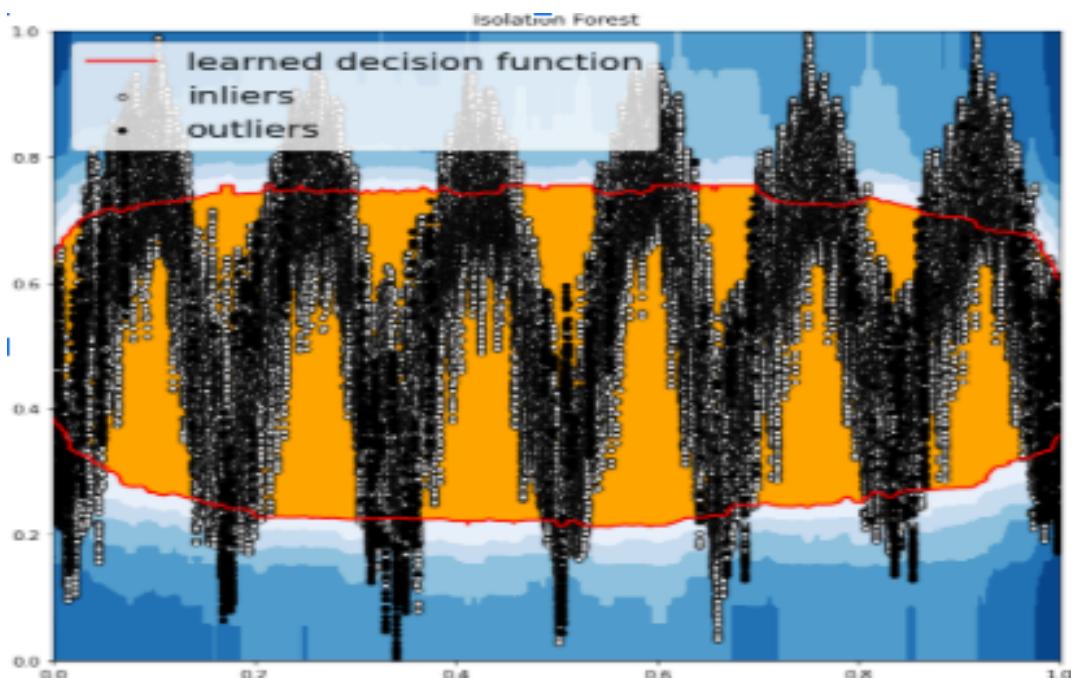


Figure 3.26: Isolation Forest considering Air temperature v/s Hour

Considering Pressure v/s Hour, the results were: OUTLIERS : 2628 INLIERS : 49932
Isolation Forest

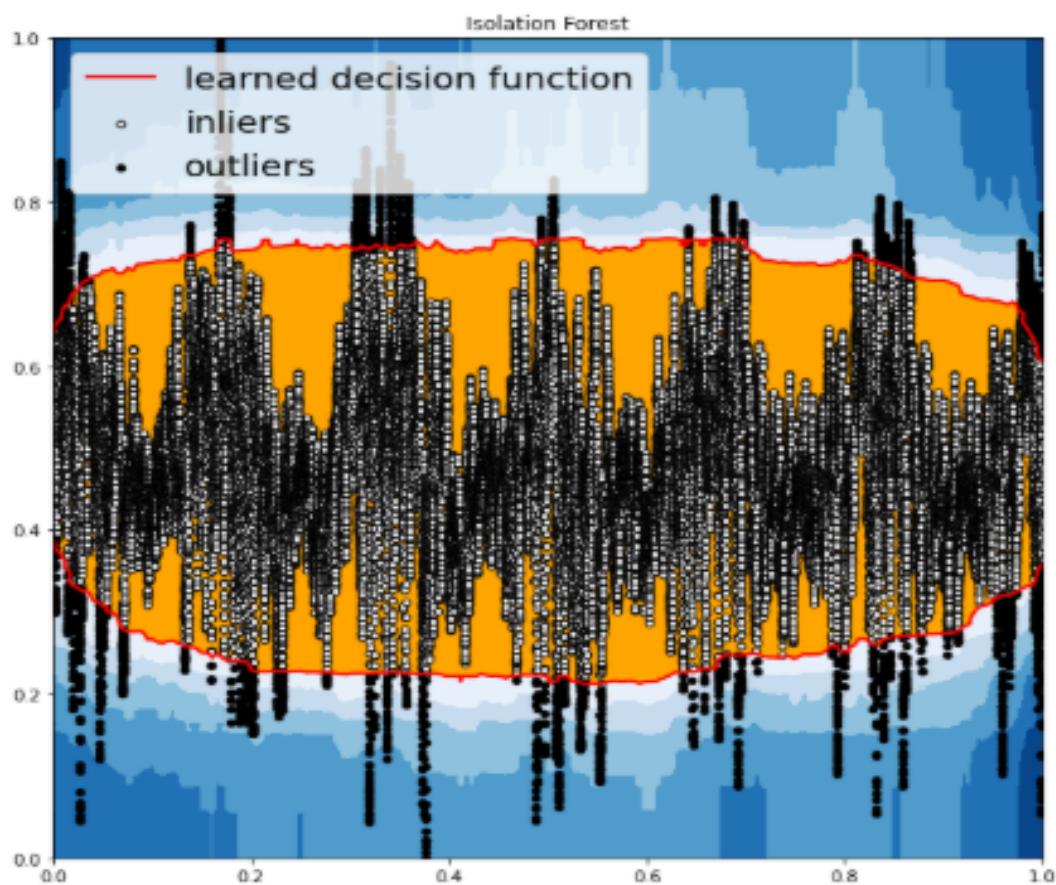


Figure 3.27: Isolation Forest considering Pressure v/s Hour

Considering Wind speed v/s Hour, the results were: OUTLIERS : 2628 INLIERS : 49932
Isolation Forest

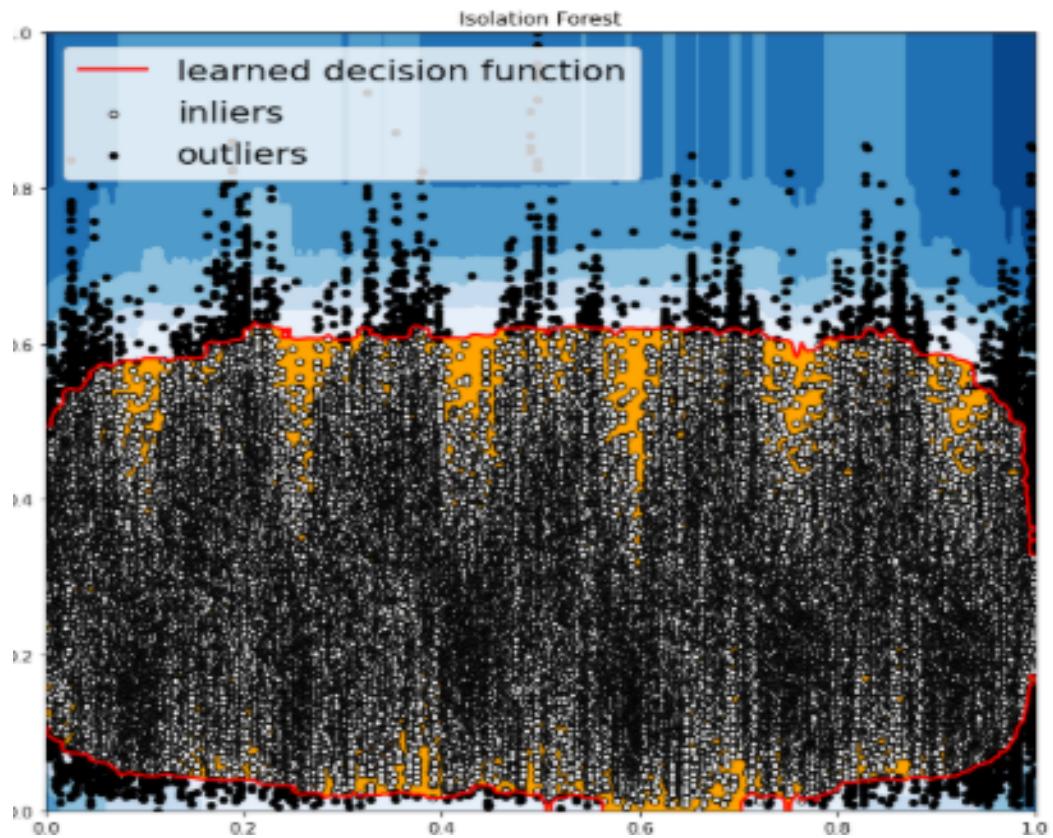


Figure 3.28: Isolation Forest considering Wind speed v/s Hour

Considering Wind direction v/s Hour, the results were: OUTLIERS : 2628 INLIERS : 49932 Isolation Forest

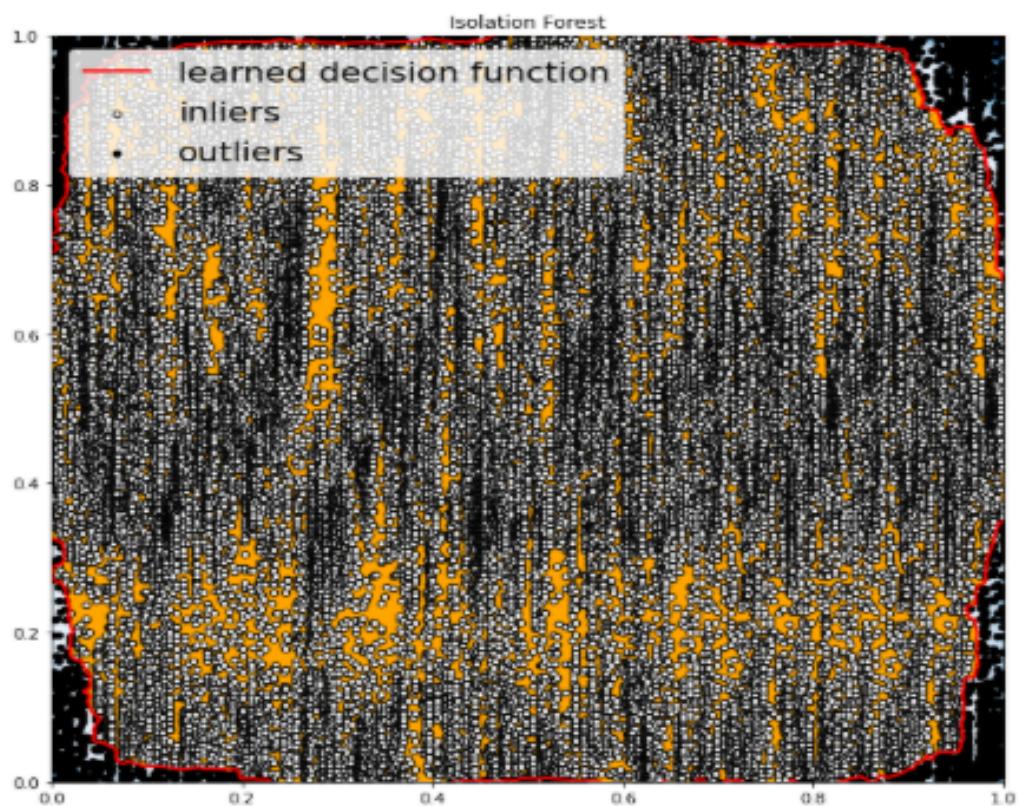


Figure 3.29: Isolation Forest considering Wind direction v/s Hour

Considering Power generated by system v/s Hour, the results were: OUTLIERS : 2628
INLIERS : 49932 Isolation Forest

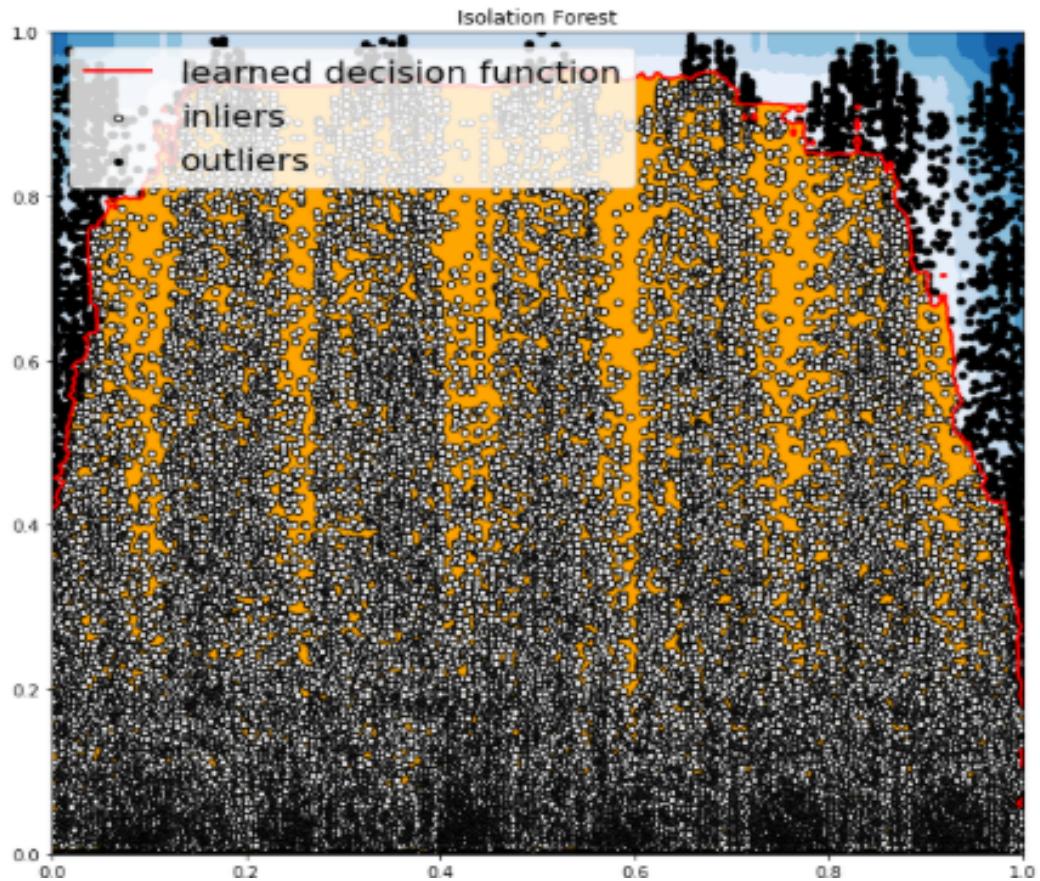


Figure 3.30: Isolation Forest considering Power generated by system v/s Hour

5. K Nearest Neighbors (KNN)

Considering Air temperature v/s Hour, the results were: OUTLIERS : 1936 INLIERS : 50624 K Nearest Neighbors (KNN)

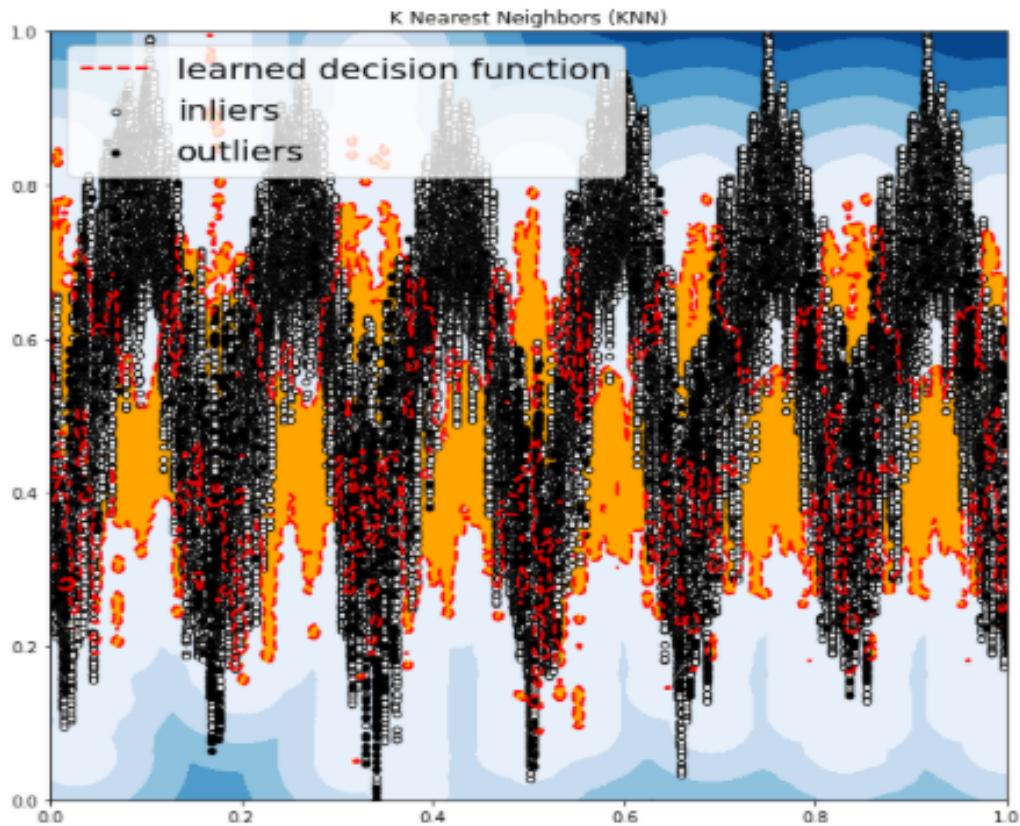


Figure 3.31: KNN considering Wind direction v/s Hour

Considering Pressure v/s Hour, the results were: OUTLIERS : 1936 INLIERS : 50624 K
Nearest Neighbors (KNN)

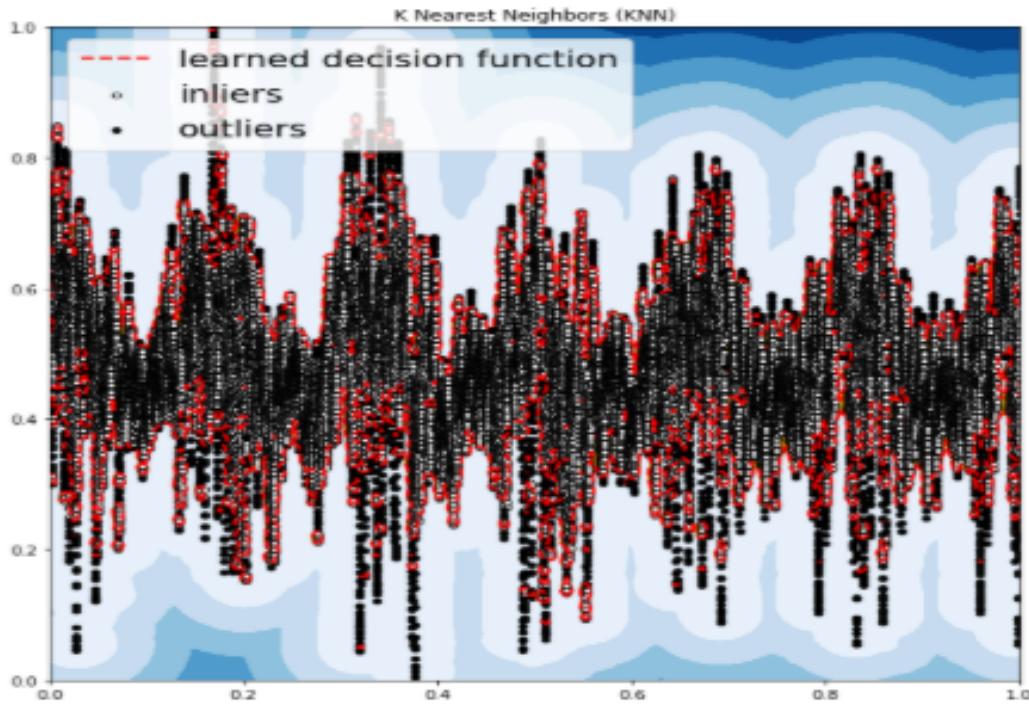


Figure 3.32: KNN considering Pressure v/s Hour

Considering Wind speed v/s Hour, the results were: OUTLIERS : 1893 INLIERS : 50667
K Nearest Neighbors (KNN)

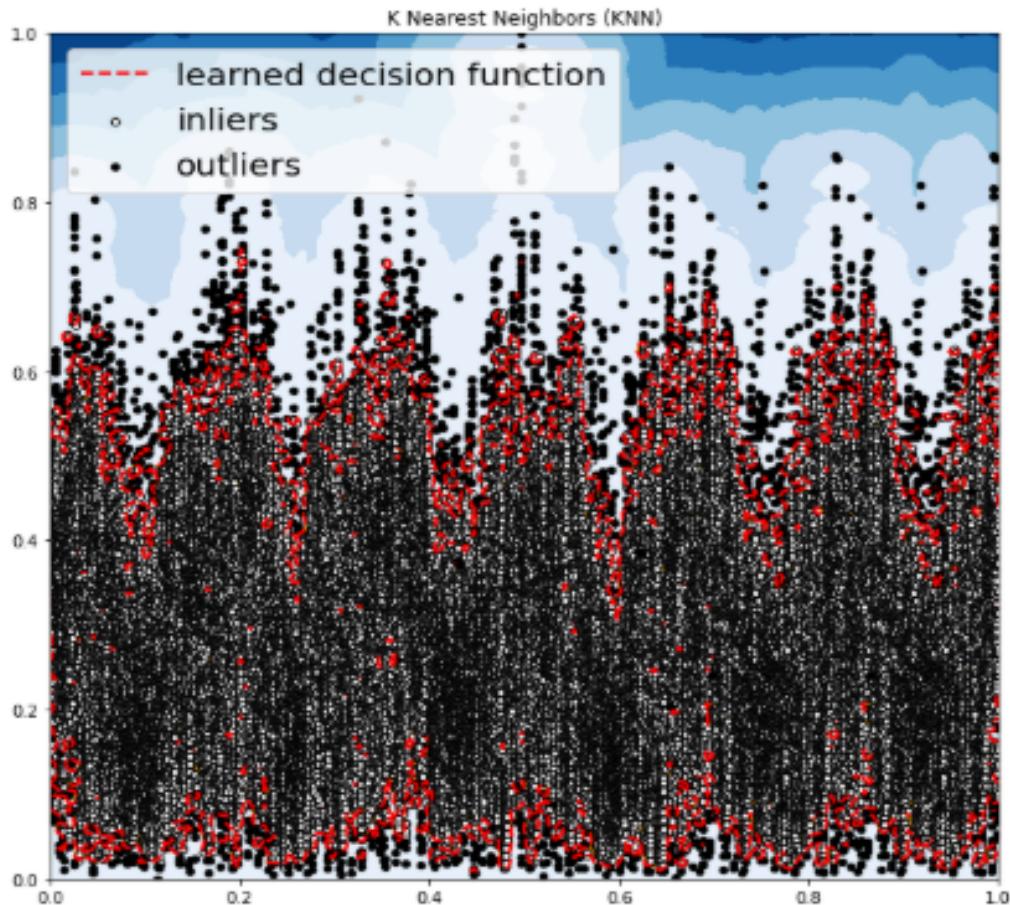


Figure 3.33: KNN considering Wind speed v/s Hour

Considering Wind direction v/s Hour, the results were: OUTLIERS : 1553 INLIERS : 51007 K Nearest Neighbors (KNN)

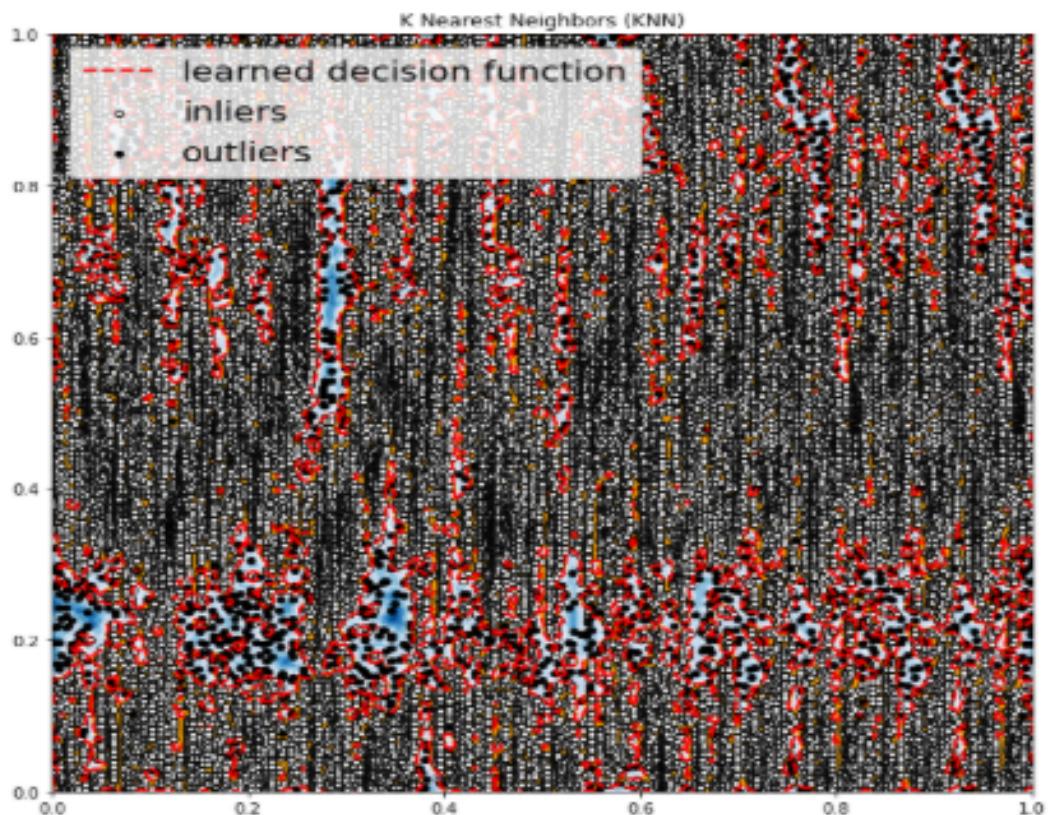


Figure 3.34: KNN considering Wind direction v/s Hour

Considering Power generated by system v/s Hour, the results were: OUTLIERS : 1712
INLIERS : 50848 K Nearest Neighbors (KNN)

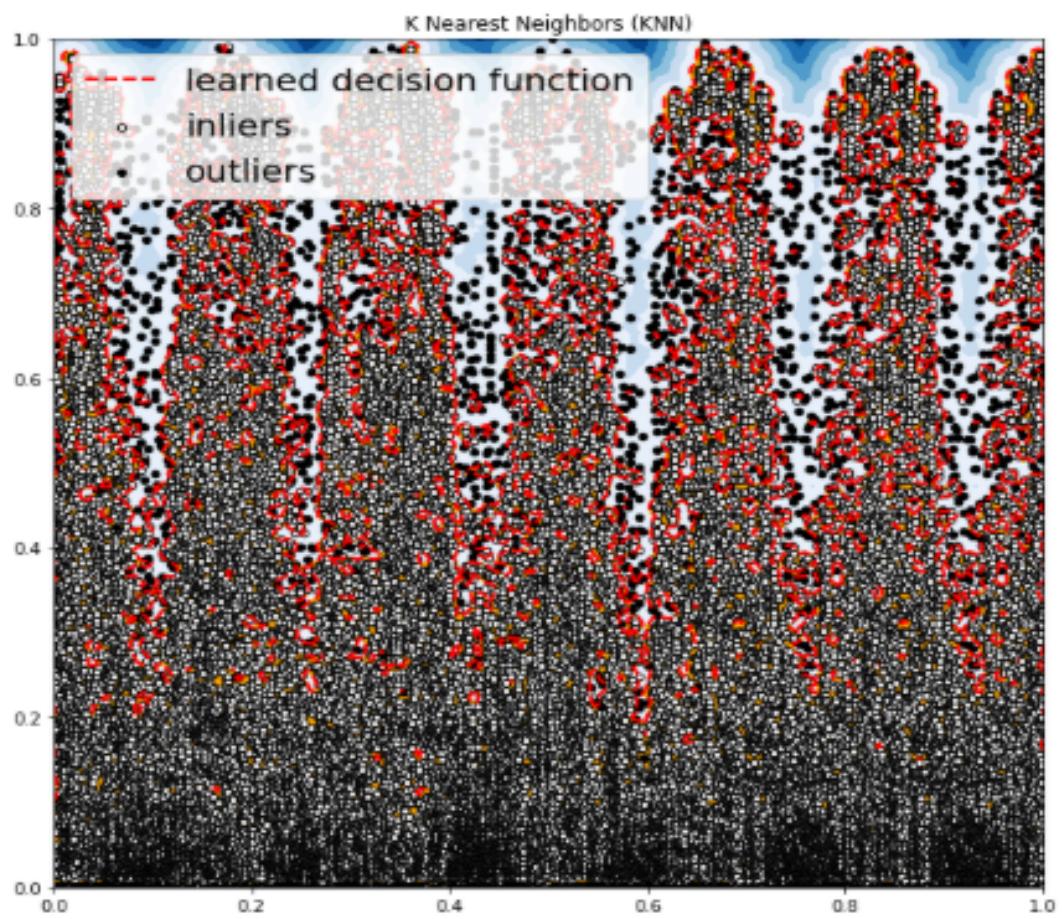


Figure 3.35: KNN considering Power generated by system v/s Hour

Chapter 4

Feature Engineering

Feature Engineering is a fundamental step in machine learning. A good model gets the best results from the data. Essentially, feature engineering also aims to do just that. It transforms raw data into features and keeps the most relevant ones to eliminate overfitting as well as shortens the time required for training, optimizing hyperparameters and inferring results. Ultimately it improves the model performance.

4.1 PEARSON CORRELATION COEFFICIENT

Pearson Correlation Coefficient measures the strength of association of two variables, in both the magnitude and direction of the relationship. Correlation between two variables can lie between -1 and +1. 1 would indicate a perfect relationship and 0 indicates there is no relationship between the variables. The sign indicates the direction of the relationship, + sign means a positive relationship and - sign stands for a negative relationship.

4.2 JOINT MUTUAL INFORMATION BETWEEN POWER AND OTHER INPUT VARIABLES

Mutual Information can be expressed as the amount of information provided by variable, which reduces the uncertainty of another variable. It shows how similar the joint distribution of these two variables is to the products of the factored marginal distributions. In feature selection, the intent is to maximise the mutual information between the subset of selected features.

4.3 MEAN SQUARE ERROR CONTRIBUTION BY VARIABLES

Mean Square Error (MSE) contribution by individual variables is used as a criteria in feature engineering by considering how different variables contribute to the MSE.

4.4 RESULTS AND DISCUSSION

The following heat maps show the correlation between various variables in the dataset under consideration for various time periods.

4.4.1 Pearson Correlation Coefficient

1 year

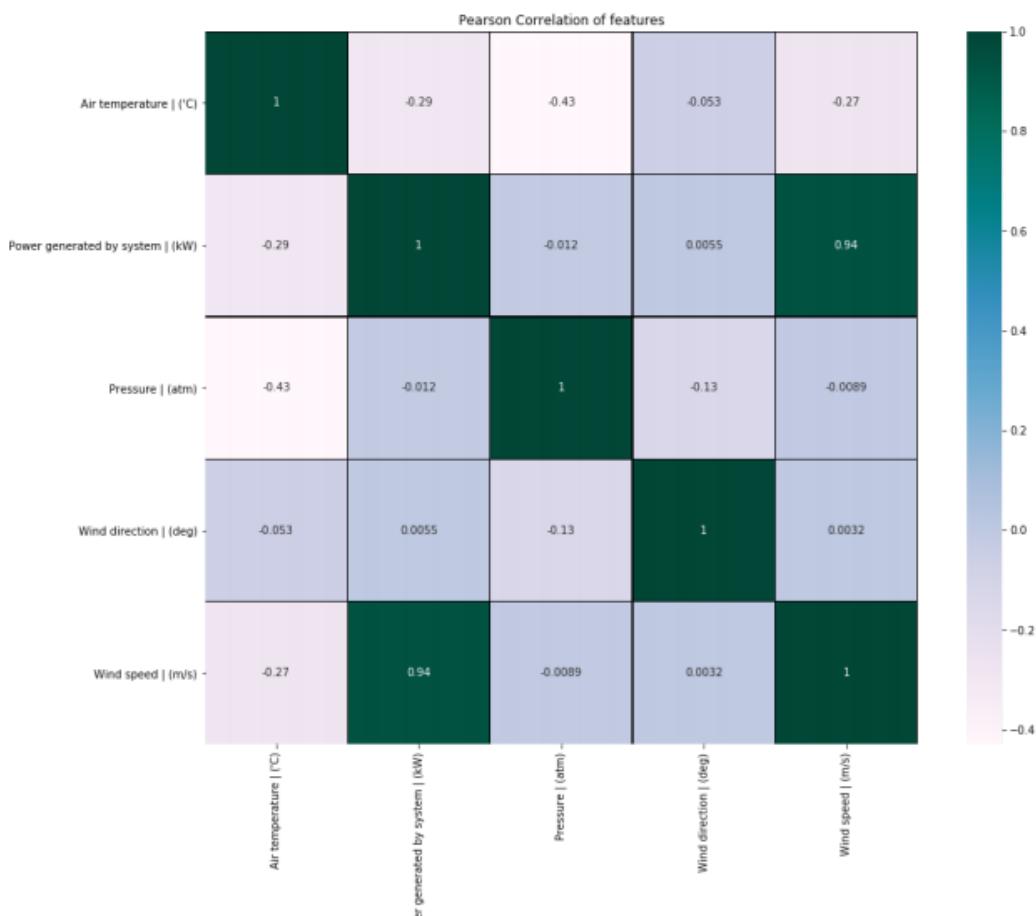


Figure 4.1: Heat Map - 1 year data

2 years

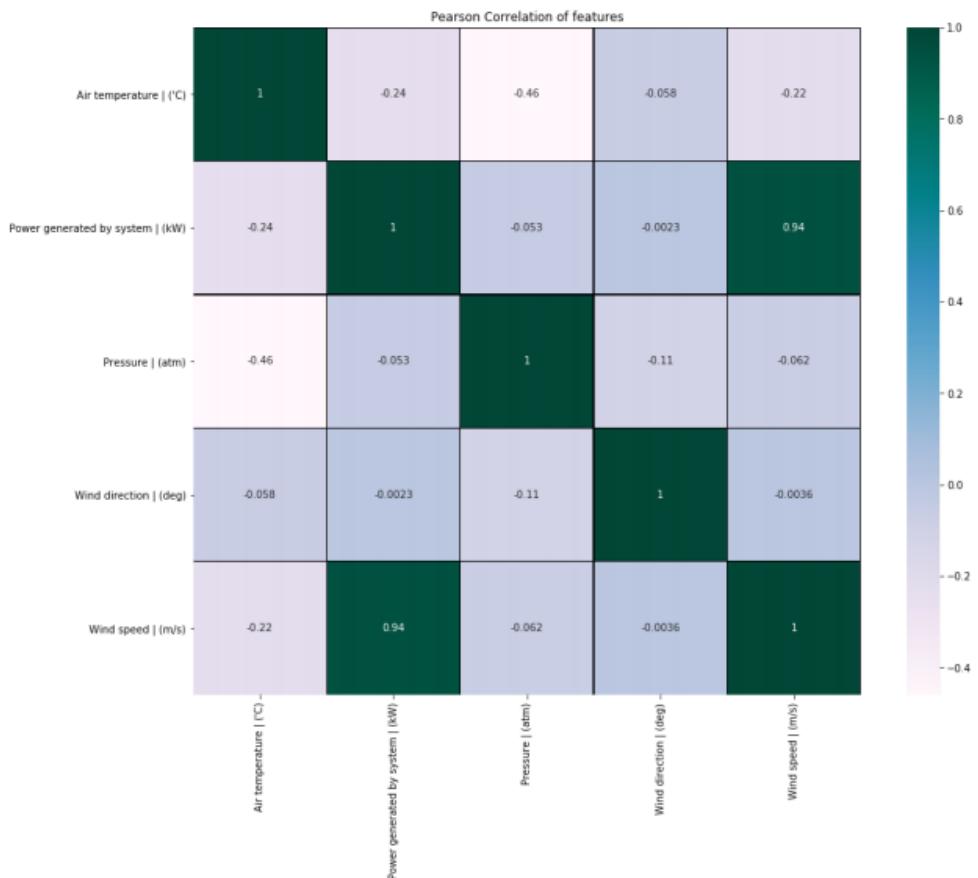


Figure 4.2: Heat Map - 2 years data

3 years

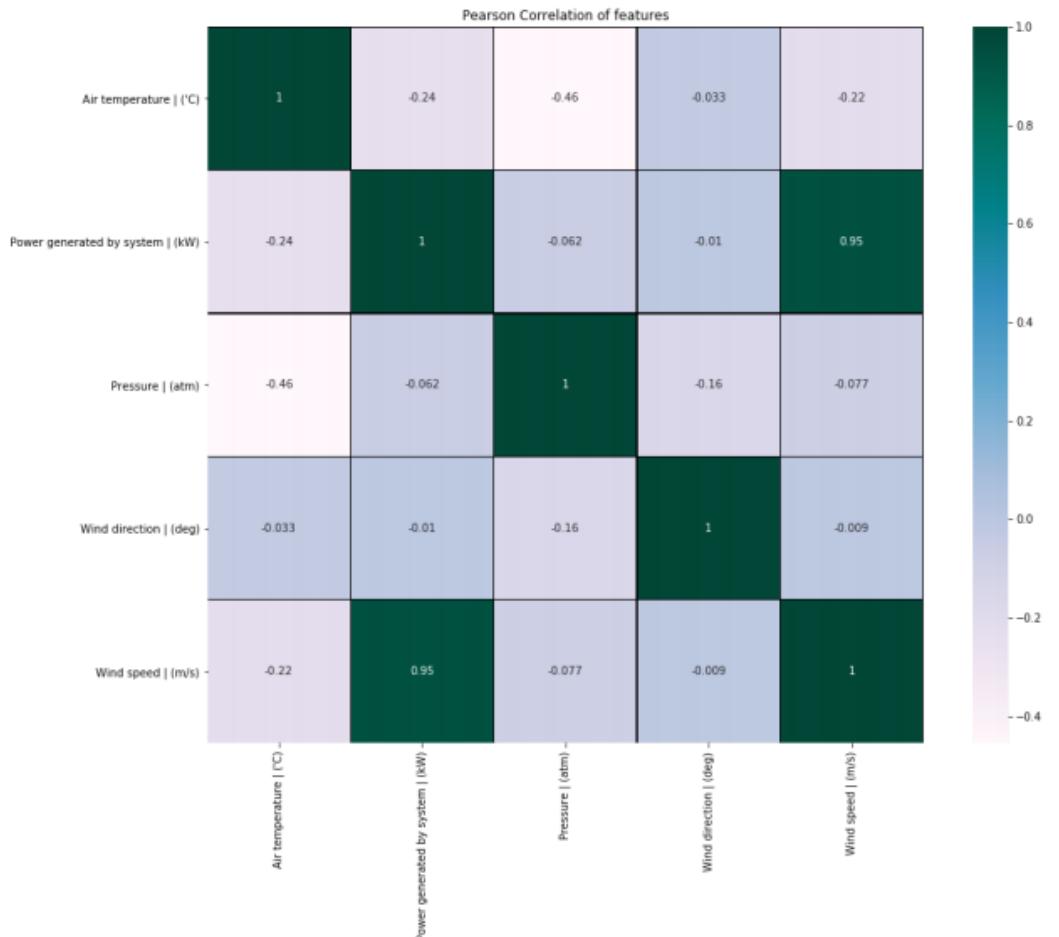


Figure 4.3: Heat Map - 3 years data

4 years

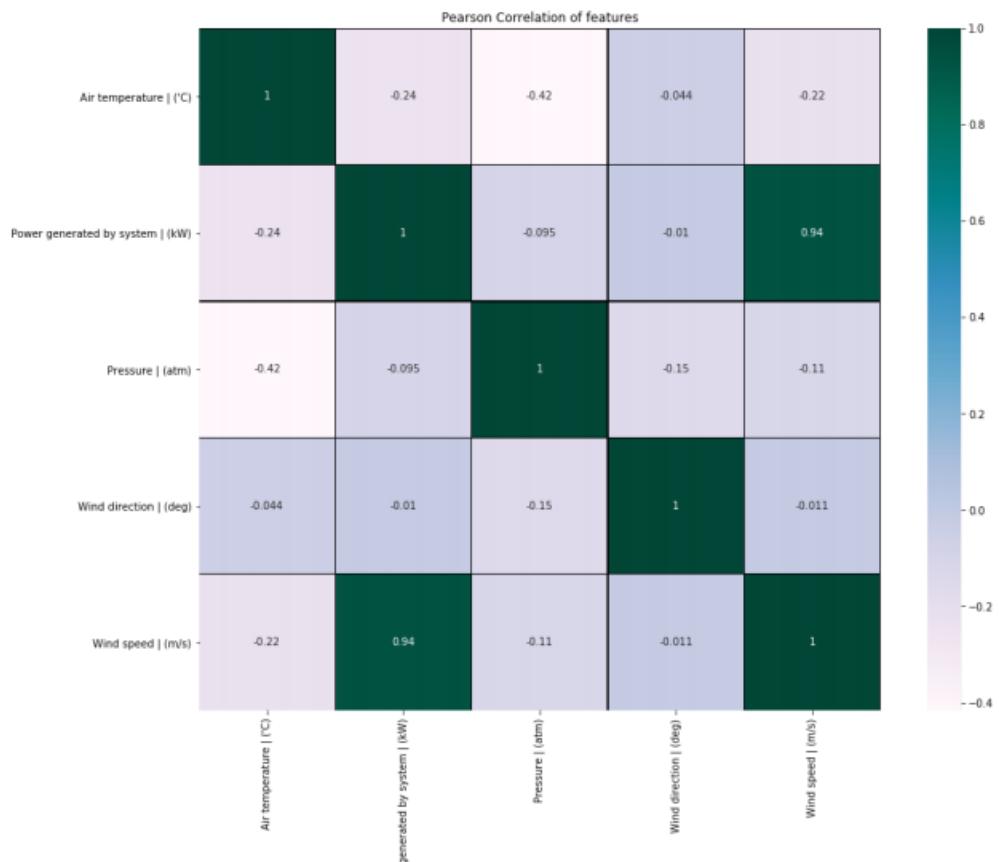


Figure 4.4: Heat Map - 4 years data

5 years

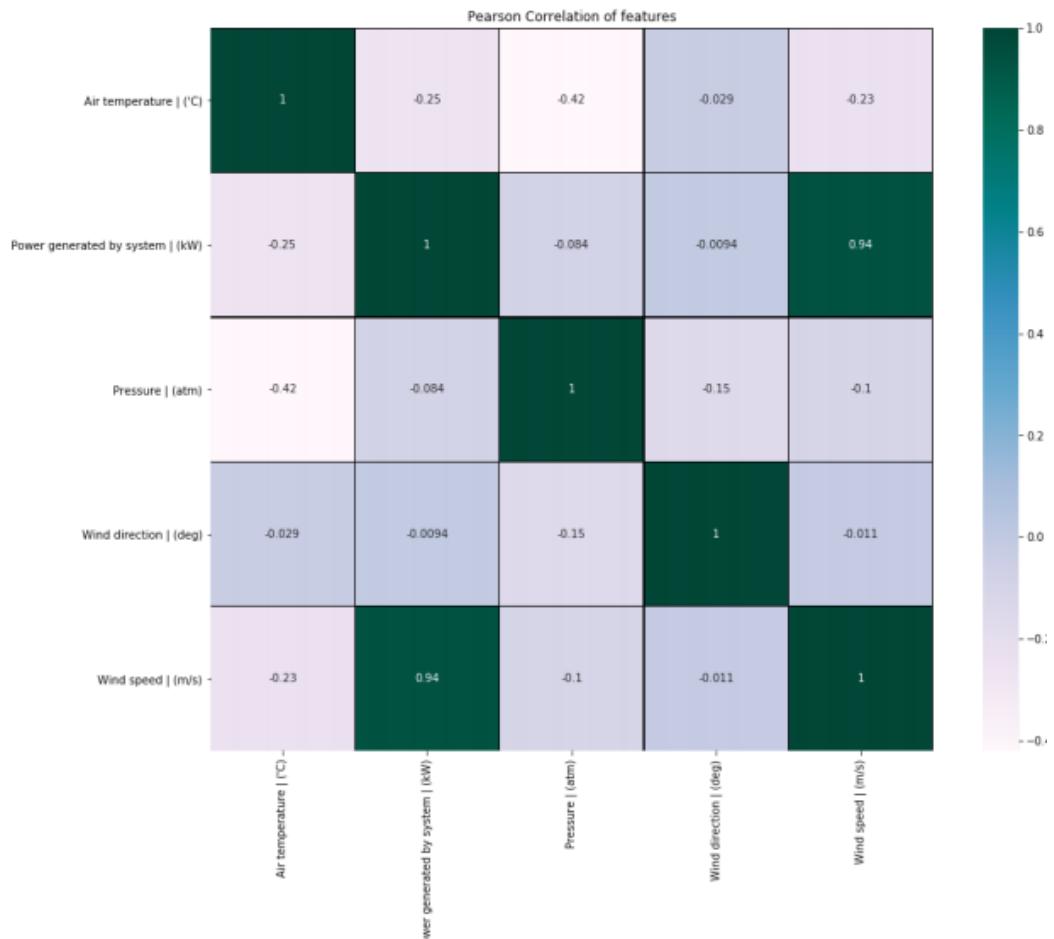


Figure 4.5: Heat Map - 5 years data

6 years

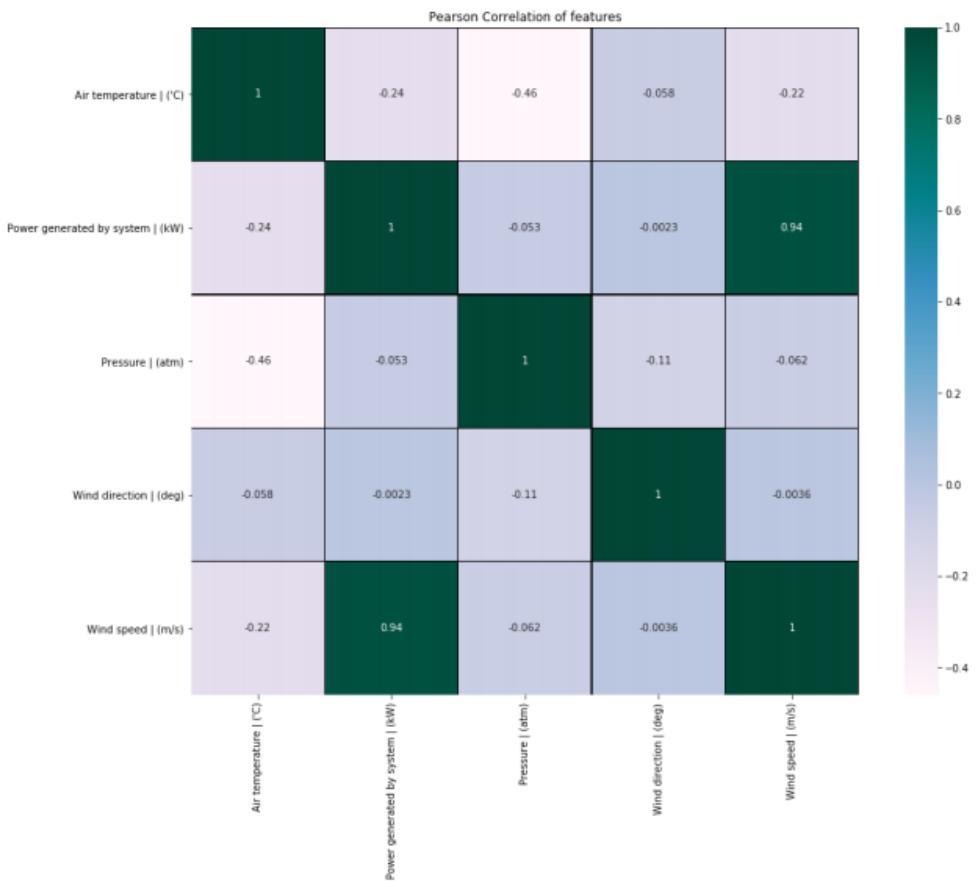


Figure 4.6: Heat Map - 6 years data

It is observed that for all time periods, the highest correlation is between pressure and wind speed, which is +0.94 which is considered a high degree of correlation. There is a moderate degree of correlation between pressure and air temperature which lies between -0.42 to -0.46. Wind speed vs air temperature and wind direction vs pressure both have low correlations while it is negligible for all the other pairs of variables.

4.4.2 Joint Mutual Information between Power and Other Input Variables

For the dataset, the joint mutual information measurements for various test sizes are as follows

10 % test size

Wind speed (m/s)	2.945681
Air temperature ('C)	0.097564
Pressure (atm)	0.046577
Wind direction (deg)	0.034303

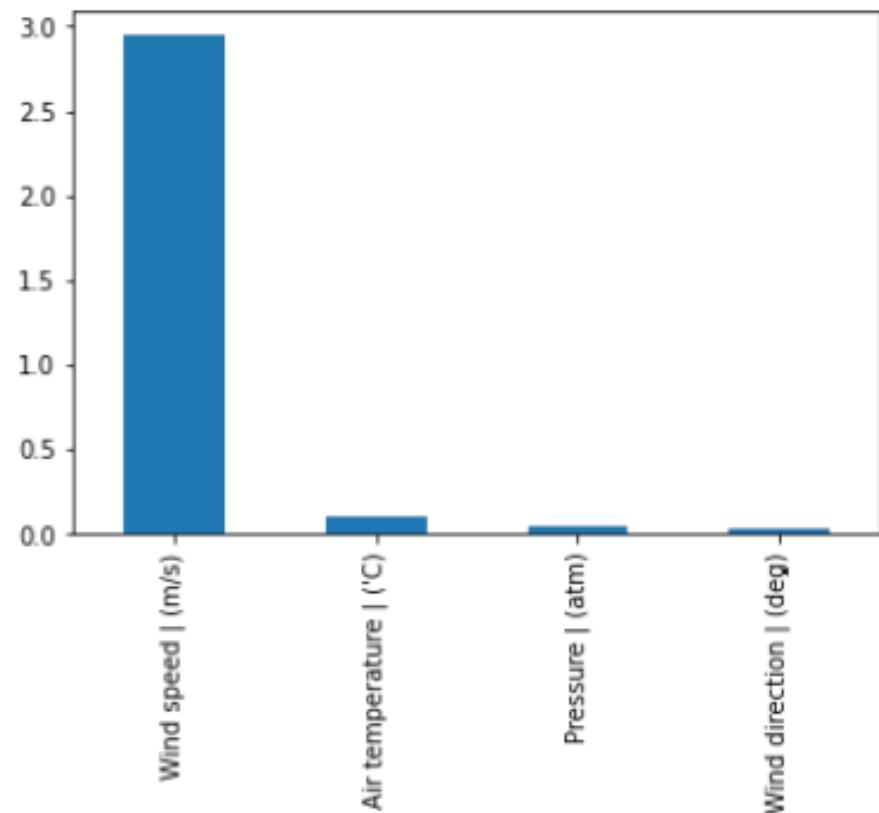


Figure 4.7: Joint Mutual Information - test size 10%

20 % test size

Wind speed (m/s)	2.943454
Air temperature ('C)	0.094376
Pressure (atm)	0.043357
Wind direction (deg)	0.034355

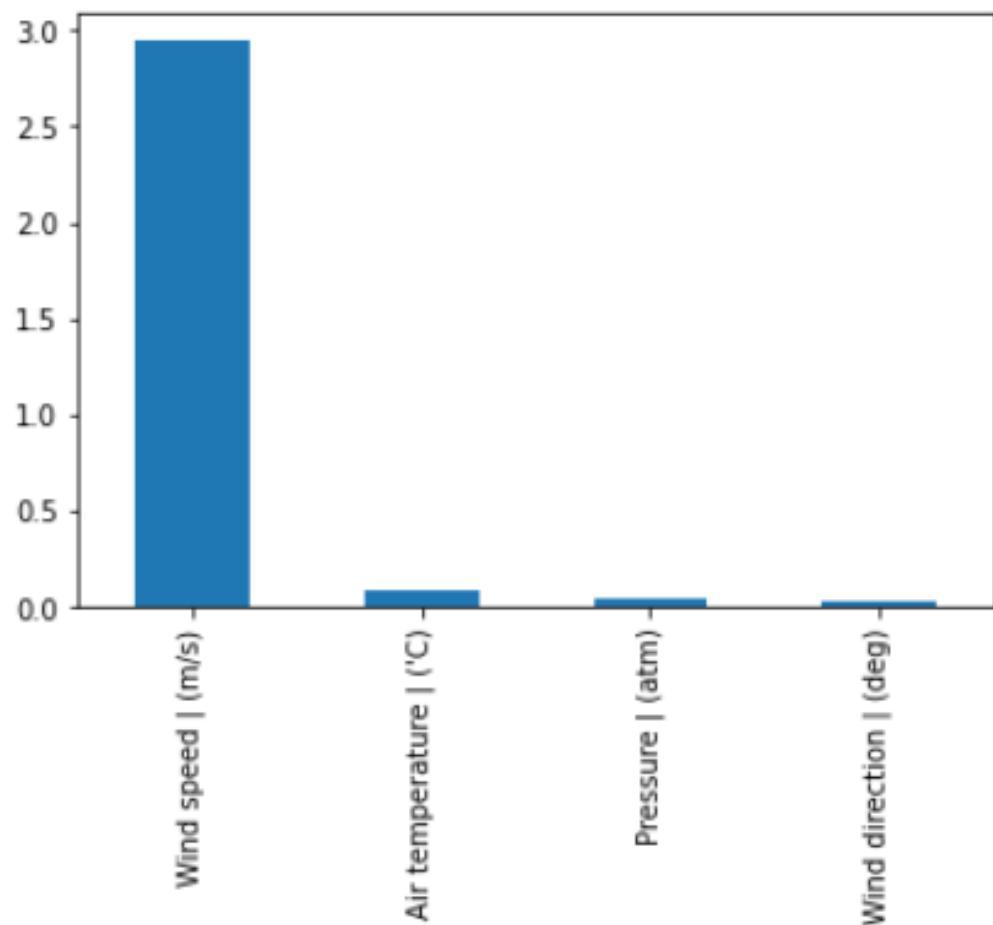


Figure 4.8: Joint Mutual Information - test size 20%

30 % test size

Wind speed (m/s)	2.949441
Air temperature ('C)	0.103266
Pressure (atm)	0.045281
Wind direction (deg)	0.040628

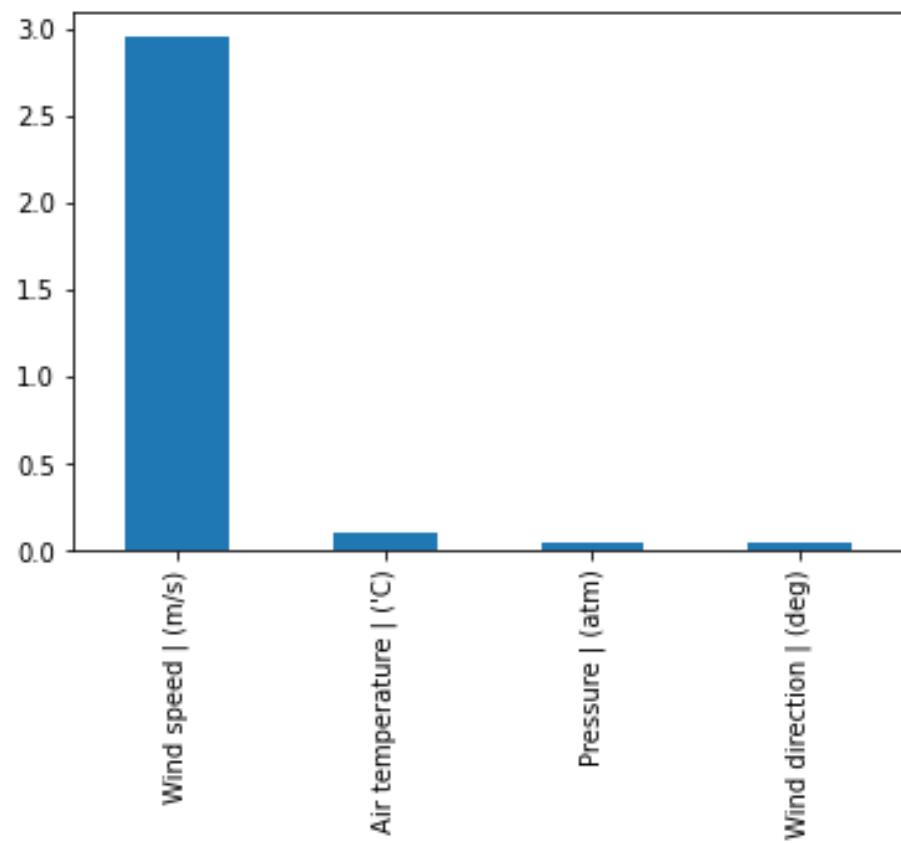


Figure 4.9: Joint Mutual Information - test size 30%

40 % test size

Wind speed (m/s)	2.949908
Air temperature ('C)	0.101034
Pressure (atm)	0.046669
Wind direction (deg)	0.028559

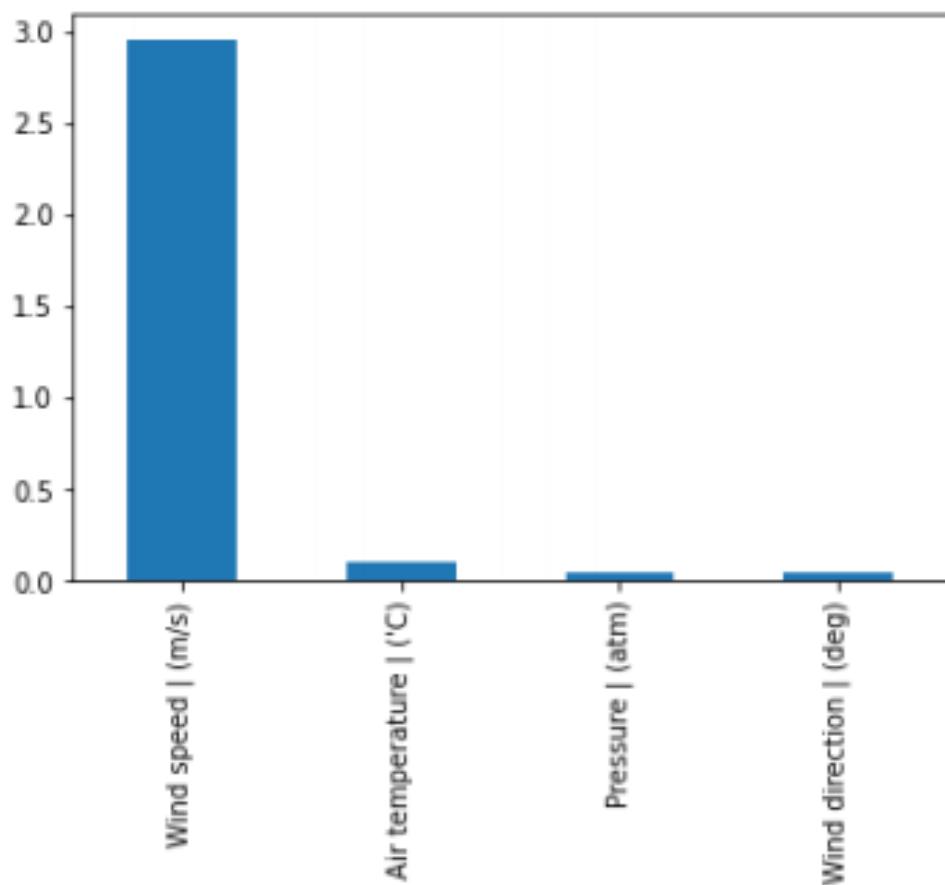


Figure 4.10: Joint Mutual Information - test size 40%

50 % test size

Wind speed (m/s)	2.942307
Air temperature ('C)	0.095968
Pressure (atm)	0.039919
Wind direction (deg)	0.024412

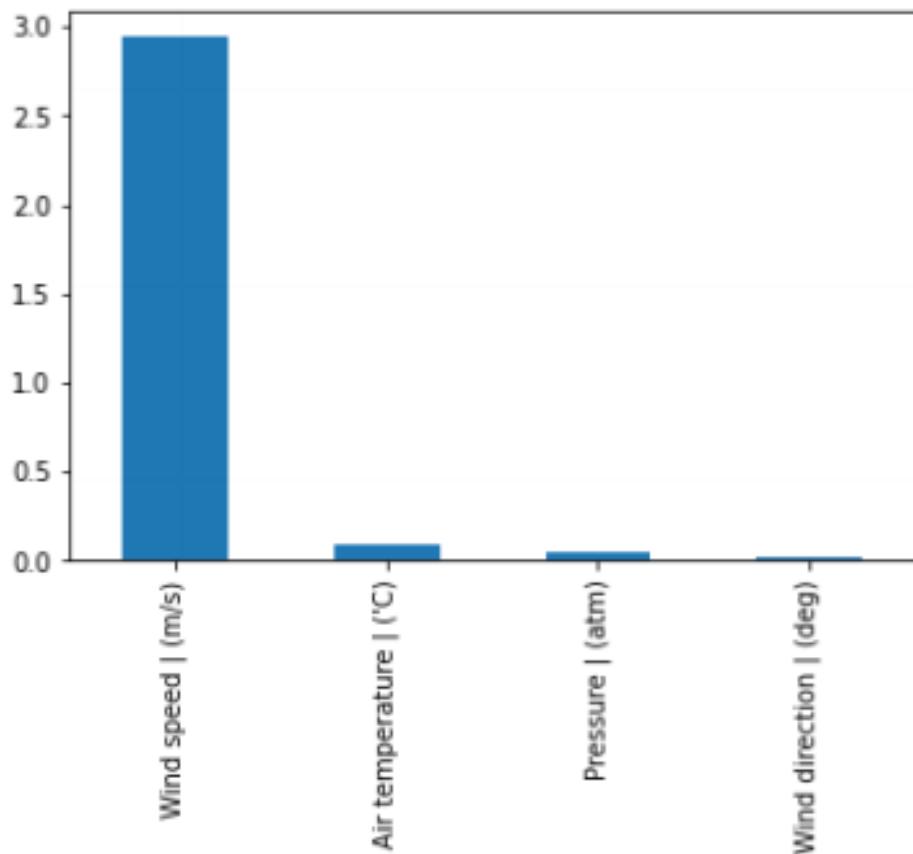


Figure 4.11: Joint Mutual Information - test size 50%

4.4.3 Mean Square Error Contribution by Variables

For all different test sizes, Wind direction and pressure are found to be the major contributing variables to mean square error, except for 10% test size, where wind speed dominates.

