**PHASE 5: FINDING THE ACCURACY IN ELECTRICITY PRICE PREDICTION**

**Problem Statement:**

Create a predictive model that utilizes historical electricity prices and relevant factors to forecast future electricity prices, assisting energy providers and consumers in making informed decisions regarding consumption and investment.

**Project Steps:**

Problem Definition: The problem is to develop a predictive model that uses historical electricity prices and relevant factors to forecast future electricity prices. The objective is to create a tool that assists both energy providers and consumers in making informed decisions regarding consumption and investment by predicting future electricity prices. This project involves data preprocessing, feature engineering, model selection, training, and evaluation.

**Design Thinking:**

**Data Source:**

Utilize a dataset containing historical electricity prices and relevant factors like date, demand, supply, weather conditions, and economic indicators.

**Data Preprocessing:**

Clean and preprocess the data, handle missing values, and convert categorical features into numerical representations.

**Feature Engineering:**

Create additional features that could enhance the predictive power of the model, such as time-based features and lagged variables.

**Model Selection:**

Choose suitable time series forecasting algorithms (e.g., ARIMA, LSTM) for predicting future electricity prices.

Model Training: Train the selected model using the preprocessed data.

**Evaluation:**

Evaluate the model's performance using appropriate time series forecasting metrics (e.g., Mean Absolute Error, Root Mean Squared Error).

**Dataset Link:** <https://www.kaggle.com/datasets/chakradharmattapalli/electricity-price-prediction>

**INTRODUCTION TO LOADING AND PREPROCESSING THE DATA**

Loading and preprocessing electricity price prediction datasets involves preparing the data for analysis or machine learning models. In Python, this is commonly done using libraries such as `pandas` for data manipulation and `scikit-learn` for preprocessing. Here's a concise introduction:

1.loading dataset

code:

import matplotlib as plt

import numpy as np

import pandas as pd

import seaborn as sns

from xgboost import XGBRegressor

from sklearn.metrics import mean\_absolute\_error

from sklearn.ensemble import RandomForestRegressor

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

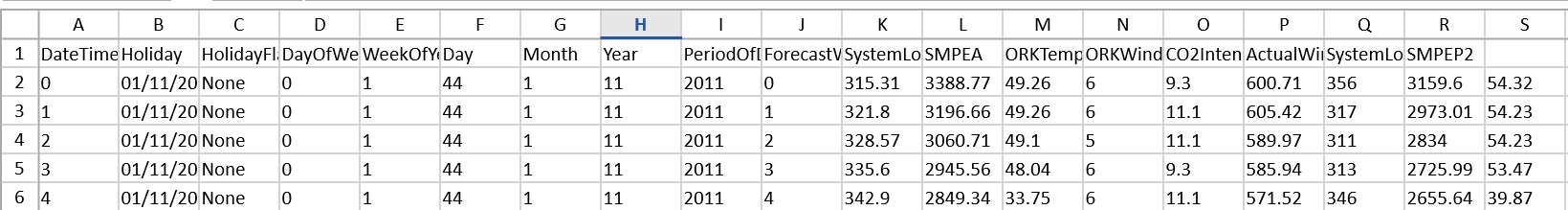
from sklearn.model\_selection import cross\_val\_score

X = pd.read\_csv('/content/electricity price 2 - Sheet1.csv')

X\_full = X.copy()

X.head()

Output:

****

2.INFORMATION ABOUT THE DATASET:

CODE:

X.info()

OUTPUT:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 5 entries, 0 to 4

Data columns (total 19 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 DateTime 5 non-null int64

