Weather-Based Prediction of Wind Turbine Energy Output: A Next-Generation Approach to Renewable Energy Management

Project Description:

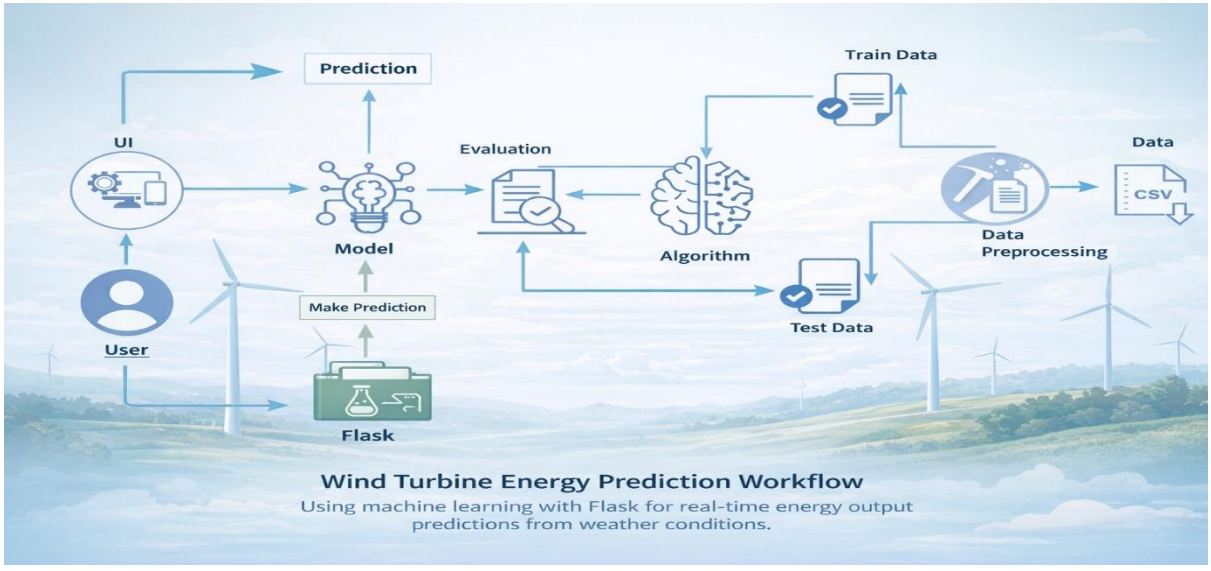
The project aims to predict the energy output of a wind turbine based on weather conditions. This is valuable for energy companies and grid operators to better manage and optimize energy production. By analyzing historical data of weather conditions and energy output, machine learning models can be trained to predict the energy output of a wind turbine given current weather conditions.

Scenario 1: Energy Production Forecasting Energy companies want to forecast the energy production of their wind turbines for a given period. They can use machine learning models to predict the energy output based on weather forecasts, helping them make informed decisions about energy distribution and pricing.

Scenario 2: Maintenance Planning Wind farm operators want to plan maintenance schedules for their turbines to minimize downtime and maximize energy production. By predicting energy output based on weather conditions, they can schedule maintenance during periods of low wind activity.

Scenario 3: Grid Integration Grid operators want to integrate wind energy into the grid efficiently. By predicting the energy output of wind turbines, they can better balance the grid by adjusting the output of other energy sources accordingly.

Technical Architecture:



Pre requisites:

To complete this project, you must required following software's, concepts and packages

- **Anaconda navigator and pycharm:**
  - Refer the link below to download anaconda navigator
  - Link : https://youtu.be/1ra4zH2G4o0

- **To build Machine learning models you must require the following packages**
  - **Sklearn:** Scikit-learn is a library in Python that provides many unsupervised and supervised learning algorithms.
  - **NumPy:** NumPy is a Python package that stands for 'Numerical Python'. It is the core library for scientific computing, which contains a powerful n-dimensional array object
  - **Pandas:** pandas is a fast, powerful, flexible, and easy to use open-source data analysis and manipulation tool, built on top of the Python programming language.
  - **Matplotlib:** It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits
  - **Flask:** Web framework used for building Web applications.
  - Watch the video below to learn how to install packages.

- **Python packages:**
  - Open anaconda prompt as administrator
  - Type "pip install numpy" and click enter.
  - Type "pip install pandas" and click enter.
  - Type "pip install scikit-learn" and click enter.
  - Type "pip install matplotlib" and click enter.
  - Type "pip install scipy" and click enter.
  - Type "pip install pickle-mixin" and click enter.
  - Type "pip install seaborn" and click enter.
  - Type "pip install Flask" and click enter.

