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

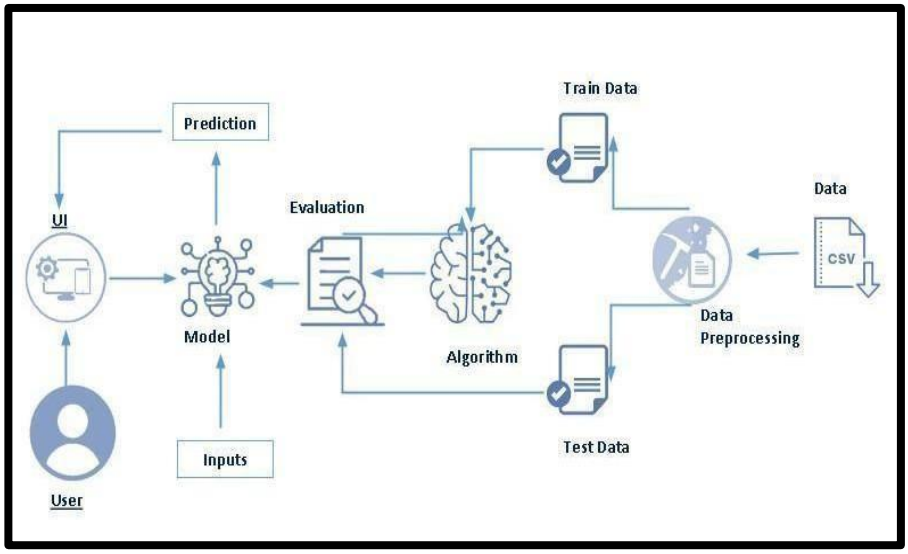
Team ID : LTVIP2025TMIDS40838

Team Leader : Lakshmi Swarupa Tadiboina
Team member : Tanniru Ramyasri
Team member : Tata Bhavya Sri
Team member : Telaganeti Sambasiva Rao

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

The project **"Weather-Based Prediction of Wind Turbine Energy Output: A Next-Generation Approach to Renewable Energy Management"** focuses on developing a predictive model that estimates the energy output of wind turbines using weather data. Wind energy generation is highly dependent on environmental factors such as wind speed, temperature, humidity, and air pressure. Accurate prediction of turbine output helps energy companies and grid operators to manage power distribution efficiently and minimize wastage. By analyzing historical weather patterns and corresponding turbine performance data, machine learning algorithms can learn complex correlations between weather conditions and energy generation. The system leverages these insights to forecast energy production in real time, aiding in operational planning. This approach not only enhances energy reliability but also supports sustainable energy utilization. It enables better resource allocation, maintenance scheduling, and grid load balancing. Ultimately, this next-generation solution contributes to the **optimization of renewable energy management** and promotes a cleaner, data-driven future for power systems.

Technical Architecture:



Project Objectives

By the end of this project, you will be able to:

1. **Understand the Problem Type:**
    - Analyze a given dataset and determine whether the problem is a regression (predicting continuous values) or classification (predicting discrete categories) problem.
    - Identify the target variable and understand its nature for appropriate modeling.
2. **Data Preprocessing and Cleaning:**
    - Handle missing, inconsistent, or duplicate data using suitable techniques.
    - Apply data transformation methods such as normalization, scaling, encoding categorical variables, and feature engineering to make data suitable for modeling.
    - Detect and treat outliers to improve model accuracy.
    -
3. **Data Analysis and Visualization:**
    - Explore the dataset using statistical methods and visualization tools.
    - Identify patterns, correlations, and trends within the data.
    - Generate actionable insights from visualizations like histograms, scatter plots, box plots, heatmaps, and pair plots.
4. **Algorithm Selection and Application:**
    - Understand different machine learning algorithms suitable for classification or regression tasks.
    - Train, test, and evaluate multiple models to compare performance metrics such as accuracy, precision, recall, F1-score, RMSE, or MAE.
    - Apply hyperparameter tuning and cross-validation techniques to optimize model performance.
5. **Model Deployment through Web Application:**
    - Learn how to integrate machine learning models into a Flask web application.
    - Design user-friendly interfaces for users to input data and receive predictions in real-time.
    - Deploy the application locally or on cloud platforms to make it accessible for end-users.
6. **End-to-End Project Experience:**
    - Gain practical experience in taking a project from problem understanding → data preprocessing → model building → evaluation → deployment.
    - Develop critical thinking and problem-solving skills for real-world machine learning projects.
    - Understand best practices for maintaining reproducibility, documentation, and scalability of ML projects.

Project Flow

The project follows a systematic workflow to develop a machine learning-based web application.

You will go through all the steps mentioned below to complete the project successfully:

1. **User Interaction with the Interface:**
    - Users will interact with a web-based User Interface (UI) to input their data.
    - The UI is designed to be intuitive and user-friendly, ensuring smooth data entry.

2. **Data Analysis by the Model:**
    - Once the data is entered, it is sent to the integrated machine learning model.
    - The model analyzes the data and processes it to generate predictions.

3. **Displaying Predictions:**
    - After analysis, the predictions or results are displayed on the UI in a clear and understandable format.
    - This allows users to quickly interpret the outcomes without technical knowledge.

4. **Activities and Tasks for Completion:**

A. Data Collection:

- Collect existing datasets from reliable sources or create a custom dataset relevant to the problem.
- Ensure the dataset contains sufficient and representative samples for accurate model training.

B. Data Preprocessing:

- Import Libraries: Import essential Python libraries such as pandas, numpy, scikit-learn, matplotlib, and seaborn.
- Import Dataset: Load the dataset into the workspace for processing.
- Check for Null Values: Identify and handle missing or inconsistent data.

C. Data Visualization and Cleaning:

- Visualize the data to understand patterns, distributions, and correlations.
- Handle missing data using techniques like mean/mode imputation or removal of incomplete records.
- Apply Label Encoding for categorical variables.
- Use One-Hot Encoding for variables with multiple categories.
- Perform Feature Scaling to normalize or standardize numerical features.

