Predicting the energy output of wind turbine based on weather condition

Team ID: PNT2022TMID40686

Bachelor of Engineering

Electronics and Communication Engineering

Jayalakshmi institute of technology-636352

Industrial Mentor : Nidhi Faculty Mentor :Jagatheesan

Team Members :-

V.ARTHI

G. SivaKumar

K. Gayathri

M. Boopathy

1 : INTRODUCTION

1.1: Project Overview

Predict the output power of a Wind Turbine at any given time provided with Weather Conditions. Using Machine Learning that takes on previous performance data and real time weather parameters to predict the energy output will help in integrating with the grid and make use of its full potential. The wind speed and wind direction can be given as input and the model will predict the output power of the turbine. Different machine learning models have been evaluated to determine the best fitting model.

1.2 : Purpose

Due to the unpredictable nature of Wind speed and direction (weather condition). Because of this the power generated by a wind mill is irregular and unpredictable. The power generated depends on a large number of variables like, season, temperature, yearly currents, humidity, pressure, location, altitude, height off the turbine, blade size, blade pitch and many more. Owing to the irregular nature of the output power it is very difficult to integrate this source of renewable energy with the grid. In consequence Wind Farms loss revenue unable to supply the power at the right time to the grid.

2 : LITERATURE SURVEY

2.1 : Existing Problem

The Existing solution for has been studied. It has been that inferred that, the output energy of the wind mill is being predicted approximately manually or using some basic software. The result obtained is often unreliable and the margin of error is very high owing to the multiple variables affecting the output. There are several thousand Wind Mills across the globe. The output performance varies significantly even between two wind mills that are located very closely to each other. A generalized at the same time customizable to meet the requirement of a single wind mill is required.

Literature survey:

SI.NO	TITLE	ABSTRACT	MERITS	DEMERITS
1.	Predicting The Energy Output Of Wind Farms Based On Weather Data: Important Variables And Their Correlation	The energy output of the wind farm is highly depend on the weather conditions present at the wind fram.	Wind energy output can be predicted from publicly available weather data with accuracy at best 80%	Default settings to run the symbolic regression experiments as well as variable importance.
2.	Wind power forecasting based on time series model using deep learning algorithms.	Wind energy is created due to uneven heating of the earth surface and coriolis acceleration	To minimize risk and to improve performance.	Concerning to predict difficult operation problems.
3.	Using machine learning to predict wind turbine power output	In this work, new aerostructural simulations of a generic 1.5 MW turbine are used to rank atmospheric influences on power output.	Simulations of a utility-scale wind turbine have been used to develop a database	Application of the data to wind turbine deployment sites does not require any new instrumentation compared to what is currently used.

Reference link:

https://hpi.de/friedrich/docs/paper/RE1.pdf

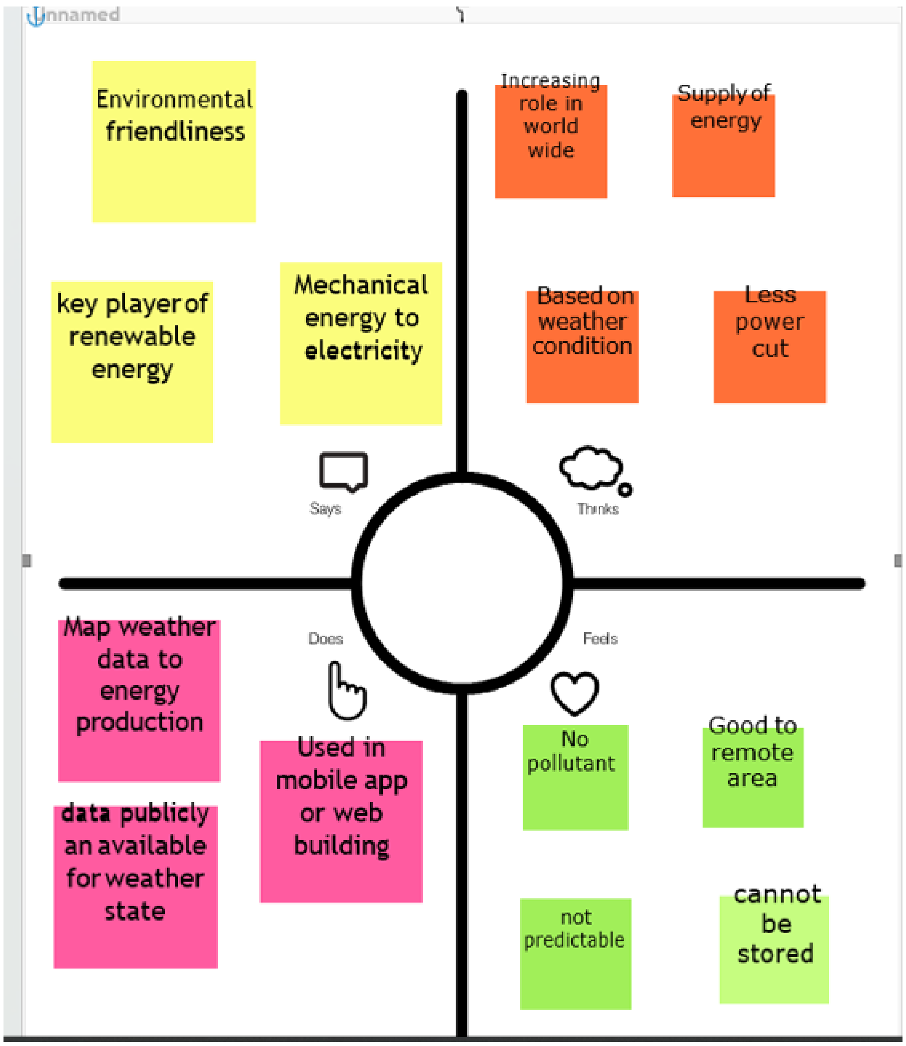
https://iopscience.iop.org/article/10.1088/1748-9326/8/2/024009/pdf

2.3 : Problem Statement Definition

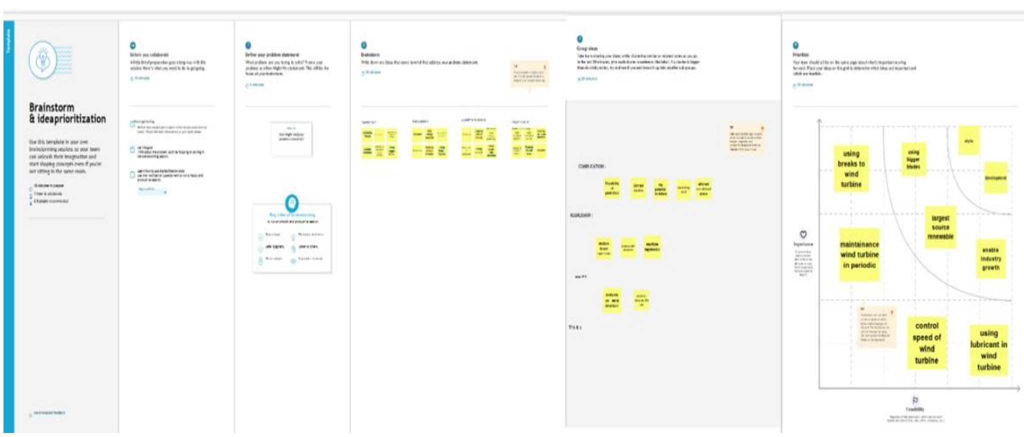
Predict the output power of a Wind Turbine at any given time provided with Weather Conditions. Using Machine Learning that takes on previous performance data and real time weather parameters to predict the energy output will help in integrating with the grid and make use of its full potential. Accurate wind power forecasting reduces the need for additional balancing energy and reserve power to integrate wind power. To make use wind energy efficiently the accurate power output is required. When power output of a wind mill at a given time is known we can integrate it with grid and make use of this renewable source of energy rather than conventional non-renewable sources.

