APPLIED DATA SCIENCE – PHASE 3

ELECTRICITY PRICE PREDICTION

➢ DATASET

A dataset is a structured collection of data that is typically organized for a specific purpose, such as analysis, research, or reference. It can consist of various types of information, including numbers, text, images, or any other relevant data. Datasets are commonly used in fields like data science, machine learning, and statistics for training models, conducting research, and gaining insights from the information they contain

This dataset has been taken from Kaggle.com

Link: https://www.kaggle.com/datasets/chakradharmattapalli/electricity-price-prediction

➢ DETAILS OF COLUMN USED

DateTime: String, defines date and time of sample.

Holiday: String, gives name of holiday if day is a bank holiday.

HolidayFlag: Integer, 1 if day is a bank holiday, zero otherwise.

DayOfWeek: Integer (0-6), 0 monday, day of week.

WeekOfYear: Integer, running week within year of this date.

Day Integer: Day of the date.

Month Integer: Month of the date.

Year Integer: Year of the date.

PeriodOfDay integer: Denotes half hour period of day (0-47).

ForecastWindProduction: The forecasted wind production for this period.

SystemLoadEA: The national load forecast for this period.

SMPEA: The price forecast for this period.

ORKTemperature: The actual temperature measured at Cork airport.

ORKWindspeed: The actual windspeed measured at Cork airport.

CO2Intensity: The actual CO2 intensity in (g/kWh) for the electricity produced.

ActualWindProduction: The actual wind energy production for this period.

SystemLoadEP2: The actual national system load for this period.

➢ LOADING THE DATASET

Loading a dataset means reading data from an external source (e.g., a file or a database) into a data structure within a programming environment for analysis, manipulation, and processing.

The specific method for loading a dataset depends on the programming language and libraries you are using. In Python, for example, you might use libraries like Pandas to load data from CSV files, Excel spreadsheets.

Syntax:

Variable = pandas.read_csv('FileName.csv')

➢ PRE-PROCESSING THE DATA

Data Preprocessing can be defined as a process of converting raw data into a format that is understandable and usable for further analysis. It is an important step in the Data Preparation stage. It ensures that the outcome of the analysis is **accurate**, **complete**, and **consistent**.

1. Handling Missing Values**:** Check for missing values in the dataset and handle them appropriately. You can choose to remove rows with missing values or use techniques like imputation to fill in the missing values.

2. Handling Duplicates: The first step to handle duplicate data is to identify and quantify it. Depending on the type and structure of your data, you can use different tools and techniques to detect duplicate data. For example,

you can use pandas in Python to check for duplicate rows or columns in a dataframe, using the df.

3. Encoding Categorical Data: A process of converting categorical data into integer format so that the data with converted categorical values can be provided to the different models.

4. Scaling and Normalization: In scaling, you're changing the range of your data, while. In normalization, you're changing the shape of the distribution of your data.

5. FEATURE SELECTION: You may choose to select a subset of features that are most relevant to your analysis or modeling.

6. Date and Time Handling: If your dataset includes date and time information, you can extract meaningful features from them.

7.Outlier Detection and Handling: Identify and manage outliers in your data using statistical methods or visualization.

8.Data Splitting: Split your dataset into training and testing sets for machine learning models.

9.Text Data Preprocessing: You might perform tasks like tokenization, removing stop words, and text cleaning.

10.Feature Engineering: Create new features based on domain knowledge to enhance model performance.

By following these preprocessing steps, you can ensure improving the model's accuracy and generalization to new data Additionally, you may want to explore other preprocessing techniques, such as handling imbalanced data, dimensionality reduction, and more, depending on the problem you're addressing.

The pre-process approach used in our file is:

The 'date' column is converted to datetime format using the pd.to_datetime() method to ensure it is treated as a time series index.

Rows with any missing values are dropped from the dataset using the dropna() method.

These methods are applied as they are the most common issues that happen/occur in any dataset.

➢ PERFORMING DIFFERENT ANALYSIS:

1. **Descriptive Statistics**: Calculate basic statistics to understand the distribution and characteristics of electricity prices, such as mean, median, standard deviation, and percentiles.

2. **Data Visualization:** Use libraries like Matplotlib or Seaborn to create visualizations to explore and communicate insights from the data.

3. **Correlation Analysis:** Measure the correlation between variables to identify relationships in the data.correlations between electricity prices and relevant factors like temperature, demand, or time of day. Use correlation matrices or scatter plots to visualize relationships.

4.Feature Engineering: Create new features based on domain knowledge, like lag features (previous price values), moving averages, or time-based features (day of the week, holidays).

