PowerPulse: Household Energy Usage Forecast

1.Data Understanding and Exploration

- Load and explore household power consumption data from an Excel file using common data analysis libraries.
- pandas as pd: This library is used for data manipulation and analysis. It helps in reading and working with tabular data like Excel or CSV files.
- matplotlib.pyplot as plt: A plotting library used for creating static, animated, and interactive visualizations.
- seaborn as sns: Built on top of matplotlib, seaborn provides a higher-level interface for attractive and informative statistical graphics.
- skew from scipy.stats: This function is used to measure the skewness (asymmetry) of a dataset's distribution.
- The parameter nrows=200 tells pandas to load only the first 200 rows of data, which is helpful for quick testing or performance optimization.
- The initial script prepares the environment for data analysis by importing essential libraries, loading the dataset, and confirming successful import. The dataset contains household power consumption information, and further analysis (e.g., visualization or statistical summaries) would follow using the loaded data in the df DataFrame.

Perform EDA:

- df.info() prints a concise summary of the DataFrame:
  - Number of rows and columns
  - Column names and their data types (e.g., int64, float64, object)
  - Non-null (non-missing) counts
- This is useful for understanding the overall structure and identifying columns with missing data.
- df.head() displays the first 5 rows of the dataset.
- It gives a quick look at sample values in each column, helping verify that the data has been loaded and interpreted correctly.
- df.describe() returns basic statistical details for all numeric columns:
  - Count: Number of non-null values
  - Mean: Average value
  - Std: Standard deviation
  - Min/Max: Minimum and maximum values
  - 25%, 50%, 75%: Percentile values (also known as quartiles)

- These metrics help understand the data distribution and identify outliers or inconsistencies.
- df.isnull() generates a Boolean DataFrame indicating True for missing values.
- .sum() aggregates the total number of missing values per column.
- This helps identify which features may require cleaning or imputation.

1.Correlation Matrix Heatmap

- Purpose: Displays the pairwise correlation between numeric columns.
- df.corr(numeric_only=True): Computes the correlation coefficients (e.g., Pearson correlation) for all numeric columns in the dataset.
- Creates a heatmap using the seaborn library.

Output: A color-coded matrix that helps identify strongly related features, multicollinearity, or independent variables.

2. Histograms of Key Features

- Purpose: Visualizes the distribution of each numeric variable.

- plt.tight_layout(): Adjusts spacing to prevent overlapping titles and axes.

- **Output**: A grid of histograms showing the shape of each feature's distribution (e.g., normal, skewed, bimodal).

3. Data Cleaning and Distribution Analysis

This section prepares the dataset for accurate statistical analysis by handling missing or malformed data and visualizing the distribution of key numeric features. It also calculates skewness to assess data symmetry.

a.Handling Missing Values

- Replace Placeholder Values: Some datasets use '?' to represent missing values. This line replaces such entries with pd.NA (pandas' standard missing value marker).
- Drop Incomplete Rows: df.dropna() removes all rows that contain any missing values, ensuring a clean dataset for numerical analysis.

b. Converting Data Types

- Ensures that the listed columns are correctly interpreted as floating-point numbers (e.g., for plotting and numerical calculations).
- This step is crucial if the dataset was originally read as strings due to invalid or missing entries.

c. Plotting Distributions & Calculating Skewness

- Loop Through Numeric Columns: The script iterates through each numeric column.
- Histogram with KDE
- Skewness:
  - df[col].skew() calculates skewness, indicating the asymmetry of the distribution:
    - 0 = symmetric (normal)
      - 0 = right-skewed (tail on right)
    - < 0 = left-skewed (tail on left)
  - Skewness is displayed in the plot title.
- **Final Presentation**: Each distribution is plotted in its own chart with proper labels and a grid for clarity.

2. Data Preprocessing

This section processes the raw time-series dataset to extract useful time-based features and ensures all numerical fields are in the correct format for further analysis or machine learning tasks.

1.Importing MinMaxScaler

- MinMaxScaler is imported from sklearn.preprocessing.
- Although not applied in this code, it is typically used to scale numeric data into a normalized range (e.g., 0 to 1), which is essential for many machine learning models.

2. Data Cleaning: Handling Missing Entries

- Replace Invalid Entries: Converts '?' symbols (often used as placeholders for missing data) into pd.NA.
- Drop Missing Rows: Removes any rows that contain missing values to avoid errors in downstream processing.

