INSTITUTE FOR ADVANCED COMPUTING AND SOFTWARE DEVELOPMENT AKURDI, PUNE

Documentation On

"Wind Energy Power Prediction Using Machine Learning"

PG-DBDA SEPT 2021

Submitted By:

Group No: 14

Jyoti vishwakarma_219320

Aniket kamble_219321

Mr. Prashant Karhale
Centre Coordinator

Mr. Akshay Tilekar
Project Guide

Contents

List of Figures:

1. Introduction

1.1 PROBLEM STATEMENT

Wind Turbine Power Prediction Using Machine Learning

1.2 Abstract

We are facing the problem of finding new renewable sources of energy, which is why we need to properly utilize the sources which are available. In this problem statement, we had to get accurate energy predictions for the wind farms based on some previous data so that they can operate efficiently.

We are providing an ML-based solution in which we would give 72 hrs future predictions of power output. And the web app would also recommend the time when you could have maximum efficiency for the power output.

1.3 Purpose

Predicting wind energy output will enable us to cut down on production costs and collaborate on different energy sources more efficiently

1.4 Product Scope

Wind power generation is rapidly picking up in many countries. With the ever-increasing demand for electricity which powers our industries, technology and our homes, it is of utmost importance to consider using it in a responsible way. That is where the concept of non-conventional energy sources like wind energy comes in.

The main use of these models is to provide the prediction of the wind turbine generated output by the user in our web interface. The user will input the wind direction and wind speed and press the Predict Question Button after this the model will predict the output power generated by wind turbine for the next 72 hours and send that prediction back to the user.

1.5 Aims & Objectives

The primary goal of this project is to predict the power generated by the wind turbine from the wind speed and direction and use the trained models to predict any input given by the user . Before giving the Power generated, the training of the models will be done using a bunch of different ML models and after the training is done the ML models will be compared based on their RMSE(Root-mean-square-error) and R2 score and the best model will be selected which will then be used to make prediction's for the wind speed and direction input given by the user.

2. Overall Description

2.1 Workflow of Project:

The diagram below shows the workflow of this project.

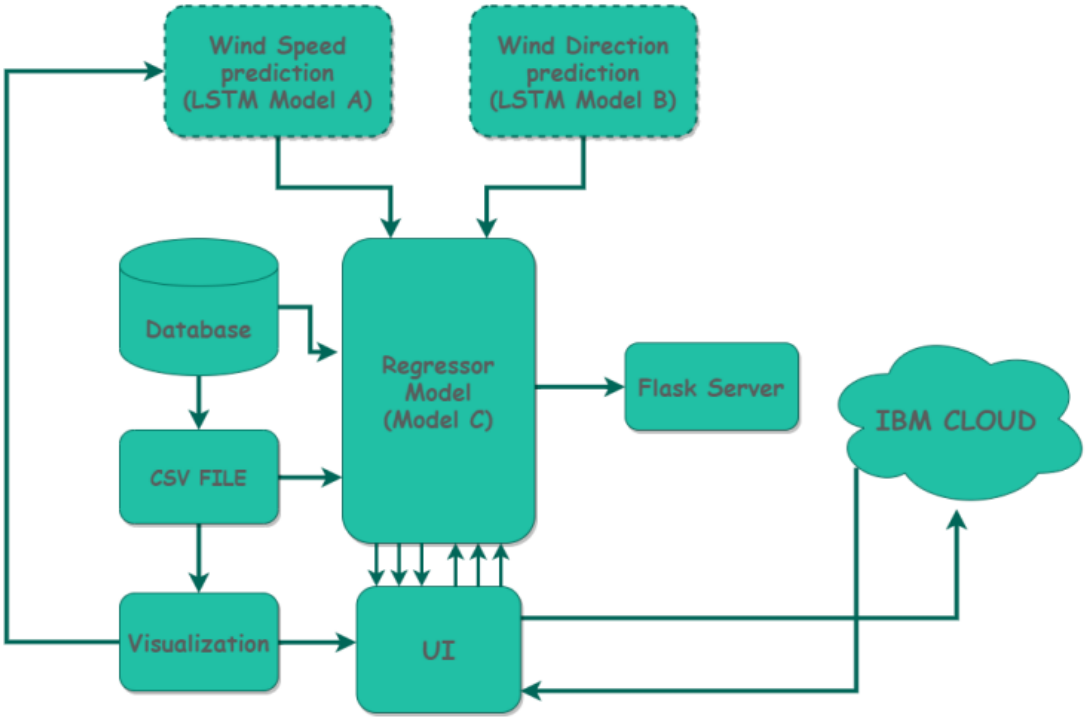


Figure 1:Workflow Diagram

2.2 Data Preprocessing and Cleaning:

2.2.1 Data Cleaning:

Remove any NULL values if present

Remove any and all negatives

To handle this part, data cleaning is done.

2.2 Feature Engineering:

Preparing the proper input dataset, compatible with the machine learning algorithm requirements. - Improving the performance of machine learning models.

2.3 Exploratory Data Analysis:

Exploratory Data Analysis refers to the critical process of performing initial investigations on data so as to discover patterns, to spot anomalies, to test hypothesis and to check assumptions with the help of summary statistics and graphical representations.

Following are some plots we used to extract some useful information

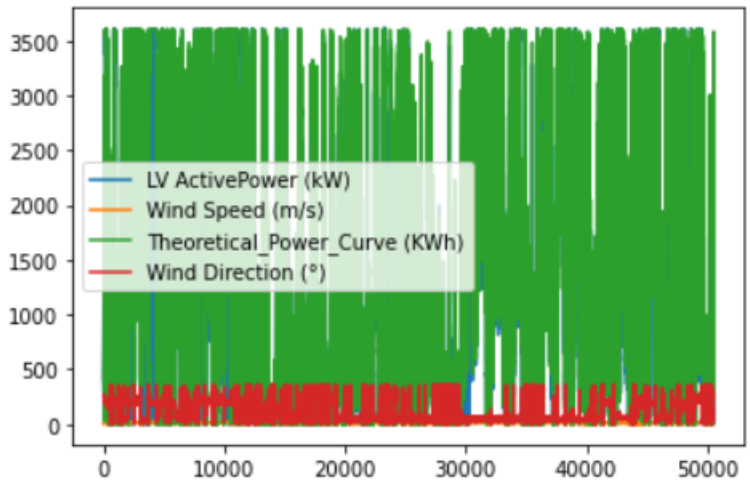


Figure 2: Graph showing the plot of the dataset

EXPLANATION:

Here we can see that our dataset distribution along 4 types of column namely LVActivePower(KW), Windspeed(m/s), Theoretical_Power_Curve (KWh), Wind Direction (°).