3 : IDEATION AND PROPOSED SOLUTION

3.1 : Empathy Map Canvas



3.2 : Ideation And Brainstorming

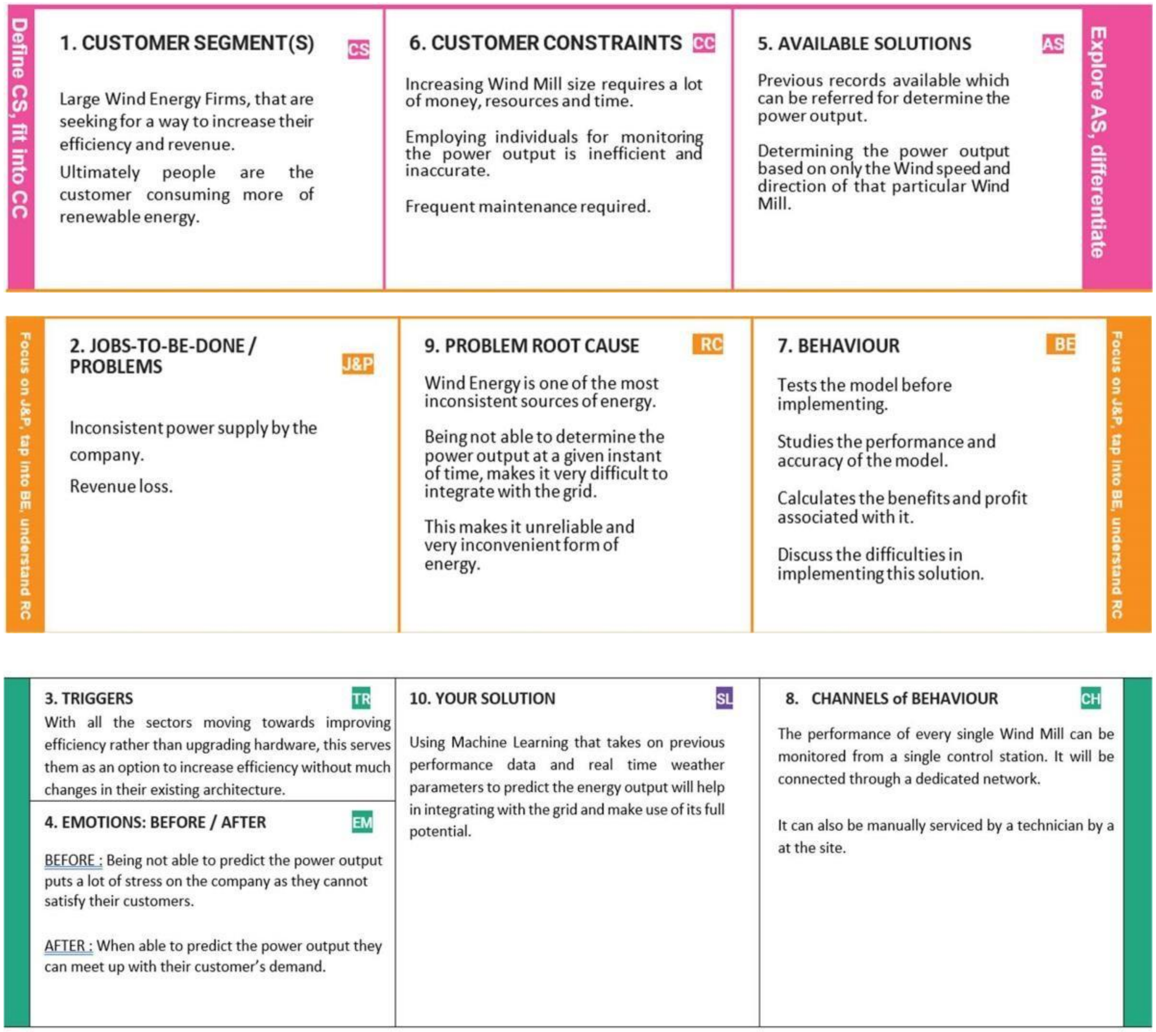


3.3 : Proposed Solution

S.No.	Parameter	Description
1.	Problem Statement (Problem to be solved)	Wind Energy is the widely used Renewable energy source, but it is not a sustained source. The power generated is affected by various environmental factors. Thus it cannot be relied upon completely, thereby reducing its efficiency.

2.	Idea / Solution description	Using Machine Learning that takes on previous performance data and real time weather parameters to predict the energy output will help in integrating with the grid and make use of its full potential.
3.	Novelty / Uniqueness	This model takes in the previous years energy outputs and corelate it with the weather and other parameters that affected it. By using this model we can give the Weather conditions as input and obtain the energy output. It also dynamically alters the algorithm based on the predicted value and actual output value.
4.	Social Impact / Customer Satisfaction	This model helps in increasing the usage of renewable energy. It optimizes the operation of Wind Turbines. The cost of Implementing this solution makes it an Unformidable one.
5.	Business Model (Revenue Model)	Wind Energy Companies will be able to increase their energy output thereby increasing revenue. Wind Energy can be trusted as a consistent source as we are able to predict the total power output for any given time.
6.	Scalability of the Solution	This doesn’t require any additional equipment to be set up at the Wind turbine. The existing Sensors can be used to get the Weather parameters for predicting the power output. With Weather stations all across the world, the data can be obtained easily in real time. The prediction can be carried out at the control station of the Wind mills. The algorithm can be easily modified to work for every single Wind Turbine to get accurate results.

3.4 : Problem Solution Fit



Define CS, fit into CC

1. CUSTOMER SEGMENT(S) CS

Large Wind Energy Firms, that are seeking for a way to increase their efficiency and revenue.

Ultimately people are the customer consuming more of renewable energy.

6. CUSTOMER CONSTRAINTS CC

Increasing Wind Mill size requires a lot of money, resources and time.

Employing individuals for monitoring the power output is inefficient and inaccurate.

Frequent maintenance required.

5. AVAILABLE SOLUTIONS AS

Previous records available which can be referred for determine the power output.

Determining the power output based on only the Wind speed and direction of that particular Wind Mill.

Explore AS, differentiate

Focus on J&P, tap into BE, understand RC

2. JOBS-TO-BE-DONE / PROBLEMS J&P

Inconsistent power supply by the company.

Revenue loss.

9. PROBLEM ROOT CAUSE RC

Wind Energy is one of the most inconsistent sources of energy.

Being not able to determine the power output at a given instant of time, makes it very difficult to integrate with the grid.

This makes it unreliable and very inconvenient form of energy.

7. BEHAVIOUR BE

Tests the model before implementing.

Studies the performance and accuracy of the model.

Calculates the benefits and profit associated with it.

Discuss the difficulties in implementing this solution.

Focus on J&P, tap into BE, understand RC

3. TRIGGERS TR

With all the sectors moving towards improving efficiency rather than upgrading hardware, this serves them as an option to increase efficiency without much changes in their existing architecture.

4. EMOTIONS: BEFORE / AFTER EM

BEFORE : Being not able to predict the power output puts a lot of stress on the company as they cannot satisfy their customers.

AFTER : When able to predict the power output they can meet up with their customer's demand.

10. YOUR SOLUTION SL

Using Machine Learning that takes on previous performance data and real time weather parameters to predict the energy output will help in integrating with the grid and make use of its full potential.

8. CHANNELS of BEHAVIOUR CH

The performance of every single Wind Mill can be monitored from a single control station. It will be connected through a dedicated network.

It can also be manually serviced by a technician by a at the site.