Prior Knowledge:

You must have prior knowledge of following topics to complete this project.

- **ML Concepts**
  - Supervised learning: https://www.javatpoint.com/supervised-machine-learning
  - Unsupervised learning: https://www.javatpoint.com/unsupervised-machine-learning
  - Regression and classification
  - Decision tree: https://www.javatpoint.com/machine-learning-decision-tree-

classification-algorithm

- Random forest: https://www.javatpoint.com/machine-learning-random-forest-algorithm
- KNN: https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning
- Xgboost: https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/
- Evaluation metrics: https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/

- **Flask Basics** : https://www.youtube.com/watch?v=lj4I_CvBnt0

Project Objectives:

By the end of this project, you will:

- You'll be able to understand the problem to classify if it is a regression or a classification kind of problem.
- You will be able to know how to pre-process/clean the data using different data preprocessing techniques.
- You will able to analyze or get insights into data through visualization.
- Applying different algorithms according to the dataset and based on visualization.
- You will be able to know how to build a web application using the Flask framework.

Project Flow:

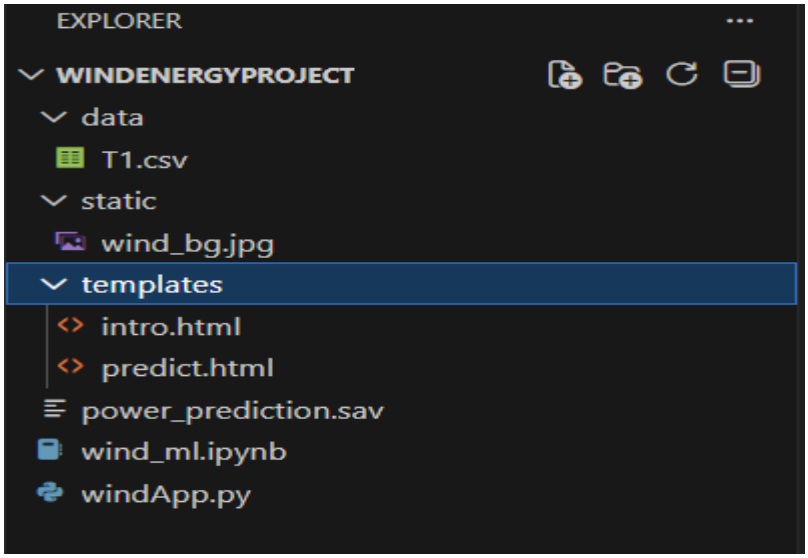
- User interacts with the UI to enter the input.
- Entered input is analyzed by the model which is integrated.
- Once model analyses the input the prediction is showcased on the UI

To accomplish this, we have to complete all the activities listed below,

- Data Collection.
  - Collect the dataset or Create the dataset
- Data Preprocessing.
  - Import the Libraries.
  - Importing the dataset.
  - Checking for Null Values.
  - Data Visualization.
  - Taking care of Missing Data.
  - Label encoding.
  - One Hot Encoding.
  - Feature Scaling.
  - Splitting Data into Train and Test.
- Model Building
  - Training and testing the model
  - Evaluation of Model
- Application Building
  - Create an HTML file
  - Build a Python Code

Project Structure:

Create the Project folder which contains files as shown below



- We are building a flask application which needs HTML pages stored in the templates folder and a python script windApp.py for scripting.
- power_prediction.sav is our saved model. Further we will use this model for flask integration.
- static folder contains the images for web application
- templates folder contains the HTML pages
- .csv file is the dataset
- .ipynb files are the training and testing files.

Milestone 1: Data Collection

ML depends heavily on data, It is most crucial aspect that makes algorithm training possible. So this section allows you to download the required dataset.

Activity 1: Download the dataset

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

In this project we have used T1.csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

Link: https://www.kaggle.com/datasets/camila2401andrea/t1-csv

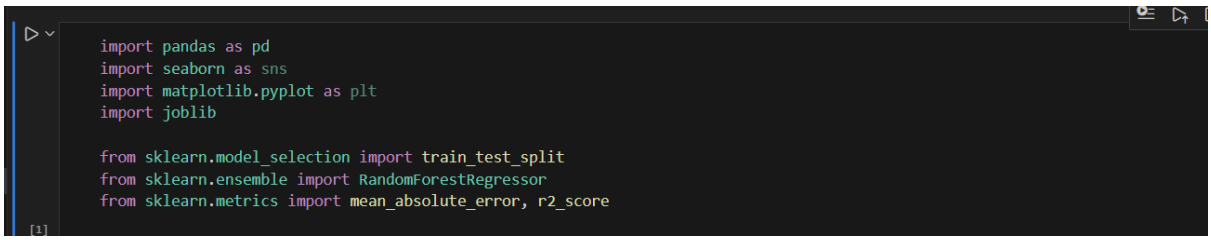
Milestone 2: Visualizing and analysing the data

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualization techniques and some analysing techniques.