Figure 3 : correlation between the attributes of dataset

EXPLANATION:

Here we can see that our correlation of 4 types of column namely LVActivePower(KW), Windspeed(m/s), Theoretical_Power_Curve (KWh), Wind Direction (°) in which the dark areas shows the correlation between the columns while the lighter areas shows there is no correlation between the attributes.

```
[ ]  series['Wind Speed (m/s)'].plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd03f75df10>
```

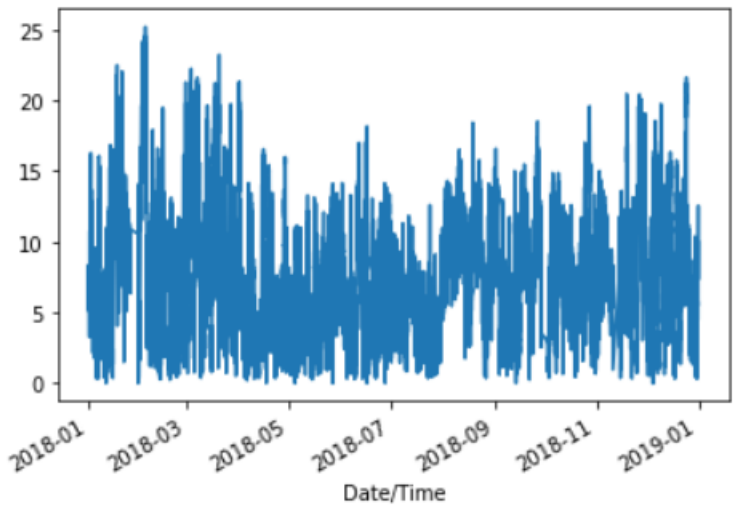


Figure 4 : Distribution of Wind speed(m/s)

EXPLANATION:

Here we can see that our Windspeed(m/s) distribution , with respect to Date/Time.

```
series['Wind Direction (°)'].plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd0414095d0>
```

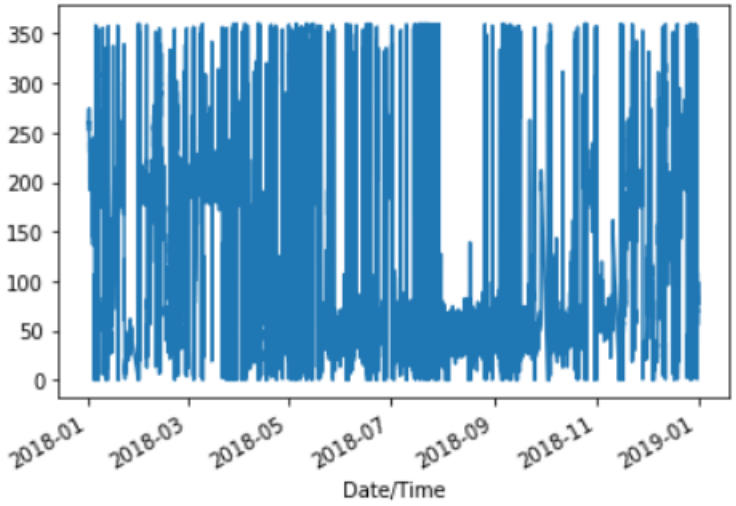


Figure 5 : Distribution of Wind direction(°)

EXPLANATION:

Here we can see that our Wind direction(°) distribution , with respect to Date/Time

.

