**APPLIED DATA SCIENCE – PHASE 4**

**ELECTRICITY PRICE PREDICTION**

**FEATURE ENGINEERING :**

Feature engineering is the process of creating new features or transforming existing ones to improve the performance of machine learning models. In the context of electricity price prediction, feature engineering can be critical for capturing relevant patterns and improving prediction accuracy.

Here are some feature engineering techniques you can apply to your dataset:

* **Lag Features :** Create lag features for the target variable (electricity price) to incorporate past values. For example, you can add lagged prices from previous hours or days.
* **Rolling Statistics :** Calculate rolling statistics, such as rolling mean or rolling standard deviation, for features like electricity demand or temperature. This can help capture short-term trends.
* **Time-Related Features :** Extract time-related information from timestamps, such as day of the week, hour of the day, or holidays. These features can help model daily or weekly patterns.
* **Seasonal Features :** Identify and create features that capture seasonal patterns in electricity prices, like summer/winter indicators or seasonal dummies.
* **Weather Data Integration :** Integrate weather data into your dataset. Features like temperature, wind speed, or precipitation can significantly impact electricity prices.
* **Time Series Decomposition :** Use time series decomposition methods (e.g., seasonal decomposition of time series - STL) to separate your data into trend, seasonality, and residual components.
* **Moving Averages :** Compute moving averages to capture trends and reduce noise in the data.
* **Feature Scaling and Normalization : Scale and normalize numerical features to ensure all features have a similar impact on the model.**
* **Interaction Terms :** Create interaction terms between relevant features to capture complex relationships.
* **Domain-Specific Features :** Leverage domain knowledge to engineer features specific to the electricity market, like energy source availability or market sentiment indicators.
* **Cross-Correlations :** Calculate cross-correlations between electricity prices and external factors like gas prices, which can reveal interesting relationships.
* **Outlier Handling :** Identify and handle outliers in your dataset, which can distort predictions.
* **Aggregated Features :** Create aggregated features based on historical data, like rolling averages over the past week or month.
* **Sentiment Analysis :** If you have access to news data, perform sentiment analysis to incorporate the market sentiment's effect on prices.
* **Principal Component Analysis (PCA) :** Use PCA for dimensionality reduction if your dataset has many correlated features.

**MODEL TRAINING AND EVALUTION :**

**Data Preprocessing :** Before training models, clean and preprocess your data as discussed in previous responses. Handle missing values, encode categorical variables, and perform feature engineering.

**Data Splitting :** Split your dataset into training, validation, and test sets. Common splits are 70/15/15 or 80/10/10, depending on the dataset size and characteristics.

**Model Selection :** Choose the appropriate machine learning models based on your task. Common choices for regression tasks like electricity price prediction include Linear Regression, Random Forest, Decision Tree, and others.

**Hyperparameter Tuning :** Use techniques like grid search, random search, or Bayesian optimization to fine-tune the hyperparameters of your chosen models. This can significantly impact model performance.

**Model Training :** Train multiple models using the training dataset.

**Model Evaluation :** Evaluate the models using appropriate evaluation metrics. For regression tasks, commonly used metrics include Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared (R^2).

**Cross-Validation :** Perform cross-validation to assess model performance more robustly. Common techniques include k-fold cross-validation.

**CODE :**

#importing required libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error

from sklearn.ensemble import RandomForestRegressor

from sklearn.tree import DecisionTreeRegressor

from sklearn.linear\_model import LinearRegression

from sklearn.neighbors import KNeighborsRegressor

# Load the dataset

df=pd.read\_csv("Electricity.csv", low\_memory=False)

df.head()

df.info()

# Remove rows with missing values denoted by "?"

data=df[['ForecastWindProduction',

       'SystemLoadEA', 'SMPEA', 'ORKTemperature', 'ORKWindspeed',

       'CO2Intensity', 'ActualWindProduction', 'SystemLoadEP2', 'SMPEP2']]

data.isin(['?']).any()

for col in data.columns:

    data.drop(data.index[data[col] == '?'], inplace=True)

data=data.apply(pd.to\_numeric)

data=data.reset\_index()

data.drop('index', axis=1, inplace=True)

data.info()