Note: There is n number of techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

Activity 1: Importing the libraries

Import the necessary libraries as shown in the image.

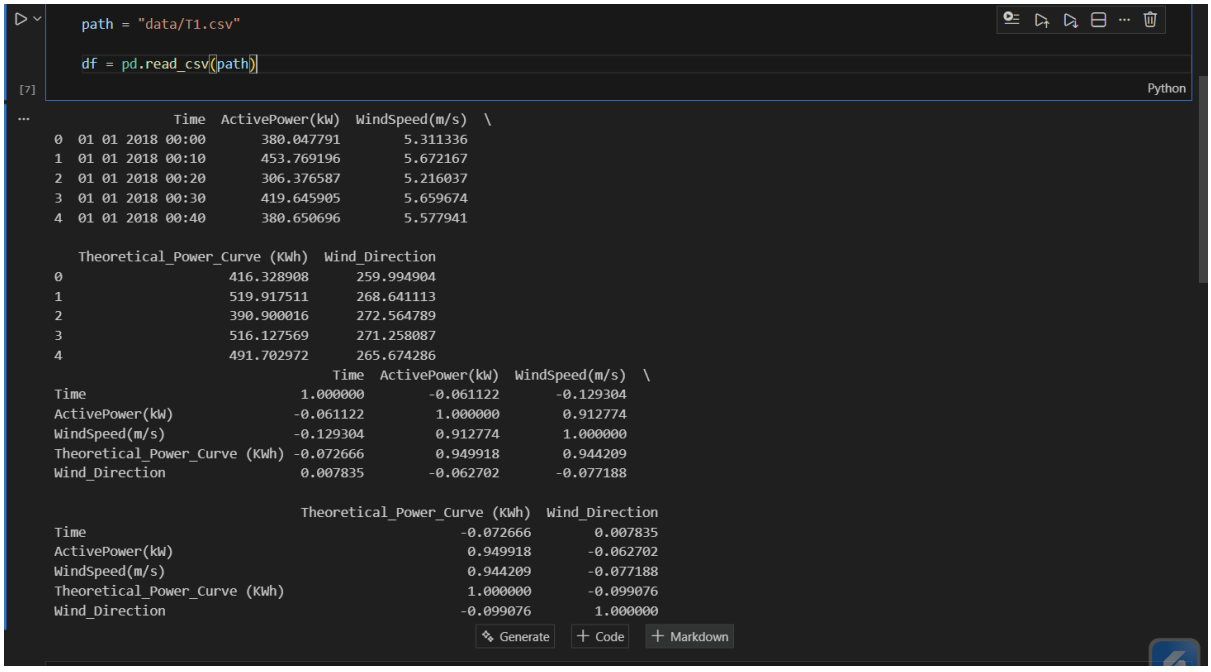


```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import joblib

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, r2_score
```

Activity 2: Read the Dataset

Our dataset format might be in .csv, excel files, .txt, .json, etc. We can read the dataset with the help of pandas.

In pandas we have a function called read_csv() to read the dataset. As a parameter we have to give the directory of csv file.



```
path = "data/T1.csv"

df = pd.read_csv(path)
```

```
                 Time  ActivePower(kW)  WindSpeed(m/s)  \
0  01 01 2018 00:00       380.047791        5.311336
1  01 01 2018 00:10       453.769196        5.672167
2  01 01 2018 00:20       306.376587        5.216037
3  01 01 2018 00:30       419.645905        5.659674
4  01 01 2018 00:40       380.650696        5.577941

   Theoretical_Power_Curve (KWh)  Wind_Direction
0                     416.328908      259.994904
1                     519.917511      268.641113
2                     390.900016      272.564789
3                     516.127569      271.258087
4                     491.702972      265.674286
                                Time  ActivePower(kW)  WindSpeed(m/s)  \
Time                        1.000000        -0.061122       -0.129304
ActivePower(kW)            -0.061122         1.000000        0.912774
WindSpeed(m/s)             -0.129304         0.912774        1.000000
Theoretical_Power_Curve (KWh) -0.072666      0.949918        0.944209
Wind_Direction              0.007835        -0.062702       -0.077188

                            Theoretical_Power_Curve (KWh)  Wind_Direction
Time                                            -0.072666        0.007835
ActivePower(kW)                                  0.949918       -0.062702
WindSpeed(m/s)                                   0.944209       -0.077188
Theoretical_Power_Curve (KWh)                    1.000000       -0.099076
Wind_Direction                                  -0.099076        1.000000
```