4 : REQUIREMENT ANALYSIS

4.1 : Functional Requirements

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User Registration	Registration through Form Registration through Gmail Registration through LinkedIN
FR-2	User Confirmation	Confirmation via meeting Confirmation via mail

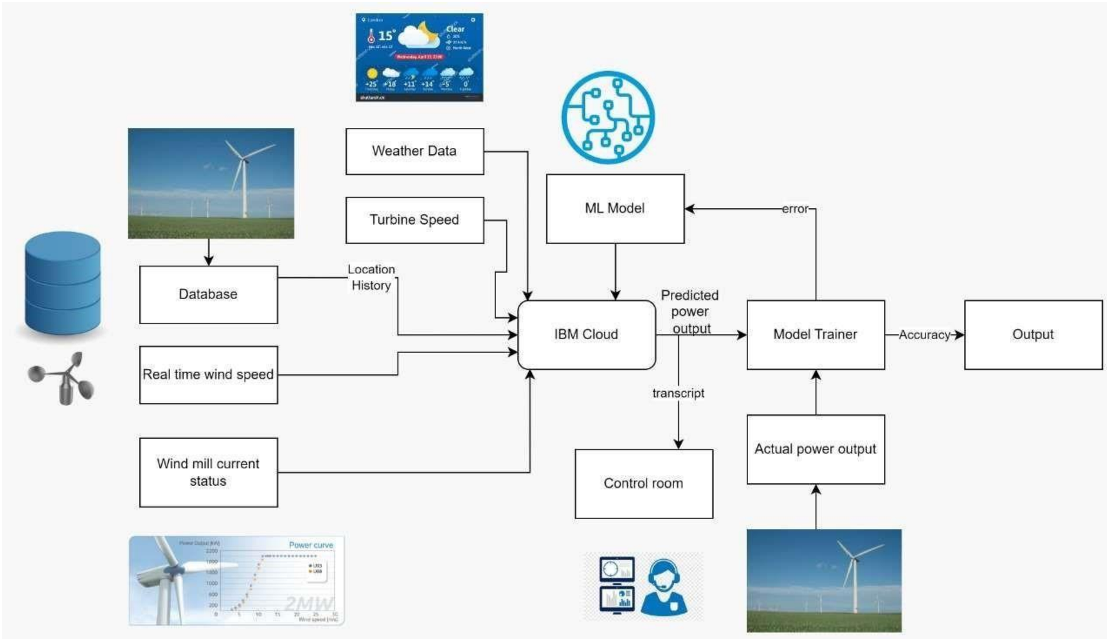
FR-3	User Requirements	Knowledge about inputting the data Teaching on using the ML Model
FR-4	User Infrastructure	A system to support ML data modelling A Suitable GPU and CPU
FR-5	User Network	Network infrastructure to connect the wind mill to the Control station
FR-6	User Cost	User has to spend only for attaining the software, additional components are not required

4.2 : Non – Functional Requirements

FR No.	Non-Functional Requirement	Description
NFR-1	Usability	It can be used for various Wind mill and trained for specific models.
NFR-2	Security	The data can be stored in a secure cloud.
NFR-3	Reliability	The predicted will be accurate and can be relied upon.
NFR-4	Performance	The performance of the model depends on the Computer it runs on the accuracy of the data it is fed with.
NFR-5	Availability	It is available as a software package.
NFR-6	Scalability	It can be scaled up and interconnected with other Wind Mills to create a connected Wind Form.

5 : PROJECT DESIGN

5.1 : Data Flow Diagrams



1. The Weather data and wind speed data is fed as input to the ML model.
2. The previous years data stored in a cloud is fed as input to the ML model.
3. The expected power production is predicted.
4. The predicted value is sent to the user, and also stored in cloud.
5. The predicted output power is compared with the actual power generated.
6. The error in prediction is calculated and used as a feedback to train the model.
7. The model accuracy increases over the course.

5.2 : Solution and Technical Architecture

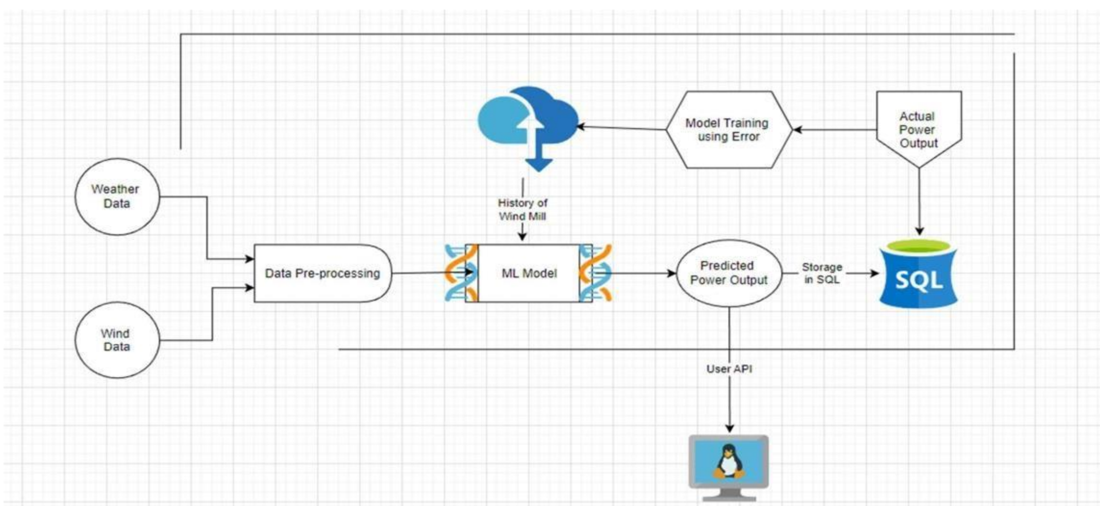


Table-1 : Components & Technologies:

S.No	Component	Description	Technology

1.	User Interface	API	HTML, CSS, JavaScript / Angular Js / React Js etc.
2.	Application Logic-1	Data Pre-processing	Java / Python
3.	Application Logic-2	Data Input	IBM Watson STT service
4.	Database	Previous Year data	MySQL, NoSQL, etc.
5.	Cloud Database	Database Service on Cloud	IBM DB2, IBM Cloudant etc.
6.	File Storage	File storage requirements	IBM Block Storage or Other Storage Service or Local Filesystem
7.	External API	Purpose of External API used in the application	IBM Weather API, etc.
8.	Machine Learning Model	Purpose of Machine Learning Model	Weather prediction Model, etc.
9.	Infrastructure (Server / Cloud)	Application Deployment on Local System / Cloud Local Server Configuration: Cloud Server Configuration:	Local, Cloud Foundry, Kubernetes, etc.

Table-2: Application Characteristics:

S.No	Characteristics	Description	Technology
1.	Open-Source Frameworks	List the open-source frameworks used	FLASK
2.	Security Implementations	List all the security / access controls implemented, use of firewalls etc.	SHA-256, Encryptions, IAM Controls, OWASP etc.
3.	Scalable Architecture	Justify the scalability of architecture (3 – tier, Microservices)	Cloud

4.	Availability	Justify the availability of application (e.g. use of load balancers, distributed servers etc.)	Distributed cloud service
5.	Performance	Design consideration for the performance of the application (number of requests per sec, use of Cache, use of CDN's) etc.	SDN

5.3 : User Stories

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priorit y	Release
Customer (Mobile user)	Registration	USN-1	As a user I can buy the ML model and train it customize according to the needs.	I can access my account / dashboard	High	Sprint- 1
		USN-2	My Identity can be verified through a mail.	I can receive confirmation email & click confirm	High	Sprint- 1
	Login	USN-1	As a user, I can log into the application by entering email & password.	I can login into the admin window	Low	Sprint- 1

		USN-2	For different Wind Mill, various login can be used.	Different id and password can be used for different Wind Mills.	High	Sprint- 2
	Dashboard	USN-3	The various functionalities can be viewed and navigated from the dashboard.	The options are clear and easily understandable.	Medium	Sprint- 2
User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer Care Executive	Queries	USN-1	The Customer Care executive answered my call and guided me.	He was calm and helped me through the process,	Medium	Sprint- 2
	Initial Setup	USN-2	After the ML model has been bought the sales executive helped me setup the model.	He was clear and knowledgeable.	High	Sprint- 3
Administrator	Remote Access	USN-1	I have remote access to all the models and can be worked on.	The remote access works flawlessly.	Medium	Sprint4

		USN-2	The integration with cloud was easy and simple.	The cloud access is very good.	Low	Sprint4

6 : PROJECT PLANNING AND SCHEDULING

6.1 : Sprint Planning and Estimation

Sprint	Milestone
Sprint 1	1. User Registers into the application through entering Email Id and Password for confirmation. 2. User Receives a confirmation mail for their registered Email. 3. User can also register to the application through Mobile number. 4. User logs in into the website using Email Id and password.
Sprint 2	1. User can access the dashboard. 2. User enters the required details of weather conditions to get the desired turbine power output based on our model's prediction.
Sprint 3	1. Application stores the predictions, that can be used for future analysis. 2. The data stored has to be maintained securely.
Sprint 4	1. Administrator should properly maintain the website and update it whenever required.