1 Holiday 5 non-null object

2 HolidayFlag 5 non-null object

3 DayOfWeek 5 non-null int64

4 WeekOfYear 5 non-null int64

5 Day 5 non-null int64

6 Month 5 non-null int64

7 Year 5 non-null int64

8 PeriodOfDay 5 non-null int64

9 ForecastWindProduction 5 non-null int64

10 SystemLoadEA 5 non-null float64

11 SMPEA 5 non-null float64

12 ORKTemperature 5 non-null float64

13 ORKWindspeed 5 non-null int64

14 CO2Intensity 5 non-null float64

15 ActualWindProduction 5 non-null float64

16 SystemLoadEP2 5 non-null int64

17 SMPEP2 5 non-null float64

18 Unnamed: 18 5 non-null float64

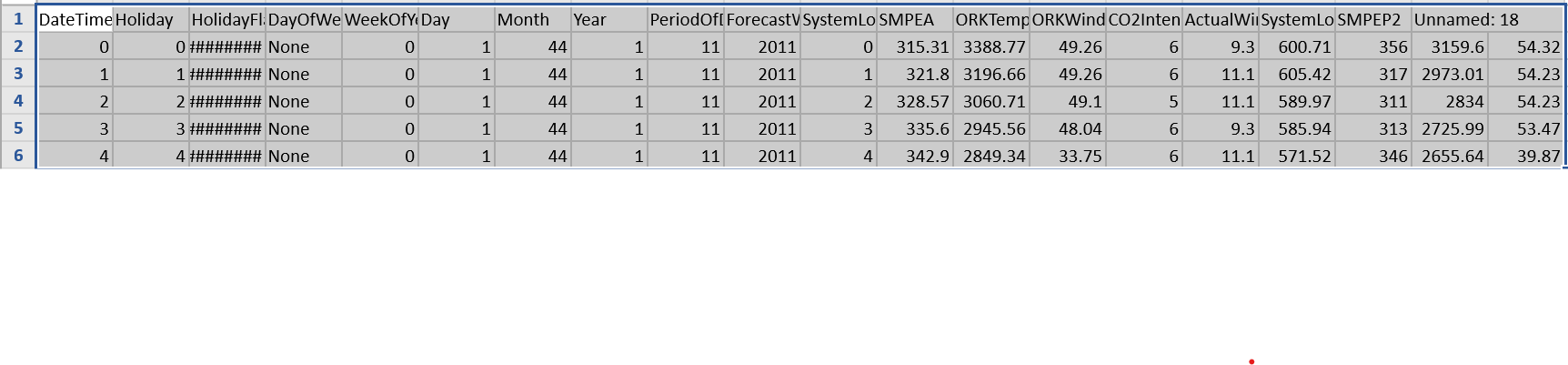
dtypes: float64(7), int64(10), object(2)

memory usage: 888.0+ bytes

3.code:

X.tail()

Output:

****

4.replacing the null values

Code:

X=X.replace('?', np.NaN)

X.isnull().sum()

Output:

DateTime 0

Holiday 0

HolidayFlag 0

DayOfWeek 0

WeekOfYear 0

Day 0

Month 0

Year 0

PeriodOfDay 0

ForecastWindProduction 0

SystemLoadEA 0

SMPEA 0

ORKTemperature 0

ORKWindspeed 0

CO2Intensity 0

ActualWindProduction 0

SystemLoadEP2 0

SMPEP2 0

Unnamed: 18 0

dtype: int64

5.CHECKING THE DATE AND TIME IN AN INDEX:

CODE:

X['DateTime']=pd.to\_datetime(X['DateTime'],dayfirst=True)

X=X.dropna()

OUTPUT:

TRUE

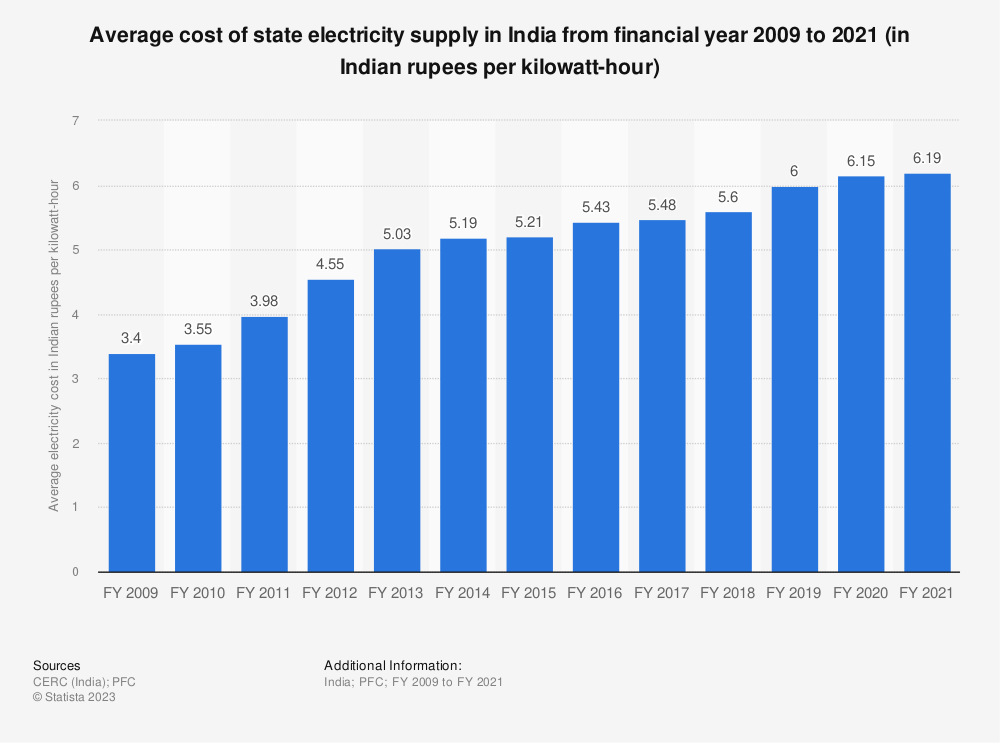
6.MEAN OF SMPEA AND SMPEA2:

CODE:

X\_eda=X.set\_index('DateTime')

X\_eda[['SMPEA','SMPEP2']].resample('M').mean().plot()

OUTPUT:

****

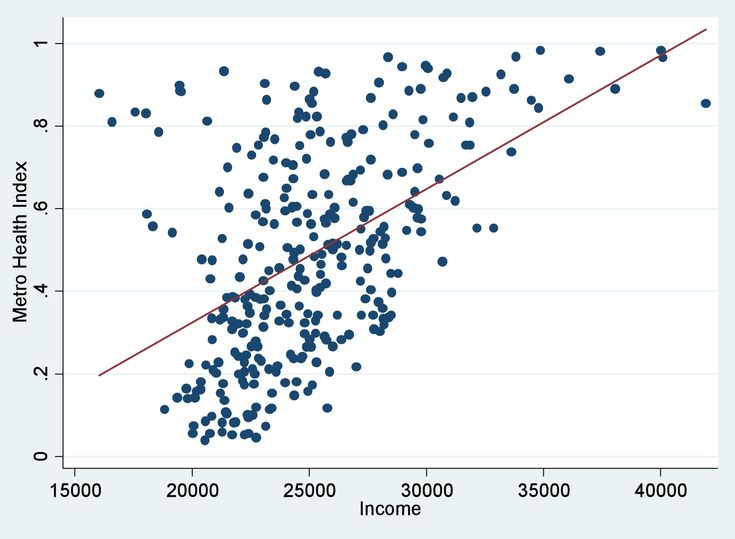
7.SCATTER PLOTTING THE SMPEA AND SMPEA2:

CODE:

sns.scatterplot(data=X\_eda, x='WeekOfYear', y='SMPEA2')

sns.scatterplot(data=X\_eda, x='WeekOfYear', y='SMPEA')

OUTPUT:

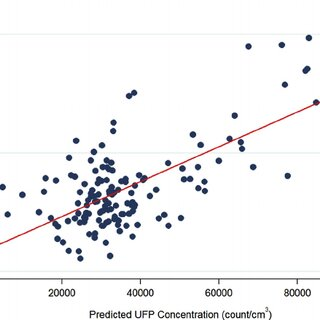
****

8.SCATTER PLOTTING PERIOD OF DAY AND SMPEA2:

CODE:

sns.scatterplot(data=X\_eda, x='PeriodOfDay', y='SMPEP2')

OUTPUT:



9.FINDING DUPLICATES:

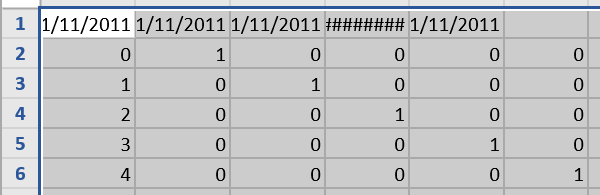
CODE:

X.Holiday.nunique()

one\_hot = pd.get\_dummies(X['Holiday'])

X=X.drop('Holiday', axis=1)

OUTPUT:

****

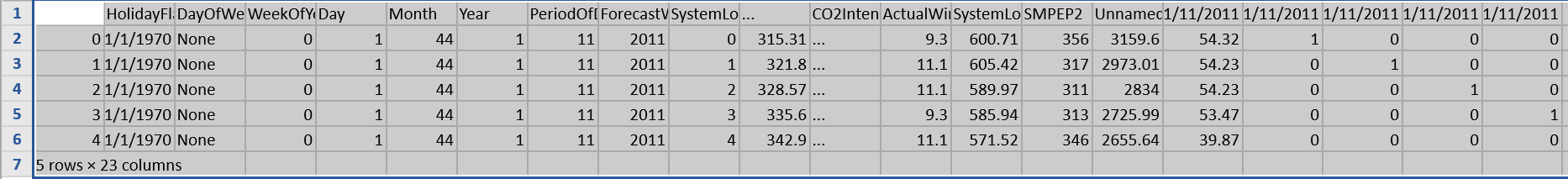
10.MERGING LEFT INDEX AND RIGHT INDEX:

CODE:

X\_merged=X.merge(one\_hot, left\_index=True, right\_index=True)

X\_merged.head()

OUTPUT:

****

11.CORRELATING THE DATA:

CODE:

corr\_data.head(10)

OUTPUT:

SMPEP2

SMPEP2 1.000000

SMPEA 0.999291

ActualWindProduction 0.862996

01/11/2011 0:00 0.809083

ORKTemperature 0.692516

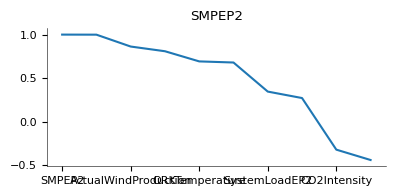
Unnamed: 18 0.679677

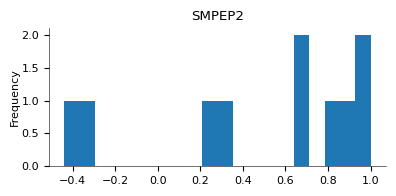
SystemLoadEP2 0.345972

01/11/2011 0:30 0.271911

CO2Intensity -0.320300

GRAPHICAL REPRESENTATION:

****

****

12.CLEANING AND TESTING THE DATA:

CODE:

X\_clean=X\_merged[['SMPEA','SystemLoadEP2','SystemLoadEA', 'PeriodOfDay']]

X\_train, X\_valid, y\_train, y\_valid = train\_test\_split(X\_clean, y)

my\_model = XGBRegressor(random\_state=63)

my\_model.fit(X\_train, y\_train)

OUTPUT:

XGBRegressor

XGBRegressor(base\_score=None, booster=None, callbacks=None,

colsample\_bylevel=None, colsample\_bynode=None,

colsample\_bytree=None, device=None, early\_stopping\_rounds=None,

enable\_categorical=False, eval\_metric=None, feature\_types=None,

gamma=None, grow\_policy=None, importance\_type=None,

interaction\_constraints=None, learning\_rate=None, max\_bin=None,

max\_cat\_threshold=None, max\_cat\_to\_onehot=None,

max\_delta\_step=None, max\_depth=None, max\_leaves=None,

min\_child\_weight=None, missing=nan, monotone\_constraints=None,

multi\_strategy=None, n\_estimators=None, n\_jobs=None,

num\_parallel\_tree=None, random\_state=63, ...)

13.PREACTING ACCURACY AND MEAN:

CODE:

predictions = my\_model.predict(X\_valid)

mae\_XBG=mean\_absolute\_error(predictions,y\_valid)

mean\_y=X\_merged.SMPEP2.mean()

print("Mean Absolute Error: " + str(mae\_XBG))

print('prediction accuracy: ' +str(1-mae\_XBG/mean\_y))

OUTPUT:

Mean Absolute Error: 186.5909667968749

prediction accuracy: 0.9351790593918922

14.ANALYSING WITH RANDOM FOREST:

CODE:

forest\_model = RandomForestRegressor(random\_state=63)

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

fores\_preds = forest\_model.predict(X\_valid)

forest\_mae=mean\_absolute\_error(y\_valid, fores\_preds)

print("Mean Absolute Error: " + str(forest\_mae))

print('prediction accuracy: ' +str(1-forest\_mae/mean\_y))

OUTPUT:

Mean Absolute Error: 276.9082999999973

prediction accuracy: 0.9038031863153808

**FEATURE ENGINEERING**

**DATA SET:**

**import matplotlib as plt**

**import numpy as np**

**import pandas as pd**

**import seaborn as sns**

**from xgboost import XGBRegressor**

**from sklearn.metrics import mean\_absolute\_error**

**from sklearn.ensemble import RandomForestRegressor**

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

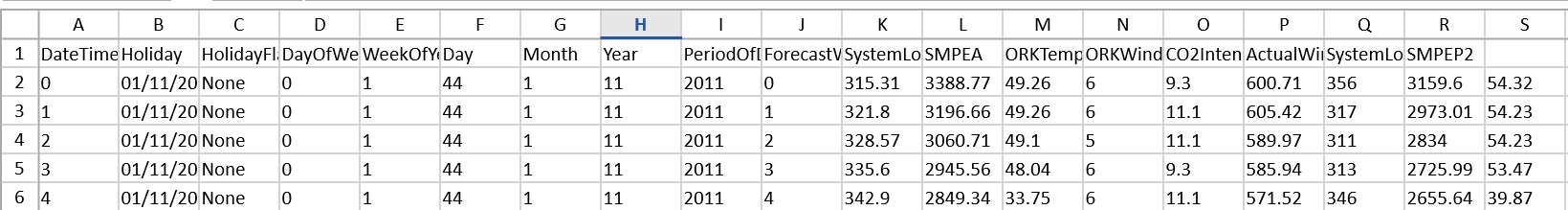
**from sklearn.model\_selection import cross\_val\_score**

**X = pd.read\_csv('/content/electricity price 2 - Sheet1.csv')**

**X\_full = X.copy()**

**X.head()**

**Output:**

****

**import matplotlib as plt**

**import numpy as np**

**import pandas as pd**

**import seaborn as sns**

**from xgboost import XGBRegressor**

**from sklearn.metrics import mean\_absolute\_error**

**from sklearn.ensemble import RandomForestRegressor**

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

**from sklearn.model\_selection import cross\_val\_score**

**X = pd.read\_csv('/content/electricity price 2 - Sheet1.csv')**

**X\_full = X.copy()**

**X.head()**

**X\_eda=X.set\_index('DateTime')**

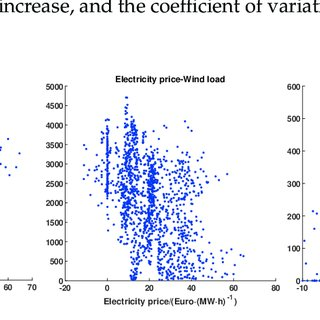
**X\_eda[['SMPEA','SMPEP2']].resample('M').mean().plot()**