Activity 3: Analyze the Datasets

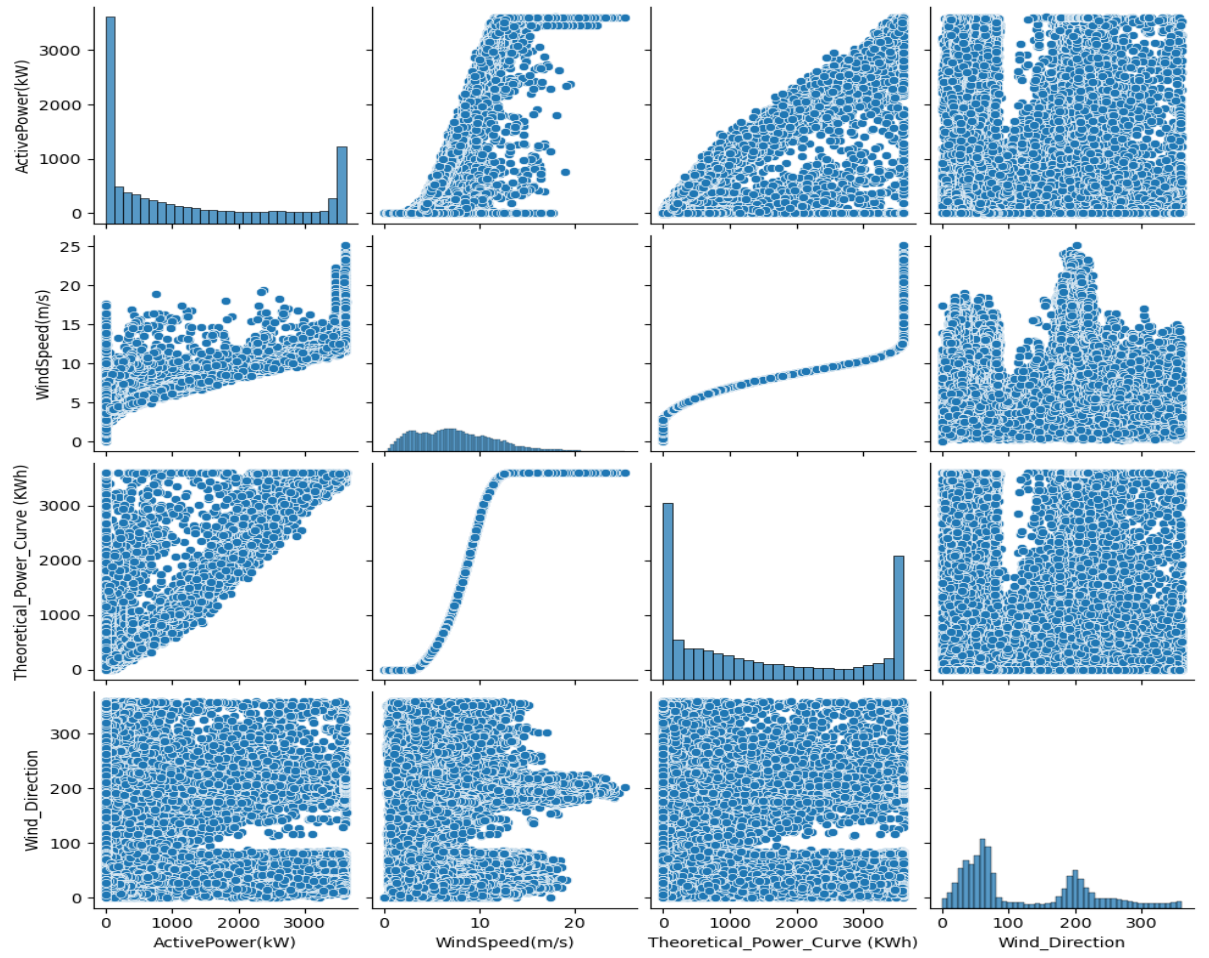
Step 1: The datasets are imported as data frames using the pandas library. Rename the columns

with suitable column names for better understanding.
*Dataset contains the wind speed and wind direction along with the power generated.

```
path = "data/T1.csv"

df = pd.read_csv(path)

df.rename(columns={
'Date/Time':'Time',
'LV ActivePower (kW)':'ActivePower(kW)',
'Wind Speed (m/s)':'WindSpeed(m/s)',
'Wind Direction (°)':'Wind_Direction'
}, inplace=True)

print(df.head())

# Convert Time
df["Time"] = pd.to_datetime(df["Time"], errors="coerce")

# Correlation
corr = df.corr()
print(corr)
```