10 % test size

Wind direction (deg)	3.015242e+08
Pressure (atm)	3.004923e+08
Air temperature ('C)	2.808879e+08
Wind speed (m/s)	3.305857e+07

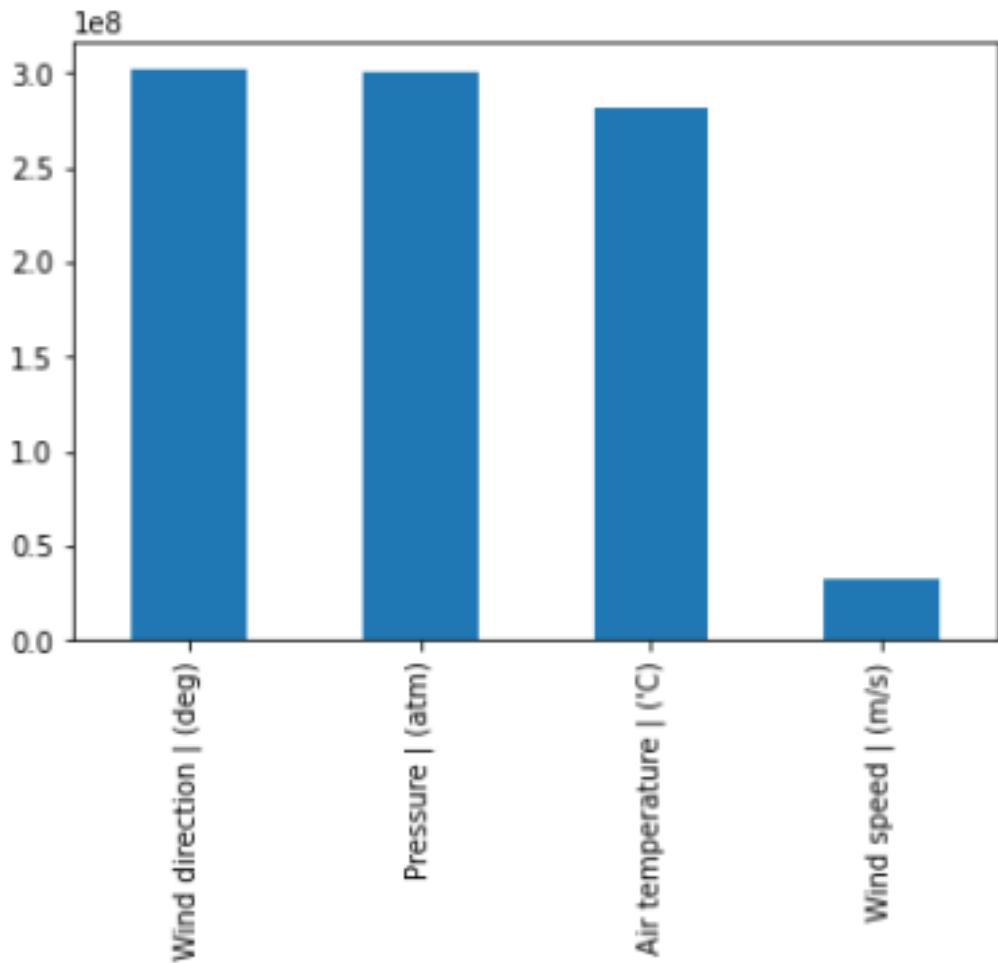


Figure 4.12: MSE contribution - test size 10%

20 % test size

Test Size = 20%

Wind direction (deg)	2.882219e+08
Pressure (atm)	2.867022e+08
Air temperature ('C)	2.688274e+08
Wind speed (m/s)	3.175021e+07

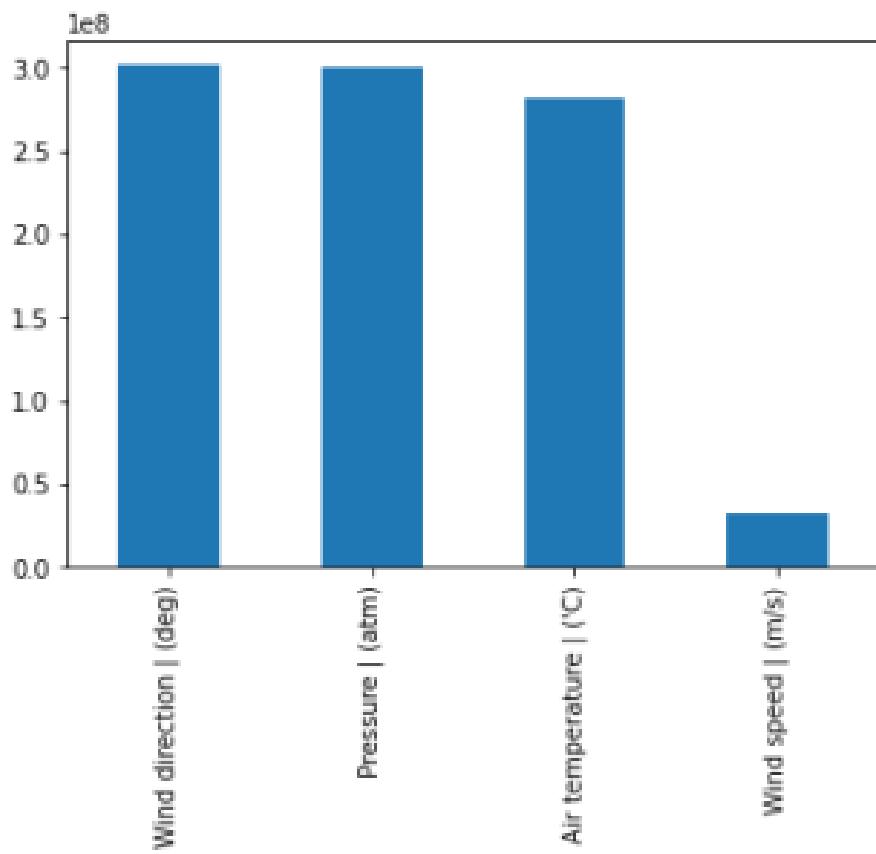


Figure 4.13: MSE contribution - test size 20%

30% test size

Wind direction (deg)	2.952275e+08
Pressure (atm)	2.926561e+08
Air temperature ('C)	2.774466e+08
Wind speed (m/s)	3.215393e+07

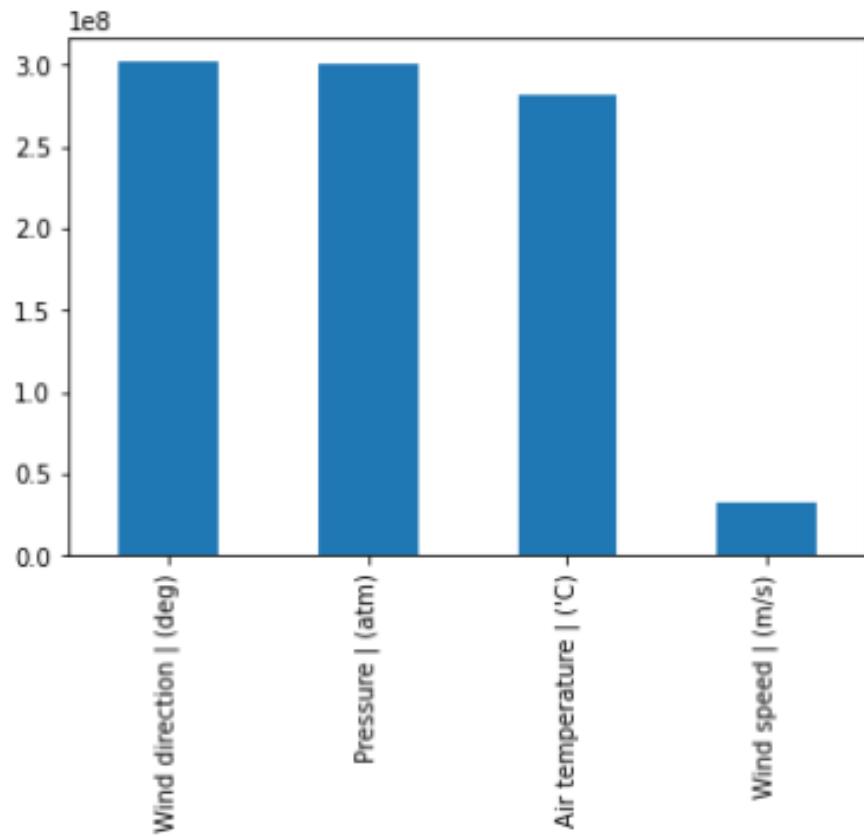


Figure 4.14: MSE contribution - test size 30%

40% test size

Wind direction (deg)	2.965121e+08
Pressure (atm)	2.942574e+08
Air temperature ('C)	2.774915e+08
Wind speed (m/s)	3.254670e+07

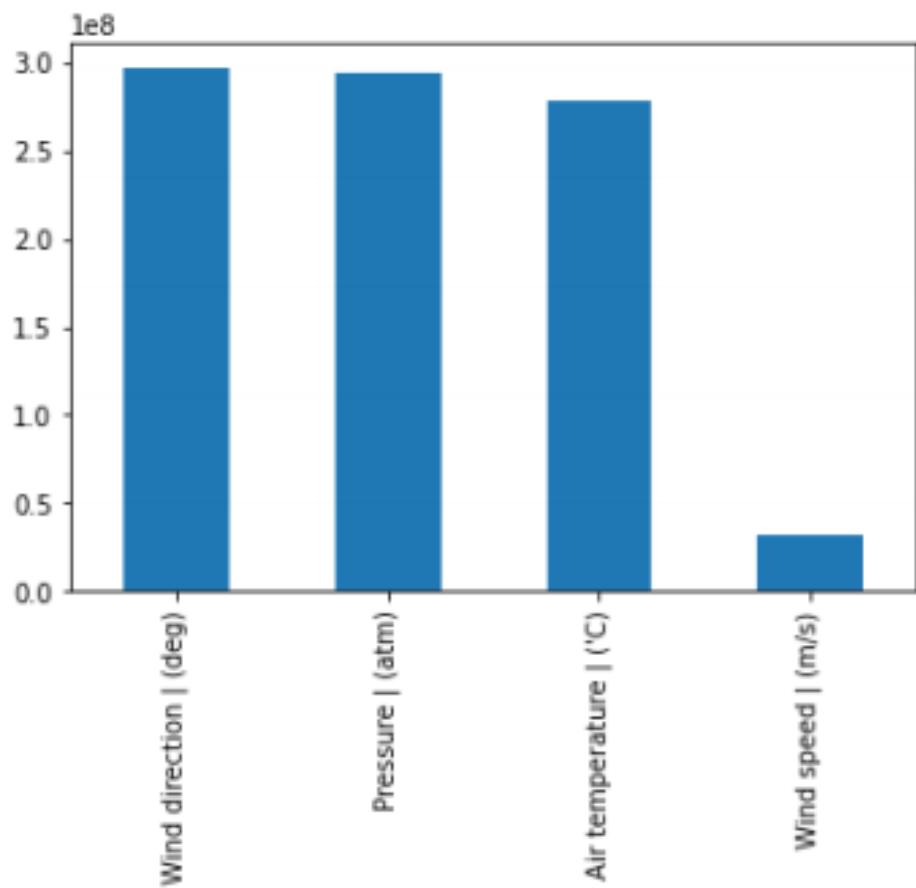


Figure 4.15: MSE contribution - test size 40%

50 % test size

Wind direction (deg)	2.963426e+08
Pressure (atm)	2.941518e+08
Air temperature ('C)	2.771177e+08
Wind speed (m/s)	3.252326e+07

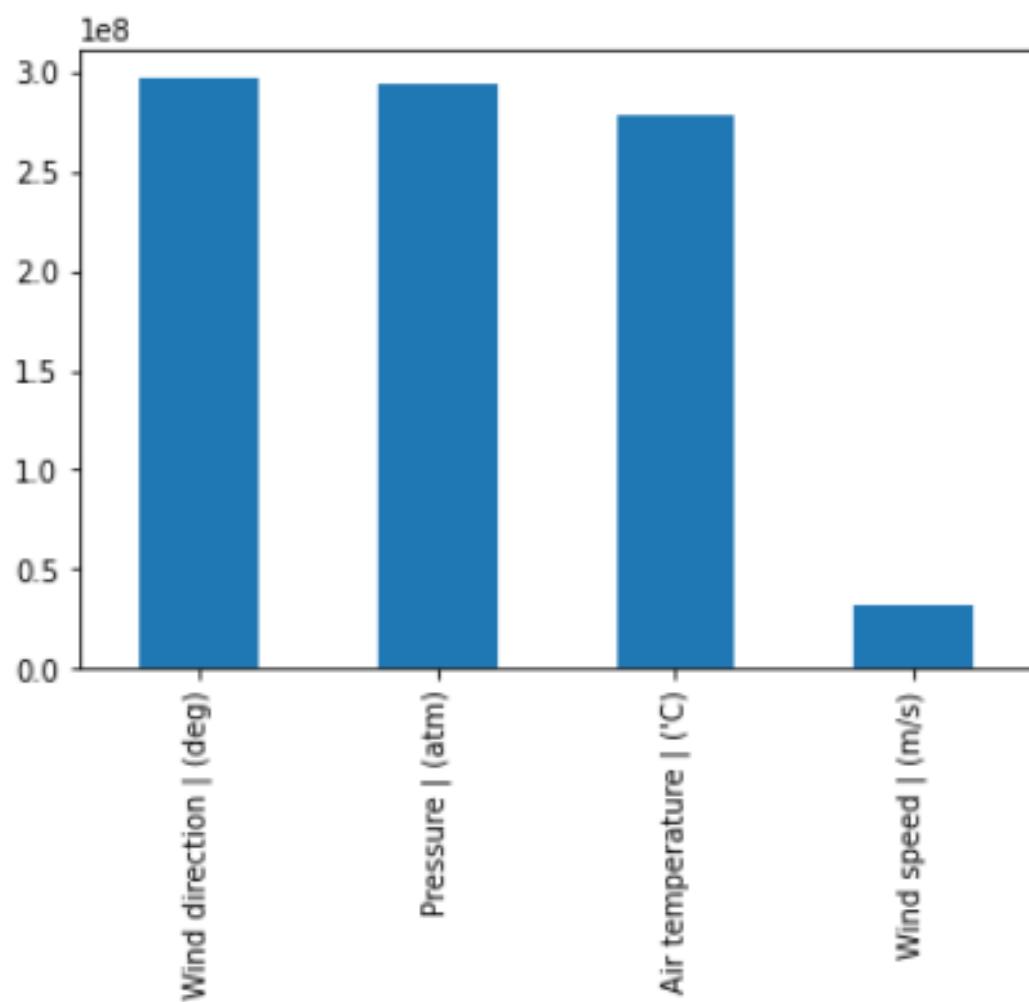


Figure 4.16: MSE contribution - test size 50%

Chapter 5

Test for Randomness

5.1 RANDOM WALK

A random walk is a stochastic process that describes a random motion along a mathematical space. The simplest example of a random walk can be described on the set of integers which starts at 0 and moves +1 or -1 in each step with equal probability. The future step depends only on its previous step. Random walks can be of different types, but by default are used to refer to Markov chains or processes. Time dependent processes are also defined as random walks where their characteristics and properties are considered. To give a formal definition, it is a time series represented by

$$x_t = x_{t-1} + w_t \quad (5.1)$$

where w_t denotes the discrete white noise series which is independently and identically distributed (IID) with a mean of 0. The white noise series is assumed to be having a Gaussian nature here. The theory of random walks have been applied to various fields to study the behaviour of many processes. It is therefore considered a fundamental model of recorded stochastic activity. Random walks have long trends up or down and sudden unpredictable changes in direction.

5.2 DIFFERENCING OF TIME SERIES

Differencing is a method of making a non-stationary time series stationary by computing the differences between successive observations. Obtained with the help of certain transformations like logarithms, differencing helps remove the trend and seasonality and stabilize the mean of a non-stationary time series.

The differenced series is the difference between successive observations in the original series and can be written as $x'_t = x_t - x_{t-1}$

When the differenced series is white noise, the original series can be written as $x_{t-1} - x_t = w_t$ where w_t denotes the white noise series. Rearranging this gives the random walk model $x_t = x_{t-1} + w_t$.

5.3 TESTS FOR RANDOMNESS

5.3.1 Visual Test

Random walks have long trends up or down and sudden unpredictable changes in direction. The random walk nature is observed for the given time series in wind speed vs hour and is shown below.

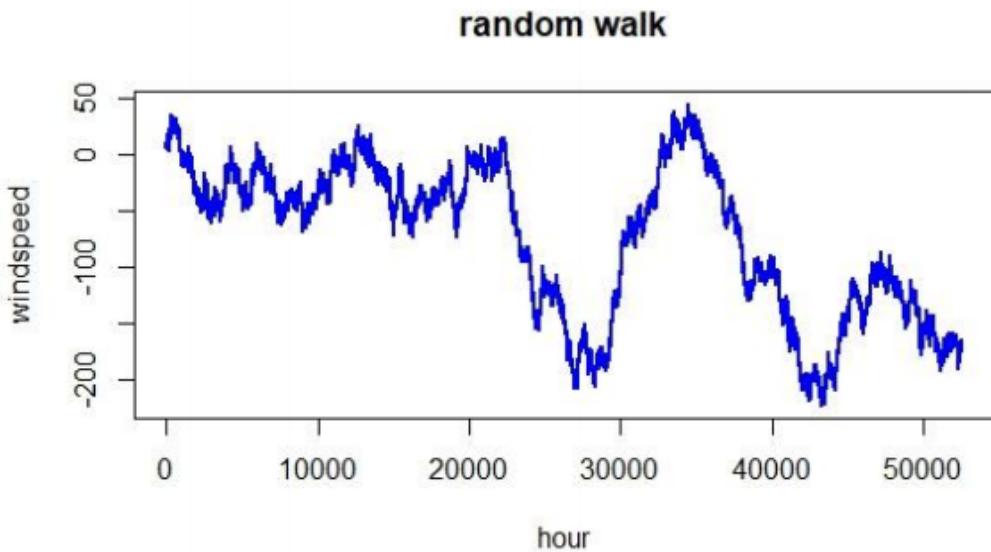


Figure 5.1: Random walk nature - wind speed vs time(in hour)

5.3.2 Differencing of Time Series

Differencing of time series given by $x_t - x_{t-1}$ results in a series involving w_t terms. Since the white noise series follows a Gaussian nature, the plot depicting the white noise series (differenced series) follows a stationary behavior (constant mean and constant variance) which is shown below.

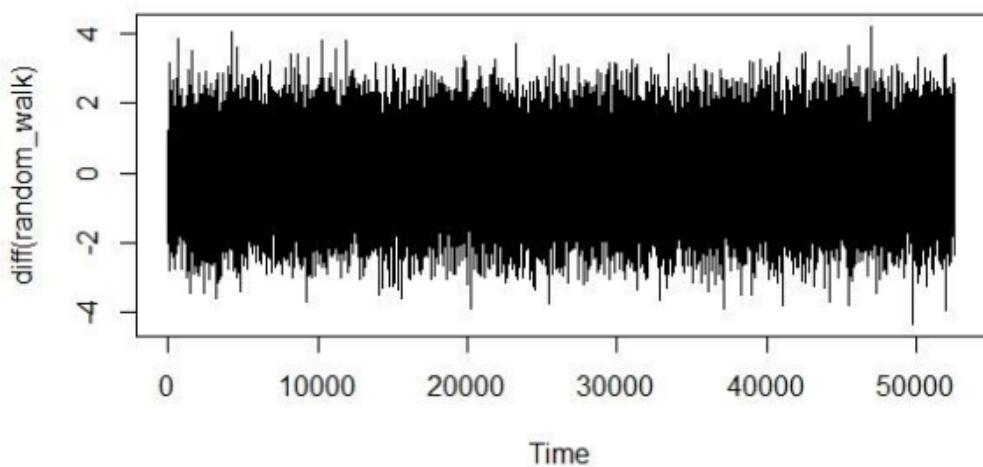


Figure 5.2: Stationary Behaviour of Differenced Series

5.3.3 Comparison of Standard Deviations

Upon comparison of standard deviations of the original series and the differenced series, it was found that the differenced series had a standard deviation less than the standard deviation of the original series. The statistics of both the series are given below. The time series had a standard deviation of 3.009 whereas the differenced series had a standard deviation of 1.101.

<i>Wind speed</i>	<i>Difference</i>		
Mean	5.944999	Mean	-0.00013
Standard I	0.013125	Standard I	0.004806
Median	5.678	Median	0.001
Mode	3.273	Mode	-0.032
Standard I	3.009099	Standard I	1.101778
Sample V:	9.054677	Sample V:	1.213915
Kurtosis	-0.50835	Kurtosis	6.455446
Skewness	0.369913	Skewness	0.172704
Range	19.585	Range	21.141
Minimum	0.14	Minimum	-9.794
Maximum	19.725	Maximum	11.347
Sum	312469.1	Sum	-6.882
Count	52560	Count	52560

Figure 5.3: Comparison of Series Statistics

Chapter 6

Test for Non-linearity

6.1 NON-LINEARITY

Most real world systems are nonlinear in nature. A nonlinear system is one which does not give a proportional change in output to a change in input. Analysis of such systems are much more complex than linear systems. Linear analysis does not take into account many characteristics exhibited by a nonlinear system such as time-changing variance, asymmetric cycles, higher-moment structures, thresholds and breaks. The behaviour of nonlinear systems may appear unpredictable and chaotic and hence random, but they are not random and can be found out.

A number of tests are available in literature to identify non-linearity, one of them being the surrogate test. But since the validity of surrogate tests heavily depend on whether the time series is stationary, a number of tests are conducted check for non-stationarity before checking non-linearity.

6.2 STATIONARITY

Stationarity refers to the property of the time series where the statistical properties like mean, variance and autocorrelation structure does not vary over time. If a time series is stationary, then the process referring to the series is considered to be stochastic where the unconditional joint probability distribution does not vary over time.

6.3 VISUAL TEST

One of the easiest ways of determining whether the time series is stationary is by visual analysis after plotting the series. Even though it is very unreliable, it is possible to look for properties like level, trend, seasonality, etc. from simple plot against time. Heuristic methods can be employed with the help of some of these plots, to improve the reliability of the estimation of stationarity.

6.4 AUTOCORRELATION FUNCTION

Autocorrelation is the correlation of a signal with a delayed copy of itself. An autocorrelation function(ACF) refers to a plot of autocorrelation of a time series by lag. This is called a autocorrelation plot or correlogram. If the value of ACF function gradually decreases over time to zero, then the given time series can be said to be stationary.

The autocorrelation function is calculated using either of the following methods. Based on sample autocorrelations, the autocorrelation function is given by,

$$r_u = \frac{\sum_{t=u+1}^T (X_t - \bar{X}_{1+u}^T)(X_{t-u} - \bar{X}_1^{T-u})}{\sqrt{\sum_{t=u+1}^T (X_t - \bar{X}_{1+u}^T)^2 (X_{t-u} - \bar{X}_1^{T-u})^2}} \quad (6.1)$$

Based on scaled sample autocovariances, the autocorrelation function is given by,

$$g_u = \frac{\sum_{t=u+1}^T (X_t - \bar{X}_1^T)(X_{t-u} - \bar{X}_1^T)}{\sum_{t=u+1}^T (X_t - \bar{X}_1^T)^2} \quad (6.2)$$

where $X_1, X_2, X_3, \dots, X_N$ is the given time series and \bar{X}_v^{T-w} is the sample average X_v, \dots, X_{T-w} , for $0 \leq u \leq T$.

6.5 PARTIAL AUTOCORRELATION FUNCTION

A partial autocorrelation is a summary of the relationship between an observation in a time series with observations at prior time steps with the relationships of intervening observations removed. The partial autocorrelation at lag k is the correlation that results after removing the

effect of any correlations due to the terms at shorter lags.

When autocorrelation is calculated, it includes both direct and indirect correlations. Indirect correlations can be referred as the ones having linear relationship with direct correlation. This removed by partial autocorrelation to obtain only direct correlation.

6.6 KPSS (KWIATKOWSKI-PHILLIPS-SCHMIDT-SHIN) TEST

In the KPSS test, the null hypothesis assumes that stationarity around a mean or linear trend, while alternate hypothesis assumes that the presence of a unit root. This is exactly opposite to most unit root tests like Augmented Dickey Fuller Test(ADF) where null hypothesis assumes the presence of a unit root.

Null Hypothesis : The process is trend stationary.

Alternate Hypothesis : The series has a unit root (series is not stationary).

The test is based on linear regression and it calculates KPSS statistic value, p-value, truncation lag parameter and critical values. If the test statistic is greater than the critical value, we reject the null hypothesis and series is not stationary. If the test statistic is less than the critical value, if fail to reject the null hypothesis and series is stationary.

6.7 SURROGATE TEST

The surrogate test assumes that all the correlations are due to linear stochastic dynamics and all the spikiness comes from a distortion by the measurement procedure. Therefore, the null hypothesis assumed is that the data is generated from a linear stochastic process. In order to check whether the hypothesis is true, a set of statistic parameters generated from the process by Monte Carlo re-sampling. It is then compared to that of the time series and if the parameters remain close to each other, then the null hypothesis is accepted and the time series is considered to be linear.

The most used statistic is time asymmetry since it is strong signature of non-linearity. The validity of the surrogate test depends on the non-stationarity of the time series, and since it is already tested for non-stationarity before, it is assumed that the surrogate test can provide valid results for non-linearity.

6.8 RESULTS AND DISCUSSION

6.8.1 Visual Test

The Autocorrelation function is plotted using Python. The function `plotacf()` can be used for this. Figure 6.1 shows the autocorrelation function against lag.

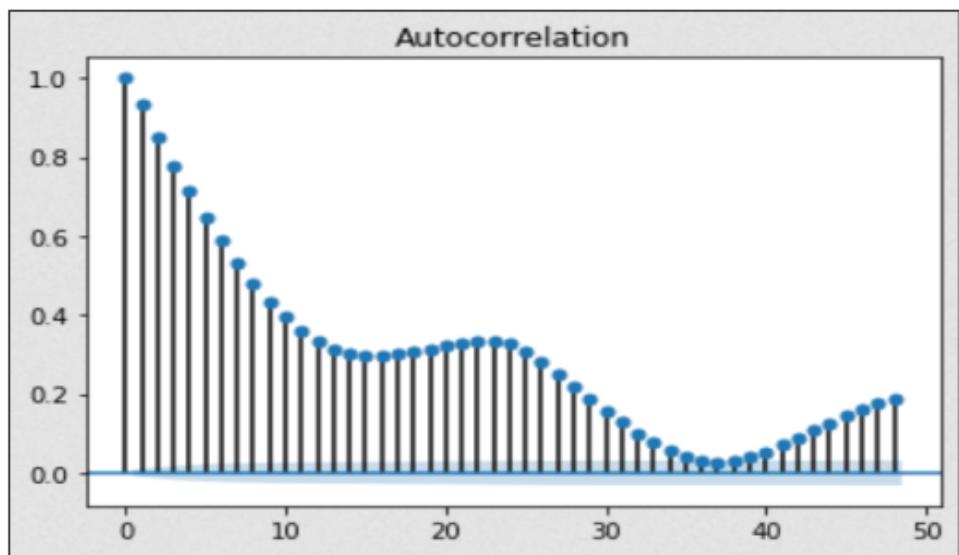


Figure 6.1: Autocorrelation vs lag

It is observed that the autocorrelation function doesn't degrade to zero quickly, showing the non-stationarity of the given time series as shown in Figure 6.1.

The partial autocorrelation function is plotted using `plotpacf()` function of Python. Figure 6.2 shows partial autocorrelation function against lag and is limited to the first 50 lags.

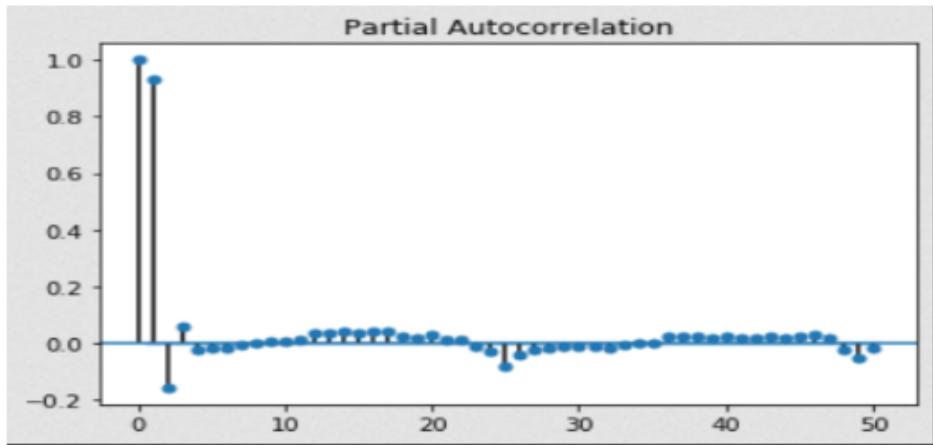


Figure 6.2: Partial Autocorrelation vs lag

Unlike ACF, PACF shows the direct relationship between series and lag as shown in Figure 6.2. Since, PACF values exist even after the first few values, the given time series can be said as non-stationary.

The same can be observed in logarithmic transformations of autocorrelation functions as shown in Figure 6.3 and Figure 6.4.

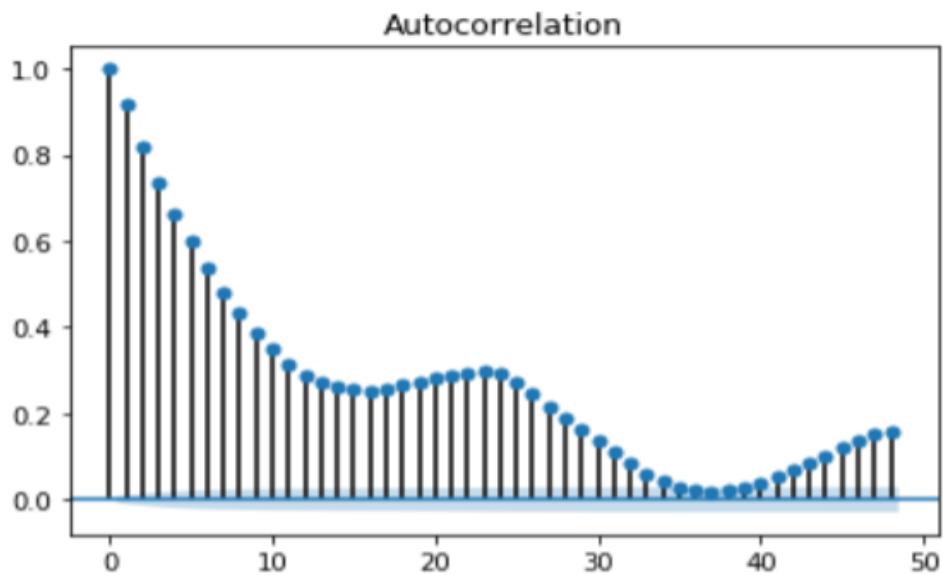


Figure 6.3: Logarithmic transformation of Autocorrelation vs lag

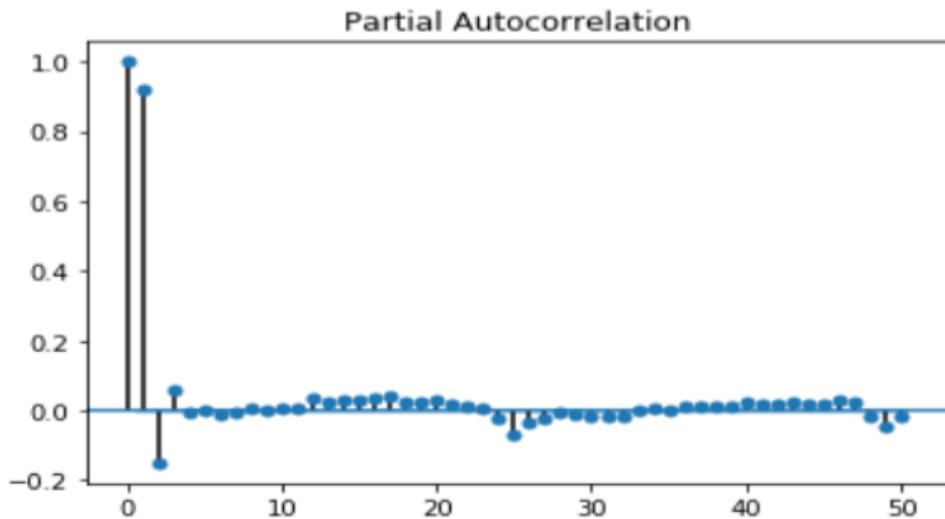


Figure 6.4: Logarithmic transformation of Partial Autocorrelation vs lag

6.8.2 KPSS (Kwiatkowski-Phillips-Schmidt-Shin) test

Null hypothesis: The process is stationary

Alternate hypothesis: The process is not stationary

The KPSS test obtains following results:

KPSS test statistic: 0.6136373975628752

p-value: 0.021396600221556802

Numerical lag: 58

Critical values for different confidence intervals is given in Table 6.1.

Confidence Intervals	Critical Values
10%	0.347
5%	0.463
2.5%	0.574
1%	0.735

Table 6.1: Critical Values

Since, the p-value is lower than all critical values, the alternate hypothesis is considered true and therefore, the given series is non-stationary.

6.8.3 Surrogate Test

The *nonlinearTseries* package of R programming environment is used for the surrogate test. This function tests the null hypothesis stating that the series is a Gaussian linear process. The test is performed by generating several surrogate data according to hypothesis and comparing the values of a discriminating statistic between both original data and the surrogate data. If the value of the statistic is the same as that of the original time series, then the null hypothesis is accepted and series is considered to be linear. Otherwise, the null hypothesis and the series is non-linear.

Null hypothesis: The data comes from a linear stochastic process

The statistic used for surrogate test is time symmetry which measures the asymmetry of a time series under time reversal.

The Figure 6.5 shows time symmetry for both original data and surrogate data.

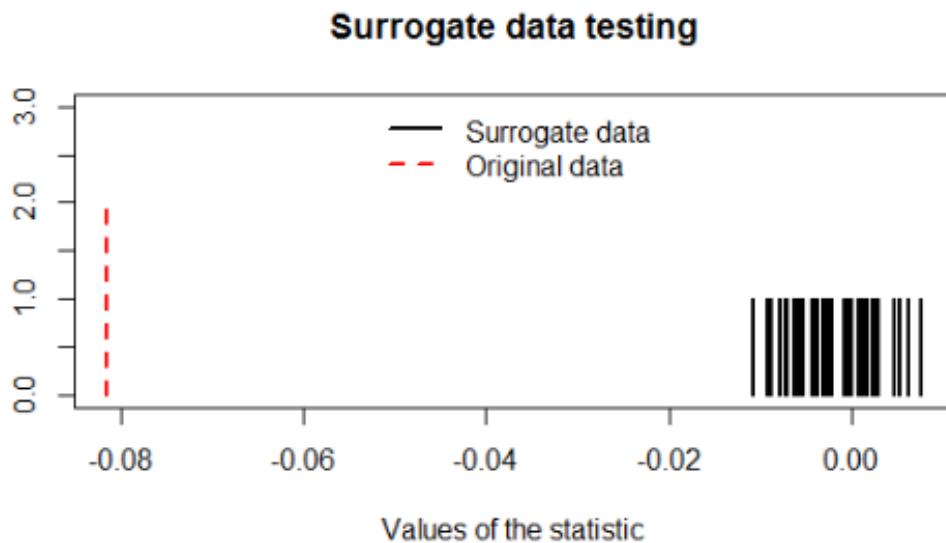


Figure 6.5: Surrogate data testing

The null hypothesis is rejected as the original statistic is significantly higher than the

surrogate statistic. Therefore, the time series is considered to be non-linear.

Chapter 7

Test for Chaoticity

7.1 CHAOS THEORY

Chaos theory deals with the study of chaos, where the system may appear random, but is actually deterministic with high complexity that after a while it is near impossible to predict. A chaotic dynamical system is also highly sensitive to initial conditions, that a small change in the initial value will result in a much bigger change as the time goes on.

One of the best examples of a chaotic system is a double pendulum which is shown in Figure 7.1. If the double pendulum starts in a slightly different position, it will end in an entirely different path.

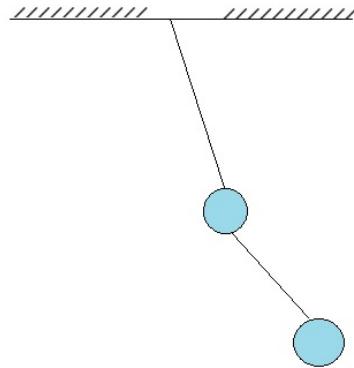


Figure 7.1: Double pendulum

Another example will be billiards games. You can never recreate a game of billiards exactly

same as it had happened once. Multiple factors like air resistance, friction, small errors by the players will slightly affect the trajectory the ball. Being a chaotic system, the game will then end up in a completely different trajectory as shown in Figure 7.2 It is therefore near impossible to recreate a game after it has played out.

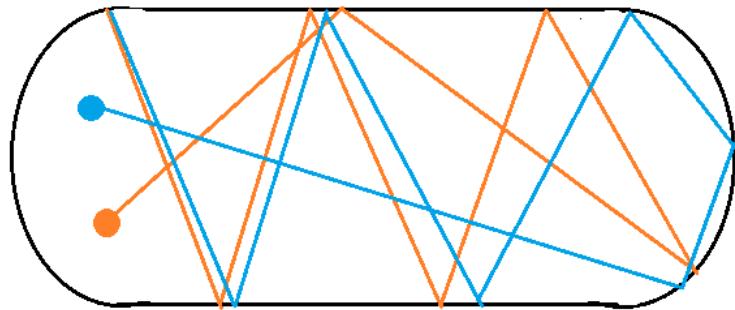


Figure 7.2: Billiards game

Almost all the systems in nature can be considered chaotic. Weather can be a suitable example for this. Chaotic nature of the weather is what it makes it very hard to predict. Similarly, dynamics of clouds, population dynamics, stock market values all are chaotic systems.

7.2 CONDITIONS FOR DETERMINING A DYNAMICAL SYSTEM TO BE CHAOTIC

1. Sensitive dependence on initial conditions: Sensitive dependence on initial conditions is called as butterfly effect. Initially, we pick any point in the phase-space of the dynamical system as the first point and another point approximately close to it is chosen randomly as the second point. As the time carries on and the system evolves, the two points will start to diverge and from being arbitrarily close to each other they end up in totally different trajectories. This is called butterfly effect. The term butterfly comes from the saying that the flapping of wings of a butterfly in one side of the world can lead to a hurricane on the other side of the world.

2. System should be topologically transitive: Consider 2 sets U and V containing points from the map F which shows all the states of a chaotic problem. If the map is topologically transitive, then there exist a point in set U, that over time will end up in a point in set V following its trajectory. Or, it can be said that it is possible for any state in the phase-space of a chaotic problem to reach any other state if given infinite time.

Consider a problem with $f(x) = x^2$. This problem is not chaotic even though it is sensitive to initial conditions. Two close values $x=2$ and $x=2.0001$ will end up with a large difference over time. What makes the problem non-chaotic is that, it is not topologically transitive. For $x=2$, the problem contains 4 and 16 in its solution set. But the point $x=16$ will never come back to the point $x=4$, no matter how much the trajectory proceeds further.

3. System should have dense periodic orbits: Dense periodic orbits are not always considered as a certain criteria for chaotic model. Usually, being topologically transitive will also lead to dense periodic orbits. It means that every point in phase-space of chaotic system is approached by periodic orbits. This is based on Sharkovskii's theorem which shows that any continuous system that exhibits a cycle of period three will also display regular cycles of every other length and chaotic orbits

7.3 WEAK AND STRONG SENSITIVE DEPENDENCE

Weak sensitive dependence refers to the cases where the degree of closeness of the initial points does not matter. Even though the two initial points will diverge no matter what, there is no measure for rate of divergence. Systems with weak sensitive dependence usually end up as non-chaotic systems.

Strong sensitive dependence on the other hand has a measure of divergence and degree of initial closeness also matters in the rate of divergence. Exponential growth is measured in the difference between the points and largest Lyapunov Exponent is taken to measure the average rate of divergence.

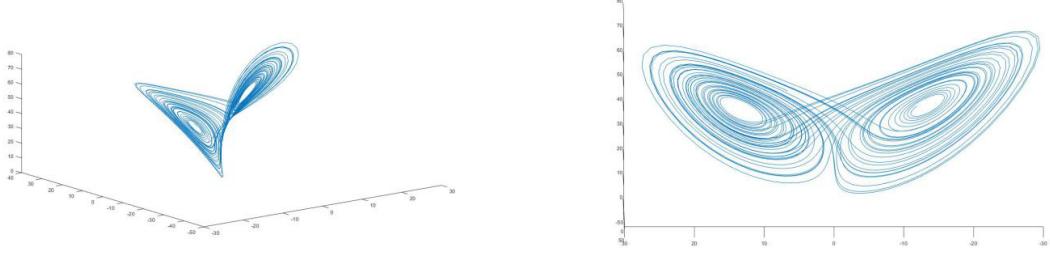


Figure 7.3: Attractor

7.4 ATTRACTORS

Attractor is defined as a point, line, curve, area or set of points in any dimensional space around which a system tends to evolve. For a subset S of the phase-space to be an attractor:

1. Subset A should be invariant, i.e, a point within the subset A will remain in it, for its entire trajectory.
2. There should be a basin of attraction or subset B where all the points in B will enter A somewhere along its trajectory
3. A point cannot exist both in A and B.

Strange attractors refers to the cases where the attractor's local geometric structure is fractal. A fractal is when there is an infinite complex of surfaces appear to merge together. It can be said as a set of point with zero volume but infinite surface area. It should be noted that there exist strange attractors which are non-chaotic in nature. They can be sensitive to initial conditions, but will not exponential growth in rate of separation of two initial values. They have a negative Lyapunov Exponent even though they have dependence on initial conditions.
eg:- Lorenz attractor as shown in Figure 7.3

Lorenz attractor is an attractor in that system described by Lorenz equations. Lorenz equations are a set of 3 partial differential equations used to describe a 2D fluid flow uniformly warmed from below and cooled from above.

$$\frac{dx}{dt} = \sigma(y - x) \quad (7.1)$$

$$\frac{dy}{dt} = x(\rho - z) - y \quad (7.2)$$

$$\frac{dz}{dt} = xy - \beta z \quad (7.3)$$

where x is proportional to the rate of convection, y is horizontal temperature variation, and z is the vertical temperature variation.

In Lorenz system the entire phase-space of the system can be figured out without the initial condition. This is because even though the state at any particular point needs initial conditions and the entire trajectory remains almost same because of the presence of attractors. But the trajectory will switch between attractors at any point of time when they come within area where 2 basin of attraction of attractors coincide. A slight change in the initial conditions can cause a big change when switching between attractors doesn't coincide.

7.5 LYAPUNOV TIME

Lyapunov time shows the maximum time until which a chaotic system can be predicted within given tolerance. It is defined as the inverse off the system's largest Lyapunov Exponent. This depends upon the type of system and the required accuracy of the prediction. Lyapunov time is especially important for chaotic systems as it shows until where predictions should be considered reliable and after which the predictions should be discarded as they have a chance of large error.

Lyapunov time of the solar system is considered as 5 million years while that of a chemical chaotic oscillation is considered 5.4 minutes.

7.6 PHASE SPACE RECONSTRUCTION

One of the first steps in dynamical system analysis is phase space reconstruction. A phase space is a multi-dimensional space in which all possible states of the systems are represented.

For the measurement of parameters of the system including the dimension and Lyapunov exponent, state space reconstruction is used, which is built from a scalar time series obtained from the system.

It has been shown by Taken that a state space can be reconstructed from a scalar time series when there is no noise from a scalar time series by the methods of delays as the noise levels are constant for each delay component. Delay coordinates reconstruct the attractor dynamics and multiple state-space vectors are obtained. The reconstructed state of the system at each discrete time n is

$$\bar{X}_n = [X_n, X_{n+1}, X_{n+2}, \dots, X_{n+(d-1)}] \quad (7.4)$$

Here τ is the reconstruction delay and d the embedding dimension.

Taken's theorem is dependent on the choice of the time delay. This is a drawback when considering real world data which contains noise. There are no hard rules regarding the choice of time delay. However, Abarbanel.H.D.I offered the following guidelines. First, it must be a multiple of sampling time, for data is available only at these intervals. Second, If time delay is too short it leads to redundancy, where the coordinates in the phase space are not independent enough. In other words, not enough time will have evolved for the system, for its phase space to produce new information about that phase space. Third, if the time delay chosen is too large it leads to irrelevance, where the difference between dynamics at different times are too large and this will lead to complexities in the presence of chaos and noise.

The embedding dimension is the minimum number of dimensions required to represent all the information of the system. Taken's theorem gives a sufficient (but not necessary) condition $d \geq D$, where D is the fractal dimensions of the underlying attractor.

In using real-world data for state-space reconstruction, much of the effort will be focused on finding the appropriate time delay τ and embedding dimension d . While there has been

no clear method to find time lag, Kantz.H and Schreiber.T, developed the first zero of the autocorrelation function heuristic. Another heuristic used is the first minimum of the auto-mutual information curve, suggested by Fraser.A.M and Swinney.H.L. Both methods are based on minimizing redundancy.

7.7 ESTIMATION OF LAG USING AUTOCORRELATION FUNCTION

The Autocorrelation function (ACF) shows the value of the autocorrelation coefficient for different time lags

$$C(\tau) = \frac{E[(X_n - \mu)(X_{n+\tau} - \mu)]}{\sigma^2} \quad (7.5)$$

The autocorrelation function is expected to provide a reasonable measure of the transition from redundancy to irrelevance as a function of delay. A common choice for delay is the time at which the autocorrelation function has its first zero, which makes the coordinates linearly uncorrelated.

Although methods that use ACF ensure faster results, they do not account for non-linear dynamics and measure only linear dependence. Since the relationship between the spatial distribution of a reconstructed attractor and the temporal autocorrelation of a single time series is an ill-defined one, the method tends to be inconsistent. To counter these shortcomings, a spatial measure based on mutual information is used.

7.8 ESTIMATION OF LAG USING MUTUAL INFORMATION FUNCTION

Suggested by Fraser and Swinney, this method gives a better measure of the shift from redundancy to irrelevance with non-linear systems as it uses mutual information function $I(\tau)$ which supplies a measure of general dependence instead of linear dependence given by the autocorrelation function. Mutual information measures the accuracy of predicting $I(t + \tau)$ from an observation in hand $x(t)$. Smaller mutual information suggests that successive delay

coordinates are relatively independent. According to Fraser and Swinney greatest independence, or the lowest $I(\tau)$, can be associated with the least redundancy and, therefore is the best for attractor reconstruction. Hence the proper time delay can be selected as the lag that produces a local minimum of $I(\tau)$.

The amount of information that is shared by two data sets is called the mutual information of two random variables. It is used to measure the mutual dependence of two random variables. According to Shannon's information theory, if $s_1, s_2, s_3, \dots, s_n$ are the measurements of a system S and P_s are their associated probabilities, P_s maps measurements to probabilities. The amount of information gained from a measurement that specifies s is called the entropy of the system and is given by,

$$H(s) = - \sum_i P_s(s_i) \log P_s(s_i) \quad (7.6)$$

For a coupled system S and Q given s has been measured and found to be s_i , the uncertainty in the measurement of q can be found as

$$H(Q|s_i) = - \sum_j P_{q|s}(q_j|s_i) \log(P_{q|s}(q_j|s_i)) \quad (7.7)$$

where $P_{q|s}(q_j|s_i)$ is the probability that a measurement of q will yield q_j , given that the measured value of s is s_i . But the probability associated with the combined system S and Q , that a measurement of q will yield q_j , and the measurement of s yield s_i , $P_{sq}(s_i, q_j)$ is given by,

$$P_{sq}(s_i, q_j) = P_{q|s}P_s(s_i) \quad (7.8)$$

$$H(Q|s_i) = - \sum_j P_{sq}(s_i, q_j) \log(P_{sq}(s_i, q_j)) \quad (7.9)$$

For a time series $x(t)$, it can be determined how dependent the values of $x(t + \tau)$ are on the values of $x(t)$ by making the assignment $[s, q] = [x(t), x(t + \tau)]$. If x has been measured at time t , then the average uncertainty in a measurement of x at time $t + \tau$ is found by averaging

$H(Q|S_i$ over s_i . This gives

$$H(Q|S) = \sum_i P_s(s_i) H(Q|s_i) \quad (7.10)$$

$$H(Q|S) = \sum_{i,j} P_{sq}(s_i, q_j) \log[P_{sq}(s_i, q_j)/P_s(s_i)] \quad (7.11)$$

$$H(Q|S) = \sum_{i,j} P_{sq}(s_i, q_j) \log[P_{sq}(s_i, q_j)] - \log[P_s(s_i)] \quad (7.12)$$

$$H(Q|S) = \sum_{i,j} P_{sq}(s_i, q_j) \log[P_{sq}(s_i, q_j)] + \sum_{i,j} P_{sq}(s_i, q_j) \log[P_s(s_i)] \quad (7.13)$$

Using the mutual information method overcomes the disadvantage of the autocorrelation based method, but this comes with more computational effort. Martinerie et al showed that mutual information is also inconsistent in identifying the optimal value of τ .

7.9 DETERMINING EMBEDDING DIMENSIONS FOR PHASE SPACE RECONSTRUCTION

The embedding dimension is the minimum number of the dimensions required to represent all the information within the system. There are different methods used to estimate the embedding dimensions for phase space reconstruction including false nearest neighbours and Cao's method.

The embedding dimension is the minimum number of the dimensions required to represent all the information within the system. There are different methods used to estimate the embedding dimensions for phase space reconstruction including false nearest neighbours and Cao's method.

But, False Nearest Neighbours has its own demerits. The results are affected by noise present in data. Another problem is with the selection of threshold limits. Using different threshold limits will result in different embedding dimensions. These problem can be avoided by using Cao's method.

Consider a time series $x_1, x_2, x_3, \dots, X_N$ with reconstructed time series as,

$$y_i(d) = x_i, x_{i+\tau}, x_{i+2\tau}, \dots, x_{i+(d-1)\tau} \quad (7.14)$$

where $i = 1, 2, \dots, N - (d-1)\tau$

Similar to False Nearest Neighbour method,

$$a(i, d) = \frac{\|y_i(d+1) - y_{n(i,d)}(d+1)\|}{\|y_i(d) - y_{n(i,d)}(d)\|} \quad (7.15)$$

where $\|$ is measurement of Euclidean distance, $y_i(d+1)$ is the reconstructed vector with embedding dimension $d+1$, and $y_{n(i,d)}$ is its nearest neighbour.

While using False Nearest Neighbour Method, determining the best threshold value to use is quite hard as it is independent of the dimension or point in the trajectory. To avoid this, Cao's method uses mean of all $a(i, d)$ values,

$$E(d) = \frac{1}{N-d\tau} \sum_{i=1}^{N-d\tau} a(i, d) \quad (7.16)$$

Also, the variation of dimension from d to $d+1$ is given by,

$$E_1(d) = \frac{E(d+1)}{E(d)} \quad (7.17)$$

Also,

$$E^*(d) = \frac{1}{N-d\tau} \sum_{i=1}^{N-d\tau} x_{i+d\tau} + x_{n(i,d)+d\tau} \quad (7.18)$$

This means $E_2(d)$ can be defined as,

$$E_2(d) = \frac{E^*(d+1)}{E^*(d)} \quad (7.19)$$

$E_2(d)$ is used because it is practically hard to check whether the given data is random or deterministic by using $E_1(d)$ alone. This is because it is impossible to measure whether $E_1(d)$ has stopped changing or it is slowly increasing at some value of d . But $E_2(d)$ on the other hand remains constant at 1 as the future values are independent of past values. If the values

are deterministic in nature then $E_2(d)$ will not be constant.

$E_1(d)$ will change as dimension(d) value increases, but will stop changing once the dimension(d) exceeds over some value d_0 . Then, the minimum embedding dimension will be equal to $d_0 + 1$.

7.10 APPROXIMATE ENTROPY

Entropy is used to classify the dynamics or the complexity of a system and is hence of interest in the analysis of real world time-series data. Time series with repeating elements are associated with more ordered systems and they have a low value of entropy. For determining the precise entropy, an infinite time-series is required. Since this is not the case with real world data, estimation methods are used such as Approximate Entropy or *ApEn*.

ApEn is the negative logarithm of the conditional probability that a subset of the data, called a template, is repeated in the time series. For calculating *ApEn*, a template of a particular length is chosen. Let it be of m points. Then the templates similar to this template are found and the templates that remain similar for the next, or $(m+1)^{th}$ point are determined. The similarity is often determined using a parameter called tolerance(r) and is selected as a factor of standard deviation. The negative logarithm of the conditional probability is calculated for each possible template and the results averaged.

$$ApEn = \sum -\log \frac{1 + A_i}{1 + B_i} \quad (7.20)$$

where A_i is the number of matches of length $m+1$ with i^{th} template and B_i is the number of matches of length m with i_{th} template.

If the data are ordered, then templates that are similar for m points are often similar for $m+1$ points, CP approaches 1, and the negative logarithm and entropy approach 0.

For ordered data, the templates similar for m points are, only to be similar for $m+1$

points and the conditional probability tends to 1 and hence entropy tends to zero, hence explaining the idea of using entropy to measure order.

7.11 CORRELATION DIMENSION

Correlation Dimension is useful in measuring the dimensions of fractal objects and is the simplest dimension measurement to perform. The dimensions of fractals may be fractional and hence correlation dimension takes non-integer values also. Correlation dimension(d) is defined as

$$\lim_{\delta \rightarrow 0} \frac{\log(C(\delta))}{\log(\delta)} \quad (7.21)$$

where δ is the Euclidean distance between embedded points in the phase space and $C(\delta)$ is the correlation integral function and stands for the number of distances that are equal and smaller than δ .

7.12 LYAPUNOV EXPONENT

Lyapunov exponent of a dynamical system is referred as the measure of rate of separation of two arbitrarily chosen values which infinitesimally close to each other.

Consider two points $x_1(T)$ and $x_2(T)$ which are infinitesimally close to each other in the space of a chaotic system. Let $\Delta x(T)$ be the initial separation between the two points. After a time t , the two points following their trajectory reaches $x_1(T + t)$ and $x_2(T + t)$ and their separation changes to $\Delta x(T + t)$. Since the system is chaotic it the rate of separation increases in an exponential rate.

$$\Delta x(T + t) - \Delta x(T) = e^{\lambda T} \quad (7.22)$$

$$\lambda = \frac{1}{T} \ln(\Delta x(T + t) - \Delta x(T)) \quad (7.23)$$

where λ is the Lyapunov exponent.

Since the rate of separation varies depending on the values chosen, we calculate maximal

Lyapunov Exponent to determine whether the given system is chaotic. If Maximal Lyapunov Exponent is positive, it denotes that the given dynamical system is chaotic. Maximal Lyapunov exponent can be calculated using algorithms like Wolf's Algorithm or Kantz' Algorithm. Otherwise it can be found from the plot of divergence rate of nearby trajectories.

7.13 RESULTS AND DISCUSSION

7.13.1 Estimation of lag using Autocorrelation Function

The *nonlinearTseries* package of R programming environment is used for the estimation. It provides functions for estimating both delay parameter and embedding dimension. Autocorrelation function is estimated for lag values from 0 to 20 and is then plotted against it. The figure 7.4 shows the graph of autocorrelation function against lag parameter obtained using R programming.

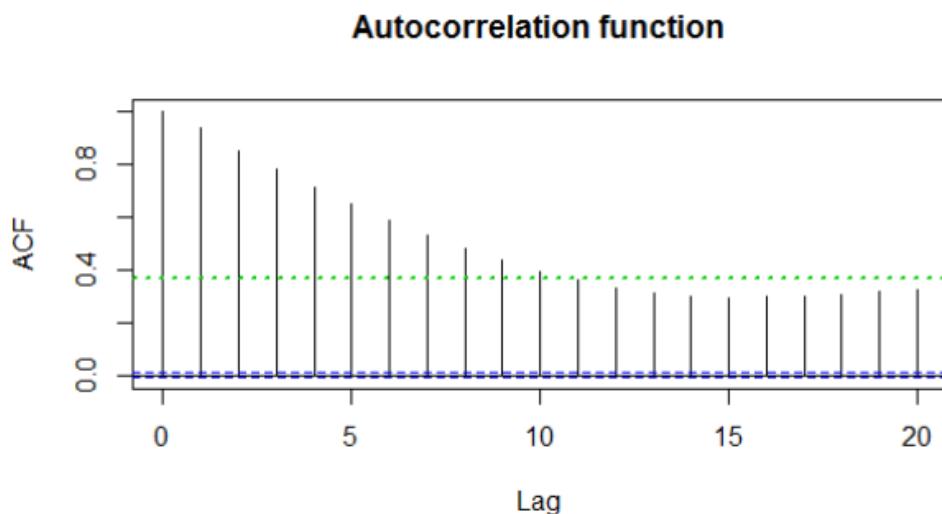


Figure 7.4: Autocorrelation function vs lag

Lag parameter is estimated by either taking lag value corresponding to when autocorrelation becomes zero or when it becomes equal to $1/e$. Here, we take lag parameter corresponding to when autocorrelation function becomes zero. Therefore, from the autocorrelation vs lag parameter graph, estimated lag would be 11.

7.13.2 Estimation of lag using Mutual information function

Similar to estimation using autocorrelation function, estimation using mutual information function also uses *nonlinearTseries* package of R programming. Average Mutual Information is estimated for lag values of 0 to 20 and is then plotted against lag parameter. The Figure 7.5 shows the graph of Average Mutual Information function against lag parameter obtained using R programming.

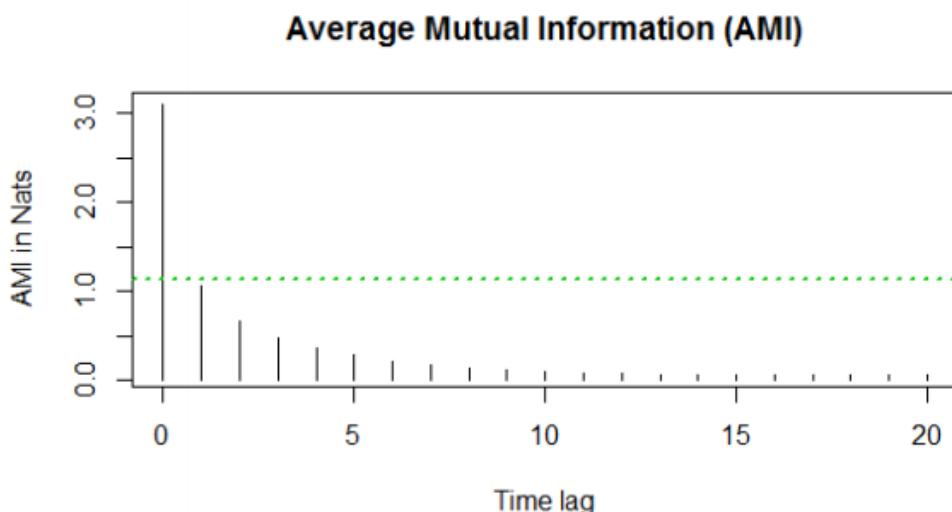


Figure 7.5: Average Mutual Information function vs lag

Lag parameter is estimated to be the value corresponding to when average mutual information becomes equal to 1 . Therefore, from the AMI vs lag parameter graph, estimated lag would be 1.

7.13.3 Estimation of embedding dimension using Cao's Method

In order to estimate the embedding dimension, we use *nonlinearTseries* package of R programming. The *estimateEmbeddingDim* function allows to calculate the embedding dimension of the time series. The maximum limit of embedding dimension is set at 15. For each embedding dimension from 0 to 15, we calculate $E_1(d)$ and $E_2(d)$ values and plot them against dimension(d) as shown in the Figure 7.6 and Figure 7.7

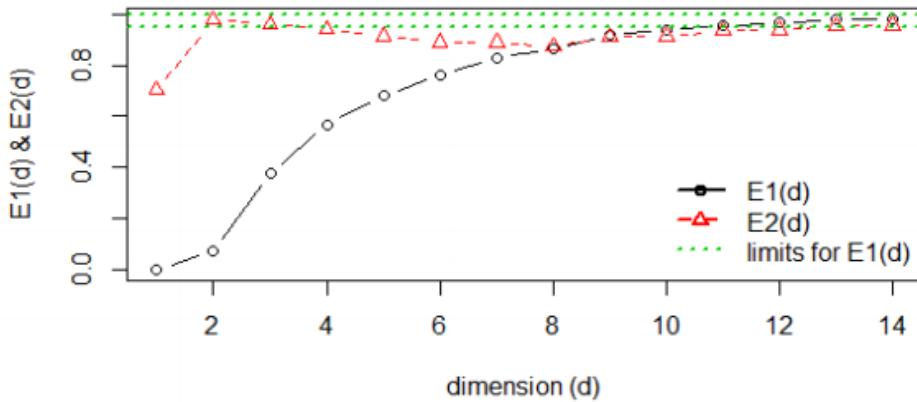


Figure 7.6: Plot of $E_1(d)$ and $E_2(d)$ vs dimension(d) for lag = 11

The minimum embedding dimension for lag=11 is 1 as observed from Figure 7.6

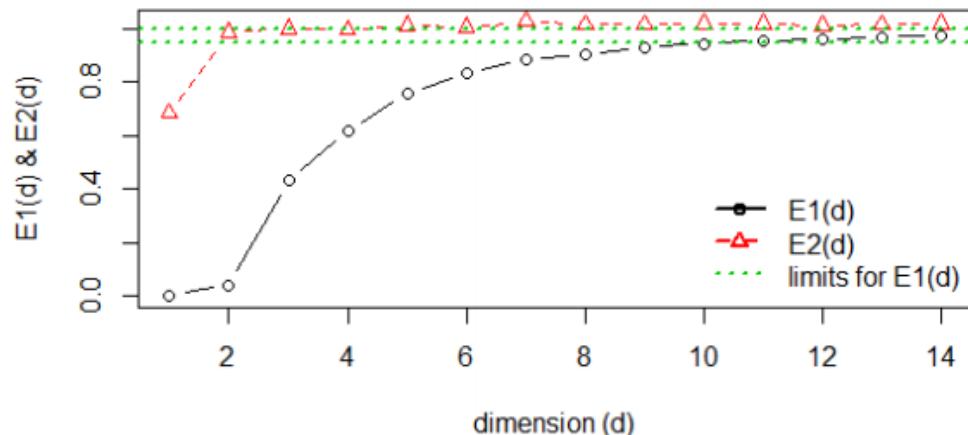


Figure 7.7: Plot of $E_1(d)$ and $E_2(d)$ vs dimension(d) for lag = 1

The minimum embedding dimension for lag=1 is 1 as observed from Figure 7.7

7.13.4 Phase space reconstruction

MATLAB is used for the phase space reconstruction of the time series based on the found time lag and embedding dimension.

For lag = 1, and embedding dimension = 11, the phase space is reconstructed for the given time series. Figure 7.8 shows the reconstructed time series.

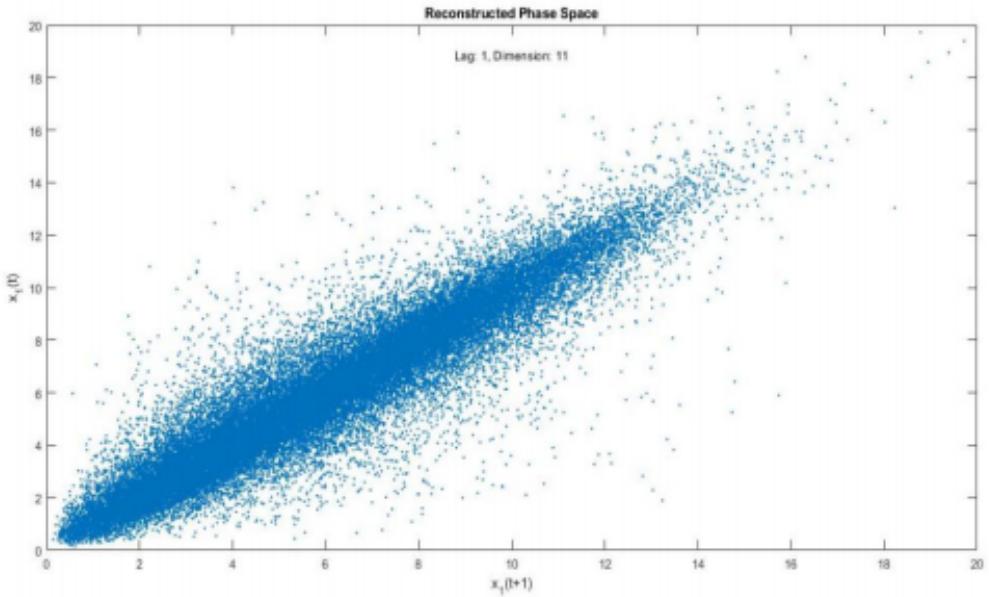


Figure 7.8: Phase space reconstruction for lag = 1 and embedding dimension = 11

For lag = 11, and embedding dimension = 11, the phase space is reconstructed for the given time series. Figure 7.9 shows the reconstructed time series.

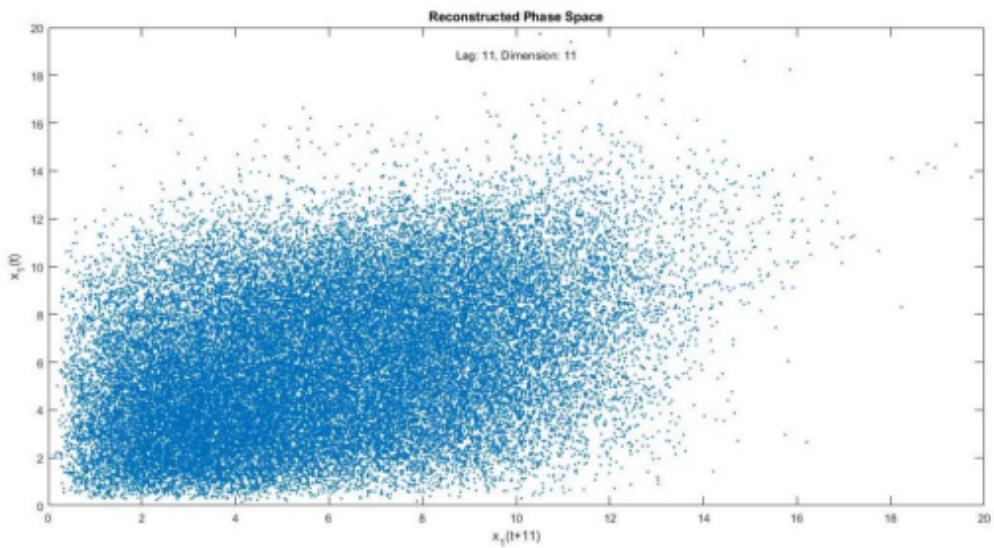


Figure 7.9: Phase space reconstruction for lag = 11 and embedding dimension = 11

7.13.5 Estimation of Approximate Entropy

Using MATLAB, approximate entropy is computed for found lag and embedding dimension. The tolerance is set randomly at 0.60018.

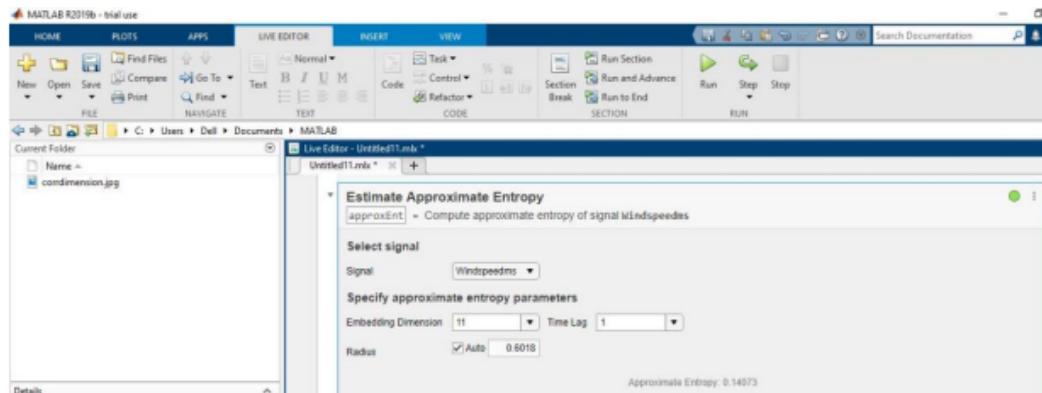


Figure 7.10: Approximate Entropy for lag = 1 and embedding dimension = 11

For lag = 1 and embedding dimension = 11, the approximate entropy was found to be 0.14073 as shown in Figure 7.10.