# Convert data to numeric

data = data.apply(pd.to\_numeric)

data.corrwith(data['SMPEP2']).abs().sort\_values(ascending=False)

X=data.drop('SMPEP2', axis=1)

y=data['SMPEP2']

# Feature Engineering

# Add lag features (e.g., lag of SMPEP2 for 24 hours)

data['SMPEP2\_lag\_24'] = data['SMPEP2'].shift(24)

#To Machine Learning

# Data Splitting

x\_train, x\_test, y\_train, y\_test=train\_test\_split(X,y, test\_size=0.2, random\_state=42)

# Model Training and Evaluation

#LinearRegression

linear\_model=LinearRegression()

linear\_model.fit(x\_train, y\_train)

linear\_predict=linear\_model.predict(x\_test)

np.sqrt(mean\_squared\_error(y\_test, linear\_predict))

#RandomForestRegressor

# Model Training and Evaluation

forest\_model=RandomForestRegressor()

forest\_model.fit(x\_train, y\_train)

forest\_predict=forest\_model.predict(x\_test)

print(np.sqrt(mean\_squared\_error(y\_test, forest\_predict)))

#DecisionTreeRegressor

# Model Training and Evaluation

tree\_model=DecisionTreeRegressor(max\_depth=50)

tree\_model.fit(x\_train, y\_train)

tree\_predict=tree\_model.predict(x\_test)

print(np.sqrt(mean\_squared\_error(y\_test, tree\_predict)))

#KNeighborsRegressor

# Model Training and Evaluation

knn\_model=KNeighborsRegressor()

knn\_model.fit(x\_train, y\_train)

knn\_predict=knn\_model.predict(x\_test)

print(np.sqrt(mean\_squared\_error(y\_test, knn\_predict)))

#Let's see some sample prediction and difference between label and prediction

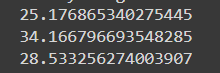
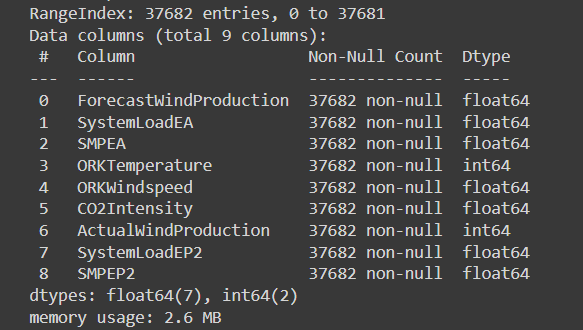
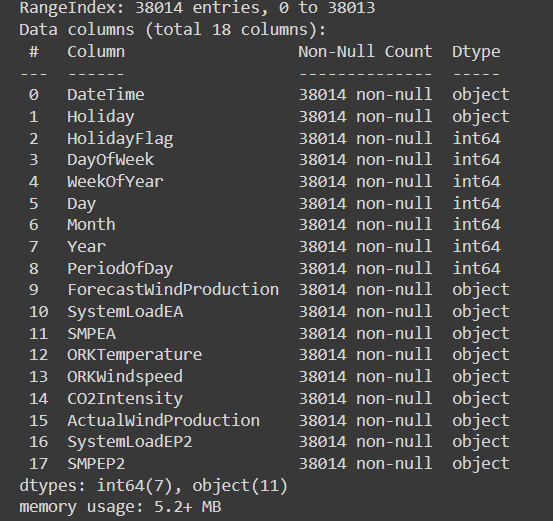
some\_data=x\_test.iloc[50:60]