D. Splitting Data:

- Divide the dataset into training and testing subsets to evaluate model performance objectively.

E. Model Building:

- Select suitable algorithms based on the problem type (classification or regression).
- Train the model using the training dataset.
- Test the model using the testing dataset to validate its performance.

F. Model Evaluation:

- Evaluate the model using metrics such as accuracy, precision, recall, F1-score (for classification), or RMSE/MAE (for regression).
- Optimize the model using techniques like hyperparameter tuning, cross-validation, or feature selection.

G. Application Development:

- Create HTML Files: Design front-end pages for user interaction, input forms, and displaying results.
- Build Python Backend: Use Flask to integrate the trained model with the web application.

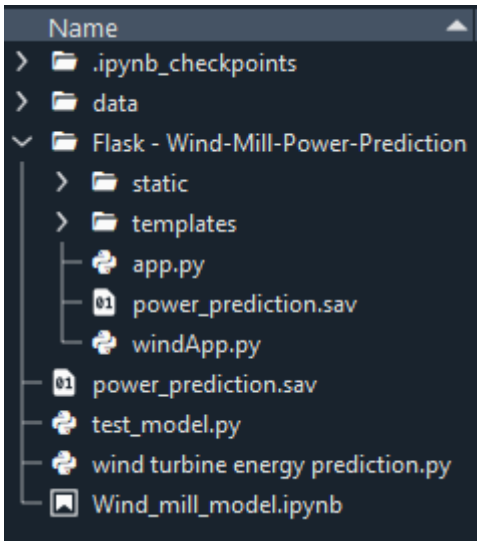
- Connect Frontend and Backend: Ensure smooth communication between the UI and the model for real-time predictions.

5. **Final Deployment:**
   - Test the complete application to ensure proper functionality.
   - Deploy locally or on cloud platforms to make the application accessible to users.

6. **End-to-End Integration:**
   - The workflow ensures a complete pipeline from data collection → preprocessing → model building → evaluation → web deployment, providing hands-on experience with real-world machine learning projects**.**

Project Structure

Create a Project folder which contains files as shown below



- Flask folder contains the necessary files for a web application:
- static folder contains the images for web application

- templates folder contains the HTML pages
- .sav file is the model file
- windApp.py is for server-side scripting
- .csv file is the dataset
- .py files are the training and testing files.

Download Dataset / Create Dataset

For the **Wind Energy Prediction** project, the dataset is a crucial component, as it contains the environmental conditions that influence wind turbine energy output. The dataset typically includes features such as:

- **Wind Speed** (m/s)
- **Temperature** (°C)
- **Humidity** (%)
- **Air Pressure** (hPa)
- **Wind Direction** (degrees)
- **Energy Output** (kW) – This serves as the target variable for prediction.

Steps for Obtaining the Dataset:

1. **Collecting from Open Sources:**

    - There are multiple publicly available repositories where you can obtain real-world datasets:
        - **Kaggle (www.kaggle.com):** Offers a variety of datasets related to weather, wind energy, and renewable energy projects.
        - **Data.gov (www.data.gov):** Provides datasets published by government organizations, including weather and energy production data.
        - **UCI Machine Learning Repository (archive.ics.uci.edu/ml/index.php):** Contains curated datasets for machine learning experiments.
    - Ensure that the dataset you choose is **complete, accurate, and relevant** for wind energy prediction.

2. **Creating a Custom Dataset (if needed):**

    - If no suitable dataset is available, you can create your own dataset by collecting real-time weather and energy data.

    - Sources for custom data:
        - Local weather stations or online weather APIs (e.g., OpenWeatherMap API, Weatherbit API).
        - Historical wind turbine output records from local renewable energy organizations.

    - Collect enough records to ensure the dataset is statistically significant for training and testing machine learning models.

3. **Data Formatting and Cleaning:**

    - Ensure that the dataset is structured in **tabular format** (CSV, Excel, or database) with rows representing observations and columns representing features.
    - Check for **missing or inconsistent data** and prepare it for preprocessing.

4. **Considerations for Dataset Selection:**

    - The dataset should cover **various environmental conditions** to improve the model's predictive accuracy.
    - It should include **sufficient historical records** to capture seasonal variations in wind patterns.
    - Prefer datasets with **continuous and reliable energy output measurements**.

Collecting a high-quality dataset lays the foundation for accurate prediction and ensures the reliability of the machine learning model in real-world applications.

Data Preprocessing

Import Required Libraries

In any machine learning project, importing the necessary libraries is the first step to set up the environment for data analysis, preprocessing, modeling, and visualization. Libraries are imported in Python using the import keyword.

```
# Importing Necessary Libraries
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns
import joblib
```

Analyze the Datasets

Analyzing the dataset is a crucial step in any machine learning project. It helps in understanding the structure, patterns, and relationships in the data, which in turn informs preprocessing and model-building decisions.

Step 1: Importing and Preparing the Dataset

- The dataset is imported into the workspace as a DataFrame using the pandas library:
- After importing, it is important to rename the columns with descriptive names for better readability and understanding
- The dataset contains key features such as Wind Speed, Wind Direction, Theoretical Power, and Actual Power Generated, which are essential for predicting wind turbine energy output.
- Initial exploration can be done using methods like data.head(), data.info(), and data.describe() to get an overview of the dataset, its types, and statistical summaries.

```
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)
```

Step 2: Correlation Analysis

- Correlation analysis helps identify the relationships between features, which is useful for **dimensionality reduction** and avoiding redundant variables.
- Using the corr() function, we can compute the correlation between all numerical columns:

```
#Data Description and Visualizing

#Plotting the pair plot, each variable in the data set is plotted with all other variables
sns.pairplot(df)

#Plotting Correlation between the variables
plt.figure(figsize=(10, 8))
corr = df.corr()
ax = sns.heatmap(corr, vmin = -1, vmax = 1, annot = True)
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
plt.show()

#Print the numerical form for the correlation
print(corr)
```

```
# The heat map clearly tells us that there's no realtion between wind direction and
# the Power generated but Wind speed, Theoritical power and Actual power generated
# have a very positive correlation

#df.drop(['Wind_Direction'],axis=1,inplace = True)
df["Time"] = pd.to_datetime(df["Time"], format = "%d %m %Y %H:%M", errors = "coerce")
```

Observations from Correlation Analysis:

- The **heatmap** indicates:
    - **Wind Direction** has very low or no correlation with **Actual Power Generated**, meaning it may not significantly affect the power output.
    - **Wind Speed**, **Theoretical Power**, and **Actual Power Generated** show a **very strong positive correlation**, indicating that higher wind speeds and theoretical power values directly contribute to higher actual energy output.