Figure 7.11: Approximate Entropy for lag = 11 and embedding dimension = 11

For lag = 11 and embedding dimension = 11, the approximate entropy was found to be 0.0105 as shown in Figure 7.11.

The positive value of the approximate entropy in both cases indicates that the given time series is chaotic in nature.

7.13.6 Estimation of Correlation Dimension

Using MATLAB, the correlation dimension is computed with wind speed, lag and embedding dimension, number of points of computation, minimum radius of similarity and maximum radius of similarity as input parameters. The minimum radius of similarity and maximum radius of similarity is obtained from Figure 7.10 and Figure 7.11 (the red dots indicates the two values). The number of points of computation is set at default value of 10.

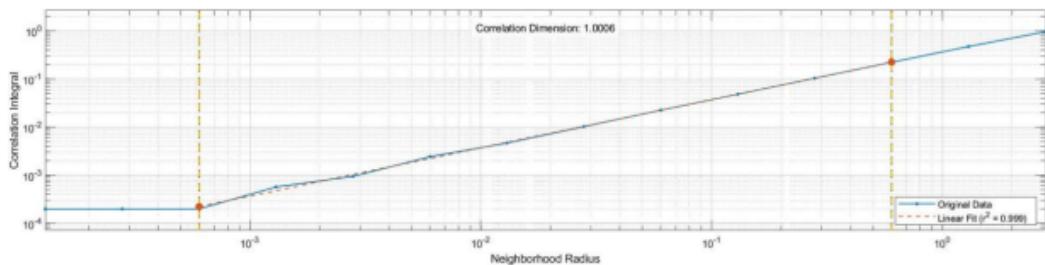


Figure 7.12: Correlation Dimension for lag = 1 and embedding dimension = 11

For lag = 1 and embedding dimension =11, the correlation dimension is found to be 1.0006.

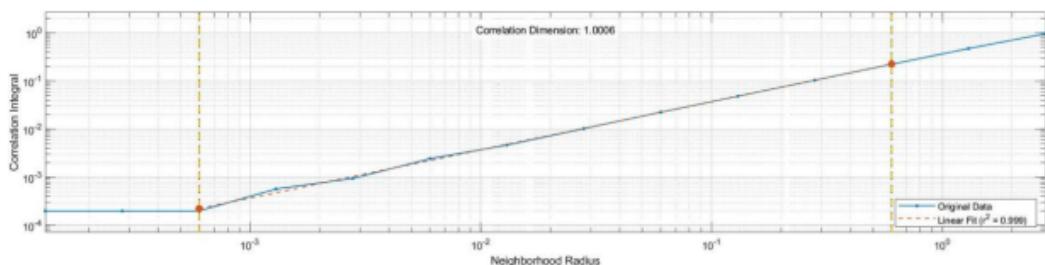


Figure 7.13: Correlation Dimension for lag = 11 and embedding dimension = 11

For lag = 11 and embedding dimension =11, the correlation dimension is found to be 1.0006.

The value of correlation dimension is directly proportional to the level of chaos in the system.

7.13.7 Estimation of Maximal Lyapunov Exponent

Maximum Lyapunov Exponent is estimated using *nonlinearTseries* package of R programming. Function *MaxLyapunov* is used for estimating the maximal Lyapunov exponent of a dynamical system from 1-dimensional time series using Takens' vectors.

For both lag = 1, plot of divergence of the points is plotted as shown in Figure 7.14. The Figure is then fitted in a regression line as shown in Figure 7.15. The same is done for lag = 11 as shown in Figure 7.16 and Figure 7.17.

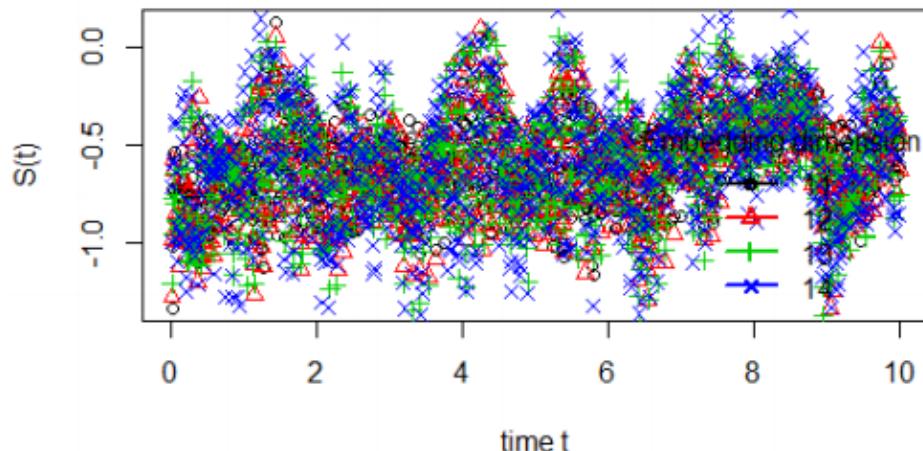


Figure 7.14: Plot of divergence for lag = 1 and correlation dimension = 11

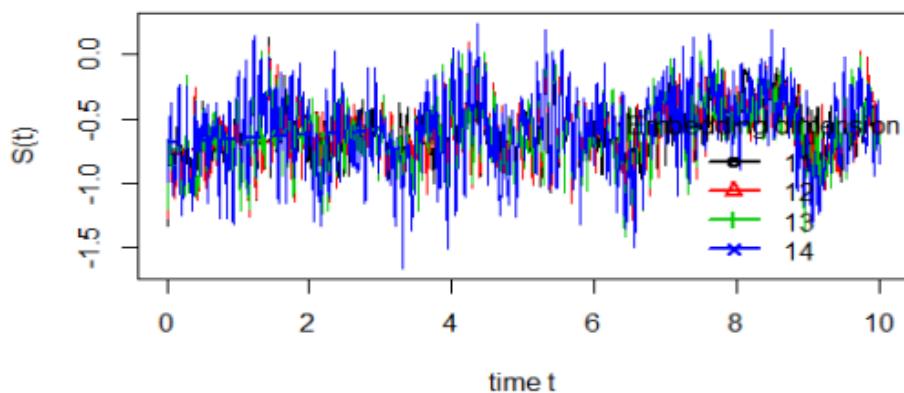


Figure 7.15: Plot of divergence on a regression line for lag = 1 and correlation dimension = 11

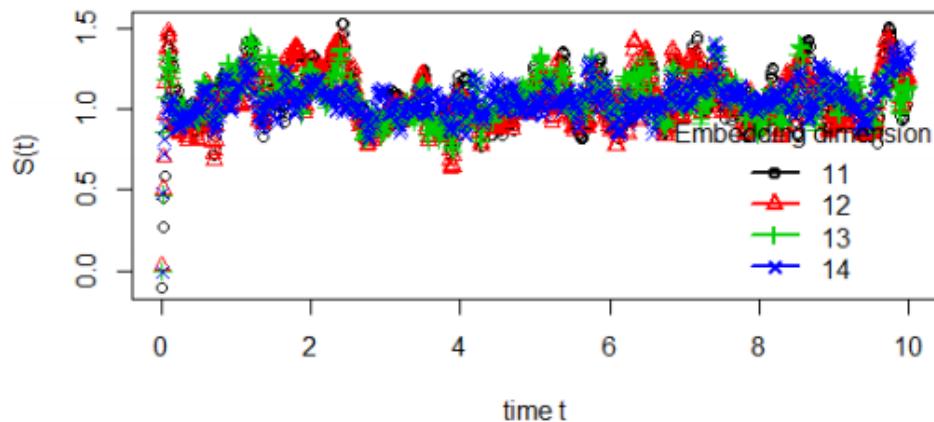


Figure 7.16: Plot of divergence for lag = 11 and correlation dimension = 11

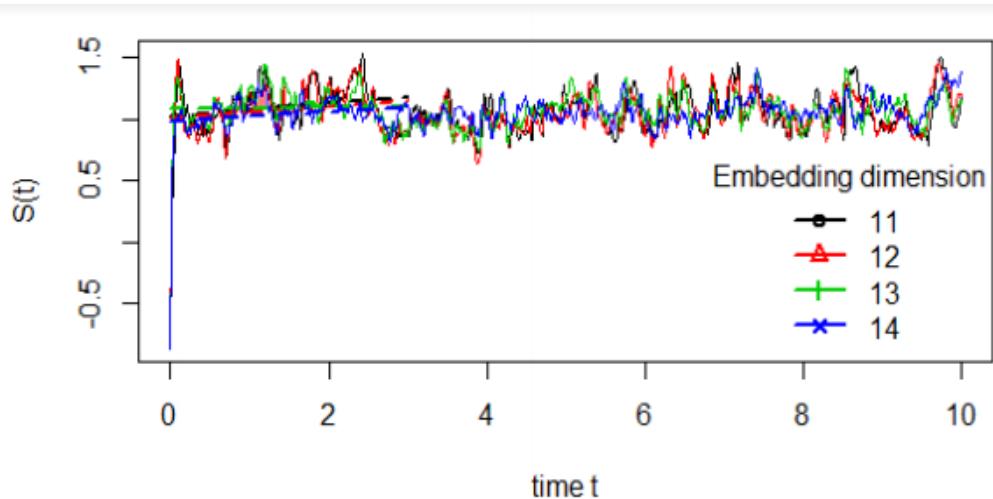


Figure 7.17: Plot of divergence on a regression line for lag = 11 and correlation dimension = 11

The slope of the regression line is considered equal to the Maximal Lyapunov Exponent. From Figure 7.15, the Maximal Lyapunov Exponent is estimated to be 0.05139 for lag = 1. From Figure 7.17, the Maximal Lyapunov Exponent is estimated to be 0.034463 for lag = 11.

The estimated values of the Maximum Lyapunov Exponent for both lag=1 and lag=11 are positive. Thus, it can be inferred that the time series is chaotic in nature.

Chapter 8

Chaotic Wind Speed Prediction

From the results of previous section we can infer that the wind speed data is chaotic in nature. That means its deterministic but relations are complex. Hence variants of Long Short Term Neural Networks Algorithm are employed to determine future speeds, which then serve as an input to stacked models to estimate future power.

8.1 LSTM

LSTM network is a type of neural network architecture that finds its application in tasks related with prediction, language translation, text-to-speech recognition and image captioning. LSTM is an abbreviation for Long Short Term Memory and considered as the successor to Recurrent Neural Networks (RNN). The shortfalls of RNN's when it came to time dependent and sequential tasks due to lack of memory utilization (being unable to learn from long-term dependencies) , and gradients exhibiting a vanishing nature when more layers were included in the network necessitated the adoption of a more efficient network manifested in the form of LSTM's. LSTM's solve these travails by utilizing a cell unit responsible for storing memory to learn from long-term dependencies and by preventing the vanishing gradient problem.

8.1.1 Architecture of LSTM

LSTM's comprise of cell units, which are regarded as the memory units of the network capable of storing information pertaining to past and present data. Information can be added to or removed from the cell structure by the means of gating structures that regulate what

information is to be retained in the cell structure and what information is to be thrown away. LSTM's take input from the output of the previous hidden state of the cell (h_{t-1}) and the current input to the cell (x_t).

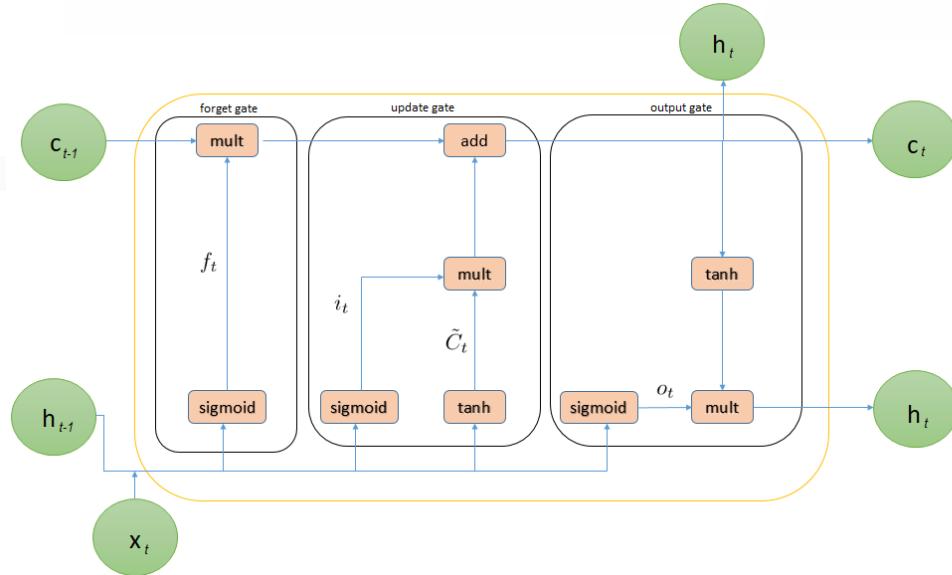


Figure 8.1: LSTMrep

LSTM's comprise of three gates:

- The Forget Gate
- The Input Gate
- The Output Gate

8.1.1.1 Forget Gate

The forget gate controls as to what information is to be retained from the output of the previous cell state and what information is to be discarded from the previous cell state. It accepts inputs h_{t-1} and x_t and passes them through a sigmoid activation function which squishes values in the range [0,1]. Closer the value to zero, greater is the chance of forgetting the information and closer the value to 1, greater is the chance of retention.

The equation for the forget gate is as follows:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \quad (8.1)$$

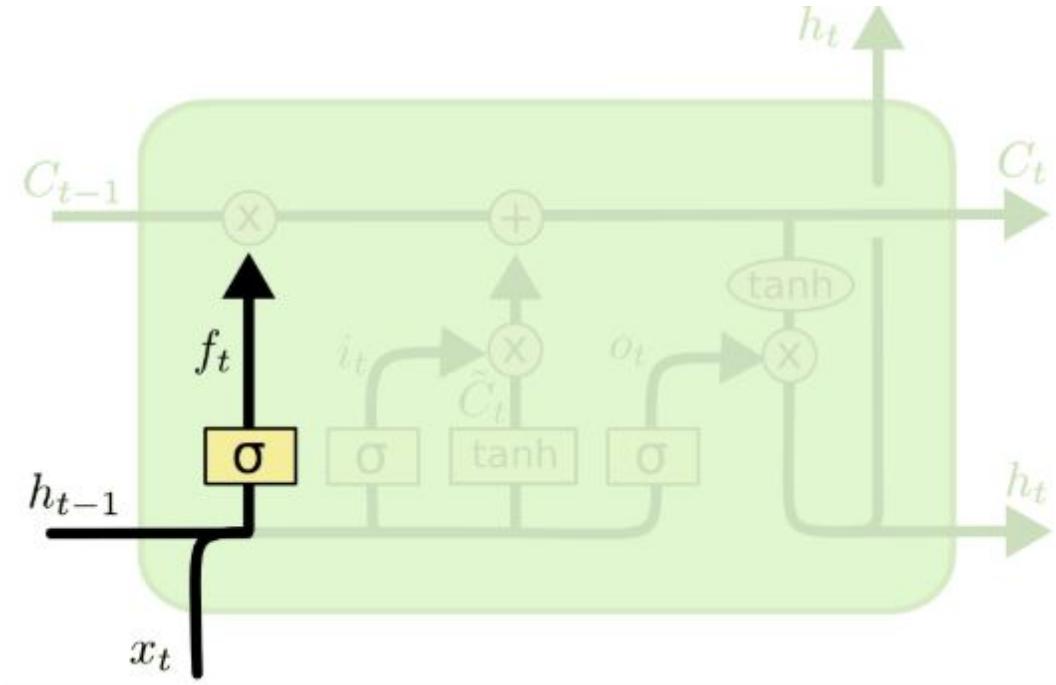


Figure 8.2: LSTM forget gate

where W_f are the weights of the forget gate which are to be trained and b_f refers to the bias term of the forget gate.

8.1.1.2 Input Gate

The input gate is also known as the update gate. For updating the cell state, inputs h_{t-1} and x_t are accepted by the input gate through a sigmoid activation function. This sigmoid activation function decides what all values are to be accepted and what all values are to be rejected by squishing inputs in the range $[0,1]$. Also, the same set of inputs are also passed through an input modulation gate comprising of a tanh activation function that squishes values within the range $[-1,1]$ helping to regulate the network. The equations for these two gates are given as

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \quad (8.2)$$

$$\tilde{C}_t = \tanh(\sigma(W_c \cdot [h_{t-1}, x_t] + b_c)) \quad (8.3)$$

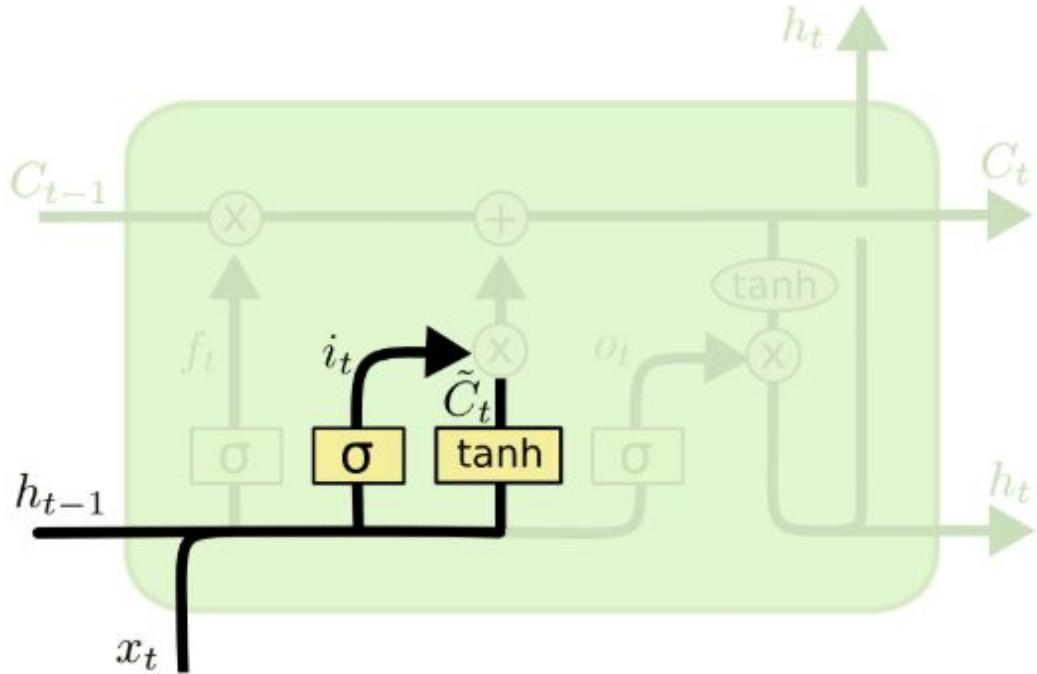


Figure 8.3: LSTM input gate

where W_i and b_i are the weights and bias term of the input gate respectively and W_c and b_c are the weights and the bias term of the input modulation gate respectively.

8.1.1.3 Cell State

The outputs from the above respective gates are multiplied. The sigmoid output decides what information is important and to be retained from the \tanh output.

The cell state updation is depicted by the below diagram.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \quad (8.4)$$

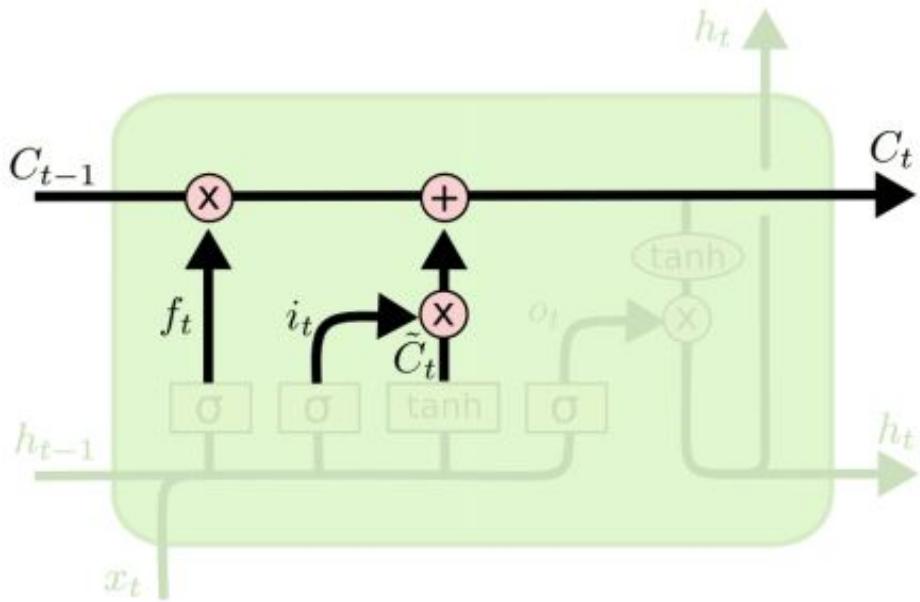


Figure 8.4: LSTM cell state

8.1.1.4 Output Gate

The output gate decides what information in the current cell state is to be passed on to the next cell state and what is to be retained in the current cell state. Here also, the output gate accepts inputs similar to the above mentioned gates and sends it through the sigmoid activation function that squishes values in the range [0,1]. Also, the output of the newly computed cell state is passed on to a tanh activation function to regulate it. Now, the sigmoid output from the output gate is multiplied with the *tanh* output of the cell state thereby to determine what information is to be send to the next cell and what not. This decision is carried out by the sigmoid output of the output gate[0 for irrelevant and 1 for relevant] and this combined output known as the hidden state of the current cell is passed on as input to the next cell for the next time step.

$$o_t = \sigma(W_0[h_{t-1}, x_t] + b_0) \quad (8.5)$$

$$h_t = o_t * \tanh(C_t) \quad (8.6)$$

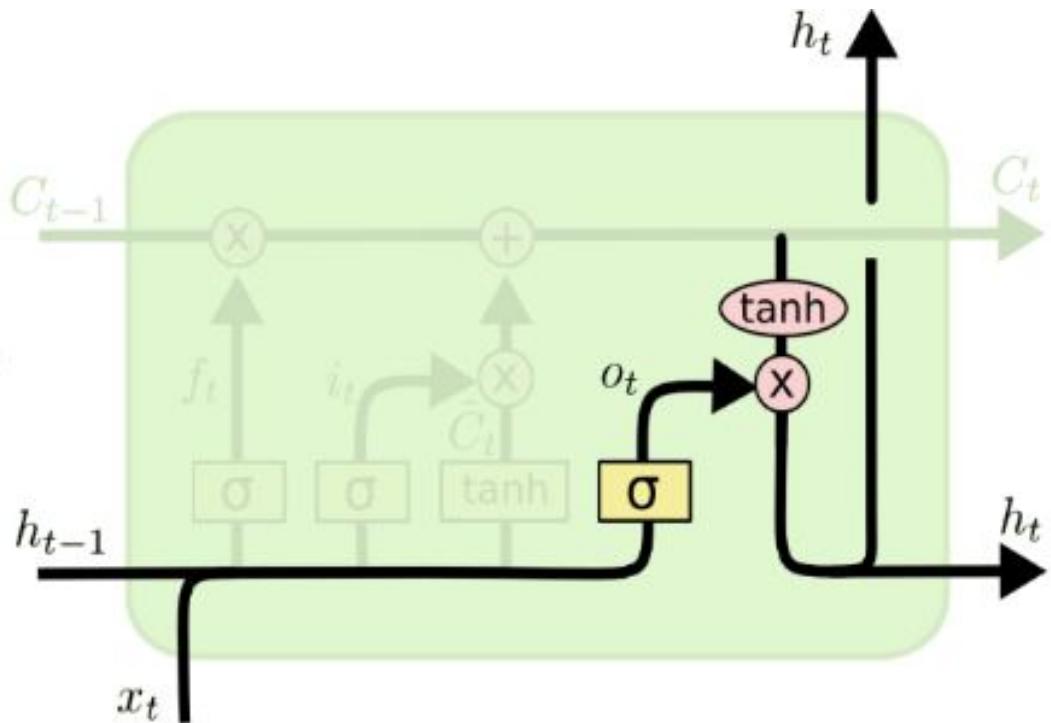


Figure 8.5: LSTM output gate

where W_0 and b_0 are the weights and bias term of the output gate respectively.

Therefore in short, the forget gate decides what information is to be retained from the previous time steps, the input gate decides on what information is to be accepted into the current cell state and the output gate decides on what information goes to the next cell.

8.1.2 Loss Function

The end goal of any machine-learning algorithm is to reduce the discrepancy between the actual (expected) values and the predicted values. In other words, the error between the above terms is to be minimized. The competency of a machine-learning algorithm lies in minimizing the error magnitude to the maximum after a prediction task. The same can be attributed with neural networks as these networks tend to reduce errors with the right choice of number of neurons in a layer, the number of layers involved, the activation functions used, and the optimizer used for minimizing the loss function. The loss function, which is of concern in this project work, is the Mean Squared Error(MSE) which is the sum of the squares

of the differences between the actual(expected) values and the predicted values across the number of training examples divided by the number of training examples.

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad (8.7)$$

Here n denotes the number of training examples, y_i denotes the actual value for the i^{th} observation and \hat{y}_i denotes the predicted value for the i^{th} observation.

8.1.3 Optimizer

The reduction in the magnitude of the error term in a machine learning algorithm or a neural network depends on the performance of the underlying optimizer which optimizes the parameters involved thereby reducing the loss or error.

The commonly used optimizer for LSTM's is the Adam optimizer. Adam is the acronym for adaptive learning rate modifier. The effectiveness of the Adam optimizer comes into play when the objective function involved is non-convex and stochastic in nature. Adam utilizes the power of adaptive learning rates to optimize the values of the parameters involved in the objective function; it does so by utilizing the first and second moments of the gradient of the function with respect to the underlying parameters at each time step.

Adam uses the concept of exponential moving averages for the estimation of the first and the second moments of the gradient, which are the mean and the centered variance respectively. Given below is the algorithm depicting the Adam optimizer.

Required: $\alpha \leftarrow$ Stepsize

Required: $\mu_1, \mu_2 \in [0, 1]$: Exponential decay rates for the moment estimate

Required: $f(\Theta)$: Stochastic objective function with parameters Θ

Required: Θ_0 : Initial parameter vector

$m_0 \leftarrow 0$ (Initializing 1st moment vector)

$v_0 \leftarrow 0$ (Initializing 2nd moment vector)

```

 $t \leftarrow 0$  (Initializing timestep)
while  $\Theta_t$  not converged
do
 $t \leftarrow t + 1$ 
 $g_t \leftarrow \text{grad}(f_t(\Theta_{t-1}))$  (Get gradients w.r.t. stochastic objective at timestep  $t$ )
 $m_t \leftarrow \mu_1.m_{t-1} + (1 - \mu_1).g_t$  (Update the biased first moment estimate)
 $v_t \leftarrow \mu_2.v_{t-1} + (1 - \mu_2).g_t^2$  (Update the biased second raw moment estimate)
 $mb_t \leftarrow m_t/(1 - \mu_1^t)$  (Compute the bias-corrected first moment estimate)
 $vb_t \leftarrow v_t/(1 - \mu_2^t)$  (Compute the bias-corrected second raw moment estimate)
 $\Theta_t \leftarrow \Theta_{t-1} - \alpha[mb_t/(\sqrt{vb_t} + \epsilon)]$  (Update parameters)
end while
return  $\Theta_t$  (Resulting parameters)

```

The values of μ_1 and μ_2 are specified beforehand and are 0.9 and 0.999.

The stepsize α is 0.001 and the required non-convex stochastic objective function is also to be specified beforehand.

The advantages of Adam optimizer are:

1. It outperforms most of the other optimizer algorithms in terms of time taken to converge to global optima.
2. The step size which itself is a hyper-parameter, is invariant to the magnitude of the gradient computed at each time step.

8.2 SEQUENCE TO SEQUENCE MODELS

It is basically a model that takes a sequence of features as input and outputs a target sequence as a continuation to the provided input. It consists of two essential components namely encoder and decoder. Encoder takes input and processes it to a context vector which is then passed to the decoder so as the decoder predicts targets.

Let $X(x_1, x_2, \dots, x_n)$ be the input wind speed values and $Y(y_1, y_2, \dots, y_n)$ be the predicted wind speed values. Both the input and target output are of variable length. The process of encoding inputs to the latent vector representation is done one at a time and later outputs are later decoded in the similar manner.

Let x_t represent the input wind speed at a time step t . Long Short Term Neural Network formulates a hidden or control state h_t and then a memory cell state c_t which is an altogether encoding of everything the cell has until time t :

$$f_t = \sigma(w_{xf}x_t + w_{hf}h_{t-1} + b_f) \quad (8.8)$$

$$i_t = \sigma(w_{xi}x_t + w_{hi}h_{t-1} + b_i) \quad (8.9)$$

$$o_t = \sigma(w_{xo}x_t + w_{ho}h_{t-1} + b_o) \quad (8.10)$$

$$g_t = \phi(w_{xg}x_t + w_{hg}h_{t-1} + b_g) \quad (8.11)$$

$$c_t = f_t \Theta c_{t-1} + i_t \Theta g_t \quad (8.12)$$

$$h_t = o_t \Theta \phi(c_t) \quad (8.13)$$

where σ denotes sigmoid function, Θ denotes element wise product with gate value operator, ϕ depicts hyperbolic tangent non linearity, W_{ij} is the weighted matrix and b_j is the trained bias parameter.

During encoding period the Long Short Term Neural Network computes a sequence of hidden states h_1, h_2, \dots, h_n which further defines a distribution over output during decoding.

$$P(y_1, y_2, \dots, y_m | x_1, x_2, \dots, x_n) = \prod_{t=1}^m P(y_t | h_{n+t-1}, y_{t-1}) \quad (8.14)$$

where the distribution of $P(y_t, h_{n+\tau})$ is given by *softmax* function.

Encoding of input is performed using Long Short Term Neural Networks and then another Long Short Term Neural Network is used to map to the output sequences. Hence we use a single Long Short Term Memory for both the encoding and decoding processes. This aids in the process of parameter sharing.

While training in the decoding stage the log likelihood of the predicted output is maximized. For a model with parameter θ it is formulated as:-

$$\theta^* = \operatorname{argmax}_{\theta} \sum_{t=1}^m \log P(y_t | h_{n+\tau-1}, y_{t-1}, \theta) \quad (8.15)$$

Which is then optimized for the entire training set. The loss is computed only in decoding phase which is then propagated back in time. Appropriate hidden state values are also computed. From the final Long Short Term Neural Network layer we will obtain a result which shows high probability when computed using *softmax* function.

8.2.1 Activation Functions

8.2.1.1 Sigmoid Function

It is an activation function which features as 's' shaped curve when plotted.

$$y = \frac{1}{1 + e^{-x}} \quad (8.16)$$

It exhibits a non linear behavior. x ranges from -2 to 2. Y values are steep this means small changes in x will bring in a greater value for y.

8.2.1.2 Tangent Hyperbolic Function

It is an activation function that works better than sigmoid function. Its actually a mathematical shifted version of sigmoid function.

$$y = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1 \quad (8.17)$$

Its value ranges from -1 to 1 and has a non linear behavior.

8.2.1.3 Softmax Function

It has a non linear behavior. Its used when we handle with multiple classes. The softmax function provides outputs in the range (0,1) and would also divide the summation of outputs.

$$P(y = j|\theta^i) = \frac{e^{\theta_j^i}}{\sum_{j=0}^k e^{\theta_j^i}} \quad (8.18)$$

Here θ is the encoded matrix which is represented by h above.

8.2.2 Model Architecture

8.2.2.1 With Unidirectional Long Short Term Memory neural networks as encoder

Model with basic encoder and decoders are made then progressively a bi-layered model is being created as shown in Figure 8.6.

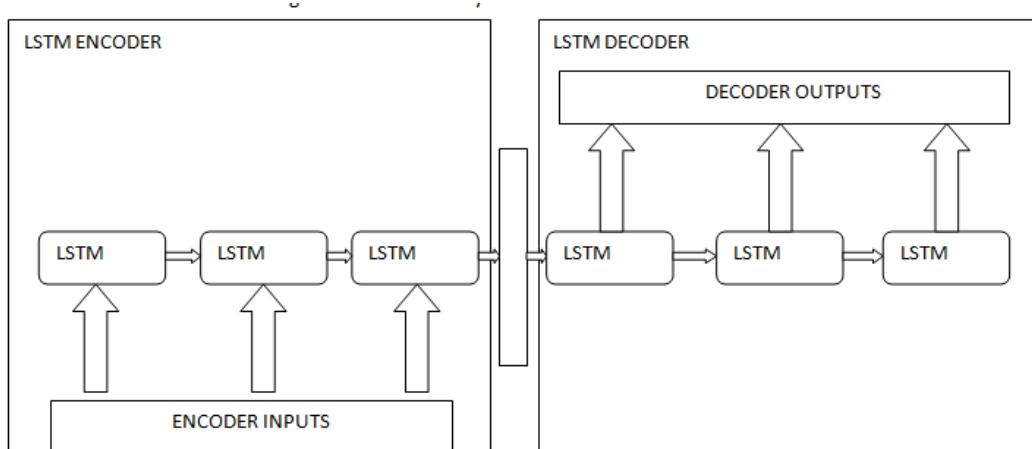


Figure 8.6: Model Architecture with Unidirectional LSTM Neural Networks as Encoder

8.2.2.2 With Bidirectional Long Short Term Neural Networks as encoders

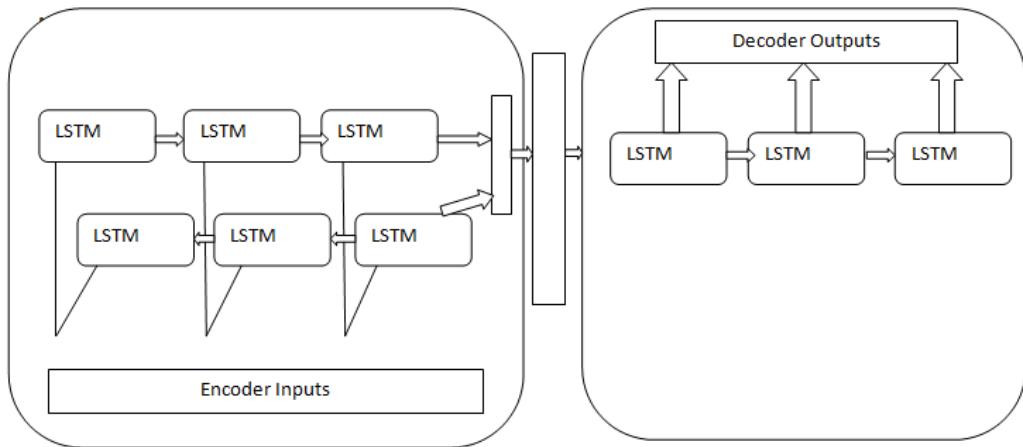


Figure 8.7: Model Architecture with Bidirectional LSTM Neural Networks as Encoder

Bidirectional Long Short Term Memory Neural Network has two entities one forward and another backward. The forward one calculates the hidden and the cell state identical to a unidirectional Long Short Term Memory Neural Network whereas backward component computes them by taking inputs in a reverse sequential order.

The idea behind this backward movement is that we are developing a way in which network is seeing future data and learns the parameters namely weight and bias. The hidden states of forward and backward are not identical so they have to be concatenated with each other

8.2.2.3 With Stacked Unidirectional Long Short Term Memory Neural Network as encoder

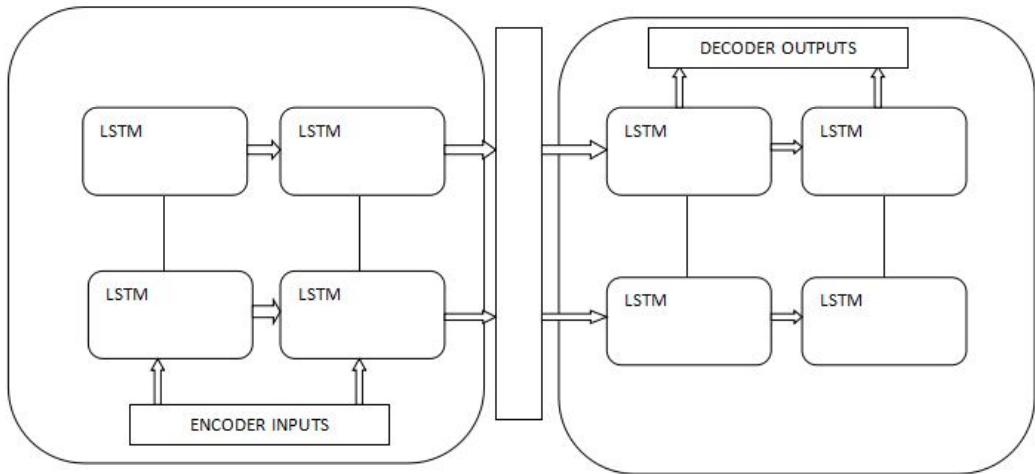


Figure 8.8: Model Architecture with Stacked Unidirectional LSTM Neural Networks as Encoder

When Long Short Term Memory Neural Network Layers are stacked together the output from one layer is given as input to the other layer. By increasing layers complex patterns could be learned by networks and more accurately predicted as shown in Figure 8.8.

8.2.2.4 With Stacked Bidirectional Long Short Term Memory Neural Network as encoder

Bidirectional Long Short Term Memory Neural Networks can be stacked with numerous layers as like unidirectional ones. Here we modeled a two layer stacking in which the outputs of the forward and backward components of the first layer is transferred to the forward and backward components of the second layer.

8.3 RESULTS AND DISCUSSION

8.3.1 LSTM Models

8.3.1.1 LSTM Model for 12 Hours

The plot between predicted and actual wind speed values is shown in Figure 8.9.

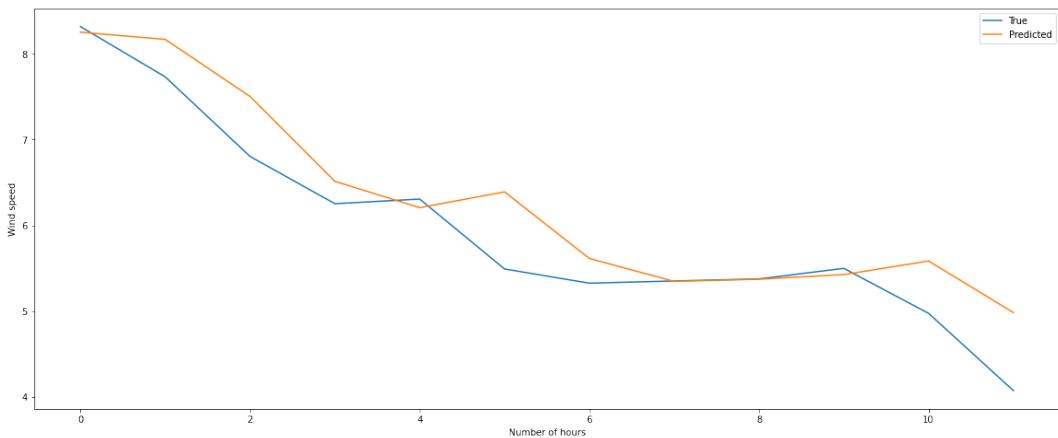


Figure 8.9: Plot between predicted and actual Wind speeds for 12 Hours

Error analysis was performed using various performance metrics and its values are as follows:

Mean Absolute Percent Error: 6.697911923764864

Expected variance score 0.8968397532917245

Max error: 0.9096996570453051

Mean absolute error: 0.362705575151524

Mean squared error: 0.2383957316866172

Root mean squared error: 0.48825785368657126

Mean squared log error: 0.005573991309542872

Median absolute error: 0.2744840249819691

R2 score: 0.8177940239839254

R2 score variance weighted: 0.8177940239839254

Mean poisson deviance: 0.041628458189248985

Mean gamma deviance: 0.007498165616548406

8.3.1.2 LSTM Model for 24 Hours

The plot between predicted and actual wind speed values is shown in Figure 8.10.

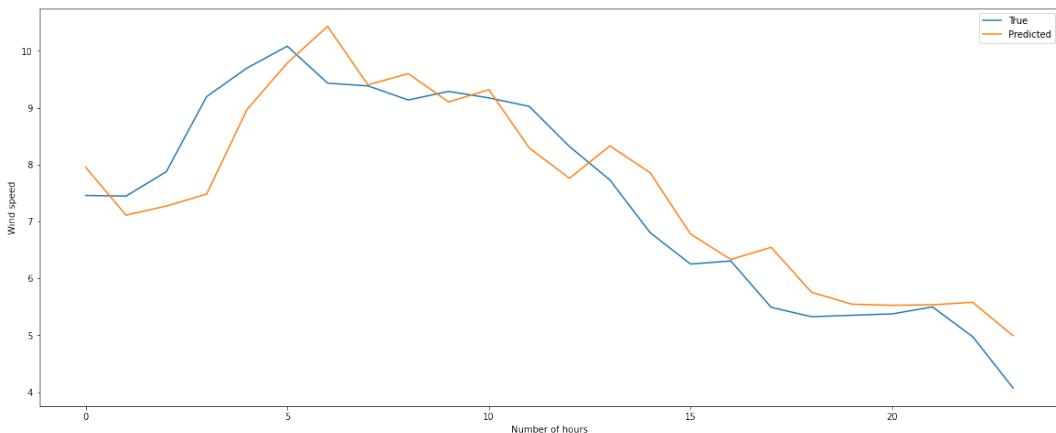


Figure 8.10: Plot between predicted and actual Wind speeds for 24 Hours

Error analysis was performed using various performance metrics and its values are as follows:

Mean Absolute Percent Error: 7.642882292125609

Expected variance score 0.8643336712035394 Max error: 1.7152421489283443

Mean absolute error: 0.5364323347464673

Mean squared error: 0.44666606228464995

Root mean squared error: 0.6683308030344329

Mean squared log error: 0.006704865747312334

Median absolute error: 0.5111627323497099

R2 score: 0.860768048117057

R2 score variance weighted: 0.8607680481170569

Mean Poisson deviance: 0.06109659742662991

Mean gamma deviance: 0.008790761496269006

8.3.1.3 LSTM Model for 2 Days

The plot between predicted and actual wind speed values is shown in Figure 8.11.

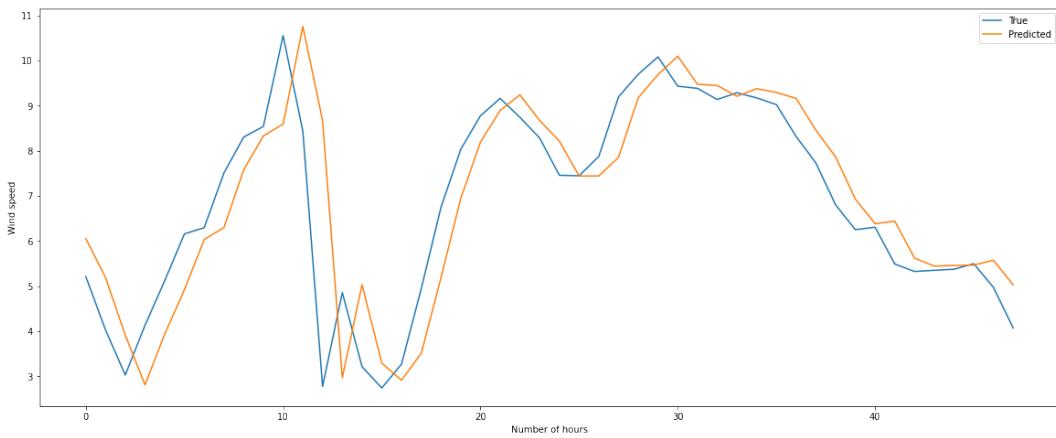


Figure 8.11: Plot between predicted and actual Wind speeds for 2 Days

Error analysis was performed using various performance metrics and its values are as follows:

Mean Absolute Percent Error: 17.03775723071719

Expected variance score 0.6662891640166949

Max error: 5.895131195034832

Mean absolute error: 0.8565962037872525

Mean squared error: 1.5869440702885569

Root mean squared error: 1.2597396835412293

Mean squared log error: 0.03884549331637394

Median absolute error: 0.6714532929752033

R2 score: 0.6640482917705166

R2 score variance weighted: 0.6640482917705166

Mean Poisson deviance: 0.26715613827021106

Mean gamma deviance: 0.050569364983956894

8.3.1.4 LSTM Model for 1 Week

The plot between predicted and actual wind speed values is shown in Figure 8.12.

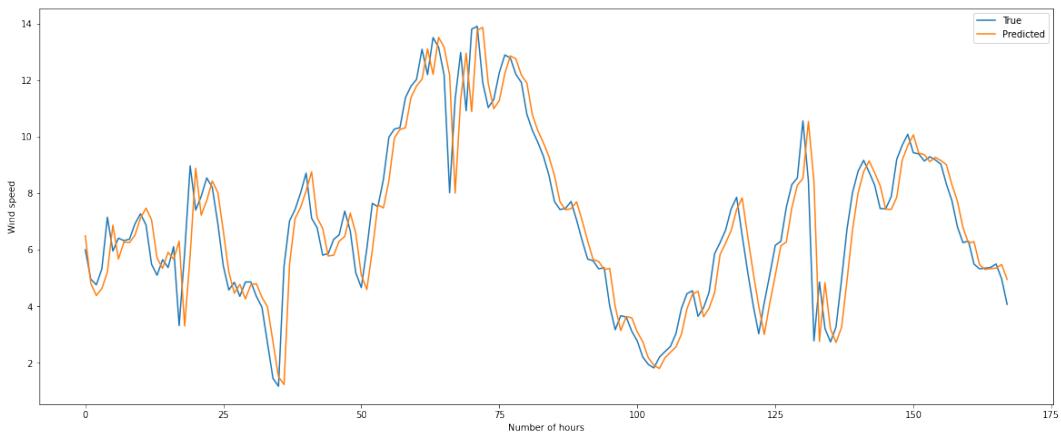


Figure 8.12: Plot between predicted and actual Wind speeds for 1 Week

Error analysis was performed using various performance metrics and its values are as follows:

Mean Absolute Percent Error: 14.479859335544349

Expected variance score 0.8375765599848187

Max error: 5.6369505921006215

Mean absolute error: 0.8223345902369779

Mean squared error: 1.3595990407884695

Root mean squared error: 1.1660184564527567

Mean squared log error: 0.03210385123922087

Median absolute error: 0.5593748205453153

R2 score: 0.8375492566467966

R2 score variance weighted: 0.8375492566467967

Mean Poisson deviance: 0.23978568807353853

Mean gamma deviance: 0.058670279199896924

8.3.1.5 LSTM Model for 1 Month

The plot between predicted and actual wind speed values is shown in Figure 8.13.

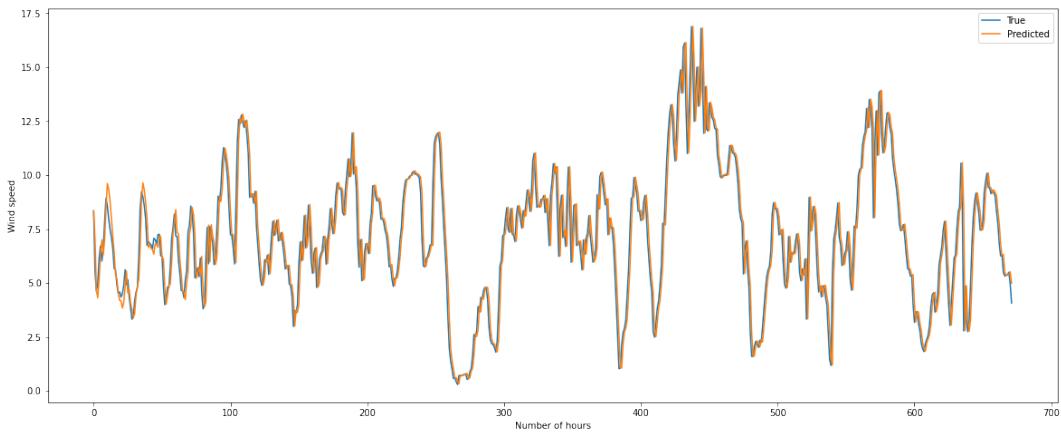


Figure 8.13: Plot between predicted and actual Wind speeds for 1 Month

Error analysis was performed using various performance metrics and its values are as follows:

Mean Absolute Percent Error: 12.699019489156745

Expected variance score 0.8851405940837489

Max error: 5.6796272471621645

Mean absolute error: 0.7581270992497808

Mean squared error: 1.0436690622706315

Root mean squared error: 1.0216012246814465

Mean squared log error: 0.020347745881376608

Median absolute error: 0.5916272077839819

R2 score: 0.8850654492017854

R2 score variance weighted: 0.8850654492017854

Mean Poisson deviance: 0.16032882697665563

Mean gamma deviance: 0.035626000506642476

8.3.2 Sequence to Sequence Models

The data is reshaped first there univariate one dimensional array is converted to two dimensional.

The parameters are initialized namely input sequence length, output sequence length, input features, output features and batch size.

Input and output sequence pairs are generated. The returned sequences are three dimensional tensors which are then fed to the encoder.

Keras by default shifts the training sets randomly.

The model is trained on 39258 samples and validated on 4363 samples. And finally results are obtained.

8.3.2.1 Single Layer Model with Unidirectional LSTM as encoder

The model was created and run using Python programming language. Packages namely Keras, Sklearn, Seaborn ,Numpy,Matplotlib and pandas were employed. Adam stochastic optimization algorithm was employed for optimizing the log likelihood. And mean square error metrics was used for the computation of loss.

The plot between predicted and actual wind speed values is shown in Figure 8.14.

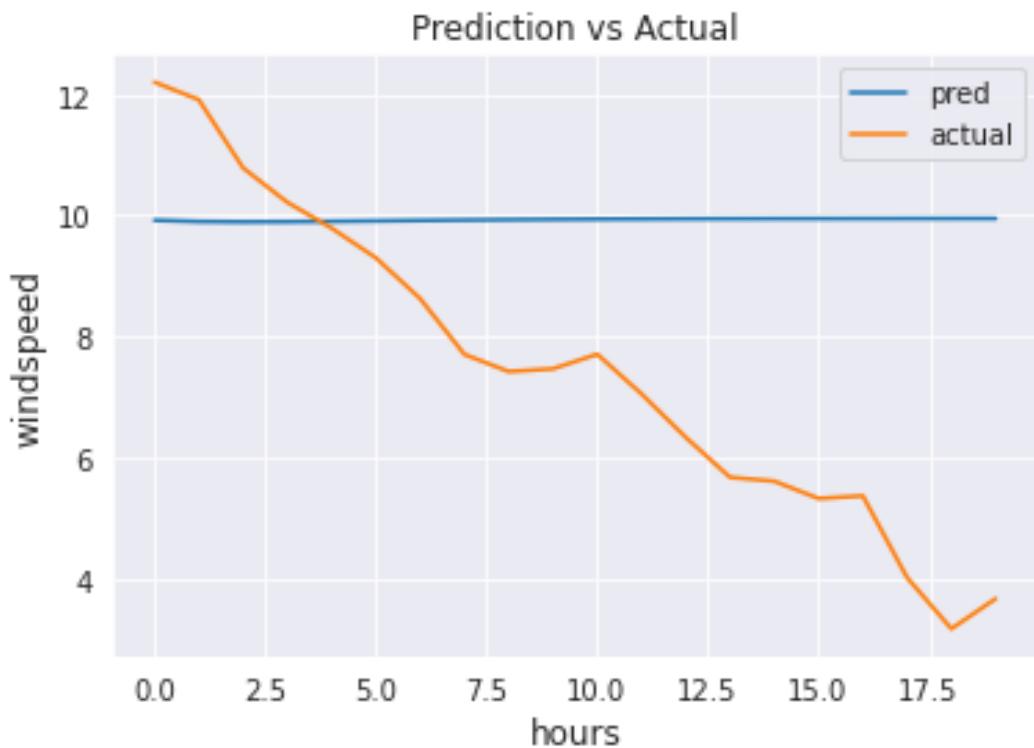


Figure 8.14: Plot between predicted and actual Wind speeds in Unidirectional LSTM

Error analysis were performed using various performance metrics and its values are as follows:

Mean Absolute Percent Error: 57.50454366686123

Expected variance score -0.013472371276716633

Max error: 6.778806213378905

Mean absolute error: 3.013322697448731

Mean squared error: 12.800426448597625

Root mean squared error: 3.5777683615066005

Mean gamma deviance: 0.2058257194858145

Mean poisson deviance: 1.5878368724026979

R2 score: -0.925859496474702

R2 score variance weighted: -0.925859496474702

Median absolute error: 2.489326995849609

8.3.2.2 Double Layer Model with Unidirectional LSTM as encoder

The model was created and run using Python programming language. Packages namely Keras, Sklearn, Seaborn ,Numpy,Matplotlib and panadas were employed. Adam stochastic optimization algorithm was employed for optimizing the log likelihood. And mean square error metrics was used for the computation of loss. The model was run for 70 epochs.

The plot between predicted and actual wind speed values is shown in Figure 8.15.

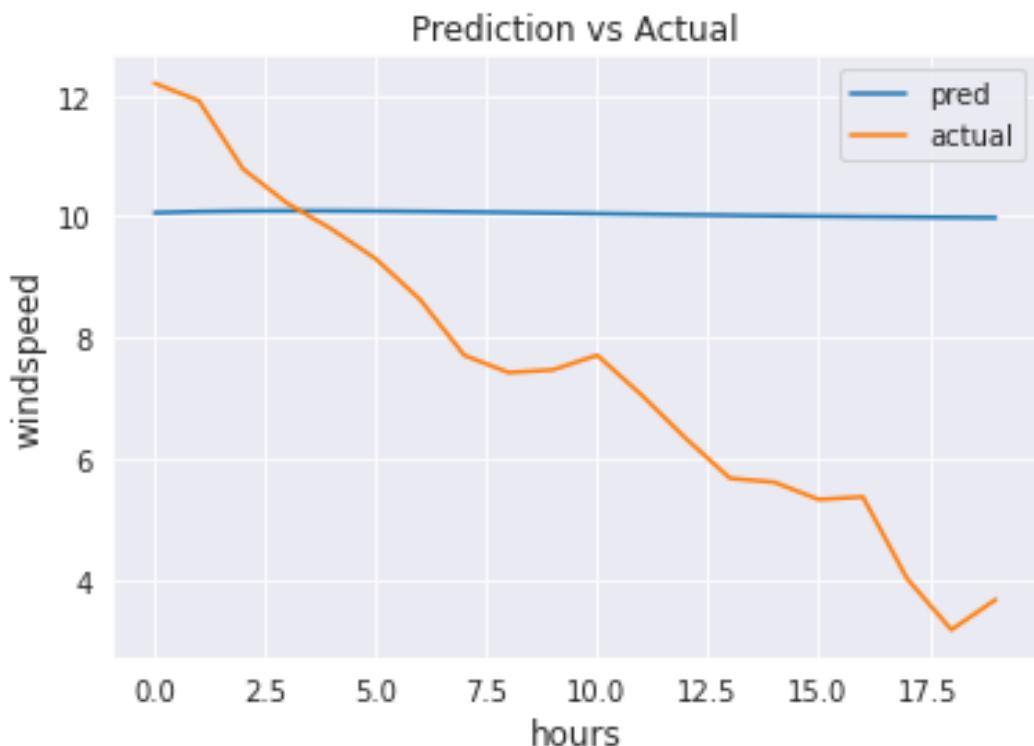


Figure 8.15: Plot between predicted and actual Wind speeds in Stacked Unidirectional LSTM

Error analysis were performed using various performance metrics and its values are as follows:

Mean Absolute Percent Error: 6.5743986160488

Explained variance score : 0.9654180660762777

Max Error : 1.2126127395629887

Mean Absolute Error: 0.4862358800888063

Mean Squared Error: 0.37750097058240706

Root Mean Squared Error: 0.6144110762204789

Mean Squared Log Error: 0.006172711870528008

Median Absolute Error: 0.4339988651275646

R2 score: 0.9432039368341361

R2 score variance weighted: 0.9432039368341361

Mean Poisson Deviance: 0.053640510600663216

8.3.2.3 Single Layer Model with Bidirectional LSTM as encoder

The model was created and run using Python programming language. Packages namely Keras, Sklearn, Seaborn ,Numpy,Matplotlib and panadas were employed. A two layer stacked model was made.Adamstochastic optimization algorithm was employed for optimizing the log likelihood. And mean square error metrics was used for the computation of loss.

The plot between predicted and actual wind speed values is shown in Figure 8.16.

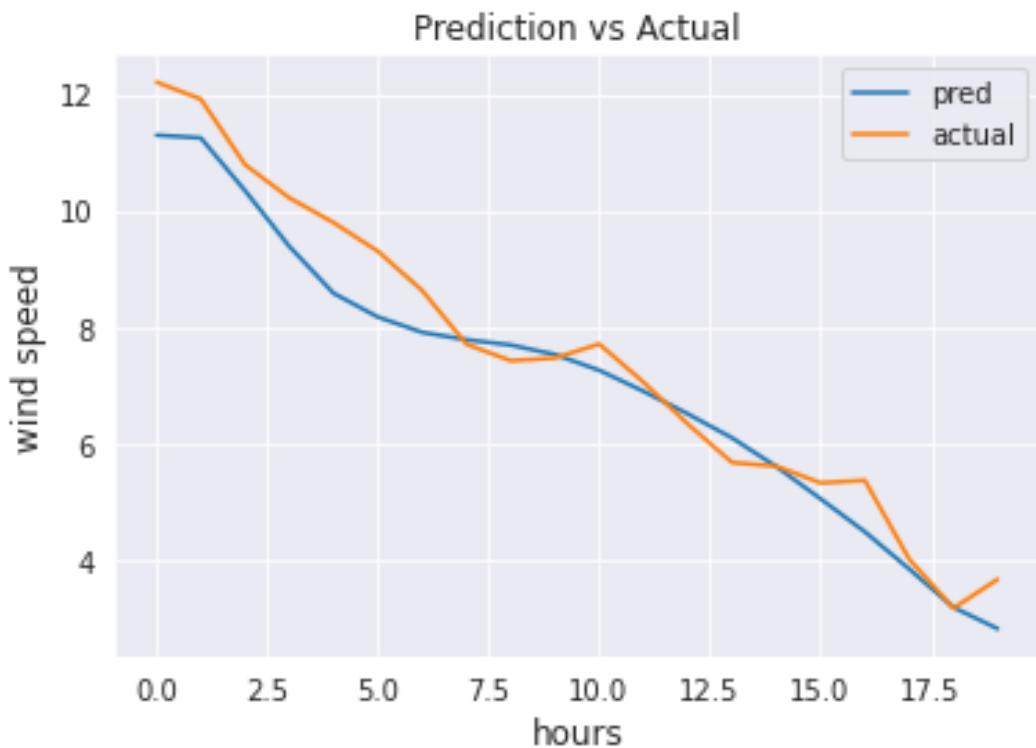


Figure 8.16: Plot between predicted and actual Wind speeds in Bidirectional LSTM

Error analysis were performed using various performance metrics and its values are as follows:

Mean Absolute Percent Error: 58.30939959579341

Expected variance score 0.02656145919657471

Max error: 6.813560966491698

Mean absolute error: 3.056222639846802

Mean squared error: 13.11049409597631

Root mean squared error: 3.620841628126852

Mean gamma deviance: 0.208935226097745

Mean poisson deviance: 1.6188876528322276

R2 score: -0.9725100300058902

R2 score variance weighted: -0.9725100300058901

Median absolute error: 2.6181517066955564.

Mean squared log error: 0.2006696996899764

8.3.2.4 Double Layer Model with Bidirectional LSTM as encoder

The model was created and run using Python programming language. Packages namely Keras, Sklearn, Seaborn ,Numpy,Matplotlib and panadas were employed. A two layer stacked model was made. Adam stochastic optimization algorithm was employed for optimizing the log likelihood. And mean square error metrics was used for the computation of loss. The model was run for 70 epochs.

The plot between predicted and actual wind speed values is shown in Figure 8.17.

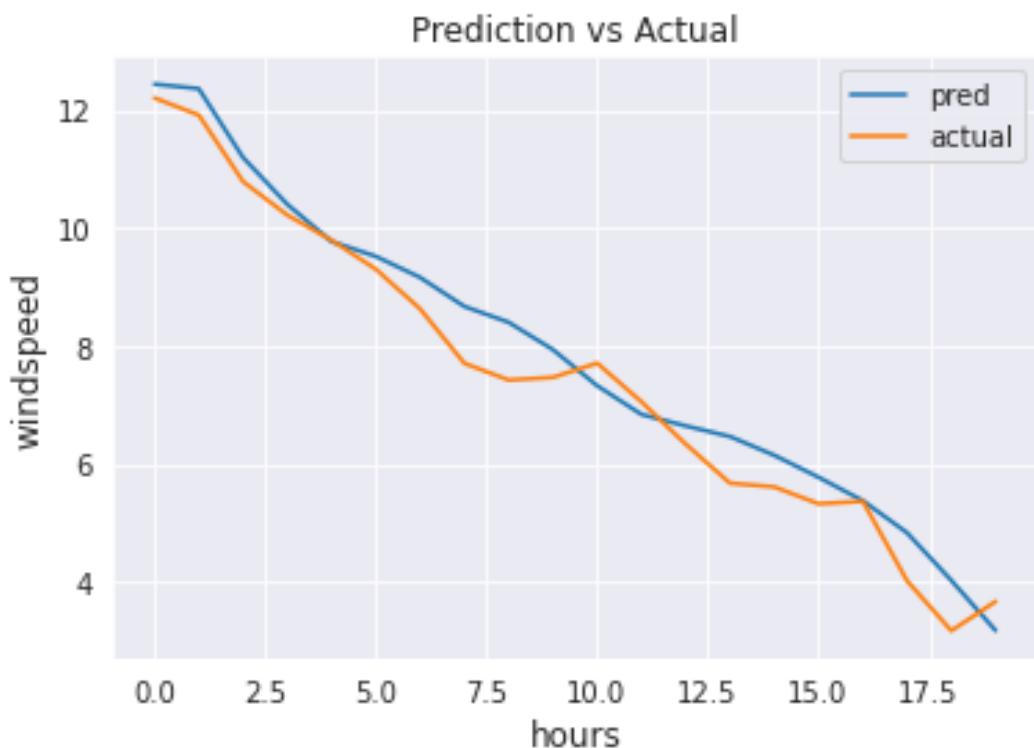


Figure 8.17: Plot between predicted and actual Wind speeds in Stacked Bidirectional LSTM

Error analysis were performed using various performance metrics and its values are as follows:

Mean Absolute Percent Error: 7.88239038870416

Explained variance score : 0.9742485285203182

Max Error : 0.9789252319335935

Mean Absolute Error: 0.4655683166503907

Mean Squared Error: 0.2966093714157968

Root Mean Squared Error: 0.5446185558864083

Mean Squared Log Error: 0.0064918513043312615

Median Absolute Error: 0.4485965080261236

R2 score: 0.9553743012407929

R2 score variance weighted: 0.9553743012407929

Mean Poisson Deviance: 0.048723004646229605

Chapter 9

Power Prediction using Stacked Model

Power generated is estimated using wind speed. Here input is wind speed and target variable is power. Model stacking is performed in order to obtain higher performances than individual parent algorithms.

9.1 MODEL STACKING

Stacking is the process of using various machine learning algorithms one after another, where one sums up the predictions from each algorithm to make a new feature. According to past researches there is no particular exact way to implement model stacking. Here in the power prediction model input feature is wind speed and the target variable is power generated. To avoid overfitting, model stacking should be always accompanied by cross validation.

