**Weather-Based Prediction of Wind Turbine Energy**

**1. Introduction**

* **Project Title: Wind Turbine Energy Prediction**
* Renewable energy plays a critical role in addressing global energy demands while reducing environmental impact. Among various renewable sources, wind energy has emerged as one of the most sustainable and rapidly growing energy solutions worldwide. However, the power output of wind turbines is highly dependent on weather conditions such as wind speed, temperature, atmospheric pressure, and air density. Due to this variability, accurately predicting wind turbine energy generation is a challenging but essential task for efficient grid management and energy planning.
* The **Wind Turbine Energy Prediction** project aims to develop a machine learning–based system capable of forecasting wind turbine power output using historical turbine data and real-time weather inputs. By analyzing patterns in wind speed and theoretical power data, the system trains a predictive model to estimate active power generation under varying environmental conditions.
* This project leverages data preprocessing techniques, a Random Forest regression algorithm for prediction, and a Flask-based web application for deployment. Additionally, integration with the OpenWeather API enables real-time weather data retrieval, making the system dynamic and practical for real-world use cases.
* The proposed solution demonstrates how data science and software engineering can be combined to build an intelligent, scalable, and user-friendly energy forecasting system. Such predictive systems can assist energy companies, wind farm operators, and grid managers in optimizing production, reducing uncertainty, and improving renewable energy utilization.

## Team Structure

**Team Leader : Ashok Kumar Uppala**

**Team Member : Sai Krishna Lanka**

**Team Member: Golagani Govindu**

**Team Member: Anand Babu Arthamudi**

**2. Project Overview**

* **Purpose:**The purpose of this project is to build a machine learning–based system that predicts wind turbine energy output using historical turbine data and live weather inputs.
* **Objective**
* The objective of this project is to develop a machine learning–based system that predicts wind turbine power output using historical turbine data and real-time weather inputs. The system helps in forecasting energy production accurately, which supports efficient energy management and grid planning.
* **Project Description**
* Wind turbine energy generation is highly influenced by environmental factors such as wind speed, temperature, humidity, and atmospheric pressure. Since these factors vary continuously, predicting power output manually is difficult.
* This project uses:
* Historical turbine dataset (T1.csv)
* Weather data from OpenWeather API
* Machine Learning (Random Forest Regression)
* Flask-based web application for deployment
* The system processes input weather conditions and predicts the expected active power output of the wind turbine.

**Features**

**Data Preprocessing**

* Cleaning missing values
* Removing noise and outliers
* Selecting important features

**Machine Learning Model**

* Algorithm used: Random Forest Regressor
* Trained on historical wind turbine dataset
* Model saved using Joblib

**Web-Based Dashboard**

* Developed using Flask
* User-friendly interface
* Displays weather data and prediction results

**Real-Time Weather Integration**

* Uses OpenWeather API
* Fetches live weather conditions
* Automatically updates prediction inputs

**Data Visualization**

* Scatter plot of Actual vs Predicted Power
* Line charts for wind speed trends
* Performance analysis graphs

**3. Architecture**

**System Architecture**

The system architecture of the Wind Turbine Energy Prediction project is designed using a modular approach. It consists of four main components:

1. Data Layer
2. Machine Learning Layer
3. Backend Layer
4. Frontend Layer

Each component works together to ensure accurate prediction and smooth user interaction.

**1. Data Layer**

**Dataset**

* Historical wind turbine dataset (T1.csv)
* Contains wind speed, theoretical power, and active power

**Real-Time Weather Data**

* Integrated using OpenWeather API
* **Fetches:**
  + Temperature
  + Humidity
  + Pressure
  + Wind Speed

This data is used as input for prediction.

**2. Machine Learning Layer**

**Data Preprocessing**

* Handling missing values
* Removing outliers
* Feature selection

**Model Training**

* Algorithm: Random Forest Regressor
* Trained using historical data
* Evaluated using R² score and error metrics

**Model Storage**

* Model saved as .sav file using Joblib
* Loaded during runtime for predictions

**3. Backend Layer (Flask Application)**

The backend is built using Python Flask.