3. Combining Date and Time into a Single Datetime Column

- Concatenates Date and Time columns into a single datetime string.
- pd.to_datetime(...) parses the string into a datetime64 object.
- This unified column is crucial for time-series analysis and plotting.

4. Extracting Time-Based Features

- These lines extract specific components of the Datetime column to create new columns:

  - **Year**: Four-digit year
  - **Month**: 1 to 12

  - **Day**: Day of the month
  - **Hour**: Hour of the day (0–23)
- These features help in identifying seasonal, daily, or hourly trends in energy consumption.

5. Converting Numeric Fields to Float

- Ensures all energy-related measurements are properly treated as float values (not strings).
- Necessary for mathematical operations, statistical analysis, and machine learning algorithms.

Summary

At this stage, the dataset is:

- **Cleaned of invalid entries**
- **Equipped with rich time-based features**
- **Numerically formatted** for analytical or predictive modeling

These steps are foundational for any time-series modeling, anomaly detection, or consumption trend forecasting.

3.Feature Engineering

1. Daily Average Power Usage

- Purpose: Calculates the average power consumption per day.
- df['Datetime'].dt.date: Extracts the date part from the timestamp.
- groupby(...).transform('mean'): Computes the mean Global_active_power for each day and applies it back to all records for that day.
- This feature helps in understanding daily consumption patterns and detecting anomalies like peak days.

2. 60-Minute Rolling Average Power

- Purpose: Smoothens short-term fluctuations by computing a rolling (moving) average of power consumption over 60 time steps (assumed to be minutes).
- min_periods=1: Ensures calculation even when there are fewer than 60 preceding values (e.g., at the start).
- Helps reveal consumption trends and identify spikes or dips in energy use.

Summary

This step enhances the dataset with:

- **Contextual time-based power patterns** (daily average and rolling average)
- **Scaled, machine-learning-ready features** that ensure consistency across models

These engineered features and scaled values are crucial for building predictive models, detecting consumption anomalies, and time-series forecasting.

4.Model Selection and Training

This section builds and evaluates multiple machine learning models to predict household power consumption using selected features. It includes scaling, training, and performance comparison using standard metrics.

Imports core modeling and evaluation components from the scikit-learn library:

- Regression algorithms (Linear, Random Forest, Gradient Boosting, Neural Network)
- Data splitting and normalization tools
- Metrics: Root Mean Squared Error (RMSE) and R^2 score for performance evaluation

Ensures that all relevant features are in float format to avoid errors during model training.

Recalculates a 60-minute rolling average of Global_active_power for use as a feature.

Selects **predictor variables** (X) that are likely to influence Global_active_power (y), the **target variable**.

- 80/20 split: 80% for training, 20% for testing
- random_state=42 ensures reproducibility

Defines four machine learning mdels:

- Linear Regression: Baseline linear model
- Random Forest: Ensemble of decision trees (bagging)
- Gradient Boosting: Sequential tree building (boosting)
- Neural Network: Multi-layer Perceptron with two hidden layers
- Training: Each model is trained on the training data (X_train, y_train)
- Prediction: Models generate predictions on X_test
- Evaluation:
  - RMSE (Root Mean Squared Error): Measures average error magnitude
  - R^2 Score: Measures the proportion of variance explained by the model
  - **MAE**: More interpretable and less sensitive to extreme errors

Summary

This pipeline performs the complete machine learning workflow:

- Feature selection
- Preprocessing (scaling, engineering)
- Model training using **four diverse algorithms**
- Evaluation using **RMSE and R^2**

These results help identify which model best predicts Global_active_power and guide further optimization or deployment.

5.Model Evaluation

This tabular summary provides a clear comparison of each model based on:

- RMSE (penalizes large errors)
- MAE (measures average prediction error)
- R^2 Score (explains variance captured by the model)

From this result:

- Random Forest shows the best performance with lowest RMSE and highest R^2, making it the top candidate for deployment or further tuning.

Best-Performing Model: Random Forest

Lowest RMSE (Root Mean Squared Error): Better at minimizing large errors.

Lowest MAE (Mean Absolute Error): Most accurate on average.

Highest R^2 Score: Explains ~92.8% of the variance in target variable.

Final Recommendation:

Use **Random Forest Regressor for predicting Global_active_power** — it's giving you the most accurate and reliable results based on your evaluation.