9.1.1 Stacking Process

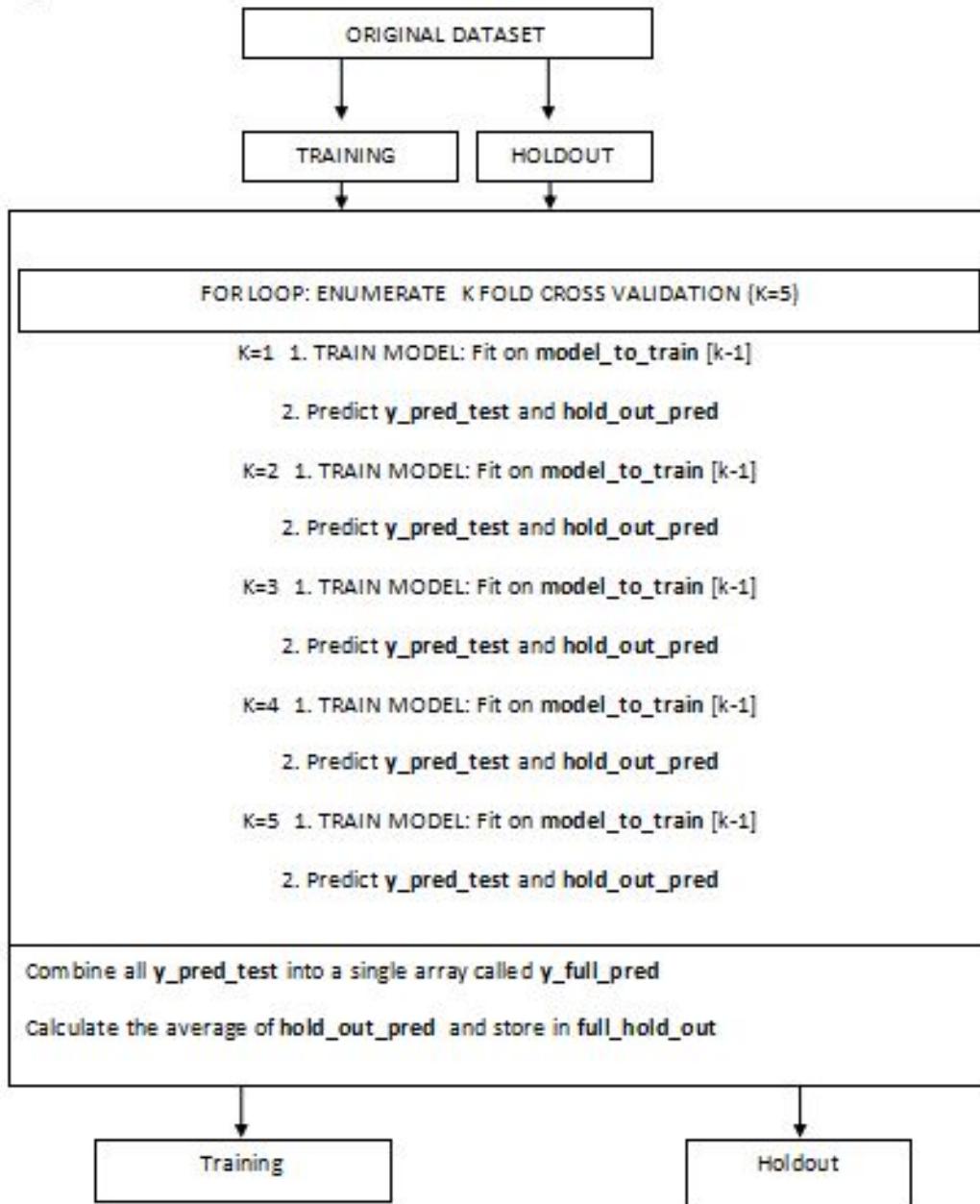


Figure 9.1: Stacking Process

In the end y_{full_pred} and $full_hold_out$ is used a new feature. Figure 9.1 clearly depicts the model stacking process.

Here we use one part of data to predict using one model at a time hence we predict one fold at a time, with a different model each time.

9.1.2 Layers of Stacking

Stacking is performed using layers and there can be arbitrarily multiple layers, which depends on exactly how many models have been trained, along with best combination of models. Model stacks are put in layers , at the end the final data set is fed into a last model. This last model is called meta learner ,its sole purpose is to generalize all the features from each layer into final predictions.

9.1.3 Stack Of Models

Stack of models aid a numerous ways like the meta learner generalizes better than a model. Model stacking deduces the bias in a model on a particular dataset.

9.1.4 Implementation

Before performing model stacking one needs to optimize the hyper parameters for each algorithm, This is performed using a searching algorithm. Grid search algorithm is used in our power prediction model.

9.1.4.1 Grid Search

The goal of any algorithm is to find a function to minimize the expected loss. For minimizing the loss, the values of the parameters involved in the function must be the optimized by optimization techniques like Stochastic Gradient Descent etc. However, the best performance of an algorithm is attained only when the best possible values of hyper-parameters controlling the external performance of an algorithm is obtained. The trial and error method of changing hyper-parameters is time consuming and impractical. Therefore, there arises the need for hyper-parameter optimization techniques.

Grid Search is a hyper-parameter optimization technique that finds the best possible combination of hyper-parameters of a machine-learning algorithm through an exhaustive

search technique along the grid specified by every combination of hyper-parameters (Cartesian product of hyper-parameters) where each hyper-parameter is having a list of values.

Under Grid search, the technique used for hyper-parameter optimization is the Cross Validation technique. In Cross Validation, the sample is partitioned into k equal sub-samples where a single sample of the k sub-samples is retained as the validation data for the purpose of testing and the remaining $k-1$ sub-samples are taken as the training data for the model. This process of cross validation is repeated for k times, where each of the k sub-samples is used exactly once as the validation data. These k results can be averaged to produce a single estimation having the best possible combination of the hyper-parameters involved. The advantage of the k fold cross validation technique is that all sub-samples are included in the training and validation process and each of the k sub-samples is used exactly once for the validation data.

For power prediction we have considered one years hourly data. The algorithms used here are Light gradient boost , Random forest and Extreme gradient boost. Here predictions are made by varying the test train split ratios and corresponding performance measures are obtained.

9.1.4.2 LightGBM

LightGBM is a gradient boosting framework that uses tree based learning algorithms. Its differentiated from other tree based algorithms as LightGBM implements a leaf-wise growth compared to other tree based algorithms that implement a level-wise growth. When it comes to growing the same leaf, leaf-wise growth reduces loss more effectively than a level-wise growth.

The reason why LightGBM is used more commonly nowadays when it comes to regression or classification tasks is that it can handle large amounts of data and takes lesser memory to

run at the same time.

The use of LightGBM is advisable to datasets having more than 10,000 rows since the algorithm is extremely sensitive to small amounts of data thereby leading to a possibility of model overfitting that fails to generalize well on test data.

The version of LightGBM used for prediction purposes is the LGBMRegressor -the sci-kit learn API for LightGBM Regression.

Parameters of LightGBM:

1. *n – estimators* (*int*, optional (default=100)) – Number of boosted trees to fit.
2. *learning – rate* (*float*, optional (default=0.1)) – Boosting learning rate.
3. *colsample – bytree* (*float*, optional (default=1.)) – Subsample ratio of columns when constructing each tree.
4. *max – depth* (*int*, optional (default=-1)) – Maximum tree depth for base learners, ≤ 0 means no limit.
5. *num – leaves* (*int*, optional (default=31)) – Maximum tree leaves for base learners.
6. *reg – alpha* (*float*, optional (default=0.)) – L1 regularization term on weights.
7. *reg – lambda* (*float*, optional (default=0.)) – L2 regularization term on weights.
8. *min – split – gain* (*float*, optional (default=0.)) – Minimum loss reduction required to make a further partition on a leaf node of the tree.
9. *subsample* (*float*, optional (default=1.)) – Subsample ratio of the training instance.
10. *subsample – freq* (*int*, optional (default=0)) – Frequency of subsample, ≤ 0 means no enable.

9.1.4.3 XGBoost

XGBoost is a decision tree based machine learning algorithm that uses the gradient boosting framework. It is used for classification, regression problems that involve huge data sets to be worked upon. Its significance for machine learning problems can be attributed to rapid learning through distributed and parallel computing and efficient use of memory.

XGBoost uses the gradient boosting framework, which is the combination of gradient descent algorithm along with the boosting ensemble learning technique.

Boosting is an ensemble machine learning technique that involves many base learners implemented in the form of decision trees. To be more specific, Boosting starts with a weak base learner at each stage of the training phase to compensate for the shortcomings of previous or existing base learners with the intention of improving the performance at each stage in terms of minimizing the expected loss. This process continues until a stopping criterion is met (desired accuracy, required number of iterations). Here at each stage a regression tree model is added to the base learner to arrive at the next base learner. The regression tree at each stage of the process is the residual term of the target values and the output given by the base learner at that stage. The loss is minimized by the means of gradient descent- where the gradient of the function at each stage of the iteration is the negative of the residual term at each of the function at each stage of the iterative process.

XGBoost is superior to most of the machine learning algorithms as it:

1. Implements the process of regularization in order to prevent overfitting of the model.
2. Includes a sparsity-aware split-finding algorithm to handle different sparsity patterns in the data.
3. Uses the weighted quantile sketch algorithm to identify optimal split points when it comes to dealing with weighted datasets.

4. Comes with a built in cross-validation at each stage taking away the need to explicitly program the search and to specify the exact no of boosting iterations required in a single run.
5. Uses the max_{depth} parameters and starts pruning trees in the backward manner thereby improving computational efficiency.
6. Has a block structure for parallel learning- for faster computing XGBoost makes use of multiple cores in the CPU. This can be attributed to a block structure in its system design, where data is stored in blocks thereby enabling the data layout to be reused by subsequent iterations instead of being computed again.

The version of XGBoost used for prediction purposes is the XGBRegressor -the sci-kit learn API for XGBoost Regression.

Parameters of XGBRegresor:

1. $n - estimators$ (*int*) – Number of gradient boosted trees. Equivalent to number of boosting rounds.
2. $learning - rate$ (*float*) – Boosting learning rate (xgb's "eta").
3. $gamma$ (*float*) – Minimum loss reduction required to make a further partition on a leaf node of the tree.
4. $subsample$ (*float*) – Subsample ratio of the training instance.
5. $colsample - bytree$ (*float*) – Subsample ratio of columns when constructing each tree.
6. $max - depth$ (*int*) – Maximum tree depth for base learners.

9.1.4.4 Random Forest

It's a flexible and convenient to use algorithm. The main merit of this algorithm is that a lot of hyper parameter tuning need not be done thereby making it easy to use. Using random forest algorithm both regression as well as classification can be performed. It makes use of multiple decision trees and a statistical technique called bagging. Here averaging of decision trees are not done. The two basic concepts that give it the name random are random sampling of training observations and random subsets of features are used for splitting nodes.

Random forest algorithm creates multiple decision trees and its merged together than relying upon individual trees. Each tree learns from a random sample of the training sets. The samples are drawn with replacement, called as bootstrapping, which means that some samples will be utilized many a times in a single tree. Here each tree observes only a subset of all the features when deciding to split a node. Each tree acts as an information source. If there is only one tree that means there is only a limited scope of information where as by combining many a trees the net information will be much greater.

Working of random forest algorithm:

1. Samples are obtained one after another from the training data so that each data point is having an equal probability of getting selected, and all the samples have the same volume as the original training set.

Let's say we have the following data:

$$x = 0.1, 0.5, 0.4, 0.8, 0.6,$$

$$y = 0.1, 0.2, 0.15, 0.11, 0.13$$

where x is an independent variable with 5 data points and y is dependent variable.

Now Bootstrap samples are obtained with replacement from the above data set. *n_estimators* is set to 3 (no of tree in random forest), then: The first tree will have a

bootstrap sample of size 5 (same as the original dataset), assuming it to be:

$x_1=0.5, 0.1, 0.1, 0.6, 0.6$ likewise

$x_2=0.4, 0.8, 0.6, 0.8, 0.1$

$x_3=0.1, 0.5, 0.4, 0.8, 0.8$

2. A Random Forest Regressor model is trained at each bootstrap sample drawn in the above step, and a prediction is recorded for each sample.
3. Now the ensemble prediction is calculated by averaging the predictions of the above trees producing the final prediction.

9.2 RESULTS AND DISCUSSION

The stacked model is run with 3 different test volumes at 10, 20, and 30 percent.

9.2.1 Stacked Model with 10 percent Test Volume

The Figure 9.2 shows the plot with predicted and actual power plotted against time.

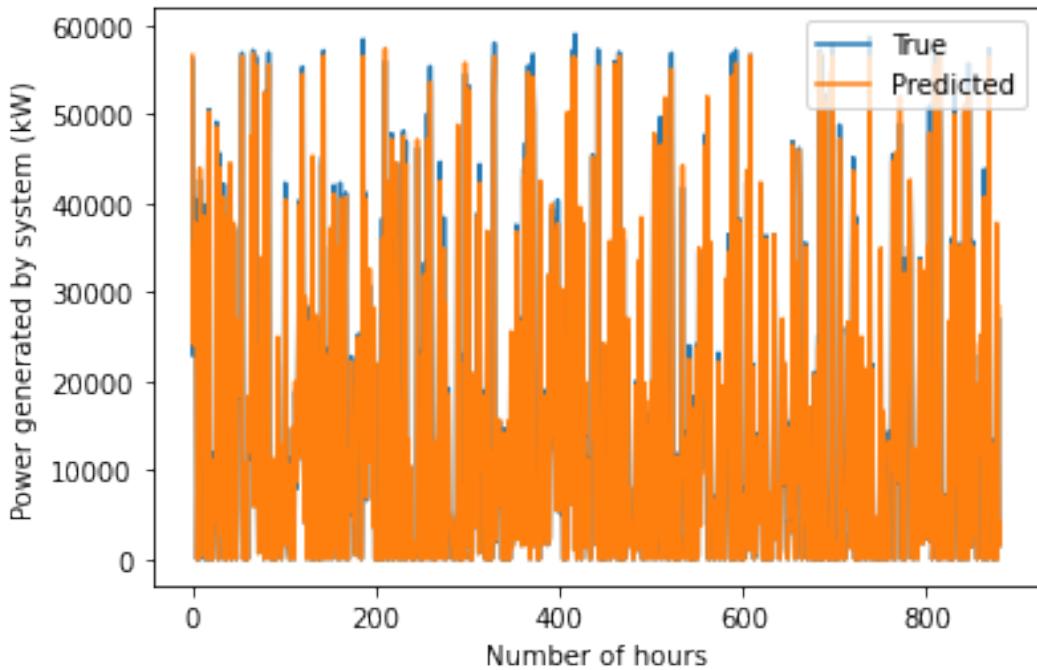


Figure 9.2: Plot between predicted and actual Power in Stacked Model - 10 percent test volume

When test volume is 10 percent following performance measures are obtained:

Explained variance score : 0.9978987278100724

Max Error : 3899.107792160765

Mean Absolute Error: 434.33228799139584

Mean Squared Error: 570418.6197171309

Root Mean Squared Error: 755.260630323818

Mean Squared Log Error: 0.9737061573425344

Median Absolute Error: 161.70321580979817

R2 score: 0.9978936170623237

R2 score variance weighted: 0.9978936170623237

Mean Poisson Deviance: 23.853459289109832

9.2.2 Stacked Model with 20 percent Test Volume

The Figure 9.3 shows the plot with predicted and actual power plotted against time.

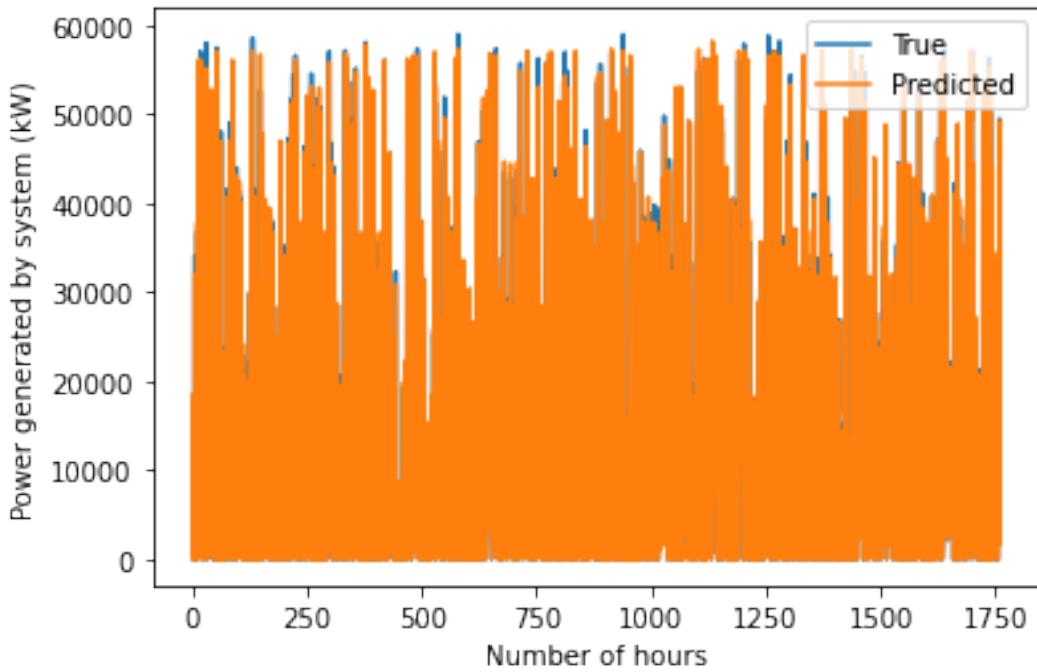


Figure 9.3: Plot between predicted and actual Power in Stacked Model - 20 percent test volume

When test volume is 20 percent following performance measures are obtained:

Explained variance score : 0.9976133126077652

Max Error : 4118.507938568931

Mean Absolute Error: 450.6997346358674

Mean Squared Error: 658047.6015100346

Root Mean Squared Error: 811.201332290594

Mean Squared Log Error: 0.9232290214948073

Median Absolute Error: 141.17370795701572

R2 score: 0.9976097642825411

R2 score variance weighted: 0.9976097642825411

Mean Poisson Deviance: 25.182864529640398

9.2.3 Stacked Model with 30 percent Test Volume

The Figure 9.4 shows the plot with predicted and actual power plotted against time.

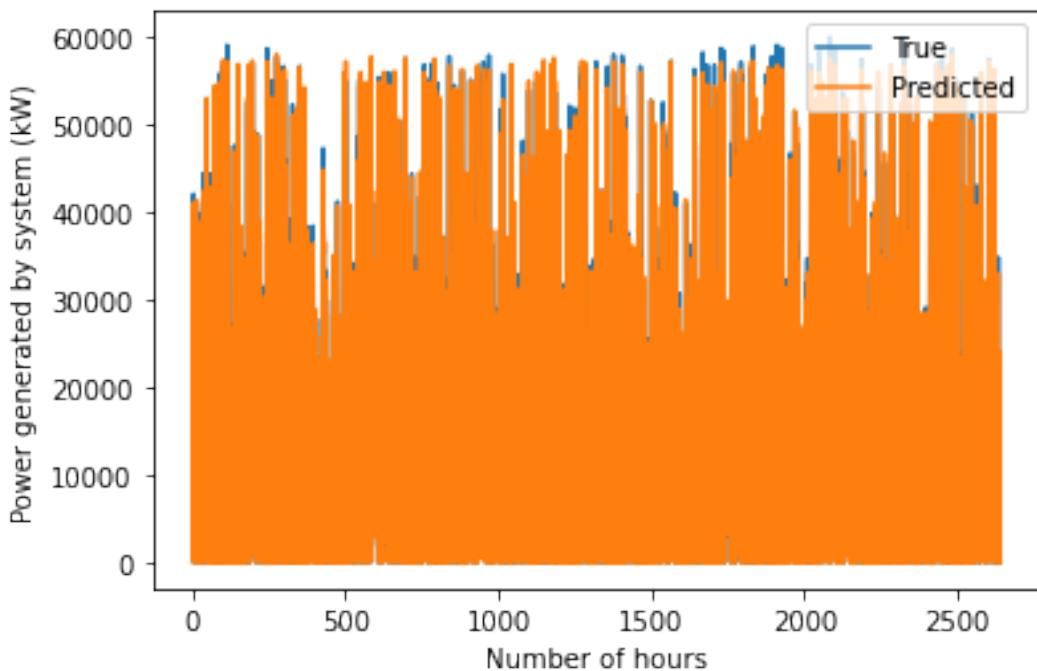


Figure 9.4: Plot between predicted and actual Power in Stacked Model - 30 percent test volume

When test volume is 30 percent following performance measures are obtained:

Explained variance score : 0.9976919890085686

Max Error : 5126.930975912845

Mean Absolute Error: 449.4109122290542

Mean Squared Error: 610873.8879718882

Root Mean Squared Error: 781.5842168134462

Mean Squared Log Error: 1.1382385421476546

Median Absolute Error: 163.98089140697402

Mean Poisson Deviance: 25.69566850246148

Chapter 10

Proposed Algorithm

10.1 PROPOSED POWER PREDICTION ALGORITHM

The proposed algorithm for wind power prediction is:

1. `(x_train,x_test,windpower_train,windpower_test) = split_train_test(training ratio)`
2. `model_xgb= fit_algorithm_xgb(x_train,windpower_train)`
3. `model_ran_forest= fit_algorithm_ran_forest(x_train,windpower_train)`
4. `model_lgbm= fit_algorithm_lgbm(x_train,windpower_train)`
5. `model=model_xgb,model_ran_forest,model_lgbm`
6. `for i in range (0,3):`
 `{`
7. `forecasted_power[i] = forecast_power(model[i],x_test)`
 `}`
8. `for j in range (0,3):`
 `{`

9. metrics[j] = calculate_performance(forecasted_power[i],windpower_test)
10. }
11. meta_learner = best_performing_alg(model,metrics)
12. model_chaotic_pred = fit_algorithm(x_train)
13. forecasted_speed = forecast_speed(model_chaotic_pred,time)
14. For loop: Enumerate KFold cross validation (K=3):
15. model_xgbk1 = fit_algorithm_xgb(x_train1,windpower_train1)
16. model_ran_forestk2 = fit_algorithm_ran_forest(x_train2,windpower_train2)
17. model_lgbmk3 = fit_algorithm_lgbm(x_train3,windpower_train3)
18. predicted_powerk1 = forecast_power(model_xgbk1,forecasted_speed1)
19. predicted_powerk2 = forecast_power(model_ran_forestk2,forecasted_speed2)
20. predicted_powerk3 = forecast_power(model_lgbmk3,forecasted_speed3)
21. predicted_powerk = predicted_powerk1+predicted_powerk2+predicted_powerk3
22. forecasted_power = forecast_power(model=metalearner,forecasted_speed,predicted_powerk)
23. metrics = calculate_alg_performance(forecasted_power,windpower_test)
24. return forecasted power

A further variants can be created by changing the algorithm used to predict wind speed.

10.2 RESULTS AND DISCUSSION

10.2.1 Using Long Short Term Neural Networks Algorithm to predict Speed

Long Short Term Neural Networks algorithm was used to predict speed which is provided as input to the stacked model for power prediction and following performance measures were obtained for various prediction windows.

10.2.1.1 For Prediction Window = 12 Hours

The Figure 10.1 shows the plot with actual and predicted power plotted against time for prediction window = 12 Hours.

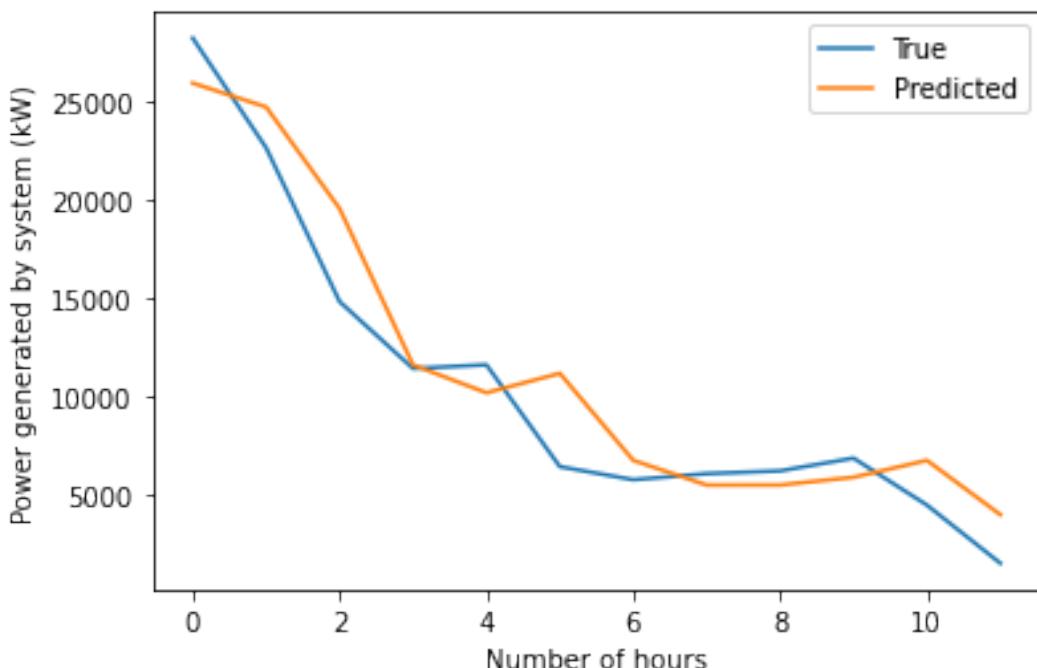


Figure 10.1: Plot between predicted and actual Power for 12 Hours

The following performance measures were obtained from the model:

Explained variance score : 0.9136213763252361

Max Error : 4737.263728395406

Mean Absolute Error: 1950.4395617726448

Mean Squared Error: 5857896.054773829

Root Mean Squared Error: 2420.3090824879846

Mean Squared Log Error: 0.12878229465969088

Median Absolute Error: 1746.9972120179427

R2 score: 0.8976929066563523

R2 score variance weighted: 0.8976929066563523

Mean Poisson Deviance: 621.7101139268603

10.2.1.2 For Prediction Window = 24 Hours

The Figure 10.2 shows the plot with actual and predicted power plotted against time for prediction window = 24 Hours.

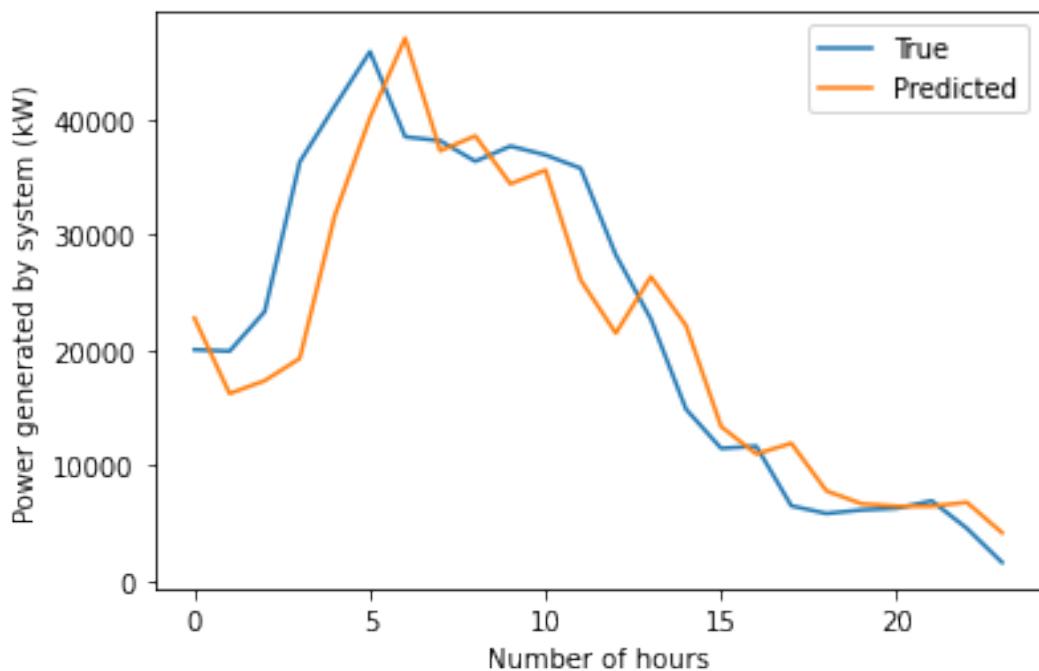


Figure 10.2: Plot between predicted and actual Power for 24 Hours

The following performance measures were obtained from the model:

Prediction window =1 day

Explained variance score : 0.8357235152889351

Max Error : 17033.961908951052

Mean Absolute Error: 4340.55200509625

Mean Squared Error: 33997483.279123016

Root Mean Squared Error: 5830.736083816778

Mean Squared Log Error: 0.1104697132354943

Median Absolute Error: 2997.79760860388

R2 score: 0.8299841185366283

R2 score variance weighted: 0.8299841185366283

Mean Poisson Deviance: 1488.4231200562042

10.2.1.3 For Prediction Window = 2 days

The Figure 10.3 shows the plot with actual and predicted power plotted against time for prediction window = 2 days.

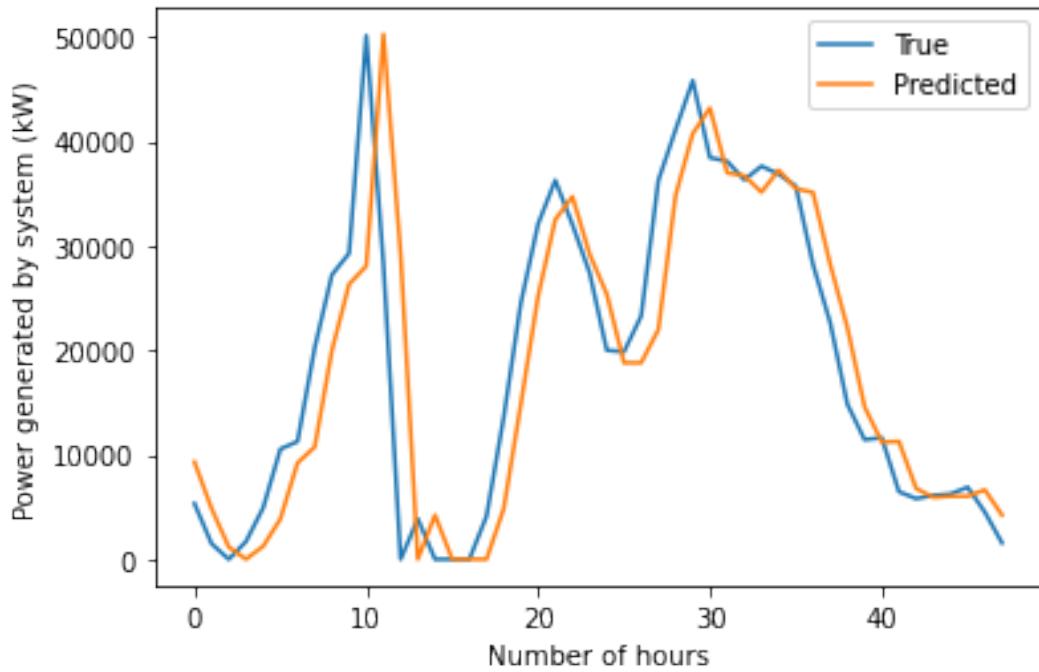


Figure 10.3: Plot between predicted and actual Power for 2 days

The following performance measures were obtained from the model:

Explained variance score : 0.7235197052202127

Max Error : 29214.2350029321

Mean Absolute Error: 5055.96102782156

Mean Squared Error: 60267618.957658134

Root Mean Squared Error: 7763.222202002087

Mean Squared Log Error: 7.292809590147876

Median Absolute Error: 3726.6825183727938

R2 score: 0.722959476893653

R2 score variance weighted: 0.722959476893653

Mean Poisson Deviance: 5579.778199276084

10.2.1.4 For Prediction Window = 1 week

The Figure 10.4 shows the plot with actual and predicted power plotted against time for prediction window = 1 week.

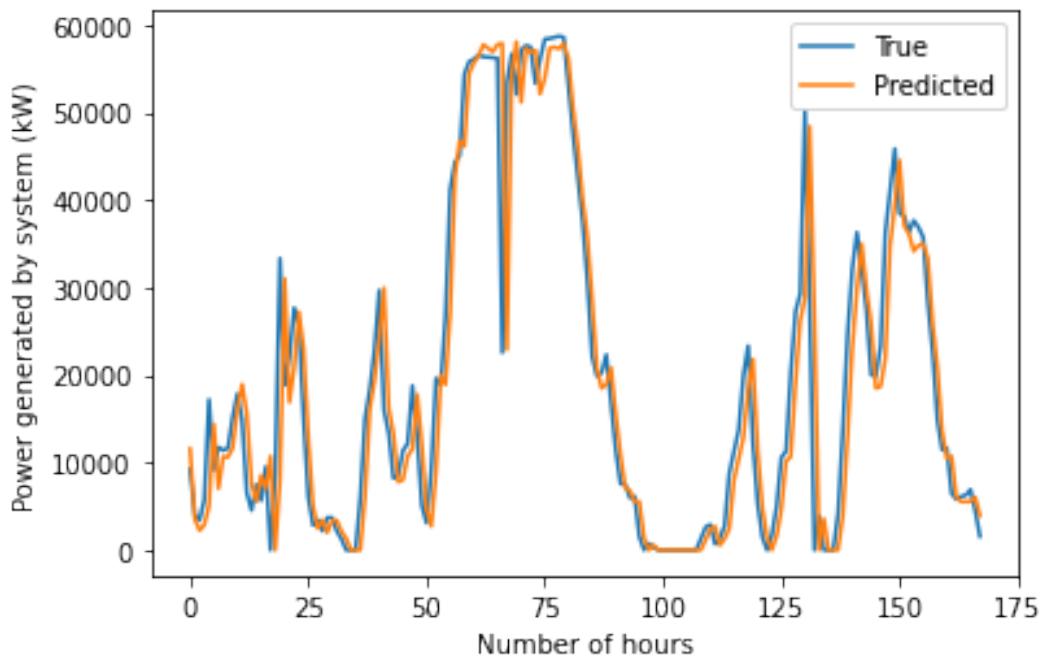


Figure 10.4: Plot between predicted and actual Power for 1 week

The following performance measures were obtained from the model:

Explained variance score : 0.8639622650462684

Max Error : 35307.187164235256

Mean Absolute Error: 4302.1604432845625

Mean Squared Error: 48016217.11685336

Root Mean Squared Error: 6929.373501035527

Mean Squared Log Error: 4.475431720975699

Median Absolute Error: 2848.910912067

R2 score: 0.8632020062024162

R2 score variance weighted: 0.8632020062024163

Mean Poisson Deviance: 4521.028378227723

10.2.1.5 For Prediction Window = 1 month

The Figure 10.5 shows the plot with actual and predicted power plotted against time for prediction window = 1 month.

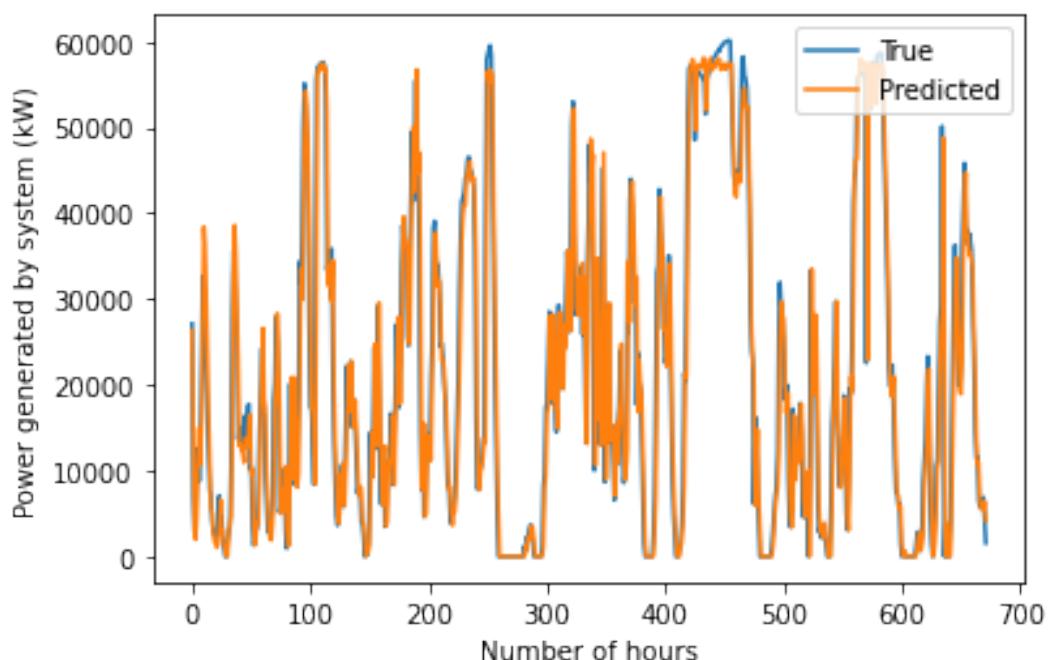


Figure 10.5: Plot between predicted and actual Power for 1 month

The following performance measures were obtained from the model:

Explained variance score : 0.8728393084109249

Max Error : 34791.99417641895

Mean Absolute Error: 4416.330823730658

Mean Squared Error: 43703867.71317309

Root Mean Squared Error: 6610.890084789876

Mean Squared Log Error: 2.4762766101905576

Median Absolute Error: 2804.6063496026563

R2 score: 0.8727515426781678

R2 score variance weighted: 0.8727515426781679

Mean Poisson Deviance: 2961.186059919755

10.2.2 Using Unidirectional Sequence to sequence Long Short Term Neural Network to Predict Power

10.2.2.1 Predictions for Unidirectional Single Layer Sequence

The Figure 10.6 shows the plot with actual and predicted power plotted against time using Single Layer Unidirectional Sequence.

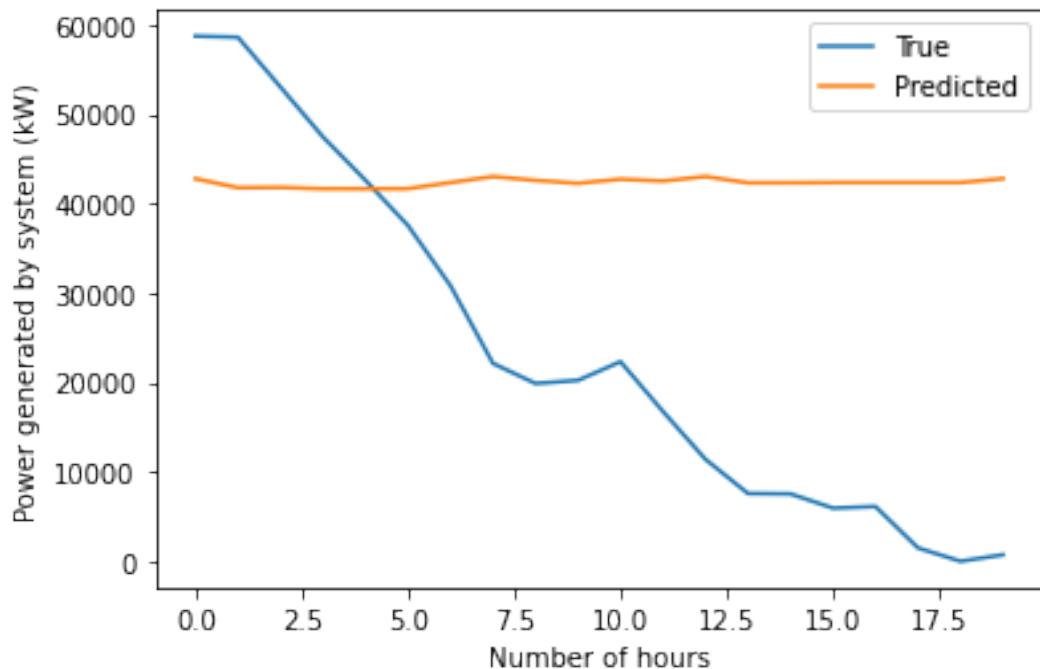


Figure 10.6: Predictions from Unidirectional Single Layer Sequence to Sequence Model

The following performance measures were obtained from the model when Unidirectional Single Layer Sequence to Sequence Long Short Term Neural Networks algorithm was used to predict speed which is provided as input to the stacked model for power :

Explained variance score : -0.02217835117593414

Max Error : 42328.909165671066

Mean Absolute Error: 23851.902113710585

Mean Squared Error: 731966084.6166536

Root Mean Squared Error: 27054.871735357636

Mean Squared Log Error: 7.992852154804507

Median Absolute Error: 22363.775645219237

R2 score: -0.9726965545886013

R2 score variance weighted: -0.9726965545886013

Mean Poisson Deviance: 27264.253117249766

10.2.2.2 Predictions for Unidirectional Double Layer Sequence

The Figure 10.7 shows the plot with actual and predicted power plotted against time using Double Layer Unidirectional Sequence.

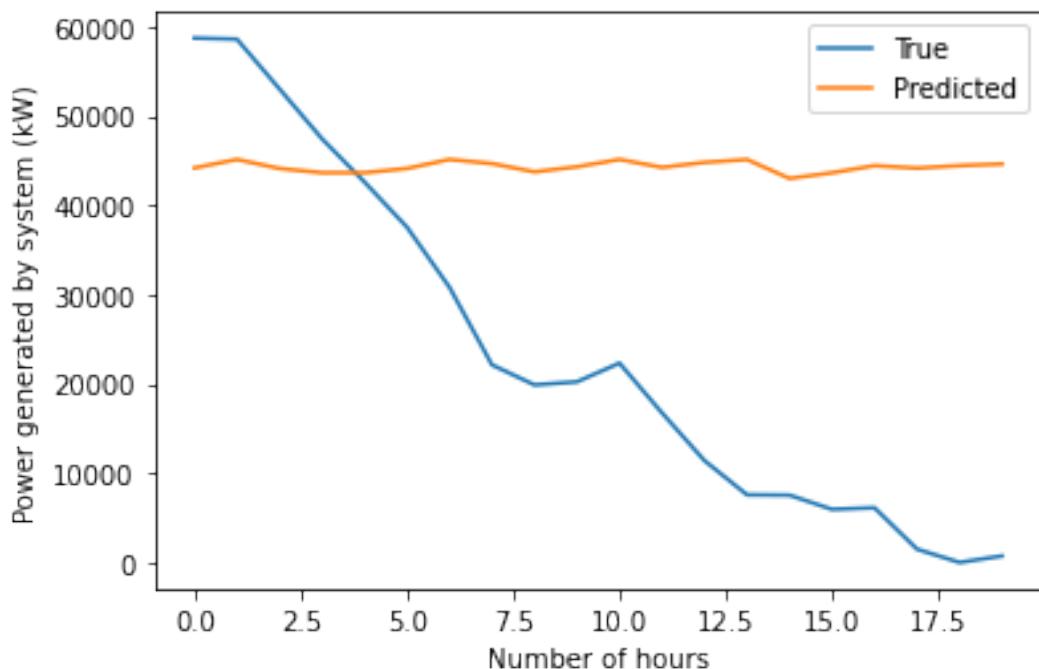


Figure 10.7: Predictions from Unidirectional Double Layer Sequence to Sequence Model

The following performance measures were obtained from the model when Unidirectional Double Layer Sequence to Sequence Long Short Term Neural Networks algorithm was used to predict speed which is provided as input to the stacked model for power prediction:

Explained variance score : -0.0010636702530013586

Max Error : 44415.67089680435

Mean Absolute Error: 24831.06421523653

Mean Squared Error: 802380360.370064

Root Mean Squared Error: 28326.319216764892

Mean Squared Log Error: 8.124067364090838

Median Absolute Error: 23958.372787410943

R2 score: -1.162467641107388

R2 score variance weighted: -1.162467641107388

Mean Poisson Deviance: 28882.389666581184

10.2.3 Using Bidirectional Sequence to sequence Long Short Term Neural Network to Predict Power

10.2.3.1 Predictions for Bidirectional Single Layer Sequence

The Figure 10.8 shows the plot with actual and predicted power plotted against time using Single Layer Bidirectional Sequence.

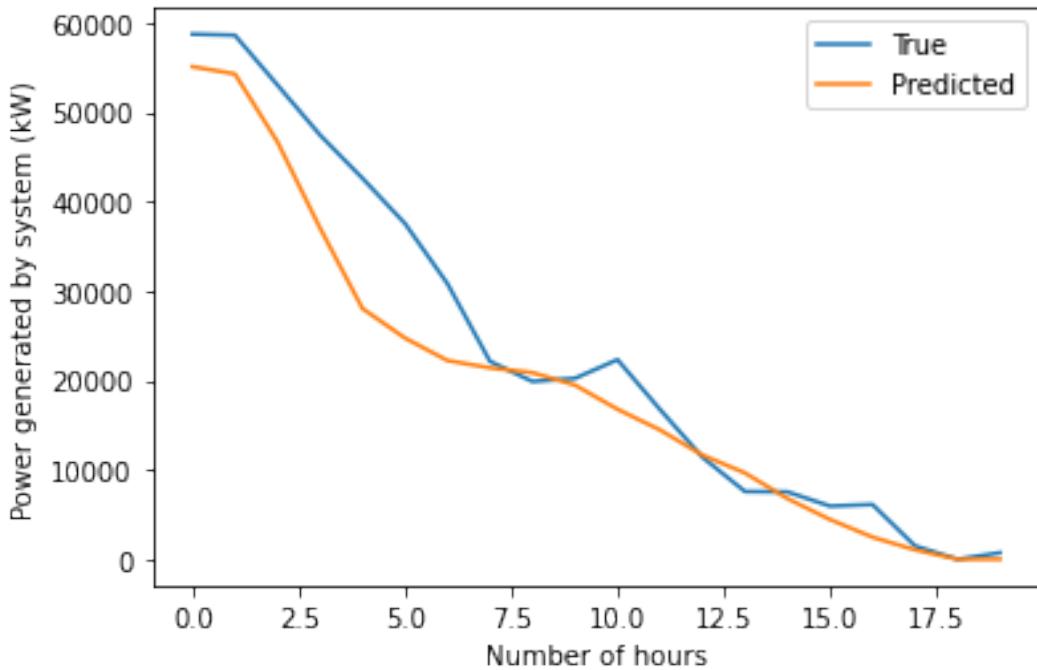


Figure 10.8: Predictions from Bidirectional Single Layer Sequence to Sequence Model

The following performance measures were obtained from the model when Bidirectional Single Layer Sequence to Sequence Long Short Term Neural Networks algorithm was used to predict speed which is provided as input to the stacked model for power :

Explained variance score : 0.9437310582677259

Max Error : 14558.524302371701

Mean Absolute Error: 4017.0000573947145

Mean Squared Error: 34470765.551090285

Root Mean Squared Error: 5871.1809332612365

Mean Squared Log Error: 1.390238235181822

Median Absolute Error: 2181.3753522574743

R2 score: 0.9070990284033713

R2 score variance weighted: 0.9070990284033713

Mean Poisson Deviance: 1605.3254868482331

10.2.3.2 Predictions for Bidirectional Double Layer Sequence

The Figure 10.9 shows the plot with actual and predicted power plotted against time using Double Layer Bidirectional Sequence.

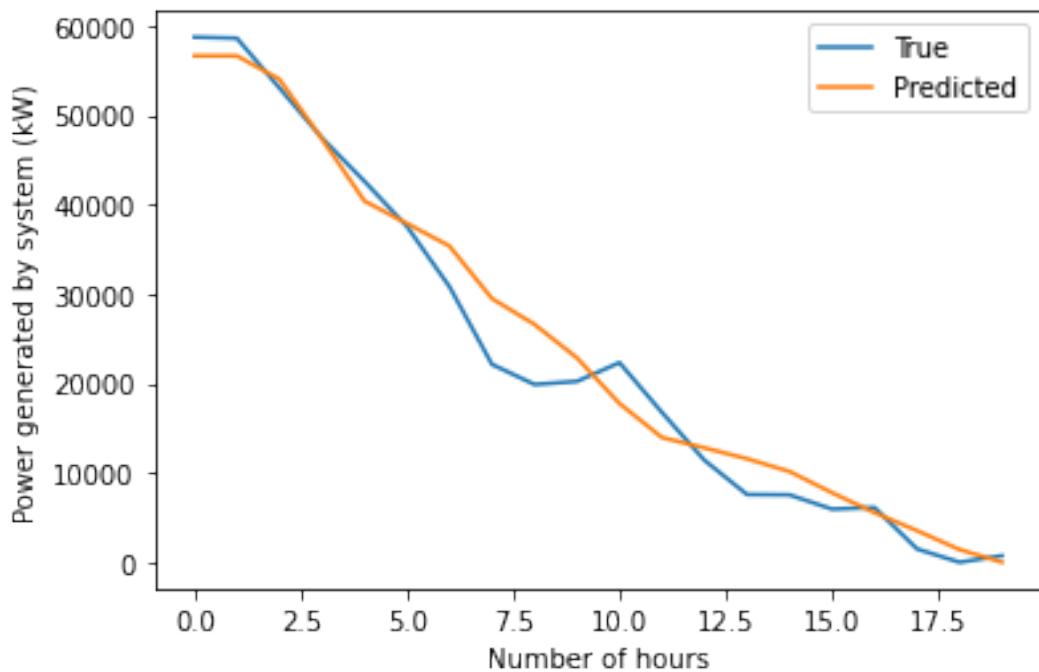


Figure 10.9: Predictions from Bidirectional Double Layer Sequence to Sequence Model

The following performance measures were obtained from the model when Bidirectional Double Layer Sequence to Sequence Long Short Term Neural Networks algorithm was used to predict speed which is provided as input to the stacked model for power prediction:

Explained variance score : 0.9753196760253324

Max Error : 7355.625689527078

Mean Absolute Error: 2544.3474053788195

Mean Squared Error: 10240075.397879397

Root Mean Squared Error: 3200.01178089697

Mean Squared Log Error: 3.788544641483022

Median Absolute Error: 2064.981607387

R2 score: 0.9724023258991519

R2 score variance weighted: 0.9724023258991519

Mean Poisson Deviance: 979.4353263963363

Chapter 11

Observations and Results

11.1 EXPLORATORY DATA ANALYSIS

The data considered was the wind speed dataset for a period of 6 years. Before going on to other aspects of the dataset, it is important to have an initial analysis of the dataset in terms of its observations, features, null values, duplicate entries and outliers. This initial analysis of the dataset was carried out by exploratory data analysis.

A general preview of the dataset was generated yielding 52560x5 entries. That is, 52560 observations for each of the 5 features involved in the dataset namely Air temperature, Pressure, Wind speed, Wind direction and Power generated by the system. It was also observed that the dataset had no null values and had 8760 duplicate entries.

Next, a boxplot analysis of the 5 features was carried out and it was observed that the boxplot depicting Air temperature showed a negatively skewed distribution whereas the boxplot depicting pressure showed a normal distribution and the remaining boxplots of Wind speed, Wind direction and Power generated by system showed a positively skewed distribution. From the boxplots, outliers were observable for Air temperature, Pressure and Wind speed.

Next, scatter plots were generated plotting each of the 5 features (on the vertical axis) with the time duration on the horizontal axis. Outliers were observable from the scatter plots for the Pressure and Wind speed features. However, the drawback of scatter plots in detecting outliers is that these plots only do convey a simplistic picture regarding the outliers

present in the dataset as the dataset considered is a multi-dimensional dataset. Therefore, multi-dimensional outlier detection techniques were employed for the dataset namely Angle Based Outlier Detection (ABOD), Feature Bagging, Histogram Based Outlier Score (HBOS), Isolation Forest and K-Nearest Neighbors (KNN).

It was observed that the ABOD technique yielded the most no of outliers for each of the 5 features as the variances of the angle between a pair of points with respect to a point gives a more accurate measure of the presence of outliers. The KNN technique yielded the least number of outliers for each of the 5 features compared to the other techniques as the distance measured between points for outlier classification becomes too simplistic when a multi-dimensional dataset is considered.

11.2 FEATURE ENGINEERING

Feature engineering techniques were employed to scale down the volume of features. That is the independent variable that has more relation with dependent variables were found. First Pearson Correlation analysis was performed on varying data volume. The results are shown in Figure 11.1.

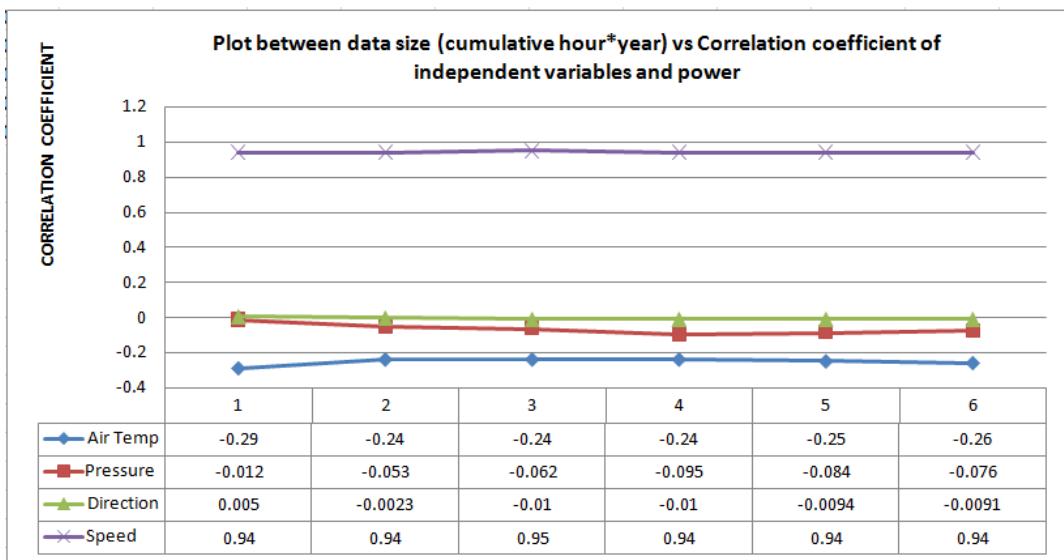


Figure 11.1: Plot between data size and correlation coefficient of independent variables and power

It can be observed that all variables exhibits similar values on changing the data volume, since the curve has a near straight line behavior parallel to x-axis. It is evident that wind power stands out as the most correlated variable with wind power with values nearing 0.95 or 0.94.

Secondly, average mutual information values were obtained for various test sizes. The prime observation made was that the average mutual information did not vary much with change in test sizes. The wind speed variable has a higher distinct mutual information value as compared to others; it is evident from the Figure 11.2.

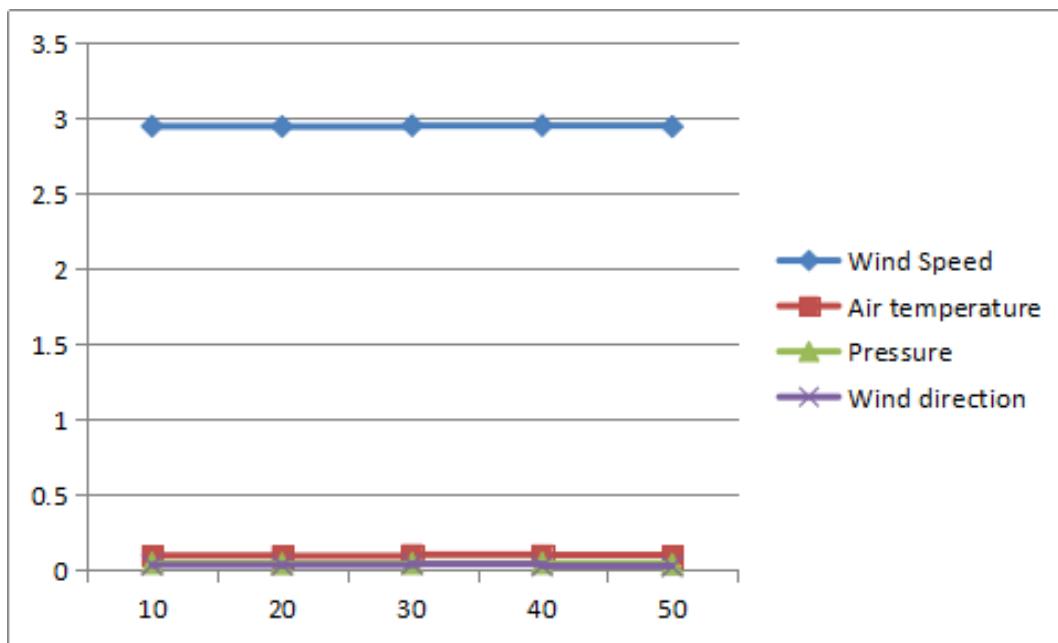


Figure 11.2: Average Mutual Information value for different Variables

So as to deduct a conclusive result study was performed using Mean square error measure. The plots and results are shown in Figure 11.3.

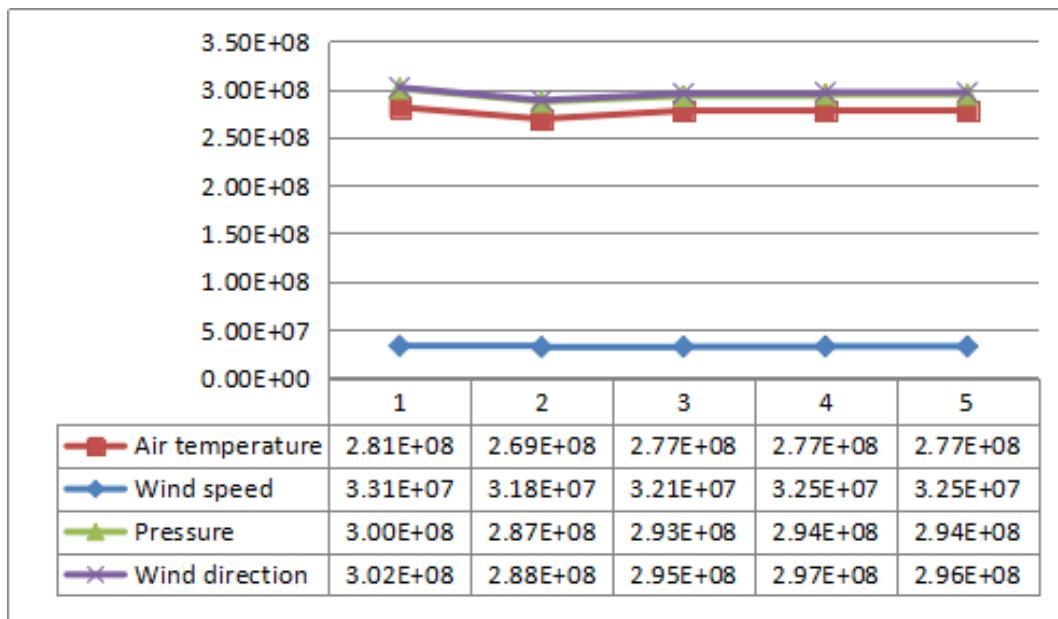


Figure 11.3: Mean Square Error values for different Variables

Here too, the mean square error is calculated for varying test sizes from 10 to 50 percentages. From Figure 11.3, it is evident that the variable wind speed contributes for minimum squared error measure. Rest of the variables had higher error measures.

From all the three different approaches, we can infer that wind speed is the variable, which can be used to estimate wind power values.

Therefore, the data corresponding to the wind speed feature (52560 observations) was considered as this feature had the highest correlation with the Power generated by the system feature.

11.3 TEST FOR RANDOMNESS

Upon a normal plot of the Wind speed time series data, the plot showed no clear pattern or trend over a period. So this data was subjected to a Random-Walk test that showed a certain extent of randomness in the behavior of the dataset over time.

11.4 TEST FOR NON-LINEARITY

After the test for Randomness, the wind speed data was subjected to tests for non-linearity to confirm as to whether the behavior is stationary or non-stationary. The tests used were the visual test, the KPSS test and the surrogate test. From the first test, it was observed that the plot of the autocorrelation function v/s lag, the autocorrelation function did not show a quick degradation to zero thereby indicating a non-stationary behavior. Also for the plot of the partial autocorrelation v/s lag, the partial autocorrelation function had non-zero values after long lag periods thereby confirming a non-stationary behavior. In the KPSS test as the p-value was lesser than all the critical values for different confidence intervals considered, the time series data was concluded to be non-stationary. Finally, from the Surrogate test as the original statistic was found to be higher than the Surrogate statistic, the time series thereby

showed a non-stationary behavior. Therefore, from the above 3 tests it was concluded the time series data had a non-linear behavior.

After showing a certain extent of randomness and depicting a non-stationary and non-linear behavior, the dataset was subjected to the test for chaotic nature.

11.5 TEST FOR CHAOTICITY

The Maximum Lyapunov exponent being positive is a nature of chaotic behavior. For estimating the Lyapunov exponent, the lag values and the embedding dimension are required. At first, the lag was computed by two methods the first one by considering the autocorrelation function and the second one by considering the mutual information function. After estimating the lag values from the above two methods, the embedding dimension was found for the data using Cao's method. The scalar time series data that is, the data corresponding to wind speed was then used to perform phase space reconstruction since by the method of delays or lags, the scalar time series data can yield the nature of the underlying attractor behavior as the noise levels are constant for each delay component. Finally, the maximum Lyapunov exponent for the lag values was computed and the results were found to give a positive value thereby depicting a chaotic behavior.

11.6 CHAOTIC WIND SPEED PREDICTION

11.6.1 LSTM Models for different Prediction Windows

Wind speed was predicted using Long Short Term Neural Networks. Predictions were performed for five prediction windows. Plots were made for various performance measures.

Mean absolute percent error was low for 12 hours ahead prediction. It rose up to a peak value for 2 days ahead prediction as shown in Figure 11.4.

mean absolute percent error

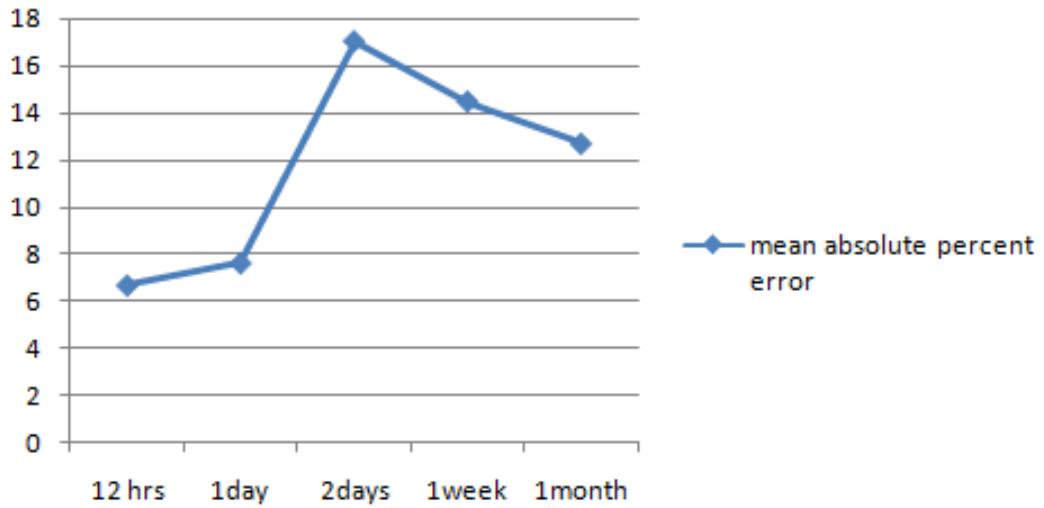


Figure 11.4: Mean Absolute Percent Error values for LSTM Models

Variance score first decreased and a minimum value was obtained at 2 days ahead prediction. Then its subsequent increments ensured its return to initial score as shown in Figure 11.5.

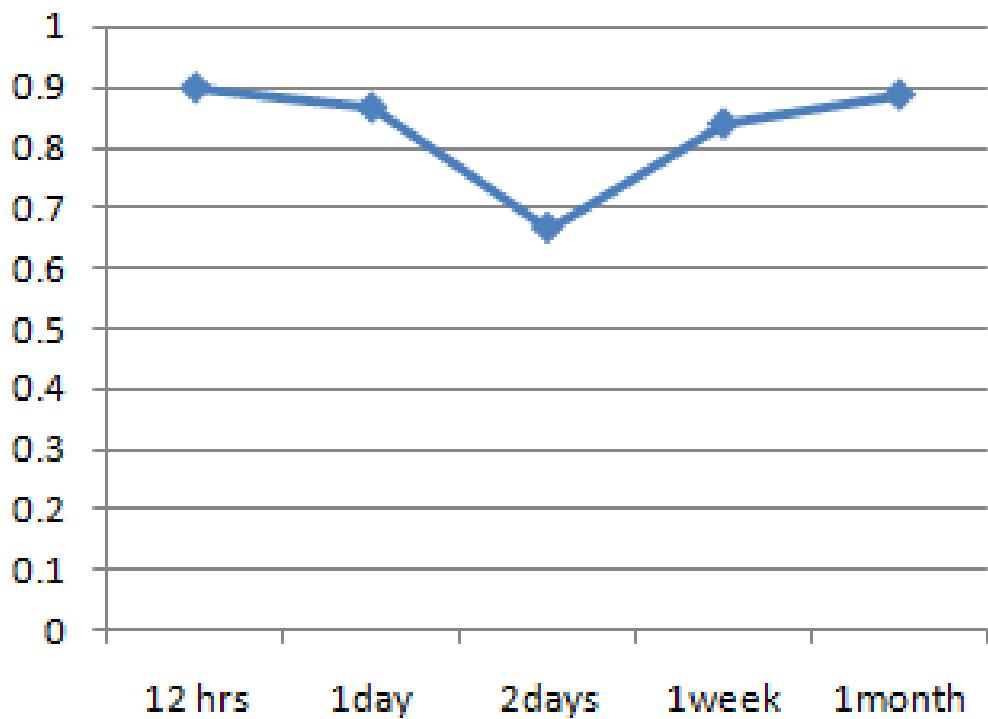


Figure 11.5: Expected Variance values for LSTM Models

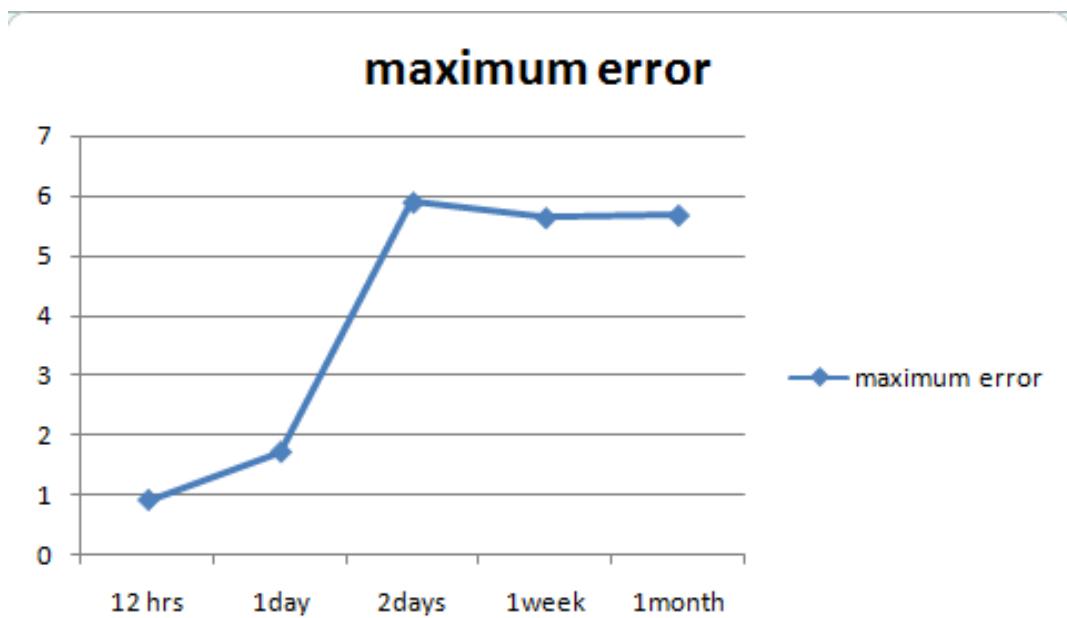


Figure 11.6: Maximum Error values for LSTM Models

Maximum error rose to a maximum value at 2 days ahead prediction window then it

almost became constant as shown in Figure 11.6.