- Based on this, we can consider **dropping features with very low correlation** (like Wind Direction) to simplify the model and reduce computational complexity.

Step 3: Data Insights and Next Steps

- Strong correlations provide a foundation for **feature selection**, which improves model efficiency and accuracy.
- This analysis also informs **data preprocessing decisions**, such as scaling features with large ranges (like Wind Speed and Power values) or encoding categorical variables if present.

- Overall, analyzing the dataset ensures that the machine learning model is trained on the **most relevant and informative features**, leading to more accurate predictions.

Splitting Data into Independent and Dependent Variables

In machine learning, it is essential to distinguish between **independent (input) variables** and the **dependent (target/output) variable**. This separation allows the model to learn patterns from the input features and predict the output effectively.

Step 1: Identify Independent and Dependent Variables

- The **independent variables** (also called features or predictors) are the columns that influence the target variable. These are usually denoted as X.
- The **dependent variable** (also called target or response variable) is the column we want to predict. This is denoted as y.
- For the wind energy prediction project:
    - **Independent Variables (X):** Wind Speed, Theoretical Power, and any other relevant environmental features.
    - **Dependent Variable (y):** Actual Power Generated by the wind turbine.

Step 2: Splitting the Dataset into Training and Testing Sets

- Once the features and target are identified, the dataset must be split into two subsets: **training set** and **testing set**.
- **Training Set:** Used to train the machine learning model so it can learn the relationship between independent and dependent variables.
- **Testing Set:** Used to evaluate the model's performance on unseen data, ensuring that it generalizes well and is not overfitted.
- A common split ratio is **80% for training** and **20% for testing**. This ensures the model has enough data to learn while retaining a portion for validation.

Step 3: Considerations for Splitting:

- **Random State:** Setting a random_state ensures that the split is reproducible.
- **Stratification (if applicable):** For classification tasks, it's sometimes important to maintain the proportion of classes in both training and testing sets.
- **Shuffling:** Data should be shuffled before splitting to avoid any bias due to the order of observations.

Step 4: Next Steps:

- After splitting the data, the training set will be used to **train the regression model** to predict actual power.
- The testing set will be used to **evaluate the model's accuracy** using metrics such as RMSE, MAE, or R^2.
- Proper splitting ensures that the model performance evaluation reflects **real-world predictive capability** rather than memorization of training data.

```
#Splitting the Data into train and test :
y = df['ActivePower(kW)'] #'Theoretical_Power_Curve (KWh)'
X = df[['Theoretical_Power_Curve (KWh)','WindSpeed(m/s)']]#'ActivePower(kW)'

from sklearn.model_selection import train_test_split
train_X, val_X, train_y, val_y = train_test_split(X, y,random_state = 0)
```

Model Building

Choose the Appropriate Model

Choosing the right machine learning model is critical for achieving accurate predictions and reliable performance. For the Wind Energy Prediction project, the Random Forest Regressor is selected as the primary model due to its advantages in handling complex datasets and providing robust predictions.

Implementing the Random Forest Regressor

1. **Import the Model**
2. **Initialize the Model:**

   Set hyperparameters such as the number of trees (n_estimators), maximum depth (max_depth), and random state for reproducibility.

3. **Fit the Model:**

- Train the model using the training dataset (X_train and y_train).

4: Next Steps:

- After fitting the model, it can be used to **predict power generation** on the test dataset.
- Evaluate the model performance using metrics such as **Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R^2 score**.
- If needed, **hyperparameter tuning** can be applied to optimize the model further.

```
#Importing libraries for Model training and fitting
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error,r2_score

forest_model = RandomForestRegressor(n_estimators = 750, max_depth = 4,
                                     max_leaf_nodes =500, random_state=1)
forest_model.fit(train_X, train_y)
```

Check the Metrics of the Model

After building the Random Forest Regressor, the next crucial step is to evaluate its performance using the test dataset. The test set, which the model has not seen during training, is used to generate predictions that are stored in a variable, commonly named y_pred. These predictions are then compared with the actual target values to assess how accurately the model can predict wind turbine energy output. Evaluating the model on unseen data helps ensure that it generalizes well and is not overfitted to the training data.

To quantify the model's performance, several regression metrics are used. The R^2 score measures how well the model explains the variance in the actual power output, with values closer to 1 indicating better predictions. Additionally, Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) provide insight into the average deviation of predicted values from actual values, with lower values indicating higher accuracy. By analyzing these metrics, we can determine the reliability of the model and identify areas for improvement, such as hyperparameter tuning, feature engineering, or cross-validation, to further enhance prediction accuracy.

```
#Predicting for Test Data
power_preds = forest_model.predict(val_X)

#Evaluating the score of our model
print(mean_absolute_error(val_y, power_preds))
print(r2_score(val_y,power_preds))
```

Save the Model

Once the model has been trained and evaluated successfully, the final step is to save it for future use. Saving the model allows us to reuse it for predictions without retraining, which is particularly useful when deploying the model in a web application or sharing it with others. In Python, machine learning models are commonly saved using the pickle module, which serializes the trained model object into a file that can be loaded later.

The model can be saved in different file formats, such as .pkl or .sav. For example, using pickle, the trained Random Forest Regressor can be saved.

Later, the saved model can be loaded back into memory without retraining, making it convenient for real-time predictions or integration with a web application.