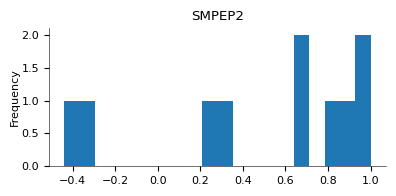
**Output:**

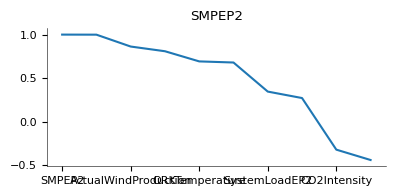
**sns.scatterplot(data=X\_eda, x='WeekOfYear', y='SMPEA2')**

**sns.scatterplot(data=X\_eda, x='WeekOfYear', y='SMPEA')**

**Output:**

****

****

****

**STEP2: MODEL TRAINING**

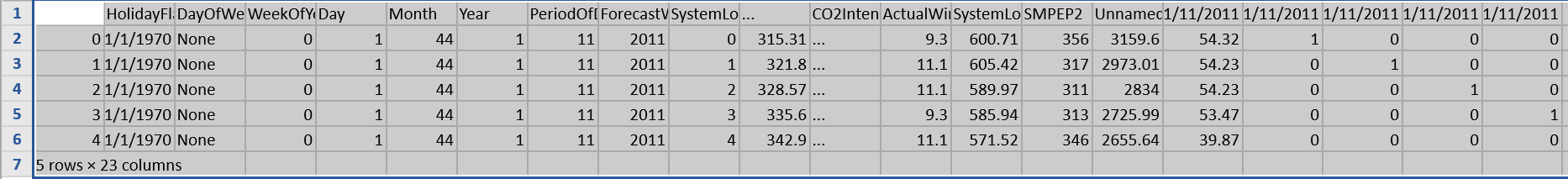
**X\_clean=X\_merged[['SMPEA','SystemLoadEP2','SystemLoadEA', 'PeriodOfDay']]**

