IBM Hack Challenge 2020

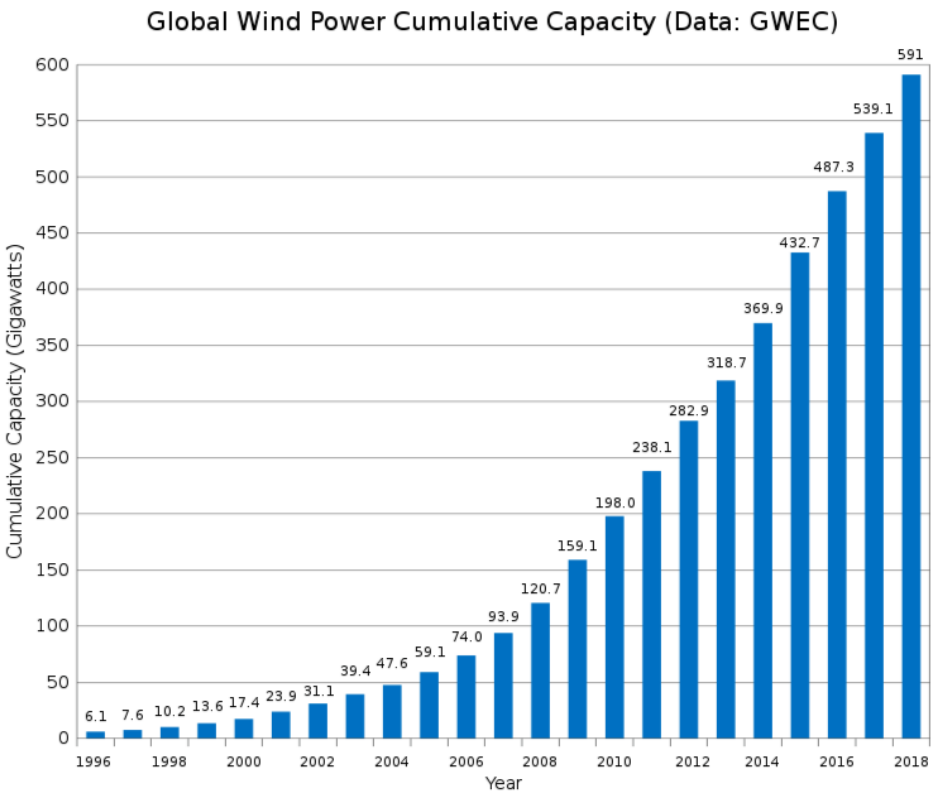
Predicting the energy output of wind turbine based on weather condition

1. INTRODUCTION

1.1 Overview

As the world moves forward, renewable energy sources will play a crucial role in promoting a green, sustainable society and hence, will directly impact on the health and well-being of all humans across Earth.

Renewable sources are expected to make about 45 per cent of the global electricity demand by 2040. Today, wind energy accounts for 651 GW of electrical power, which accounts for more than 5% of global demand. With the continuous, exponential growth of wind farms and the development of more efficient wind turbines, the impact and contribution of wind energy for energy is only going to increase. Therefore, there is an ever-increasing need for a solution which can efficiently integrate various conventional power sources with wind farms to reduce overproduction by conventional sources which can unnecessarily contribute towards pollution. By collaborating traditional and renewable sources, we can efficiently match demand with supply without excessive production and therefore save up on monetary assets as well.

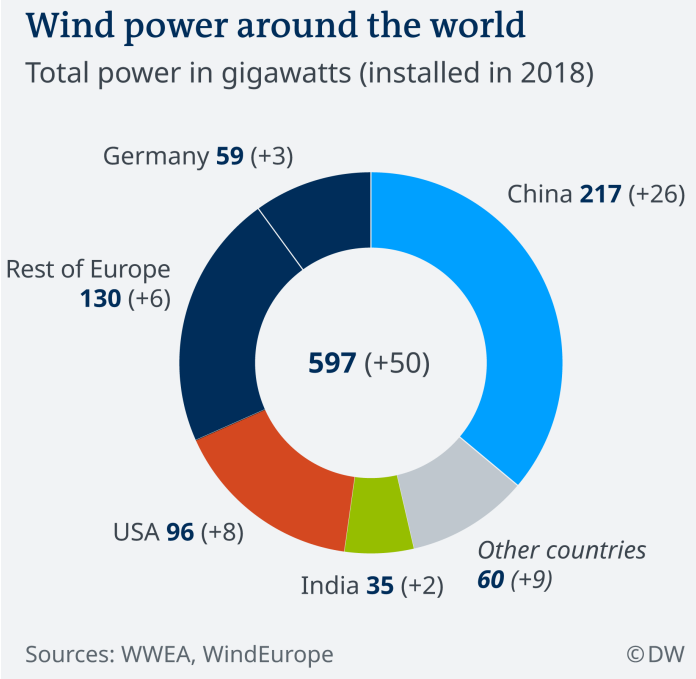


Our AI model predicts the Active power output of a wind turbine using the weather data of the previous five days and forecasts the same for the next 3 days. Thus, an effective management system can be set up between various energy sources.

1.2 Purpose and Objectives

The objective of our proposed solution is to provide our client with a pragmatic solution for better collaboration between wind farms and conventional energy sources. Our motive is to ensure that the energy demand and supply is efficiently optimized so as to minimize overproduction and monetary loss.

Our solution model includes the implementation of a time series model derived from deep learning to predict future electrical power production by wind turbines.



We have implemented a web-based application which will predict the power output of the wind turbine given the current weather conditions for the next 72 hours. The app will provide a visual representation of the power requirement versus the power currently generated. The app will automatically notify the client in case the future demand is higher than the supply provided by the wind farm. It will also act as a portal to collaborate with alternative power sources in case the energy requirements are not fulfilled by the wind farm. The custom-designed website will enhance the integration of various power plants.

2. LITERATURE SURVEY

2.1 Existing Problem

Levels of production of wind energy are hard to predict as they rely on potentially unstable weather conditions present at the wind farm. In particular, wind speed is crucial for energy production based on wind, and it may vary drastically over time.

Energy suppliers are interested in accurate predictions, as they can avoid overproduction by coordinating the collaborative production of traditional power plants and weather-dependent energy sources. This will ultimately result in substantial monetary saving.

2.2 Proposed solution

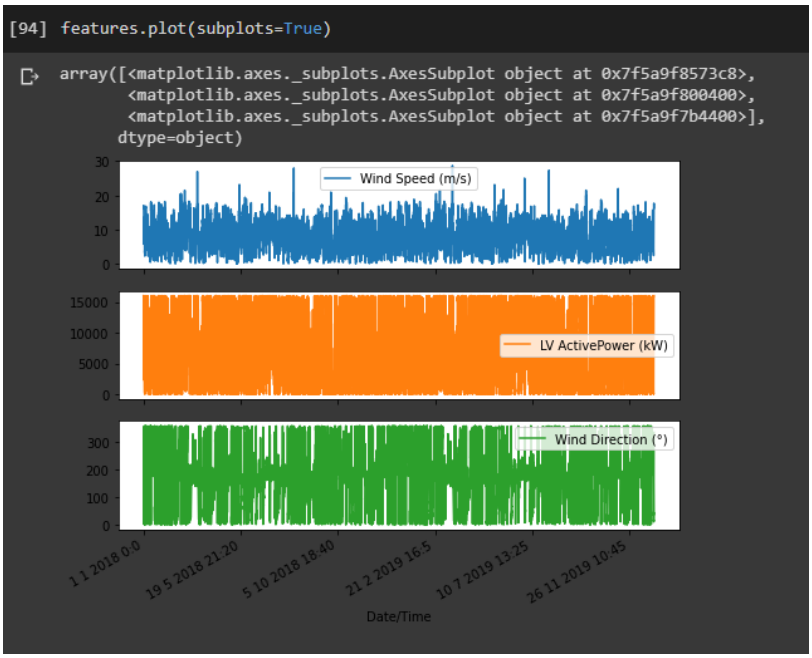
We have devised a novel solution for overcoming the stated problem. Our solution consists of two components which have been integrated for a holistic and complete energy management system for efficient collaboration between wind farms and conventional sources of electricity.