6.2 Sprint Delivery Schedule

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	20	6 Days	24 Oct 2022	29 Oct 2022	20	27 Oct 2022

Sprint-2	20	6 Days	31 Oct 2022	05 Nov 2022	20	04 Nov 2022
Sprint-3	20	6 Days	07 Nov 2022	12 Nov 2022	20	09 Nov 2022
Sprint-4	20	6 Days	14 Nov 2022	19 Nov 2022	20	16 Nov 2022

6.3: Reports from JIRA

7 : CODING AND SOLUTIONING

7.1 : Feature 1

Dataset taken for training :

Date/Time	LV ActivePower (kW)	Wind Speed (m/s)	Theoretical_Power_Curve (KWh)	Wind Direction (°)
01 01 2018 00:00	380.0477905	5.31133604	416.3289078	259.9949036
01 01 2018 00:10	453.7691956	5.672166824	519.9175111	268.6411133
01 01 2018 00:20	306.3765869	5.216036797	390.9000158	272.5647888
01 01 2018 00:30	419.6459045	5.659674168	516.127569	271.2580872
01 01 2018 00:40	380.6506958	5.577940941	491.702972	265.6742859
01 01 2018 00:50	402.3919983	5.604052067	499.436385	264.5786133
01 01 2018 01:00	447.6057129	5.793007851	557.3723633	266.1636047
01 01 2018 01:10	387.2421875	5.306049824	414.8981788	257.9494934
01 01 2018 01:20	463.6512146	5.584629059	493.6776521	253.4806976
01 01 2018 01:30	439.725708	5.523228168	475.7067828	258.7237854
01 01 2018 01:40	498.1817017	5.724115849	535.841397	251.8509979
01 01 2018 01:50	526.8162231	5.934198856	603.0140765	265.5046997
01 01 2018 02:00	710.5872803	6.547413826	824.6625136	274.2329102
01 01 2018 02:10	655.1942749	6.199746132	693.4726411	266.7331848
01 01 2018 02:20	754.7625122	6.505383015	808.0981385	266.7604065
01 01 2018 02:30	790.1732788	6.634116173	859.4590208	270.4931946

This is the Excel sheet visualization of the dataset that has been taken for the ML Model. It contains 5 attributes. Date/Time, Active power generated, Theoretical power generated, Wind speed and Wind Direction. The data set has 50,000+ samples. It has data of a single wind mill’s power production over a period of one year. The samples are taken at an interval of every 10 mins making 144 samples per day.

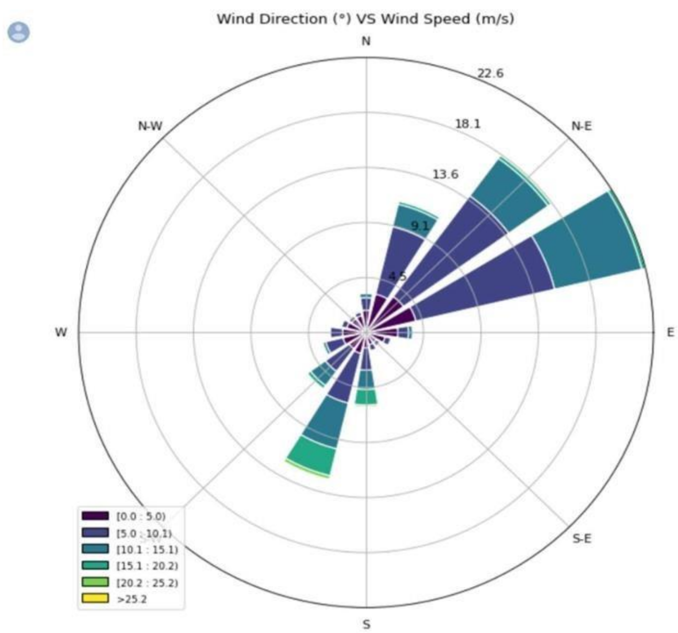
Dataset Description :

```
[ ] data.describe()
```

	ActivePower(kW)	WindSpeed(m/s)	TheoreticalPowerCurve(KWh)	WindDirection
count	50530.000000	50530.000000	50530.000000	50530.000000
mean	1307.684332	7.557952	1492.175463	123.687559
std	1312.459242	4.227166	1368.018238	93.443736
min	-2.471405	0.000000	0.000000	0.000000
25%	50.677890	4.201395	161.328167	49.315437
50%	825.838074	7.104594	1063.776283	73.712978
75%	2482.507568	10.300020	2964.972462	201.696720
max	3618.732910	25.206011	3600.000000	359.997589

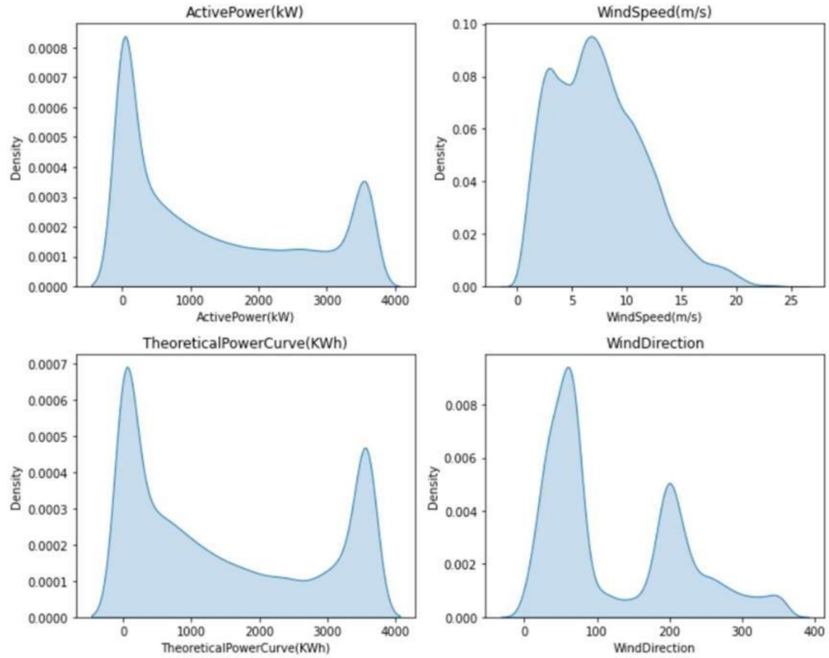
This command displays the various parameters like count, mean, Standard deviation, minimum value, maximum value for the four attributes.

Wind Direction vs Wind Speed :



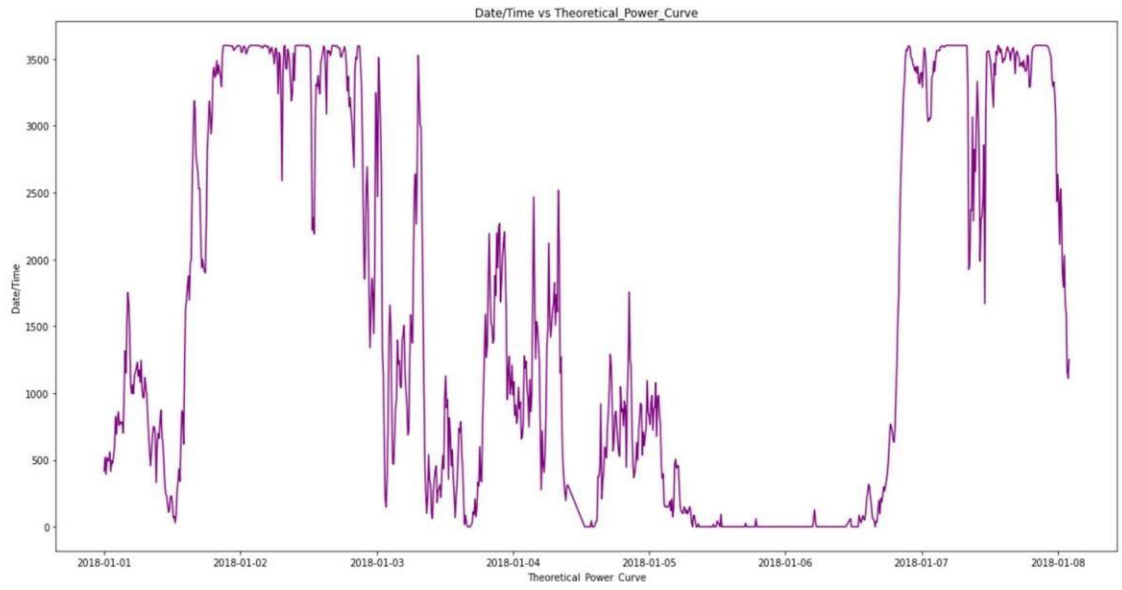
The plot gives a visualization of the direction of the Wind at the location over the year. It also shows the speed of the wind in that particular direction. From this we can infer that this particular Wind Mill experiences wind in a North-easterly direction primarily and southwesterly direction occasionally. The wind speed varies between 5 and 20 m/s.

