Project planning phase

1.planning logic:

A. Data Collection Logic

The foundation of the model relies on high-fidelity historical data.

- Source: Historical sensor data from wind farms (e.g., Kaggle, NREL, or UCI Repository).
- Features Collected: * Wind Speed (m/s): The primary driver of turbine rotation.
    - Wind Direction (°): To identify "yaw" alignment and wake effects.
    - Theoretical Power Curve (kWh): The manufacturer's expected output for specific speeds.
    - LV ActivePower (kW): The Target Variable (the actual energy produced).

B. Data Cleaning & Preprocessing

Raw data often contains errors due to sensor icing, maintenance downtime, or transmission gaps.

B.1 Handling Anomalies

- Zero-Power Logic: If Wind Speed > 3.5 m/s (cut-in speed) but Power = 0, the turbine was likely under maintenance. These rows are removed to avoid confusing the model.
- Negative Values: Any negative power readings (caused by sensor noise) are clipped to 0.

B.2 Feature Engineering & Transformation

- Null Value Imputation: Using Linear Interpolation for time-series gaps.
- Label/One-Hot Encoding: Converting categorical wind directions (N, NE, E) into numerical formats for the algorithm.
- Feature Scaling: Using StandardScaler to ensure features like Wind Speed (0-25) and Pressure (~1000) are treated with equal mathematical weight.

2.Project planning:

C.Model Selection

- Random Forest Regressor is chosen due to its ability to:
    - Handle nonlinear relationships between wind speed, temperature, and power output.
    - Reduce overfitting by combining multiple decision trees.

- Work effectively with noisy or real-world environmental data.

D Model Training & Evaluation

- Train the model on historical data.
- Evaluate using metrics like R^2 Score, MSE, RMSE, and MAE.
- Fine-tune hyperparameters to improve prediction accuracy.

E. Deployment

- Save the trained model as a .sav file using Python's pickle module.
- Build a Flask web application to provide a simple interface where users can enter current weather conditions.
- The Flask app loads the trained model, predicts energy output, and displays results instantly.

Above planning logic

1. **Data Collection & Feature Logic**

The predictive power of the model depends on selecting environmental variables that physically impact wind turbine rotation.

- Wind Speed (v): The primary input. The kinetic energy available in the wind is proportional to the cube of the wind speed (v^3).
- Theoretical Power: The manufacturer's "Power Curve" logic. This serves as a benchmark for what the turbine *should* produce under ideal conditions.
- Wind Direction: Logic to account for "Yaw" (the turbine's ability to face the wind). If the turbine cannot align perfectly, efficiency drops.
- Target Logic: The model aims to predict Active Power, which is the actual electricity delivered to the grid after mechanical and electrical losses.

2**. Data Cleaning & Preprocessing Logic**

Raw data from wind farms is often "noisy" due to sensor errors or turbine downtime. The following logic is applied to sanitize the dataset:

A. Outlier Removal (Physics-Based)

- Zero-Generation Logic: If $Wind Speed > 3.5\text{ m/s}$ (the "Cut-in" speed) but $Power = 0$, the turbine was likely stopped for maintenance. These records are removed to prevent the model from learning "false" inefficiencies.
- Negative Value Clipping: Sensors occasionally report negative power due to internal consumption. Logic is applied to clip all values < 0 to 0.

B. **Missing Data Strategy**

- Temporal Interpolation: Since weather is continuous, missing values are filled using Linear Interpolation (estimating the gap based on the value before and after) rather than simple averages.

C. Feature **Transformation**

- Encoding Logic: Categorical wind directions (North, South, East, West) are converted into numerical vectors using One-Hot Encoding so the model treats them as spatial coordinates rather than a ranked list.
- Scaling Logic: We use Standardization ($\frac{x - \mu}{\sigma}$) to ensure that Wind Speed (small range) and Pressure (large range) contribute equally to the weight calculation.

---

3. **Machine Learning Model Logic**

The project utilizes a Supervised Learning approach to map weather inputs to energy outputs.

- Algorithm Logic (Ensemble Learning): We use algorithms like Random Forest or XGBoost. These work by creating multiple "Decision Trees" and averaging their results. This is ideal for wind data because the relationship between wind speed and power is non-linear (it follows a curve, not a straight line).
- Validation Logic: A Train-Test Split (80/20) is used. The model "studies" 80% of the data and is "tested" on 20% to ensure it can predict weather patterns it has never seen before.

---

4. **Integration Logic (Flask & UI)**

This logic bridges the gap between a static script and a functional application.

1. Serialization: The trained logic is "frozen" into a .sav file.
2. Request Pipeline:
    - Input: User enters Wind Speed via the UI.
    - Processing: Flask receives the input, converts it to a float, and passes it to the loaded model.
    - Output: The model calculates the predicted kW and sends it back to the UI.
3. Real-World Scenario Logic: * Forecasting: Logic uses future weather API data.
    - Maintenance: Logic triggers an alert if predicted output is significantly lower than the theoretical power curve.