Saving the model ensures reproducibility, efficiency, and portability. It also allows seamless deployment in web applications, cloud platforms, or other production environments, making the trained model readily accessible for predicting wind energy output using new input data**.**

```
#Saving the model for future reference
joblib.dump(forest_model, "power_prediction.sav")
```

API Integration

To make the wind energy prediction system dynamic and capable of handling real-time data, we integrate it with the **OpenWeather API**, which provides current weather forecasts for any location worldwide. This allows the model to receive up-to-date weather data, such as wind speed, temperature, and humidity, and use it for predicting energy output in real time.

Step 1: Sign Up for OpenWeather API

- Visit the OpenWeather website and sign up for a free or paid account to access the API.
- Complete the verification process to activate your account.

Step 2: API Key Activation

- After signing up and subscribing to the desired plan, the **API key** is typically activated within 24 hours.
- This key is essential for authenticating your requests to the OpenWeather API and ensures that only authorized users can access the service.

Step 3: Using the API Key

- The API key is used in API requests to fetch weather data for a specific city or location. The city name is passed as a parameter q, and the API key is passed as appid.

- The returned JSON data includes parameters such as **wind speed, temperature, humidity, and atmospheric pressure**, which can be extracted and fed into the wind energy prediction model.

Step 4: Integration with the Prediction Model

- Once the weather data is retrieved, it can be preprocessed to match the format expected by the trained machine learning model.
- This integration allows the system to **predict wind energy output in real-time** for any city, making the application dynamic and more practical for real-world usage.

By integrating the OpenWeather API, the project moves beyond static datasets and becomes capable of handling **live, actionable data**, which significantly enhances the utility and relevance of the wind energy prediction system.

Application Building

Build the Python Flask App

The Flask web application acts as the interface between the user, the OpenWeather API, and the trained wind energy prediction model. It handles API requests, retrieves real-time weather data, processes it, and returns the predicted energy output to the user in a user-friendly format.

Step 1: Import Required Libraries

- Import the essential Python libraries for Flask, API requests, and model handling:

```
import numpy as np
from flask import Flask, request, jsonify, render_template
import joblib
import requests
```

Step 2: Load the Model and Initialize Flask App

- Load the trained Random Forest model from the saved .pkl file and initialize the Flask app:

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

- This ensures the model is ready to make predictions whenever the app receives input from the user.

Step 3: Configure Flask for API Requests

- The Flask app accepts a **city name** from the UI and sends a request to the OpenWeather API to retrieve current weather conditions.

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

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

@app.route('/windapi',methods=['POST'])
def windapi():
    city=request.form.get('city')
    apikey="43ce69715e2133b2300e0f8f7289befd"
    url="http://api.openweathermap.org/data/2.5/weather?q="+city+"&appid="+apikey
    resp = requests.get(url)
    resp=resp.json()
    temp = str(resp["main"]["temp"])+" °C"
    humid = str(resp["main"]["humidity"])+" %"
    pressure = str(resp["main"]["pressure"])+" mmHG"
    speed = str(resp["wind"]["speed"])+" m/s"
    return render_template('predict.html', temp=temp, humid=humid, pressure=pressure,speed=speed)
```

The API response is processed and returned to the front-end, so the UI can display current weather conditions.

Step 4: Configure Flask for Predictions

- The app also accepts input values from the UI, passes them to the trained model, and returns the predicted wind energy output:

```
@app.route('/y_predict',methods=['POST'])
def y_predict():
    '''
    For rendering results on HTML GUI
    '''
    x_test = [[float(x) for x in request.form.values()]]