Step 2: Check the correlation between the columns for dimensionality reduction (knowing which columns are necessary and which are not)

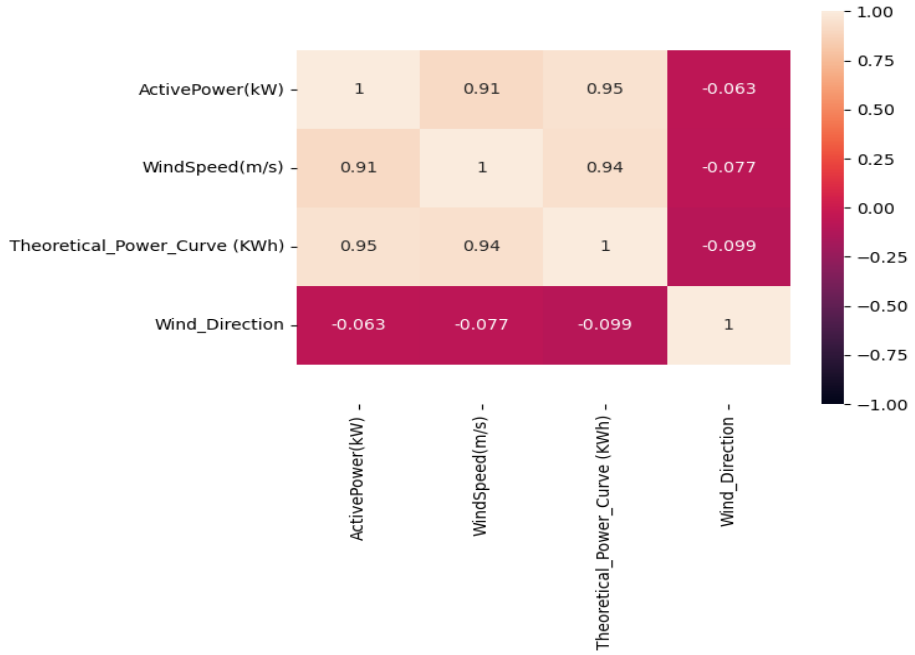
```
sns.pairplot(df)
plt.figure(figsize=(10,8))
plt.show()
```



HEATMAP:

```
ax=sns.heatmap(corr,vmin=-1,vmax=1,annot=True)
bottom,top=ax.get_ylim()
ax.set_ylim(bottom+0.5,top-0.5)
```

[6] ✓ 0.2s



The heat map clearly tells us that there's no relation between wind direction and the Power generated but Wind speed, Theoretical power and Actual power generated to have a very positive correlation.

Activity 4: Descriptive analysis

Descriptive analysis is to study the basic features of data with the statistical process. Here pandas has a worthy function called describe. With this describe function we can understand the unique, top and frequent values of categorical features. And we can find mean, std, min, max and percentile values of continuous features.

```
df.describe()
```

[8] ✓ 0.0s

	ActivePower(kW)	WindSpeed(m/s)	Theoretical_Power_Curve (KWh)	Wind_Direction
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

Milestone 3: Data Pre-processing

As we have understood how the data is lets pre-process the collected data.

The download data set is not suitable for training the machine learning model as it might have so much of randomness so we need to clean the dataset properly in order to fetch good results. This activity includes the following steps.

- Handling missing values
- Handling categorical data
- Handling outliers
- Scaling Techniques
- Splitting dataset into training and test set

Note: These are the general steps of pre-processing the data before using it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps.

Activity 1: Checking for null values

- Let's find the shape of our dataset first, To find the shape of our data, df.shape method is used. To find the data type, df.info() function is used.

```
df.info()
```
```
[9]  ✓ 0.0s
```
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50530 entries, 0 to 50529
Data columns (total 5 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   Time                          50530 non-null  datetime64[ns]
 1   ActivePower(kW)               50530 non-null  float64
 2   WindSpeed(m/s)                50530 non-null  float64
 3   Theoretical_Power_Curve (KWh) 50530 non-null  float64
 4   Wind_Direction                50530 non-null  float64
dtypes: datetime64[ns](1), float64(4)
memory usage: 1.9 MB
```

- For checking the null values, df.isnull() function is used. To sum those null values we use .sum() function to it. From the below image we found that there are no null values present in our dataset. So we can skip handling of missing values step.