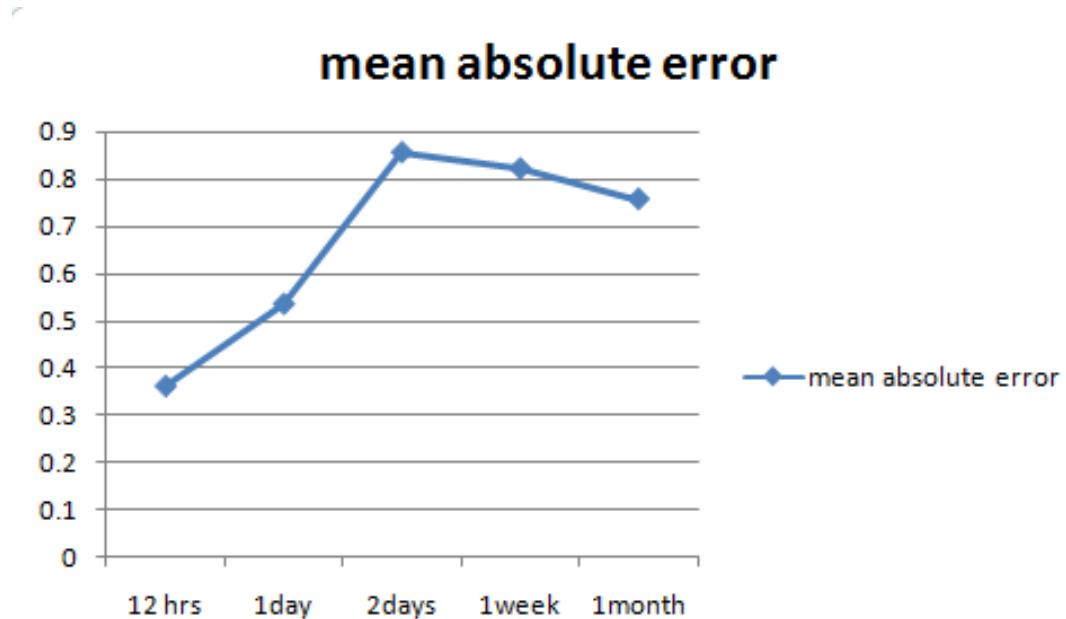


Figure 11.7: Mean Absolute Error values for LSTM Models

Similar to previous observations it peaked at 2 days ahead prediction window, then it decreased but it never came back to its initial rate of decrement as shown in Figure 11.7.

mean squared error

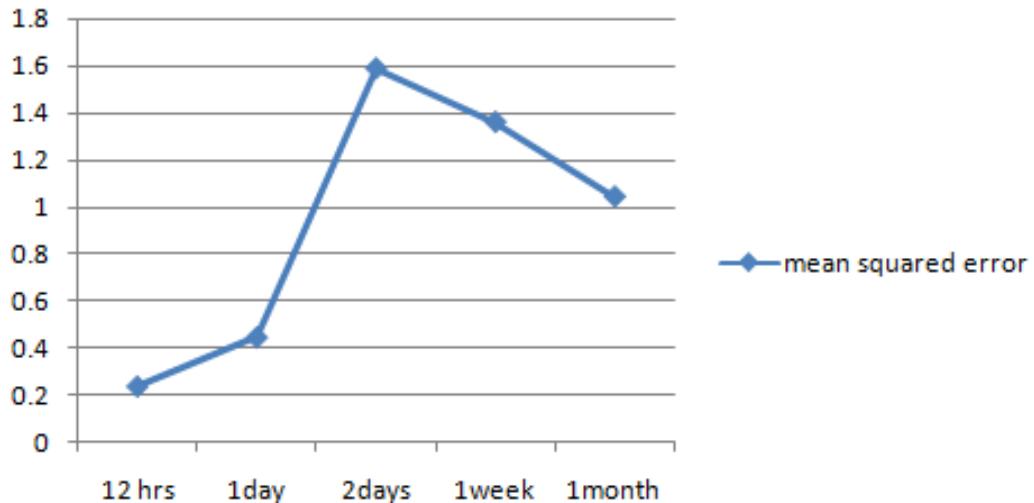


Figure 11.8: Mean Squared Error values for LSTM Models

root mean squared error

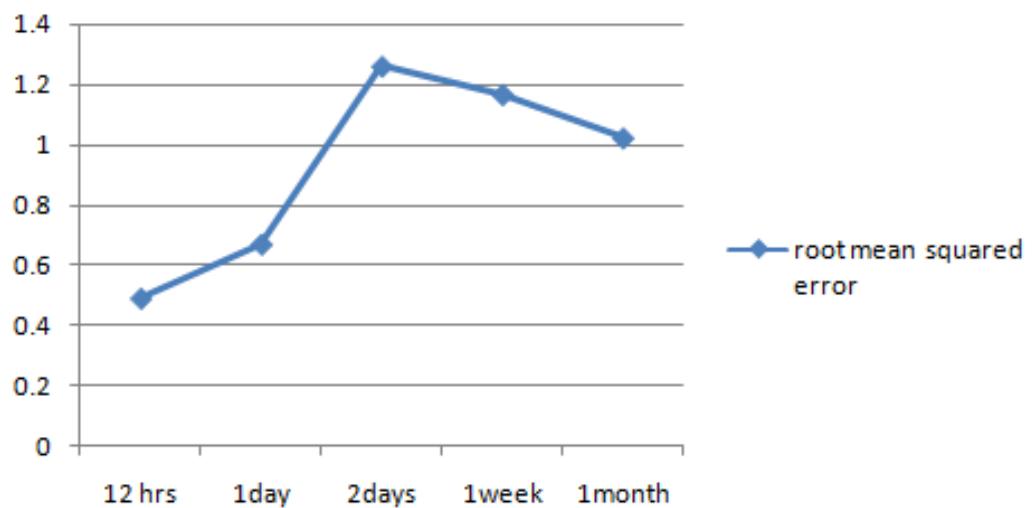


Figure 11.9: Root Mean Squared Error values for LSTM Models

Mean squared error as well as root mean squared error shows similar behavior it peaks at 2 hours ahead prediction then the value decreases at a slower rate as shown in Figure 11.8 and Figure 11.9.

mean squared log error

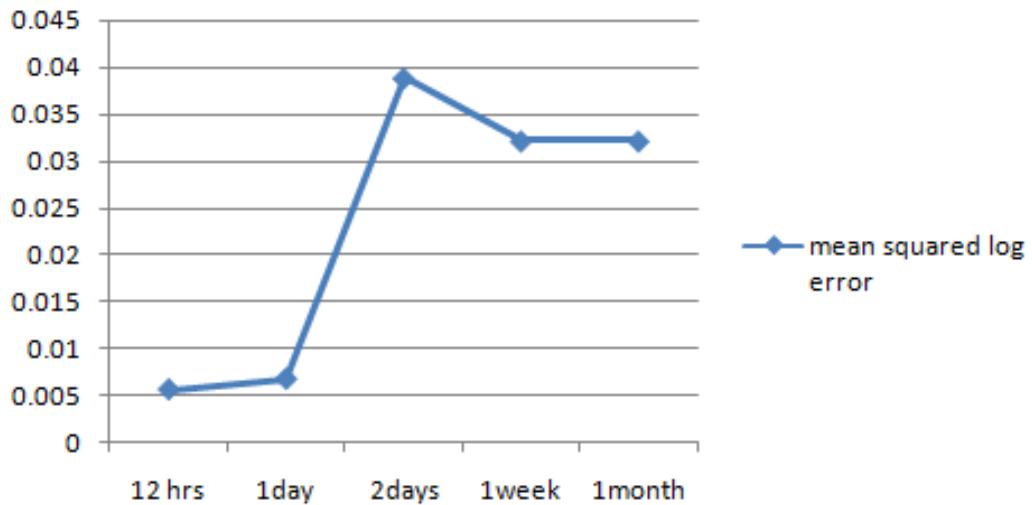


Figure 11.10: Mean Squared Log Error values for LSTM Models

median absolute error

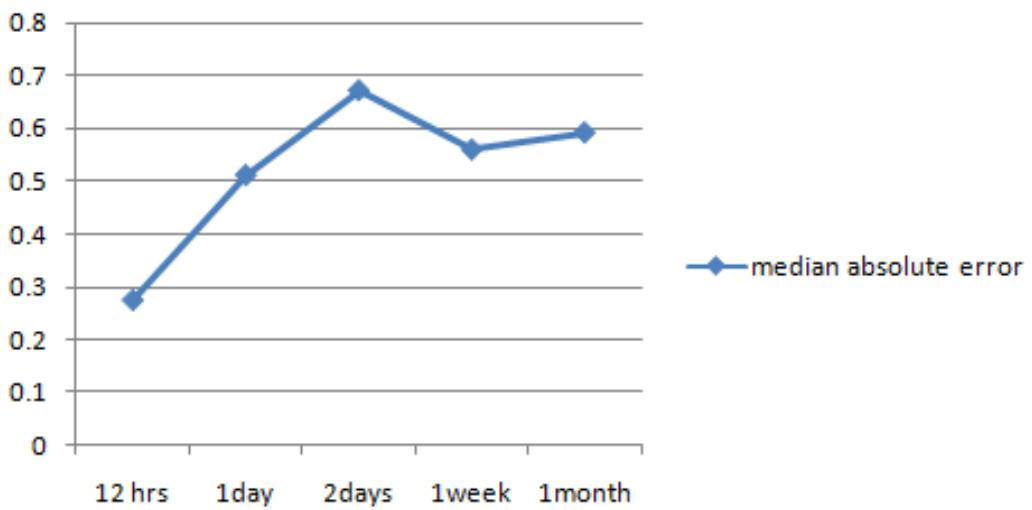


Figure 11.11: Median Absolute Error values for LSTM Models

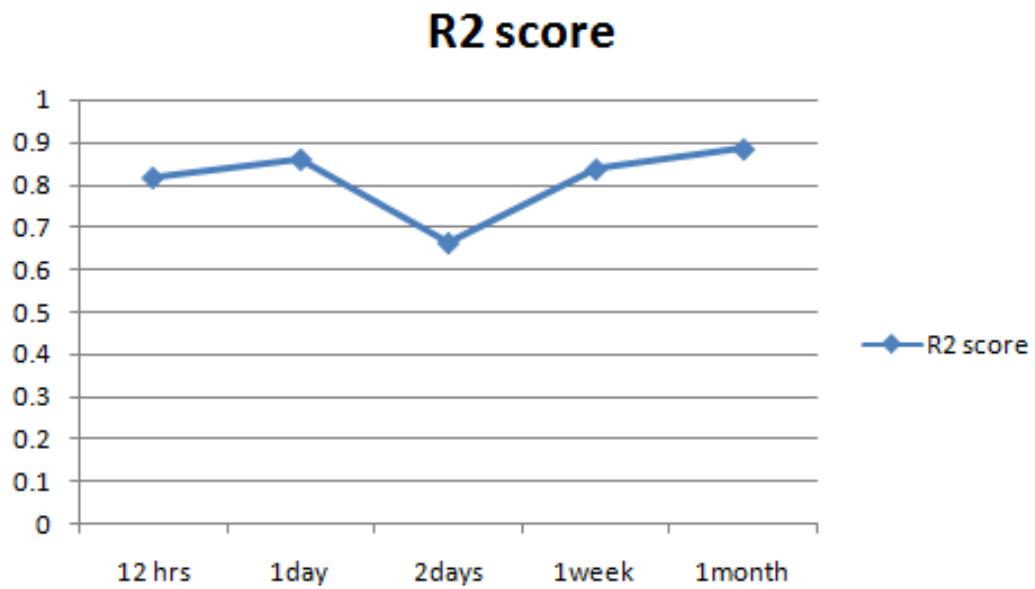


Figure 11.12: R2 score values for LSTM Models

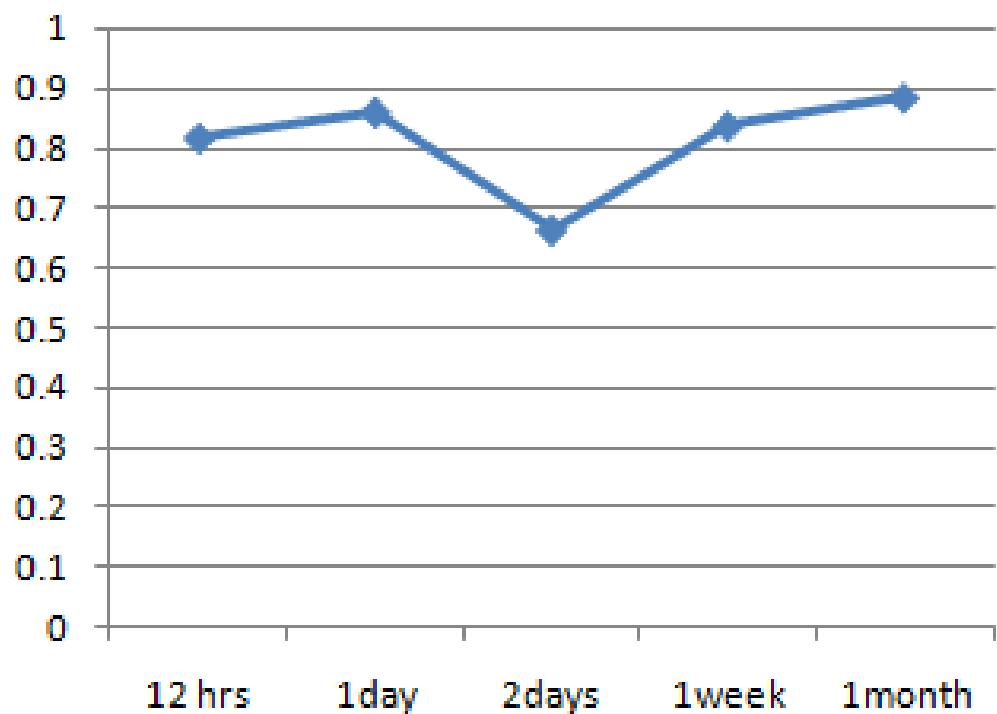


Figure 11.13: R2 score variance weighted values for LSTM Models

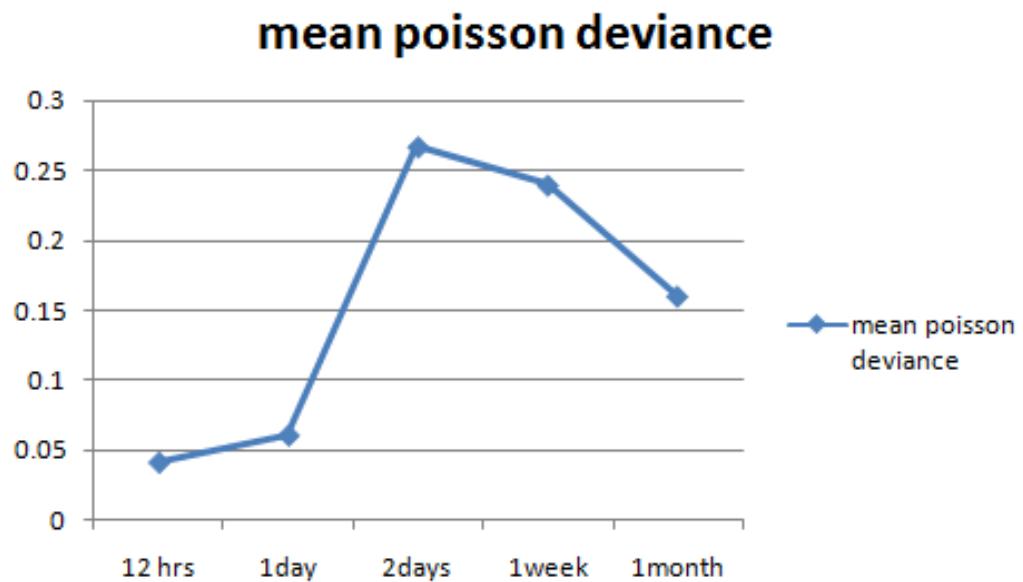


Figure 11.14: Mean Poisson values for LSTM Models

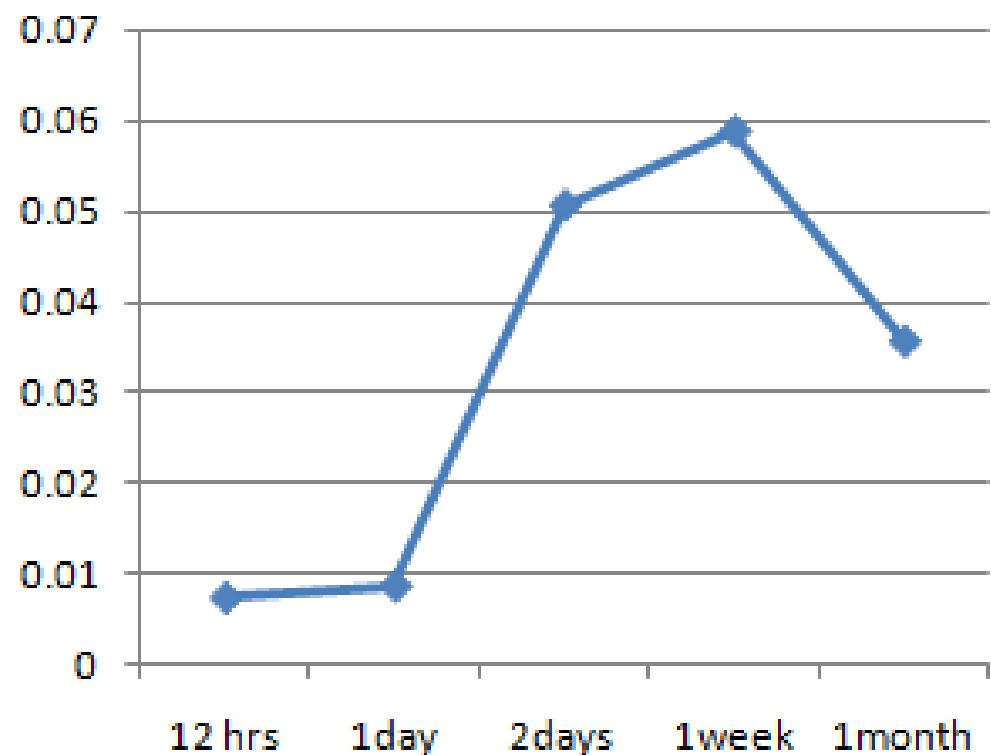


Figure 11.15: Mean Gamma values for LSTM Models

While rest of the performance measures showed, similar behavior that is the performance gets lowered at 2 days ahead prediction. Whereas for mean gamma deviance peaked at 1 week ahead prediction window then its value got lowered gradually as shown in Figure 11.15.

11.6.2 Sequence to Sequence Models

For obtaining higher performances sequence to sequence long short term neural networks needs to be employed. There are two basic classifications for this sequence to sequence model namely unidirectional and bidirectional. Its efficiency can be improved by adding more and more layers. Here single layered and double-layered models were created, thus resulting in four different models. The performance measures are compared for all these variants using plots.

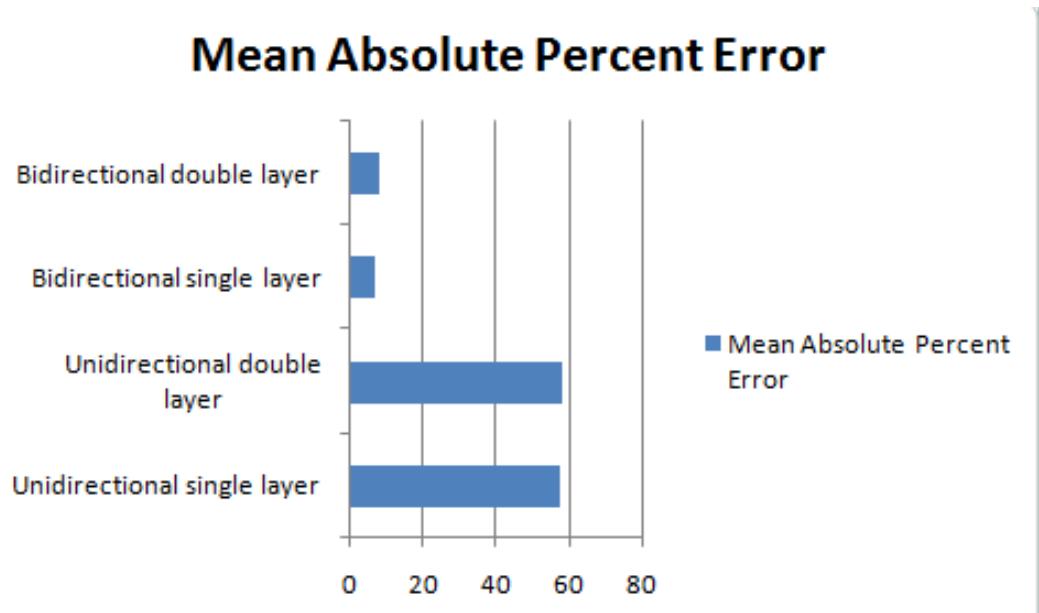


Figure 11.16: Mean Absolute Percent Error values for Sequence Models

Expected variance score

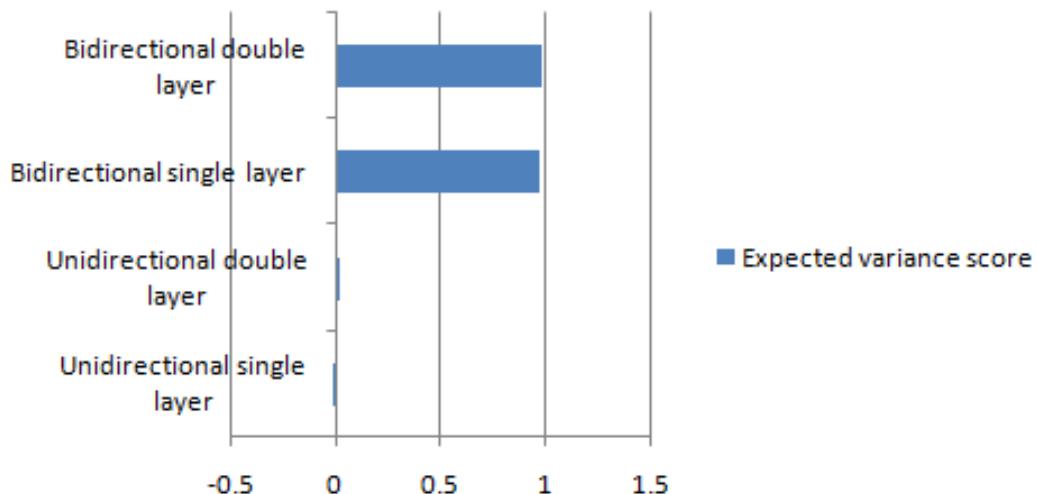


Figure 11.17: Expected Variance values for Sequence Models

Maximum error

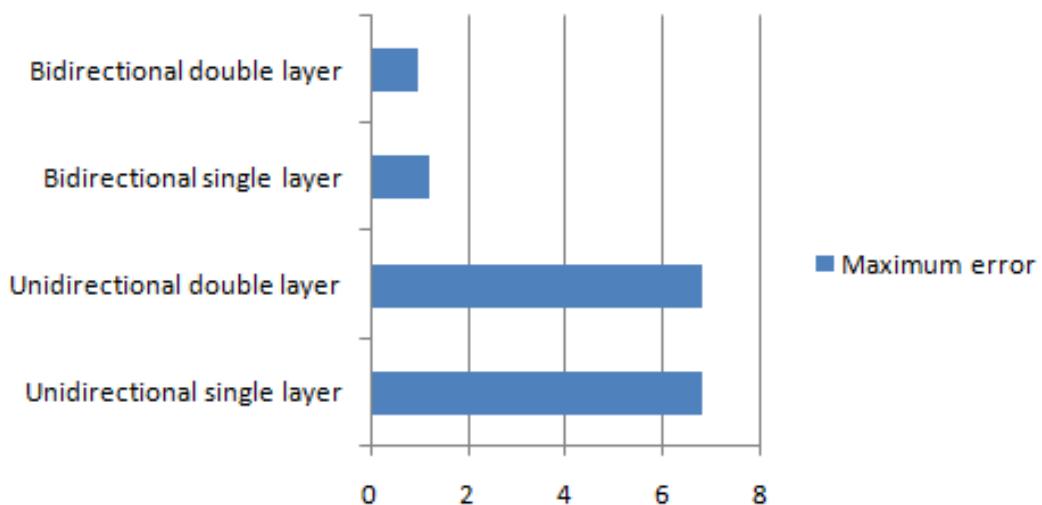


Figure 11.18: Maximum Error values for Sequence Models

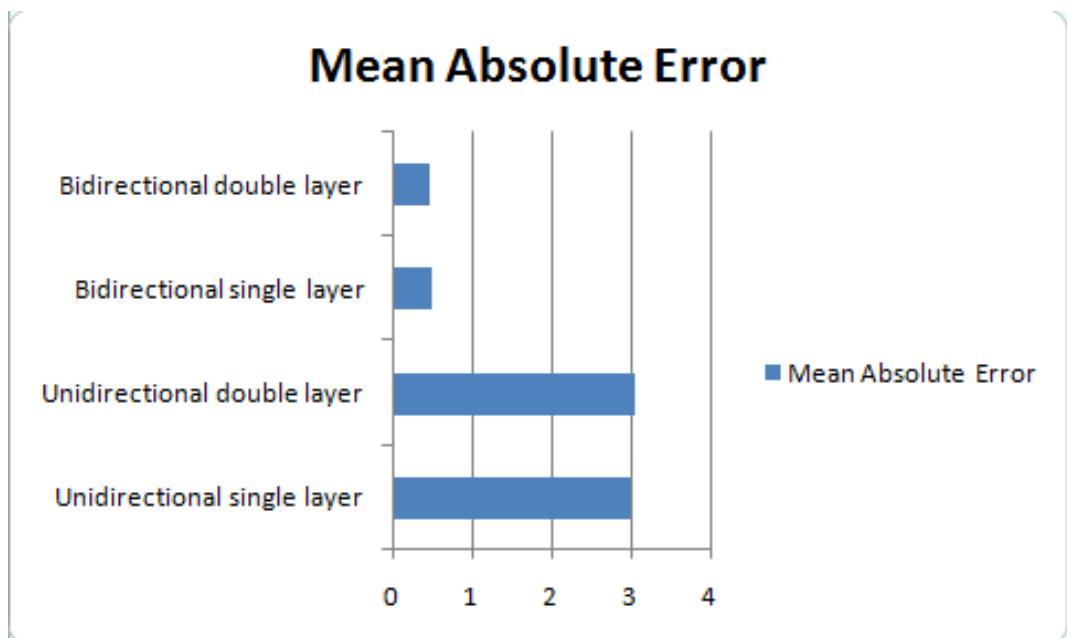


Figure 11.19: Mean Absolute Error values for Sequence Models

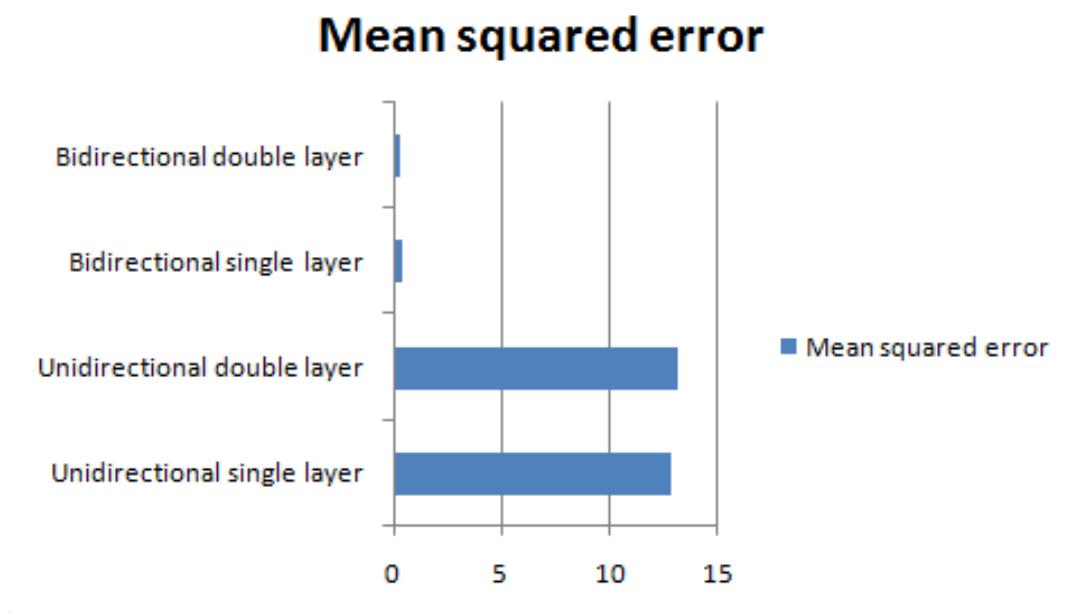


Figure 11.20: Mean Squared Error values for Sequence Models

Root mean squared error

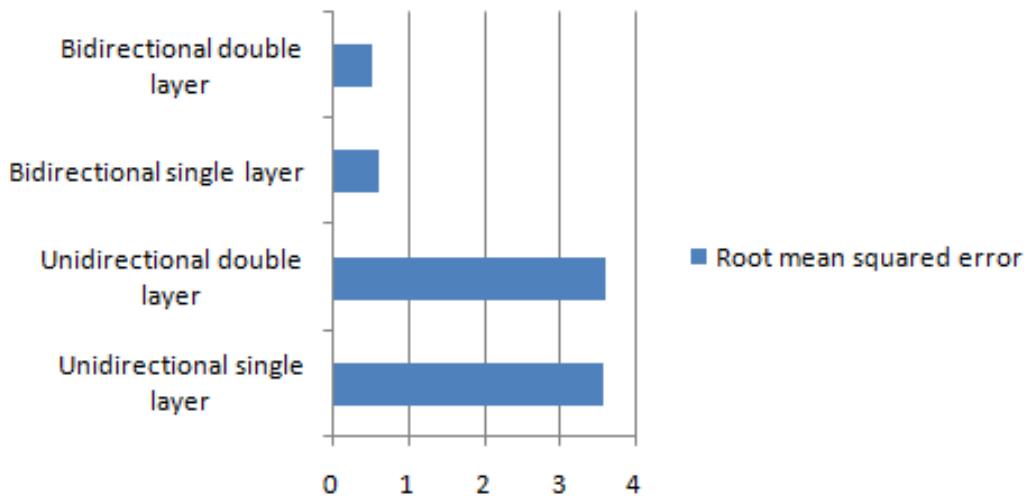


Figure 11.21: Root Mean Squared Error values for Sequence Models

Mean squared log error

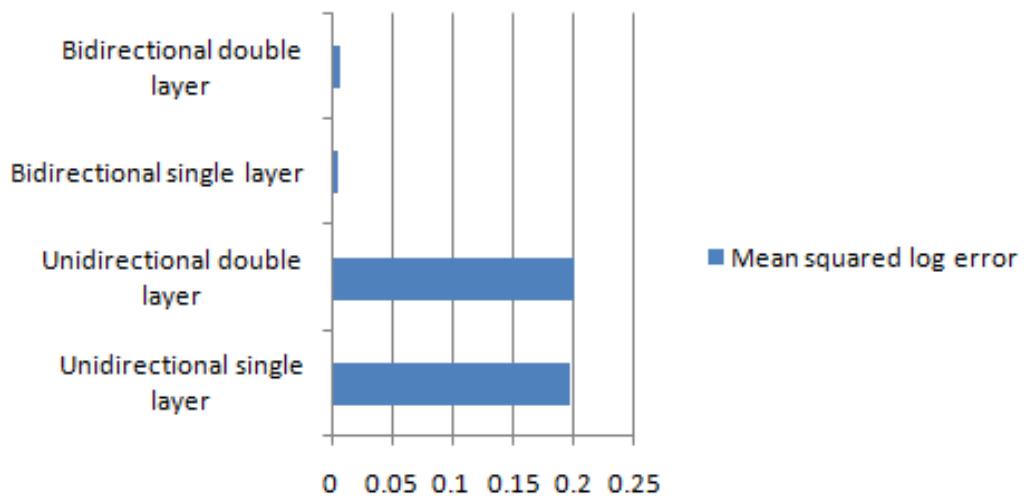


Figure 11.22: Mean Squared Log Error values for Sequence Models

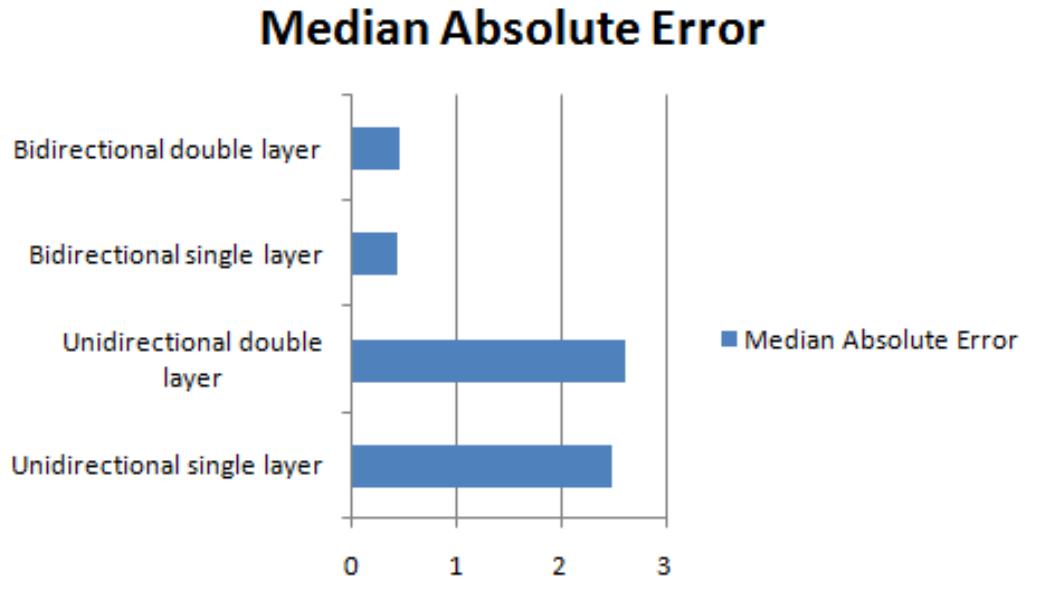


Figure 11.23: Median Absolute Error values for Sequence Models

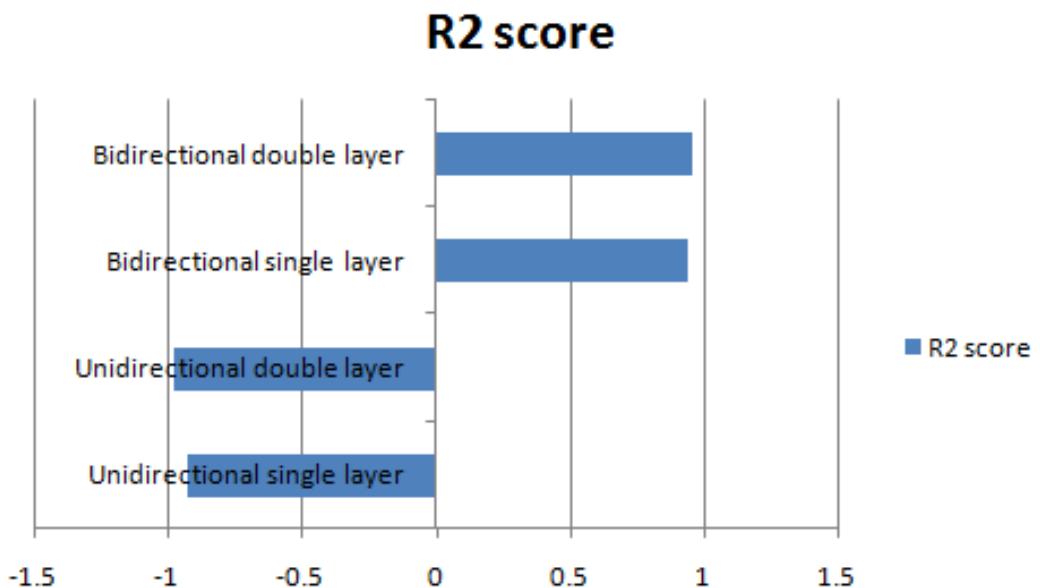


Figure 11.24: R2 score values for Sequence Models

R2 score variance weighted

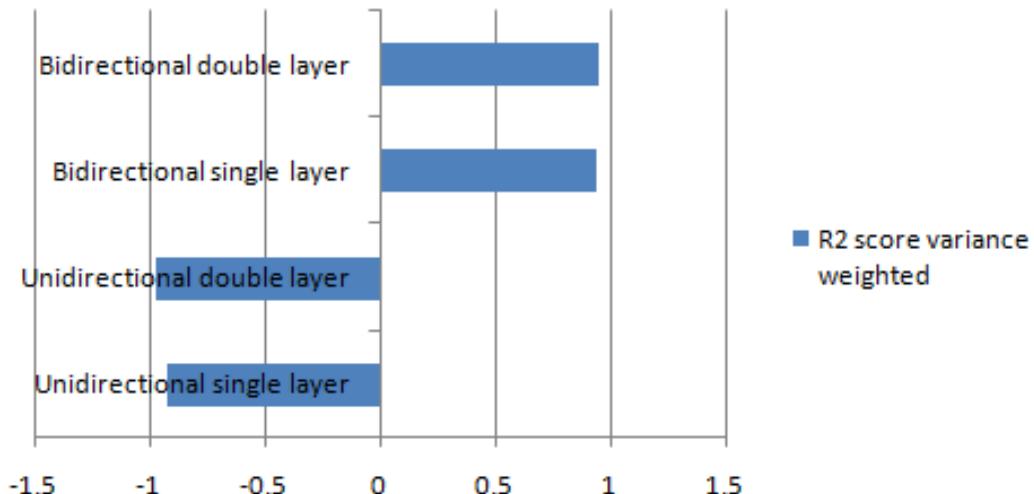


Figure 11.25: R2 score variance weighted values for Sequence Models

Mean poisson deviance

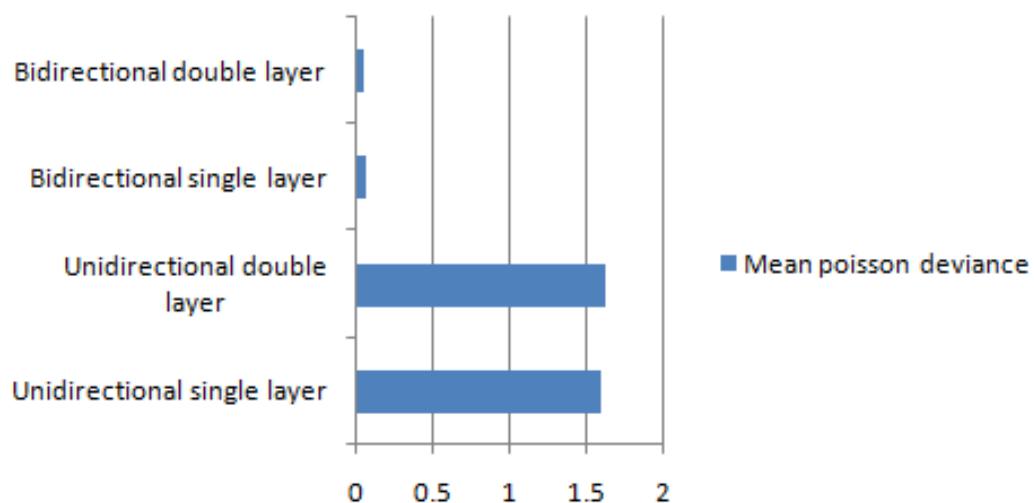


Figure 11.26: Mean Poisson values for Sequence Models

It is evident from above comparison plots that bidirectional models show higher extent of performance when compared with unidirectional models. In case of unidirectional models double layering yields better performances whereas in bidirectional models single layered

ones dominated double layered ones with a small margin. This contrasting observation is due to overfitting. Double layered Bidirectional model exhibits a global trend than the high variance single layered counterpart.

11.7 POWER PREDICTION USING STACKED MODEL

To get a definite intuition regarding the stacked power prediction model, the models error measures were compared with varying test train split ratios. In effect the data used for training changes. One year * sixty hour data was used for this analysis. Test train split was performed using 0.1, 0.2 and 0.3 ratios respectively. Following plots represents the variation in performance measures with respect to the variations in test train split ratios. Here the x axis represents the test train split ratios.

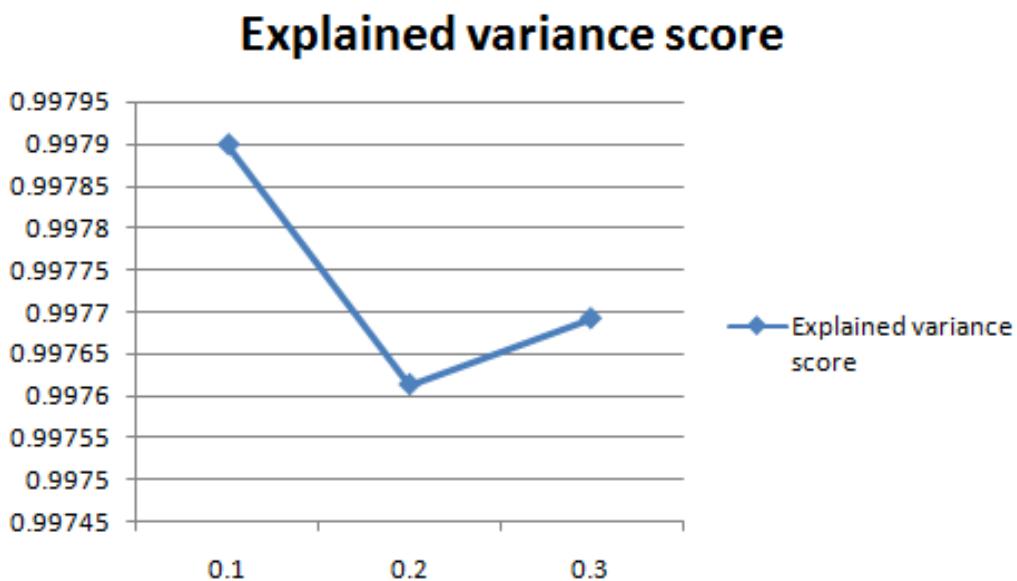


Figure 11.27: Expected Variance values for Stacked Models



Figure 11.28: Maximum Error values for Stacked Models

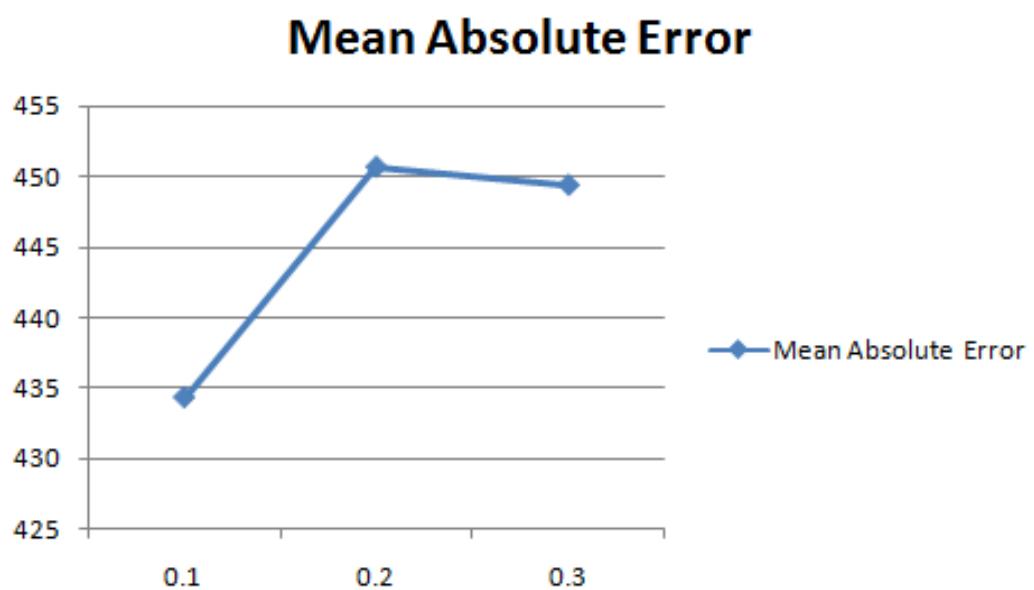


Figure 11.29: Mean Absolute Error values for Stacked Models

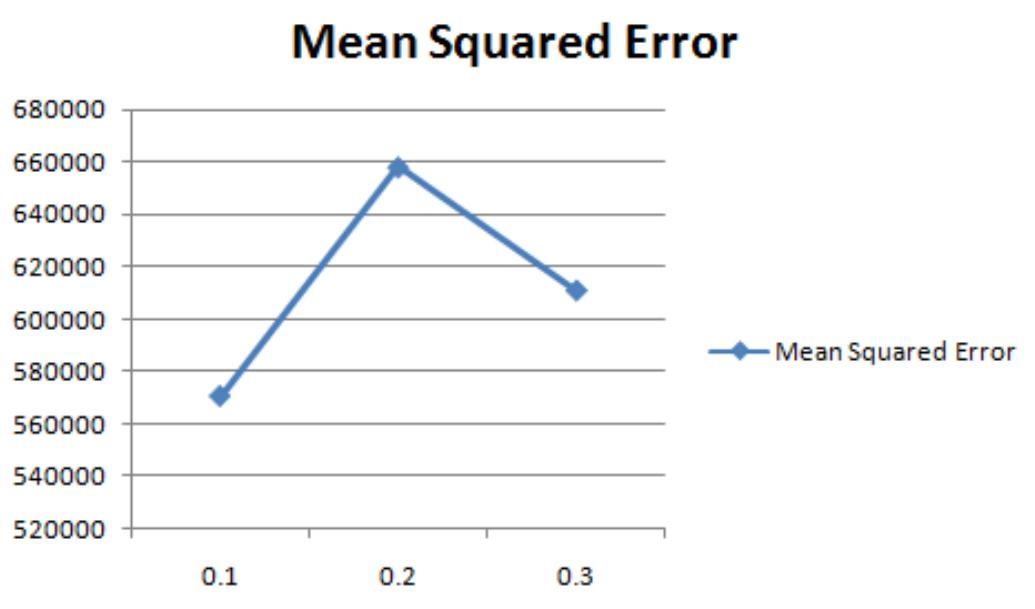


Figure 11.30: Mean Squared Error values for Stacked Models

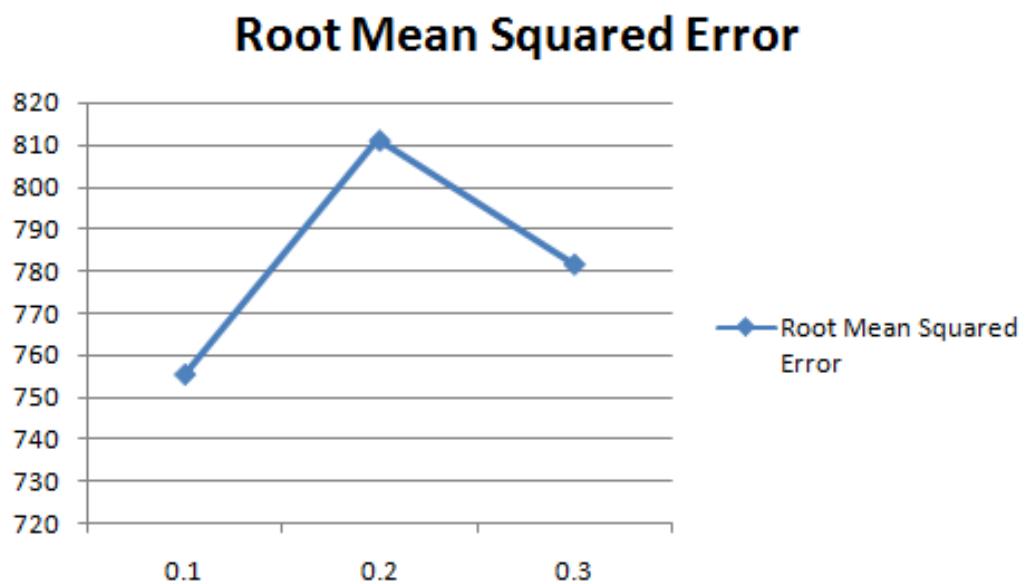


Figure 11.31: Root Mean Squared Error values for Stacked Models

Mean Squared Log Error

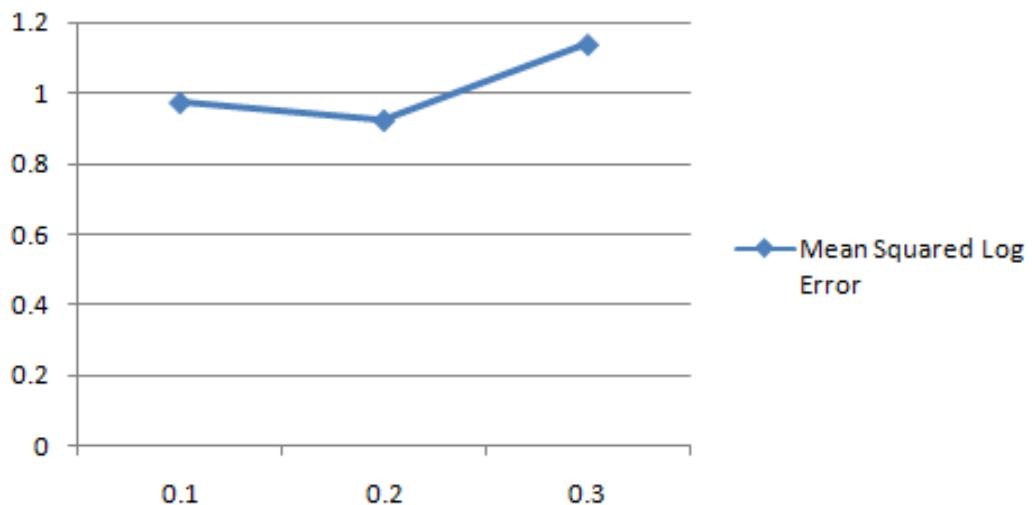


Figure 11.32: Mean Squared Log Error values for Stacked Models

Median Absolute Error

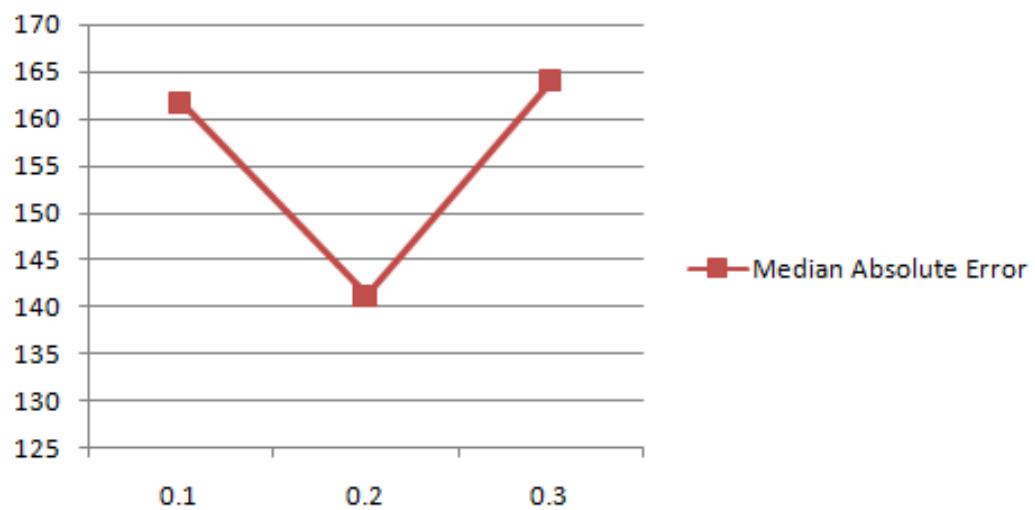


Figure 11.33: Median Absolute Error values for Stacked Models

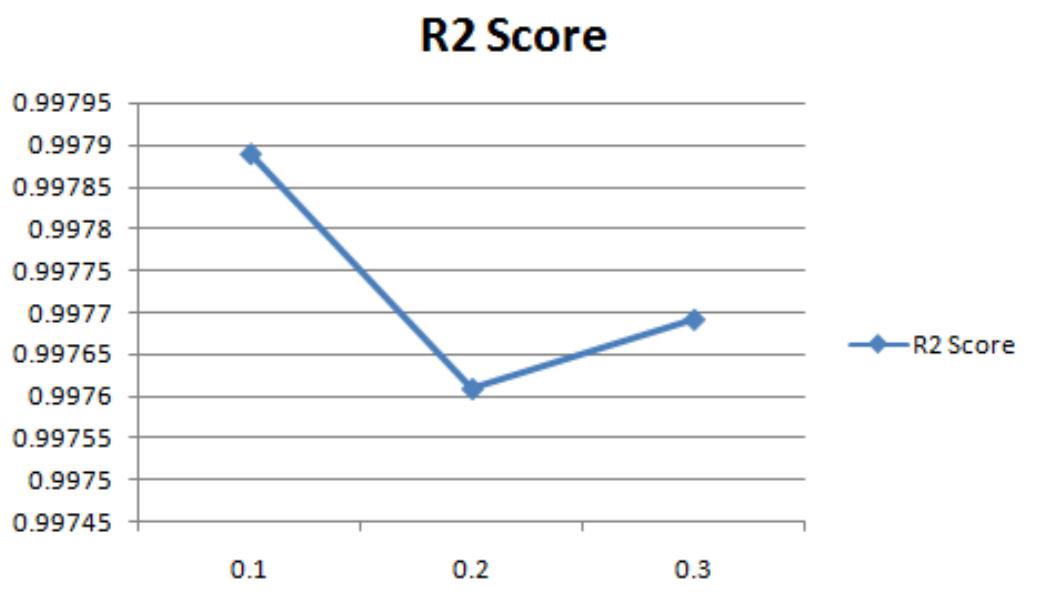


Figure 11.34: R2 score values for Stacked Models

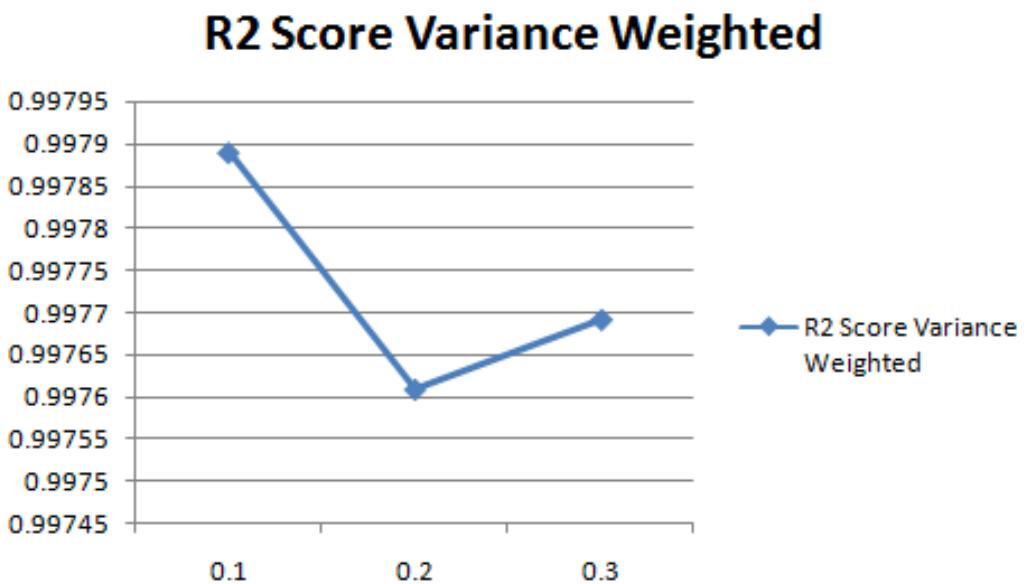


Figure 11.35: R2 score variance weighted values for Stacked Models

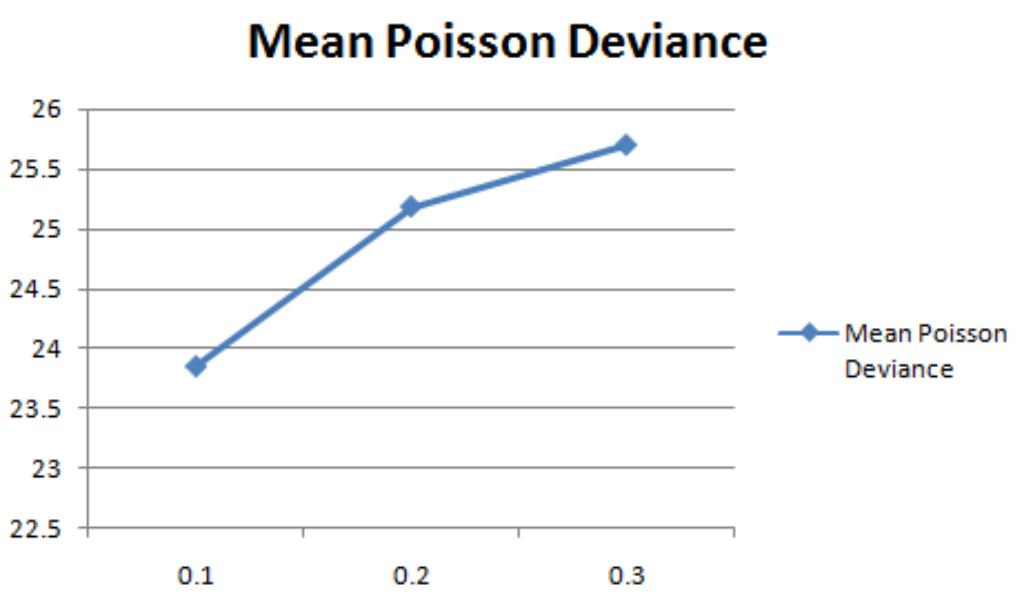


Figure 11.36: Mean Poisson values for Stacked Models

From above plots, it's evident that as the test train split ratio decreases (which in turn increases the data volume used for model training), the performance of the model trained increases. Hence the whole one year data will be used for model training.

11.8 PROPOSED ALGORITHM

Here, the predicted wind speeds belonging to prediction windows namely 12 hours, 1 day, 2 days, 1 week and 1 month are provided as inputs to stacked algorithm so as to predict corresponding powers. The prediction models performance is evaluated using various performance measures and are compared with respect to varying prediction window.

11.8.1 Stacked Models with predicted wind speed from LSTM models as Input

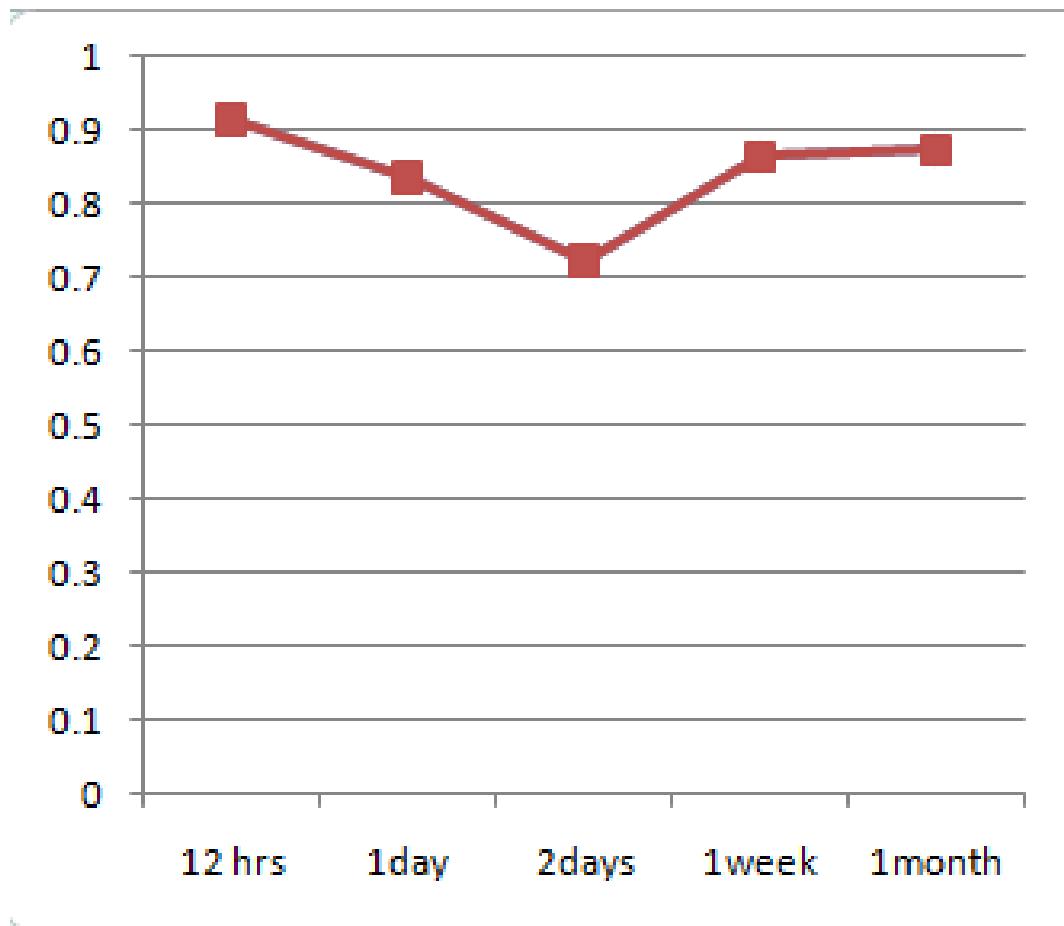


Figure 11.37: Expected Variance values for Stacked Models with predicted wind speed from LSTM models as Input

Here it is observed that the expected variance score decreases up to prediction window of 2 days then it gradually increases as shown in Figure 11.37.

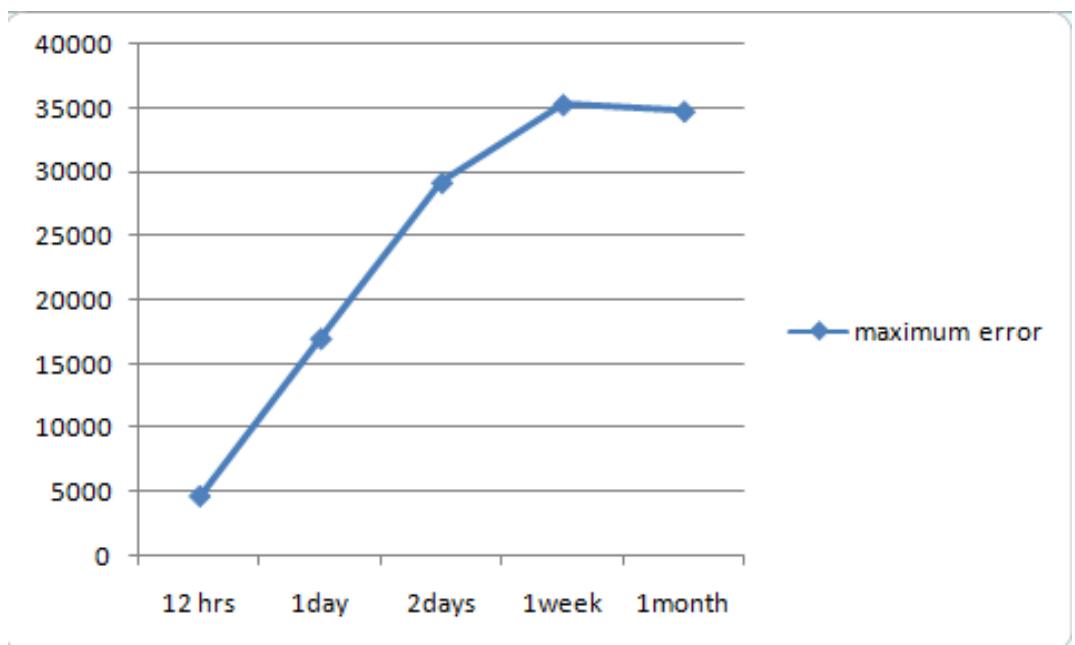


Figure 11.38: Maximum Error values for Stacked Models with predicted wind speed from LSTM models as Input

Maximum error value increases rapidly till two days prediction window then its climbs slowly and descends slowly as shown in Figure 11.38.