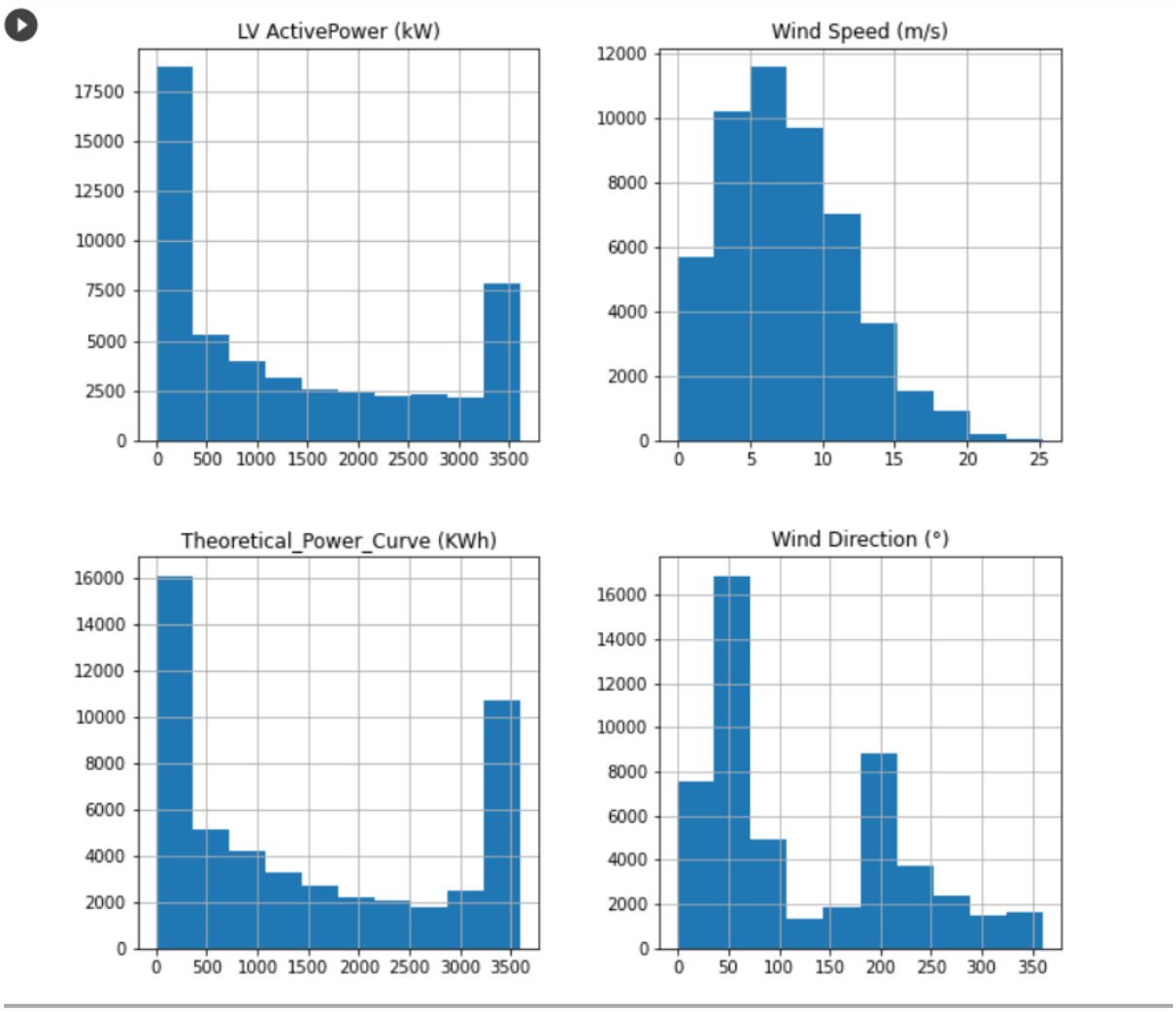


Figure 6: Histogram graph of attributes

EXPLANATION:

Here we can see that our dataset is not normally distributed, so we have to scale it for normalization.

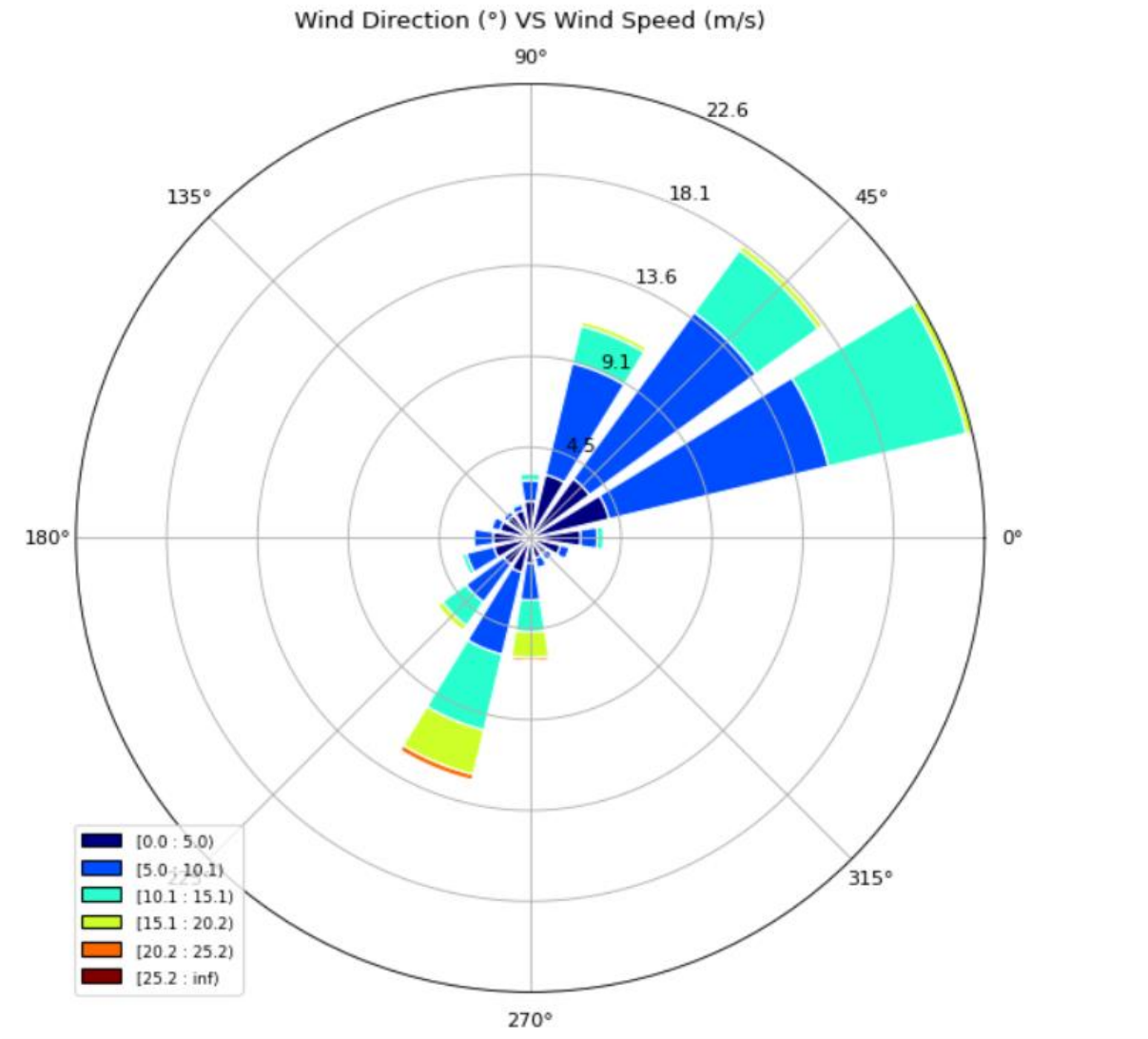


Figure 7: Pie chart of wind speed and wind direction

EXPLANATION:

Here we can see our chart of Windrose of Wind direction(°) and Wind Speed(m/s) which tells us about the wind rose is the time honored method of graphically presenting the wind conditions, direction and speed, over a period of time at a specific location.