    prediction = model.predict(x_test)
    print(prediction)
    output=prediction[0]
    return render_template('predict.html', prediction_text='The energy predicted is {:.2f} KWh'.format(output))
```

This route handles the main prediction functionality and dynamically updates the UI with the result.

Step 5: Run the App

- Start the Flask server by running the following command in your terminal:

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

- Open a web browser and go to http://127.0.0.1:5000/ to access the application.
- Users can now enter a city or input weather parameters, and the app will fetch real-time weather data and predict the corresponding wind energy output.
-

Build an HTML Page

To make the wind energy prediction system interactive, we build an **HTML page** that serves as the user interface. This page allows users to **input a city name** for fetching real-time weather data through the OpenWeather API and also **enter custom parameters** for energy prediction. The HTML page acts as the front-end for the Flask application, sending user inputs to the backend and displaying the results dynamically.

The page typically contains:

1. **Weather Forecast Form:**
    - An input field where users can enter a city name.
    - A submit button that triggers a request to the /get_weather route in the Flask app.
    - A section to display the weather data returned by the API, such as wind speed, temperature, and humidity.

2. **Energy Prediction Form:**

   - Input fields for required parameters like wind speed and theoretical power.
   - A submit button that sends the data to the /predict route of the Flask app.
   - A section to display the predicted power output dynamically on the page.

3. **Styling and Layout:**

   - The page can be styled using **CSS** to make it visually appealing and user-friendly.

   1. Responsive design ensures that the page works well on both desktops and mobile devices.

4. **Integration with Flask:**

   - The HTML page uses **form actions** to submit user inputs to the Flask backend.

   1. Data returned from Flask routes (weather data and predictions) is displayed using placeholders or dynamically updated sections in the HTML page.

Project Structure

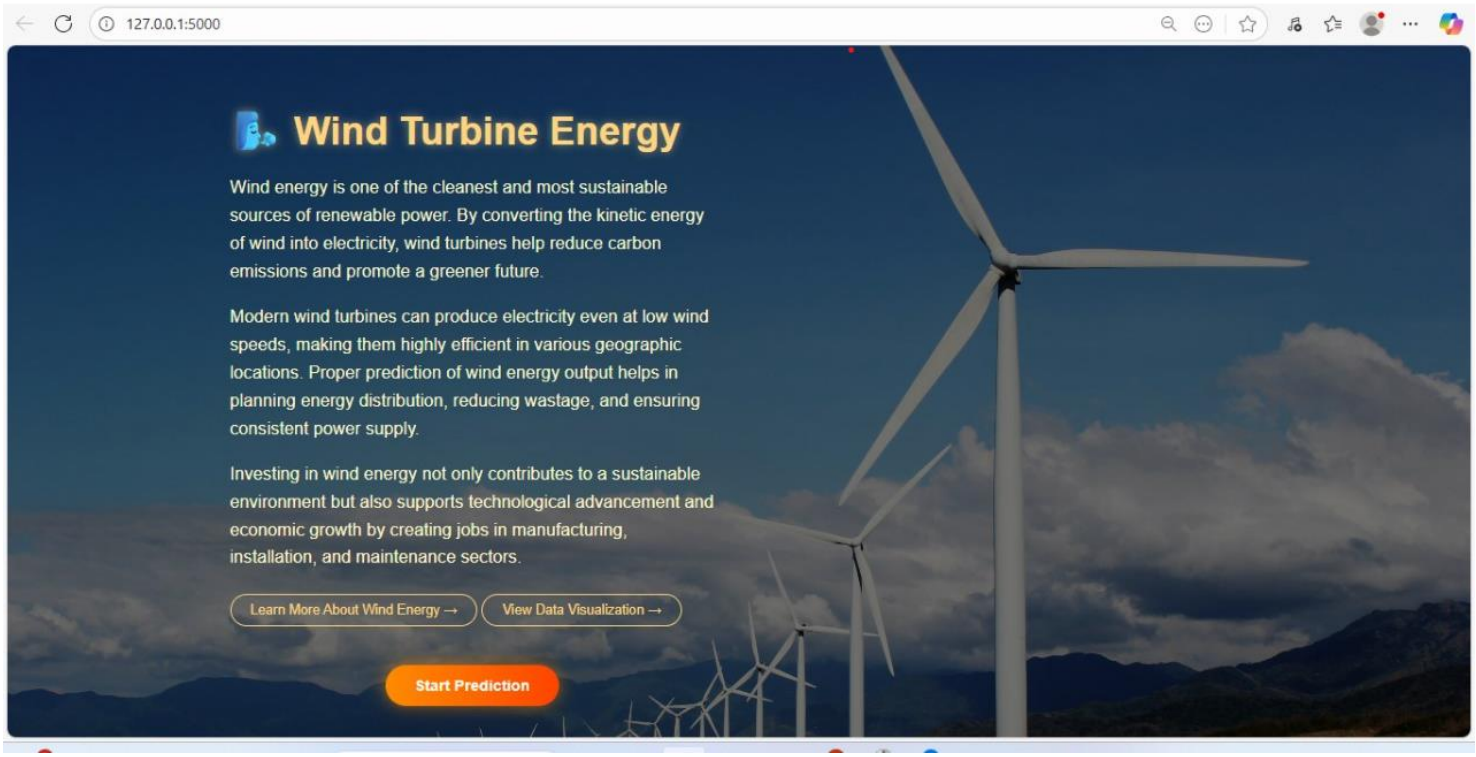
```
wind-energy-prediction/
│
├── app.py                  # Main Flask application file
├── wind_energy_model.pkl       # Trained Random Forest model saved using pickle
├── requirements.txt          # List of Python dependencies and libraries
├── templates/              # HTML templates for Flask
│   └── index.html           # Main web page for input and displaying predictions
├── static/                 # Static files (CSS, JavaScript, images)