```
df.isnull().sum()
```
[10] ✓ 0.0s

```
Time                              0
ActivePower(kW)                   0
WindSpeed(m/s)                    0
Theoretical_Power_Curve (KWh)     0
Wind_Direction                    0
dtype: int64
```

From the above code of analysis, we can infer that columns are not having the missing values.

We will fill the missing values in numeric data type using mean value of that particular column and categorical data type using the most repeated value.

Activity 2: Splitting data into train and test

Now let's split the Dataset into train and test sets

Changes: first split the dataset into x and y and then split the data set

Here x and y variables are created. On x variable, df is passed with dropping the target variable. And on y target variable is passed. For splitting training and testing data we are using train_test_split() function from sklearn. As parameters, we are passing x, y, test_size, random_state.

```
# Splitting data
y = df['ActivePower(kW)']
X = df[['Theoretical_Power_Curve (KWh)','WindSpeed(m/s)']]

train_X, val_X, train_y, val_y = train_test_split(X, y, random_state=0)
```
[11] ✓ 0.0s

Milestone 4: Model Building

Now our data is cleaned and it's time to build the model. We can train our data on different algorithms. for this project we are applying four classification algorithms. The best model is saved based on its performance.

Activity 1: Random forest model

A function named randomForest is created and train and test data are passed as the parameters. Inside the function, RandomForestClassifier algorithm is initialized and training data is passed

to the model with .fit() function. Test data is predicted with .predict() function and saved in new variable. For evaluating the model, confusion matrix and classification report is done.

```
[9]
# Model
forest_model = RandomForestRegressor(n_estimators=300, random_state=1)
forest_model.fit(train_X, train_y)
```

RandomForestRegressor

Parameters

n_estimators	300
criterion	'squared_error'
max_depth	None
min_samples_split	2
min_samples_leaf	1
min_weight_fraction_leaf	0.0
max_features	1.0
max_leaf_nodes	None
min_impurity_decrease	0.0
bootstrap	True
oob_score	False
n_jobs	None
random_state	1
verbose	0
warm_start	False
ccp_alpha	0.0
max_samples	None
monotonic_cst	None

Activity 2: Evaluating performance of the model and saving the model

Here we will be evaluating the model built. We will be using the test set for evaluation. The test set is given to the model for prediction and prediction values are stored in another variable called y_pred. The r2 score of the model is calculated and its performance is estimated.

Note: To understand cross validation, refer this link. https://towardsdatascience.com/cross-validation-explained-evaluating-estimator-performance-e51e5430ff85.

```
[10]
# Prediction
power_preds = forest_model.predict(val_X)

print("MAE:", mean_absolute_error(val_y, power_preds))
print("R2:", r2_score(val_y, power_preds))
```

```
MAE: 198.272716823753
R2: 0.8711607797474977
```

```
# Save model
joblib.dump(forest_model, "power_prediction.sav")

print("Model saved successfully")
```

[11]