(a) The LSTM Neural Network model

After testing multiple Neural Network algorithms, we decided to finalize Long Short Term Memory (LSTM) for predicting the power output of the turbines. LSTM being a type of Recurrent Neural Network is an algorithm which is efficient for time series forecasting, wherein you require long term dependencies. This was crucial for our project as we were required to predict the power output given a set of previous and present weather information. As LSTM is very capable of handling data for long periods, which is required in our case, it performed much better for the same data set than other machine learning models like linear regression, logistic regression etc.

```
[94] features.plot(subplots=True)
```

```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x7f5a9f8573c8>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f5a9f800400>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f5a9f7b4400>],
      dtype=object)
```



We selected "Adam" as our optimizer due to the fact it performs much better than Root mean square(RMS prop) and the weighted mean average for data with a lot of noise/sparse gradients (which is generally the case for weather data sets). Adam was also chosen as it is computationally very efficient as well.

For our loss function, we went with "Mean Absolute Error" as it gave better accuracy and took less to optimize the model when compared to Mean Square error.

A total of 250 epochs were performed on the data set.

The data set was compiled using the Supervisory Control and Data Acquisition (SCADA) system integrated with modern wind

```
[105120 rows x 4 columns]
```

```
[93] df = dataset
     features_considered = ['Wind Speed (m/s)', 'LV ActivePower (kW)', 'Wind Direction (°)']
     features = df[features_considered]
     features.index = df['Date/Time']
     features.head()
```

Date/Time	Wind Speed (m/s)	LV ActivePower (kW)	Wind Direction (°)
1 1 2018 0:0	16.418	16000	336.296
1 1 2018 0:10	16.653	16000	337.510
1 1 2018 0:20	16.919	16000	337.927
1 1 2018 0:30	16.989	16000	338.174
1 1 2018 0:40	17.096	16000	338.433

turbines. It consisted of 1,05,120 rows of multi-variate data. After careful evaluation of inter dependencies of the various variables using graphical and statistical methods, it was concluded that the Active power output of the wind turbines had a maximum correlation with Wind speed and wind direction. To reduce the complexity of the model, other variables were dropped. The data set was then split into the training data, which amounted to 52,561 rows of data and test data, which amounted to 52,559 rows of data. For predictions, a window of five days weather data was taken as input. For each hour, one set of data was recorded, totaling to 120 weather input. An output prediction of 12 hours is given using the provided data. For each hour, six predictions for every ten minutes is made, totaling to 72 active power output predictions for the next 12 hours.

```
[5]: dataset.info

[5]: <bound method DataFrame.info of              Date/Time  ...  Wind Speed (m/s)
     0           1 1 2018 0:0  ...           16.418
     1          1 1 2018 0:10  ...           16.653
     2          1 1 2018 0:20  ...           16.919
     3          1 1 2018 0:30  ...           16.989
     4          1 1 2018 0:40  ...           17.096
     ...                  ...  ...              ...
     105115  31 12 2019 23:15  ...           17.670
     105116  31 12 2019 23:25  ...           16.965
     105117  31 12 2019 23:35  ...           16.214
     105118  31 12 2019 23:45  ...           16.116
     105119  31 12 2019 23:55  ...           16.345

     [105120 rows x 4 columns]>
```

(b) Web-Based Application

To make our Machine Learning model user friendly and help the client to visualize the output better, we have created a Web Application using Flask, which will be deployed on Heroku. The web-app will have three sections, Home Page, About Page and Statistics Page. We have used the OpenWeatherMap API to get the current and historical data (Code snippet for real time data collection shown in the picture). The trained ML

```
91  API_KEY = "<API KEY>"
92  CITY = "Istanbul"
93  url = f"http://api.openweathermap.org/data/2.5/weather?q={CITY}&appid={API_KEY}"
94  print(url)
95
96  hist = pd.read_csv("historical.csv")
97  if int(hist[-1:]["date_time"].values[0][:2]) != int((datetime.now().day)-1):
98      update_data()
99  plt.style.use("fivethirtyeight")
100
101 resp = requests.get(url)
102 resp = resp.json()
103 data = {}
104 data['date'] = datetime.utcfromtimestamp(resp['dt']).strftime('%m-%d-%Y %H:%M')
105 data['windspeed'] = resp['wind']['speed']
106 data['winddir'] = resp['wind']['deg']
107 data['temp'] = round(resp['main']['temp']-273.15,3)
108 data['mintemp'] = round(resp['main']['temp_min']-273.15,3)
109 data['maxtemp'] = round(resp['main']['temp_max']-273.15,3)
```

model has been embedded into our application which will directly use the data received from the API and show the predicted output on the Home page.

```
def real_plot():
    data = pd.read_csv("historical.csv")

    model = tf.keras.models.load_model('model5.h5')

    pred_data = data[-120:]
    pred_data.index = pred_data["date_time"]
    pred_data = pred_data[["wind_speed","wind_direction"]]
    pred_data = pred_data.values
    pred_data_std = pred_data.std(axis=0)
    pred_data_mean = pred_data.mean(axis=0)
    dataset_mean = df.mean(axis = 0)[1]
    dataset_std = df.std(axis = 0)[1]
    x_predict_multi = (pred_data-pred_data_mean)/pred_data_std

    x_predict_multi= tf.data.Dataset.from_tensor_slices((x_predict_multi))

    x_predict_multi= x_predict_multi.batch(BATCH_SIZE_predict).repeat()
    x_predict_multi= x_predict_multi.batch(BATCH_SIZE_predict).repeat()
    predictions_raw = model.predict(x_predict_multi.take(1))
    predictions_array = (predictions_raw*dataset_std) + dataset_mean
    predictions = predictions_array[0]

    return predictions
```

Code snippet for integration of the model with the application

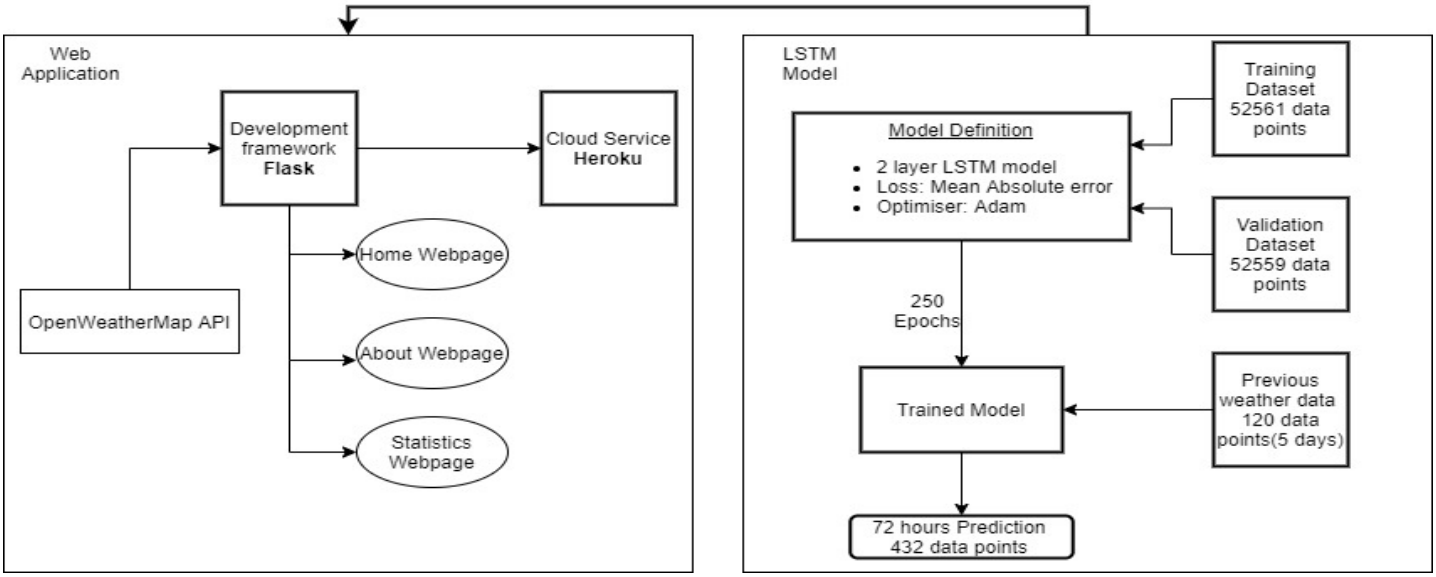
1. The Home page will have the information about the current weather status and will show a graph of expected power output for the next 72 hours as predicted by our ML model. This webpage will also have a section for the user to enter a few parameters, such as wind speed and direction, that will allow the user to get a rough estimate of output power from the wind turbine for the given condition.