│   ├── css/
│   │   └── style.css         # Styling for HTML pages
│   ├── js/
│   │   └── script.js         # Optional JS for dynamic UI updates
│   └── images/
│       └── logo.png          # Project logo or background images
├── data/                   # Folder containing datasets
│   └── wind_energy_data.csv   # Historical wind and weather dataset
├── notebooks/              # Jupyter notebooks for development and experimentation
│   └── data_analysis.ipynb    # Data preprocessing, visualization, and model building
├── utils/                  # Optional utility scripts
│   └── preprocessing.py       # Functions for data cleaning and feature scaling
└── README.md               # Project documentation and instructions
```

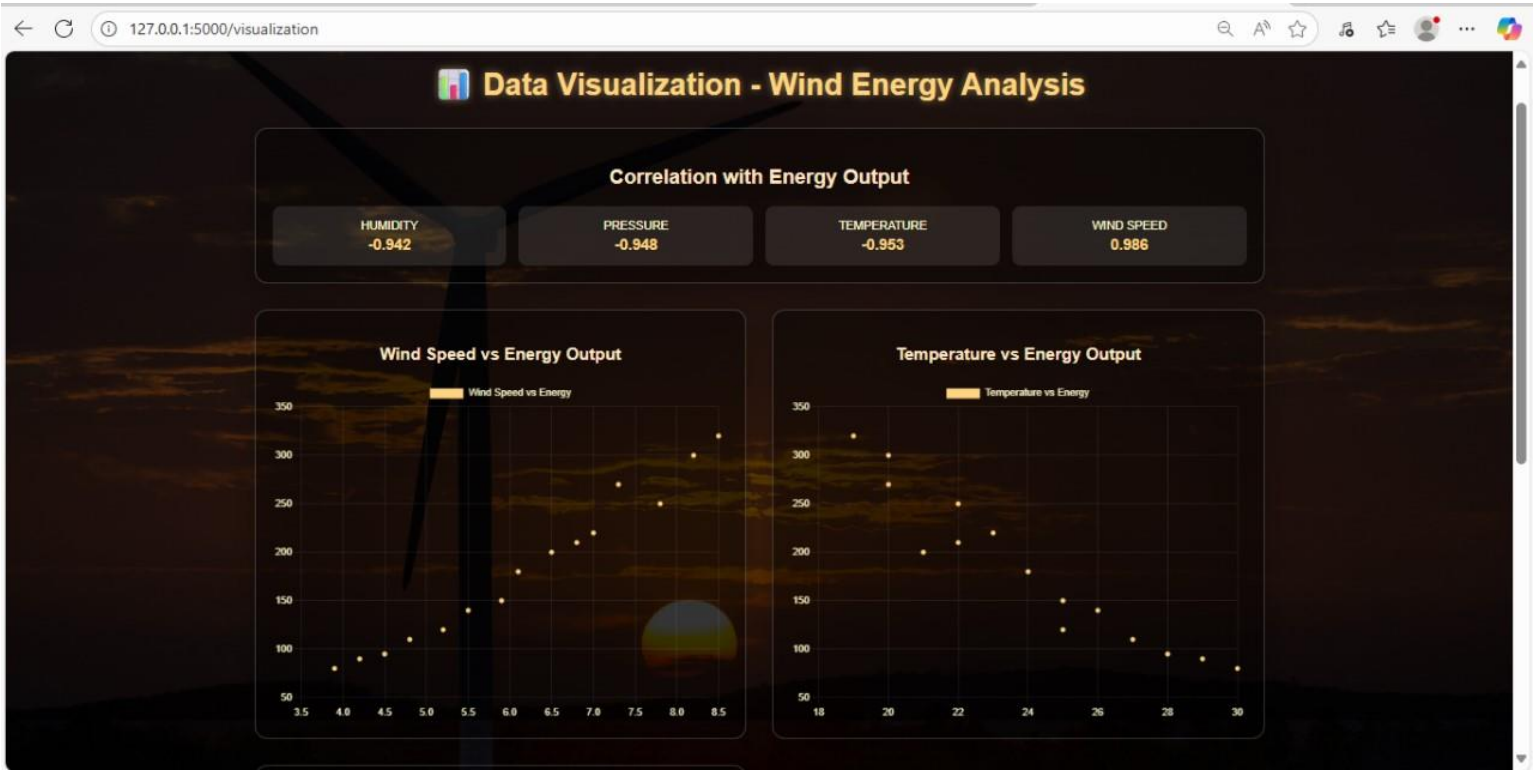
Explanation of the Structure:

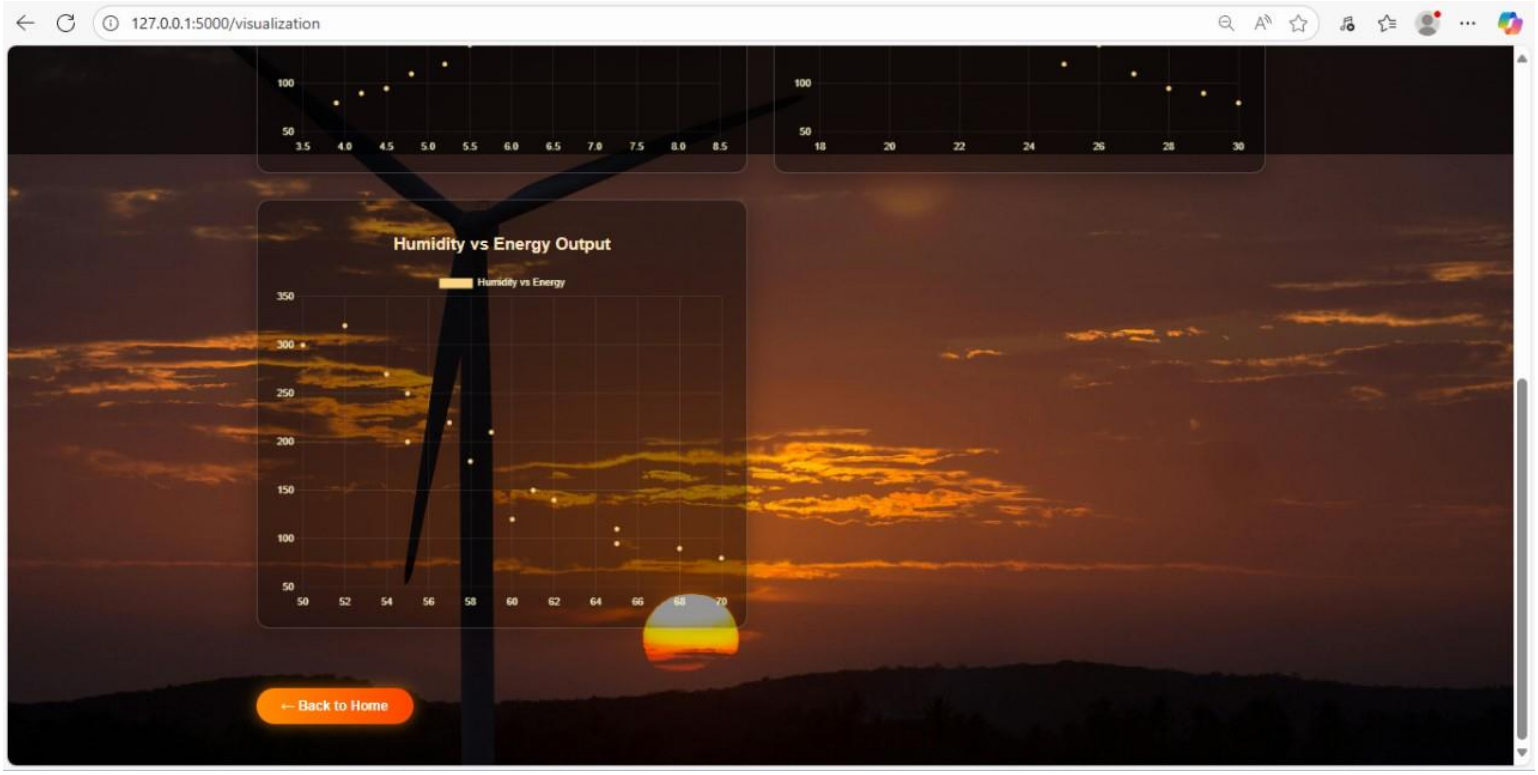
- **app.py:** The core Flask app handling API requests, model predictions, and routing.
- **wind_energy_model.pkl:** The serialized trained model ready for deployment.
- **requirements.txt:** Specifies all Python libraries (Flask, pandas, scikit-learn, etc.) for easy environment setup.
- **templates/**: Stores HTML files used by Flask for rendering web pages.
- **static/**: Holds all static assets like CSS, JS, and images to make the UI interactive and visually appealing.
- **data/**: Contains the dataset(s) used for training and testing the machine learning model.
- **notebooks/**: Jupyter notebooks for experimentation, analysis, and model development.
- **utils/**: Optional folder for reusable functions or scripts to clean, preprocess, or transform data.
- **README.md:** Provides an overview of the project, instructions, and notes for users or developers.

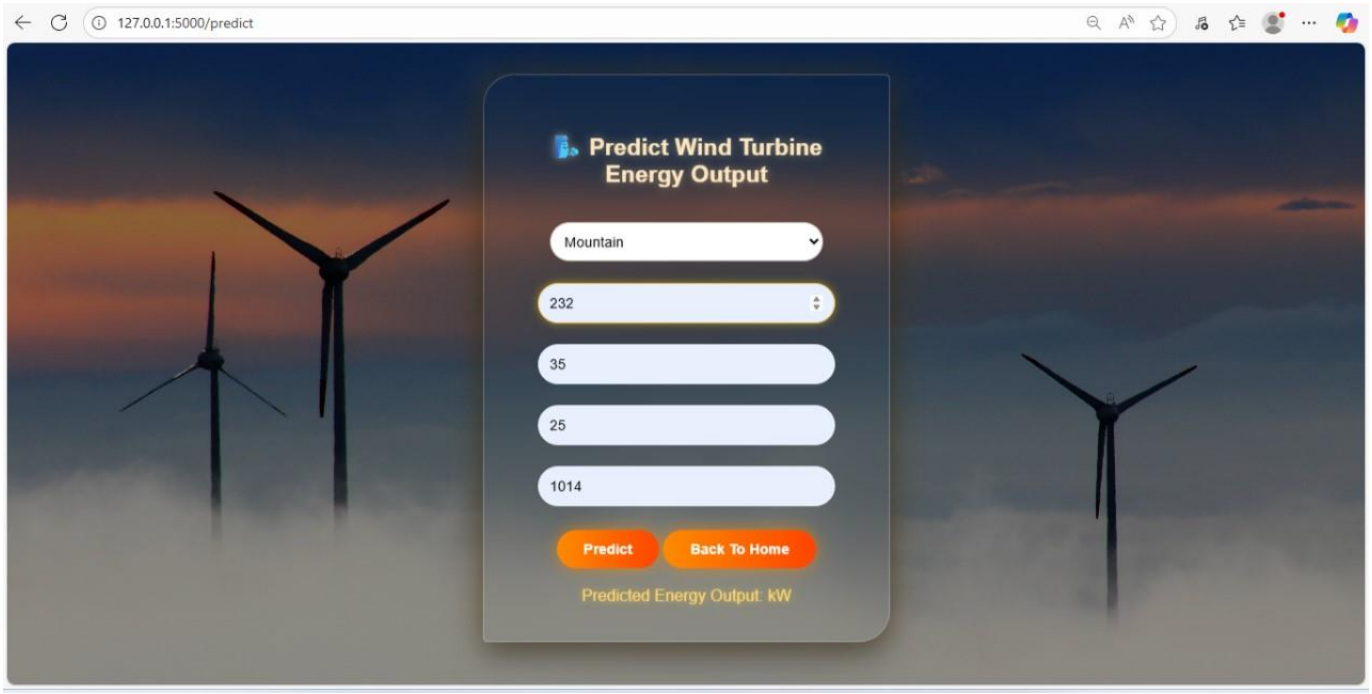
Execute and test your model

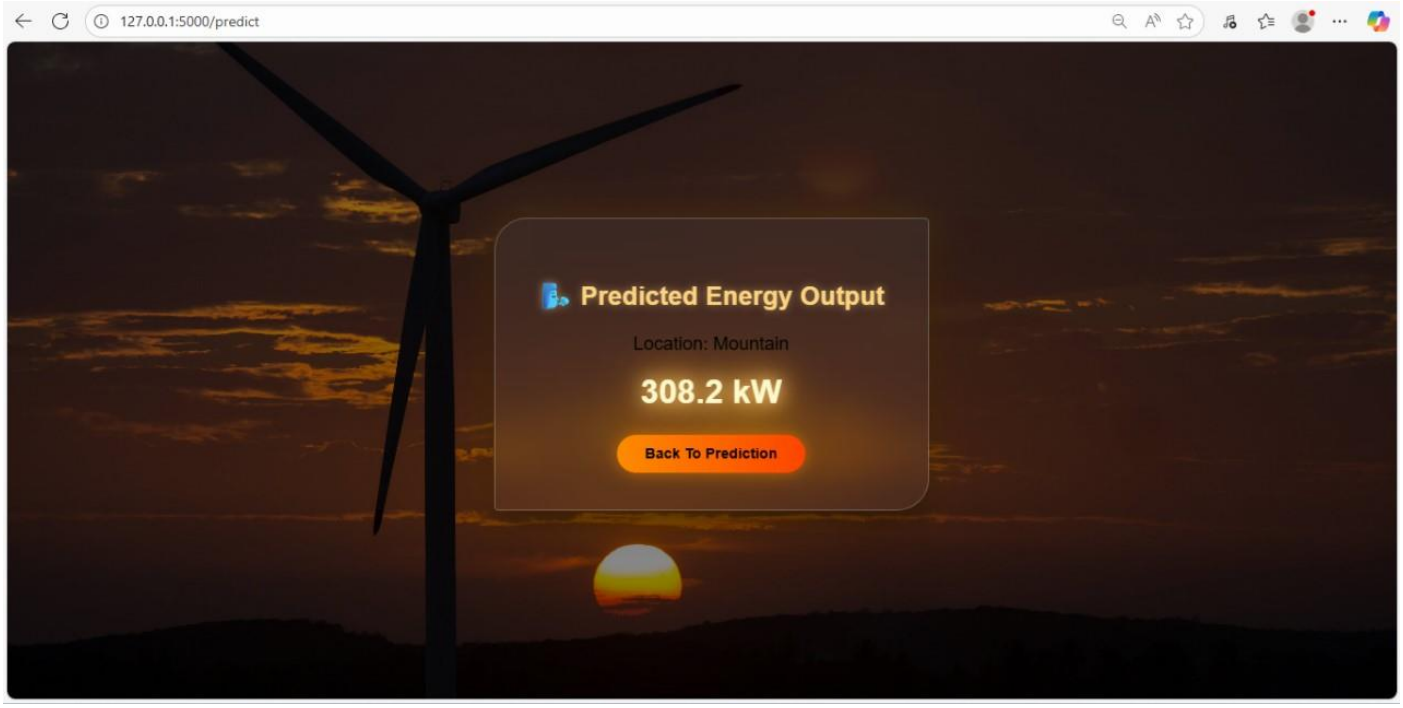
Execute the python code and after the model is running, open localhost:5000 page and test it.




127.0.0.1:5000/visualization
Data Visualization - Wind Energy Analysis
Correlation with Energy Output
HUMIDITY
-0.942
PRESSURE
-0.948
TEMPERATURE
-0.953
WIND SPEED
0.986
Wind Speed vs Energy Output
Wind Speed vs Energy
Temperature vs Energy Output
Temperature vs Energy


127.0.0.1:5000/visualization
Humidity vs Energy Output
Humidity vs Energy
← Back to Home





Conclusion

The Wind Energy Prediction project demonstrates a complete end-to-end approach to predicting wind turbine energy output using machine learning techniques and real-time weather data. Through this project, we were able to integrate multiple components of a typical data science workflow, including data collection, preprocessing, exploratory analysis, model building, evaluation, and