Feature Inference :



The above graph plots the weightage of each attribute of the dataset. It helps to understand the dataset quickly and easily. The usual Windspeed in the region can be seen as 2 to 12 m/s. The prominent Wind direction is 30° to 75° and mildly along 190° to 210° measures from magnetic north. The actual power generated is also less compared to the theoretical power calculated with the wind speed. This is due to the mechanical and aerodynamic losses faced by the wind mill.

Output Power Visualization :



This is a graph plotted with time as x-axis and power generated in y-axis. 1000 samples (8 days) data has been taken for viewing the plot clearly. It shows the trend in power generation.

On one day there is maximum output and next two days the power output is less owing to low wind speed.

Data Pre-processing:

```
data.shape
```

```
(50530, 5)
```

This command returns the dimension of our dataset. We have 50530 rows and 5 columns which are the features of the dataset.

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50530 entries, 0 to 50529
Data columns (total 5 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   Date/Time                     50530 non-null  object
 1   ActivePower(kW)               50530 non-null  float64
 2   WindSpeed(m/s)                50530 non-null  float64
 3   TheoreticalPowerCurve(KWh)    50530 non-null  float64
 4   WindDirection                 50530 non-null  float64
dtypes: float64(4), object(1)
memory usage: 1.9+ MB
```

This command returns whether our dataset has any null values and the datatype of the features. From the output we can see that there is no null-data type in the dataset and the values are of 64 bit floating point integer.

Splitting Data :

```
X=data[['WindSpeed(m/s)','WindDirection']]
X.head()
```

```
y = data['ActivePower(kW)']
y.head()
```

```
0    380.047791
1    453.769196
2    306.376587
3    419.645905
4    380.650696
Name: ActivePower(kW), dtype: float64
```

```
X.to_csv('IndependentVariables.csv')
y.to_csv('DependentVariable.csv')
```

The features are then split as dependent and independent variables for training the model. Wind speed and wind direction is taken as independent variables whereas Active power generated is taken as dependent variable.

Importing the Regression Models :

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.metrics import accuracy_score,r2_score,mean_squared_error
xgr=XGBRegressor()
rf=RandomForestRegressor()
lr=LinearRegression()
dt=DecisionTreeRegressor()
sm=SVR()
```

The above command is used to import the required libraries to train the various models. Here we use five regression models for training namely Linear Regressor, XGBRegressor, Random Forest Regressor, Decision Tree Regressor and Support Vector Regression.

Fitting the Models with dataset :

```
model_xg=xgr.fit(X_train,y_train)
y_xg=model_xg.predict(X_test)
model_rf=rf.fit(X_train,y_train)
y_rf=model_rf.predict(X_test)
model_lr=lr.fit(X_train,y_train)
y_lr=model_lr.predict(X_test)
model_dt=dt.fit(X_train,y_train)
y_dt=model_dt.predict(X_test)
model_sm=sm.fit(X_train,y_train)
y_sm=model_sm.predict(X_test)
```

The above command is used to train the data. The five models are being fitted individually with the training data.

Checking the Metrics :

```
print('R2-xgb',r2_score(y_test,y_xg))
print('RMSE-xgb',np.sqrt(mean_squared_error(y_test,y_xg)))

print('R2-rf',r2_score(y_test,y_rf))
print('RMSE-rf',np.sqrt(mean_squared_error(y_test,y_rf)))

print('R2-lr',r2_score(y_test,y_lr))
print('RMSE-lr',np.sqrt(mean_squared_error(y_test,y_lr)))

print('R2-dt',r2_score(y_test,y_dt))
print('RMSE-dt',np.sqrt(mean_squared_error(y_test,y_dt)))

print('R2-svm',r2_score(y_test,y_sm))
print('RMSE-svm',np.sqrt(mean_squared_error(y_test,y_sm)))
[14]
```

```
R2-xgb 0.9222746826171284
RMSE-xgb 364.85477293970644
R2-rf 0.9097702879938478
RMSE-rf 393.10952377367164
R2-lr 0.8368251429450982
RMSE-lr 528.6465476346768
R2-dt 0.8388459591904157
RMSE-dt 525.3628747175155
R2-svm 0.005368134807760105
RMSE-svm 1305.1786596858901
```

This command prints the score of all the five models that we have fitted. It displays the accuracy of each of the model. From the above statement we can see that XGBRegressor model has the highest accuracy of 92%.

XGBRegressor Model Training :

```
xg=XGBRegressor(colsample_bylevel=0.4, colsample_bytree=0.3, gamma=0.1,
                learning_rate=0.01, max_depth=6, min_child_weight=25,
                n_estimators=1500, reg_alpha=0.1, reg_lambda=0.8, subsample=0.6)
x=xgr.fit(X_train,y_train)
y1=x.predict(X_test)
r2_score(y_test,y1)

[07:14:27] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

0.9222746826171284
```

Since XGBRegressor model best fits the model, we select that and give our dataset to obtain the trained model.

Saving the Model :

```
model_xg.save_model('test_model.bin')

data=[[5.311336,259.994904]]
df = pd.DataFrame(data, columns=['WindSpeed(m/s)','WindDirection'])
xgr.predict(df)
```

This command saves our trained model as a .bin file. This file can then be called upon by our application to perform the prediction. This model accepts Wind speed and Wind Direction as input and gives the power generated as output.

7.2 : Feature 2

Deploying the Model in IBM Cloud :

IBM Deployment

```
In [18]: !pip install -U ibm-watson-machine-learning

Requirement already satisfied: ibm-watson-machine-learning in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (1.0.255)
Collecting ibm-watson-machine-learning
  Downloading ibm_watson_machine_learning-1.0.257-py3-none-any.whl (1.8 MB)
     |████████████████████████████████| 1.8 MB 31.6 MB/s eta 0:00:01
```

Here the required library of IBM Watson Machine Learning is getting installed.

Authenticate and set Space

t1xJwH_pNvesyStso2tawTlpypHX0HEQJVMev99cmAtK

```
In [28]: wml_credentials = {
             "apikey":"iJ8fO2zR1zKFzMmJarCCyrgkg2xF1jaKtkVucFJAQJ1h",
             "url":"https://eu-de.ml.cloud.ibm.com"
         }

In [29]: wml_client = APIClient(wml_credentials)

In [30]: wml_client.spaces.list()
         #e0a978b3-0ab3-4800-987d-a39e08695233

         Note: 'limit' is not provided. Only first 50 records will be displayed if the number of records exceed 50
         ------------------------------------  -----------  ------------------------
         ID                                    NAME         CREATED
         e0a978b3-0ab3-4800-987d-a39e08695233  Wind Energy  2022-11-07T04:44:34.908Z
         ------------------------------------  -----------  ------------------------

In [32]: SPACE_ID= "e0a978b3-0ab3-4800-987d-a39e08695233"

In [33]: wml_client.set.default_space(SPACE_ID)

Out[33]: 'SUCCESS'
```

Using the unique API key generated in IBM Cloud and mentioning our server location. Using the API credentials a new space is created in IBM Watson. The space has its unique Space id.

```
In [36]: import sklearn
         sklearn.__version__

Out[36]: '1.0.2'

In [37]: MODEL_NAME = 'XGB_1'
         DEPLOYMENT_NAME = 'XGB_1'
         DEMO_MODEL = model_xg

In [38]: # Set Python Version
         software_spec_uid = wml_client.software_specifications.get_id_by_name('runtime-22.1-py3.9')

In [39]: software_spec_uid

Out[39]: '12b83a17-24d8-5082-900f-0ab31fbfd3cb'
```