**Responsibilities:**

* Handle user requests
* Fetch weather data
* Load trained ML model
* Process inputs
* Return predicted power output

**API Endpoints:**

* **/**predict → Returns predicted active power
* /weather → Returns weather details

**4. Frontend Layer**

**Developed using:**

* HTML
* CSS
* JavaScript
* Flask templates

**Dashboard Includes:**

* Weather information display
* Power prediction module
* Visualization graphs

**Data Flow Explanation (Step-by-Step)**

* User enters city name or wind parameters
* Flask calls OpenWeather API
* Weather data is processed
* Trained Random Forest model predicts power output
* Prediction is displayed on dashboard

**Architecture Type**

**This project follows a:**

* Client–Server Architecture
* ML Model Deployment Architecture
* REST API-based Communication

**Frontend:**  
Flask templates (HTML, CSS, JavaScript) used for UI design and dashboard visualization.

* Backend:  
  Python Flask application handling API requests, ML model predictions, and weather data integration.
* Database:  
  Local CSV dataset (T1.csv) for training and testing. Model stored as .sav file using Joblib. Future scope includes cloud database integration (MongoDB Atlas / AWS RDS).

1. **Setup Instructions**

**1. System Requirements**

Before starting, ensure your system meets the following requirements:

**Software Requirements**

* Python 3.9 or higher
* pip (Python package manager)
* Internet connection (for weather API)

**Python Libraries Required**

* Flask
* Pandas
* NumPy
* Scikit-learn
* Matplotlib
* Joblib
* Requests

**2. Install Python**

**If Python is not installed:**

1. Download Python from the official website:  
   👉 <https://www.python.org/downloads/>
2. Install it and make sure to check:  
   ✔ “Add Python to PATH”
3. Verify installation:

python --version

**3. Clone or Download the Project**

If using Git:

git clone <repository-url>

cd wind-turbine-project

Or download the ZIP file and extract it.

**4. Create Virtual Environment (Recommended)**

Creating a virtual environment keeps dependencies organized.

python -m venv venv

Activate it:

Windows:

venv\Scripts\activate

Mac/Linux:

source venv/bin/activate

**5. Install Required Dependencies**

If you have a requirements.txt file:

pip install -r requirements.txt

If not, install manually:

pip install flask pandas numpy scikit-learn matplotlib joblib requests

Verify installation:

pip list

**6. Setup OpenWeather API Key**

This project integrates real-time weather data using the OpenWeather API.

Steps to get API Key:

1. Visit: <https://openweathermap.org/>
2. Create an account.
3. Generate a free API key**.**

**Add API Key in Project**

Option 1: Add directly inside app.py:

API\_KEY = "your\_api\_key\_here"

Option 2 (Recommended): Use environment variable.

**Windows:**

setx OPENWEATHER\_API\_KEY "your\_api\_key\_here"

**Mac/Linux:**

export OPENWEATHER\_API\_KEY="your\_api\_key\_here"

Then access in Python:

import os

API\_KEY = os.getenv("OPENWEATHER\_API\_KEY")

**7. Train the Model (If Needed)**

If the .sav model file is not available:

**Run:**

python model\_training.py

**This will:**

* Load dataset (T1.csv)
* Train Random Forest model
* Save model as power\_prediction.sav

**8. Run the Flask Application**

Start the backend server:

python app.py

If successful, you will see:

Running on http://127.0.0.1:5000/

**9. Access the Application**

Open your browser and go to:

http://127.0.0.1:5000/

**You will see:**

* Intro page
* Dashboard
* Weather module
* Prediction results

**10. Testing the APIs**

Test Prediction API

Use Postman or browser:

POST /predict

**Parameters:**

* WindSpeed
* TheoreticalPower

Test Weather API

GET /weather?city=Hyderabad

**11. Stopping the Server**

**Press:** CTRL + C

* + Python 3.9+
  + Flask
  + Pandas, NumPy, Scikit-learn, Matplotlib
  + Joblib
  + OpenWeather API key

1. **Folder Structure**

**Main Project Folder**

This is the main directory that contains all project files.

wind-turbine-project/

**Inside this folder, we have the following files and folders:**

**1. app.py**

* This is the main file of the project.
* It runs the Flask server.
* It connects the frontend and backend.
* It loads the trained machine learning model.
* It handles API routes like:
  + /predict
  + /weather

**When we run:**

python app.py

The project starts from this file.