2. The About page will have the information and details regarding wind energy, how it is being produced and some general facts. We will also mention the approach used by us and how it would help the client. Information of the IBM Hack Challenge and the team behind the project would be included as well.

3. The Statistics Page will have statistics about the power generated in the past based on the data analysis that we have done. The webpage will also have graphs which will make the numbers more interpretable and visually informative. It would plots which would correlations between various variables such as wind speed, wind direction, power output etc. Furthermore, it will also display the power generated each hour in the future in the form of a table.

3. THEORETICAL ANALYSIS

3.1 Block Diagram



3.2 Hardware Design and Requirements

A wind turbine with supervisory control and data acquisition (SCADA) system is required for continuous storage and update of present and previous weather conditions.

For training of the model, the Google Colab service was utilised. The GPU issued was an Nvidia Tesla K80 (12GB VRAM).

3.3 Software Design

The software design of our solution can be segregated into two components.

(a) Prediction Algorithm - It is a time series model, consisting of a LSTM Neural network developed in python using the Tensorflow framework and the Keras API. The model is a sequential neural network employing two layers of LSTM and one layer of Dense. It consists of 14,960 trainable parameters.

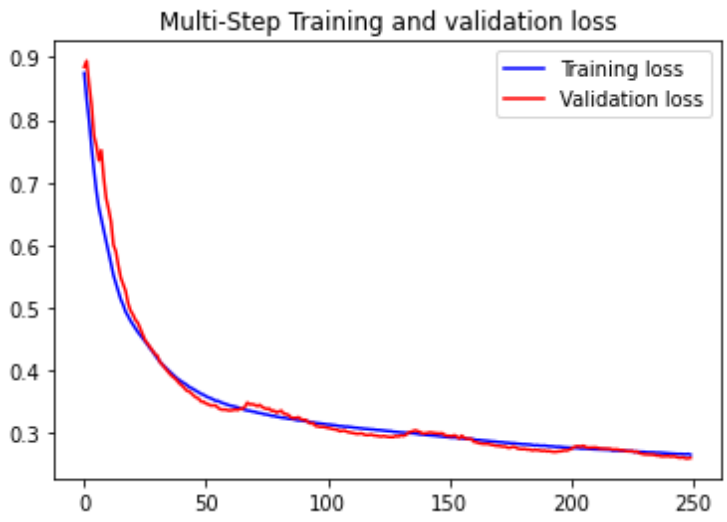
Our model is trained using a dataset comprising of 52,561 data points. A validation dataset with 52,559 data points was used for testing the accuracy of the model. We used Mean Absolute Error for loss calculation and Adam for optimization.

```
[18] # Defining the model
     multi_step_model = tf.keras.models.Sequential()
     multi_step_model.add(tf.keras.layers.LSTM(32,
                                               return_sequences=True,
                                               input_shape=x_train_multi.shape[-2:]))
     multi_step_model.add(tf.keras.layers.LSTM(16, activation='relu'))
     multi_step_model.add(tf.keras.layers.Dense(432))

     multi_step_model.compile(optimizer='adam', loss='mae')
```

```
[21] multi_step_model.summary()

Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 120, 32)           4480
_________________________________________________________________
lstm_1 (LSTM)                (None, 16)                3136
_________________________________________________________________
dense (Dense)                (None, 432)               7344
=================================================================
Total params: 14,960
Trainable params: 14,960
Non-trainable params: 0
```



Further, a training run of 250 epochs was found to be sufficient for loss to stagnate at a value of approximately 0.26.
For predictions, our model utilizes 120 data points of weather data of the previous five days, imported directly from the OpenWeatherMap API to give the power output predictions of the wind farm/turbine for the upcoming 72 hours.

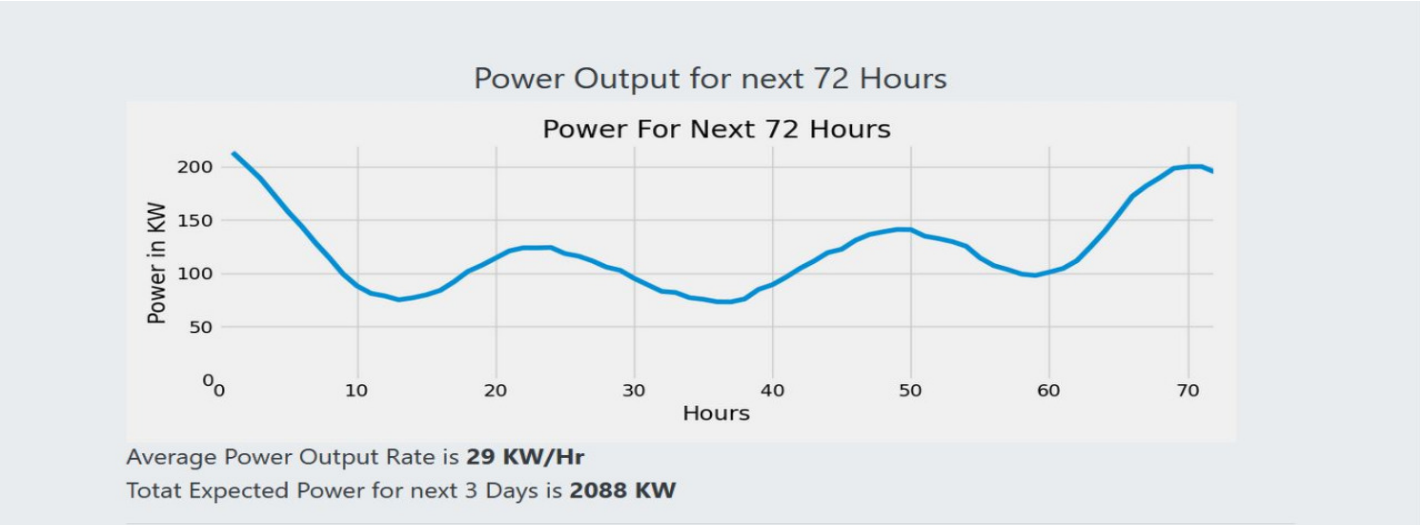
The model has been integrated with our Web Application deployed on Heroku, for ease of access and improved feasibility.

(b) Web Application - The Web-application was designed and created using flask, which is a micro web framework written in python. The web app utilises the OpenWeatherMap API to import and get live weather updates, from any pre-defined particular location. We have currently set Istanbul, Turkey, as our default location.

Home page

The historical weather data is collected from the API, updated once a day, and stored in a .csv extension file. The frequency of updates can be improved using a database.

The machine learning model is integrated into the application and automatically uses the data from the .csv file to predict the power output for the next 72 hours. Furthermore, a line chart is plotted and displayed on the home page as well as the statistics page.



Line Chart on the home page for the predicted output given by the model by importing live data

The About webpage has information about the team members, global wind energy scenario and some details about the implementation of our solution.

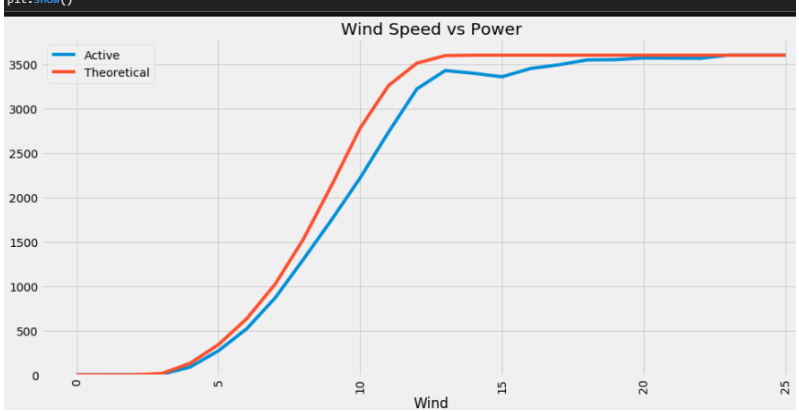
The Statistics page is a dashboard which has data of the predicted in the form of a table and some other graphical plots showing relations between various parameters, for example, Wind speed vs Power output, Wind direction vs Power output etc.