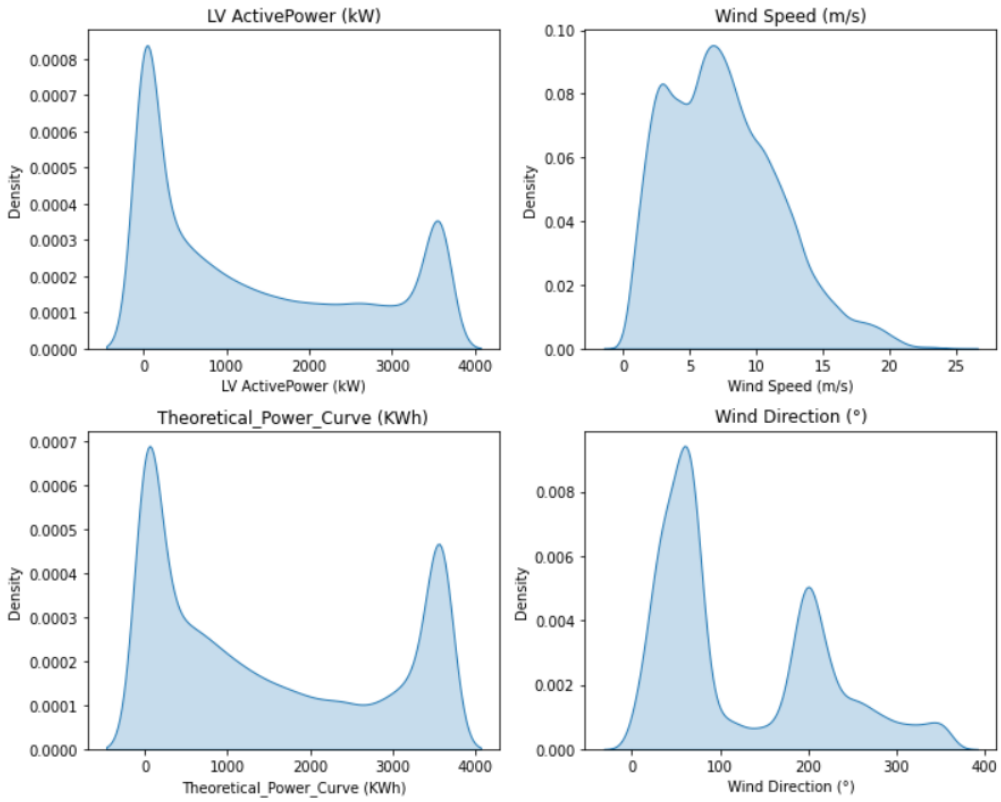


Figure 8 : KDE plot of all the attributes of the dataset

EXPLANATION:

A kernel density estimate (KDE) plot is used for visualizing the distribution of observations in a dataset, analagous to a histogram. KDE represents the data using a continuous probability density curve in one or more dimensions i.e. of LVActivePower(KW), Windspeed(m/s), Theoretical_Power_Curve (KWh), Wind Direction (°).

2.4. Data stationarity:

We have used Augmented Dickey-Fuller Test on the main columns of our data i.e. Wind speed and Wind direction that make strong assumptions about data.

They can only be used to inform the degree to which a null hypothesis can be rejected or fail to be rejected.

However, they provide a quick check and confirmatory evidence that the time series is stationary or non-stationary

```
##WIND SPEED
from statsmodels.tsa.stattools import adfuller
import random
random.seed(7)
# ADF Test
#Augmented Dickey-Fuller test
result = adfuller(series["Wind Speed (m/s)"].values, autolag='AIC')
print(f'ADF Statistic: {result[0]}')
print(f'p-value: {result[1]}')
for key, value in result[4].items():
    print('Critial Values:')
    print(f'   {key}, {value}')
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tools/_testing.py:19: FutureWarn
  import pandas.util.testing as tm
ADF Statistic: -14.932536033900613
p-value: 1.3595630952781274e-27
Critial Values:
   1%, -3.4304794464033166
Critial Values:
   5%, -2.861597212664768
Critial Values:
   10%, -2.566800452407002
```

Figure 9 : ADF test of Wind speed(m/s)

```
##WIND DIRECTION
from statsmodels.tsa.stattools import adfuller
import random
random.seed(7)
# ADF Test
#Augmented Dickey-Fuller test
result = adfuller(series["Wind Direction (°)"].values, autolag='AIC')
print(f'ADF Statistic: {result[0]}')
print(f'p-value: {result[1]}')
for key, value in result[4].items():
    print('Critial Values:')
    print(f'   {key}, {value}')
```

```
ADF Statistic: -12.650255902083122
p-value: 1.3732987440401944e-23
Critial Values:
   1%, -3.430479546414538
Critial Values:
   5%, -2.8615972568667196
Critial Values:
   10%, -2.566800475934405
```

Figure 10 : ADF test of Wind direction (°)

EXPLANATION:

Here, we can say that our data is stationary as the null hypothesis is accepted.

2.5 Model Building:

1. Train/Test split: One important aspect of all machine learning models is to determine their accuracy. Now, in order to determine their accuracy, one can train the model using the given dataset and then predict the response values for the same dataset using that model and hence, find the accuracy of the model. A better option is to split our data into two parts: first one for training our machine learning model, and second one for testing our model.

• Split the dataset into two pieces: a training set and a testing set.

• Train the model on the training set.

• Test the model on the testing set, and evaluate how well our model did.

Advantages of train/test split:

• Model can be trained and tested on different data than the one used for training.

• Response values are known for the test dataset, hence predictions can be evaluated

• Testing accuracy is a better estimate than training accuracy of out-of-sample performance.

Machine learning consists of algorithms that can automate analytical model building. Using algorithms that iteratively learn from data, machine learning models facilitate computers to find hidden insights from Big Data without being explicitly programmed where to look.

We have used the following three algorithms to build predictive model.

2. LSTM Model: It is special kind of recurrent neural network that is capable of learning long term dependencies in data. This is achieved because the recurring module of the model has a combination of four layers interacting with each other.