**2. model\_training.py**

* This file is used to train the machine learning model.
* It:
  + Loads dataset (T1.csv)
  + Cleans and preprocesses data
  + Trains Random Forest model
  + Saves the model as .sav file

We run this file only once to train the model.

**3. power\_prediction.sav**

* This is the saved trained model.
* It is created after training.
* Flask loads this file to make predictions.

**4. requirements.txt**

* Contains list of Python libraries used in the project.
* Example:
  + Flask
  + Pandas
  + NumPy
  + Scikit-learn
  + Joblib

Used to install dependencies easily:

pip install -r requirements.txt

**5. data/ Folder**

This folder stores dataset files.

data/

└── T1.csv

T1.csv

* Historical wind turbine dataset.
* Contains:
  + Wind Speed
  + Theoretical Power
  + Active Power

This dataset is used for training the model.

**6. templates/ Folder**

This folder contains HTML files.

Flask automatically reads HTML files from this folder.

templates/

├── index.html

└── dashboard.html

index.html

* Intro page of the project.

dashboard.html

* Displays:
  + Weather data
  + Predicted power output
  + Graphs

**7. static/ Folder**

This folder stores frontend resources.

static/

├── css/

├── js/

└── images/

css/

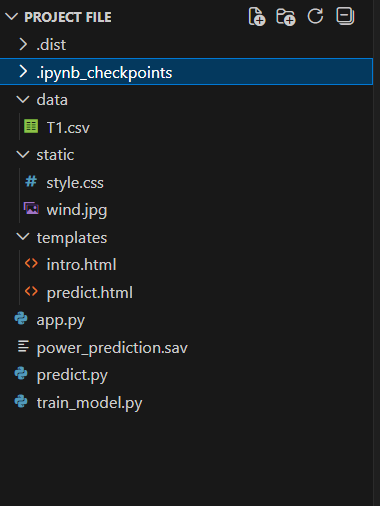
* Contains styling files.

js/

* Contains JavaScript files.