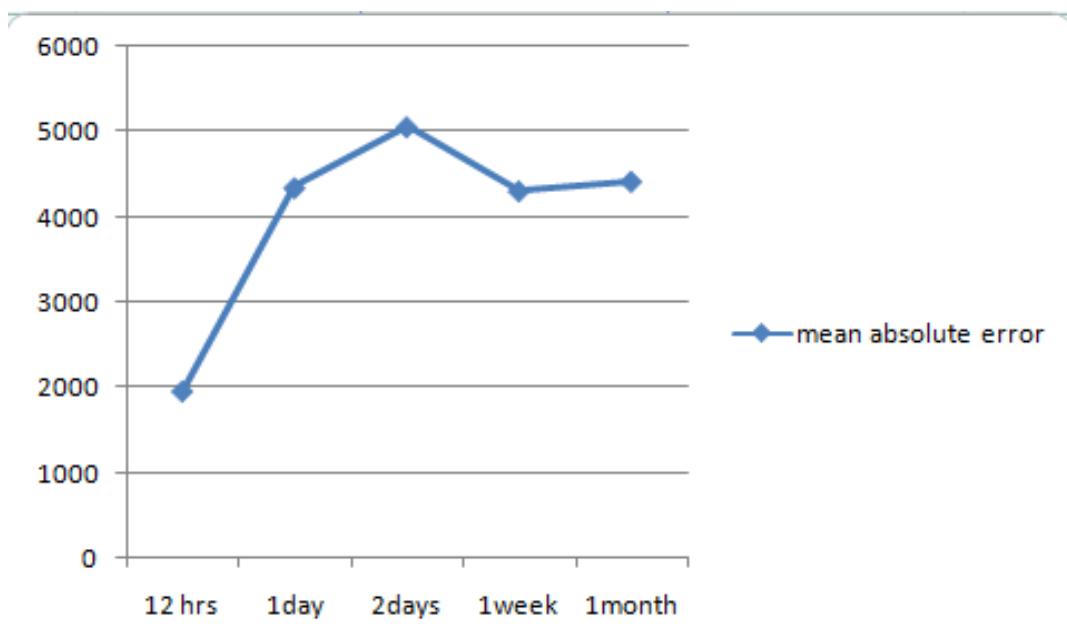


Figure 11.39: Mean Absolute Error values for Stacked Models with predicted wind speed from LSTM models as Input

Mean absolute error value attains maximum at 2 days prediction window then it descends and almost becomes constant as shown in Figure 11.39.



Figure 11.40: Mean Squared Error values for Stacked Models with predicted wind speed from LSTM models as Input

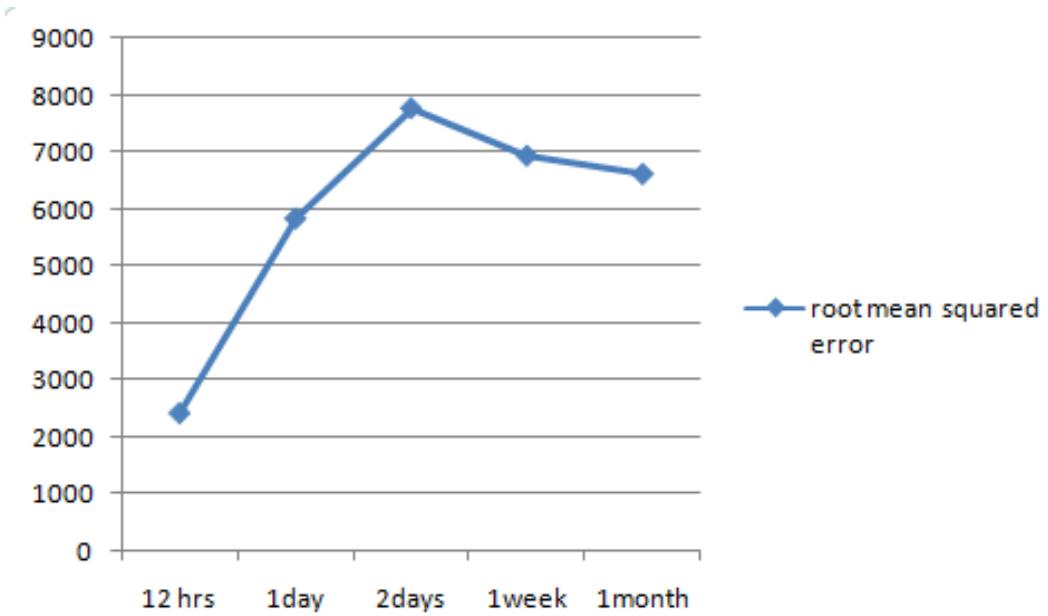


Figure 11.41: Root Mean Squared Error values for Stacked Models with predicted wind speed from LSTM models as Input

Mean squared as well as root mean squared error values shows similar behavior, attains maximum at 2 days prediction window then it descends as shown in Figure 11.40 and Figure

11.41.

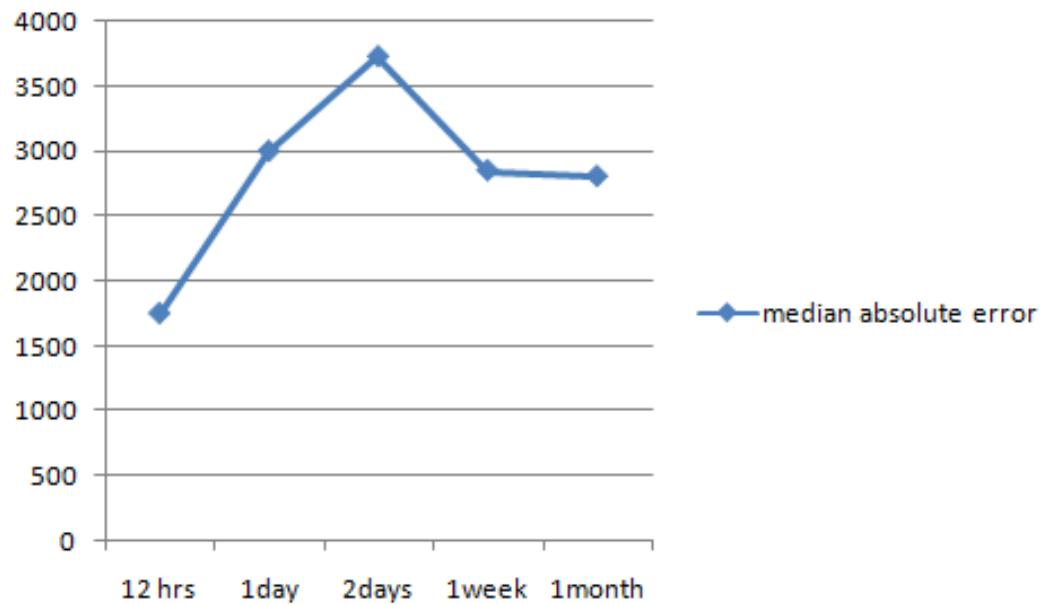


Figure 11.42: Median Absolute Error values for Stacked Models with predicted wind speed from LSTM models as Input

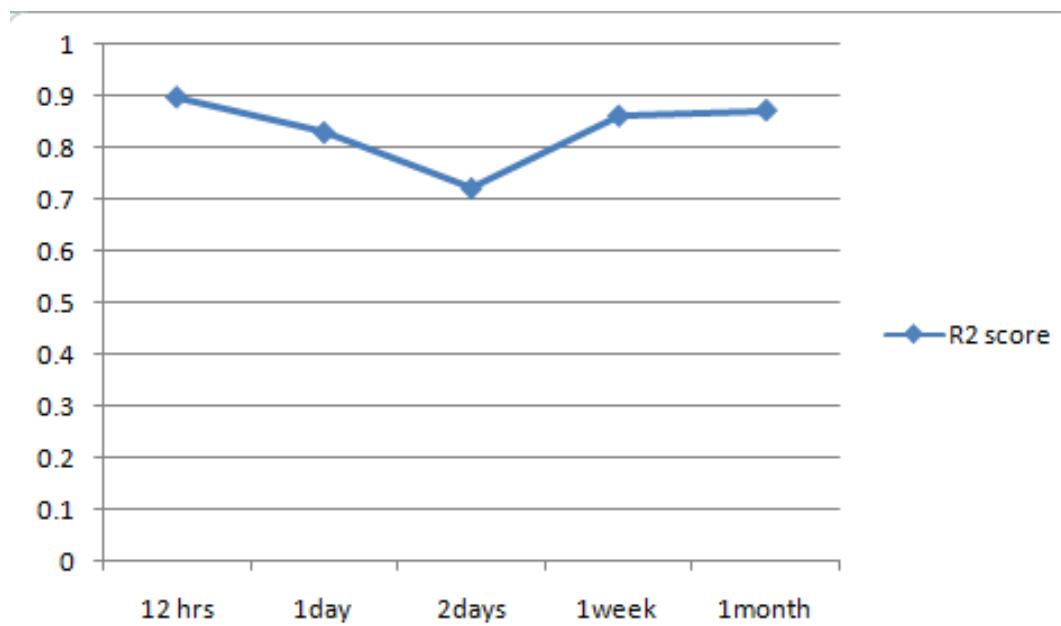


Figure 11.43: R2 score values for Stacked Models with predicted wind speed from LSTM models as Input

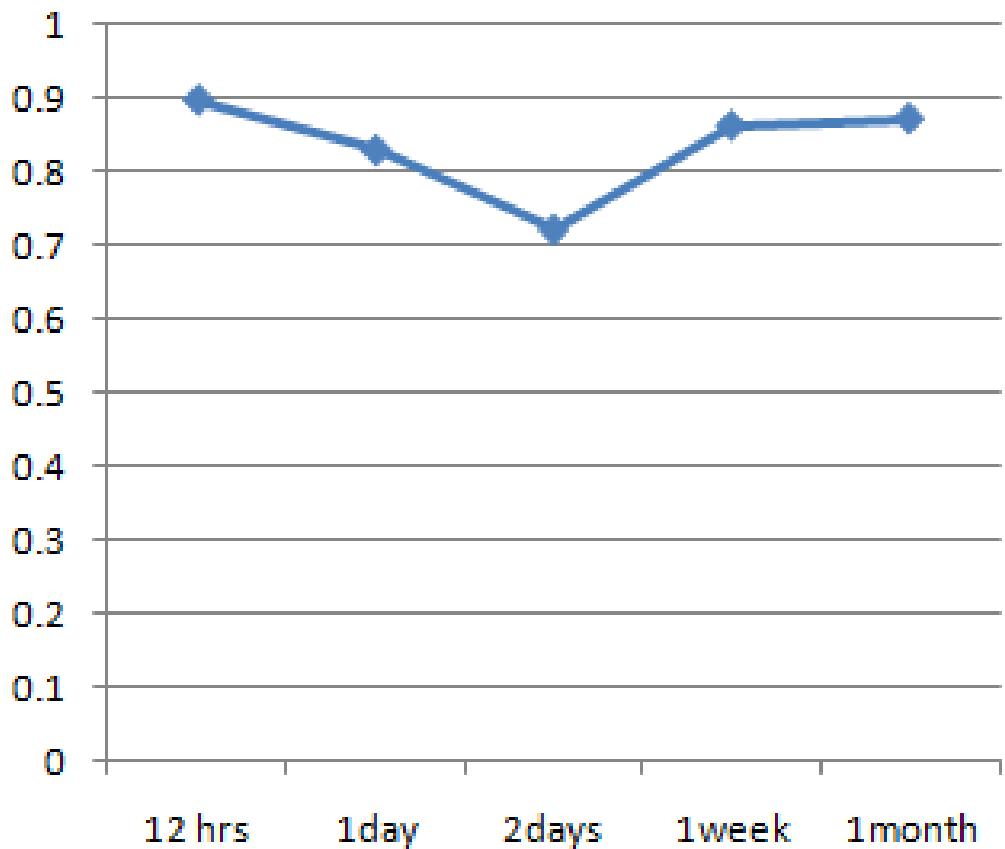


Figure 11.44: R2 score variance weighted values for Stacked Models with predicted wind speed from LSTM models as Input

Coefficient of determination is low for 2 days ahead prediction. Then it climbs up gradually as shown in Figure 11.43.

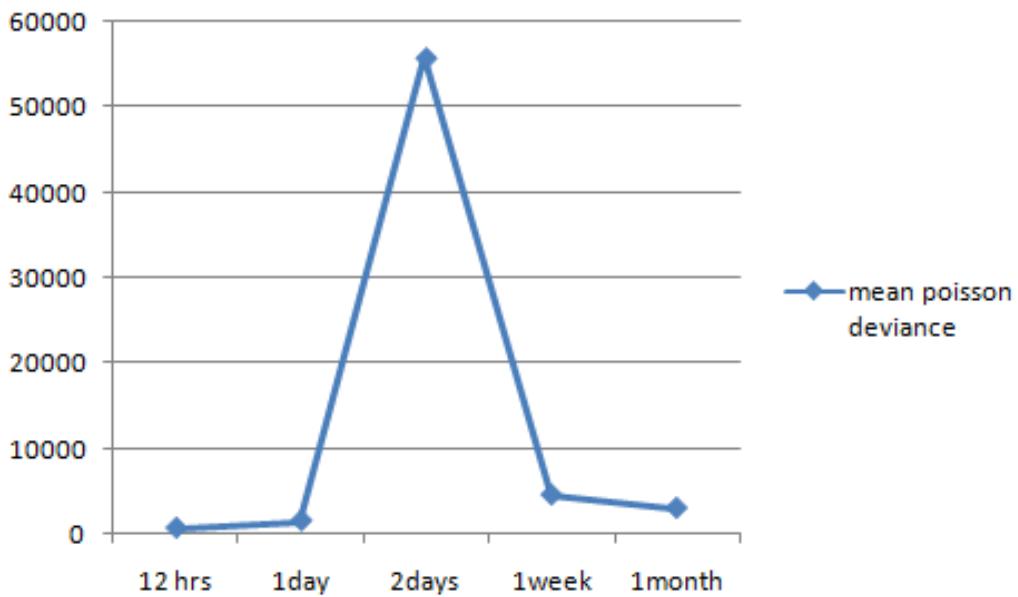


Figure 11.45: Mean Poisson values for Stacked Models with predicted wind speed from LSTM models as Input

Generally for the whole model after examining all performance measures its observed that initially the performance decreases rapidly and the worst performance is recorded at 2 days prediction window. Then its performance improves gradually but doesn't reach to its initial performance level within 1 month prediction window.

11.8.2 Stacked Models with predicted wind speed from Sequence to Sequence models as Input

Now it is important to further increase the performance of the model hence predictions from sequence to sequence model is provided as input to the stacked model to predict power. Now models are compared with respect to each performance measures.

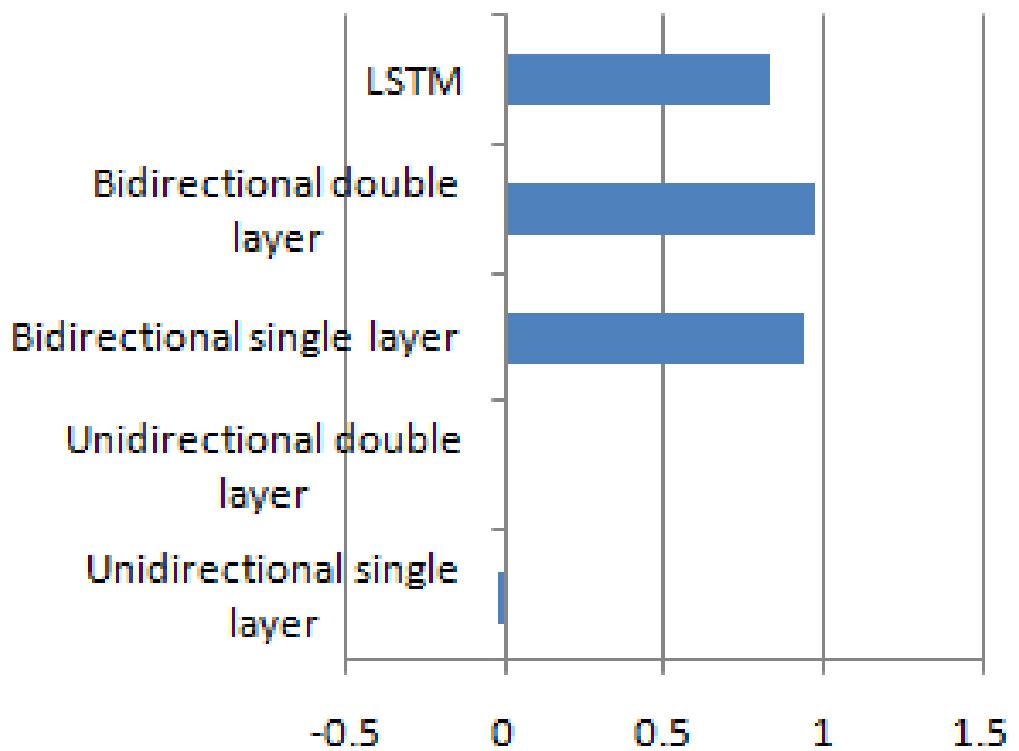


Figure 11.46: Expected Variance values for Stacked Models with predicted wind speed from Sequence to Sequence models as Input

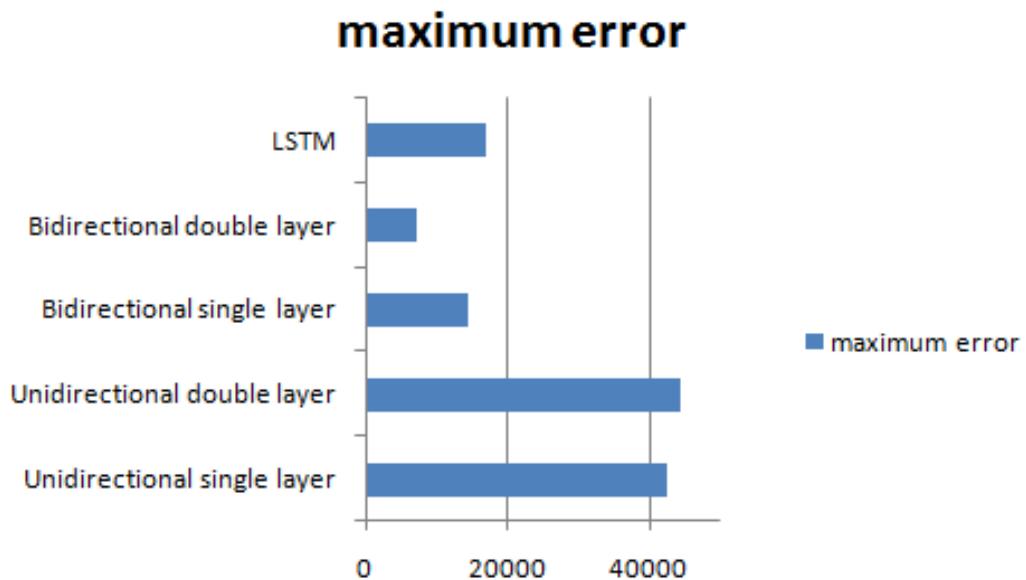


Figure 11.47: Maximum Error values for Stacked Models with predicted wind speed from Sequence to Sequence models as Input

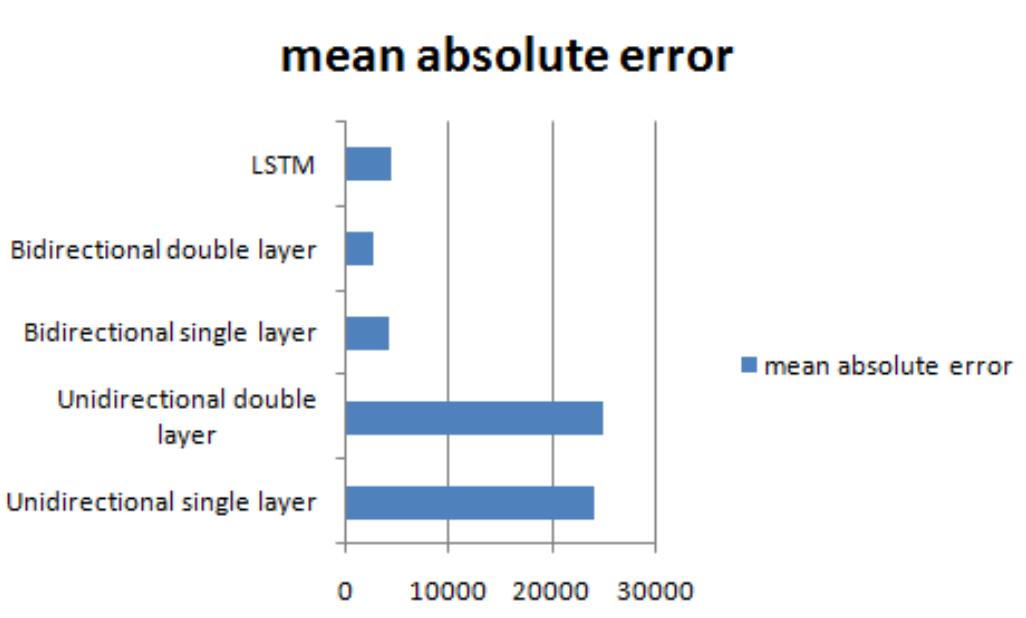


Figure 11.48: Mean Absolute Error values for Stacked Models with predicted wind speed from Sequence to Sequence models as Input

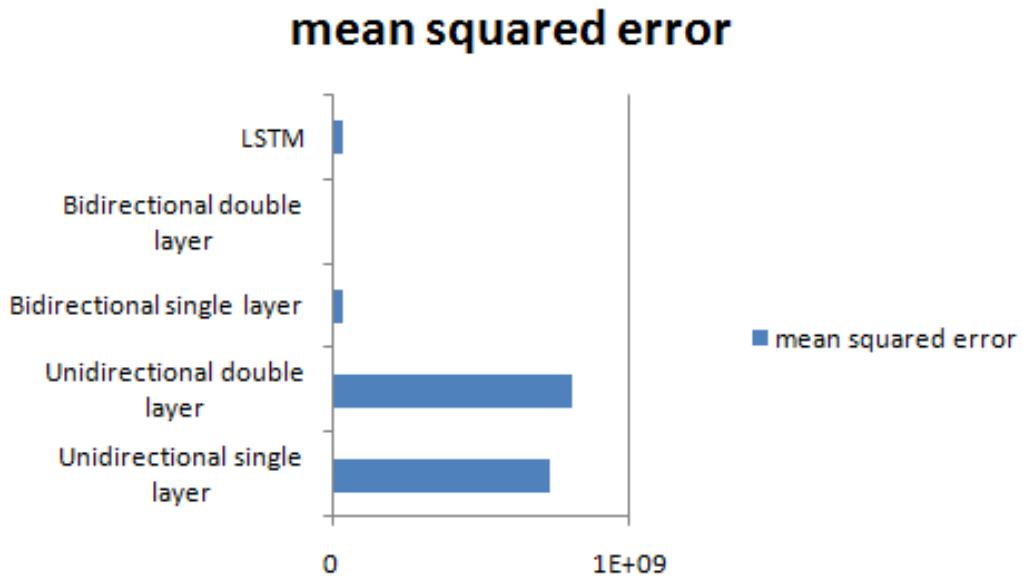


Figure 11.49: Mean Squared Error values for Stacked Models with predicted wind speed from Sequence to Sequence models as Input

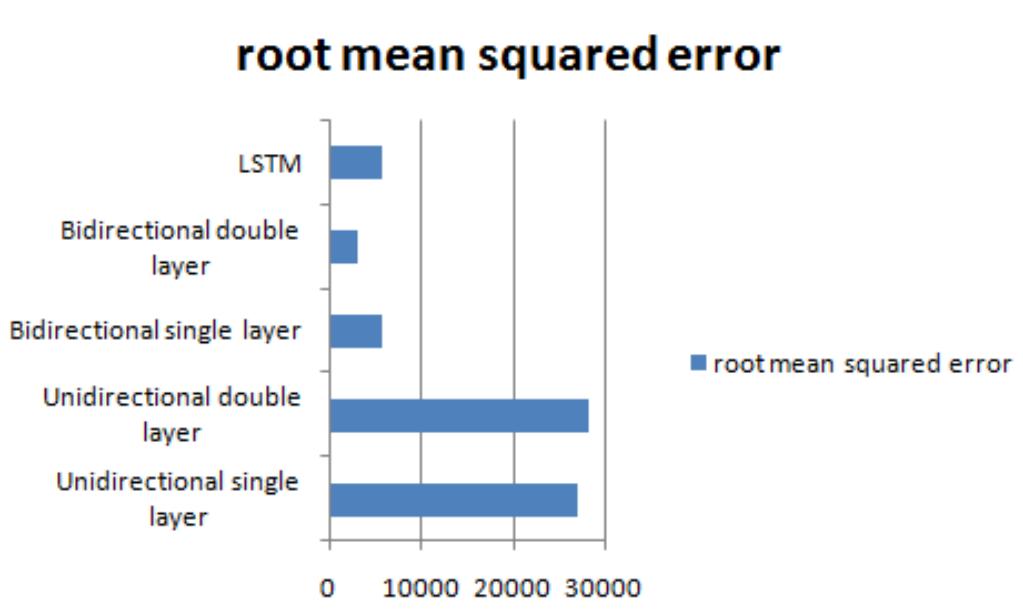


Figure 11.50: Root Mean Squared Error values for Stacked Models with predicted wind speed from Sequence to Sequence models as Input

mean squared log error

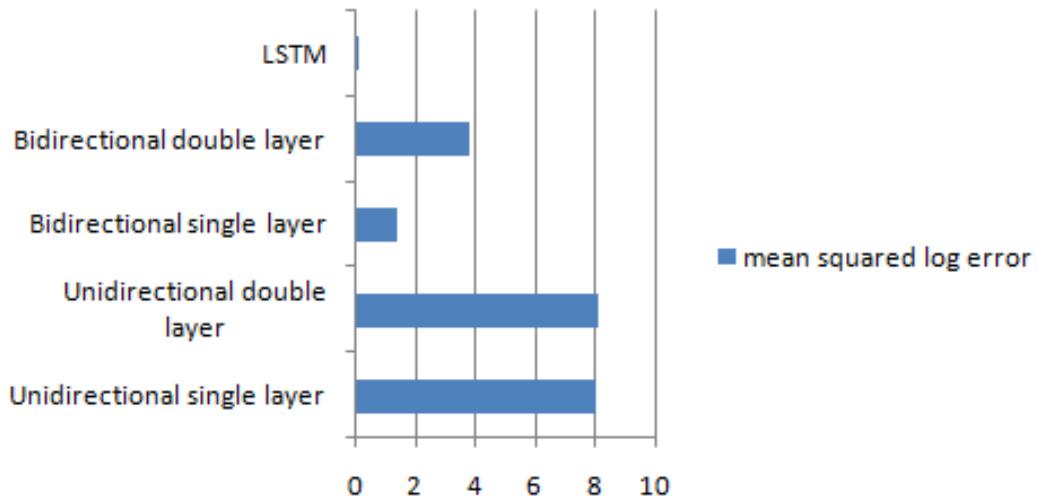


Figure 11.51: Mean Squared Log Error values for Stacked Models with predicted wind speed from Sequence to Sequence models as Input

median absolute error

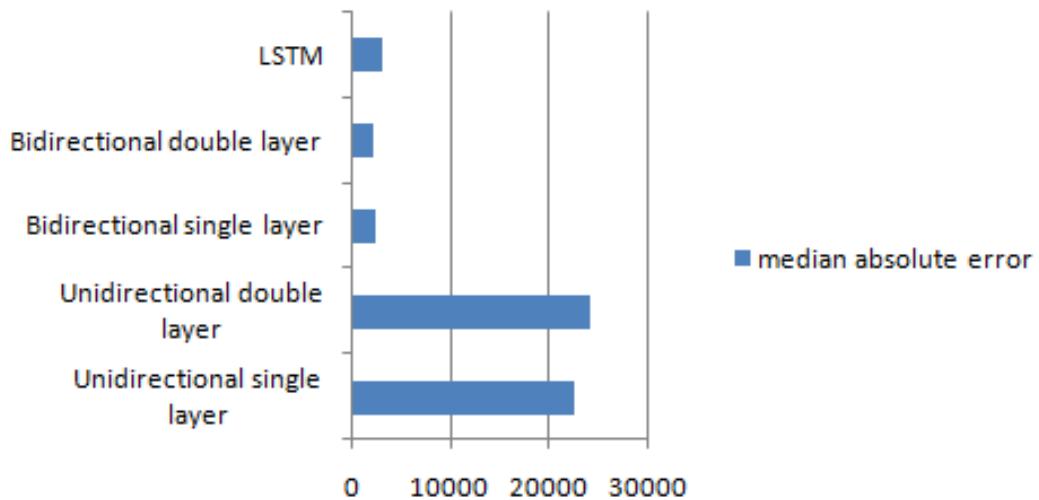


Figure 11.52: Median Absolute Error values for Stacked Models with predicted wind speed from Sequence to Sequence models as Input

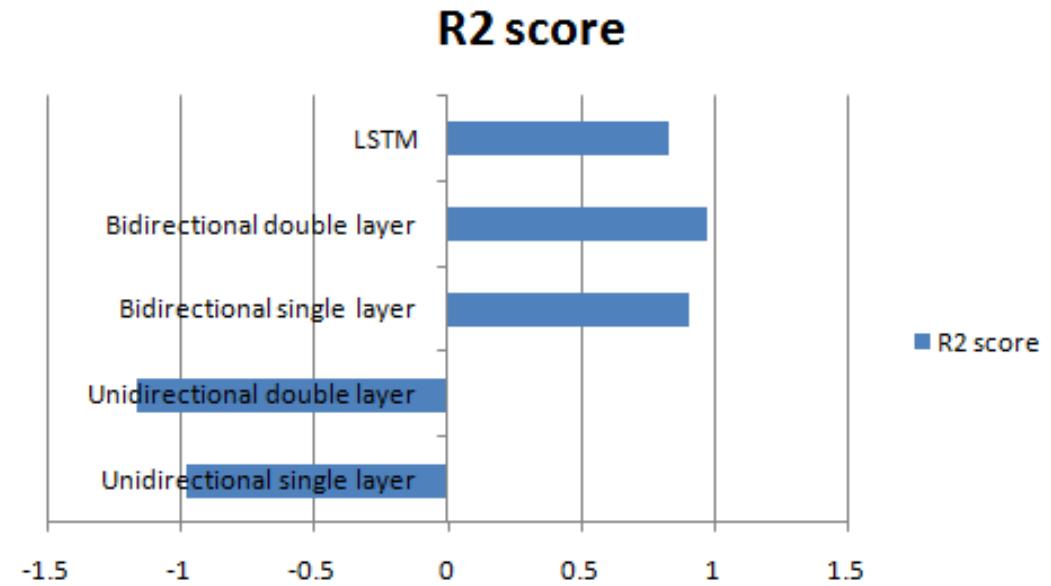


Figure 11.53: R2 score values for Stacked Models with predicted wind speed from Sequence to Sequence models as Input

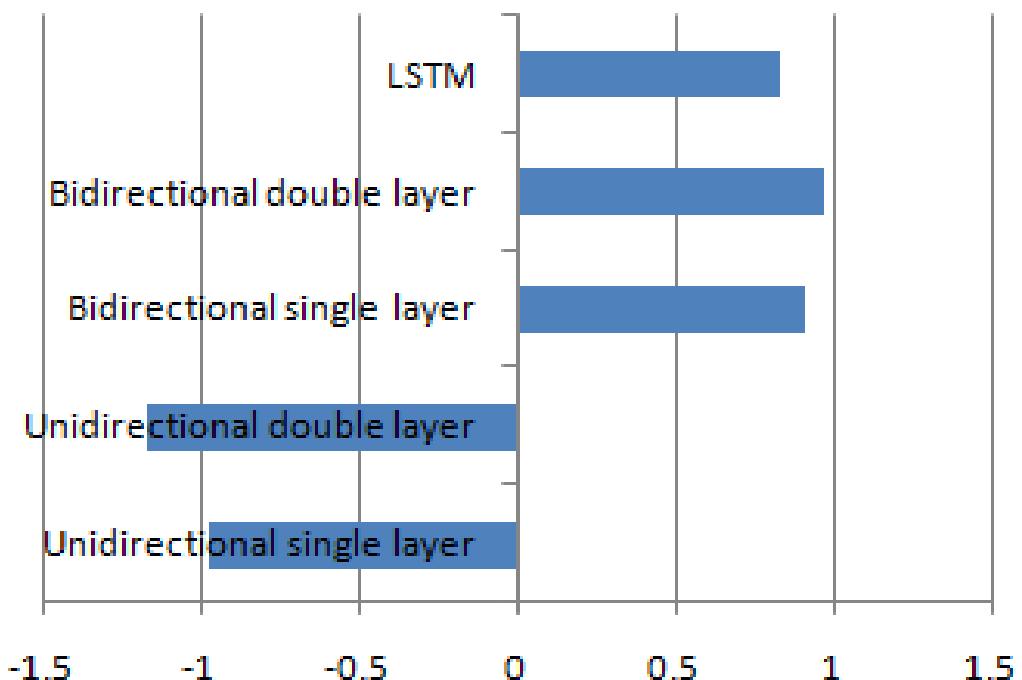


Figure 11.54: R2 score variance weighted values for Stacked Models with predicted wind speed from Sequence to Sequence models as Input

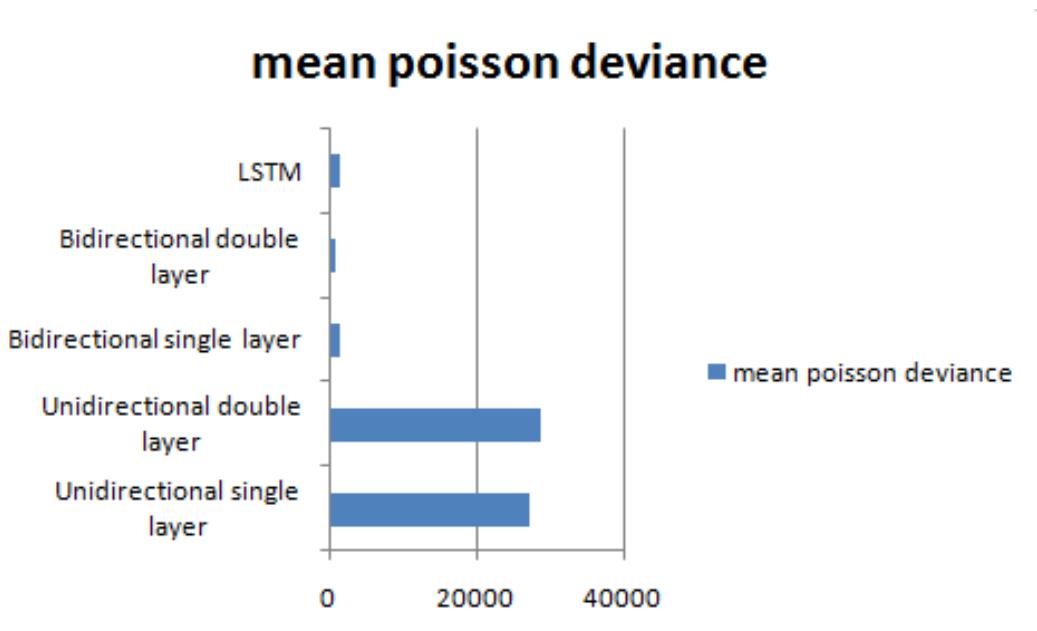


Figure 11.55: Mean Poisson values for Stacked Models with predicted wind speed from Sequence to Sequence models as Input

From the above bar charts featuring performance measures we can infer conclusively that the power prediction model with bidirectional double layer sequence to sequence long short term neural network wind prediction algorithm has relatively higher performance.

The performance of power prediction models with unidirectional sequence to sequence long short term neural network wind prediction algorithm are relatively way lower than rest of the algorithms.

Chapter 12

Conclusion and Scope of Future Works

The work primarily aims to find an algorithm with high performance so as to predict generated wind power from past wind speed data. From the analysis a conclusive evidence was found for the relationship between wind speed and wind power. Hence future wind speeds needs to be predicted. So as to predict wind speed a proper investigation needs to be performed to identify the nature of data. Series of tests were performed and at the end it was conclusively determined that the data shows chaotic nature. Chaotic nature is generally deterministic but complex relations exists. Thus throughout the work importance was given for increasing the performance of the wind speed prediction models for which comparison studies were done by varying algorithm and its functionalities, comparisons were done by varying prediction windows. For power prediction past literature suggests use of individual algorithms. In this work best performing algorithms were selected and stacked together so as to obtain better accuracy than its parent individual algorithms. In the proposed algorithms the predicted wind speed was given as input to stacked model so as to predict future power. Comparison studies were done with respect to performance measures and prediction window. It was observed that stacked power prediction model with bi directional sequence to sequence double layer long short term neural network delivers the best performance.

Throughout the work more importance was resorted in increasing the performance of the model. So as to increase performance more powerful algorithms should be used in future for wind prediction and power prediction. Echo state networks algorithm could be used for wind

speed prediction. From observations it could be inferred that the long short term memory neural networks and two variants of bi directional sequence to sequence models showed higher levels of accuracy. A stacked form of these algorithm may even yield better results. A highly complex stacked model could be made comprising of echo state networks.

The work was successful in using newly developed algorithm in the prediction of future power that could be generated at a location. The newly developed algorithm shows high levels of performance. The main advantage of this algorithm is that there is no need for future meteorological forecasts of wind. This algorithm could be used to make decisions regarding installation of a particular wind mill in an unknown geographical location. By minimizing the meteorological forecasts its initial pre-installations costs will be reduced. More potential zones for installations could be found without much expenses. Hence step by step increments in dependency towards renewable energy resources could be accelerated.

References

- [1] H.D.I. Abarbanel, *Analysis of Observed Chaotic Data* : Springer-Verlag, 1996.
- [2] J. Bergstra, Y. Bengio, "Random Search for Hyper-Parameter Optimization", *The Journal of Machine Learning Research*, vol. 13, no. 1, pp. 281-305, 2012.
- [3] B.P. Bezruchko, D.A. Smirnov, *Extracting Knowledge From Time Series: An Introduction to Nonlinear Empirical Modeling* : Springer, 2010, pp. 55-56.
- [4] G. Boeing, "Visual Analysis of Nonlinear Dynamical Systems: Chaos, Fractals, Self-Similarity and the Limits of Prediction.", *Water, Air, and Soil Pollution*, 161, pp. 121-128, 2005.
- [5] W. Caesarendra, P. Kosasih, K. Tieu, C. Moodie,"An Application of Nonlinear Feature Extraction - A Case Study for Low Speed Slewing Bearing Condition Monitoring and Prognosis", *IEEE/ASME International Conference on Advanced Intelligent Mechatronics: Mechatronics for Human Wellbeing*, pp. 1713-1718, 2013.
- [6] L. Cao, "Practical Method for Determining the Minimum Embedding Dimension of a Scalar Time Series", *Physica D: Nonlinear Phenomena*, 110(1), pp. 43-50, 1997.
- [7] G.C Cawley, N.L. Talbot, "On Over-Fitting in Model Selection and Subsequent Selection Bias in Performance Evaluation", *The Journal of Machine Learning Research*, 11, pp.2079-2107, 2010.
- [8] A. Chelani, R. Singh, S. Devotta, "Nonlinear Dynamical Characterization and Prediction of Ambient Nitrogen Dioxide Concentration", *Systems*, 4 (4), p. 37, 2016.

- [9] T. Chen, C. Guestrin, "XGBoost: A Scalable Tree Boosting System", *The 22nd ACM SIGKDD International Conference*, 2016.
- [10] Y. Cheng, C. Xu, D. Mashima, V.L. L. Thing, Y. Wu, "PowerLSTM: Power Demand Forecasting Using Long Short-Term Memory Neural Network", *ADMA*, 2017.
- [11] B. Clarke, " Comparing Bayes Model Averaging and Stacking when Model Approximation Error Cannot be Ignored", *The Journal of Machine Learning Research*, 4, pp. 683-712, 2003.
- [12] P.S.P. Cowpertwait, A.V. Metcalfe, *Introductory Time Series using R* : Springer, 2009.
- [13] R.L. Devaney, *Introduction to Chaotic Dynamical Systems* : Avalon Publishing, 1989.
- [14] A.M. Fraser, H.L. Swinney, " Independent Coordinates for Strange Attractors from Mutual Information", *Physical Review A* 33, pp. 1134-1140, 1986.
- [15] K. Greff, R.K Srivastava, J. Koutník, B.R. Steunebrink, J. Schmidhuber, "LSTM: A Search Space Odyssey", *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 10, pp. 2222-2232, 2017.
- [16] B. Hasselblatt, K. Anatole, *A First Course in Dynamics: With a Panorama of Recent Developments* : Cambridge University Press, pp. 242-251, 2003.
- [17] S. Hochreiter, J. Schmidhuber," Long Short-Term Memory", *Neural Computation*, vol. 9, no. 8, pp. 1735-1780, 1997.
- [18] R.J. Hyndman, G. Athanasopoulos, *Forecasting: Principles and Practice* : OTexts, 2018.
- [19] G.Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, T. Liu, "LightGBM: A Highly Efficient Gradient Boosting Decision Tree", *31st Conference on Neural Information Processing Systems*, 2017.
- [20] M.B. Kennel, R. Brown, H.D.I. Abarbanel, "Determining Embedding Dimension for Phase-space Reconstruction Using a Geometrical Construction", *Physical Review A* 43, pp. 3403-3411, 1992.

- [21] D. Kingma, J. Ba, "Adam: A Method for Stochastic Optimization", *International Conference on Learning Representations*, 2014.
- [22] H. Kriegel, M. Schubert, A. Zimek, "Angle-Based Outlier Detection in High-Dimensional Data", *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 444–452, 2008.
- [23] D. Kwiatkowski, B. Phillips, P. Schmidt, Y. Shin, "Testing the Null Hypothesis of Stationarity against The Alternative of a Unit Root", *Journal of Econometrics*, 54 (1–3), no. 9, pp. 44-56, 1992.
- [24] A. Lazarevic, V. Kumar, " Feature Bagging for Outlier Detection", *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 157-166, 2005.
- [25] F.T. Liu, K. Ting, Z. Zhou, " Isolation-Based Anomaly Detection", *ACM Transactions on Knowledge Discovery From Data*, pp. 1-39, 2012.
- [26] E. N. Lorenz, "Deterministic Nonperiodic Flow", *Journal of the Atmospheric Sciences*, 20 (2), pp. 130–141, 1963.
- [27] E. N. Lorenz, "Atmospheric Predictability as Revealed by Naturally Occurring Analogues", *Journal of the Atmospheric Sciences*, 26 (4), pp. 34-56, 1969.
- [28] J. M. Martinerie, A. M. Albano, A. I. Mees, P. E. Rapp, " Mutual Information, Strange Attractors, and the Optimal Estimation of Dimension", *Physical Review, A* 45, p. 7058, 1992.
- [29] B. Nielsen, "Correlograms for Non-Stationary Autoregressions", *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, vol. 36, no. 9, pp. 44-56, 1987.
- [30] C. Nwankpa, W. Ijomah, A. Gachagan, S. Marshall, *Activation Functions: Comparison of Trends in Practice and Research for Deep Learning* : arXiv preprint arXiv:1811.03378, 2018.
- [31] S.M. Pincus, "Sample Entropy", *Proc. Nati. Acad. Sci. USA*, Vol. 88, pp. 2297-2301, 1991.

- [32] J.S. Richman, D.E. Lake, J.R. Moorman, "Sample Entropy", *Methods in Enzymology*, Vol. 384, pp. 172-184, 2004.
- [33] H. Sak, A.W. Senior, F. Beaufays, *Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling*, 2014.
- [34] C.E. Shannon, "A Mathematical Theory of Communication", *The Bell System Technical Journal*, vol. 27, pp. 379–423, 623–656, 1948.
- [35] P. Smith, *Explaining Chaos* : Cambridge University Press, 1998.
- [36] F. Spitzer, *Principles of Random Walk* : Springer Science and Business Media, 2013.
- [37] S.H. Strogatz, *Nonlinear Dynamics and Chaos : With Applications to Physics, Biology, Chemistry, and Engineering* : Addison-Wesley, 1994, pp. 301-408.
- [38] G.J. Sussman & J. Wisdom, "Chaotic Evolution of the Solar System", *Science*, 257(5066), pp. 56-62, 1992.
- [39] I. Sutskever, O. Vinyals, Q.V. Le, "Sequence to Sequence Learning with Neural Networks", *Advances in Neural Information Processing Systems*, pp. 3104-3112, 2014.
- [40] F. Takens, *Detecting Strange Attractors in Turbulence, Lecture Notes in Mathematics* : Springer-Verlag, p.898, 1981.
- [41] J. Theiler, "Efficient Algorithm for Estimating the Correlation Dimension from a Set of Discrete Points", *Physical Review A* Vol. 36, No. 9, pp. 44-56, 1987.
- [42] J. Theiler, B. Galdrikian, A. Longtin, S. Eubank, J. D. Farmer, "Using Surrogate Data to Detect Nonlinearity in Time Series, in Nonlinear Modelling and Forecasting", *Physica D: Nonlinear Phenomena*, Vol. 58, No. 1–4, pp. 77-94, 1992.
- [43] D.H. Wolpert, "Stacked Generalization", *Neural Networks*, vol. 5, no. 2, pp. 241-259, 1992.
- .

Appendix-A

A.1 CODE FOR EDA IN PYTHON

```
from google.colab import drive
drive.mount('/content/drive')

import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
import missingno
import warnings

warnings.filterwarnings("ignore")
%matplotlib inline
import pandas as pd
df=pd.read_csv('drive/My Drive/AL_WIND_07_12 (1)csvupdated.csv')
df.head()

def time_series_plot(df):
    """Given dataframe, generate times series plot of numeric data
```

```

        bydaily , monthly and yearly frequency"""

print("\nTo check time series of numeric data by daily , monthly
      and yearly frequency")

if len(df.select_dtypes(include='datetime64').columns)==0:
    for col in df.select_dtypes(include='datetime64').columns:
        for p in [ 'D' , 'M' , 'Y']:
            if p=='D':
                print("Plotting daily data")
            elif p=='M':
                print("Plotting monthly data")
            else:
                print("Plotting yearly data")
for col_num in df.select_dtypes(include=np.number).columns:
    __ = df.copy()
    __ = __.set_index(col)
    __T = __.resample(p).sum()
    ax = __T[[col_num]].plot()
    ax.set_ylim(bottom=0)
    ax.get_yaxis().set_major_formatter(
        matplotlib.ticker.FuncFormatter(lambda x, p: format
            (int(x) , ',')))
    plt.show()

def numeric_eda(df, hue=None):
    """Given dataframe, generate EDA of numeric data"""
    print("\nTo check: \nDistribution of numeric data")
    display(df.describe().T)
    columns = df.select_dtypes(include=np.number).columns

```

```

figure = plt.figure(figsize=(20, 10))
figure.add_subplot(1, len(columns), 1)
for index, col in enumerate(columns):
    if index > 0:
        figure.add_subplot(1, len(columns), index + 1)
    sns.boxplot(y=col, data=df, boxprops={'facecolor': 'None'})
figure.tight_layout()
plt.show()
if len(df.select_dtypes(include='category').columns) > 0:
    for col_num in df.select_dtypes(include=np.number).columns:
        for col in df.select_dtypes(include='category').columns:
            fig = sns.catplot(x=col, y=col_num, kind='violin', data=
                df, height=5, aspect=2)
            fig.set_xticklabels(rotation=90)
    plt.show()

# Plot the pairwise joint distributions
print(" To check pairwise joint distribution of numeric data")
if hue==None:
    sns.pairplot(df.select_dtypes(include=np.number))
else:
    sns.pairplot(df.select_dtypes(include=np.number).join(df[[hue]]),
                 hue=hue)
plt.show()

def top5(df):
    """Given dataframe, generate top 5 unique values for non-numeric

```

```

data"""

columns = df.select_dtypes(include=['object', 'category']).columns
for col in columns:
    print("Top 5 unique values of " + col)
    print(df[col].value_counts().reset_index().rename(columns={"index": col, col: "Count"})[
        :min(5, len(df[col].value_counts()))])
    print(" ")

```



```

def categorical_eda(df, hue=None):
    """Given dataframe, generate EDA of categorical data"""
    print("\nTo check: \nUnique count of non-numeric data\n")
    print(df.select_dtypes(include=['object', 'category']).nunique())
    top5(df)

    # Plot count distribution of categorical data
    for col in df.select_dtypes(include='category').columns:
        fig = sns.catplot(x=col, kind="count", data=df, hue=hue)
        fig.set_xticklabels(rotation=90)
        plt.show()

```



```

def eda(df):
    """Given dataframe, generate exploratory data analysis"""
    # check that input is pandas dataframe
    if type(df) != pd.core.frame.DataFrame:
        raise TypeError("Only pandas dataframe is allowed as input")

    # replace field that's entirely space (or empty) with NaN
    df = df.replace(r'^\s*', np.nan, regex=True)

```

```

print("Preview of data:")
display(df.head(3))

print("\nTo check: \n (1) Total number of entries \n (2) Column
      types \n (3) Any null values\n")
print(df.info())

# generate preview of entries with null values
if len(df[df.isnull().any(axis=1)] != 0):
    print("\nPreview of data with null values:")
    display(df[df.isnull().any(axis=1)].head(3))
    missingno.matrix(df)
    plt.show()

# generate count statistics of duplicate entries
if len(df[df.duplicated()]) > 0:
    print("\n***Number of duplicated entries: ", len(df[df.
        duplicated()]))
    display(df[df.duplicated(keep=False)].sort_values(by=list(df.
        columns)).head())
else:
    print("\nNo duplicated entries found")

# EDA of categorical data
categorical_eda(df)

# EDA of numeric data
numeric_eda(df)

```

```

# Plot time series plot of numeric data
time_series_plot(df)

eda(df)

duplicate_rows_df = df[df.duplicated()]
print("number of duplicate rows:", duplicate_rows_df.shape)

df.count()

# Finding the null values.
print(df.isnull().sum())

# Dropping the missing values.
df = df.dropna()
df.count()

# After dropping the values
print(df.isnull().sum())

!pip install pyod
!pip install --upgrade pyod # to make sure that the latest version is
installed!

from pyod.models.abod import ABOD
from pyod.models.cblobf import CBLOF

```

```
from pyod.models.feature_bagging import FeatureBagging
from pyod.models.hbos import HBOS
from pyod.models.iforest import IForest
from pyod.models.knn import KNN
from pyod.models.lof import LOF
# reading the training data

df=pd.read_csv('drive/My Drive/AL_WIND_07_12FINAL.csv')

df.plot.scatter('Hour','Air temperature')

df.plot.scatter('Hour','Pressure')

df.plot.scatter('Hour','Wind speed')

df.plot.scatter('Hour','Wind direction')

df.plot.scatter('Hour','Power generated by system')
```

```

df.plot.scatter('Wind speed', 'Power generated by system')

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range=(0, 1))
df[['Wind speed', 'Power generated by system']] = scaler.fit_transform(
    df[['Wind speed', 'Power generated by system']])
df[['Wind speed', 'Power generated by system']].head()

X1 = df['Wind speed'].values.reshape(-1,1)
X2 = df['Power generated by system'].values.reshape(-1,1)

X = np.concatenate((X1,X2),axis=1)

random_state = np.random.RandomState(42)
outliers_fraction = 0.05
# Define seven outlier detection tools to be compared
classifiers = {
    'Angle-based Outlier Detector (ABOD)': ABOD(contamination=
        outliers_fraction),
    'Cluster-based Local Outlier Factor (CBLOF)': CBLOF(
        contamination=outliers_fraction, check_estimator=False,
        random_state=random_state),
    'Feature Bagging': FeatureBagging(LOF(n_neighbors=35),
        contamination=outliers_fraction, check_estimator=False,

```

```

        random_state=random_state) ,
'Histogram-base Outlier Detection (HBOS)': HBOS(contamination=
    outliers_fraction),
'Isolation Forest': IForest(contamination=outliers_fraction,
    random_state=random_state),
'K Nearest Neighbors (KNN)': KNN(contamination=
    outliers_fraction),
'Average KNN': KNN(method='mean', contamination=
    outliers_fraction)
}

xx , yy = np.meshgrid(np.linspace(0,1 , 200) , np.linspace(0, 1, 200))
from scipy import stats

for i , (clf_name, clf) in enumerate(classifiers.items()):
    clf . fit (X)
    # predict raw anomaly score
    scores_pred = clf . decision_function (X) * -1

    # prediction of a datapoint category outlier or inlier
    y_pred = clf . predict (X)
    n_inliers = len(y_pred) - np.count_nonzero(y_pred)
    n_outliers = np.count_nonzero(y_pred == 1)
    plt . figure (figsize=(10, 10))

    # copy of dataframe
    dfx = df
    dfx[ 'outlier' ] = y_pred . tolist ()

```

```

# IX1 - inlier feature 1, IX2 - inlier feature 2
IX1 = np.array(dfx[ 'Wind speed' ][ dfx[ 'outlier' ] == 0]).reshape( -1,1)
IX2 = np.array(dfx[ 'Power generated by system' ][ dfx[ 'outlier' ] ==
0]).reshape(-1,1)

# OX1 - outlier feature 1, OX2 - outlier feature 2
OX1 = dfx[ 'Wind speed' ][ dfx[ 'outlier' ] == 1].values.reshape(-1,1)
OX2 = dfx[ 'Power generated by system' ][ dfx[ 'outlier' ] == 1].values
.reshape(-1,1)

print('OUTLIERS : ', n_outliers, 'INLIERS : ', n_inliers, clf_name)

# threshold value to consider a datapoint inlier or outlier
threshold = stats.scoreatpercentile(scores_pred, 100 *
outliers_fraction)

# decision function calculates the raw anomaly score for every
point
Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()]) * -1
Z = Z.reshape(xx.shape)

# fill blue map colormap from minimum anomaly score to threshold
value
plt.contourf(xx, yy, Z, levels=np.linspace(Z.min(), threshold, 7),
cmap=plt.cm.Blues_r)

# draw red contour line where anomaly score is equal to threshold
a = plt.contour(xx, yy, Z, levels=[threshold], linewidths=2, colors

```

```

= 'red')

# fill orange contour lines where range of anomaly score is from
# threshold to maximum anomaly score
plt.contourf(xx, yy, Z, levels=[threshold, Z.max()], colors='orange')

b = plt.scatter(IX1, IX2, c='white', s=20, edgecolor='k')

c = plt.scatter(OX1, OX2, c='black', s=20, edgecolor='k')

plt.axis('tight')

# loc=2 is used for the top left corner
plt.legend(
    [a.collections[0], b, c],
    ['learned decision function', 'inliers', 'outliers'],
    prop=matplotlib.font_manager.FontProperties(size=20),
    loc=2)

plt.xlim((0, 1))
plt.ylim((0, 1))
plt.title(clf_name)
plt.show()

```

```

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range=(0, 1))
df[['Hour', 'Power generated by system']] = scaler.fit_transform(df[['Hour', 'Power generated by system']])
df[['Hour', 'Power generated by system']].head()

X1 = df['Hour'].values.reshape(-1,1)
X2 = df['Power generated by system'].values.reshape(-1,1)

X = np.concatenate((X1,X2),axis=1)

random_state = np.random.RandomState(42)
outliers_fraction = 0.05
# Define seven outlier detection tools to be compared
classifiers = {
    'Angle-based Outlier Detector (ABOD)': ABOD(contamination=outliers_fraction),
    'Cluster-based Local Outlier Factor (CBLOF)': CBLOF(
        contamination=outliers_fraction, check_estimator=False,
        random_state=random_state),
    'Feature Bagging': FeatureBagging(LOF(n_neighbors=35),
        contamination=outliers_fraction, check_estimator=False,
        random_state=random_state),
    'Histogram-base Outlier Detection (HBOS)': HBOS(contamination=outliers_fraction),
    'Isolation Forest': IForest(contamination=outliers_fraction,

```

```

        random_state=random_state) ,
'K Nearest Neighbors (KNN)': KNN(contamination=
    outliers_fraction),
'Average KNN': KNN(method='mean', contamination=
    outliers_fraction)
}

xx , yy = np.meshgrid(np.linspace(0,1 , 200) , np.linspace(0, 1, 200))
from scipy import stats

for i , (clf_name, clf) in enumerate(classifiers.items()):
    clf.fit(X)
    # predict raw anomaly score
    scores_pred = clf.decision_function(X) * -1

    # prediction of a datapoint category outlier or inlier
    y_pred = clf.predict(X)
    n_inliers = len(y_pred) - np.count_nonzero(y_pred)
    n_outliers = np.count_nonzero(y_pred == 1)
    plt.figure(figsize=(10, 10))

    # copy of dataframe
    dfx = df
    dfx['outlier'] = y_pred.tolist()

    # IX1 - inlier feature 1, IX2 - inlier feature 2
    IX1 = np.array(dfx['Hour'][dfx['outlier'] == 0]).reshape(-1,1)

```

```

IX2 = np.array(dfx['Power generated by system'][dfx['outlier'] == 0]).reshape(-1,1)

# OX1 - outlier feature 1, OX2 - outlier feature 2
OX1 = dfx['Hour'][dfx['outlier'] == 1].values.reshape(-1,1)
OX2 = dfx['Power generated by system'][dfx['outlier'] == 1].values
    .reshape(-1,1)

print('OUTLIERS : ', n_outliers, 'INLIERS : ', n_inliers, clf_name)

# threshold value to consider a datapoint inlier or outlier
threshold = stats.scoreatpercentile(scores_pred, 100 *
    outliers_fraction)

# decision function calculates the raw anomaly score for every
# point
Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()]) * -1
Z = Z.reshape(xx.shape)

# fill blue map colormap from minimum anomaly score to threshold
# value
plt.contourf(xx, yy, Z, levels=np.linspace(Z.min(), threshold, 7),
    cmap=plt.cm.Blues_r)

# draw red contour line where anomaly score is equal to threshold
a = plt.contour(xx, yy, Z, levels=[threshold], linewidths=2, colors
    ='red')

# fill orange contour lines where range of anomaly score is from

```

```

    threshold to maximum anomaly score

plt.contourf(xx, yy, Z, levels=[threshold, Z.max()], colors='orange')

b = plt.scatter(IX1, IX2, c='white', s=20, edgecolor='k')

c = plt.scatter(OX1, OX2, c='black', s=20, edgecolor='k')

plt.axis('tight')

# loc=2 is used for the top left corner
plt.legend(
    [a.collections[0], b,c],
    ['learned decision function', 'inliers', 'outliers'],
    prop=matplotlib.font_manager.FontProperties(size=20),
    loc=2)

plt.xlim((0, 1))
plt.ylim((0, 1))
plt.title(clf_name)
plt.show()

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range=(0, 1))
df[['Hour', 'Wind speed']] = scaler.fit_transform(df[['Hour', 'Wind speed']])

```

```

df[['Hour', 'Wind speed']].head()

X1 = df['Hour'].values.reshape(-1,1)
X2 = df['Wind speed'].values.reshape(-1,1)

X = np.concatenate((X1,X2),axis=1)

random_state = np.random.RandomState(42)
outliers_fraction = 0.05
# Define seven outlier detection tools to be compared
classifiers = {
    'Angle-based Outlier Detector (ABOD)': ABOD(contamination=
        outliers_fraction),
    'Feature Bagging': FeatureBagging(LOF(n_neighbors=35),
        contamination=outliers_fraction, check_estimator=False,
        random_state=random_state),
    'Histogram-base Outlier Detection (HBOS)': HBOS(contamination=
        outliers_fraction),
    'Isolation Forest': IForest(contamination=outliers_fraction,
        random_state=random_state),
    'K Nearest Neighbors (KNN)': KNN(contamination=
        outliers_fraction),
    'Average KNN': KNN(method='mean', contamination=
        outliers_fraction)
}

```

```

xx , yy = np.meshgrid(np.linspace(0,1 , 200) , np.linspace(0, 1, 200))
from scipy import stats

for i , (clf_name, clf) in enumerate(classifiers.items()):
    clf.fit(X)
    # predict raw anomaly score
    scores_pred = clf.decision_function(X) * -1

    # prediction of a datapoint category outlier or inlier
    y_pred = clf.predict(X)
    n_inliers = len(y_pred) - np.count_nonzero(y_pred)
    n_outliers = np.count_nonzero(y_pred == 1)
    plt.figure(figsize=(10, 10))

    # copy of dataframe
    dfx = df
    dfx['outlier'] = y_pred.tolist()

    # IX1 - inlier feature 1, IX2 - inlier feature 2
    IX1 = np.array(dfx['Hour'][dfx['outlier'] == 0]).reshape(-1,1)
    IX2 = np.array(dfx['Wind speed'][dfx['outlier'] == 0]).reshape
        (-1,1)

    # OX1 - outlier feature 1, OX2 - outlier feature 2
    OX1 = dfx['Hour'][dfx['outlier'] == 1].values.reshape(-1,1)
    OX2 = dfx['Wind speed'][dfx['outlier'] == 1].values.reshape(-1,1)

```

```

print('OUTLIERS : ', n_outliers, 'INLIERS : ', n_inliers, clf_name)

# threshold value to consider a datapoint inlier or outlier
threshold = stats.scoreatpercentile(scores_pred, 100 *
                                       outliers_fraction)

# decision function calculates the raw anomaly score for every
# point
Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()]) * -1
Z = Z.reshape(xx.shape)

# fill blue map colormap from minimum anomaly score to threshold
# value
plt.contourf(xx, yy, Z, levels=np.linspace(Z.min(), threshold, 7),
             cmap=plt.cm.Blues_r)

# draw red contour line where anomaly score is equal to threshold
a = plt.contour(xx, yy, Z, levels=[threshold], linewidths=2, colors
                ='red')

# fill orange contour lines where range of anomaly score is from
# threshold to maximum anomaly score
plt.contourf(xx, yy, Z, levels=[threshold, Z.max()], colors='orange
                ')

b = plt.scatter(IX1, IX2, c='white', s=20, edgecolor='k')

c = plt.scatter(OX1, OX2, c='black', s=20, edgecolor='k')

```

```

plt.axis('tight')

# loc=2 is used for the top left corner
plt.legend(
    [a.collections[0], b,c],
    ['learned decision function', 'inliers', 'outliers'],
    prop=matplotlib.font_manager.FontProperties(size=20),
    loc=2)

plt.xlim((0, 1))
plt.ylim((0, 1))
plt.title(clf_name)
plt.show()

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range=(0, 1))
df[['Hour', 'Pressure']] = scaler.fit_transform(df[['Hour', 'Pressure']])
df[['Hour', 'Pressure']].head()

X1 = df['Hour'].values.reshape(-1,1)
X2 = df['Pressure'].values.reshape(-1,1)

X = np.concatenate((X1,X2),axis=1)

```

```

random_state = np.random.RandomState(42)
outliers_fraction = 0.05
# Define seven outlier detection tools to be compared
classifiers = {
    'Angle-based Outlier Detector (ABOD)': ABOD(contamination=
        outliers_fraction),
    #'Cluster-based Local Outlier Factor (CBLOF)':CBLOF(
        contamination=outliers_fraction,check_estimator=False,
        random_state=random_state),
    'Feature Bagging': FeatureBagging(LOF(n_neighbors=35),
        contamination=outliers_fraction,check_estimator=False,
        random_state=random_state),
    'Histogram-base Outlier Detection (HBOS)': HBOS(contamination=
        outliers_fraction),
    'Isolation Forest': IForest(contamination=outliers_fraction,
        random_state=random_state),
    'K Nearest Neighbors (KNN)': KNN(contamination=
        outliers_fraction),
    'Average KNN': KNN(method='mean',contamination=
        outliers_fraction)
}

```

```

xx , yy = np.meshgrid(np.linspace(0,1 , 200) , np.linspace(0, 1, 200))
from scipy import stats

```

```

for i, (clf_name, clf) in enumerate(classifiers.items()):
    clf.fit(X)
    # predict raw anomaly score
    scores_pred = clf.decision_function(X) * -1

    # prediction of a datapoint category outlier or inlier
    y_pred = clf.predict(X)
    n_inliers = len(y_pred) - np.count_nonzero(y_pred)
    n_outliers = np.count_nonzero(y_pred == 1)
    plt.figure(figsize=(10, 10))

    # copy of dataframe
    dfx = df
    dfx['outlier'] = y_pred.tolist()

    # IX1 - inlier feature 1, IX2 - inlier feature 2
    IX1 = np.array(dfx['Hour'][dfx['outlier'] == 0]).reshape(-1,1)
    IX2 = np.array(dfx['Pressure'][dfx['outlier'] == 0]).reshape(-1,1)

    # OX1 - outlier feature 1, OX2 - outlier feature 2
    OX1 = dfx['Hour'][dfx['outlier'] == 1].values.reshape(-1,1)
    OX2 = dfx['Pressure'][dfx['outlier'] == 1].values.reshape(-1,1)

    print('OUTLIERS : ', n_outliers, 'INLIERS : ', n_inliers, clf_name)

    # threshold value to consider a datapoint inlier or outlier
    threshold = stats.scoreatpercentile(scores_pred, 100 *
                                         outliers_fraction)