We have tried to build the LSTM model on Wind direction and Wind speed of our dataset so that we can get the prediction on both.

```
[ ]  lstm_model = fit_lstm(train_scaled, batch_size_exp, epoch_exp, neurons_exp)

     50447/50447 [==============================] - 150s 3ms/step - loss: 0.0111
     50447/50447 [==============================] - 143s 3ms/step - loss: 0.0109
     50447/50447 [==============================] - 116s 2ms/step - loss: 0.0107
     50447/50447 [==============================] - 127s 3ms/step - loss: 0.0107
     50447/50447 [==============================] - 111s 2ms/step - loss: 0.0106
     50447/50447 [==============================] - 112s 2ms/step - loss: 0.0105
     50447/50447 [==============================] - 112s 2ms/step - loss: 0.0105
```

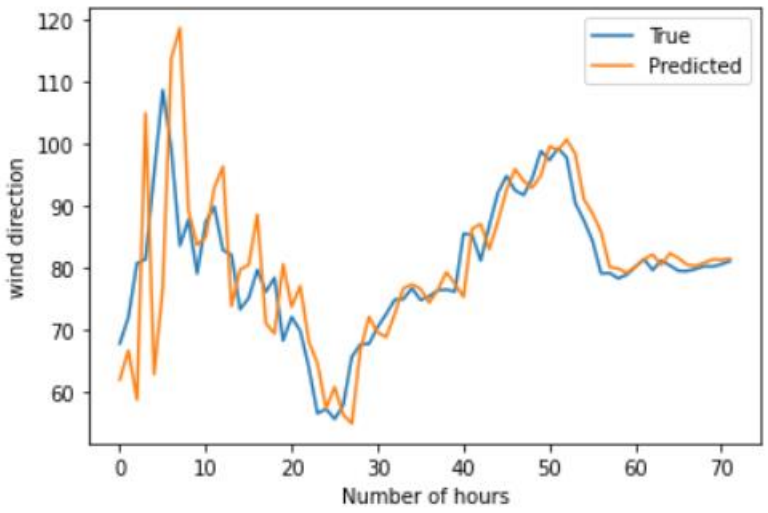


Figure 11 : LSTM model, True and prediction values of Wind direction (°)

EXPLANATION:

Here, LSTM model of the wind direction with respect to number of hours shows the loss that occur in the column is less. So that, our graph shows the True and predicted values od the wind direction, which are nearly accurate.

```
[ ]  lstm_model_speed = fit_lstm(train_scaled, batch_size_exp, epoch_exp, neurons_exp
```

```
50447/50447 [==============================] - 112s 2ms/step - loss: 0.0018
50447/50447 [==============================] - 108s 2ms/step - loss: 0.0018
50447/50447 [==============================] - 109s 2ms/step - loss: 0.0018
50447/50447 [==============================] - 108s 2ms/step - loss: 0.0018
50447/50447 [==============================] - 109s 2ms/step - loss: 0.0018
50447/50447 [==============================] - 109s 2ms/step - loss: 0.0018
50447/50447 [==============================] - 109s 2ms/step - loss: 0.0018
```

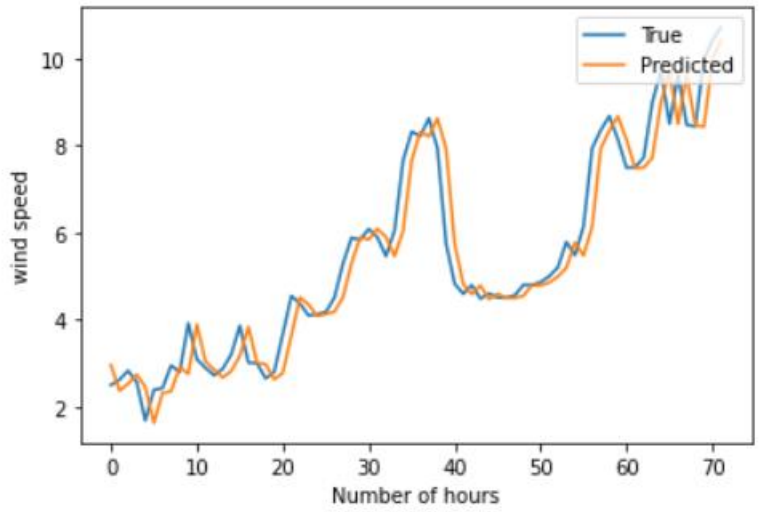


Figure 12 : LSTM model, True and prediction values of Wind speed(m/s)

EXPLANATION:

Here, LSTM model of the wind speed with respect to number of hours shows the loss that occur in the column is less. So that, our graph shows the True and predicted values of the wind speed, which are nearly accurate.

ARIMA Model

Autoregressive Integrated Moving Averages

The general process for ARIMA models is the following:

- Visualize the Time Series Data
- Make the time series data stationary
- Plot the Correlation and AutoCorrelation Charts
- Construct the ARIMA Model or Seasonal ARIMA based on the data
- Use the model to make predictions

Apart from LSTM, we have also tested our dataset by using ARIMA model on the columns Windspeed(m/s),Wind Direction (°).

```
[22] # Fit your model
     model = pm.auto_arima( train, start_p=2,start_q=2, seasonal=False, trace = True,random =True) #seasonal =False
     # random search
     # make your forecasts
     forecasts = model.predict(test.shape[0])

     Performing stepwise search to minimize aic
      ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=79964.617, Time=36.27 sec
      ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=80469.367, Time=7.14 sec
      ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=80350.795, Time=4.82 sec
      ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=80327.802, Time=5.86 sec
      ARIMA(0,1,0)(0,0,0)[0]             : AIC=80467.367, Time=0.64 sec
      ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=79974.807, Time=20.86 sec
      ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=79975.565, Time=21.19 sec
      ARIMA(3,1,2)(0,0,0)[0] intercept   : AIC=79930.748, Time=83.26 sec
      ARIMA(3,1,1)(0,0,0)[0] intercept   : AIC=inf, Time=62.86 sec
      ARIMA(4,1,2)(0,0,0)[0] intercept   : AIC=inf, Time=92.78 sec
      ARIMA(3,1,3)(0,0,0)[0] intercept   : AIC=79965.420, Time=50.05 sec
      ARIMA(2,1,3)(0,0,0)[0] intercept   : AIC=79978.714, Time=33.60 sec
      ARIMA(4,1,1)(0,0,0)[0] intercept   : AIC=inf, Time=95.67 sec
      ARIMA(4,1,3)(0,0,0)[0] intercept   : AIC=inf, Time=103.64 sec
      ARIMA(3,1,2)(0,0,0)[0]             : AIC=inf, Time=29.51 sec

     Best model:  ARIMA(3,1,2)(0,0,0)[0] intercept
     Total fit time: 648.219 seconds
```

```
mean_squared_error(test,output)**0.5# 3,1,2

4.812397288906385
```

Figure 13 : ARIMA model and RMSE of Wind speed(m/s)

EXPLANATION:

Here, ARIMA model of the wind speed shows the best model that occur in the column and according to that model we got respective Root Mean Squared error.

```
[23] # Fit your model
model = pm.auto_arima( train, start_p=2,start_q=2, seasonal=False, trace = True,random =True) #seasonal =False
# random search
# make your forecasts
forecasts = model.predict(test.shape[0])
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=358103.278, Time=85.59 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=364768.796, Time=1.03 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=361338.612, Time=2.36 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=359482.835, Time=7.86 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=364766.797, Time=0.44 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=358336.620, Time=23.87 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=358351.174, Time=21.24 sec
 ARIMA(3,1,2)(0,0,0)[0] intercept   : AIC=358355.104, Time=37.88 sec
 ARIMA(2,1,3)(0,0,0)[0] intercept   : AIC=358177.139, Time=80.57 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=358409.569, Time=15.47 sec
 ARIMA(1,1,3)(0,0,0)[0] intercept   : AIC=358172.978, Time=51.42 sec
 ARIMA(3,1,1)(0,0,0)[0] intercept   : AIC=358338.836, Time=33.37 sec
 ARIMA(3,1,3)(0,0,0)[0] intercept   : AIC=inf, Time=84.73 sec
 ARIMA(2,1,2)(0,0,0)[0]             : AIC=358096.536, Time=23.18 sec
 ARIMA(1,1,2)(0,0,0)[0]             : AIC=358334.629, Time=9.35 sec
 ARIMA(2,1,1)(0,0,0)[0]             : AIC=358349.183, Time=8.20 sec
 ARIMA(3,1,2)(0,0,0)[0]             : AIC=358353.111, Time=13.04 sec
 ARIMA(2,1,3)(0,0,0)[0]             : AIC=358101.225, Time=32.48 sec
 ARIMA(1,1,1)(0,0,0)[0]             : AIC=358407.574, Time=5.20 sec
 ARIMA(1,1,3)(0,0,0)[0]             : AIC=358171.001, Time=18.14 sec
 ARIMA(3,1,1)(0,0,0)[0]             : AIC=358336.845, Time=12.44 sec
 ARIMA(3,1,3)(0,0,0)[0]             : AIC=inf, Time=32.19 sec

Best model:  ARIMA(2,1,2)(0,0,0)[0]
Total fit time: 600.129 seconds
```

```
[33] mean_squared_error(test,output)**0.5# 2,1,2

     90.07887604772485
```

Figure 14 : ARIMA model and RMSE of Wind direction()

EXPLANATION:

Here, ARIMA model of the wind direction shows the best model that occur in the column and according to that model we got respective Root Mean Squared error.

Model Stacking

Stacking or Stacked Generalization is an ensemble machine learning algorithm.

It uses a meta-learning algorithm to learn how to best combine the predictions from two or more base machine learning algorithms.

The benefit of stacking is that it can harness the capabilities of a range of well-performing models on a classification or regression task and make predictions that have better performance than any single model in the ensemble.

We have used Model stacking by combining our both the LSTM model of Wind speed and Wind direction by downloading the model in csv format and then loading it to different regressor models to compare the best model and to give the correct prediction of the output generated by the Wind Turbine.

We have used different regressor models as follows:

1. Linear Regression: Linear regression is a **linear model**, e.g. a model that assumes a linear relationship between the input variables (x) and the single output variable (y). More specifically, that y can be calculated from a linear combination of the input variables (x).
When there is a single input variable (x), the method is referred to as **simple linear regression**. When there are **multiple input variables**, literature from statistics often refers to the method as multiple linear regression.

2. Random Forest

Random forest is a ***Supervised Machine Learning Algorithm*** that is ***used widely in Classification and Regression problems***. It builds decision trees on different samples and takes their majority vote for classification and average in case of regression.One of the most important features of the Random Forest Algorithm is that it can handle the data set containing ***continuous variables*** as in the case of regression and ***categorical variables*** as in the case of classification. It performs better results for classification problems.

Steps involved in random forest algorithm:

Step 1: In Random forest n number of random records are taken from the data set having k number of records.
Step 2: Individual decision trees are constructed for each sample.
Step 3: Each decision tree will generate an output.
Step 4: Final output is considered based on ***Majority Voting or Averaging*** for Classification and regression respectively.

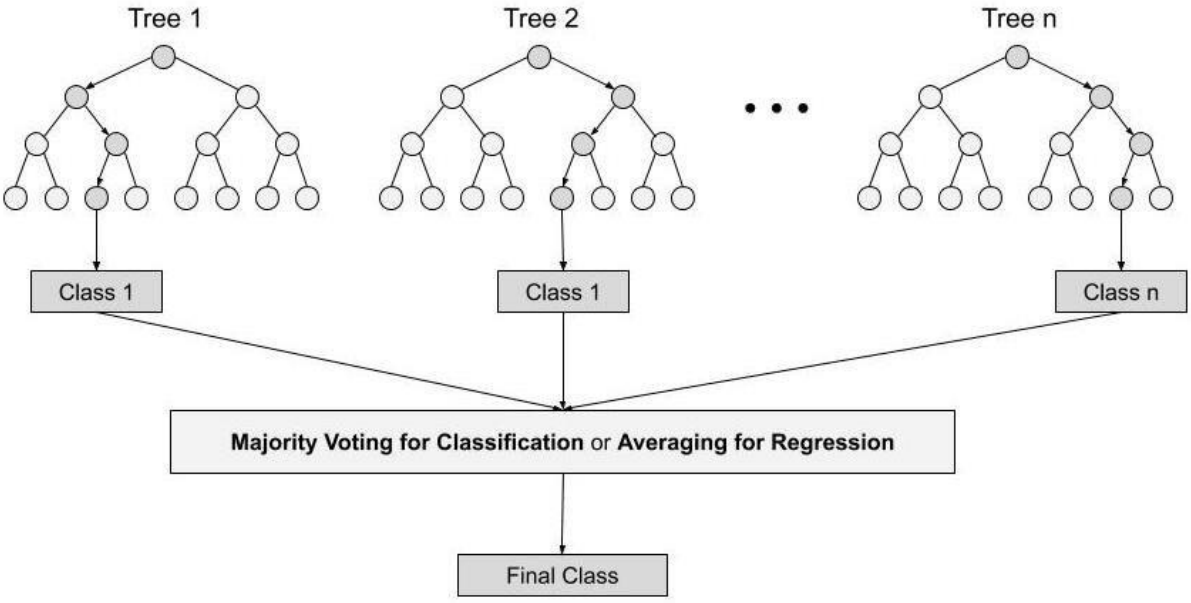


Figure 15 : Random forest regressor

3. XG Boost

XgBoost stands for Extreme Gradient Boosting.