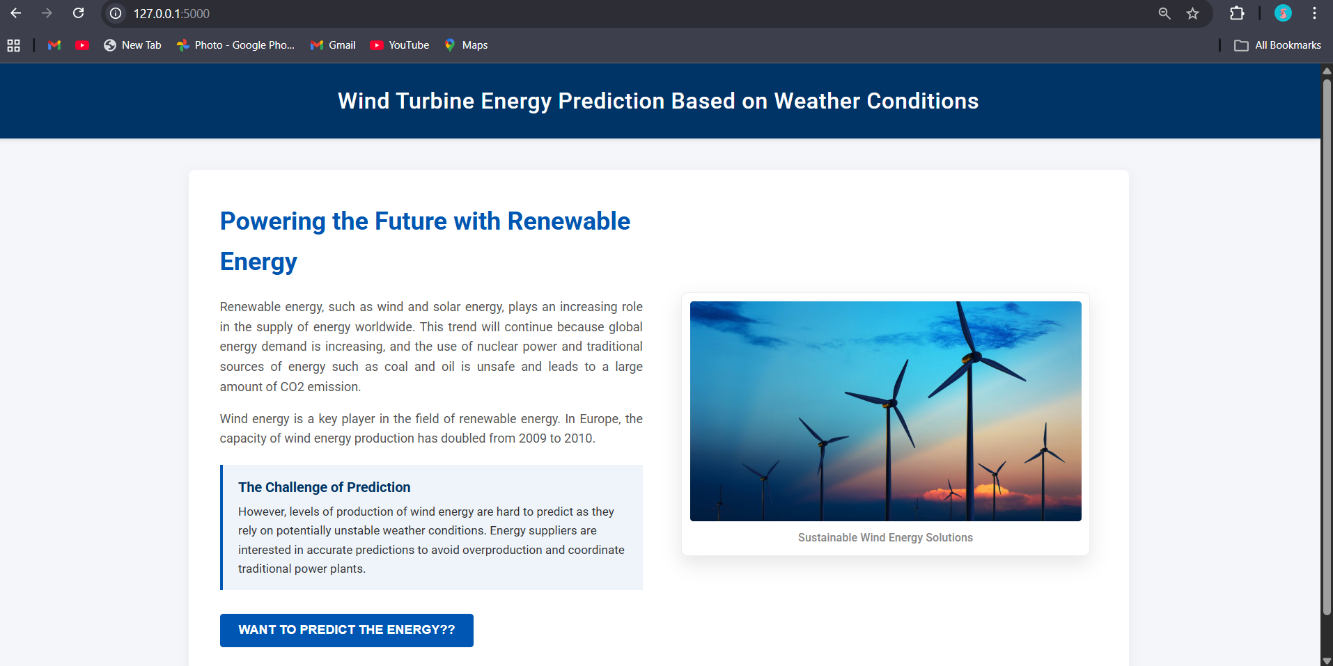
images/

* Stores project images and charts.



**6. Running the Application**

* Frontend: Runs automatically via Flask templates.
* Backend: Start with python app.py.
* Access at <http://127.0.0.1:5000/>.



1. **API Documentation**

* Endpoint 1: /predict
  + Method: POST
  + Parameters: TheoreticalPower, WindSpeed
  + Response: Predicted Active Power
* Endpoint 2: /weather
  + Method: GET
  + Parameters: City name
  + Response: Weather data (temperature, humidity, pressure, wind speed)

1. **Authentication**

**Current Authentication Status**

In the current version of the Wind Turbine Energy Prediction system:

* The application is open access
* No login or user authentication is required
* Anyone who runs the application locally can access it

However, the system uses an API key authentication mechanism for accessing real-time weather data.

**OpenWeather API Key Authentication**

The system integrates real-time weather data using:

OpenWeather

To access weather data, an API key is required.

**How It Works**

* User enters city name
* Flask backend sends request to OpenWeather
* API key is included in the request
* OpenWeather verifies API key.
* If valid → Weather data is returned
* If invalid → Error message is generated

1. **User Interface**

The User Interface (UI) of the Wind Turbine Energy Prediction system is designed to be simple, interactive, and user-friendly. It allows users to easily input data, view weather information, and see predicted power output.

The UI is developed using:

* HTML
* CSS
* JavaScript
* Flask Templates

**1. Intro Page (index.html)**

**Purpose:**

* Provides project overview
* Explains objective and features
* Contains navigation button to dashboard

**Features:**

* Project title
* Short description
* “Go to Dashboard” button

**2. Dashboard Page (dashboard.html)**

This is the main working interface of the system.

**A. Weather Data Section**

* User enters city name
* Displays:
  + Temperature
  + Humidity
  + Pressure
  + Wind Speed

Data is fetched using OpenWeather API.

**B. Prediction Module**

* Takes input:
  + Wind Speed
  + Theoretical Power
* When user clicks “Predict”
* Displays:
  + Predicted Active Power Output

**C. Visualization Section**

Includes:

* Scatter plot (Actual vs Predicted)
* Line chart (Wind Speed vs Power Output)
* Performance graphs

These visualizations help users understand model accuracy.