```

```

# decision function calculates the raw anomaly score for every
point

Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()]) * -1
Z = Z.reshape(xx.shape)

# fill blue map colormap from minimum anomaly score to threshold
value
plt.contourf(xx, yy, Z, levels=np.linspace(Z.min(), threshold, 7),
cmap=plt.cm.Blues_r)

# draw red contour line where anomaly score is equal to thresold
a = plt.contour(xx, yy, Z, levels=[threshold], linewidths=2, colors
='red')

# fill orange contour lines where range of anomaly score is from
threshold to maximum anomaly score
plt.contourf(xx, yy, Z, levels=[threshold, Z.max()], colors='orange
')

b = plt.scatter(IX1, IX2, c='white', s=20, edgecolor='k')

c = plt.scatter(OX1, OX2, c='black', s=20, edgecolor='k')

plt.axis('tight')

# loc=2 is used for the top left corner
plt.legend(
    [a.collections[0], b, c],

```

```

[ 'learned decision function' , 'inliers' , 'outliers' ] ,
prop=matplotlib.font_manager.FontProperties(size=20) ,
loc=2)

plt.xlim((0, 1))
plt.ylim((0, 1))
plt.title(clf_name)
plt.show()

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range=(0, 1))
df[['Hour', 'Air temperature']] = scaler.fit_transform(df[['Hour', 'Air
temperature']])
df[['Hour', 'Air temperature']].head()

X1 = df['Hour'].values.reshape(-1,1)
X2 = df['Pressure'].values.reshape(-1,1)

X = np.concatenate((X1,X2),axis=1)

random_state = np.random.RandomState(42)
outliers_fraction = 0.05

```

```

# Define seven outlier detection tools to be compared
classifiers = {
    'Angle-based Outlier Detector (ABOD)': ABOD(contamination=
        outliers_fraction),
    #'Cluster-based Local Outlier Factor (CBLOF)':CBLOF(
        contamination=outliers_fraction,check_estimator=False,
        random_state=random_state),
    'Feature Bagging': FeatureBagging(LOF(n_neighbors=35),
        contamination=outliers_fraction,check_estimator=False,
        random_state=random_state),
    'Histogram-base Outlier Detection (HBOS)': HBOS(contamination=
        outliers_fraction),
    'Isolation Forest': IForest(contamination=outliers_fraction,
        random_state=random_state),
    'K Nearest Neighbors (KNN)': KNN(contamination=
        outliers_fraction),
    'Average KNN': KNN(method='mean',contamination=
        outliers_fraction)
}

```

```

xx , yy = np.meshgrid(np.linspace(0,1 , 200) , np.linspace(0, 1, 200))
from scipy import stats

```

```

for i, (clf_name, clf) in enumerate(classifiers.items()):
    clf.fit(X)
    # predict raw anomaly score
    scores_pred = clf.decision_function(X) * -1

```

```

# prediction of a datapoint category outlier or inlier
y_pred = clf.predict(X)
n_inliers = len(y_pred) - np.count_nonzero(y_pred)
n_outliers = np.count_nonzero(y_pred == 1)
plt.figure(figsize=(10, 10))

# copy of dataframe
dfx = df
dfx['outlier'] = y_pred.tolist()

# IX1 - inlier feature 1, IX2 - inlier feature 2
IX1 = np.array(dfx['Hour'][dfx['outlier'] == 0]).reshape(-1,1)
IX2 = np.array(dfx['Air temperature'][dfx['outlier'] == 0]).\
    reshape(-1,1)

# OX1 - outlier feature 1, OX2 - outlier feature 2
OX1 = dfx['Hour'][dfx['outlier'] == 1].values.reshape(-1,1)
OX2 = dfx['Air temperature'][dfx['outlier'] == 1].values.reshape
    (-1,1)

print('OUTLIERS : ', n_outliers, 'INLIERS : ', n_inliers, clf_name)

# threshold value to consider a datapoint inlier or outlier
threshold = stats.scoreatpercentile(scores_pred, 100 *
    outliers_fraction)

# decision function calculates the raw anomaly score for every
point

```

```

Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()]) * -1
Z = Z.reshape(xx.shape)

# fill blue map colormap from minimum anomaly score to threshold
# value
plt.contourf(xx, yy, Z, levels=np.linspace(Z.min(), threshold, 7),
cmap=plt.cm.Blues_r)

# draw red contour line where anomaly score is equal to threshold
a = plt.contour(xx, yy, Z, levels=[threshold], linewidths=2, colors
='red')

# fill orange contour lines where range of anomaly score is from
# threshold to maximum anomaly score
plt.contourf(xx, yy, Z, levels=[threshold, Z.max()], colors='orange
')

b = plt.scatter(IX1, IX2, c='white', s=20, edgecolor='k')

c = plt.scatter(OX1, OX2, c='black', s=20, edgecolor='k')

plt.axis('tight')

# loc=2 is used for the top left corner
plt.legend(
    [a.collections[0], b, c],
    ['learned decision function', 'inliers', 'outliers'],
    prop=matplotlib.font_manager.FontProperties(size=20),
    loc=2)

```

```

plt.xlim((0, 1))
plt.ylim((0, 1))
plt.title(clf_name)
plt.show()

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range=(0, 1))
df[['Hour', 'Wind direction']] = scaler.fit_transform(df[['Hour', 'Wind
direction']])
df[['Hour', 'Wind direction']].head()

X1 = df['Hour'].values.reshape(-1,1)
X2 = df['Wind direction'].values.reshape(-1,1)

X = np.concatenate((X1,X2),axis=1)

random_state = np.random.RandomState(42)
outliers_fraction = 0.05
# Define seven outlier detection tools to be compared
classifiers = {
    'Angle-based Outlier Detector (ABOD)': ABOD(contamination=

```

```

        outliers_fraction) ,
 #'Cluster-based Local Outlier Factor (CBLOF)':CBLOF(
    contamination=outliers_fraction ,check_estimator=False ,
    random_state=random_state) ,
 'Feature Bagging':FeatureBagging(LOF(n_neighbors=35) ,
    contamination=outliers_fraction ,check_estimator=False ,
    random_state=random_state) ,
 'Histogram-base Outlier Detection (HBOS)': HBOS(contamination=
    outliers_fraction) ,
 'Isolation Forest': IForest(contamination=outliers_fraction ,
    random_state=random_state) ,
 'K Nearest Neighbors (KNN)': KNN(contamination=
    outliers_fraction) ,
 'Average KNN': KNN(method='mean' ,contamination=
    outliers_fraction)
}

```

```

xx , yy = np.meshgrid(np.linspace(0,1 , 200) , np.linspace(0, 1, 200))
from scipy import stats

for i , (clf_name, clf) in enumerate(classifiers.items()):
    clf.fit(X)
    # predict raw anomaly score
    scores_pred = clf.decision_function(X) * -1

    # prediction of a datapoint category outlier or inlier
    y_pred = clf.predict(X)

```

```

n_inliers = len(y_pred) - np.count_nonzero(y_pred)
n_outliers = np.count_nonzero(y_pred == 1)
plt.figure(figsize=(10, 10))

# copy of dataframe
dfx = df
dfx['outlier'] = y_pred.tolist()

# IX1 - inlier feature 1, IX2 - inlier feature 2
IX1 = np.array(dfx['Hour'][dfx['outlier'] == 0]).reshape(-1,1)
IX2 = np.array(dfx['Wind direction'][dfx['outlier'] == 0]).reshape
(-1,1)

# OX1 - outlier feature 1, OX2 - outlier feature 2
OX1 = dfx['Hour'][dfx['outlier'] == 1].values.reshape(-1,1)
OX2 = dfx['Wind direction'][dfx['outlier'] == 1].values.reshape
(-1,1)

print('OUTLIERS : ', n_outliers, 'INLIERS : ', n_inliers, clf_name)

# threshold value to consider a datapoint inlier or outlier
threshold = stats.scoreatpercentile(scores_pred, 100 *
outliers_fraction)

# decision function calculates the raw anomaly score for every
point
Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()]) * -1
Z = Z.reshape(xx.shape)

```

```

# fill blue map colormap from minimum anomaly score to threshold
# value
plt.contourf(xx, yy, Z, levels=np.linspace(Z.min(), threshold, 7),
cmap=plt.cm.Blues_r)

# draw red contour line where anomaly score is equal to threshold
a = plt.contour(xx, yy, Z, levels=[threshold], linewidths=2, colors
='red')

# fill orange contour lines where range of anomaly score is from
# threshold to maximum anomaly score
plt.contourf(xx, yy, Z, levels=[threshold, Z.max()], colors='orange
')

b = plt.scatter(IX1, IX2, c='white', s=20, edgecolor='k')

c = plt.scatter(OX1, OX2, c='black', s=20, edgecolor='k')

plt.axis('tight')

# loc=2 is used for the top left corner
plt.legend(
    [a.collections[0], b, c],
    ['learned decision function', 'inliers', 'outliers'],
    prop=matplotlib.font_manager.FontProperties(size=20),
    loc=2)

plt.xlim((0, 1))
plt.ylim((0, 1))

```

```
plt.title(clf_name)
plt.show()
```

A.2 CODE FOR FEATURE ENGINEERING IN PYTHON

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import math
from scipy.stats import pearsonr
import warnings
warnings.filterwarnings("ignore")
import seaborn as sns
from google.colab import drive
drive.mount('/content/drive')
data1 = pd.read_csv("/content/drive/My Drive/data1.csv")
data1.head()
data1 = pd.read_csv("/content/drive/My Drive/data1.csv")
data1.head()
data2 = pd.read_csv("/content/drive/My Drive/data2.csv")
data2.head()
data3 = pd.read_csv("/content/drive/My Drive/data3.csv")
data3.head()
data4 = pd.read_csv("/content/drive/My Drive/data4.csv")
data4.head()
data5 = pd.read_csv("/content/drive/My Drive/data5.csv")
data5.head()
data6 = pd.read_csv("/content/drive/My Drive/data6.csv")
```

```

data6.head()

features = data1.iloc[:, 2: ].columns.tolist()
target = data1.iloc[:, 1].name

features = data5.iloc[:, 2: ].columns.tolist()
target = data5.iloc[:, 1].name

features = data4.iloc[:, 2: ].columns.tolist()
target = data4.iloc[:, 1].name

features = data2.iloc[:, 2: ].columns.tolist()
target = data2.iloc[:, 1].name

features = data3.iloc[:, 2: ].columns.tolist()
target = data3.iloc[:, 1].name

features = data6.iloc[:, 2: ].columns.tolist()
target = data6.iloc[:, 1].name

str_list = []
for colname, colvalue in data1.iteritems():
    if type(colvalue[1]) == str:
        str_list.append(colname)

num_list = data1.columns.difference(str_list)
house_num = data1[num_list]

f, ax = plt.subplots(figsize=(16, 12))
plt.title('Pearson Correlation of features')
sns.heatmap(house_num.astype(float).corr(), linewidths=0.25, vmax=1,
            square=True, cmap="PuBuGn", linecolor='k', annot=True)

x=data1['Wind speed | (m/s)']
y=data1['Power generated by system | (kW)']

p_value = pearsonr(x,y)
print(p_value)

```

```

str_list = []
for colname, colvalue in data2.iteritems():
    if type(colvalue[1]) == str:
        str_list.append(colname)
num_list = data2.columns.difference(str_list)
house_num = data2[num_list]
f, ax = plt.subplots(figsize=(16, 12))
plt.title('Pearson Correlation of features')
sns.heatmap(house_num.astype(float).corr(), linewidths=0.25, vmax=1,
            square=True, cmap="PuBuGn", linecolor='k', annot=True)
str_list = []
for colname, colvalue in data3.iteritems():
    if type(colvalue[1]) == str:
        str_list.append(colname)
num_list = data3.columns.difference(str_list)
house_num = data3[num_list]
f, ax = plt.subplots(figsize=(16, 12))
plt.title('Pearson Correlation of features')
sns.heatmap(house_num.astype(float).corr(), linewidths=0.25, vmax=1,
            square=True, cmap="PuBuGn", linecolor='k', annot=True)

str_list = []
for colname, colvalue in data4.iteritems():
    if type(colvalue[1]) == str:
        str_list.append(colname)
num_list = data4.columns.difference(str_list)
house_num = data4[num_list]
f, ax = plt.subplots(figsize=(16, 12))

```

```

plt.title('Pearson Correlation of features')
sns.heatmap(house_num.astype(float).corr(), linewidths=0.25, vmax=1,
            square=True, cmap="PuBuGn", linecolor='k', annot=True)
str_list = []
for colname, colvalue in data5.iteritems():
    if type(colvalue[1]) == str:
        str_list.append(colname)
num_list = data5.columns.difference(str_list)
house_num = data5[num_list]
f, ax = plt.subplots(figsize=(16, 12))
plt.title('Pearson Correlation of features')
sns.heatmap(house_num.astype(float).corr(), linewidths=0.25, vmax=1,
            square=True, cmap="PuBuGn", linecolor='k', annot=True)
str_list = []
for colname, colvalue in data6.iteritems():
    if type(colvalue[1]) == str:
        str_list.append(colname)
num_list = data6.columns.difference(str_list)
house_num = data6[num_list]
f, ax = plt.subplots(figsize=(16, 12))
plt.title('Pearson Correlation of features')
sns.heatmap(house_num.astype(float).corr(), linewidths=0.25, vmax=1,
            square=True, cmap="PuBuGn", linecolor='k', annot=True)
correlation={}
for f in features:
    data = data1[[f, target]]
    x1= data[f].values
    x2= data[target].values
    key = f + ' vs ' + target

```

```

correlation[key] = pearsonr(x1,x2)[0]
data_correlations = pd.DataFrame(correlation, index=['Value']).T
data_correlations.loc[data_correlations['Value'].abs().sort_values(
    ascending=False).index]

#CODE FOR ESTIMATING MUTUAL INFORMATION
!pip install lightgbm xgboost scikit-learn pandas mlxtend --upgrade
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import xgboost
import math
from scipy.stats import pearsonr
from sklearn.model_selection import cross_validate
#from sklearn import cross_validation, tree, linear_model
from sklearn.model_selection import train_test_split
from sklearn.metrics import explained_variance_score
from sklearn.linear_model import LinearRegression
#regr = linear_model.LinearRegression()
regr = LinearRegression()
import warnings
warnings.filterwarnings("ignore")
import seaborn as sns
# Loading the dataset from Scikit-Learn
from google.colab import drive
drive.mount('/content/drive')
data = pd.read_csv(r"/content/drive/My Drive/data5(2).csv")
data.head()

```

```

sorted(data.columns)
print(data.shape)
print(len(data))
print(len(data.columns))
print(data.dtypes.unique()) #3 types data h, int, objective and float
print(data.dtypes)
data.select_dtypes(include=['O']).columns.tolist()
print(data.isnull().any().sum(), ' / ', len(data.columns))
print(data.isnull().any(axis=1).sum(), ' / ', len(data))
features = data.iloc[:,2:].columns.tolist()
target = data.iloc[:,1].name
data_correlations = pd.DataFrame(correlation, index=['Value']).T #
    checking which one is mostly co realted with price
data_correlations.loc[data_correlations['Value'].abs().sort_values(
    ascending=False).index]
X = data.drop(['Power generated by system | (kW)', 'DateTime'], axis =
    1)
y = data['Power generated by system | (kW)']
X.shape, y.shape
#X_train, X_test, y_train, y_test = cross_validate.train_test_split(X,
    y ,test_size=0.3)
X_train, X_test, y_train, y_test=train_test_split(X,y,test_size=0.5)
from sklearn.feature_selection import VarianceThreshold,
    mutual_info_classif, mutual_info_regression
from sklearn.feature_selection import SelectKBest, SelectPercentile
mi = mutual_info_regression(X_train, y_train)
mi = pd.Series(mi)
mi.index = X_train.columns
mi.sort_values(ascending=False, inplace = True)

```

```

mi
mi.plot.bar()

sel = SelectKBest(mutual_info_regression, k = 1).fit(X_train, y_train)
X_train.columns[sel.get_support()]

model = LinearRegression()
model.fit(X_train, y_train)
y_predict = model.predict(X_test)

#CODE FOR ESTIMATION OF MEAN SQUARED ERROR CONTRIBUTION BY FEATURES
!pip install lightgbm xgboost scikit-learn pandas mlxtend --upgrade
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import xgboost
import math
from scipy.stats import pearsonr
from sklearn.model_selection import cross_validate
#from sklearn import cross_validation, tree, linear_model
from sklearn.model_selection import train_test_split
from sklearn.metrics import explained_variance_score
from sklearn.linear_model import LinearRegression
#regr = linear_model.LinearRegression()
regr = LinearRegression()
import warnings
warnings.filterwarnings("ignore")
import seaborn as sns
# Loading the dataset from Scikit-Learn
from google.colab import drive

```

```

drive.mount('/content/drive')

data = pd.read_csv(r"/content/drive/My Drive/data5(2).csv")
data.head()
sorted(data.columns)
print(data.shape)
print(len(data))
print(len(data.columns))
print(data.dtypes.unique()) #3 types data h, int, objective and float
print(data.dtypes)
data.select_dtypes(include=['O']).columns.tolist()
print(data.isnull().any().sum(), ' / ', len(data.columns))
print(data.isnull().any(axis=1).sum(), ' / ', len(data))
features = data.iloc[:,2: ].columns.tolist()
target = data.iloc[:,1].name # price is the target to get the
prediction
data_correlations = pd.DataFrame(correlation, index=['Value']).T
data_correlations.loc[data_correlations['Value'].abs().sort_values(
    ascending=False).index]
X = data.drop(['Power generated by system | (kW)', 'DateTime'], axis =
1)
y = data['Power generated by system | (kW)']
X.shape, y.shape
#X_train, X_test, y_train, y_test = cross_validate.train_test_split(X,
#    y ,test_size=0.3)
X_train, X_test, y_train, y_test=train_test_split(X,y, test_size=0.5)

from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score
r2_score(y_test, y_predict)

```

```

np.sqrt(mean_squared_error(y_test, y_predict))
np.std(y)
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
mse = []
for feature in X_train.columns:
    clf = LinearRegression()
    clf.fit(X_train[feature].to_frame(), y_train)
    y_pred = clf.predict(X_test[feature].to_frame())
    mse.append(mean_squared_error(y_test, y_pred))

mse
mse = pd.Series(mse, index = X_train.columns)
mse.sort_values(ascending=False, inplace = True)
mse
mse.plot.bar()

```

A.3 CODE FOR RANDOM WALK TEST IN R

```

#Generation of Random walk
for(i in 2:52560){
  x[i]=x[i-1]+rnorm(1)
  print(x)
}
#Conversion into time series
random_walk=ts(x)
#Plotting the time series
plot(random_walk,main='radom walk',ylab='windspeed',xlab='hour',col='blue',lwd=2)
#Generating Differenced Series
diff(random_walk)
#Plotting Differenced Series
plot(diff(random_walk))

```

A.4 CODE FOR STATIONARITY IN PYTHON

```
#CODE FOR CHECKING STATIONARITY
from google.colab import drive
drive.mount('/content/gdrive')
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
#reading the dataset
train = pd.read_csv('/content/gdrive/My Drive/AirPassengers.csv')

#preprocessing
train.timestamp = pd.to_datetime(train.date_hr , format = '%Y-%m')
train.index = train.timestamp
train.drop('date_hr', axis = 1, inplace = True)
#looking at the first few rows
train.head()
df = pd.read_csv('/content/gdrive/My Drive/AirPassengers.csv',
                 parse_dates=['date_hr'] , index_col='date_hr')
df.plot(title='wind_spd' , figsize=(14,8) , legend=None);
from statsmodels.tsa.stattools import kpss
def kpss_test(series , **kw):
    statistic , p_value , n_lags , critical_values = kpss(series , **kw)
    # Format Output
    print(f'KPSS Statistic: {statistic}')
    print(f'p-value: {p_value}')
    print(f'num lags: {n_lags}')
    print('Critical Values:')
    for key, value in critical_values.items():
```

```

    print(f'{key} : {value}')

print(f'Result: The series is {"not " if p_value < 0.05 else ""}
stationary')

series = df['wind_spd']

kpss_test(series)

from statsmodels.tsa.stattools import adfuller
result = adfuller(series)

print(f'ADF Statistic: {result[0]}')
print(f'n_lags: {result[1]}')
print(f'p-value: {result[1]}')


for key, value in result[4].items():
    print('Critical Values:')
    print(f'{key}, {value}')


from matplotlib import pyplot
from statsmodels.graphics.tsaplots import plot_acf
plot_acf(series)
pyplot.show()

from statsmodels.graphics.tsaplots import plot_pacf
plot_pacf(series, lags=50)
pyplot.show()

from matplotlib import pyplot
from numpy import log
from statsmodels.graphics.tsaplots import plot_acf
plot_acf(log(series))
pyplot.show()

from statsmodels.graphics.tsaplots import plot_pacf
plot_pacf(log(series), lags=50)
pyplot.show()

```

A.5 CODE FOR CHAOTIC ANALYSIS IN R

```
library('nonlinearTseries')
speed= data[1:52560,2]
# tau-delay estimation based on the autocorrelation function
tau.acf = timeLag(speed, technique = "acf",lag.max = 20, do.plot = T)
# tau-delay estimation based on the average mutual information
tau.ami = timeLag(speed, technique = "ami",lag.max = 20, do.plot = T)
# estimating embedding dimension for lag tau.acf
emb.dimacf = estimateEmbeddingDim(speed, time.lag = tau.acf,max.
embedding.dim = 15)
# estimating embedding dimension for lag tau.ami
emb.dimami = estimateEmbeddingDim(speed, time.lag = tau.ami,max.
embedding.dim = 15)
# phase space reconstruction
takami = buildTakens(speed,embedding.dim = emb.dimami, time.lag = tau.
ami)
takacf = buildTakens(speed,embedding.dim = emb.dimacf, time.lag = tau.
acf)
# plotting divergence points
mlami = maxLyapunov(speed,sampling.period=0.01,min.embedding.dim = emb.
dimami,
max.embedding.dim = emb.dimami +3,time.lag = tau.ami,
radius=1, max.time.steps=1000,
do.plot=TRUE)
# estimating maximum lyapunov exponent
ml.estami = estimate(ml, regression.range = c(0,3),do.plot = T,type="l
")
```

```

# plotting divergence points
mlacf = maxLyapunov(speed, sampling.period=0.01,min.embedding.dim = emb.
    dimacf, max.embedding.dim = emb.dimacf + 3, time.lag = tau.acf,
    radius, max.time.steps=1000, do.plot=TRUE)
# estimating maximum lyapunov exponent
ml.estacf = estimate(mlacf, regression.range = c(0,3),do.plot = T,type
    ="l")

```

A.6 CODE FOR LSTM 12 HRS MODEL IN PYTHON

```

from pandas import DataFrame
from pandas import Series
from pandas import concat
from pandas import read_csv
from pandas import datetime
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from math import sqrt
from matplotlib import pyplot
import numpy as np
import pandas as pd

# Hardcode all variables
batch_size_exp = 1
epoch_exp = 20

```

```

neurons_exp = 10
predict_values_exp = 12
lag_exp=24

from google.colab import drive
drive.mount('/content/drive')

# frame a sequence as a supervised learning problem
def timeseries_to_supervised(data, lag=1):
    df = DataFrame(data)
    columns = [df.shift(i) for i in range(1, lag+1)]
    columns.append(df)
    df = concat(columns, axis=1)
    df.fillna(0, inplace=True)
    return df

# create a differenced series
def difference(dataset, interval=1):
    diff = list()
    for i in range(interval, len(dataset)):
        value = dataset[i] - dataset[i - interval]
        diff.append(value)
    return Series(diff)

# invert differenced value
def inverse_difference(history, yhat, interval=1):
    return yhat + history[-interval]

# scale train and test data to [-1, 1]

```

```

def scale(train, test):
    # fit scaler
    scaler = MinMaxScaler(feature_range=(-1, 1))
    scaler = scaler.fit(train)
    # transform train
    train = train.reshape(train.shape[0], train.shape[1])
    train_scaled = scaler.transform(train)
    # transform test
    test = test.reshape(test.shape[0], test.shape[1])
    test_scaled = scaler.transform(test)
    return scaler, train_scaled, test_scaled

# inverse scaling for a forecasted value
def invert_scale(scaler, X, value):
    new_row = [x for x in X] + [value]
    array = np.array(new_row)
    array = array.reshape(1, len(array))
    inverted = scaler.inverse_transform(array)
    return inverted[0, -1]

# fit an LSTM network to training data
def fit_lstm(train, batch_size, nb_epoch, neurons):
    X, y = train[:, 0:-1], train[:, -1]
    X = X.reshape(X.shape[0], 1, X.shape[1])
    model = Sequential()
    model.add(LSTM(neurons, batch_input_shape=(batch_size, X.shape[1],
                                                X.shape[2]), stateful=True))
    model.add(Dense(1))
    model.compile(loss='mean_squared_error', optimizer='adam')

```

```

for i in range(nb_epoch):
    model.fit(X, y, epochs=1, batch_size=batch_size, verbose=1,
               shuffle=False)
    model.reset_states()
return model

# make a one-step forecast
def forecast_lstm(model, batch_size, X):
    X = X.reshape(1, 1, len(X))
    #print(X)
    yhat = model.predict(X, batch_size=1)
    return yhat[0,0]

import pandas as pd
series = pd.read_csv('/content/drive/My Drive/lstm5yr.csv', index_col="DateTime")
series.head()

'''Drop all the features as we will not be having any in production'''
del series['airtemp']
del series['pressure']
del series['winddirection']
del series['power']
series.head()

for i in range(0,10):
    series = series[:-1]
series.tail()

```

```

# transform data to be stationary
raw_values = series.values
diff_values = difference(raw_values, 1)

# transform data to be supervised learning
supervised = timeseries_to_supervised(diff_values, lag_exp)
supervised_values = supervised.values

# split data into train and test-sets
train, test = supervised_values[0:-predict_values_exp],
              supervised_values[-predict_values_exp:]

# transform the scale of the data
scaler, train_scaled, test_scaled = scale(train, test)

# fit the model
lstm_model = fit_lstm(train_scaled, batch_size_exp, epoch_exp,
                      neurons_exp)

# walk-forward validation on the test data
predictions = list()
expectations = list()
predictions_plot = list()
expectations_plot = list()
test_pred = list()
for i in range(len(test_scaled)):
    # make one-step forecast
    X, y = test_scaled[i, 0:-1], test_scaled[i, -1]

```

```

yhat = forecast_lstm(lstm_model, 1, X)#batch_size_exp to 1
'''# Start Debug prints
print("X: %", X)
print("yhat: %", yhat)
# End Debug prints'''
# Replacing value in test scaled with the predicted value.
test_pred = [yhat] + test_pred
if len(test_pred) > lag_exp+1:
    test_pred = test_pred[:-1]
if i+1<len(test_scaled):
    if i+1 > lag_exp+1:
        test_scaled[i+1] = test_pred
    else:
        test_scaled[i+1] = np.concatenate((test_pred, test_scaled[i
+1, i+1:])),axis=0)

# invert scaling
yhat = invert_scale(scaler, X, yhat)
# invert differencing
yhat = inverse_difference(raw_values, yhat, len(test_scaled)+1-i)
# store forecast
expected = raw_values[len(train) + i + 1]
predictions_plot.append(yhat)
expectations_plot.append(expected)
if expected != 0:
    predictions.append(yhat)
    expectations.append(expected)
print('Hour=%d, Predicted=%f, Expected=%f' % (i+1, yhat, expected))

```

```

# line plot of observed vs predicted
pyplot.figure(figsize=(20,8))
pyplot.plot(expectations_plot, label="True")
pyplot.plot(predictions_plot, label="Predicted")
pyplot.legend(loc='upper right')
pyplot.xlabel("Number of hours")
pyplot.ylabel("Wind speed")
pyplot.show()

expectations = np.array(expectations)
predictions = np.array(predictions)
print("Mean Absolute Percent Error: ", (np.mean(np.abs((expectations -
predictions) / expectations)) * 100))

from sklearn.metrics import explained_variance_score
print("expected variance score",explained_variance_score(expectations ,
predictions))

from sklearn.metrics import max_error
print("max error:",max_error(expectations , predictions))

from sklearn.metrics import mean_absolute_error
print("mean absolute error:",mean_absolute_error(expectations ,
predictions))

from sklearn.metrics import mean_squared_error
print("mean squared error:",mean_squared_error(expectations ,
predictions))

```

```

from sklearn.metrics import mean_squared_error
print("root mean squared error:",mean_squared_error(expectations,
predictions, squared=False))

from sklearn.metrics import mean_squared_log_error
print("mean squared log error:",mean_squared_log_error(predictions,
expectations))

from sklearn.metrics import median_absolute_error
print("median absolute error:",median_absolute_error(expectations,
predictions))

from sklearn.metrics import r2_score
print("r2 score:",r2_score(expectations, predictions))
print("r2 score variance weighted:",r2_score(expectations, predictions,
multioutput='variance_weighted'))

from sklearn.metrics import mean_poisson_deviance
print("mean poisson deviance:",mean_poisson_deviance(expectations,
predictions))

from sklearn.metrics import mean_gamma_deviance
print("mean gamma deviance:",mean_gamma_deviance(expectations,
predictions))

```

A.7 CODE FOR LSTM 24 HRS MODEL IN PYTHON

```
from pandas import DataFrame
from pandas import Series
from pandas import concat
from pandas import read_csv
from pandas import datetime
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from math import sqrt
from matplotlib import pyplot
import numpy as np
import pandas as pd

# Hardcode all variables
batch_size_exp = 1
epoch_exp = 20
neurons_exp = 10
predict_values_exp = 24
lag_exp=24

from google.colab import drive
drive.mount('/content/drive')

# frame a sequence as a supervised learning problem
def timeseries_to_supervised(data, lag=1):
    df = DataFrame(data)
    columns = [df.shift(i) for i in range(1, lag+1)]
```

```

columns.append(df)
df = concat(columns, axis=1)
df.fillna(0, inplace=True)
return df

# create a differenced series
def difference(dataset, interval=1):
    diff = list()
    for i in range(interval, len(dataset)):
        value = dataset[i] - dataset[i - interval]
        diff.append(value)
    return Series(diff)

# invert differenced value
def inverse_difference(history, yhat, interval=1):
    return yhat + history[-interval]

# scale train and test data to [-1, 1]
def scale(train, test):
    # fit scaler
    scaler = MinMaxScaler(feature_range=(-1, 1))
    scaler = scaler.fit(train)
    # transform train
    train = train.reshape(train.shape[0], train.shape[1])
    train_scaled = scaler.transform(train)
    # transform test
    test = test.reshape(test.shape[0], test.shape[1])
    test_scaled = scaler.transform(test)
    return scaler, train_scaled, test_scaled

```

```

# inverse scaling for a forecasted value
def invert_scale(scaler, X, value):
    new_row = [x for x in X] + [value]
    array = np.array(new_row)
    array = array.reshape(1, len(array))
    inverted = scaler.inverse_transform(array)
    return inverted[0, -1]

# fit an LSTM network to training data
def fit_lstm(train, batch_size, nb_epoch, neurons):
    X, y = train[:, 0:-1], train[:, -1]
    X = X.reshape(X.shape[0], 1, X.shape[1])
    model = Sequential()
    model.add(LSTM(neurons, batch_input_shape=(batch_size, X.shape[1],
                                                X.shape[2]), stateful=True))
    model.add(Dense(1))
    model.compile(loss='mean_squared_error', optimizer='adam')
    for i in range(nb_epoch):
        model.fit(X, y, epochs=1, batch_size=batch_size, verbose=1,
                  shuffle=False)
        model.reset_states()
    return model

# make a one-step forecast
def forecast_lstm(model, batch_size, X):
    X = X.reshape(1, 1, len(X))
    #print(X)
    yhat = model.predict(X, batch_size=1)

```

```

    return yhat[0,0]

import pandas as pd
series = pd.read_csv('/content/drive/My Drive/lstm5yr.csv',index_col="DateTime")
series.head()

'''Drop all the features as we will not be having any in production'''
del series['airtemp']
del series['pressure']
del series['winddirection']
del series['power']
series.head()

for i in range(0,10):
    series = series[:-1]
series.tail()

# transform data to be stationary
raw_values = series.values
diff_values = difference(raw_values, 1)

# transform data to be supervised learning
supervised = timeseries_to_supervised(diff_values, lag_exp)
supervised_values = supervised.values

# split data into train and test-sets
train, test = supervised_values[0:-predict_values_exp],
supervised_values[-predict_values_exp:]

```

```

# transform the scale of the data
scaler, train_scaled, test_scaled = scale(train, test)

# fit the model
lstm_model = fit_lstm(train_scaled, batch_size_exp, epoch_exp,
                      neurons_exp)

# walk-forward validation on the test data
predictions = list()
expectations = list()
predictions_plot = list()
expectations_plot = list()
test_pred = list()
for i in range(len(test_scaled)):
    # make one-step forecast
    X, y = test_scaled[i, 0:-1], test_scaled[i, -1]
    yhat = forecast_lstm(lstm_model, 1, X)#batch_size_exp to 1
    '''# Start Debug prints
    print("X: %", X)
    print("yhat: %", yhat)
    # End Debug prints'''
    # Replacing value in test scaled with the predicted value.
    test_pred = [yhat] + test_pred
    if len(test_pred) > lag_exp+1:
        test_pred = test_pred[:-1]
    if i+1<len(test_scaled):
        if i+1 > lag_exp+1:

```

```

        test_scaled[i+1] = test_pred
    else:
        test_scaled[i+1] = np.concatenate((test_pred, test_scaled[i
            +1, i+1:])), axis=0)

# invert scaling
yhat = invert_scale(scaler, X, yhat)
# invert differencing
yhat = inverse_difference(raw_values, yhat, len(test_scaled)+1-i)
# store forecast
expected = raw_values[len(train) + i + 1]
predictions_plot.append(yhat)
expectations_plot.append(expected)
if expected != 0:
    predictions.append(yhat)
    expectations.append(expected)
print('Hour=%d, Predicted=%f, Expected=%f' % (i+1, yhat, expected))

# line plot of observed vs predicted
pyplot.figure(figsize=(20,8))
pyplot.plot(expectations_plot, label="True")
pyplot.plot(predictions_plot, label="Predicted")
pyplot.legend(loc='upper right')
pyplot.xlabel("Number of hours")
pyplot.ylabel("Wind speed")
pyplot.show()

expectations = np.array(expectations)
predictions = np.array(predictions)

```

```
print("Mean Absolute Percent Error: ", (np.mean(np.abs((expectations -  
predictions) / expectations))*100))  
  
from sklearn.metrics import explained_variance_score  
print("expected variance score",explained_variance_score(expectations ,  
predictions))  
  
  
  
from sklearn.metrics import max_error  
print("max error:",max_error(expectations , predictions))  
  
  
from sklearn.metrics import mean_absolute_error  
print("mean absolute error:",mean_absolute_error(expectations ,  
predictions))  
  
  
from sklearn.metrics import mean_squared_error  
print("mean squared error:",mean_squared_error(expectations ,  
predictions))  
  
  
from sklearn.metrics import mean_squared_error  
print("root mean squared error:",mean_squared_error(expectations ,  
predictions , squared=False))  
  
  
  
from sklearn.metrics import mean_squared_log_error  
print("mean squared log error:",mean_squared_log_error(predictions ,  
expectations))  
  
  
from sklearn.metrics import median_absolute_error
```

```

print("median absolute error:", median_absolute_error(expectations,
predictions))

from sklearn.metrics import r2_score
print("r2 score:", r2_score(expectations, predictions))
print("r2 score variance weighted:", r2_score(expectations, predictions,
multioutput='variance_weighted'))

from sklearn.metrics import mean_poisson_deviance
print("mean poisson deviance:", mean_poisson_deviance(expectations,
predictions))

from sklearn.metrics import mean_gamma_deviance
print("mean gamma deviance:", mean_gamma_deviance(expectations,
predictions))

```

A.8 CODE FOR LSTM 2 DAYS MODEL IN PYTHON

```

from pandas import DataFrame
from pandas import Series
from pandas import concat
from pandas import read_csv
from pandas import datetime
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from math import sqrt

```

```

from matplotlib import pyplot
import numpy as np
import pandas as pd

# Hardcode all variables
batch_size_exp = 1
epoch_exp = 10
neurons_exp = 10
predict_values_exp = 48
lag_exp=24

from google.colab import drive
drive.mount('/content/drive')

# frame a sequence as a supervised learning problem
def timeseries_to_supervised(data, lag=1):
    df = DataFrame(data)
    columns = [df.shift(i) for i in range(1, lag+1)]
    columns.append(df)
    df = concat(columns, axis=1)
    df.fillna(0, inplace=True)
    return df

# create a differenced series
def difference(dataset, interval=1):
    diff = list()
    for i in range(interval, len(dataset)):
        value = dataset[i] - dataset[i - interval]
        diff.append(value)


```

```

    return Series(diff)

# invert differenced value
def inverse_difference(history, yhat, interval=1):
    return yhat + history[-interval]

# scale train and test data to [-1, 1]
def scale(train, test):
    # fit scaler
    scaler = MinMaxScaler(feature_range=(-1, 1))
    scaler = scaler.fit(train)
    # transform train
    train = train.reshape(train.shape[0], train.shape[1])
    train_scaled = scaler.transform(train)
    # transform test
    test = test.reshape(test.shape[0], test.shape[1])
    test_scaled = scaler.transform(test)
    return scaler, train_scaled, test_scaled

# inverse scaling for a forecasted value
def invert_scale(scaler, X, value):
    new_row = [x for x in X] + [value]
    array = np.array(new_row)
    array = array.reshape(1, len(array))
    inverted = scaler.inverse_transform(array)
    return inverted[0, -1]

# fit an LSTM network to training data
def fit_lstm(train, batch_size, nb_epoch, neurons):

```

```

X, y = train[:, 0:-1], train[:, -1]
X = X.reshape(X.shape[0], 1, X.shape[1])
model = Sequential()
model.add(LSTM(neurons, batch_input_shape=(batch_size, X.shape[1],
X.shape[2]), stateful=True))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
for i in range(nb_epoch):
    model.fit(X, y, epochs=1, batch_size=batch_size, verbose=1,
shuffle=False)
    model.reset_states()
return model

# make a one-step forecast
def forecast_lstm(model, batch_size, X):
    X = X.reshape(1, 1, len(X))
    #print(X)
    yhat = model.predict(X, batch_size=1)
    return yhat[0,0]

import pandas as pd
series = pd.read_csv('/content/drive/My Drive/lstm5yr.csv', index_col="DateTime")
series.head()

'''Drop all the features as we will not be having any in production'''
del series['airtemp']
del series['pressure']
del series['winddirection']

```

```

del series[ 'power' ]
series.head()

for i in range(0,10):
    series = series[:-1]
series.tail()

# transform data to be stationary
raw_values = series.values
diff_values = difference(raw_values, 1)

# transform data to be supervised learning
supervised = timeseries_to_supervised(diff_values, lag_exp)
supervised_values = supervised.values

# split data into train and test-sets
train, test = supervised_values[0:-predict_values_exp],
              supervised_values[-predict_values_exp:]

# transform the scale of the data
scaler, train_scaled, test_scaled = scale(train, test)

# fit the model
lstm_model = fit_lstm(train_scaled, batch_size_exp, epoch_exp,
                      neurons_exp)

# walk-forward validation on the test data
predictions = list()

```

```

expectations = list()
predictions_plot = list()
expectations_plot = list()
test_pred = list()
for i in range(len(test_scaled)):
    # make one-step forecast
    X, y = test_scaled[i, 0:-1], test_scaled[i, -1]
    yhat = forecast_lstm(lstm_model, 1, X)#batch_size_exp to 1
    '''# Start Debug prints
    print("X: %", X)
    print("yhat: %", yhat)
    # End Debug prints'''
    # Replacing value in test scaled with the predicted value.
    test_pred = [yhat] + test_pred
    if len(test_pred) > lag_exp+1:
        test_pred = test_pred[:-1]
    if i+1<len(test_scaled):
        if i+1 > lag_exp+1:
            test_scaled[i+1] = test_pred
        else:
            test_scaled[i+1] = np.concatenate((test_pred, test_scaled[i+1, i+1:])), axis=0)

    # invert scaling
    yhat = invert_scale(scaler, X, yhat)
    # invert differencing
    yhat = inverse_difference(raw_values, yhat, len(test_scaled)+1-i)
    # store forecast
    expected = raw_values[len(train) + i + 1]

```

```

predictions_plot.append(yhat)
expectations_plot.append(expected)
if expected != 0:
    predictions.append(yhat)
    expectations.append(expected)
print('Hour=%d, Predicted=%f, Expected=%f' % (i+1, yhat, expected))

# line plot of observed vs predicted
pyplot.figure(figsize=(20,8))
pyplot.plot(expectations_plot, label="True")
pyplot.plot(predictions_plot, label="Predicted")
pyplot.legend(loc='upper right')
pyplot.xlabel("Number of hours")
pyplot.ylabel("Wind speed")
pyplot.show()

expectations = np.array(expectations)
predictions = np.array(predictions)
print("Mean Absolute Percent Error: ", (np.mean(np.abs((expectations -
predictions) / expectations)) * 100))

from sklearn.metrics import explained_variance_score
print("expected variance score", explained_variance_score(expectations,
predictions))

from sklearn.metrics import max_error
print("max error:", max_error(expectations, predictions))

```

```
from sklearn.metrics import mean_absolute_error
print("mean absolute error:", mean_absolute_error(expectations,
predictions))

from sklearn.metrics import mean_squared_error
print("mean squared error:", mean_squared_error(expectations,
predictions))

from sklearn.metrics import mean_squared_error
print("root mean squared error:", mean_squared_error(expectations,
predictions, squared=False))

from sklearn.metrics import mean_squared_log_error
print("mean squared log error:", mean_squared_log_error(predictions,
expectations))

from sklearn.metrics import median_absolute_error
print("median absolute error:", median_absolute_error(expectations,
predictions))

from sklearn.metrics import r2_score
print("r2 score:", r2_score(expectations, predictions))
print("r2 score variance weighted:", r2_score(expectations, predictions,
multioutput='variance_weighted'))

from sklearn.metrics import mean_poisson_deviance
print("mean poisson deviance:", mean_poisson_deviance(expectations,
predictions))
```

```
from sklearn.metrics import mean_gamma_deviance
print("mean gamma deviance:", mean_gamma_deviance(expectations,
predictions))
```

A.9 CODE FOR LSTM 1 WEEK MODEL IN PYTHON

```
from pandas import DataFrame
from pandas import Series
from pandas import concat
from pandas import read_csv
from pandas import datetime
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from math import sqrt
from matplotlib import pyplot
import numpy as np
import pandas as pd

# Hardcode all variables
batch_size_exp = 1
epoch_exp = 20
neurons_exp = 10
predict_values_exp = 168
lag_exp=24
```

```

from google.colab import drive
drive.mount('/content/drive')

# frame a sequence as a supervised learning problem
def timeseries_to_supervised(data, lag=1):
    df = DataFrame(data)
    columns = [df.shift(i) for i in range(1, lag+1)]
    columns.append(df)
    df = concat(columns, axis=1)
    df.fillna(0, inplace=True)
    return df

# create a differenced series
def difference(dataset, interval=1):
    diff = list()
    for i in range(interval, len(dataset)):
        value = dataset[i] - dataset[i - interval]
        diff.append(value)
    return Series(diff)

# invert differenced value
def inverse_difference(history, yhat, interval=1):
    return yhat + history[-interval]

# scale train and test data to [-1, 1]
def scale(train, test):
    # fit scaler
    scaler = MinMaxScaler(feature_range=(-1, 1))
    scaler = scaler.fit(train)

```

```

# transform train
train = train.reshape(train.shape[0], train.shape[1])
train_scaled = scaler.transform(train)

# transform test
test = test.reshape(test.shape[0], test.shape[1])
test_scaled = scaler.transform(test)

return scaler, train_scaled, test_scaled

# inverse scaling for a forecasted value
def invert_scale(scaler, X, value):
    new_row = [x for x in X] + [value]
    array = np.array(new_row)
    array = array.reshape(1, len(array))
    inverted = scaler.inverse_transform(array)
    return inverted[0, -1]

# fit an LSTM network to training data
def fit_lstm(train, batch_size, nb_epoch, neurons):
    X, y = train[:, 0:-1], train[:, -1]
    X = X.reshape(X.shape[0], 1, X.shape[1])
    model = Sequential()
    model.add(LSTM(neurons, batch_input_shape=(batch_size, X.shape[1],
                                                X.shape[2]), stateful=True))
    model.add(Dense(1))
    model.compile(loss='mean_squared_error', optimizer='adam')
    for i in range(nb_epoch):
        model.fit(X, y, epochs=1, batch_size=batch_size, verbose=1,
                  shuffle=False)
    model.reset_states()

```

```

    return model

# make a one-step forecast
def forecast_lstm(model, batch_size, X):
    X = X.reshape(1, 1, len(X))
    #print(X)
    yhat = model.predict(X, batch_size=1)
    return yhat[0,0]

import pandas as pd
series = pd.read_csv('/content/drive/My Drive/lstm5yr.csv', index_col="DateTime")
series.head()

'''Drop all the features as we will not be having any in production'''
del series['airtemp']
del series['pressure']
del series['winddirection']
del series['power']
series.head()

for i in range(0,10):
    series = series[:-1]
series.tail()

# transform data to be stationary
raw_values = series.values
diff_values = difference(raw_values, 1)

```

```

# transform data to be supervised learning
supervised = timeseries_to_supervised(diff_values, lag_exp)
supervised_values = supervised.values

# split data into train and test-sets
train, test = supervised_values[0:-predict_values_exp],
              supervised_values[-predict_values_exp:]

# transform the scale of the data
scaler, train_scaled, test_scaled = scale(train, test)

# fit the model
lstm_model = fit_lstm(train_scaled, batch_size_exp, epoch_exp,
                      neurons_exp)

# walk-forward validation on the test data
predictions = list()
expectations = list()
predictions_plot = list()
expectations_plot = list()
test_pred = list()

for i in range(len(test_scaled)):
    # make one-step forecast
    X, y = test_scaled[i, 0:-1], test_scaled[i, -1]
    yhat = forecast_lstm(lstm_model, 1, X)#batch_size_exp to 1
    '''# Start Debug prints
    print("X: %", X)
    print("yhat: %", yhat)

```

```

# End Debug prints ''

# Replacing value in test scaled with the predicted value.
test_pred = [yhat] + test_pred
if len(test_pred) > lag_exp+1:
    test_pred = test_pred[:-1]
if i+1<len(test_scaled):
    if i+1 > lag_exp+1:
        test_scaled[i+1] = test_pred
    else:
        test_scaled[i+1] = np.concatenate((test_pred, test_scaled[i
+1, i+1:])), axis=0)

# invert scaling
yhat = invert_scale(scaler, X, yhat)
# invert differencing
yhat = inverse_difference(raw_values, yhat, len(test_scaled)+1-i)
# store forecast
expected = raw_values[len(train) + i + 1]
predictions_plot.append(yhat)
expectations_plot.append(expected)
if expected != 0:
    predictions.append(yhat)
    expectations.append(expected)
print('Hour=%d, Predicted=%f, Expected=%f' % (i+1, yhat, expected))

# line plot of observed vs predicted
pyplot.figure(figsize=(20,8))
pyplot.plot(expectations_plot, label="True")
pyplot.plot(predictions_plot, label="Predicted")

```

```

pyplot.legend(loc='upper right')
pyplot.xlabel("Number of hours")
pyplot.ylabel("Wind speed")
pyplot.show()

expectations = np.array(expectations)
predictions = np.array(predictions)
print("Mean Absolute Percent Error: ", (np.mean(np.abs((expectations -
predictions) / expectations)) * 100))

from sklearn.metrics import explained_variance_score
print("expected variance score", explained_variance_score(expectations ,
predictions))

from sklearn.metrics import max_error
print("max error:", max_error(expectations , predictions))

from sklearn.metrics import mean_absolute_error
print("mean absolute error:", mean_absolute_error(expectations ,
predictions))

from sklearn.metrics import mean_squared_error
print("mean squared error:", mean_squared_error(expectations ,
predictions))

from sklearn.metrics import mean_squared_error
print("root mean squared error:", mean_squared_error(expectations ,
predictions , squared=False))

```

```

from sklearn.metrics import mean_squared_log_error
print("mean squared log error:", mean_squared_log_error(predictions ,
expectations))

from sklearn.metrics import median_absolute_error
print("median absolute error:", median_absolute_error(expectations ,
predictions))

from sklearn.metrics import r2_score
print("r2 score:", r2_score(expectations , predictions))
print("r2 score variance weighted:", r2_score(expectations , predictions ,
multioutput='variance_weighted'))

from sklearn.metrics import mean_poisson_deviance
print("mean poisson deviance:", mean_poisson_deviance(expectations ,
predictions))

from sklearn.metrics import mean_gamma_deviance
print("mean gamma deviance:", mean_gamma_deviance(expectations ,
predictions))

```

A.10 CODE FOR LSTM 1 MONTH MODEL IN PYTHON

```

from pandas import DataFrame
from pandas import Series
from pandas import concat
from pandas import read_csv

```

```

from pandas import datetime
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from math import sqrt
from matplotlib import pyplot
import numpy as np
import pandas as pd

from google.colab import drive
drive.mount('/content/drive')

# Hardcode all variables
batch_size_exp = 1
epoch_exp = 20
neurons_exp = 10
predict_values_exp = 672
lag_exp=24

# frame a sequence as a supervised learning problem
def timeseries_to_supervised(data, lag=1):
    df = DataFrame(data)
    columns = [df.shift(i) for i in range(1, lag+1)]
    columns.append(df)
    df = concat(columns, axis=1)

```

```

df.fillna(0, inplace=True)
return df

# create a differenced series
def difference(dataset, interval=1):
    diff = list()
    for i in range(interval, len(dataset)):
        value = dataset[i] - dataset[i - interval]
        diff.append(value)
    return Series(diff)

# invert differenced value
def inverse_difference(history, yhat, interval=1):
    return yhat + history[-interval]

# scale train and test data to [-1, 1]
def scale(train, test):
    # fit scaler
    scaler = MinMaxScaler(feature_range=(-1, 1))
    scaler = scaler.fit(train)
    # transform train
    train = train.reshape(train.shape[0], train.shape[1])
    train_scaled = scaler.transform(train)
    # transform test
    test = test.reshape(test.shape[0], test.shape[1])
    test_scaled = scaler.transform(test)
    return scaler, train_scaled, test_scaled

# inverse scaling for a forecasted value

```

```

def invert_scale(scaler, X, value):
    new_row = [x for x in X] + [value]
    array = np.array(new_row)
    array = array.reshape(1, len(array))
    inverted = scaler.inverse_transform(array)
    return inverted[0, -1]

# fit an LSTM network to training data
def fit_lstm(train, batch_size, nb_epoch, neurons):
    X, y = train[:, 0:-1], train[:, -1]
    X = X.reshape(X.shape[0], 1, X.shape[1])
    model = Sequential()
    model.add(LSTM(neurons, batch_input_shape=(batch_size, X.shape[1]),
                  X.shape[2]), stateful=True))
    model.add(Dense(1))
    model.compile(loss='mean_squared_error', optimizer='adam')
    for i in range(nb_epoch):
        model.fit(X, y, epochs=1, batch_size=batch_size, verbose=1,
                  shuffle=False)
        model.reset_states()
    return model

# make a one-step forecast
def forecast_lstm(model, batch_size, X):
    X = X.reshape(1, 1, len(X))
    #print(X)
    yhat = model.predict(X, batch_size=1)
    return yhat[0,0]

```

```

import pandas as pd
series = pd.read_csv('/content/drive/My Drive/lstm5yr.csv', index_col="DateTime")
series.head()

'''Drop all the features as we will not be having any in production'''
del series['airtemp']
del series['pressure']
del series['winddirection']
del series['power']
series.head()

for i in range(0,10):
    series = series[:-1]
series.tail()

# transform data to be stationary
raw_values = series.values
diff_values = difference(raw_values, 1)

# transform data to be supervised learning
supervised = timeseries_to_supervised(diff_values, lag_exp)
supervised_values = supervised.values

# split data into train and test-sets
train, test = supervised_values[0:-predict_values_exp],
              supervised_values[-predict_values_exp:]

```

```

print(len(train))
print(len(test))

# transform the scale of the data
scaler, train_scaled, test_scaled = scale(train, test)

# fit the model
lstm_model = fit_lstm(train_scaled, batch_size_exp, epoch_exp,
                      neurons_exp)

# walk-forward validation on the test data
predictions = list()
expectations = list()
predictions_plot = list()
expectations_plot = list()
test_pred = list()

for i in range(len(test_scaled)):
    # make one-step forecast
    X, y = test_scaled[i, 0:-1], test_scaled[i, -1]
    yhat = forecast_lstm(lstm_model, 1, X)#batch_size_exp to 1
    '''# Start Debug prints
    print("X: %", X)
    print("yhat: %", yhat)
    # End Debug prints'''
    # Replacing value in test scaled with the predicted value.
    test_pred = [yhat] + test_pred
    if len(test_pred) > lag_exp+1:
        test_pred = test_pred[:-1]

```

```

if i+1<len(test_scaled):
    if i+1 > lag_exp+1:
        test_scaled[i+1] = test_pred
    else:
        test_scaled[i+1] = np.concatenate((test_pred, test_scaled[i
+1, i+1:])), axis=0)

# invert scaling
yhat = invert_scale(scaler, X, yhat)
# invert differencing
yhat = inverse_difference(raw_values, yhat, len(test_scaled)+1-i)
# store forecast
expected = raw_values[len(train) + i + 1]
predictions_plot.append(yhat)
expectations_plot.append(expected)
if expected != 0:
    predictions.append(yhat)
    expectations.append(expected)
print('Hour=%d, Predicted=%f, Expected=%f' % (i+1, yhat, expected))

# line plot of observed vs predicted
pyplot.figure(figsize=(20,8))
pyplot.plot(expectations_plot, label="True")
pyplot.plot(predictions_plot, label="Predicted")
pyplot.legend(loc='upper right')
pyplot.xlabel("Number of hours")
pyplot.ylabel("Wind speed")
pyplot.show()

```

```

expectations = np.array(expectations)
predictions = np.array(predictions)
print("Mean Absolute Percent Error: ", (np.mean(np.abs((expectations -
predictions) / expectations)) * 100))

from sklearn.metrics import explained_variance_score
print("expected variance score", explained_variance_score(expectations,
predictions))

from sklearn.metrics import max_error
print("max error:", max_error(expectations, predictions))

from sklearn.metrics import mean_absolute_error
print("mean absolute error:", mean_absolute_error(expectations,
predictions))

from sklearn.metrics import mean_squared_error
print("mean squared error:", mean_squared_error(expectations,
predictions))

from sklearn.metrics import mean_squared_error
print("root mean squared error:", mean_squared_error(expectations,
predictions, squared=False))

from sklearn.metrics import mean_squared_log_error
print("mean squared log error:", mean_squared_log_error(predictions,
expectations))

```

```

from sklearn.metrics import median_absolute_error
print("median absolute error:", median_absolute_error(expectations,
predictions))

from sklearn.metrics import r2_score
print("r2 score:", r2_score(expectations, predictions))
print("r2 score variance weighted:", r2_score(expectations, predictions,
multioutput='variance_weighted'))

from sklearn.metrics import mean_poisson_deviance
print("mean poisson deviance:", mean_poisson_deviance(expectations,
predictions))

from sklearn.metrics import mean_gamma_deviance
print("mean gamma deviance:", mean_gamma_deviance(expectations,
predictions))

```

A.11 CODE FOR 1 YEAR STACKED MODEL WITH TEST TRAIN RATIO OF 0.1 IN PYTHON

```

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import xgboost
import math
from scipy.stats import pearsonr
from sklearn.model_selection import cross_validate

```

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import explained_variance_score
from sklearn.linear_model import LinearRegression
regr = LinearRegression()
import warnings
warnings.filterwarnings("ignore")
import seaborn as sns

from google.colab import drive
drive.mount('/content/drive')

# New Section

data = pd.read_csv("/content/drive/My Drive/one_year - Sheet1.csv")
data.head()

sorted(data.columns)

print(data.shape)
print(len(data))
print(len(data.columns))
print(data.dtypes.unique())
print(data.dtypes)

data.select_dtypes(include=['O']).columns.tolist()

print(data.isnull().any().sum(), ' / ', len(data.columns))
```

```

print(data.isnull().any(axis=1).sum(), ' / ', len(data))

features = data.iloc[:, 2: ].columns.tolist()
target = data.iloc[:, 1].name

str_list = []
for colname, colvalue in data.iteritems():
    if type(colvalue[1]) == str:
        str_list.append(colname)
num_list = data.columns.difference(str_list)
house_num = data[num_list]
f, ax = plt.subplots(figsize=(16, 12))
plt.title('Pearson Correlation of features')
sns.heatmap(house_num.astype(float).corr(), linewidths=0.25, vmax=1,
            square=True, cmap="PuBuGn", linecolor='k', annot=True)

correlation={}
for f in features:
    data1 = data[[f, target]]
    x1= data1[f].values
    x2= data1[target].values
    key = f + ' vs ' + target
    correlation[key] = pearsonr(x1,x2)[0]

data_correlations = pd.DataFrame(correlation, index=['Value']).T
data_correlations.loc[data_correlations['Value'].abs().sort_values(
    ascending=False).index]

a = data[['wind_speed']]

```

```

print(a)
b = data[ 'power' ]
print(b)

X = a.values
y = b.values

X_train , X_test , y_train , y_test=train_test_split(X,y, test_size=0.1)

regr.fit(X_train , y_train)
print(regr.predict(X_test))

regr.score(X_test , y_test)

print("RMSE: %.2f"
      % math.sqrt(np.mean((regr.predict(X_test) - y_test) ** 2)))

xgb = xgboost.XGBRegressor(n_estimators=100, learning_rate=0.08, gamma
=0, subsample=0.75,
                           colsample_bytree=1, max_depth=7)

traindf , testdf = train_test_split(X_train , test_size = 0.3)
xgb.fit(X_train , y_train)

predictions = xgb.predict(X_test)
print(explained_variance_score(predictions , y_test))

print("RMSE: %.2f"
      % math.sqrt(np.mean((xgb.predict(X_test) - y_test) ** 2)))

```

```

from sklearn.model_selection import GridSearchCV
from sklearn.metrics import r2_score

def algorithm_pipeline(X_train, X_test, y_train, y_test,
                      model, param_grid, cv=10, scoring_fit='
                        neg_mean_squared_error',
                      scoring_test=r2_score, do_probabilities = False)
                      :
    gs = GridSearchCV(
        estimator=model,
        param_grid=param_grid,
        cv=cv,
        n_jobs=-1,
        scoring=scoring_fit ,
        verbose=2
    )
    fitted_model = gs.fit(X_train, y_train)
    best_model = fitted_model.best_estimator_

    if do_probabilities:
        pred = fitted_model.predict_proba(X_test)
    else:
        pred = fitted_model.predict(X_test)

    score = scoring_test(y_test, pred)

    return [best_model, pred, score]

```

```

from sklearn.ensemble import RandomForestRegressor
from lightgbm import LGBMRegressor
from xgboost import XGBRegressor

models_to_train = [XGBRegressor() , LGBMRegressor() ,
RandomForestRegressor()]

grid_parameters = [
    { # XGBoost
        'n_estimators': [400, 700, 1000],
        'colsample_bytree': [0.7, 0.8],
        'max_depth': [15,20,25],
        'reg_alpha': [1.1, 1.2, 1.3],
        'reg_lambda': [1.1, 1.2, 1.3],
        'subsample': [0.7, 0.8, 0.9]
    },
    { # LightGBM
        'n_estimators': [400, 700, 1000],
        'learning_rate': [0.12],
        'colsample_bytree': [0.7, 0.8],
        'max_depth': [4],
        'num_leaves': [10, 20],
        'reg_alpha': [1.1, 1.2],
        'reg_lambda': [1.1, 1.2],
        'min_split_gain': [0.3, 0.4],
        'subsample': [0.8, 0.9],
        'subsample_freq': [10, 20]
    },
    { # Random Forest

```

```

'max_depth':[3 , 5 , 10, 13],
'n_estimators':[100 , 200, 400, 600, 900],
'max_features':[1,2,3,4,5,6]
}

]

models_preds_scores = []

for i , model in enumerate(models_to_train):
    params = grid_parameters[i]

    result = algorithm_pipeline(X_train , X_test , y_train , y_test ,
                                model, params, cv=5)
    models_preds_scores.append(result)

for result in models_preds_scores:
    print('Model: {0} , Score: {1}'.format(type(result[0]).__name__,
                                             result[2]))


from mlxtend.regressor import StackingCVRegressor
from sklearn.linear_model import Ridge, Lasso
from sklearn.svm import SVR

xgb = XGBRegressor()
lgbm = LGBMRegressor()
rf = RandomForestRegressor()
ridge = Ridge()

```

```

lasso = Lasso()
svr = SVR(kernel='linear')
stack = StackingCVRegressor(regressors=(rf, lgbm, xgb),
                            meta_regressor=lgbm, cv=12,
                            use_features_in_secondary=True,
                            store_train_meta_features=True,
                            shuffle=False,
                            random_state=42)

stack.fit(X_train, y_train)

pred = stack.predict(X_test)
score = r2_score(y_test, pred)
print(score)

from sklearn.metrics import explained_variance_score
print("Explained variance score :" ,explained_variance_score(y_test ,
pred))

from sklearn.metrics import max_error
print("Max Error :" ,max_error(y_test , pred))

from sklearn.metrics import mean_absolute_error
print("Mean Absolute Error:" ,mean_absolute_error(y_test , pred))

from sklearn.metrics import mean_squared_error

```

```

print("Mean Squared Error:",mean_squared_error(y_test, pred))

from sklearn.metrics import mean_squared_error
print("Root Mean Squared Error:",mean_squared_error(y_test, pred,
squared=False))

from sklearn.metrics import mean_squared_log_error
print("Mean Squared Log Error:",mean_squared_log_error(y_test,pred))

from sklearn.metrics import median_absolute_error
print("Median Absolute Error:",median_absolute_error(y_test, pred))

from sklearn.metrics import r2_score
print("R2 score:",r2_score(y_test, pred))
print("R2 score variance weighted:",r2_score(y_test, pred, multioutput
='variance_weighted'))

from sklearn.metrics import mean_poisson_deviance
print("Mean Poisson Deviance:",mean_poisson_deviance(y_test,pred))

plt.plot(y_test, label="True")
plt.plot(pred, label="Predicted")
plt.legend(loc='upper right')
plt.xlabel("Number of hours")
plt.ylabel("Power generated by system (kW)")
plt.show()

```

A.12 CODE FOR 1 YEAR STACKED MODEL WITH TEST TRAIN RATIO OF 0.2 IN PYTHON

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import xgboost
import math
from scipy.stats import pearsonr
from sklearn.model_selection import cross_validate
from sklearn.model_selection import train_test_split
from sklearn.metrics import explained_variance_score
from sklearn.linear_model import LinearRegression
regr = LinearRegression()
import warnings
warnings.filterwarnings("ignore")
import seaborn as sns

from google.colab import drive
drive.mount('/content/drive')

# New Section

data = pd.read_csv("/content/drive/My Drive/one_year – Sheet1.csv")
data.head()

sorted(data.columns)
```

```

print(data.shape)
print(len(data))
print(len(data.columns))
print(data.dtypes.unique())
print(data.dtypes)

data.select_dtypes(include=['O']).columns.tolist()

print(data.isnull().any().sum(), ' / ', len(data.columns))
print(data.isnull().any(axis=1).sum(), ' / ', len(data))

features = data.iloc[:,2:].columns.tolist()
target = data.iloc[:,1].name

str_list = []
for colname, colvalue in data.iteritems():
    if type(colvalue[1]) == str:
        str_list.append(colname)
num_list = data.columns.difference(str_list)
house_num = data[num_list]
f, ax = plt.subplots(figsize=(16, 12))
plt.title('Pearson Correlation of features')
sns.heatmap(house_num.astype(float).corr(), linewidths=0.25,vmax=1,
            square=True, cmap="PuBuGn", linecolor='k', annot=True)

correlation={}
for f in features:
    data1 = data[[f,target]]
    x1= data1[f].values

```

```

x2= data1[target].values
key = f + ' vs ' + target
correlation[key] = pearsonr(x1,x2)[0]

data_correlations = pd.DataFrame(correlation, index=['Value']).T
data_correlations.loc[data_correlations['Value'].abs().sort_values(
    ascending=False).index]

a = data[['wind_speed']]
print(a)
b = data['power']
print(b)

X = a.values
y = b.values

X_train, X_test, y_train, y_test=train_test_split(X,y, test_size=0.2)

regr.fit(X_train, y_train)
print(regr.predict(X_test))

regr.score(X_test, y_test)

print("RMSE: %.2f"
      % math.sqrt(np.mean((regr.predict(X_test) - y_test) ** 2)))

xgb = xgboost.XGBRegressor(n_estimators=100, learning_rate=0.08, gamma
=0, subsample=0.75,
                           colsample_bytree=1, max_depth=7)

```

```

traindf, testdf = train_test_split(X_train, test_size = 0.3)

xgb.fit(X_train, y_train)

predictions = xgb.predict(X_test)
print(explained_variance_score(predictions, y_test))

print("RMSE: %.2f"
      % math.sqrt(np.mean((xgb.predict(X_test) - y_test) ** 2)))

from sklearn.model_selection import GridSearchCV
from sklearn.metrics import r2_score

def algorithm_pipeline(X_train, X_test, y_train, y_test,
                      model, param_grid, cv=10, scoring_fit='
                        neg_mean_squared_error',
                      scoring_test=r2_score, do_probabilities = False):
    :
    gs = GridSearchCV(
        estimator=model,
        param_grid=param_grid,
        cv=cv,
        n_jobs=-1,
        scoring=scoring_fit,
        verbose=2
    )
    fitted_model = gs.fit(X_train, y_train)
    best_model = fitted_model.best_estimator_

```