deployment through a web application. The use of historical datasets containing environmental parameters such as wind speed, wind direction, temperature, humidity, and theoretical power allowed us to develop a robust model capable of understanding the relationship between these features and the actual power generated. Exploratory data analysis and correlation studies provided insight into which features had the most influence on energy output, allowing us to focus on the most impactful variables, such as wind speed and theoretical power, while reducing redundancy and noise in the dataset.

The project employed the Random Forest Regressor, a powerful ensemble learning algorithm, to model the non-linear relationships inherent in wind energy data. The model was trained on a substantial portion of the dataset and rigorously tested on a separate test set to ensure generalization. Performance metrics, including the R^2 score, Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE), indicated that the model was able to predict wind energy output with high accuracy. These evaluations confirmed that the Random Forest algorithm could effectively capture the complexities of the dataset and provide reliable predictions under varying conditions. The trained model was then saved using the pickle module, allowing for seamless reuse and integration into the web application without retraining, which enhances efficiency and practicality for real-world applications.

Integration with the OpenWeather API enabled the system to retrieve real-time weather data for any specified city, ensuring that predictions could be dynamically generated based on current conditions. The Flask-based web application served as the interface between users, the API, and the model, allowing for both automated data retrieval and manual input of environmental parameters. Users can enter a city to obtain current weather conditions or provide specific values for wind speed and theoretical power to receive predicted energy output in real time. The HTML front-end, combined with Flask routing, created a user-friendly and interactive platform that is accessible even to individuals without technical expertise.

Overall, this project provides a practical demonstration of how machine learning can be applied to renewable energy management. It highlights the importance of data-driven decision-making in optimizing wind turbine operations and planning energy distribution. By leveraging historical data, real-time weather information, and an accurate predictive model, the system can assist energy companies, grid operators, and researchers in improving efficiency and reliability in wind energy generation. Beyond its