```
Model saved successfully
```

Milestone 5: Application Building

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.

This section has the following tasks

- Building HTML Pages
- Building serverside script

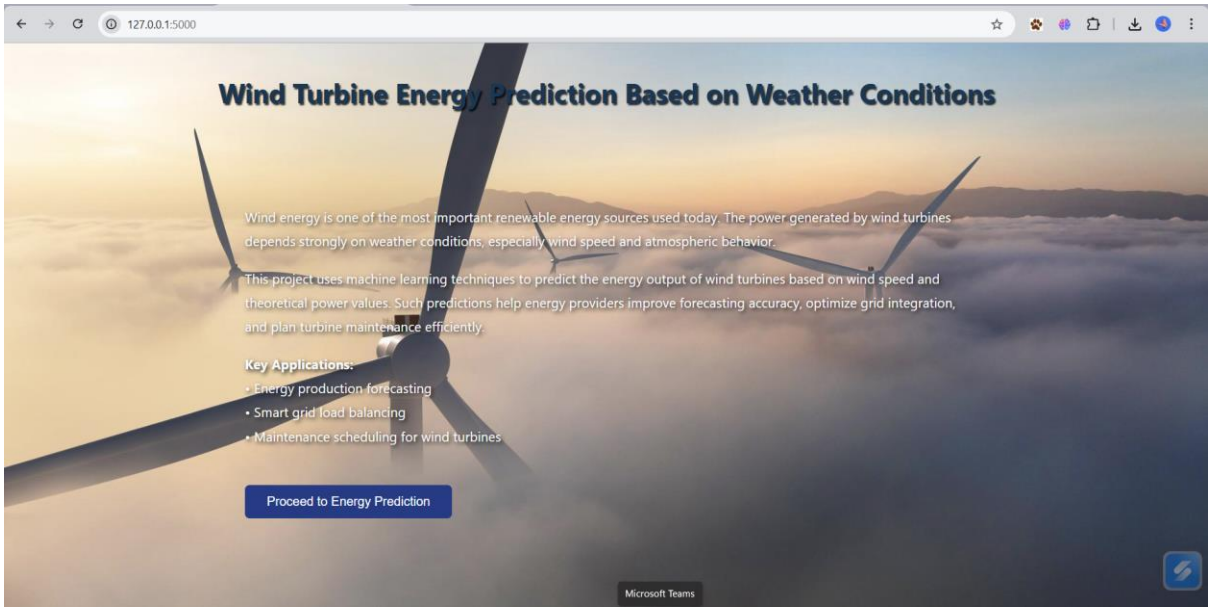
Activity1: Building Html Pages:

For this project create three HTML files namely

- into.html
- predict.html

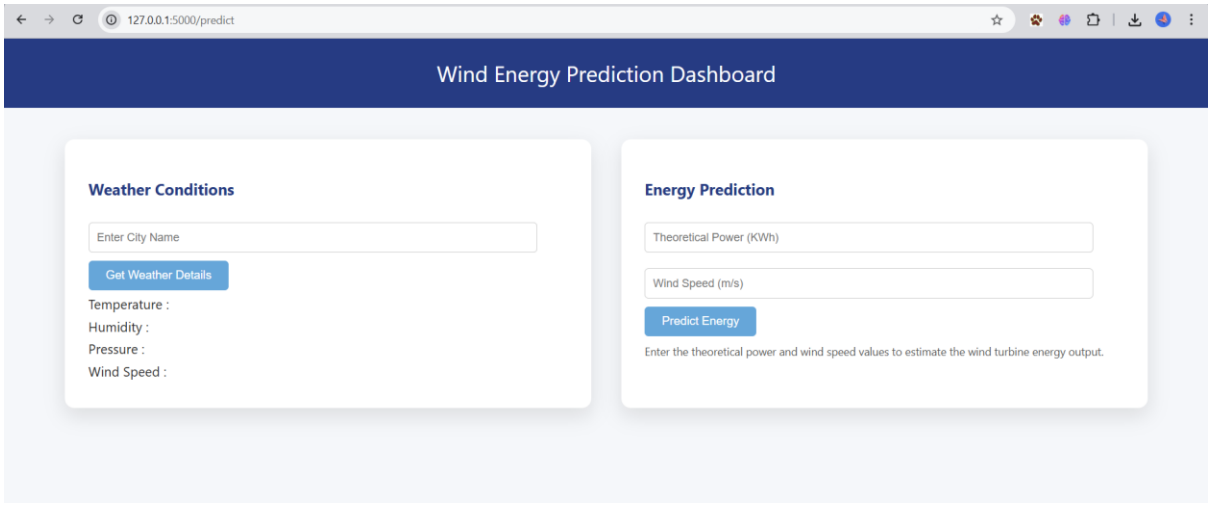
and save them in templates folder.

Let's see how our intro.html page looks like:

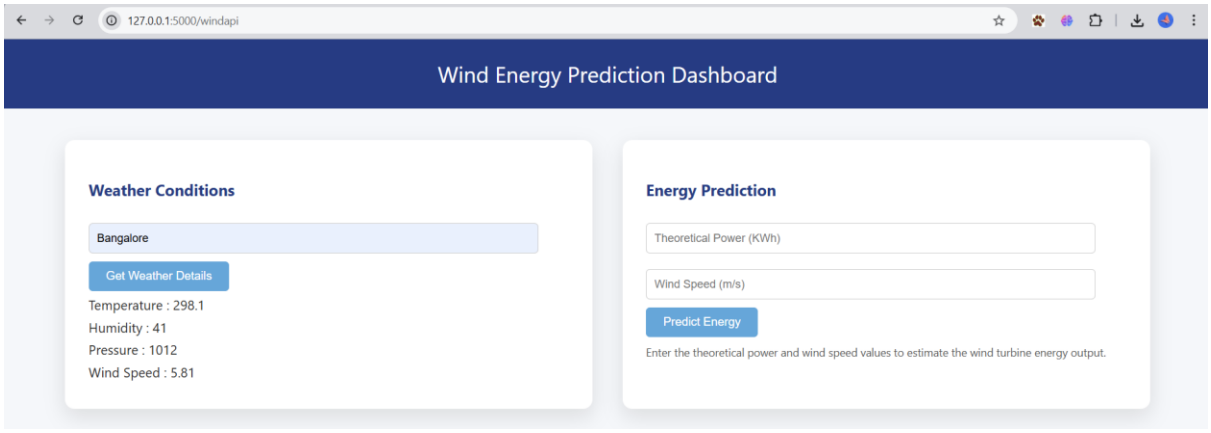


Now when you click on predict button from bottom you will get redirected to predict.html

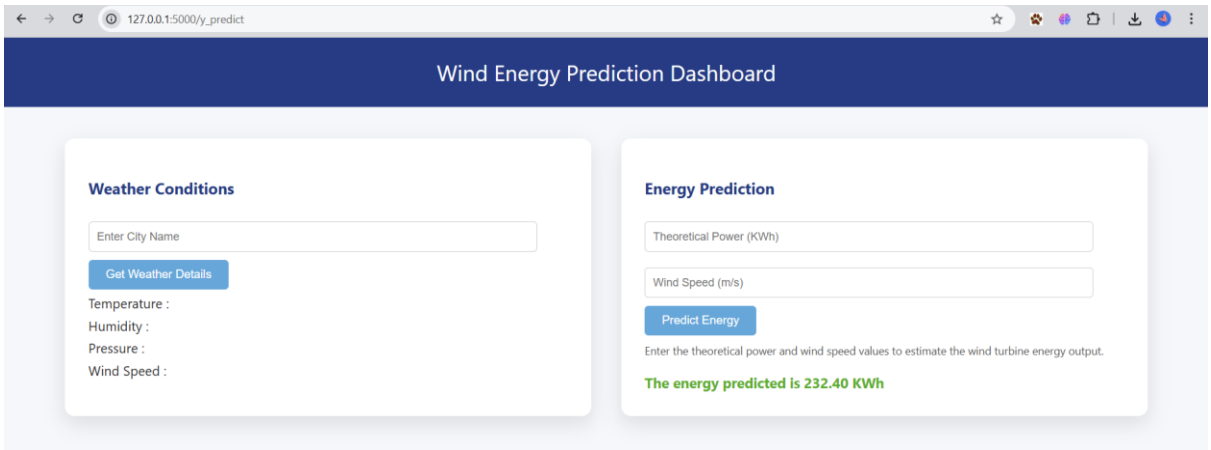
Let’s look how our predict.html file looks like:



After entering a city name and click Get Weather Details but to show these details.



After giving details like Theoretical Power and Wind Speed and click Predict Energy button:



Activity 2: Build Python code:

In the flask application, the API requests, as well as energy prediction requests, are taken and the results are processed.

Import the libraries