Downloading the required ML model. Looking for the version that is being supported by IBM and downloading the correct version. Creating a new deployment space for the model.

```
In [40]: # Setup model meta
model_props = {
    wml_client.repository.ModelMetaNames.NAME: MODEL_NAME,
    wml_client.repository.ModelMetaNames.TYPE: 'scikit-learn_1.0',
    wml_client.repository.ModelMetaNames.SOFTWARE_SPEC_UID: software_spec_uid
}
```

```
In [41]: #Save model

model_details = wml_client.repository.store_model(
    model=DEMO_MODEL,
    meta_props=model_props,
    training_data=X_train,
    training_target=y_train
)
```

To set up the model requirements and link it to the deployment space. Saving the model to the space by mentioning the attributes of the model.

```
In [42]: model_details
```


```
Out[42]: {'entity': {'hybrid_pipeline_software_specs': [],
  'label_column': 'ActivePower(kW)',
  'schemas': {'input': [{'fields': [{'name': 'WindSpeed(m/s)',
       'type': 'float64'},
      {'name': 'WindDirection', 'type': 'float64'}],
     'id': '1',
     'type': 'struct'}],
   'output': []},
  'software_spec': {'id': '12b83a17-24d8-5082-900f-0ab31fbfd3cb',
   'name': 'runtime-22.1-py3.9'},
  'type': 'scikit-learn_1.0'},
 'metadata': {'created_at': '2022-11-07T04:56:31.773Z',
  'id': '7dd1db0c-ed59-4f73-b91b-e04cffd42347',
  'modified_at': '2022-11-07T04:56:34.488Z',
  'name': 'XGB_1',
  'owner': 'IBMid-666002NS6H',
  'resource_key': 'ae81f1ad-fa3a-4cb8-8dee-014487923830',
  'space_id': 'e0a978b3-0ab3-4800-987d-a39e08695233'},
 'system': {'warnings': []}}
```


To view the details of the model created.

```
In [44]: # Set meta
deployment_props = {
    wml_client.deployments.ConfigurationMetaNames.NAME:DEPLOYMENT_NAME,
    wml_client.deployments.ConfigurationMetaNames.ONLINE: {}
}
```

To set the configuration of the deployment. Giving the name for the deployment in IBM Watson.

```
In [45]: # Deploy
deployment = wml_client.deployments.create(
    artifact_uid=model_id,
    meta_props=deployment_props
)

#######################################################################################

Synchronous deployment creation for uid: '7dd1db0c-ed59-4f73-b91b-e04cffd42347' started

#######################################################################################


initializing
Note: online_url is deprecated and will be removed in a future release. Use serving_urls instead.

ready


-----------------------------------------------------------------------------------------------
Successfully finished deployment creation, deployment_uid='48a87a28-d849-4ab0-9203-ca3924b43312'
-----------------------------------------------------------------------------------------------
```

Deploying the model in IBM Cloud using model id. An id is created for the model using which the model can be accessed online.

Flask Application :

```
import flask
from flask import request, render_template
from flask_cors import CORS
import joblib
import pandas as pd
from xgboost import XGBRegressor
import requests
app = flask.Flask(__name__, static_url_path='')
CORS(app)
```

To import the required libraries

```python
API_KEY = "iJ8fO2zR1zKFzMmJarCCyrgkg2xF1jaKtkVucFJAQJ1h"
token_response = requests.post('https://iam.cloud.ibm.com/identity/token', data={"apikey":
 API_KEY, "grant_type": 'urn:ibm:params:oauth:grant-type:apikey'})
mltoken = token_response.json()["access_token"]

header = {'Content-Type': 'application/json', 'Authorization': 'Bearer ' + mltoken}
```

The API key and model id are used to link to the model that has been trained in IBM Cloud.

```python
@app.route('/', methods=['GET'])
def sendHomePage():
    return render_template('index.html')

@app.route('/predict', methods=['POST'])
def predictSpecies():
    ws = float(request.form['ws'])
    wd = float(request.form['wd'])

    X = [[ws,wd]]
    xgr=XGBRegressor()
    df = pd.DataFrame(X, columns=['WindSpeed(m/s)','WindDirection'])
    payload_scoring = {"input_data": [{"field": [['ws', 'wd']], "values":X}]}

    response_scoring = requests.post('https://eu-de.ml.cloud.ibm.com/ml/v4/deployments/782741b9-1e46-
    headers={'Authorization': 'Bearer ' + mltoken})
    print(response_scoring)
    predictions = response_scoring.json()
    print(predictions)
    predict = predictions['predictions'][0]['values'][0][0]
    print("Final prediction :",predict)
    return render_template('predict.html',predict=predict)

if __name__ == '__main__':
    app.run()
```

This program serves as the backend for our Web page API and linking our Machine Learning model with it. The input that has been received from the home page is then sent to out ML model to do the prediction and the output will be displayed at the next web page. It is the connection between the Frontend and backend.

HTML Code :

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>WIND TURBINE ENERGY PREDICTION</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/index.css') }}">
  </head>
  <body>
    <div class="container">
      <div class="glass">
        <h1 class="text" >WIND TURBINE <br>ENERGY PREDICTION</h1>
      <h2 class="text">Using XGBoost Model</h2>
        <br>
      <form method="POST" action="/predict">
        <p class="text">Wind Speed</p>
        <input name="ws" required />
        <p class="text">Wind Direction</p>
        <input name="wd" required />
        <br />
        <br />
        <button type="submit" class="submit">Submit</button>
      </div>
    </div>
  </body>
</html>
```

Code to design the home page. The page consists of a from wherein the user can enter the wind speed and Wind directions. When submitted the values are given to the model.

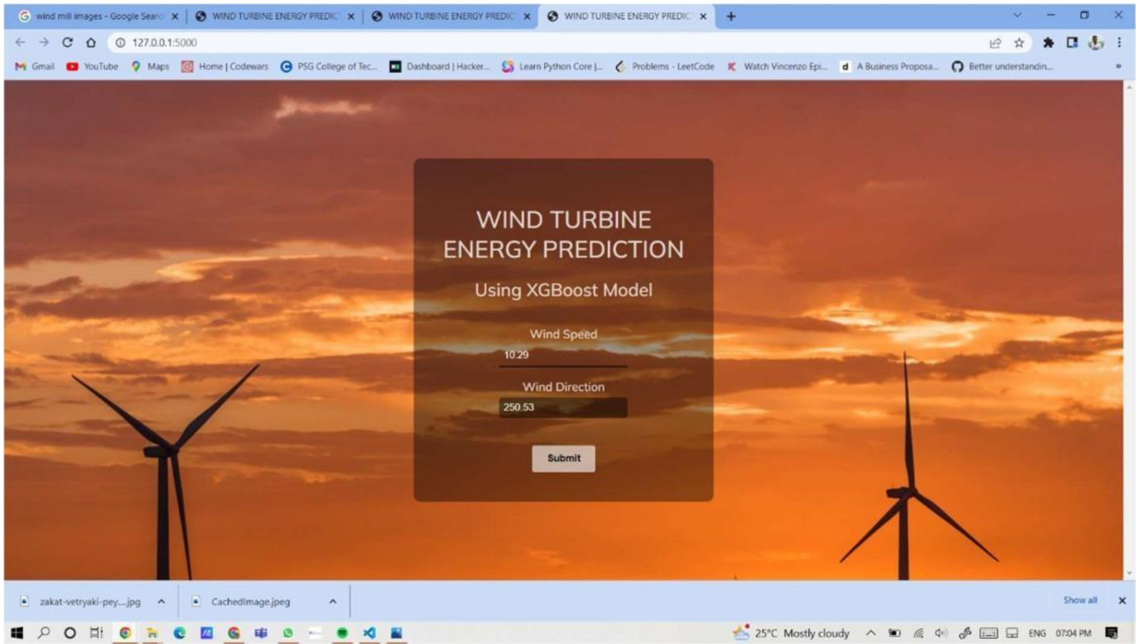
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="./css/index.css" />
    <title>Prediction</title>
</head>
<body>

    <div class="container">
        <div class="glassdoor">
            <h1 class="text">The predicted Output power is</h1>
            <h1 class="highlight">{{predict}}</h1>
            <a href="/" class="submit">Go Back</a>
        </div>
      </div>
</body>
</html>
```