```

if do_probabilities:
    pred = fitted_model.predict_proba(X_test)
else:
    pred = fitted_model.predict(X_test)

score = scoring_test(y_test, pred)

return [best_model, pred, score]

from sklearn.ensemble import RandomForestRegressor
from lightgbm import LGBMRegressor
from xgboost import XGBRegressor

models_to_train = [XGBRegressor(), LGBMRegressor(),
RandomForestRegressor()]

grid_parameters = [
    { # XGBoost
        'n_estimators': [400, 700, 1000],
        'colsample_bytree': [0.7, 0.8],
        'max_depth': [15,20,25],
        'reg_alpha': [1.1, 1.2, 1.3],
        'reg_lambda': [1.1, 1.2, 1.3],
        'subsample': [0.7, 0.8, 0.9]
    },
    { # LightGBM
        'n_estimators': [400, 700, 1000],
        'learning_rate': [0.12],
        'colsample_bytree': [0.7, 0.8],
    }
]

```

```

'max_depth': [4],
'num_leaves': [10, 20],
'reg_alpha': [1.1, 1.2],
'reg_lambda': [1.1, 1.2],
'min_split_gain': [0.3, 0.4],
'subsample': [0.8, 0.9],
'subsample_freq': [10, 20]
} ,
{ # Random Forest
'max_depth':[3, 5, 10, 13],
'n_estimators':[100, 200, 400, 600, 900],
'max_features':[1,2,3,4,5,6]
}
]

models_preds_scores = []

for i, model in enumerate(models_to_train):
    params = grid_parameters[i]

    result = algorithm_pipeline(X_train, X_test, y_train, y_test,
                                model, params, cv=5)
    models_preds_scores.append(result)

for result in models_preds_scores:
    print('Model: {} , Score: {}'.format(type(result[0]).__name__,
                                           result[2]))

```

```
from mlxtend.regressor import StackingCVRegressor
from sklearn.linear_model import Ridge, Lasso
from sklearn.svm import SVR

xgb = XGBRegressor()
lgbm = LGBMRegressor()
rf = RandomForestRegressor()
ridge = Ridge()
lasso = Lasso()
svr = SVR(kernel='linear')
stack = StackingCVRegressor(regressors=(rf, lgbm, xgb),
                            meta_regressor=lgbm, cv=12,
                            use_features_in_secondary=True,
                            store_train_meta_features=True,
                            shuffle=False,
                            random_state=42)

stack.fit(X_train, y_train)

pred = stack.predict(X_test)
score = r2_score(y_test, pred)
print(score)

from sklearn.metrics import explained_variance_score
print("Explained variance score : ", explained_variance_score(y_test,
                                                               pred))
```

```

from sklearn.metrics import max_error
print("Max Error :" ,max_error(y_test , pred))

from sklearn.metrics import mean_absolute_error
print("Mean Absolute Error:" ,mean_absolute_error(y_test , pred))

from sklearn.metrics import mean_squared_error
print("Mean Squared Error:" ,mean_squared_error(y_test , pred))

from sklearn.metrics import mean_squared_error
print("Root Mean Squared Error:" ,mean_squared_error(y_test , pred ,
 squared=False))

from sklearn.metrics import mean_squared_log_error
print("Mean Squared Log Error:" ,mean_squared_log_error(y_test , pred))

from sklearn.metrics import median_absolute_error
print("Median Absolute Error:" ,median_absolute_error(y_test , pred))

from sklearn.metrics import r2_score
print("R2 score:" ,r2_score(y_test , pred))
print("R2 score variance weighted:" ,r2_score(y_test , pred , multioutput
 ='variance_weighted' ))

from sklearn.metrics import mean_poisson_deviance
print("Mean Poisson Deviance:" ,mean_poisson_deviance(y_test , pred))

```

```

plt.plot(y_test, label="True")
plt.plot(pred, label="Predicted")
plt.legend(loc='upper right')
plt.xlabel("Number of hours")
plt.ylabel("Power generated by system (kW)")
plt.show()

```

A.13 CODE FOR 1 YEAR STACKED MODEL WITH TEST TRAIN RATIO OF 0.3 IN PYTHON

```

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import xgboost
import math
from scipy.stats import pearsonr
from sklearn.model_selection import cross_validate
from sklearn.model_selection import train_test_split
from sklearn.metrics import explained_variance_score
from sklearn.linear_model import LinearRegression
regr = LinearRegression()
import warnings
warnings.filterwarnings("ignore")
import seaborn as sns

from google.colab import drive
drive.mount('/content/drive')

```

```

# New Section

data = pd.read_csv("/content/drive/My Drive/one_year – Sheet1.csv")
data.head()

sorted(data.columns)

print(data.shape)
print(len(data))
print(len(data.columns))
print(data.dtypes.unique())
print(data.dtypes)

data.select_dtypes(include=['O']).columns.tolist()

print(data.isnull().any().sum(), ' / ', len(data.columns))
print(data.isnull().any(axis=1).sum(), ' / ', len(data))

features = data.iloc[:,2:].columns.tolist()
target = data.iloc[:,1].name

str_list = []
for colname, colvalue in data.iteritems():
    if type(colvalue[1]) == str:
        str_list.append(colname)
num_list = data.columns.difference(str_list)
house_num = data[num_list]

```

```

f , ax = plt.subplots(figsize=(16, 12))
plt.title('Pearson Correlation of features')
sns.heatmap(house_num.astype(float).corr(), linewidths=0.25,vmax=1,
            square=True, cmap="PuBuGn", linecolor='k', annot=True)

correlation={}
for f in features:
    data1 = data[[f,target]]
    x1= data1[f].values
    x2= data1[target].values
    key = f + ' vs ' + target
    correlation[key] = pearsonr(x1,x2)[0]

data_correlations = pd.DataFrame(correlation, index=['Value']).T
data_correlations.loc[data_correlations['Value'].abs().sort_values(
    ascending=False).index]

a = data[['wind_speed']]
print(a)
b = data['power']
print(b)

X = a.values
y = b.values

X_train, X_test, y_train, y_test=train_test_split(X,y, test_size=0.3)

regr.fit(X_train, y_train)
print(regr.predict(X_test))

```

```

regr.score(X_test,y_test)

print("RMSE: %.2f"
      % math.sqrt(np.mean((regr.predict(X_test) - y_test) ** 2)))

xgb = xgboost.XGBRegressor(n_estimators=100, learning_rate=0.08, gamma
=0, subsample=0.75,
                           colsample_bytree=1, max_depth=7)

traindf, testdf = train_test_split(X_train, test_size = 0.3)
xgb.fit(X_train,y_train)

predictions = xgb.predict(X_test)
print(explained_variance_score(predictions,y_test))

print("RMSE: %.2f"
      % math.sqrt(np.mean((xgb.predict(X_test) - y_test) ** 2)))

from sklearn.model_selection import GridSearchCV
from sklearn.metrics import r2_score

def algorithm_pipeline(X_train, X_test, y_train, y_test,
                      model, param_grid, cv=10, scoring_fit='
                      neg_mean_squared_error',
                      scoring_test=r2_score, do_probabilities = False)
:
    gs = GridSearchCV(
        estimator=model,

```

```

        param_grid=param_grid ,
        cv=cv ,
        n_jobs=-1,
        scoring=scoring_fit ,
        verbose=2
    )
fitted_model = gs.fit(X_train , y_train)
best_model = fitted_model.best_estimator_

if do_probabilities:
    pred = fitted_model.predict_proba(X_test)
else:
    pred = fitted_model.predict(X_test)

score = scoring_test(y_test , pred)

return [best_model , pred , score]

```

```

from sklearn.ensemble import RandomForestRegressor
from lightgbm import LGBMRegressor
from xgboost import XGBRegressor

```

```

models_to_train = [XGBRegressor() , LGBMRegressor() ,
RandomForestRegressor()]

```

```

grid_parameters = [
    { # XGBoost
        'n_estimators': [400, 700, 1000],
        'colsample_bytree': [0.7, 0.8],

```

```

'max_depth': [15,20,25],
'reg_alpha': [1.1, 1.2, 1.3],
'reg_lambda': [1.1, 1.2, 1.3],
'subsample': [0.7, 0.8, 0.9]
} ,
{ # LightGBM
  'n_estimators': [400, 700, 1000],
  'learning_rate': [0.12],
  'colsample_bytree': [0.7, 0.8],
  'max_depth': [4],
  'num_leaves': [10, 20],
  'reg_alpha': [1.1, 1.2],
  'reg_lambda': [1.1, 1.2],
  'min_split_gain': [0.3, 0.4],
  'subsample': [0.8, 0.9],
  'subsample_freq': [10, 20]
} ,
{ # Random Forest
  'max_depth':[3, 5, 10, 13],
  'n_estimators':[100, 200, 400, 600, 900],
  'max_features':[1,2,3,4,5,6]
}
]

models_preds_scores = []

for i, model in enumerate(models_to_train):

```

```

params = grid_parameters[i]

result = algorithm_pipeline(X_train, X_test, y_train, y_test,
                           model, params, cv=5)
models_preds_scores.append(result)

for result in models_preds_scores:
    print('Model: {} , Score: {}'.format(type(result[0]).__name__,
                                           result[2]))


from mlxtend.regressor import StackingCVRegressor
from sklearn.linear_model import Ridge, Lasso
from sklearn.svm import SVR

xgb = XGBRegressor()
lgbm = LGBMRegressor()
rf = RandomForestRegressor()
ridge = Ridge()
lasso = Lasso()
svr = SVR(kernel='linear')
stack = StackingCVRegressor(regressors=(rf, lgbm, xgb),
                            meta_regressor=lgbm, cv=12,
                            use_features_in_secondary=True,
                            store_train_meta_features=True,
                            shuffle=False,
                            random_state=42)

stack.fit(X_train, y_train)

```

```
pred = stack.predict(X_test)
score = r2_score(y_test, pred)
print(score)

from sklearn.metrics import explained_variance_score
print("Explained variance score :" ,explained_variance_score(y_test ,
pred))

from sklearn.metrics import max_error
print("Max Error :" ,max_error(y_test , pred))

from sklearn.metrics import mean_absolute_error
print("Mean Absolute Error:" ,mean_absolute_error(y_test , pred))

from sklearn.metrics import mean_squared_error
print("Mean Squared Error:" ,mean_squared_error(y_test , pred))

from sklearn.metrics import mean_squared_error
print("Root Mean Squared Error:" ,mean_squared_error(y_test , pred ,
squared=False))

from sklearn.metrics import mean_squared_log_error
print("Mean Squared Log Error:" ,mean_squared_log_error(y_test , pred))

from sklearn.metrics import median_absolute_error
print("Median Absolute Error:" ,median_absolute_error(y_test , pred))
```

```

from sklearn.metrics import r2_score
print("R2 score:",r2_score(y_test, pred))
print("R2 score variance weighted:",r2_score(y_test, pred, multioutput
='variance_weighted')))

from sklearn.metrics import mean_poisson_deviance
print("Mean Poisson Deviance:",mean_poisson_deviance(y_test,pred))

plt.plot(y_test, label="True")
plt.plot(pred, label="Predicted")
plt.legend(loc='upper right')
plt.xlabel("Number of hours")
plt.ylabel("Power generated by system (kW)")
plt.show()

```

A.14 CODE FOR LSTM UNIDIRECTIONAL 1 AND 2 LAYER MODEL IN PYTHON

```

from google.colab import drive
drive.mount('/content/drive')

import pandas as pd
import numpy as np
import time

%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns

```

```
sns.set_style("darkgrid")

from keras.models import Model
from keras.layers import Input, LSTM, Dense, LSTMCell, RNN,
    Bidirectional, concatenate
from keras.optimizers import Adam

from sklearn.preprocessing import MinMaxScaler

data = pd.read_csv(r"/content/drive/My Drive/lstm.csv", usecols =[3])
data.head()

scaler = MinMaxScaler(feature_range=(-1,1))
data =scaler.fit_transform(data.values.reshape(-1,1))

x_train = data[:-100]
y = data[-100:]

input_seq_len = 60
output_seq_len = 20
n_in_features = 1
n_out_features = 1
batch_size = 10

def generate_train_sequences(data):
```

```

total_start_points = len(data) - input_seq_len - output_seq_len
start_x_idx = np.random.choice(range(total_start_points),
                               total_start_points, replace = False)

input_batch_idxs = [(range(i, i+input_seq_len)) for i in
                    start_x_idx]
input_seq = np.take(data, input_batch_idxs, axis = 0)

output_batch_idxs = [(range(i+input_seq_len, i+input_seq_len+
                           output_seq_len)) for i in start_x_idx]
output_seq = np.take(data, output_batch_idxs, axis = 0)

input_seq =(input_seq.reshape(input_seq.shape[0],input_seq.shape
                             [1],n_in_features))
output_seq=(output_seq.reshape(output_seq.shape[0],output_seq.shape
                                [1],n_out_features))

return input_seq, output_seq

```

```

def create_model(layers, bidirectional=False):

    n_layers = len(layers)

    ## Encoder
    encoder_inputs = Input(shape=(None, n_in_features))
    lstm_cells = [LSTMCell(hidden_dim) for hidden_dim in layers]
    if bidirectional:
        encoder = Bidirectional(RNN(lstm_cells, return_state=True))

```

```

encoder_outputs_and_states = encoder(encoder_inputs)
bi_encoder_states = encoder_outputs_and_states[1:]
encoder_states = []
for i in range(int(len(bi_encoder_states)/2)):
    temp = concatenate([bi_encoder_states[i], bi_encoder_states
[2*n_layers + i]], axis=-1)
    encoder_states.append(temp)
else:
    encoder = RNN(lstm_cells, return_state=True)
    encoder_outputs_and_states = encoder(encoder_inputs)
    encoder_states = encoder_outputs_and_states[1:]

## Decoder
decoder_inputs = Input(shape=(None, n_out_features))
if bidirectional:
    decoder_cells = [LSTMCell(hidden_dim*2) for hidden_dim in
layers]
else:
    decoder_cells = [LSTMCell(hidden_dim) for hidden_dim in layers]

decoder_lstm = RNN(decoder_cells, return_sequences=True,
return_state=True)

decoder_outputs_and_states = decoder_lstm(decoder_inputs,
                                         initial_state=encoder_states)
decoder_outputs = decoder_outputs_and_states[0]

decoder_dense = Dense(n_out_features)
decoder_outputs = decoder_dense(decoder_outputs)

```

```
model = Model([encoder_inputs,decoder_inputs], decoder_outputs)
return model

def run_model(model,batches,epochs,batch_size):

    for _ in range(batches):

        input_seq, output_seq = generate_train_sequences(x_train)

        encoder_input_data = input_seq
        decoder_target_data = output_seq
        decoder_input_data = np.zeros(decoder_target_data.shape)

        history = model.fit([encoder_input_data, decoder_input_data],
                            decoder_target_data,
                            batch_size=batch_size,
                            epochs=epochs,
                            validation_split=0.1,
                            shuffle=False)

        total_loss.append(history.history['loss'])
        total_val_loss.append(history.history['val_loss'])

model_1 = create_model(layers=[60],bidirectional=False)
```

```

total_loss = []
total_val_loss = []
model_1.compile(Adam() , loss = 'mean_squared_error')

total_loss = [j for i in total_loss for j in i]
total_val_loss = [j for i in total_val_loss for j in i]

input_seq_test = x_train[-60:].reshape((1,60,1))
output_seq_test = y[:20]
decoder_input_test = np.zeros((1,output_seq_len,1))

pred1 = model_1.predict([input_seq_test,decoder_input_test])

pred_values1 = scaler.inverse_transform(pred1.reshape(-1,1))
output_seq_test1 = scaler.inverse_transform(output_seq_test)
print(pred_values1)
print(output_seq_test1)

plt.plot(pred_values1, label = "pred")
plt.plot(output_seq_test1, label = "actual")
plt.title("Prediction vs Actual")
plt.ylabel("windspeed", fontsize=12)
plt.xlabel("hours", fontsize=12)
plt.legend()

expectations = np.array(output_seq_test1)

```

```
predictions = np.array(pred_values1)
print("Mean Absolute Percent Error: ", (np.mean(np.abs((expectations -
    predictions) / expectations))*100))

from sklearn.metrics import explained_variance_score
print("expected variance score",explained_variance_score(expectations,
    predictions))

from sklearn.metrics import max_error
print("max error:",max_error(expectations , predictions))

from sklearn.metrics import mean_absolute_error
print("mean absolute error:",mean_absolute_error(expectations ,
    predictions))

from sklearn.metrics import mean_squared_error
print("mean squared error:",mean_squared_error(expectations ,
    predictions))

from sklearn.metrics import mean_squared_error
print("root mean squared error:",mean_squared_error(expectations ,
    predictions , squared=False))

from sklearn.metrics import mean_gamma_deviance
print("mean gamma deviance:",mean_gamma_deviance(expectations ,
```

```

predictions))

from sklearn.metrics import mean_poisson_deviance
print("mean poisson deviance:",mean_poisson_deviance(expectations ,
predictions))

from sklearn.metrics import r2_score
print("r2 score:",r2_score(expectations , predictions))
print("r2 score variance weighted:",r2_score(expectations , predictions ,
multioutput='variance_weighted')))

from sklearn.metrics import median_absolute_error
print("median absolute error:",median_absolute_error(expectations ,
predictions))

from sklearn.metrics import mean_squared_log_error
print("mean squared log error:",mean_squared_log_error(predictions ,
expectations))

model_2 = create_model([60,60],bidirectional=False)

total_loss = []
total_val_loss = []
model_2.compile(Adam() , loss = 'mean_squared_error')

input_seq_test = x_train[-60:].reshape((1,60,1))
output_seq_test = y[:20]
decoder_input_test = np.zeros((1,output_seq_len,1))

```

```

pred2 = model_2.predict([input_seq_test,decoder_input_test])

pred_values2 = scaler.inverse_transform(pred2.reshape(-1,1))
output_seq_test2 = scaler.inverse_transform(output_seq_test)
print(pred_values2)
print(output_seq_test2)

plt.plot(pred_values2, label = "pred")
plt.plot(output_seq_test2, label = "actual")
plt.title("Prediction vs Actual")
plt.ylabel("windspeed", fontsize=12)
plt.xlabel("hours", fontsize=12)
plt.legend()

expectations = np.array(output_seq_test2)
predictions = np.array(pred_values2)
print("Mean Absolute Percent Error: ", (np.mean(np.abs((expectations -
predictions) / expectations)) * 100))

from sklearn.metrics import explained_variance_score
print("expected variance score",explained_variance_score(expectations,
predictions))

from sklearn.metrics import max_error
print("max error:",max_error(expectations, predictions))

```

```
from sklearn.metrics import mean_absolute_error
print("mean absolute error:", mean_absolute_error(expectations,
predictions))

from sklearn.metrics import mean_squared_error
print("mean squared error:", mean_squared_error(expectations,
predictions))

from sklearn.metrics import mean_squared_error
print("root mean squared error:", mean_squared_error(expectations,
predictions, squared=False))

from sklearn.metrics import mean_gamma_deviance
print("mean gamma deviance:", mean_gamma_deviance(expectations,
predictions))

from sklearn.metrics import mean_poisson_deviance
print("mean poisson deviance:", mean_poisson_deviance(expectations,
predictions))

from sklearn.metrics import r2_score
print("r2 score:", r2_score(expectations, predictions))
print("r2 score variance weighted:", r2_score(expectations, predictions,
multioutput='variance_weighted'))

from sklearn.metrics import median_absolute_error
print("median absolute error:", median_absolute_error(expectations,
```

```
predictions))

from sklearn.metrics import mean_squared_log_error
print("mean squared log error:", mean_squared_log_error(predictions,
expectations))
```

A.15 CODE FOR LSTM BIDIRECTIONAL 1 LAYER MODEL IN PYTHON

```
from google.colab import drive
drive.mount('/content/drive')

import pandas as pd
import numpy as np
import time

%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("darkgrid")

from keras.models import Model
from keras.layers import Input, LSTM, Dense, LSTMCell, RNN,
Bidirectional, concatenate
from keras.optimizers import Adam

from sklearn.preprocessing import MinMaxScaler
```

```

data = pd.read_csv(r"/content/drive/My Drive/lstm.csv", usecols =[3])
data.head()
scaler = MinMaxScaler(feature_range=(-1,1))
data =scaler.fit_transform(data.values.reshape(-1,1))

x_train = data[:-100]
y = data[-100:]

input_seq_len = 60
output_seq_len = 20
n_in_features = 1
n_out_features = 1
batch_size = 10

def generate_train_sequences(data):

    total_start_points = len(data) - input_seq_len - output_seq_len
    start_x_idx = np.random.choice(range(total_start_points),
                                    total_start_points, replace = False)

    input_batch_idxs = [(range(i, i+input_seq_len)) for i in
                        start_x_idx]
    input_seq = np.take(data, input_batch_idxs, axis = 0)

    output_batch_idxs = [(range(i+input_seq_len, i+input_seq_len+
                                output_seq_len)) for i in start_x_idx]

```

```

output_seq = np.take(data, output_batch_idxs, axis = 0)

input_seq =(input_seq.reshape(input_seq.shape[0],input_seq.shape
[1],n_in_features))
output_seq=(output_seq.reshape(output_seq.shape[0],output_seq.shape
[1],n_out_features))

return input_seq, output_seq

def create_model(layers, bidirectional=False):

    n_layers = len(layers)

    ## Encoder
    encoder_inputs = Input(shape=(None, n_in_features))
    lstm_cells = [LSTMCell(hidden_dim) for hidden_dim in layers]
    if bidirectional:
        encoder = Bidirectional(RNN(lstm_cells, return_state=True))
        encoder_outputs_and_states = encoder(encoder_inputs)
        bi_encoder_states = encoder_outputs_and_states[1:]
        encoder_states = []
        for i in range(int(len(bi_encoder_states)/2)):
            temp = concatenate([bi_encoder_states[i],bi_encoder_states
[2*n_layers + i]], axis=-1)
            encoder_states.append(temp)
    else:
        encoder = RNN(lstm_cells, return_state=True)
        encoder_outputs_and_states = encoder(encoder_inputs)
        encoder_states = encoder_outputs_and_states[1:]

```

```

## Decoder
decoder_inputs = Input(shape=(None, n_out_features))
if bidirectional:
    decoder_cells = [LSTMCell(hidden_dim*2) for hidden_dim in
                    layers]
else:
    decoder_cells = [LSTMCell(hidden_dim) for hidden_dim in layers]

decoder_lstm = RNN(decoder_cells, return_sequences=True,
                    return_state=True)

decoder_outputs_and_states = decoder_lstm(decoder_inputs,
                                            initial_state=encoder_states)
decoder_outputs = decoder_outputs_and_states[0]

decoder_dense = Dense(n_out_features)
decoder_outputs = decoder_dense(decoder_outputs)

model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
return model

def run_model(model, batches, epochs, batch_size):
    for _ in range(batches):
        input_seq, output_seq = generate_train_sequences(x_train)

        encoder_input_data = input_seq

```

```

decoder_target_data = output_seq
decoder_input_data = np.zeros(decoder_target_data.shape)

history = model.fit([encoder_input_data, decoder_input_data],
                     decoder_target_data,
                     batch_size=batch_size,
                     epochs=epochs,
                     validation_split=0.1,
                     shuffle=False)

total_loss.append(history.history['loss'])
total_val_loss.append(history.history['val_loss'])

model1_bi = create_model([60], bidirectional=True)

total_loss = []
total_val_loss = []
model1_bi.compile(Adam(), loss = 'mean_squared_error')

#bi 1 layered model
start_time = time.time()
run_model(model1_bi, batches=1, epochs=70, batch_size=batch_size)
end_time = time.time()

input_seq_test = x_train[-60:].reshape((1,60,1))
output_seq_test = y[:20]
decoder_input_test = np.zeros((1,output_seq_len,1))

```

```

pred = model1_bi.predict([input_seq_test,decoder_input_test])

pred_values = scaler.inverse_transform(pred.reshape(-1,1))
output_seq_test = scaler.inverse_transform(output_seq_test)
input_seq_test = scaler.inverse_transform(input_seq_test.reshape(-1,1))

print(pred_values)
print(output_seq_test)

plt.plot(pred_values, label = "pred")
plt.plot(output_seq_test, label = "actual")
plt.title("Prediction vs Actual")
plt.ylabel("wind speed", fontsize=12)
plt.xlabel("hours", fontsize=12)
plt.legend()

expectations = np.array(output_seq_test)
predictions = np.array(pred_values)
print("Mean Absolute Percent Error: ", (np.mean(np.abs((expectations -
predictions) / expectations)) * 100))

from sklearn.metrics import explained_variance_score
print("Explained variance score : ", explained_variance_score(
expectations, predictions))

from sklearn.metrics import max_error
print("Max Error : ", max_error(expectations, predictions))

```

```
from sklearn.metrics import mean_absolute_error
print("Mean Absolute Error:",mean_absolute_error(expectations,
predictions))

from sklearn.metrics import mean_squared_error
print("Mean Squared Error:",mean_squared_error(expectations,
predictions))

from sklearn.metrics import mean_squared_error
print("Root Mean Squared Error:",mean_squared_error(expectations,
predictions, squared=False))

from sklearn.metrics import mean_squared_log_error
print("Mean Squared Log Error:",mean_squared_log_error(expectations,
predictions))

from sklearn.metrics import median_absolute_error
print("Median Absolute Error:",median_absolute_error(expectations,
predictions))

from sklearn.metrics import r2_score
print("R2 score:",r2_score(expectations, predictions))
print("R2 score variance weighted:",r2_score(expectations, predictions,
multioutput='variance_weighted'))

from sklearn.metrics import mean_poisson_deviance
print("Mean Poisson Deviance:",mean_poisson_deviance(expectations,
predictions))
```

A.16 CODE FOR LSTM BIDIRECTIONAL 2 LAYER MODEL IN PYTHON

```
from google.colab import drive
drive.mount('/content/drive')

import pandas as pd
import numpy as np
import time

%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("darkgrid")

from keras.models import Model
from keras.layers import Input, LSTM, Dense, LSTMCell, RNN,
    Bidirectional, concatenate
from keras.optimizers import Adam

from sklearn.preprocessing import MinMaxScaler

data = pd.read_csv(r"/content/drive/My Drive/lstm.csv", usecols =[3])
data.head()
scaler = MinMaxScaler(feature_range=(-1,1))
data =scaler.fit_transform(data.values.reshape(-1,1))
```

```

x_train = data[:-100]
y = data[-100:]

input_seq_len = 60
output_seq_len = 20
n_in_features = 1
n_out_features = 1
batch_size = 10

def generate_train_sequences(data):

    total_start_points = len(data) - input_seq_len - output_seq_len
    start_x_idx = np.random.choice(range(total_start_points),
                                    total_start_points, replace = False)

    input_batch_idxs = [(range(i, i+input_seq_len)) for i in
                        start_x_idx]
    input_seq = np.take(data, input_batch_idxs, axis = 0)

    output_batch_idxs = [(range(i+input_seq_len, i+input_seq_len+
                                output_seq_len)) for i in start_x_idx]
    output_seq = np.take(data, output_batch_idxs, axis = 0)

    input_seq =(input_seq.reshape(input_seq.shape[0],input_seq.shape
                                [1],n_in_features))
    output_seq=(output_seq.reshape(output_seq.shape[0],output_seq.shape
                                [1],n_out_features))

```

```

    return input_seq, output_seq

def create_model(layers, bidirectional=False):

    n_layers = len(layers)

    ## Encoder

    encoder_inputs = Input(shape=(None, n_in_features))
    lstm_cells = [LSTMCell(hidden_dim) for hidden_dim in layers]
    if bidirectional:
        encoder = Bidirectional(RNN(lstm_cells, return_state=True))
        encoder_outputs_and_states = encoder(encoder_inputs)
        bi_encoder_states = encoder_outputs_and_states[1:]
        encoder_states = []
        for i in range(int(len(bi_encoder_states)/2)):
            temp = concatenate([bi_encoder_states[i], bi_encoder_states
                [2*n_layers + i]], axis=-1)
            encoder_statesMI.append(temp)
    else:
        encoder = RNN(lstm_cells, return_state=True)
        encoder_outputs_and_states = encoder(encoder_inputs)
        encoder_states = encoder_outputs_and_states[1:]

    ## Decoder

    decoder_inputs = Input(shape=(None, n_out_features))
    if bidirectional:
        decoder_cells = [LSTMCell(hidden_dim*2) for hidden_dim in
            layers]
    else:

```

```

decoder_cells = [LSTMCell(hidden_dim) for hidden_dim in layers]

decoder_lstm = RNN(decoder_cells, return_sequences=True,
                    return_state=True)

decoder_outputs_and_states = decoder_lstm(decoder_inputs,
                                            initial_state=encoder_states)
decoder_outputs = decoder_outputs_and_states[0]

decoder_dense = Dense(n_out_features)
decoder_outputs = decoder_dense(decoder_outputs)

model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
return model

def run_model(model, batches, epochs, batch_size):

    for _ in range(batches):

        input_seq, output_seq = generate_train_sequences(x_train)

        encoder_input_data = input_seq
        decoder_target_data = output_seq
        decoder_input_data = np.zeros(decoder_target_data.shape)

        history = model.fit([encoder_input_data, decoder_input_data],
                            decoder_target_data,
                            batch_size=batch_size,
                            epochs=epochs,

```

```

        validation_split=0.1,
        shuffle=False)

total_loss.append(history.history['loss'])
total_val_loss.append(history.history['val_loss'])

model2_bi = create_model([60,60], bidirectional=True)

total_loss = []
total_val_loss = []
model2_bi.compile(Adam() , loss = 'mean_squared_error')

start_time = time.time()
run_model(model2_bi, batches=1, epochs=100, batch_size=batch_size)
end_time = time.time()

input_seq_test = x_train[-60:].reshape((1,60,1))
output_seq_test = y[:20]
decoder_input_test = np.zeros((1,output_seq_len,1))

pred2_bi = model2_bi.predict([input_seq_test,decoder_input_test])

pred_values2_bi = scaler.inverse_transform(pred2_bi.reshape(-1,1))
output_seq_test2_bi = scaler.inverse_transform(output_seq_test)

```

```

input_seq_test = scaler.inverse_transform(input_seq_test.reshape(-1,1))
print(pred_values2_bi)
print(output_seq_test2_bi)

plt.plot(pred_values2_bi, label = "pred")
plt.plot(output_seq_test2_bi, label = "actual")
plt.title("Prediction vs Actual")
plt.ylabel("windspeed", fontsize=12)
plt.xlabel("hours", fontsize=12)
plt.legend()

expectations = np.array(output_seq_test2_bi)
predictions = np.array(pred_values2_bi)
print("Mean Absolute Percent Error: ", (np.mean(np.abs((expectations -
predictions) / expectations)) * 100))

from sklearn.metrics import explained_variance_score
print("Explained variance score : ", explained_variance_score(
expectations, predictions))

from sklearn.metrics import max_error
print("Max Error : ", max_error(expectations, predictions))

from sklearn.metrics import mean_absolute_error
print("Mean Absolute Error: ", mean_absolute_error(expectations,
predictions))

from sklearn.metrics import mean_squared_error

```

```

print("Mean Squared Error:",mean_squared_error(expectations ,
predictions))

from sklearn.metrics import mean_squared_error
print("Root Mean Squared Error:",mean_squared_error(expectations ,
predictions , squared=False))

from sklearn.metrics import mean_squared_log_error
print("Mean Squared Log Error:",mean_squared_log_error(expectations ,
predictions))

from sklearn.metrics import median_absolute_error
print("Median Absolute Error:",median_absolute_error(expectations ,
predictions))

from sklearn.metrics import r2_score
print("R2 score:",r2_score(expectations , predictions))
print("R2 score variance weighted:",r2_score(expectations , predictions ,
multioutput='variance_weighted'))

from sklearn.metrics import mean_poisson_deviance
print("Mean Poisson Deviance:",mean_poisson_deviance(expectations ,
predictions))

```

A.17 CODE FOR FINAL STACKED MODEL IN PYTHON

```
!pip install lightgbm xgboost scikit-learn pandas mlxtend --upgrade
```

```
import numpy as np
```

```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import xgboost
import math
from scipy.stats import pearsonr
from sklearn.model_selection import cross_validate
from sklearn.model_selection import train_test_split
from sklearn.metrics import explained_variance_score
from sklearn.linear_model import LinearRegression
regr = LinearRegression()
import warnings
warnings.filterwarnings("ignore")
import seaborn as sns

from google.colab import drive
drive.mount('/content/drive')

data_1 = pd.read_csv("/content/drive/My Drive/Windspeed csv file /
wind_speed_1_day.csv")
data_1.head()

sorted(data_1.columns)

print(data_1.shape)
print(len(data_1))
print(len(data_1.columns))
print(data_1.dtypes.unique())
print(data_1.dtypes)
```

```
print(data_1.isnull().any().sum() , ' / ', len(data_1.columns))
print(data_1.isnull().any(axis=1).sum() , ' / ', len(data_1))

windspeed_1_day= data_1[['wind_speed']]
print(windspeed_1_day)

data_2 = pd.read_csv("/content/drive/My Drive/Windspeed csv file /
                     wind_speed_12_hour.csv")
data_2.head()

sorted(data_2.columns)

print(data_2.shape)
print(len(data_2))
print(len(data_2.columns))
print(data_2.dtypes.unique())
print(data_2.dtypes)

print(data_2.isnull().any().sum() , ' / ', len(data_2.columns))
print(data_2.isnull().any(axis=1).sum() , ' / ', len(data_2))

windspeed_12_hour= data_2[['wind_speed']]
print(windspeed_12_hour)

data_3 = pd.read_csv("/content/drive/My Drive/Windspeed csv file /
                     wind_speed_2_days.csv")
data_3.head()
```

```
sorted(data_3.columns)

print(data_3.shape)
print(len(data_3))
print(len(data_3.columns))
print(data_3.dtypes.unique())
print(data_3.dtypes)

print(data_3.isnull().any().sum(), ' / ', len(data_3.columns))
print(data_3.isnull().any(axis=1).sum(), ' / ', len(data_3))

windspeed_2_days= data_3[['wind_speed']]
print(windspeed_2_days)

data_4 = pd.read_csv("/content/drive/My Drive/Windspeed csv file /
wind_speed_1_week.csv")
data_4.head()

sorted(data_4.columns)

print(data_4.shape)
print(len(data_4))
print(len(data_4.columns))
print(data_4.dtypes.unique())
print(data_4.dtypes)

print(data_4.isnull().any().sum(), ' / ', len(data_4.columns))
print(data_4.isnull().any(axis=1).sum(), ' / ', len(data_4))
```

```

windspeed_1_week= data_4[['wind_speed']]
print(windspeed_1_week)

data_5 = pd.read_csv("/content/drive/My Drive/Windspeed csv file /
                     wind_speed_1_month.csv")
data_5.head()

sorted(data_5.columns)

print(data_5.shape)
print(len(data_5))
print(len(data_5.columns))
print(data_5.dtypes.unique())
print(data_5.dtypes)

print(data_5.isnull().any().sum(), ' / ', len(data_5.columns))
print(data_5.isnull().any(axis=1).sum(), ' / ', len(data_5))

windspeed_1_month= data_5[['wind_speed']]
print(windspeed_1_month)

data_6 = pd.read_csv("/content/drive/My Drive/Windspeed csv file /
                     Wind_speed_bidirectional_1_layer.csv")
data_6.head()

sorted(data_6.columns)

print(data_6.shape)
print(len(data_6))

```

```

print(len(data_6.columns))
print(data_6.dtypes.unique())
print(data_6.dtypes)

print(data_6.isnull().any().sum(), ' / ', len(data_6.columns))
print(data_6.isnull().any(axis=1).sum(), ' / ', len(data_6))

windspeed_bidirectional_1_layer= data_6[['wind_speed']]
print(windspeed_bidirectional_1_layer)

data_7 = pd.read_csv("/content/drive/My Drive/Windspeed csv file /
    Wind_speed_bidirectional_2_layer.csv")
data_7.head()

sorted(data_7.columns)

print(data_7.shape)
print(len(data_7))
print(len(data_7.columns))
print(data_7.dtypes.unique())
print(data_7.dtypes)

print(data_7.isnull().any().sum(), ' / ', len(data_7.columns))
print(data_7.isnull().any(axis=1).sum(), ' / ', len(data_7))

windspeed_bidirectional_2_layer= data_7[['wind_speed']]
print(windspeed_bidirectional_2_layer)

data_8 = pd.read_csv("/content/drive/My Drive/Windspeed csv file /

```

```

Wind_speed_unidirectional_1_layer.csv")
data_8.head()

sorted(data_8.columns)

print(data_8.shape)
print(len(data_8))
print(len(data_8.columns))
print(data_8.dtypes.unique())
print(data_8.dtypes)

print(data_8.isnull().any().sum(), ' / ', len(data_8.columns))
print(data_8.isnull().any(axis=1).sum(), ' / ', len(data_8))

windspeed_unidirectional_1_layer= data_8[['wind_speed']]
print(windspeed_unidirectional_1_layer)

data_9 = pd.read_csv("/content/drive/My Drive/Windspeed csv file /
Wind_speed_unidirectional_2_layer.csv")
data_9.head()

sorted(data_9.columns)

print(data_9.shape)
print(len(data_9))
print(len(data_9.columns))
print(data_9.dtypes.unique())
print(data_9.dtypes)

```

```

print(data_9.isnull().any().sum() , ' / ', len(data_9.columns))
print(data_9.isnull().any(axis=1).sum() , ' / ', len(data_9))

windspeed_unidirectional_2_layer= data_9[['wind_speed']]
print(windspeed_unidirectional_2_layer)

data_10= pd.read_csv("/content/drive/My Drive/Lstm power csv files/
Lstm_power_1_day.csv")
data_10.head()

sorted(data_10.columns)

print(data_10.shape)
print(len(data_10))
print(len(data_10.columns))
print(data_10.dtypes.unique())
print(data_10.dtypes)

print(data_10.isnull().any().sum() , ' / ', len(data_10.columns))
print(data_10.isnull().any(axis=1).sum() , ' / ', len(data_10))

windpower_1_day= data_10[['power']]
print(windpower_1_day)

data_11= pd.read_csv("/content/drive/My Drive/Lstm power csv files/
Lstm_power_2_day.csv")
data_11.head()

sorted(data_11.columns)

```

```

print(data_11.shape)
print(len(data_11))
print(len(data_11.columns))
print(data_11.dtypes.unique())
print(data_11.dtypes)

print(data_11.isnull().any().sum(), ' / ', len(data_11.columns))
print(data_11.isnull().any(axis=1).sum(), ' / ', len(data_11))

windpower_2_days= data_11[['power']]
print(windpower_2_days)

data_12= pd.read_csv("/content/drive/My Drive/Lstm power csv files/
Lstm_power_1_week.csv")
data_12.head()

sorted(data_12.columns)

print(data_12.shape)
print(len(data_12))
print(len(data_12.columns))
print(data_12.dtypes.unique())
print(data_12.dtypes)

print(data_12.isnull().any().sum(), ' / ', len(data_12.columns))
print(data_12.isnull().any(axis=1).sum(), ' / ', len(data_12))

windpower_1_week= data_12[['power']]

```

```

print(windpower_1_week)

data_13= pd.read_csv("/content/drive/My Drive/Lstm power csv files/
Lstm_power_1_month.csv")
data_13.head()

sorted(data_13.columns)

print(data_13.shape)
print(len(data_13))
print(len(data_13.columns))
print(data_13.dtypes.unique())
print(data_13.dtypes)

print(data_13.isnull().any().sum() , ' / ', len(data_13.columns))
print(data_13.isnull().any(axis=1).sum() , ' / ', len(data_13))

windpower_1_month= data_13[['power']]
print(windpower_1_month)

data_14= pd.read_csv("/content/drive/My Drive/Lstm power csv files/
Lstm_power_Bidirectional_1_layer.csv")
data_14.head()

sorted(data_14.columns)

print(data_14.shape)
print(len(data_14))
print(len(data_14.columns))

```

```

print(data_14.dtypes.unique())
print(data_14.dtypes)

print(data_14.isnull().any().sum(), ' / ', len(data_14.columns))
print(data_14.isnull().any(axis=1).sum(), ' / ', len(data_14))

windpower_bidirectional= data_14[['power']]
print(windpower_bidirectional)

data_15= pd.read_csv("/content/drive/My Drive/Lstm power csv files/
LSTM12HRS.csv")
data_15.head()

sorted(data_15.columns)

print(data_15.shape)
print(len(data_15))
print(len(data_15.columns))
print(data_15.dtypes.unique())
print(data_15.dtypes)

print(data_15.isnull().any().sum(), ' / ', len(data_15.columns))
print(data_15.isnull().any(axis=1).sum(), ' / ', len(data_15))

windpower_12_hour= data_15[['power']]
print(windpower_12_hour)

data_16 = pd.read_csv("/content/drive/My Drive/one_year – Sheet1.csv")
data_16.head()

```

```
sorted(data_16.columns)

print(data_16.shape)
print(len(data_16))
print(len(data_16.columns))
print(data_16.dtypes.unique())
print(data_16.dtypes)

data_16.select_dtypes(include=['O']).columns.tolist()

print(data_16.isnull().any().sum(), ' / ', len(data_16.columns))
print(data_16.isnull().any(axis=1).sum(), ' / ', len(data_16))

a = data_16[['wind_speed']]
print(a)
b = data_16['power']
print(b)

X = a.values
y = b.values

X_train, X_test, y_train, y_test=train_test_split(X,y, test_size=0.1)

from sklearn.model_selection import GridSearchCV
from sklearn.metrics import r2_score

def algorithm_pipeline(X_train, X_test, y_train, y_test,
                      model, param_grid, cv=10, scoring_fit='
```

```

        neg_mean_squared_error' ,
scoring_test=r2_score , do_probabilities = False)
:

gs = GridSearchCV(
    estimator=model,
    param_grid=param_grid,
    cv=cv,
    n_jobs=-1,
    scoring=scoring_fit ,
    verbose=2
)
fitted_model = gs.fit(X_train , y_train)
best_model = fitted_model.best_estimator_

if do_probabilities:
    pred = fitted_model.predict_proba(X_test)
else:
    pred = fitted_model.predict(X_test)

score = scoring_test(y_test , pred)

return [best_model, pred, score]

from sklearn.ensemble import RandomForestRegressor
from lightgbm import LGBMRegressor
from xgboost import XGBRegressor

models_to_train = [XGBRegressor() , LGBMRegressor() ,
RandomForestRegressor()]

```

```

grid_parameters = [
    { # XGBoost
        'n_estimators': [400, 700, 1000],
        'colsample_bytree': [0.7, 0.8],
        'max_depth': [15,20,25],
        'reg_alpha': [1.1, 1.2, 1.3],
        'reg_lambda': [1.1, 1.2, 1.3],
        'subsample': [0.7, 0.8, 0.9]
    },
    { # LightGBM
        'n_estimators': [400, 700, 1000],
        'learning_rate': [0.12],
        'colsample_bytree': [0.7, 0.8],
        'max_depth': [4],
        'num_leaves': [10, 20],
        'reg_alpha': [1.1, 1.2],
        'reg_lambda': [1.1, 1.2],
        'min_split_gain': [0.3, 0.4],
        'subsample': [0.8, 0.9],
        'subsample_freq': [10, 20]
    },
    { # Random Forest
        'max_depth':[3, 5, 10, 13],
        'n_estimators':[100, 200, 400, 600, 900],
        'max_features':[1,2,3,4,5,6]
    }
]

```

```

models_preds_scores = []

for i, model in enumerate(models_to_train):
    params = grid_parameters[i]

    result = algorithm_pipeline(X_train, X_test, y_train, y_test,
                                model, params, cv=5)
    models_preds_scores.append(result)

for result in models_preds_scores:
    print('Model: {} , Score: {}'.format(type(result[0]).__name__,
                                           result[2]))


from mlxtend.regressor import StackingCVRegressor
from sklearn.linear_model import Ridge, Lasso
from sklearn.svm import SVR

xgb = XGBRegressor()
lgbm = LGBMRegressor()
rf = RandomForestRegressor()
ridge = Ridge()
lasso = Lasso()
svr = SVR(kernel='linear')
stack = StackingCVRegressor(regressors=(rf, lgbm, xgb),
                            meta_regressor=lgbm, cv=12,
                            use_features_in_secondary=True,
                            store_train_meta_features=True,

```

```

        shuffle=False ,
        random_state=42)

stack.fit(X_train, y_train)

pred_1 = stack.predict(windspeed_12_hour.values)
score_1 = r2_score(windpower_12_hour.values, pred_1)
print(score_1)

from sklearn.metrics import explained_variance_score
print("Explained variance score :" ,explained_variance_score(
    windpower_12_hour.values, pred_1))

from sklearn.metrics import max_error
print("Max Error :" ,max_error(windpower_12_hour.values, pred_1))

from sklearn.metrics import mean_absolute_error
print("Mean Absolute Error:" ,mean_absolute_error(windpower_12_hour.
    values, pred_1))

from sklearn.metrics import mean_squared_error
print("Mean Squared Error:" ,mean_squared_error(windpower_12_hour.values
    , pred_1))

from sklearn.metrics import mean_squared_error
print("Root Mean Squared Error:" ,mean_squared_error(windpower_12_hour.

```

```

values , pred_1 , squared=False))

from sklearn.metrics import mean_squared_log_error
print("Mean Squared Log Error:",mean_squared_log_error(
    windpower_12_hour.values,pred_1))

from sklearn.metrics import median_absolute_error
print("Median Absolute Error:",median_absolute_error(windpower_12_hour.
    values , pred_1))

from sklearn.metrics import r2_score
print("R2 score:",r2_score(windpower_12_hour.values , pred_1))
print("R2 score variance weighted:",r2_score(windpower_12_hour.values ,
    pred_1 , multioutput='variance_weighted' ) )

from sklearn.metrics import mean_poisson_deviance
print("Mean Poisson Deviance:",mean_poisson_deviance(windpower_12_hour.
    values , pred_1))

plt.plot(windpower_12_hour.values , label="True")
plt.plot(pred_1 , label="Predicted")
plt.legend(loc='upper right')
plt.xlabel("Number of hours")
plt.ylabel("Power generated by system (kW) ")
plt.show()

from mlxtend.regressor import StackingCVRegressor
from sklearn.linear_model import Ridge , Lasso
from sklearn.svm import SVR

```

```
xgb = XGBRegressor()  
lgbm = LGBMRegressor()  
rf = RandomForestRegressor()  
ridge = Ridge()  
lasso = Lasso()  
svr = SVR(kernel='linear')  
stack = StackingCVRegressor(regressors=(rf,lgbm, xgb),  
                           meta_regressor=lgbm, cv=12,  
                           use_features_in_secondary=True,  
                           store_train_meta_features=True,  
                           shuffle=False,  
                           random_state=42)
```

```
stack.fit(X_train, y_train)
```

```
pred_2 = stack.predict(windspeed_1_day.values)  
score_2 = r2_score(windpower_1_day.values, pred_2)  
print(score_2)
```

```
from sklearn.metrics import explained_variance_score  
print("Explained variance score :" ,explained_variance_score(  
      windpower_1_day.values, pred_2))
```

```
from sklearn.metrics import max_error  
print("Max Error :" ,max_error(windpower_1_day.values, pred_2))
```

```
from sklearn.metrics import mean_absolute_error
print("Mean Absolute Error:", mean_absolute_error(windpower_1_day.values
, pred_2))

from sklearn.metrics import mean_squared_error
print("Mean Squared Error:", mean_squared_error(windpower_1_day.values,
pred_2))

from sklearn.metrics import mean_squared_error
print("Root Mean Squared Error:", mean_squared_error(windpower_1_day.
values, pred_2, squared=False))

from sklearn.metrics import mean_squared_log_error
print("Mean Squared Log Error:", mean_squared_log_error(windpower_1_day.
values, pred_2))

from sklearn.metrics import median_absolute_error
print("Median Absolute Error:", median_absolute_error(windpower_1_day.
values, pred_2))

from sklearn.metrics import r2_score
print("R2 score:", r2_score(windpower_1_day.values, pred_2))
print("R2 score variance weighted:", r2_score(windpower_1_day.values,
pred_2, multioutput='variance_weighted'))

from sklearn.metrics import mean_poisson_deviance
print("Mean Poisson Deviance:", mean_poisson_deviance(windpower_1_day.
values, pred_2))
```

```

plt.plot(windpower_1_day.values, label="True")
plt.plot(pred_2, label="Predicted")
plt.legend(loc='upper right')
plt.xlabel("Number of hours")
plt.ylabel("Power generated by system (kW)")
plt.show()

from mlxtend.regressor import StackingCVRegressor
from sklearn.linear_model import Ridge, Lasso
from sklearn.svm import SVR

xgb = XGBRegressor()
lgbm = LGBMRegressor()
rf = RandomForestRegressor()
ridge = Ridge()
lasso = Lasso()
svr = SVR(kernel='linear')
stack = StackingCVRegressor(regressors=(rf, lgbm, xgb),
                           meta_regressor=lgbm, cv=12,
                           use_features_in_secondary=True,
                           store_train_meta_features=True,
                           shuffle=False,
                           random_state=42)

stack.fit(X_train, y_train)

```

```
pred_3 = stack.predict(windspeed_2_days.values)
score_3 = r2_score(windpower_2_days.values, pred_3)
print(score_3)

from sklearn.metrics import explained_variance_score
print("Explained variance score : ", explained_variance_score(
    windpower_2_days.values, pred_3))

from sklearn.metrics import max_error
print("Max Error : ", max_error(windpower_2_days.values, pred_3))

from sklearn.metrics import mean_absolute_error
print("Mean Absolute Error: ", mean_absolute_error(windpower_2_days.
    values, pred_3))

from sklearn.metrics import mean_squared_error
print("Mean Squared Error: ", mean_squared_error(windpower_2_days.values,
    pred_3))

from sklearn.metrics import mean_squared_error
print("Root Mean Squared Error: ", mean_squared_error(windpower_2_days.
    values, pred_3, squared=False))

from sklearn.metrics import mean_squared_log_error
print("Mean Squared Log Error: ", mean_squared_log_error(windpower_2_days
    .values, pred_3))

from sklearn.metrics import median_absolute_error
```

```

print("Median Absolute Error:", median_absolute_error(windpower_2_days.
values, pred_3))

from sklearn.metrics import r2_score
print("R2 score:", r2_score(windpower_2_days.values, pred_3))
print("R2 score variance weighted:", r2_score(windpower_2_days.values,
pred_3, multioutput='variance_weighted')))

from sklearn.metrics import mean_poisson_deviance
print("Mean Poisson Deviance:", mean_poisson_deviance(windpower_2_days.
values, pred_3))

plt.plot(windpower_2_days.values, label="True")
plt.plot(pred_3, label="Predicted")
plt.legend(loc='upper right')
plt.xlabel("Number of hours")
plt.ylabel("Power generated by system (kW)")
plt.show()

from mlxtend.regressor import StackingCVRegressor
from sklearn.linear_model import Ridge, Lasso
from sklearn.svm import SVR

xgb = XGBRegressor()
lgbm = LGBMRegressor()
rf = RandomForestRegressor()
ridge = Ridge()

```

```

lasso = Lasso()
svr = SVR(kernel='linear')
stack = StackingCVRegressor(regressors=(rf, lgbm, xgb),
                            meta_regressor=lgbm, cv=12,
                            use_features_in_secondary=True,
                            store_train_meta_features=True,
                            shuffle=False,
                            random_state=42)

stack.fit(X_train, y_train)

pred_4 = stack.predict(windspeed_1_week.values)
score_4 = r2_score(windpower_1_week.values, pred_4)
print(score_4)

from sklearn.metrics import explained_variance_score
print("Explained variance score : ", explained_variance_score(
    windpower_1_week.values, pred_4))

from sklearn.metrics import max_error
print("Max Error : ", max_error(windpower_1_week.values, pred_4))

from sklearn.metrics import mean_absolute_error
print("Mean Absolute Error: ", mean_absolute_error(windpower_1_week.
values, pred_4))

```

```

from sklearn.metrics import mean_squared_error
print("Mean Squared Error:",mean_squared_error(windpower_1_week.values ,
pred_4))

from sklearn.metrics import mean_squared_error
print("Root Mean Squared Error:",mean_squared_error(windpower_1_week.
values , pred_4, squared=False))

from sklearn.metrics import mean_squared_log_error
print("Mean Squared Log Error:",mean_squared_log_error(windpower_1_week
.values , pred_4))

from sklearn.metrics import median_absolute_error
print("Median Absolute Error:",median_absolute_error(windpower_1_week.
values , pred_4))

from sklearn.metrics import r2_score
print("R2 score:",r2_score(windpower_1_week.values , pred_4))
print("R2 score variance weighted:",r2_score(windpower_1_week.values ,
pred_4, multioutput='variance_weighted' ))

from sklearn.metrics import mean_poisson_deviance
print("Mean Poisson Deviance:",mean_poisson_deviance(windpower_1_week.
values , pred_4))

plt.plot(windpower_1_week.values , label="True")
plt.plot(pred_4, label="Predicted")
plt.legend(loc='upper right ')
plt.xlabel("Number of hours")

```

```

plt.ylabel("Power generated by system (kW ")
plt.show()

from mlxtend.regressor import StackingCVRegressor
from sklearn.linear_model import Ridge, Lasso
from sklearn.svm import SVR

xgb = XGBRegressor()
lgbm = LGBMRegressor()
rf = RandomForestRegressor()
ridge = Ridge()
lasso = Lasso()
svr = SVR(kernel='linear')
stack = StackingCVRegressor(regressors=(rf,lgbm, xgb),
                           meta_regressor=lgbm, cv=12,
                           use_features_in_secondary=True,
                           store_train_meta_features=True,
                           shuffle=False,
                           random_state=42)

stack.fit(X_train, y_train)

pred_5 = stack.predict(windspeed_1_month.values)
score_5 = r2_score(windpower_1_month.values, pred_5)
print(score_5)

from sklearn.metrics import explained_variance_score

```

```

print("Explained variance score :" ,explained_variance_score(
    windpower_1_month.values , pred_5))

from sklearn.metrics import max_error
print("Max Error :" ,max_error(windpower_1_month.values , pred_5))

from sklearn.metrics import mean_absolute_error
print("Mean Absolute Error:" ,mean_absolute_error(windpower_1_month.
    values , pred_5))

from sklearn.metrics import mean_squared_error
print("Mean Squared Error:" ,mean_squared_error(windpower_1_month.values
    , pred_5))

from sklearn.metrics import mean_squared_error
print("Root Mean Squared Error:" ,mean_squared_error(windpower_1_month.
    values , pred_5 , squared=False))

from sklearn.metrics import mean_squared_log_error
print("Mean Squared Log Error:" ,mean_squared_log_error(
    windpower_1_month.values ,pred_5))

from sklearn.metrics import median_absolute_error
print("Median Absolute Error:" ,median_absolute_error(windpower_1_month.
    values , pred_5))

from sklearn.metrics import r2_score
print("R2 score:" ,r2_score(windpower_1_month.values , pred_5))

```

```

print("R2 score variance weighted:",r2_score(windpower_1_month.values ,
pred_5, multioutput='variance_weighted')))

from sklearn.metrics import mean_poisson_deviance
print("Mean Poisson Deviance:",mean_poisson_deviance(windpower_1_month.
values ,pred_5))

plt.plot(windpower_1_month.values , label="True")
plt.plot(pred_5, label="Predicted")
plt.legend(loc='upper right')
plt.xlabel("Number of hours")
plt.ylabel("Power generated by system (kW )")
plt.show()

from mlxtend.regressor import StackingCVRegressor
from sklearn.linear_model import Ridge, Lasso
from sklearn.svm import SVR

xgb = XGBRegressor()
lgbm = LGBMRegressor()
rf = RandomForestRegressor()
ridge = Ridge()
lasso = Lasso()
svr = SVR(kernel='linear')
stack = StackingCVRegressor(regressors=(rf,lgbm, xgb) ,
                           meta_regressor=lgbm, cv=12,
                           use_features_in_secondary=True,
                           store_train_meta_features=True,
                           shuffle=False ,

```

```

        random_state=42)

stack.fit(X_train, y_train)

pred_6 = stack.predict(windspeed_unidirectional_1_layer.values)
score_6 = r2_score(windpower_bidirectional.values, pred_6)
print(score_6)

from sklearn.metrics import explained_variance_score
print("Explained variance score :" ,explained_variance_score(
    windpower_bidirectional.values, pred_6))

from sklearn.metrics import max_error
print("Max Error :" ,max_error(windpower_bidirectional.values, pred_6))

from sklearn.metrics import mean_absolute_error
print("Mean Absolute Error:" ,mean_absolute_error(
    windpower_bidirectional.values, pred_6))

from sklearn.metrics import mean_squared_error
print("Mean Squared Error:" ,mean_squared_error(windpower_bidirectional.
    values, pred_6))

from sklearn.metrics import mean_squared_error
print("Root Mean Squared Error:" ,mean_squared_error(
    windpower_bidirectional.values, pred_6, squared=False))

```

```

from sklearn.metrics import mean_squared_log_error
print("Mean Squared Log Error:",mean_squared_log_error(
    windpower_bidirectional.values , pred_6))

from sklearn.metrics import median_absolute_error
print("Median Absolute Error:",median_absolute_error(
    windpower_bidirectional.values , pred_6))

from sklearn.metrics import r2_score
print("R2 score:",r2_score(windpower_bidirectional.values , pred_6))
print("R2 score variance weighted:",r2_score(windpower_bidirectional.
    values , pred_6 , multioutput='variance_weighted')))

from sklearn.metrics import mean_poisson_deviance
print("Mean Poisson Deviance:",mean_poisson_deviance(
    windpower_bidirectional.values , pred_6))

plt.plot(windpower_bidirectional.values , label="True")
plt.plot(pred_6, label="Predicted")
plt.legend(loc='upper right')
plt.xlabel("Number of hours")
plt.ylabel("Power generated by system (kW)")
plt.show()

from mlxtend.regressor import StackingCVRegressor
from sklearn.linear_model import Ridge, Lasso

```

```

from sklearn.svm import SVR

xgb = XGBRegressor()
lgbm = LGBMRegressor()
rf = RandomForestRegressor()
ridge = Ridge()
lasso = Lasso()
svr = SVR(kernel='linear')
stack = StackingCVRegressor(regressors=(rf,lgbm, xgb),
                             meta_regressor=lgbm, cv=12,
                             use_features_in_secondary=True,
                             store_train_meta_features=True,
                             shuffle=False,
                             random_state=42)

stack.fit(X_train, y_train)

pred_7 = stack.predict(windspeed_unidirectional_2_layer.values)
score_7 = r2_score(windpower_bidirectional.values, pred_7)
print(score_7)

from sklearn.metrics import explained_variance_score
print("Explained variance score : ",explained_variance_score(
    windpower_bidirectional.values, pred_7))

from sklearn.metrics import max_error

```

```

print("Max Error :" ,max_error(windpower_bidirectional.values , pred_7))

from sklearn.metrics import mean_absolute_error
print("Mean Absolute Error:" ,mean_absolute_error(
    windpower_bidirectional.values , pred_7))

from sklearn.metrics import mean_squared_error
print("Mean Squared Error:" ,mean_squared_error(windpower_bidirectional .
    values , pred_7))

from sklearn.metrics import mean_squared_error
print("Root Mean Squared Error:" ,mean_squared_error(
    windpower_bidirectional.values , pred_7 , squared=False))

from sklearn.metrics import mean_squared_log_error
print("Mean Squared Log Error:" ,mean_squared_log_error(
    windpower_bidirectional.values , pred_7))

from sklearn.metrics import median_absolute_error
print("Median Absolute Error:" ,median_absolute_error(
    windpower_bidirectional.values , pred_7))

from sklearn.metrics import r2_score
print("R2 score:" ,r2_score(windpower_bidirectional.values , pred_7))
print("R2 score variance weighted:" ,r2_score(windpower_bidirectional .
    values , pred_7 , multioutput='variance_weighted '))

from sklearn.metrics import mean_poisson_deviance
print("Mean Poisson Deviance:" ,mean_poisson_deviance(

```

```

windpower_bidirectional.values , pred_7))

plt.plot(windpower_bidirectional.values , label="True")
plt.plot(pred_7, label="Predicted")
plt.legend(loc='upper right')
plt.xlabel("Number of hours")
plt.ylabel("Power generated by system (kW)")
plt.show()

from mlxtend.regressor import StackingCVRegressor
from sklearn.linear_model import Ridge, Lasso
from sklearn.svm import SVR

xgb = XGBRegressor()
lgbm = LGBMRegressor()
rf = RandomForestRegressor()
ridge = Ridge()
lasso = Lasso()
svr = SVR(kernel='linear')

stack = StackingCVRegressor(regressors=(rf,lgbm, xgb),
                           meta_regressor=lgbm, cv=12,
                           use_features_in_secondary=True,
                           store_train_meta_features=True,
                           shuffle=False ,
                           random_state=42)

stack.fit(X_train, y_train)

```

```

pred_8 = stack.predict(windspeed_bidirectional_1_layer.values)
score_8 = r2_score(windpower_bidirectional.values, pred_8)
print(score_8)

from sklearn.metrics import explained_variance_score
print("Explained variance score : ", explained_variance_score(
    windpower_bidirectional.values, pred_8))

from sklearn.metrics import max_error
print("Max Error : ", max_error(windpower_bidirectional.values, pred_8))

from sklearn.metrics import mean_absolute_error
print("Mean Absolute Error: ", mean_absolute_error(
    windpower_bidirectional.values, pred_8))

from sklearn.metrics import mean_squared_error
print("Mean Squared Error: ", mean_squared_error(windpower_bidirectional.
    values, pred_8))

from sklearn.metrics import mean_squared_error
print("Root Mean Squared Error: ", mean_squared_error(
    windpower_bidirectional.values, pred_8, squared=False))

from sklearn.metrics import mean_squared_log_error
print("Mean Squared Log Error: ", mean_squared_log_error(
    windpower_bidirectional.values, pred_8))

```

```

from sklearn.metrics import median_absolute_error
print("Median Absolute Error:", median_absolute_error(
    windpower_bidirectional.values, pred_8))

from sklearn.metrics import r2_score
print("R2 score:", r2_score(windpower_bidirectional.values, pred_8))
print("R2 score variance weighted:", r2_score(windpower_bidirectional.
    values, pred_8, multioutput='variance_weighted')))

from sklearn.metrics import mean_poisson_deviance
print("Mean Poisson Deviance:", mean_poisson_deviance(
    windpower_bidirectional.values, pred_8))

plt.plot(windpower_bidirectional.values, label="True")
plt.plot(pred_8, label="Predicted")
plt.legend(loc='upper right')
plt.xlabel("Number of hours")
plt.ylabel("Power generated by system (kW)")
plt.show()

from mlxtend.regressor import StackingCVRegressor
from sklearn.linear_model import Ridge, Lasso
from sklearn.svm import SVR

xgb = XGBRegressor()
lgbm = LGBMRegressor()
rf = RandomForestRegressor()
ridge = Ridge()
lasso = Lasso()

```

```

svr = SVR(kernel='linear')

stack = StackingCVRegressor(regressors=(rf, lgbm, xgb),
                           meta_regressor=lgbm, cv=12,
                           use_features_in_secondary=True,
                           store_train_meta_features=True,
                           shuffle=False,
                           random_state=42)

stack.fit(X_train, y_train)

pred_9 = stack.predict(windspeed_bidirectional_2_layer.values)
score_9 = r2_score(windpower_bidirectional.values, pred_9)
print(score_9)

from sklearn.metrics import explained_variance_score
print("Explained variance score :" ,explained_variance_score(
    windpower_bidirectional.values, pred_9))

from sklearn.metrics import max_error
print("Max Error :" ,max_error(windpower_bidirectional.values, pred_9))

from sklearn.metrics import mean_absolute_error
print("Mean Absolute Error:" ,mean_absolute_error(
    windpower_bidirectional.values, pred_9))

from sklearn.metrics import mean_squared_error

```

```

print("Mean Squared Error:",mean_squared_error(windpower_bidirectional.
values , pred_9))

from sklearn.metrics import mean_squared_error
print("Root Mean Squared Error:",mean_squared_error(
windpower_bidirectional.values , pred_9 , squared=False))

from sklearn.metrics import mean_squared_log_error
print("Mean Squared Log Error:",mean_squared_log_error(
windpower_bidirectional.values , pred_9))

from sklearn.metrics import median_absolute_error
print("Median Absolute Error:",median_absolute_error(
windpower_bidirectional.values , pred_9))

from sklearn.metrics import r2_score
print("R2 score:",r2_score(windpower_bidirectional.values , pred_9))
print("R2 score variance weighted:",r2_score(windpower_bidirectional.
values , pred_9 , multioutput='variance_weighted'))

from sklearn.metrics import mean_poisson_deviance
print("Mean Poisson Deviance:",mean_poisson_deviance(
windpower_bidirectional.values , pred_9))

plt.plot(windpower_bidirectional.values , label="True")
plt.plot(pred_9 , label="Predicted")
plt.legend(loc='upper right')
plt.xlabel("Number of hours")
plt.ylabel("Power generated by system (kW)")

```

```
plt.show()
```