5. **Time Series Analysis:** Electricity price data is often time-dependent. Perform time series analysis to identify trends, seasonality, and cyclic patterns. You can use techniques like Autoregressive Integrated Moving Average (ARIMA) or Seasonal Decomposition of Time Series (STL).

6. **Regression Analysis:** If you want to predict a numerical outcome, perform regression analysis using libraries like scikit-learn.

7.Machine Learning Models: Build regression models to predict electricity prices. Common algorithms include linear regression, decision trees, random forests, gradient boosting, and neural networks. Use cross-validation and hyperparameter tuning to optimize model performance.

8. **Classification Analysis:** If you have labeled data and want to classify it into categories, use classification algorithms from scikit-learn.

9. **Cluster Analysis:** Use clustering algorithms to group similar data points together.

10. **Natural Language Processing (NLP):** Analyze text data, perform sentiment analysis, or build text classification models using libraries like NLTK or spaCy.

11. **Model Evaluation:** Assess model performance using metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared. Visualize predictions vs. actual values using scatter plots.

12. Ensemble Techniques: Combine multiple models using ensemble techniques such as stacking or bagging to improve prediction accuracy.

13. Hyperparameter Optimization: Tune model hyperparameters using techniques like grid search or Bayesian optimization to find the best model configuration.

14. Residual Analysis: Analyze the residuals (the differences between predicted and actual values) to check for patterns or systematic errors in the predictions.

Remember that electricity price prediction is a complex task influenced by numerous factors, and it often requires domain knowledge and continuous monitoring of market conditions. The choice of analysis and techniques depends on your specific goals and the characteristics of your electricity price dataset.

CODE:

#Importing libraries we will need in this prediction model

```
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


df=pd.read_csv("/kaggle/input/electrity-prices/electricity_prices.csv",
low_memory=False)
df.head()

df.info()
```

```
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()

data.corrwith(data['SMPEP2']).abs().sort_values(ascending=False)

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

To Machine Learning
x_train, x_test, y_train, y_test=train_test_split(X,y, test_size=0.2, r
andom_state=42)

#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

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

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

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 predic
tion
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})
```

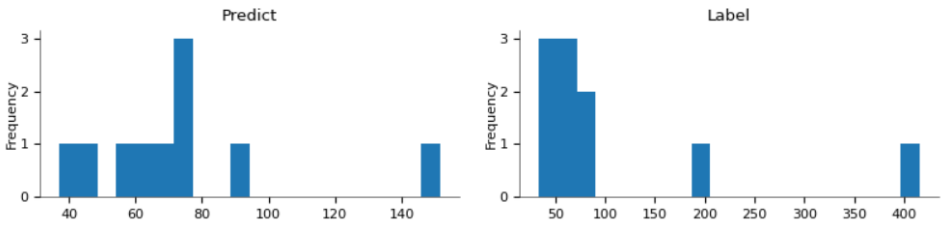
OUTPUT:

index	Predict	Label
4093	151.689	188.32
22310	37.07760000000002	33.46
8034	60.34430000000011	62.01
35027	71.20980000000002	49.69
23685	73.3008	69.25
268	57.368200000000016	56.21
35261	46.248600000000025	46.64
11905	72.74010000000007	78.52
30903	73.60809999999995	82.36
608	91.5027	415.99