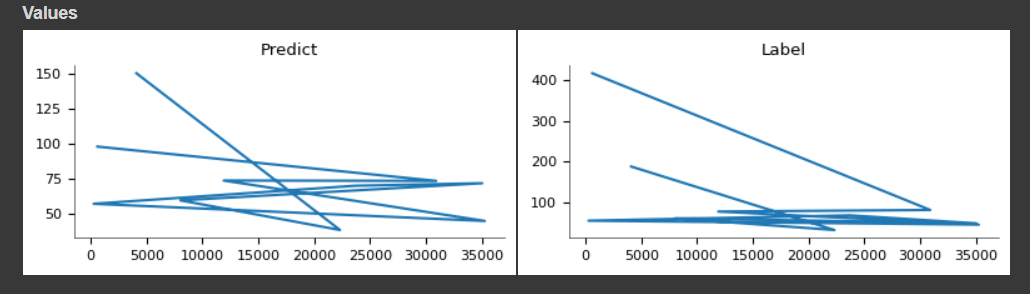
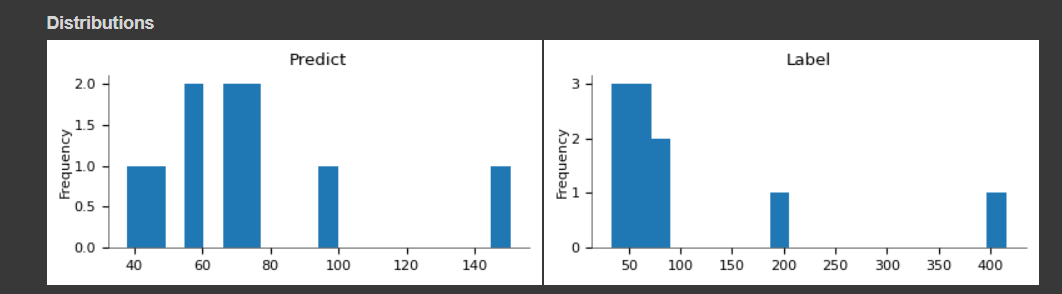
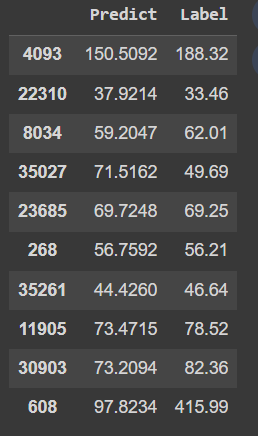
some\_data\_label=y\_test.iloc[50:60]

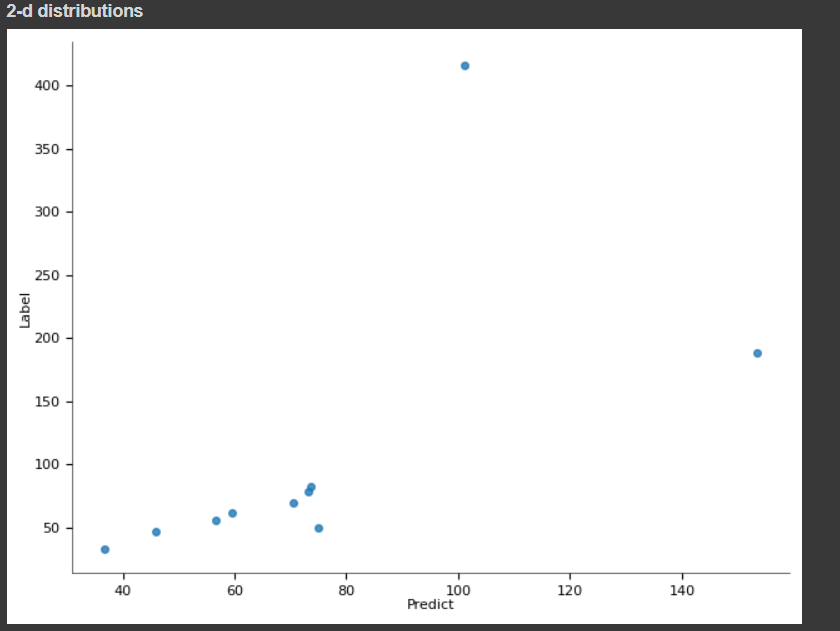
some\_predict=forest\_model.predict(some\_data)

pd.DataFrame({'Predict':some\_predict,'Label':some\_data\_label})

**OUTPUT :**

****

****

****