4. EXPERIMENTAL INVESTIGATIONS

The dataset utilised for the model contained many variables, including dew, humidity, air pressure, air density etc. All the variables were plotted against the Active power of the wind turbine using the Matplotlib module to determine the covariance of each variable with output power to analyse its impact and contribution.

Using visual and statistical analysis, we confirmed that Wind speed and Wind direction were the significant contributors toward Active output power and hence, other variables were dropped to reduce the complexity of model which ultimately decreased training time.

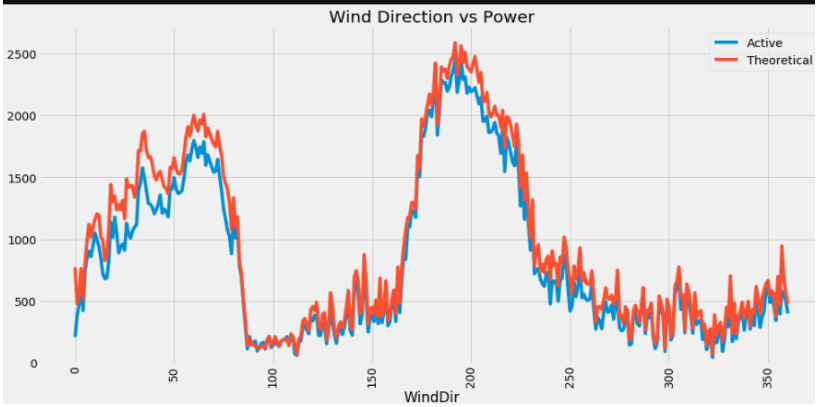
```
plt.ylim(0)
plt.xticks(rotation = 90)
plt.legend()
plt.title("Wind Speed vs Power")
plt.show()
```



As for our optimiser, a comparison of RMS prop and Adam and verified the loss gradient and the loss at which the model stagnated for a constant number of epochs. It was observed that Adam achieved a lower loss of approximately ~10 per cent than RMS prop.

Further, for the loss function, we finalised Mean Absolute Error and Mean Square Error as those two are the most used for time series forecasting. After analysis, keeping the number of epochs constant, it was validated that the model achieved a lower loss using Mean Absolute Error.

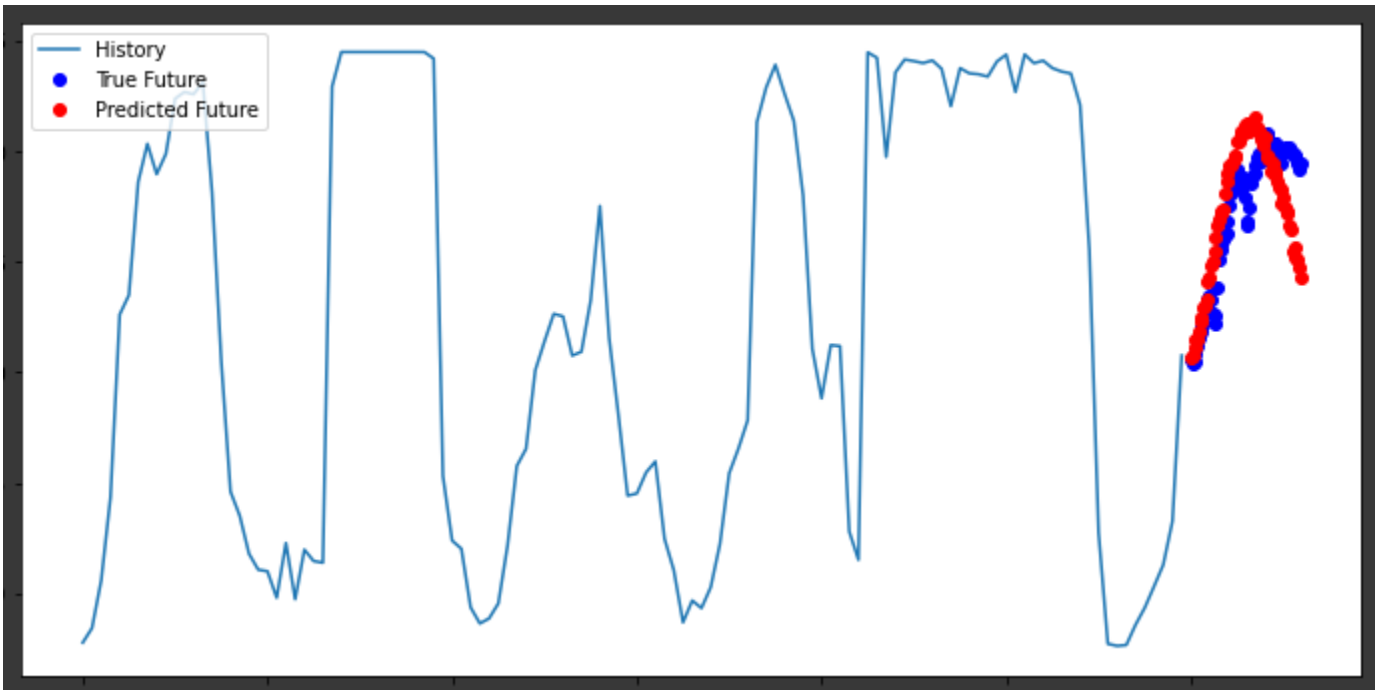
```
plt.ylim(0)
plt.xticks(rotation = 90)
plt.legend()
plt.title("Wind Direction vs Power")
plt.show()
```



A training run of 250 epochs was performed on the dataset, after which the training loss stagnated at 0.267 and the validation loss at 0.260.

```
Epoch 250/250
200/200 [==============================] - 36s 179ms/step - loss: 0.2667 - val_loss: 0.2604
```

Our model's predictions on the validation set were calculated to have an accuracy of about 75 per cent.