immediate application, the project lays a foundation for future enhancements, such as incorporating additional environmental factors, using more advanced predictive models like deep learning networks, or deploying the application on cloud platforms for broader accessibility. In conclusion, the Wind Energy Prediction project successfully combines data science, machine learning, and web development to deliver a fully functional, real-time, and accurate predictive system for renewable energy management, demonstrating the potential of AI-driven solutions in the sustainable energy sector.

Future Scope of Wind Energy Prediction Project

Integration with IoT Devices: Connect wind turbines with IoT sensors for real-time data collection, improving prediction accuracy.

Advanced Machine Learning Models: Explore deep learning models like LSTM or GRU for capturing temporal patterns in wind energy data.

Inclusion of Additional Weather Parameters: Incorporate factors such as air density, precipitation, and solar radiation for more comprehensive predictions.

Forecasting for Multiple Time Horizons: Extend the system to predict energy output for short-term (hourly), medium-term (daily), and long-term (weekly/monthly) horizons.

Cloud-Based Deployment: Host the web application on cloud platforms like AWS, Azure, or Google Cloud for scalability and global access.

Mobile Application Integration: Develop mobile apps to allow operators and users to access predictions on-the-go.

Grid Optimization: Use predictions to assist in energy grid management, load balancing, and optimal energy distribution planning.

Predictive Maintenance: Combine predictions with turbine condition monitoring to anticipate maintenance needs and prevent downtime.

Multi-Turbine Farm Analysis: Extend the model to handle data from multiple wind farms simultaneously for aggregated energy forecasting.

Integration with Renewable Energy Mix: Combine wind energy predictions with solar, hydro, and other renewable sources for holistic energy management and planning.

References

1. Open WeatherMap API Documentation – *Real-time Weather Data for Developers.* Available at: https://openweathermap.org/api
2. Kaggle Datasets – *Wind Turbine Power and Weather Data.* Available at: https://www.kaggle.com/
3. Scikit-learn Documentation – *Machine Learning in Python.* Available at: https://scikit-learn.org/stable/
4. Flask Framework Documentation – *Lightweight WSGI Web Application Framework.* Available at: https://flask.palletsprojects.com/
5. Pandas Documentation – *Data Analysis and Manipulation Tool.* Available at: https://pandas.pydata.org/docs/

GitHUB Link : https://github.com/Lakshmi-Swarupa/wind-turbine-energy