XGBoost is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework. In prediction problems involving unstructured data (images, text, etc.) artificial neural networks tend to outperform all other algorithms or frameworks. However, when it comes to small-to-medium structured/tabular data, decision tree based algorithms are considered best-in-class right now.

4.SVR

Support Vector Machine (SVM) is a very popular Machine Learning algorithm that is used in both Regression and Classification. Support Vector Regression is similar to Linear Regression in that the equation of the line is $y= wx+b$ In SVR, this straight line is referred to as ***hyperplane***. The data points on either side of the hyperplane that are closest to the hyperplane are called ***Support Vectors*** which is used to plot the boundary line.

Unlike other Regression models that try to minimize the error between the real and predicted value, the SVR tries to fit the best line within a threshold value (Distance between hyperplane and boundary line), **a**. Thus, we can say that SVR model tries satisfy the condition $-a < y-wx+b < a$. It used the points with this boundary to predict the value.

5.Voting Regressor

Prediction voting regressor for unfitted estimators.

We have used voting regressor to form the final prediction.

A voting regressor is an ensemble meta-estimator that fits several base regressors, each on the whole dataset. Then it averages the individual predictions to form a final prediction.

```
xgr=XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=0.7,
             colsample_bynode=1, colsample_bytree=0.3, gamma=0.2,
             importance_type='gain', learning_rate=0.03, max_delta_step=0,
             max_depth=8, min_child_weight=25, missing=None, n_estimators=800,
             n_jobs=1, nthread=None, objective='reg:linear', random_state=0,
             reg_alpha=0.2, reg_lambda=0.8, scale_pos_weight=1, seed=None,
             silent=None, subsample=0.1, verbosity=1)
```

```
[ ] sm=SVR(gamma='auto',C=50,epsilon=0.3)
```

```
[ ] rf=RandomForestRegressor(n_estimators=500,max_depth=4)
```

```
[ ] lr=LinearRegression()
```

```
[ ] model=VotingRegressor([('lr',lr), ('rf',rf),('sm', sm),('xgr',xgr)],weights=[1,1,2,3])
```

```
[ ] Model=model.fit(X_train, y_train)
```

```
[12:52:29] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
```

```
[20] y_pred=Model.predict(X_test)
     print(y_pred)
     print('R2',r2_score(y_test,y_pred))
     print('RMSE',np.sqrt(mean_squared_error(y_test,y_pred)))
```

```
[ -45.87290324 1141.22807139  189.93258528 ... 1239.42880128  496.29144379
 3261.16924561]
R2 0.909550394464911
RMSE 394.46098818564974
```

Figure 16 : Final Prediction using voting regressor

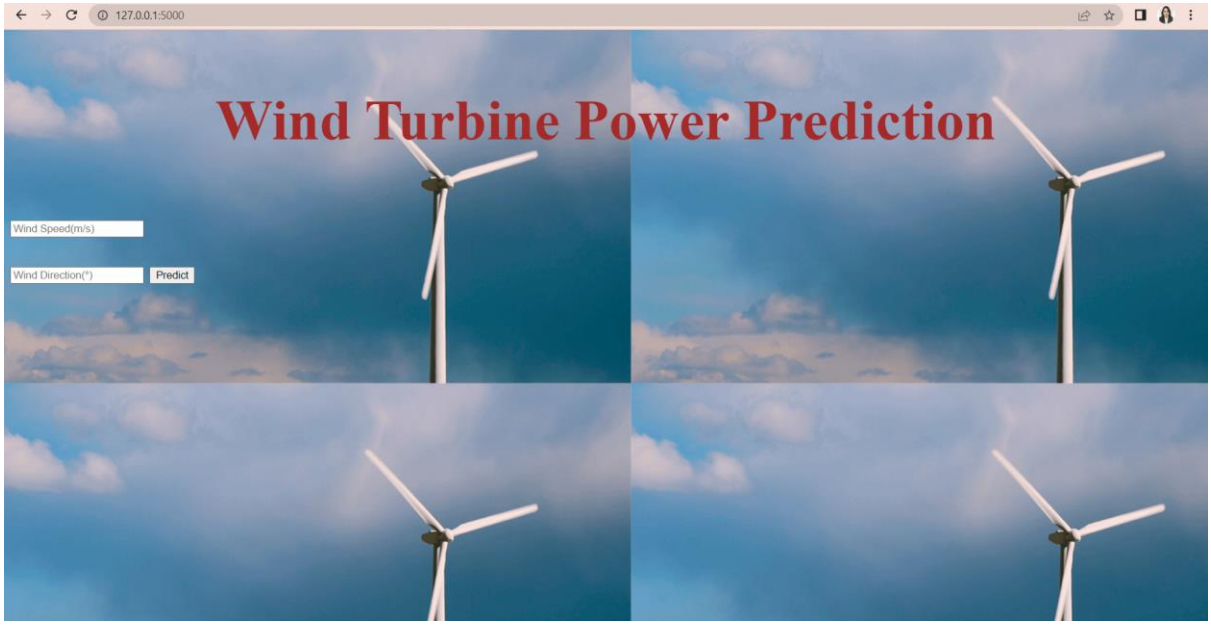
EXPLANATION:

By using voting regression, we have used different model and voting regressor selects the best model among all and give the prediction.