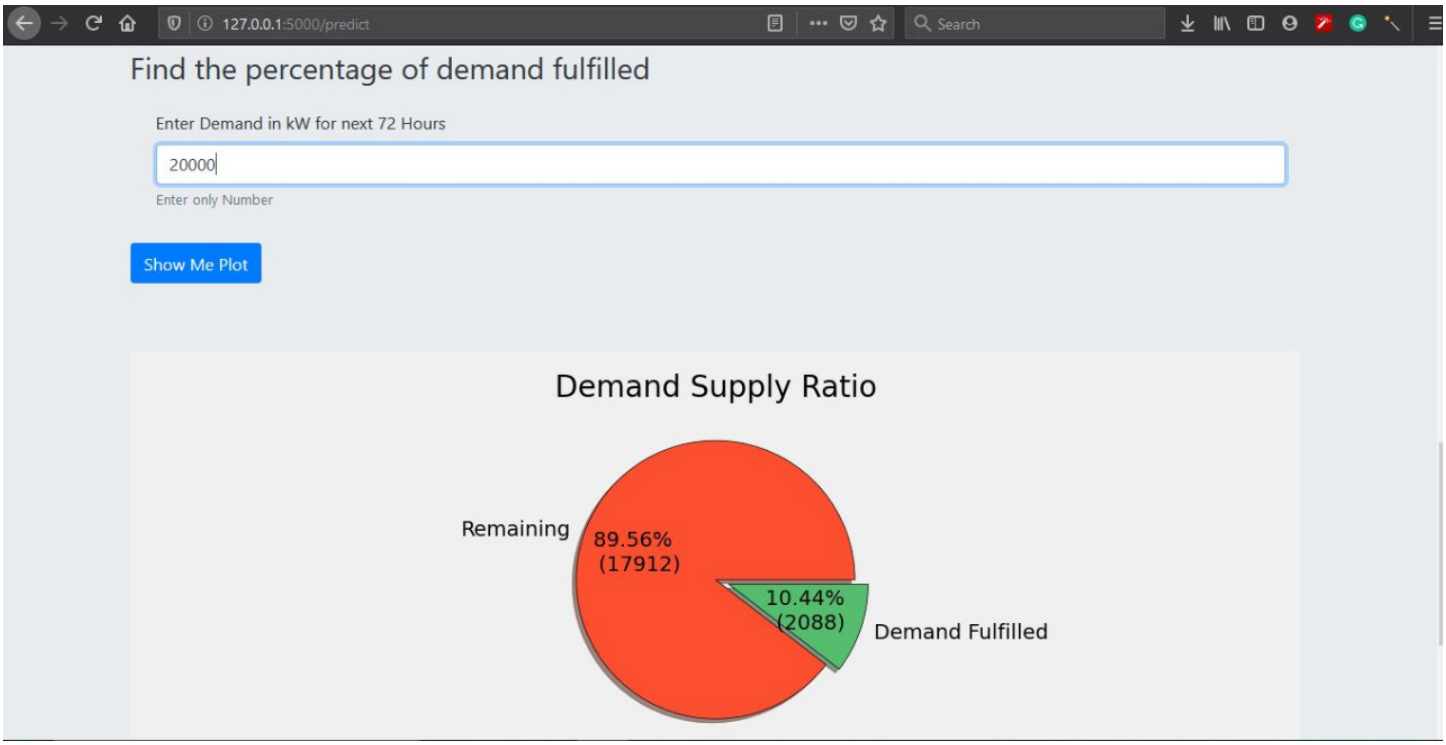
Model prediction on the validation set

5. RESULT

We have integrated our ML model with our web application for data visualisation, easier accessibility to increased feasibility to enable our clients to make informed and pragmatic decisions based on the predicted power output values.

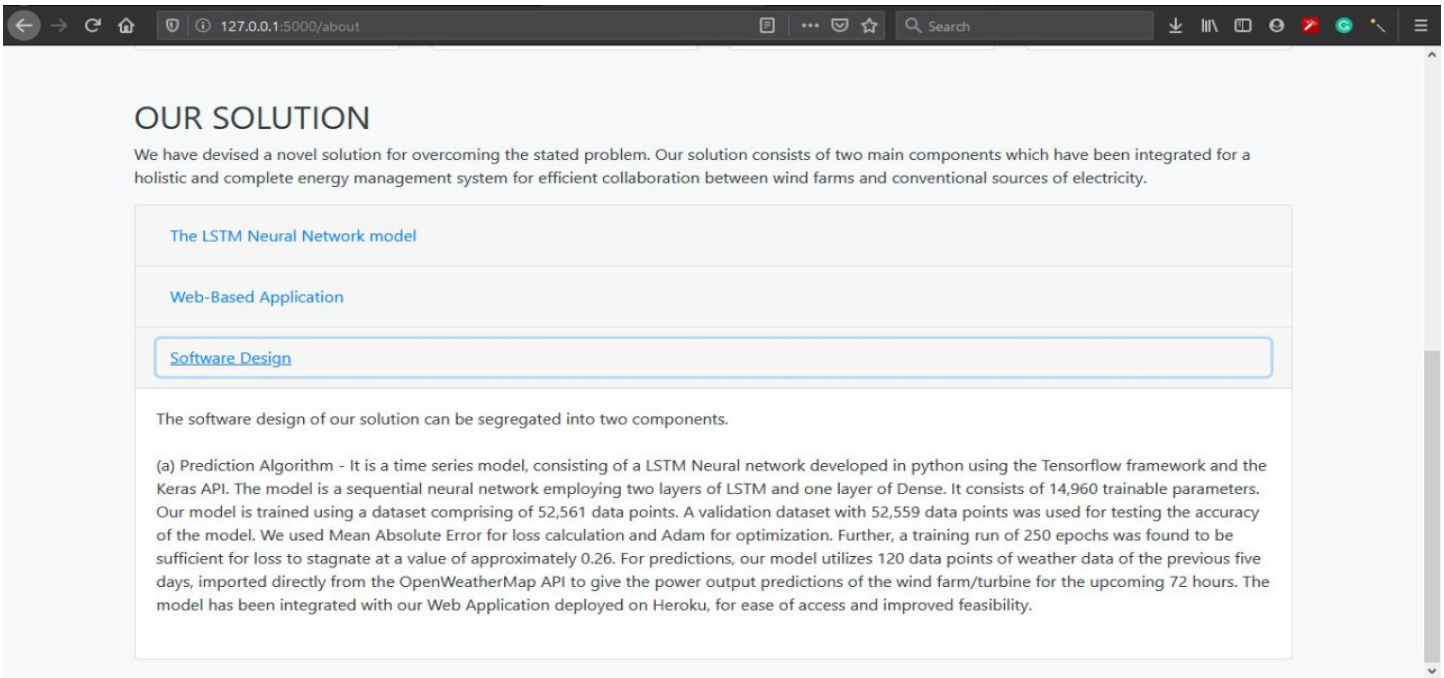
Our web app has a convenient and user-friendly interface that displays statistical representations of various parameters which have an impact on the power generated by the wind farm.

The home page of the application will have a feature, using which the client can enter the amount of electricity required by the city and can get a graphic representation in the form of a pie-chart of the supply which can be fulfilled by the wind farm with respect to the total demand. This will ensure efficient collaboration between various sources of energy, thus preventing overproduction.



Pie - chart displaying the demand - supply ratio on the home page

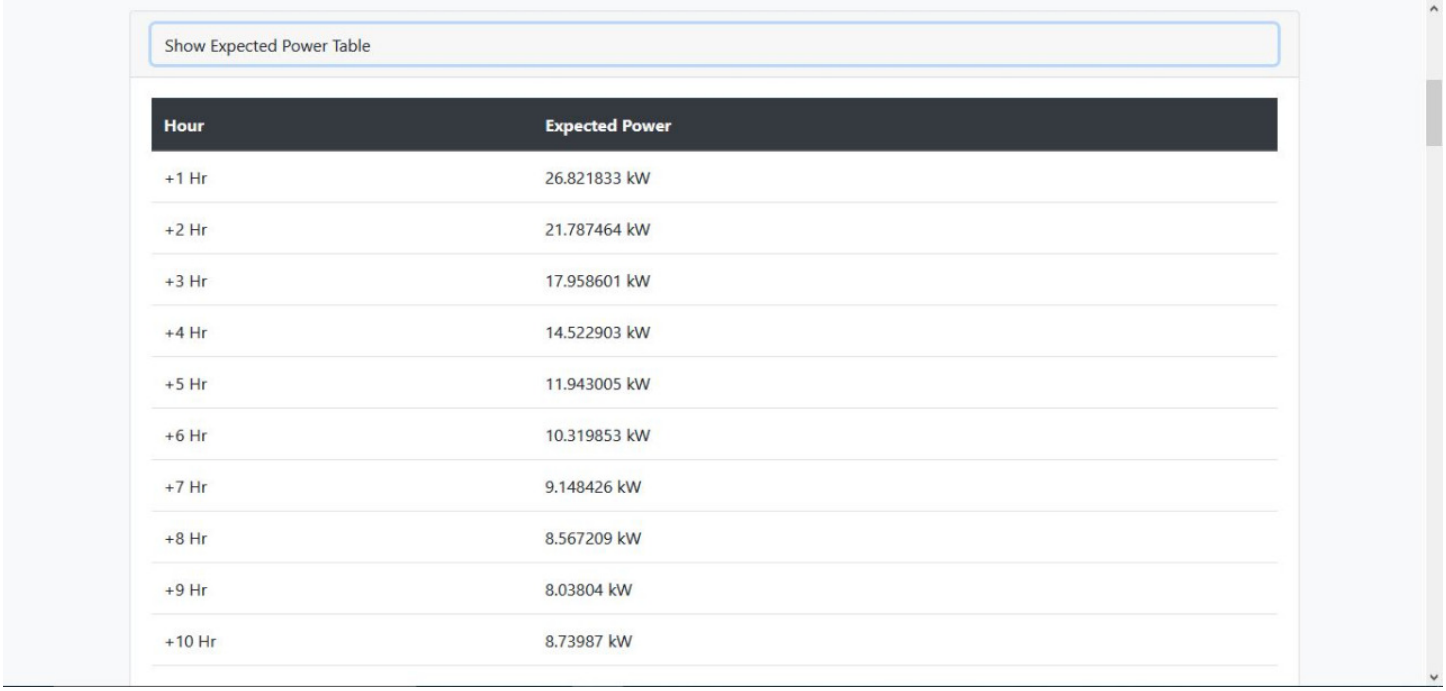
The about page will display, the team behind the project and some valuable information regarding the IBM Hack challenge, Wind energy and the details about the implementation of our solution.jf