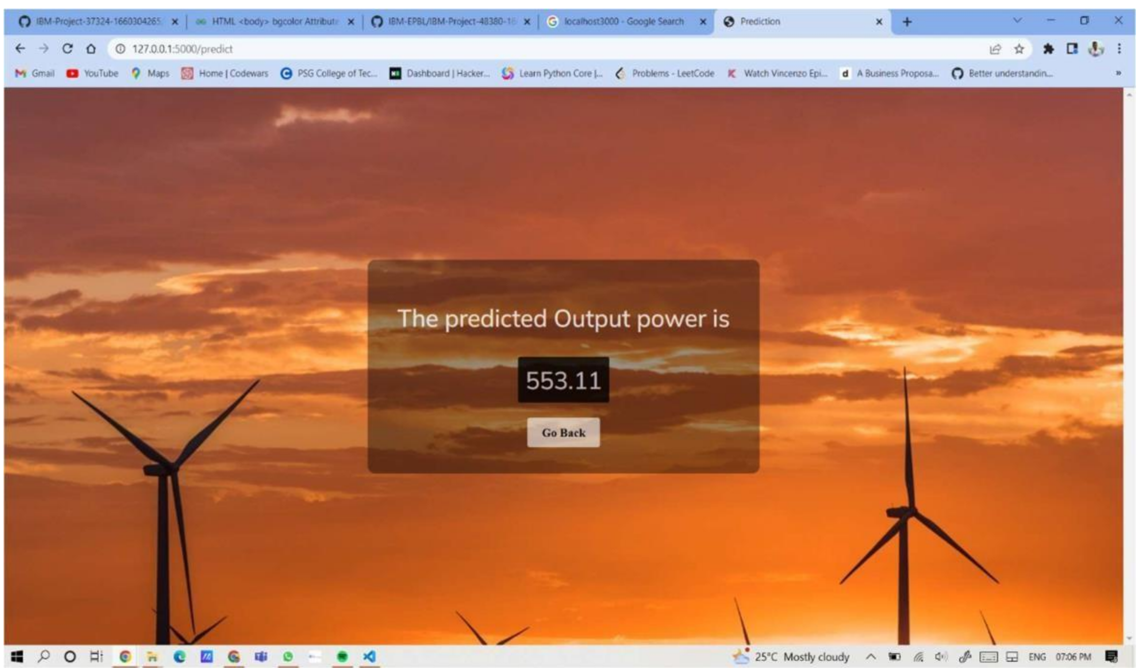
This page displays the output predicted value. This is a post method and hence receives the value from model and displays on the web page.

Web Page Design :

Home Page :



Output Page :



8 : TESTING

8.1 : Test Cases

Wind Speed (m/s)	Wind Direction (°)	Predicted Power Output (KW)

10.5	100.9	2695.02
6.6	290	751.88
30.7	220	3303.57
25.5	45	3595.69
19.1	0	1135.50
14.8	295	3758.29
8.3	180	1524.59
0.5	88	6.82
3.7	325	34.03
35.2	355	3819.80

8.2 : User Acceptance Testing

The project has been tested extensively with a number of users. The users found the interface very easy to use. The Web pages were colourful and attractive. There was no unnecessary details in the web page. It was clean and simple that any new user could master it. The data input format was also simple. The user need not enter any unit. He could simply enter the value. The prediction time is fairly low at an average time of 3 seconds. This delay primarily varies depending on the internet connectivity. The model has been hosted in IBM cloud. Thus with the API available, the model can be accessed remotely from any system provided IBM access key is given. The model predicts the power output close to the actual power generated. The users are satisfied with the predicted output power. Although the prediction is not very accurate it comes closer to the actual power. Various inputs have been given by the users to test the consistency of the model. The model proved itself and all the users accepted the model as a reliable and convenient.

9 : RESULTS

9.1 : Performance Metrics

The XGBRegressor ML model that we have used here has better performance in speed and accuracy compared to other models. We have compared the performance metrics of 5 models and selected this as the best for the application. The model performed well for all the test cases. The API developed also performed good with no glitches or lag found during the testing phase.

10 : Advantages and Disadvantages

10.1 : Advantages

This model takes in the previous years energy outputs and corelate it with the weather and other parameters that affected it. By using this model we can give the Weather conditions as input and obtain the energy output. It also dynamically alters the algorithm based on the predicted value and actual output value. This model helps in increasing the usage of renewable energy. It optimizes the operation of Wind Turbines. The cost of Implementing this solution makes it an Unformidable one. Wind Energy Companies will be able to increase their energy output thereby increasing revenue. Wind Energy can be trusted as a consistent source as we are able to predict the total power output for any given time. This doesn't require any additional equipment to be set up at the Wind turbine. The existing Sensors can be used to get the Weather parameters for predicting the power output. With Weather stations all across the world, the data can be obtained easily in real time. The prediction can be carried out at the control station of the Wind mills. The algorithm can be easily modified to work for every single Wind Turbine.

10.2 : Disadvantages

Wind Mill companies hesitate to completely rely on this model. Data availability is difficult for all the individual Wind Mills. The Wind Mill maybe in a remote location, providing connectivity to all of it proves challenging and expensive. Data Storage cost is very high, as the data for the output power and other attributes will be stored in the cloud. This is expensive for the company. The model needs Weather inputs for the prediction process. Error in this input values like Wind speed, Wind Direction, Temperature, Altitude, Humidity due to the inaccuracy in the instruments that is being can result in errors in prediction. Sudden changes in weather conditions prove difficult for the model to predict. The changing Climatic conditions across the globe every year, means that the previous year data is insignificant. Efficiency loss at the wind mill is difficult to calculate and it varies from one wind mill to the other. Human made changes like building infrastructures in the wind path can greatly affect the prediction which cannot be given as input. Server crash or loss of internet can leave the company with no other choice as the entire model is hosted in cloud.

11 : CONCLUSION

The XGBRegressor ML model that has been used above performs well for our dataset. The model is fast and consumes less resources. The API developed is also simple and userfriendly. By using this model, we could predict the output power of a wind turbine provided the required input parameter. This increases the use of Wind power and revenue for the companies. The model is not 100% accurate but it performs sufficiently. It can be concluded as the power output cannot be predicted very accurately as there are several parameters that could affect the output and all those outputs cannot be taken in for training as it can result in a very complex and overtrained model. The features that have high weightage are considered in this model.

12 : FUTURE SCOPE

The further works that can be done in this project is to include more features in model training to study the effect on the output. A long history of data (dataset of more than 3 years) can be used for training for increased accuracy. The application can be upgraded such that the input values from the sensors are directly fed to the model without the user entering it manually. More web pages can be designed so that the user can control more Wind Mill in the same API. Navigation tabs to move across various Wind mills. The dashboard can be made for User Interactive by making it to show real time graph of the prediction and actual power. Diagnosis of wind mill which perform the least can be done remotely.

13 : APPENDIX

13.1 : Source Code

Model Training :

```
import pandas as pd import numpy as np import
matplotlib.pyplot as plt import seaborn as sns from
sklearn.model_selection import train_test_split from
sklearn.linear_model import Lasso from
sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error ,
r2_score import joblib %matplotlib inline
data = pd.read_csv('https://raw.githubusercontent.com/IBM-EPBL/IBM-Project-
37324-1660304265/main/Data%20Collection/wind_dataset.csv')
data.rename(columns = {'LV ActivePower (kW)':'ActivePower(kW)',
                "Wind Speed (m/s)":"WindSpeed(m/s)",
                "Wind Direction (°)":"WindDirection","Theoretical_Power_Curve
(KWh)":"TheoreticalPowerCurve(KWh)"},
```

```
inplace = True) data.head() data.shape data.describe() data.info()
data.isnull().any() data['Date/Time'] =
pd.to_datetime(data['Date/Time'],format='%d %m %Y
%H:%M')
data['year'] = data['Date/Time'].dt.year  data['month'] =
data['Date/Time'].dt.month  data['day'] = data['Date/Time'].dt.day data['Hour'] =
data['Date/Time'].dt.hour  data['minute'] = data['Date/Time'].dt.minute
data.head() data["Date/Time"] = pd.to_datetime(data["Date/Time"], format =
"%d %m %Y
%H:%M", errors = "coerce") data

X=data[['WindSpeed(m/s)','WindDirection']
] X.head() y = data['ActivePower(kW)']
y.head()
X_train, X_test,y_train, y_test = train_test_split(X,y ,random_state=6,
test_size=0.25)

from sklearn.tree import DecisionTreeRegressor from sklearn.svm import SVR
from sklearn.linear_model import LinearRegression from sklearn.ensemble
import RandomForestRegressor from xgboost import XGBRegressor from
sklearn.metrics import
accuracy_score,r2_score,mean_squared_error xgr=XGBRegressor()
rf=RandomForestRegressor() lr=LinearRegression()
dt=DecisionTreeRegressor() sm=SVR()

model_xg=xgr.fit(X_train,y_train)
y_xg=model_xg.predict(X_test)
model_rf=rf.fit(X_train,y_train)  y_rf=model_rf.predict(X_test)
model_lr=lr.fit(X_train,y_train)  y_lr=model_lr.predict(X_test)
model_dt=dt.fit(X_train,y_train)  y_dt=model_dt.predict(X_test)
model_sm=sm.fit(X_train,y_train)
y_sm=model_sm.predict(X_test)

print('R2-xgb',r2_score(y_test,y_xg)) print('RMSE-
xgb',np.sqrt(mean_squared_error(y_test,y_xg)))  print('R2-rf',r2_score(y_test,y_rf))
print('RMSE-rf',np.sqrt(mean_squared_error(y_test,y_rf)))  print('R2lr',r2_score(y_test,y_lr))
 print('RMSE-lr',np.sqrt(mean_squared_error(y_test,y_lr)))
print('R2dt',r2_score(y_test,y_dt))
 print('RMSE-dt',np.sqrt(mean_squared_error(y_test,y_dt)))  print('R2-
svm',r2_score(y_test,y_sm))   print('RMSE-
svm',np.sqrt(mean_squared_error(y_test,y_sm)))
```