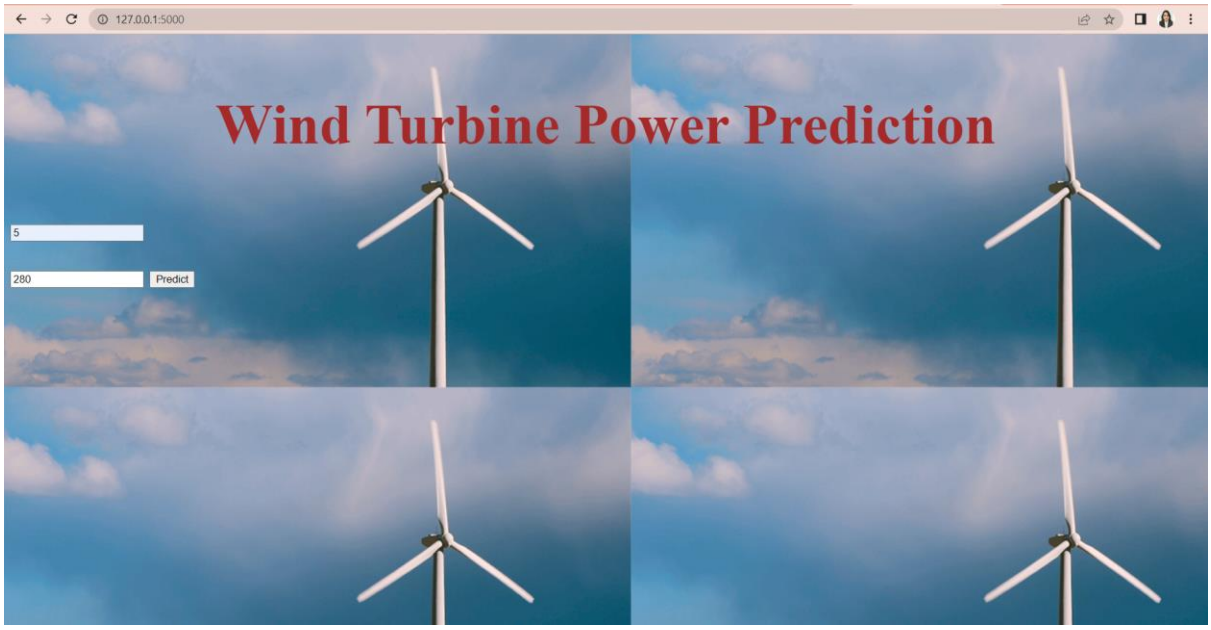
In above figure we got the R2 score and RMSE that shows the prediction of our model.

3. User Interface

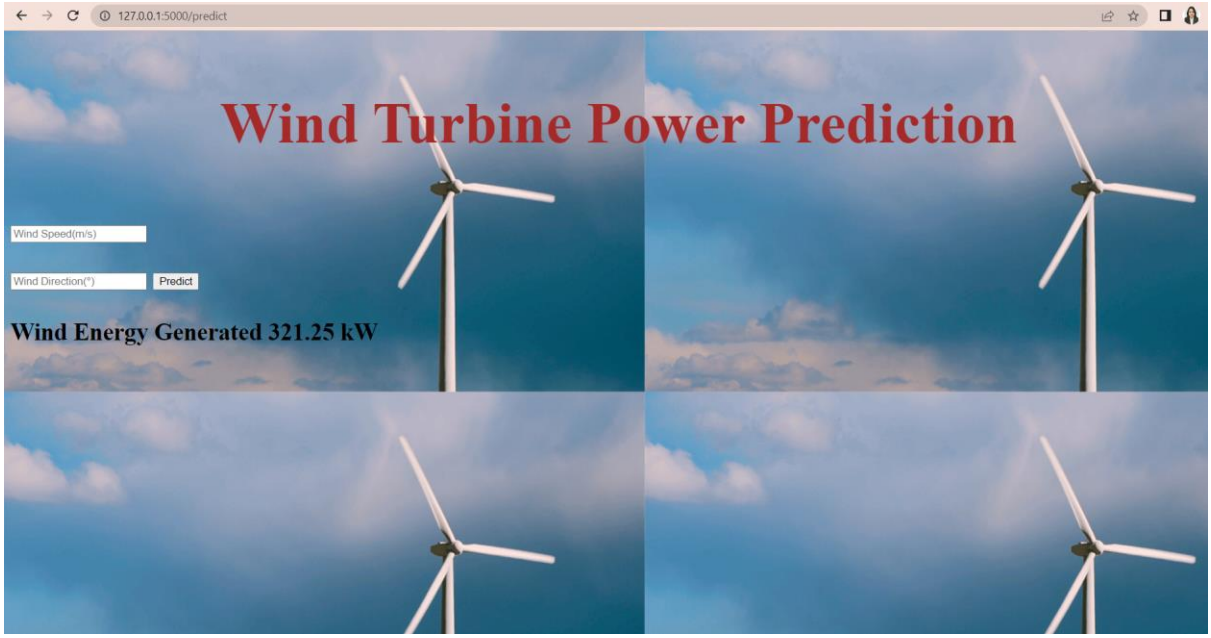
After Training the models and finding out the best model to be accurate predictor, we can go ahead with building the user interface for our models. This will give us a clean and simple way of accessing our models and make the predictions of the energy output generation. For building the user interface we adopted the Flask Web Framework which is very easy to use and robust. Here are a few screenshots of the web app delivering the quality prediction models.



Giving the input:



Getting the prediction:



4. Requirements Specification

4.1 Hardware Requirement:

• 500 GB hard drive (Minimum requirement)

• 8 GB RAM (Minimum requirement)

• PC x64-bit CPU

4.2 Software Requirement:

• Windows/Mac/Linux

• Python-3.9.1

• VS Code/Anaconda/Spyder •

Python Extension for VS Code •

Libraries:

➢ Numpy 1.18.2

➢ Pandas 1.2.1

➢ Matplotlib 3.3.3

➢ Scikit-learn 0.24.1

➢ Flask 1.1.2

• Any Modern Web Browser like Google Chrome

➢ To access the web application written in Flask

5. Conclusion:

- In this project, we have established the application to predict future wind power output values based on the regressor and deep learning models.
- The UI provides a great deal of information to anyone who would like to know about the future power output presented in the form of visualizations.
- Deploying it to the cloud makes it more scalable.

6. Future Scope

- Our attempt would be to further improve the predictions using the ARIMA model and other models that are powerful.
- Imparting more features (like location, due level, humidity, etc) to our training set will enhance the predictions and will open up a new perspective on every front of wind turbine prediction

7. References

- Predicting the power generated by wind turbine for the next 72 hours https://www.kaggle.com/datasets/berkerisen/wind-turbine-scada-dataset

- Took help from some youtubers https://www.youtube.com/results?search_query=wind+energy+power+prediction