About page

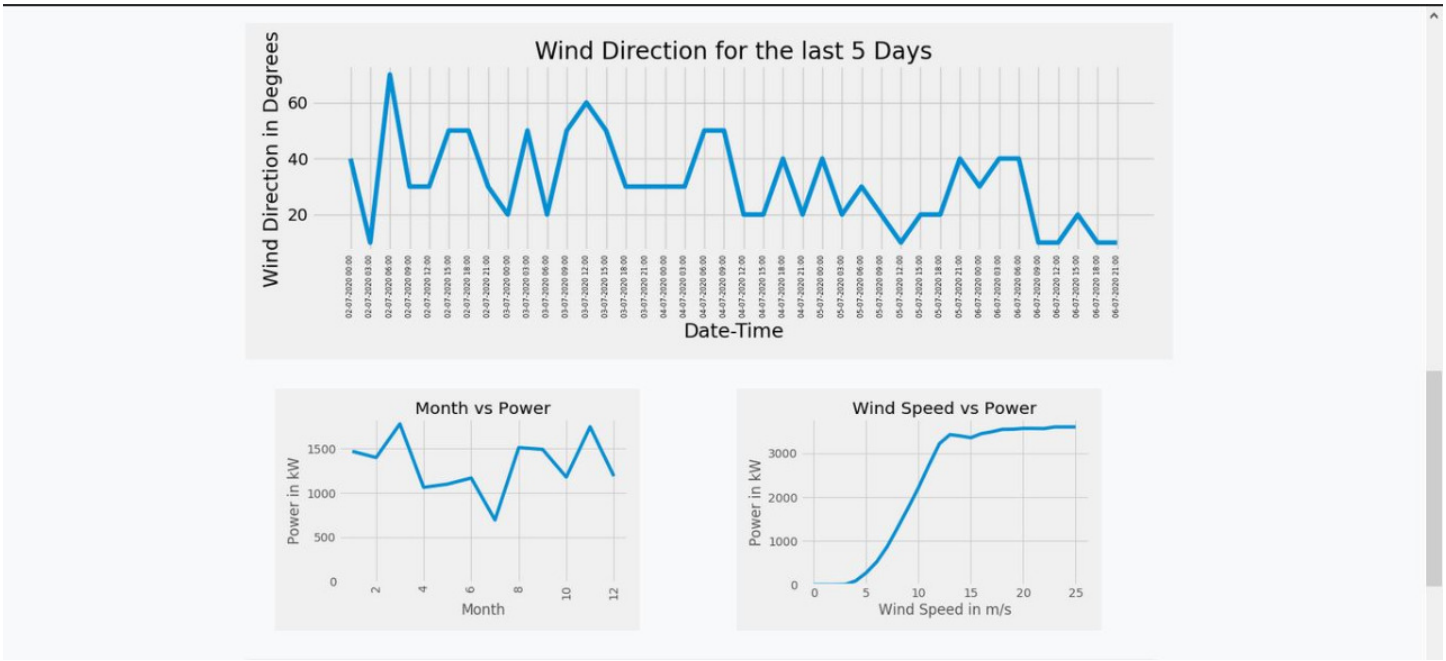
The statistics page will display graphical representations of a wide variety of parameters which have been recorded for the past five days and imported for the OpenWeatherMap API. This page will also

show the predicted power for every hour in the future for up to 72 hours. In addition to this, the total energy produced by the farm will be displayed, knowing which the user can determine how much excess power will need to be fulfilled by traditional power sources. Some screenshots from the stats webpage are shown below.

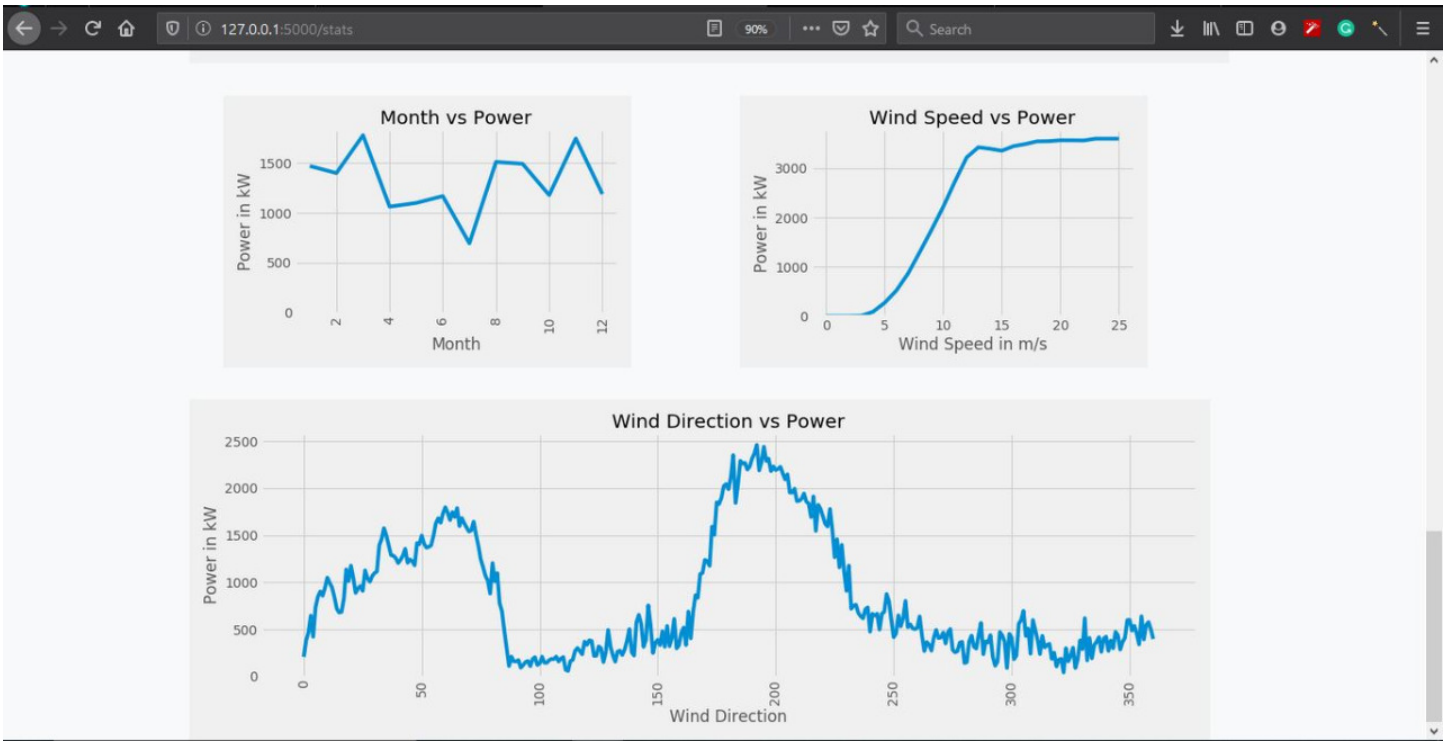


Hour	Expected Power
+1 Hr	26.821833 kW
+2 Hr	21.787464 kW
+3 Hr	17.958601 kW
+4 Hr	14.522903 kW
+5 Hr	11.943005 kW
+6 Hr	10.319853 kW
+7 Hr	9.148426 kW
+8 Hr	8.567209 kW
+9 Hr	8.03804 kW
+10 Hr	8.73987 kW