```
from flask import Flask, render_template, request
import joblib
import requests
```

Load the saved model. Importing flask module in the project is mandatory. An object of Flask class is our WSGI application. Flask constructor takes the name of the current module (__name__) as argument.

```
app = Flask(__name__)
model = joblib.load("power_prediction.sav")
```

Render HTML page:

```
@app.route('/')
def home():
    return render_template('intro.html')

@app.route('/predict')
def predict():
    return render_template('predict.html')

@app.route('/windapi',methods=['POST'])
```

Here we will be using declared constructor to route to the HTML page which we have created earlier.

In the above example, '/' URL is bound with home.html function. Hence, when the home page of the web server is opened in browser, the html page will be rendered. Whenever you enter the values from the html page the values can be retrieved using POST Method.

Retrieves the value from UI:

Flask file takes the city as input and hits the API to get the weather conditions and send it back to the UI.

It takes the inputs from the UI and passes it to the model and sends the predicted output to the UI.

```
def windapi():
    city=request.form.get("city")

    apikey="aa216216ed717b2347ecafdbfe2b7c02"
    url="http://api.openweathermap.org/data/2.5/weather?q="+city+"&appid="+apikey

    resp=requests.get(url).json()

    temp=str(resp["main"]["temp"])
    humid=str(resp["main"]["humidity"])
    pressure=str(resp["main"]["pressure"])
    speed=str(resp["wind"]["speed"])

    return render_template('predict.html',
                           temp=temp,
                           humid=humid,
                           pressure=pressure,
                           speed=speed)

@app.route('/y_predict',methods=['POST'])
def y_predict():
    x_test=[[float(x) for x in request.form.values()]]
    prediction=model.predict(x_test)
    output=prediction[0]

    return render_template('predict.html',
    prediction_text='The energy predicted is {:.2f} KWh'.format(output))
```

Here we are routing our app to predict() function. This function retrieves all the values from the HTML page using Post request. That is stored in an array. This array is passed to the model.predict() function. This function returns the prediction. And this prediction value will rendered to the text that we have mentioned in the submit.html page earlier.

Main Function:

```
if __name__=="__main__":
    app.run(debug=True)
```

Activity 3: Run the application

- Open command prompt from the new terminal
- Navigate to the folder where your python script is.
- Now type “python windApp.py” command
- Navigate to the localhost where you can view your web page.
- Click on the predict button from top left corner, enter the inputs, click on the submit button, and see the result/prediction on the web.

```
D:\WindEnergyProject>python windApp.py
 * Serving Flask app 'windApp'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a prod
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 106-245-920
```