IBM Cloud Deployment :

```
!pip install -U ibm-watson-machine-learning from
ibm_watson_machine_learning import APIClient
import json wml_credentials = {
    "apikey":"iJ8fO2zR1zKFzMmJarCCyrgkg2xF1jaKtkVucFJAQJ1h",
    "url":"https://eu-de.ml.cloud.ibm.com"
}
wml_client = APIClient(wml_credentials)
wml_client.spaces.list()
SPACE_ID= "e0a978b3-0ab3-4800-987d-a39e08695233"
wml_client.set.default_space(SPACE_ID)
wml_client.software_specifications.list(100)

import sklearn
sklearn.__version__
MODEL_NAME = 'XGB_1'
DEPLOYMENT_NAME = 'XGB_1'
DEMO_MODEL = model_xg
software_spec_uid = wml_client.software_specifications.get_id_by_name('runtime-
22.1-py3.9')
software_spec_uid

model_props = {
    wml_client.repository.ModelMetaNames.NAME: MODEL_NAME,
wml_client.repository.ModelMetaNames.TYPE:          'scikit-learn_1.0',
wml_client.repository.ModelMetaNames.SOFTWARE_SPEC_UID
:
software_spec_uid
}

model_details =
wml_client.repository.store_model(     model=DEMO_MODEL,
meta_props=model_props,      training_data=X_train,
training_target=y_train
)

model_details model_id =
wml_client.repository.get_model_id(model_details) model_id deployment_props
= {
wml_client.deployments.ConfigurationMetaNames.NAME:DEPLOYMENT_N
AM E,      wml_client.deployments.ConfigurationMetaNames.ONLINE: {}
}  deployment =
wml_client.deployments.create(
```

```
artifact_uid=model_id,      meta_props=deployment_props
)
```

FLASK Application

```
import flask from flask import request,
render_template from flask_cors import
CORS import joblib import pandas as pd
from xgboost import XGBRegressor
import requests
app = flask.Flask(__name__, static_url_path='')
CORS(app)

API_KEY = "iJ8fO2zR1zKFzMmJarCCyrgkg2xF1jaKtkVucFJAQJ1h"
token_response = requests.post('https://iam.cloud.ibm.com/identity/token',
data={"apikey":
 API_KEY, "grant_type": 'urn:ibm:params:oauth:grant-type:apikey'}) mltoken
= token_response.json()["access_token"]

header = {'Content-Type': 'application/json', 'Authorization': 'Bearer ' + mltoken}
@app.route('/', methods=['GET']) def sendHomePage():
    return render_template('index.html')

@app.route('/predict', methods=['POST'])
def predictSpecies():      ws =
float(request.form['ws'])
wd = float(request.form['wd'])

    X = [[ws,wd]]      xgr=XGBRegressor()      df = pd.DataFrame(X,
columns=['WindSpeed(m/s)','WindDirection'])     payload_scoring =
{"input_data": [{"field": [['ws', 'wd']], "values":X}]}

    response_scoring = requests.post('https://eu-
de.ml.cloud.ibm.com/ml/v4/deployments/782741b9-1e46-
4126943af0696c250c0e/predictions?version=2022-11-07',
json=payload_scoring,     headers={'Authorization': 'Bearer ' +
mltoken})     print(response_scoring)     predictions =
response_scoring.json()     print(predictions)
    predict = predictions['predictions'][0]['values'][0][0]
print("Final      prediction      :",predict)      return
render_template('predict.html',predict=predict)

if __name__ == '__main__':
app.run()
```

Home Web Page :

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>WIND TURBINE ENERGY PREDICTION</title>
    <link  rel="stylesheet"  href="{{  url_for('static',  filename='css/index.css')  }}">
</head>
  <body>
    <div class="container">
      <div class="glass">
        <h1 class="text" >WIND TURBINE <br>ENERGY PREDICTION</h1>
      <h2 class="text">Using XGBoost Model</h2>
        <br>
      <form method="POST" action="/predict">
        <p class="text">Wind Speed</p>
        <input name="ws" required />
        <p class="text">Wind Direction</p>
        <input name="wd" required />
        <br />
        <br />
        <button type="submit" class="submit">Submit</button>
      </div>
    </div>
  </body>
</html>
```

Output Web Page :

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="./css/index.css" />
    <title>Prediction</title>
</head>
<body>
```

```
    <div class="container">
        <div class="glassdoor">
            <h1 class="text">The predicted Output power is</h1>
            <h1 class="highlight">{{predict}}</h1>
            <a href="/" class="submit">Go Back</a>
        </div>
      </div>
</body>
</html>
```

CSS :

```
@import
url('https://fonts.googleapis.com/css2?family=Mulish:ital,wght@0,400;0,500;0,600;1
,400;1,500;1,600&display=swap');

html, body {
overflow-y: scroll;
overflow-x:
hidden;   padding:
0;      margin:  0;    }
body   {        height:
100vh;           width:
100vw;    }  body  {
scrollbargutter:
10px;
}
.container  {     height:  100%;
width:                         100%;
backgroundimage:
url("4.jpg");        background-
size:   cover;       background-
repeat: no-repeat;

}
.container,form{     display: flex;
justify-content: center;     alignitems:
center;     flex-direction: column; }
.glass,.glassdoor{     padding: 40px;
backgroundcolor: rgba(0,0,0,.4);
borderradius: 10px;
}
.glassdoor{     height: 200px;     display:
flex;                flex-direction:   column;
alignitems:    center;                 justify-
```

```
content:spaceevenly;          gap:10px;  }
input{     margintop: 5px;     outline: 0;
border:   none;               border-bottom:
rgba(0,0,0,.7) 2px solid;     background:
transparent;                  padding:   6px;
color:white; } input:focus{       margin-
top:   5px;               background-color:
rgba(0,0,0,.45);              border-bottom:
transparent 2px solid;      border: none;
outline:   0;             borderradius:   4px;
padding: 6px;
}

.text{     font-family: "Mulish";
color:rgba(255,255,255,.8);
margin-bottom: 0;
fontweight: 500;     text-align:
center;
} .highlight{     font-family:
"Mulish";     color:rgba(225, 214, 214,
0.8);     margin-bottom: 10px;     font-
weight: 500;     padding: 10px;
background-color: rgba(0,0,0,.8);
border-radius: 3px;
}
.submit{               padding:10px    20px;
borderradius:   3px;              border:   0;
backgroundcolor:rgba(255,255,255,.6);
fontweight:   600;    }      .submit:hover{
cursor:  pointer;  }  a{         outline:none;
textdecoration: none;     color:inherit;  }
```

13.2 : GitHub & Project Demo Link

GitHub Repo : IBM-EPBL/**IBM-Project-52161-1660990045**

s