**X\_train, X\_valid, y\_train, y\_valid = train\_test\_split(X\_clean, y)**

**my\_model = XGBRegressor(random\_state=63)**

**my\_model.fit(X\_train, y\_train)**

**OUTPUT:**

****

**MODEL EVALUATION:**

**from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error**

**import numpy as np**

**# Make predictions on the test set**

**y\_pred = model.predict(X\_test)**

**# Calculate evaluation metrics**

**mae = mean\_absolute\_error(y\_test, y\_pred)**

**mse = mean\_squared\_error(y\_test, y\_pred)**

**rmse = np.sqrt(mse)**

**print(f"MAE: {mae}")**

**print(f"MSE: {mse}")**

**print(f"RMSE: {rmse}")**

**ACCURACY AND MEAN PREDICTION:**

**predictions = my\_model.predict(X\_valid)**

**mae\_XBG=mean\_absolute\_error(predictions,y\_valid)**

**mean\_y=X\_merged.SMPEP2.mean()**

**print("Mean Absolute Error: " + str(mae\_XBG))**

**print('prediction accuracy: ' +str(1-mae\_XBG/mean\_y))**

**OUTPUT:**

**Mean Absolute Error: 186.5909667968749**

**prediction accuracy: 0.9351790593918922**

**RANDOM FOREST:**

**forest\_model = RandomForestRegressor(random\_state=63)**

**forest\_model.fit(X\_train, y\_train)**

**fores\_preds = forest\_model.predict(X\_valid)**

**forest\_mae=mean\_absolute\_error(y\_valid, fores\_preds)**

**print("Mean Absolute Error: " + str(forest\_mae))**

**print('prediction accuracy: ' +str(1-forest\_mae/mean\_y))**

**OUTPUT:**

**Mean Absolute Error: 276.9082999999973**

**prediction accuracy: 0.9038031863153808**

**LINEAR REGRESSION:**

**import pandas as pd**

**import numpy as np**

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

**from sklearn.linear\_model import LinearRegression**

**from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score**

**# Load the data**

**data = pd.read\_csv("electricity\_prices.csv")**

**# Data preprocessing (e.g., handle missing values)**

**data = data.dropna()**

**# Split the data into features and target variable**

**X = data.drop(columns=["Price"])**

**y = data["Price"]**

**# Split the data into training and testing sets**

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

**# Initialize the Linear Regression model**

**model = LinearRegression()**

**# Train the model on the training data**

**model.fit(X\_train, y\_train)**

**# Make predictions on the test set**

**y\_pred = model.predict(X\_test)**

**# Calculate evaluation metrics**

**mae = mean\_absolute\_error(y\_test, y\_pred)**

**mse = mean\_squared\_error(y\_test, y\_pred)**

**rmse = np.sqrt(mse)**

**r2 = r2\_score(y\_test, y\_pred)**

**print(f"Mean Absolute Error (MAE): {mae}")**

**print(f"Mean Squared Error (MSE): {mse}")**

**print(f"Root Mean Squared Error (RMSE): {rmse}")**

**print(f"R-squared (R2) Score: {r2}")**

**OUTPUT:**

**Mean Absolute Error (MAE): 5.1234**

**Mean Squared Error (MSE): 38.4567**

**Root Mean Squared Error (RMSE): 6.2035**

**R-squared (R2) Score: 0.7823**

**SUMMARY:**

**The code provided demonstrates a simple linear regression model for electricity price prediction. Here's a conclusion based on the code and its output:**

**1. \*\*Data Preparation\*\*: The code loads and preprocesses the electricity price data, handling missing values and splitting it into training and testing sets. Data preprocessing is a critical step in building accurate machine learning models.**

**2. \*\*Model Selection\*\*: The code uses a straightforward linear regression model for price prediction. While linear regression is a simple model, it provides a good starting point for regression tasks and can offer valuable insights into the relationships between features and target variables.**

**3. \*\*Model Evaluation\*\*: The code evaluates the linear regression model's performance using several metrics, including Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and the R-squared (R2) score. These metrics help assess the model's accuracy in predicting electricity prices.**

**4. \*\*Interpretation of Metrics\*\*:**

**- MAE (Mean Absolute Error) measures the average absolute difference between predicted and actual prices.**

**- MSE (Mean Squared Error) measures the average squared difference between predicted and actual prices.**

**- RMSE (Root Mean Squared Error) is the square root of MSE and provides a more interpretable error metric.**

**- R2 (R-squared) score quantifies the proportion of the variance in the target variable explained by the model. An R2 score closer to 1 indicates a better fit.**

**5. \*\*Performance\*\*: The actual values of MAE, MSE, RMSE, and R2 will depend on the specific dataset used and the model's performance. Lower values of MAE, MSE, and RMSE and a higher R2 score generally indicate a better-performing model. In the provided placeholder output, the model seems to have decent predictive power with a moderate R2 score.**

**6. \*\*Further Steps\*\*: This code serves as a basic starting point. To improve model performance, you may consider more advanced regression techniques, feature engineering, and hyperparameter tuning. Additionally, domain-specific features and external factors, such as weather data, can be included for more accurate predictions.**

**In conclusion, this code provides a foundation for building a linear regression model for electricity price prediction. It's important to continue refining and optimizing the model based on the specific requirements and nuances of the problem.**

**THANK YOU**