**UI Design Characteristics**

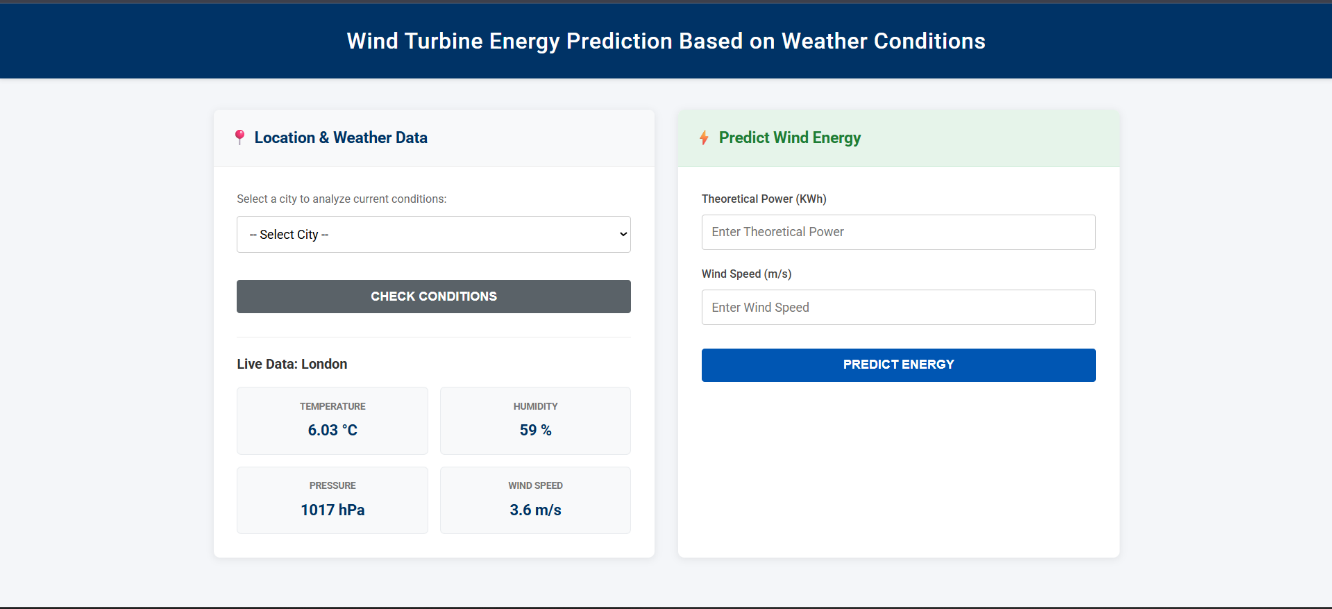
* Simple layout
* Responsive design
* Clean color scheme
* Clear buttons and input fields
* Error messages for invalid inputs.

**User Flow**

* User opens homepage
* Clicks “Go to Dashboard”
* Enters city name
* Weather data appears
* User clicks predict
* Power output result is displayed

**Technologies Used in UI**

| **Technology** | **Purpose** |
| --- | --- |
| HTML | Structure of page |
| CSS | Styling |
| JavaScript | Interactivity |
| Flask Templates | Connect backend with frontend |



1. **Testing**

Testing is an important part of our Wind Turbine Energy Prediction project. It ensures that the system works correctly, accurately predicts results, and handles errors properly.

We performed different types of testing:

**Unit Testing (Machine Learning Model Testing)**

**Purpose:**

To verify that the Random Forest model predicts accurate power output.

What We Tested:

* Model prediction accuracy
* R² score
* Mean Absolute Error (MAE)
* Mean Squared Error (MSE)

**Process:**

* Split dataset into training and testing sets
* Train the model
* Test using unseen data
* Compare actual vs predicted values

**Result:**

* Model produced high R² score
* Predictions closely matched actual power values

**API Testing**

**Purpose:**

To check whether APIs are working correctly.

**A. Prediction API (/predict)**

**Test Cases:**

| **Test Case** | **Input** | **Expected Result** |
| --- | --- | --- |
| Valid input | WindSpeed = 8 | Predicted power shown |
| Missing input | No WindSpeed | Error message |
| Invalid data type | WindSpeed = "abc" | Validation error |

**B. Weather API (/weather)**

**Test Cases:**

| **Test Case** | **Input** | **Expected Result** |
| --- | --- | --- |
| Valid city | Hyderabad | Weather details shown |
| Invalid city | xyz123 | API error message |
| Empty input | Blank | Validation error |

**Functional Testing**

**Purpose:**

To ensure complete system works properly.

**We tested:**

* Navigation between pages
* Form input validation
* Display of prediction results
* Display of weather data
* Graph generation