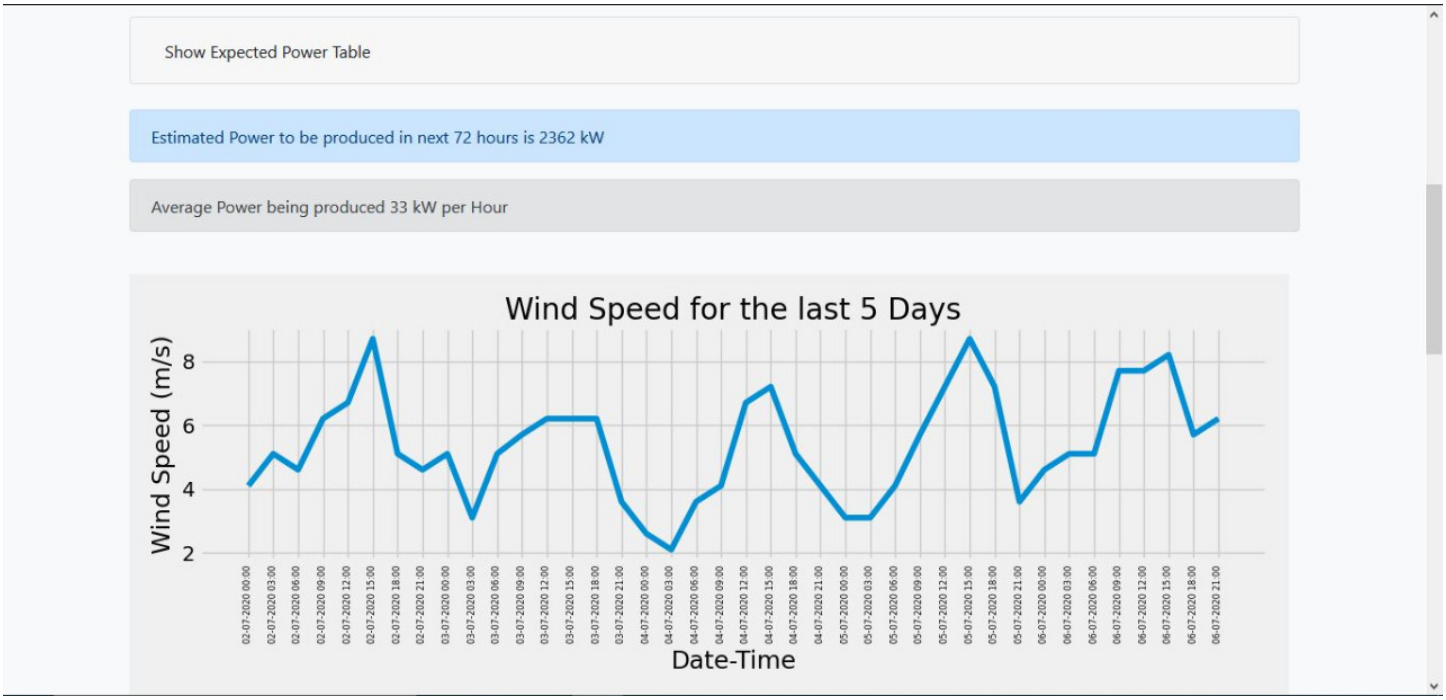
Stats page (1)



Stats page (2)



Stats page (3)



Stats page (4)

6. CONCLUSION

Our solution to the IBM Hack challenge 2020, tackles the real-world problem of overproduction of electricity to inefficient collaboration between renewable and conventional sources of energy. The approach used will significantly benefit our clients and ensure monetary and financial savings by demoting overproduction and promoting informed decision making.

6.1 Advantages and disadvantages

Advantages of our solution include high accuracy of the model predictions due to the use of state-of-the-art Machine Learning algorithms such as Long Short Term Memory (LSTM) which is very useful for time-series forecasting.

Also, we have developed a unique, user-friendly and interactive web application which enables the client to get visual updates of the predicted output values, past weather conditions at the wind farm site, correlations between various parameters and informative plots of demand versus supply requirements of the recipient city or colony.

Disadvantages include non-availability of high processing power hardware due to which we could only implement two layers of LSTM to save up on time and to enable a higher number of iterations.

Further, we could only find datasets with data points for a maximum of one year with the required variables (Wind Speed, Wind Direction and Power Output). So another disadvantage was the unavailability of large amounts of training data.

Another disadvantage is that the API only provides wind direction and speed as data. If previous output power and other variables were provided as well, we could have further increased accuracy by having more input features for predictions.

6.2 Applications

There is a broad scope of implementation areas of our solution. Apart from its application in the wind energy industry, our approach can also be applied for other renewable power sources such as Hydropower and solar energy.

By changing the input features and variables, we can accurately determine the power output for various other sources of energy and therefore, can further improve savings by decreasing overproduction. This will significantly benefit the environment, as well.

Moreover, after a few years, with a more extensive database, the accuracy of our model would be much better. This would allow us to know exactly how much demand is getting fulfilled by renewable sources. Knowing this, we can completely get rid of conventional sources of energy, thereby significantly reducing the carbon footprint of power sources.

6.3 Future scope

If deployed at an industrial scale, with an improvement in hardware, the accuracy of the model can be vastly improved by extensive experimentation including but not restricted to, adding more LSTM layers, changing the number of nodes per layer, increasing the number of epochs, trying a variety of optimisers and loss functions to reduce the loss.

Furthermore, with the use of an API and a dataset providing comprehensive and a wide variety of features, the input to the neural network can be made more diverse, which will ultimately improve the robustness of the model.

Also, by utilising weather, solar and hydro database from multiple cities, it is possible to create a multi-city system, which can monitor and predict the total power output from all these renewable sources of energy.

We have set the base of such a system by creating a login and a dummy registration page. Using this feature, multiple clients and users from various parts of the world can register themselves by putting in their details such as their location and ZIP code. Using this information, our application can automatically import the required weather data if it is available in the API's database and therefore can predict the Windpower output for wind farm situated in that region.

Login page of our web app

Dummy registration page

Such technology and system will disrupt the current global scenario of energy production and will enable humanity to take a vital step towards a better, green future.

7. BIBLIOGRAPHY

[1] https://gwec.net/
[2] https://wwindea.org/
[3] Predicting the Energy Output of Wind Farms Based on Weather Data: Important Variables and their Correlation - Katya Vladislavleva et al.
[4] https://www.tensorflow.org/tutorials/structured_data/time_series

8. APPENDIX

[1] Web application - https://mighty-mit.herokuapp.com/
[2] Machine Learning model code - https://colab.research.google.com/drive/1vooQLa9hsQrPCjhB1PxSItGYnrBXdQC5?usp=sharing
[3] Github repository - https://github.com/SmartPracticeschool/SBSPS-Challenge-1322-Predicting-the-energy-output-of-wind-turbine-based-on-weather-condition
[4] Video presentation - https://drive.google.com/file/d/1kY-niv9kU3Jquc3cabrvWrytKa1_y7fi/view
[5] PPT - https://drive.google.com/file/d/1yd6g79vODjzk3qt0cpVAwvl3IoW8qM7u/view?usp=sharing

[6] Source Code -

(a) Machine Learning Model code

```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import pandas as pd
import tensorflow as tf
import os
import torch

mpl.rcParams['figure.figsize'] = (8, 6)
mpl.rcParams['axes.grid'] = False

from google.colab import files
uploaded = files.upload()

dataset = pd.read_csv("dataset 3.csv", encoding= 'unicode_escape')
TRAIN_SPLIT = 52561

print (dataset)

dataset.info

df = dataset
features_considered = ['Wind Speed (m/s)', 'LV ActivePower (kW)',
 'Wind Direction (°)']
features = df[features_considered]
features.index = df['Date/Time']
features.head()
features.plot(subplots=True)

dataset_mean = df.mean(axis = 0)
dataset_std = df.std(axis = 0)
print(dataset_mean[1])

df3 = df
df2 = df

```

```
features_considered = ['Wind Speed (m/s)', 'Wind Direction (°)']
features = df3[features_considered]
features.index = df3['Date/Time']
features.tail()

dataset = features.values

data_mean = dataset[:TRAIN_SPLIT].mean(axis=0)
data_std = dataset[:TRAIN_SPLIT].std(axis=0)
dataset = (dataset-data_mean)/data_std

features_considered_labels = ['Wind Speed (m/s)', 'LV ActivePower (kW)', 'Wind Direction (°)']
features_mod = df2[features_considered_labels]
features_mod.index = df2['Date/Time']
features_mod.tail()

dataset_mod = features_mod.values

data_mean_mod = dataset_mod[:TRAIN_SPLIT].mean(axis=0)
data_std_mod = dataset_mod[:TRAIN_SPLIT].std(axis=0)
dataset_mod = (dataset_mod-data_mean_mod)/data_std_mod

# For extracting one data point per hour(Taking steps of 6, as data recorded every 10 min)
def multivariate_data(dataset, target, start_index, end_index, history_size,
                      target_size, step):
  data = []
  labels = []

  start_index = start_index + history_size
  if end_index is None:
    end_index = len(dataset) - target_size

  for i in range(start_index, end_index):
    indices = range(i-history_size, i, step)
    data.append(dataset[indices])
    labels.append(target[i:i+target_size])