**User Interface Testing**

**Checked:**

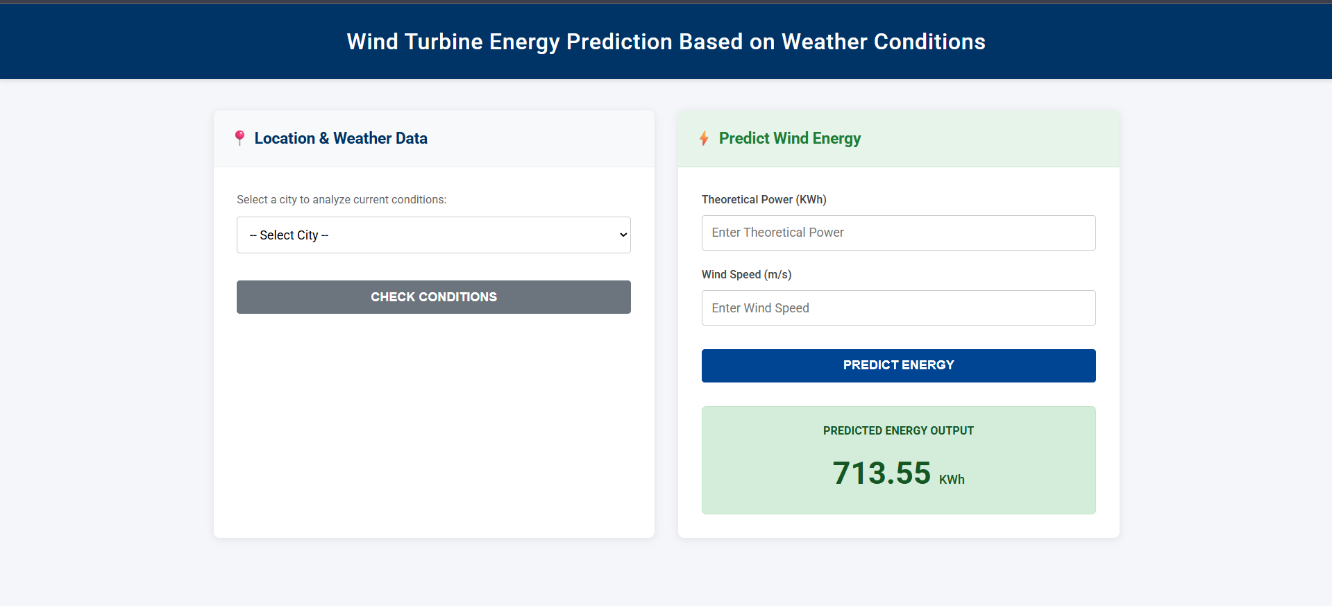
* Button functionality
* Proper display of results
* Responsive design
* Error message display

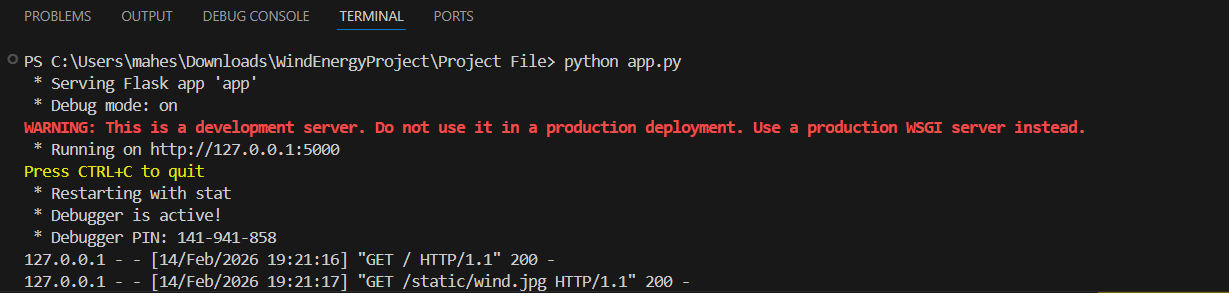
**Integration Testing**

**Purpose:**To verify that all components work together correctly.

Tested integration between:

* Flask backend
* Machine learning model
* Weather API
* Frontend dashboard
* Unit testing for ML model predictions.
* API testing for weather data retrieval.
* Functional testing for input validation and dashboard navigation.



****

**11. Screenshots or Demo**

* **Scatter plot of actual vs predicted power.**
* **Dashboard screenshot showing weather + prediction results.**
* **Line chart trends of wind speed vs power output.**

**12. Known Issues**

Although the Wind Turbine Energy Prediction system works correctly, there are some limitations and known issues in the current version.

**Limited Dataset Size**

* The model is trained on a limited historical dataset (T1.csv).
* Smaller dataset may reduce generalization ability.
* Model performance may decrease for unseen extreme weather conditions.

**Improvement:**

Use larger and multi-year wind turbine datasets.