```

```
  return np.array(data), np.array(labels)

past_history = 720
future_target = 432
STEP = 6
BUFFER_SIZE = 10000
BATCH_SIZE = 256

x_train_multi, y_train_multi = multivariate_data(dataset,
dataset_mod[:, 1], 0,
TRAIN_SPLIT, past_history,
                                                 future_target, STEP)

x_val_multi, y_val_multi = multivariate_data(dataset, dataset_mod[:,
1],
                                             TRAIN_SPLIT, None,
past_history,
                                             future_target, STEP)

print ('Single window of past history :
{}'.format(x_train_multi[0].shape))
print ('\n Target power to predict :
{}'.format(y_train_multi[0].shape))

# Conversion into tensors and splitting of data into mini-batches
train_data_multi = tf.data.Dataset.from_tensor_slices((x_train_multi,
y_train_multi))
train_data_multi =
train_data_multi.cache().shuffle(BUFFER_SIZE).batch(BATCH_SIZE).repeat
()

val_data_multi = tf.data.Dataset.from_tensor_slices((x_val_multi,
y_val_multi))
val_data_multi = val_data_multi.batch(BATCH_SIZE).repeat()

def create_time_steps(length):
  return list(range(-length, 0))

def multi_step_plot(history, true_future, prediction):
  plt.figure(figsize=(12, 6))
```

```
  num_in = create_time_steps(len(history))
  num_out = len(true_future)

  plt.plot(num_in, np.array(history[:, 1]), label='History')
  plt.plot(np.arange(num_out)/STEP, np.array(true_future), 'bo',
           label='True Future')
  if prediction.any():
    plt.plot(np.arange(num_out)/STEP, np.array(prediction), 'ro',
             label='Predicted Future')
  plt.legend(loc='upper left')
  plt.show()

for x, y in train_data_multi.take(1):
  multi_step_plot(x[0], y[0], np.array([0]))

# Defining the model
multi_step_model = tf.keras.models.Sequential()
multi_step_model.add(tf.keras.layers.LSTM(32,
                                          return_sequences=True,

  input_shape=x_train_multi.shape[-2:]))
multi_step_model.add(tf.keras.layers.LSTM(16, activation='relu'))
multi_step_model.add(tf.keras.layers.Dense(432))

multi_step_model.compile(optimizer='adam', loss='mae')

multi_step_model.summary()

for x, y in val_data_multi.take(1):
  print (multi_step_model.predict(x).shape)

EVALUATION_INTERVAL = 200
EPOCHS = 250

multi_step_history =
  multi_step_model.fit(train_data_multi,epochs=EPOCHS,steps_per_epoch=EV
  ALUATION_INTERVAL,validation_data=val_data_multi,validation_steps=50)

def plot_train_history(history, title):
  loss = history.history['loss']
```

```
  val_loss = history.history['val_loss']

  epochs = range(len(loss))

  plt.figure()

  plt.plot(epochs, loss, 'b', label='Training loss')
  plt.plot(epochs, val_loss, 'r', label='Validation loss')
  plt.title(title)
  plt.legend()

  plt.show()
plot_train_history(multi_step_history, 'Multi-Step Training and validation loss')
 def multivariate_data_predict(dataset, past_window):
  data = []
  end_index = len(dataset)
  start_index = len(dataset) - past_window

  for i in range(start_index, end_index):
    data.append(dataset[i])

  return np.array(data)

uploaded_predict = files.upload()

# Importing the previous 5 days weather data
predict_dataset = pd.read_csv("dataset_predict.csv", encoding='unicode_escape')
print(predict_dataset)

# Normalising the data w.r.t the training dataset
df4 = predict_dataset
predict_features_considered = ['Wind Speed (m/s)', 'Wind Direction (°)']
predict_features = df4[predict_features_considered]
predict_features.index = df4['Date/Time']
predict_features.tail()

predict_dataset = predict_features.values
```

```
175
176 predict_dataset_mod = (predict_dataset-data_mean)/data_std
177
178 BATCH_SIZE_predict = 120
179 past_history_predict = 120
180
181 x_predict_multi = multivariate_data_predict(predict_dataset_mod,
    past_history_predict)
182
183 x_predict_multi =
  tf.data.Dataset.from_tensor_slices((x_predict_multi))
184 x_predict_multi = x_predict_multi.batch(BATCH_SIZE_predict).repeat()
185 x_predict_multi = x_predict_multi.batch(BATCH_SIZE_predict).repeat()
186
187 # Output predictions for the next 3 days(6 predictions per hour * 72
    hours = 432 data points)
188 predictions_raw = multi_step_model.predict(x_predict_multi.take(1))
189 predictions_array = (predictions_raw*dataset_std[1]) +
  dataset_mean[1]
190 predictions = predictions_array[0]
191 for i in range(120):
192   if predictions[i] < 0:
193     predictions[i] = 0
194 print(predictions)
195
196 # Function for plotting the predicted values
197 def output_plot(output):
198   plt.plot(output)
199   plt.xlabel("Predictions made every 10 min for the next 72 hours")
200   plt.ylabel("Active Power Output (kW)")
201   plt.show()
202
203 output_plot(predictions)
204
205 from google.colab import drive
206 drive.mount('/content/gdrive')
207
208 #using model.save_weights() to save the weights of the model in HDF5
  format
209 multi_step_model.save_weights("/content/gdrive/My
```

```
Drive/weights_final.h5")
multi_step_model.save("/content/gdrive/My Drive/model_final.h5")
```

(b) Web Application code

```
def real_plot():
    data = pd.read_csv("static/historical.csv")

    model = tf.keras.models.load_model('static/model5.h5')

    pred_data = data[-120:]
    pred_data.index = pred_data["date_time"]
    pred_data = pred_data[["wind_speed","wind_direction"]]
    pred_data = pred_data.values
    pred_data_std = [3.64500726,94.60264878]
    pred_data_mean = [8.20507312,163.49269538]
    dataset_mean = 7384
    dataset_std = 5950
    x_predict_multi = (pred_data-pred_data_mean)/pred_data_std

    x_predict_multi=
tf.data.Dataset.from_tensor_slices((x_predict_multi))

    x_predict_multi=
x_predict_multi.batch(BATCH_SIZE_predict).repeat()
    x_predict_multi=
x_predict_multi.batch(BATCH_SIZE_predict).repeat()
    predictions_raw = model.predict(x_predict_multi.take(1))
    predictions_array = (predictions_raw*dataset_std) + dataset_mean
    predictions = predictions_array[0]

    return predictions

def update_data():
  one_day = 86400

  API_KEY = environ.get("WEATHER_API_KEY")
  CITY = "Istanbul"
```

```
31   date_time = []
32   wind_speed = []
33   wind_dir = []
34
35   for i in range(5,0,-1):
36       dt = datetime.utcnow().date()
37       ts =int(time.mktime(dt.timetuple()))
38       url =
   f"http://api.openweathermap.org/data/2.5/onecall/timemachine?lat=41.0
   082&lon=28.9784&dt={ts-((i-1)*one_day)}&appid={API_KEY}"
39       print(url)
40       resp = requests.get(url)
41       resp = resp.json()
42       #print(resp['hourly'])
43       for j in resp['hourly']:
44           date_time.append(j['dt'])
45           wind_speed.append(j["wind_speed"])
46           wind_dir.append(j["wind_deg"])
47   date_time = [datetime.utcfromtimestamp(x).strftime('%d-%m-%Y
   %H:%M') for x in date_time]
48   data =
   pd.DataFrame({"date_time":date_time,"wind_speed":wind_speed,"wind_dir
   ection":wind_dir})
49
   data.to_csv("static/historical.csv",header=False,mode='a',index=False
   )
50   print(data.shape)
51   data = pd.read_csv("static/historical.csv")
52   data.drop_duplicates(inplace = True)
53   data.to_csv("static/historical.csv",index = False)
```