**No User Authentication**

* Currently, the dashboard is open access.
* No login system implemented.
* APIs can be accessed without user verification.

**Improvement:**

Implement JWT-based authentication for secure access.

**Dependency on Weather API**

The system depends on real-time weather data from:

OpenWeather

Possible issues:

* Invalid API key
* API request limits (free plan restrictions)
* Internet connectivity problems
* Incorrect city name input

**Improvement:**

Add better error handling and backup data source.

**No Cloud Deployment**

* Application currently runs on local server.
* Not deployed on cloud.
* Cannot be accessed globally.

**Improvement:**

Deploy using:

* AWS
* Azure
* Google Cloud

**No Advanced Time-Series Forecasting**

* Current model predicts based on given inputs.
* Does not predict future power output over time.

**Improvement:**

Use LSTM or ARIMA for time-series forecasting.

**Basic Error Handling**

* Basic input validation implemented.
* Advanced validation and logging not implemented.

**Improvement:**

Add:

* Logging system
* Centralized error handling
* Monitoring tools

**13. Future Enhancements**

Although the Wind Turbine Energy Prediction system works successfully, there are several improvements that can make it more powerful, scalable, and industry-ready.

**Cloud Deployment**

Currently, the application runs on a local server.  
In future, it can be deployed on cloud platforms such as:

* Amazon Web Services
* Microsoft Azure
* Google Cloud Platform

**Benefits:**

* Global accessibility
* Better scalability
* High availability
* Real-time production usage

**JWT-Based Authentication**

At present, there is no login system.

Future improvement:

* Implement JWT (JSON Web Token) authentication
* Add user login & registration
* Role-based access (Admin/User)

**Benefits:**

* Secure API access
* Prevent unauthorized usage
* Production-level security

**Advanced Time-Series Forecasting**

Currently, the model predicts power based on current input only.

Future improvement:

* Use LSTM (Long Short-Term Memory)
* Use ARIMA model
* Predict future energy output (next 24 hours / next 7 days)

**Benefits:**

* Better grid planning
* Advanced forecasting capability
* Industry-level solution

**Larger and Multi-Location Dataset**

Current limitation:

* Dataset is limited to one turbine or small data size.

Future improvement:

* Add multi-year historical data
* Include multiple turbine locations
* Add seasonal variations

**Benefits:**

* Improved accuracy
* Better generalization
* Robust model performance

**IoT Sensor Integration**

Future version can connect directly with:

* Real-time wind sensors
* Turbine monitoring systems

**Benefits:**

* Live automatic predictions
* Smart wind farm management
* Real-time anomaly detection

**Improved Visualization Dashboard**

Add:

* Heatmaps
* Feature importance graph
* Real-time performance monitoring
* Energy trend comparison

This makes the dashboard more interactive and analytical.

**Grid Demand Integration**

Future enhancement:

* Integrate electricity demand APIs
* Automatically balance supply and demand

This helps energy companies optimize grid performance.

**Model Optimization & Comparison**

Future scope:

* Compare Random Forest with:
  + XGBoost
  + Gradient Boosting
  + Neural Networks

Choose best-performing model.

**14. Conclusion**

The **Wind Turbine Energy Prediction project** successfully demonstrates the application of machine learning and data-driven techniques to solve a real-world renewable energy challenge. By leveraging historical turbine data, preprocessing methods, and a **Random Forest regression model**, the project achieves strong predictive accuracy, enabling stakeholders to forecast energy output with confidence.

The integration of a **Flask-based dashboard** and **OpenWeather API** ensures that predictions are not only technically sound but also accessible and user-friendly. Visualizations such as scatter plots, line charts, and correlation heatmaps further enhance interpretability, making the solution practical for energy companies, wind farm operators, and grid managers.

Through structured **Agile sprint planning, backlog management, and performance testing**, the project was executed with clarity and measurable progress. Testing confirmed the robustness of the model, while defect analysis and bug tracking highlighted areas for improvement.

Ultimately, this project provides a scalable foundation for future enhancements, including cloud deployment, advanced authentication, expanded datasets, and integration with grid demand APIs. It stands as a strong example of combining **data science, software engineering, and agile methodology** to deliver a solution that addresses both technical and customer-centric needs in the renewable energy sector.