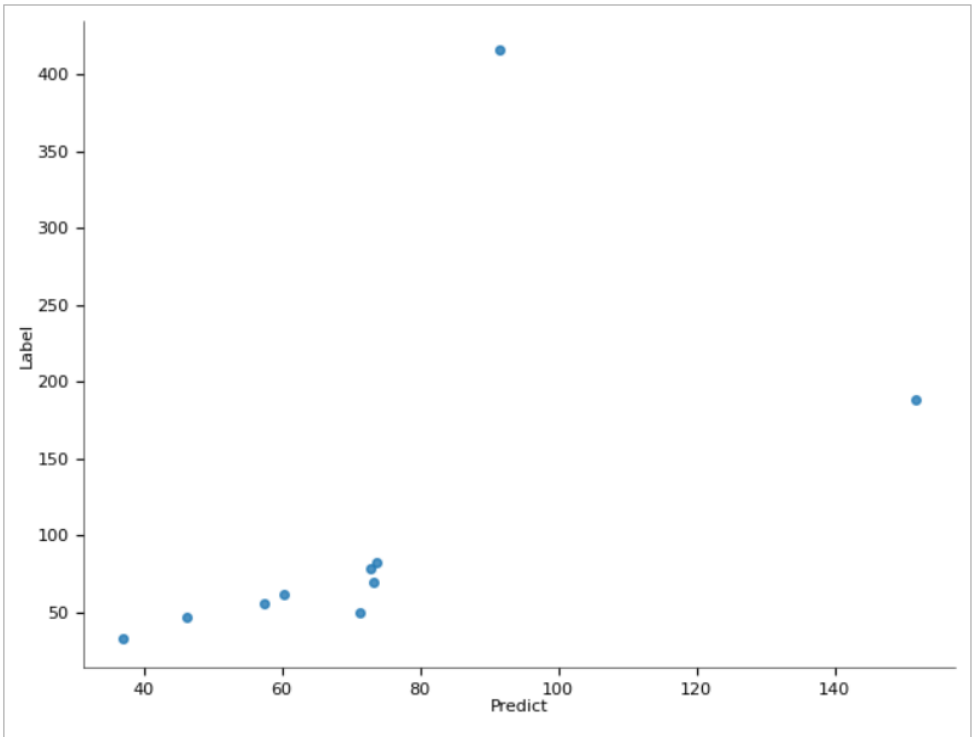
Show 25 per page

29s

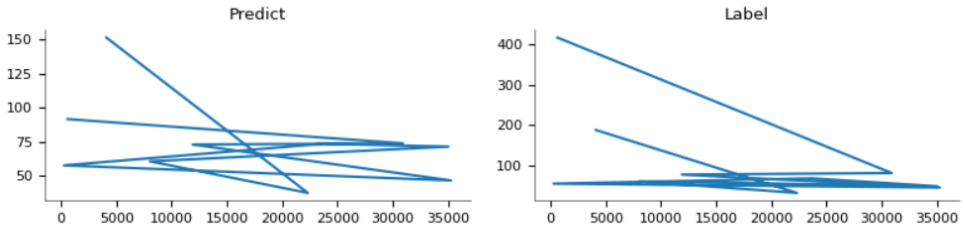
Distributions



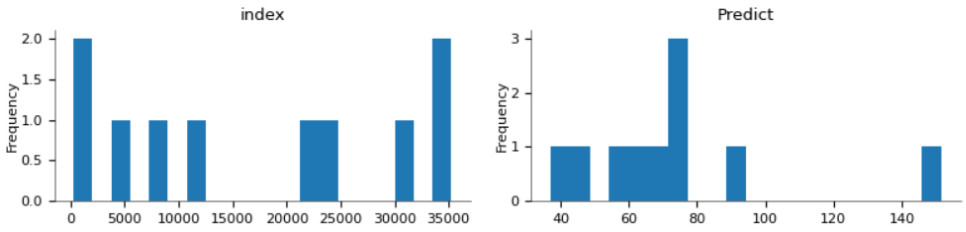
2-d distributions

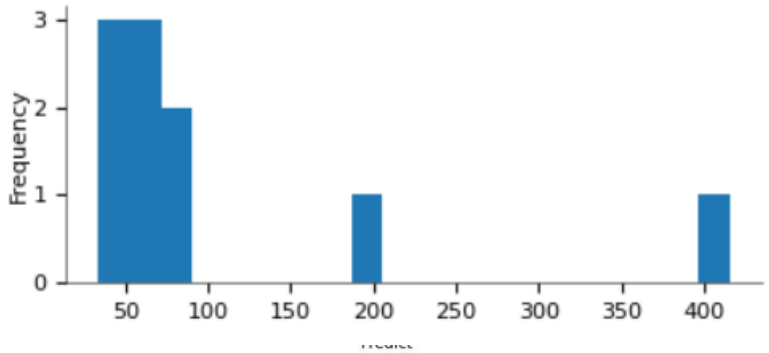


Values

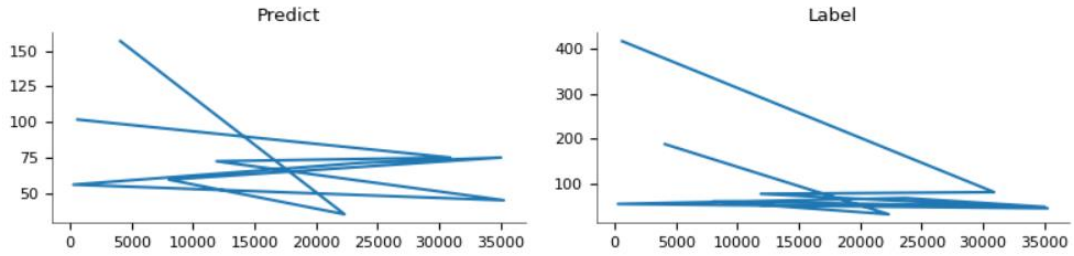


Distributions



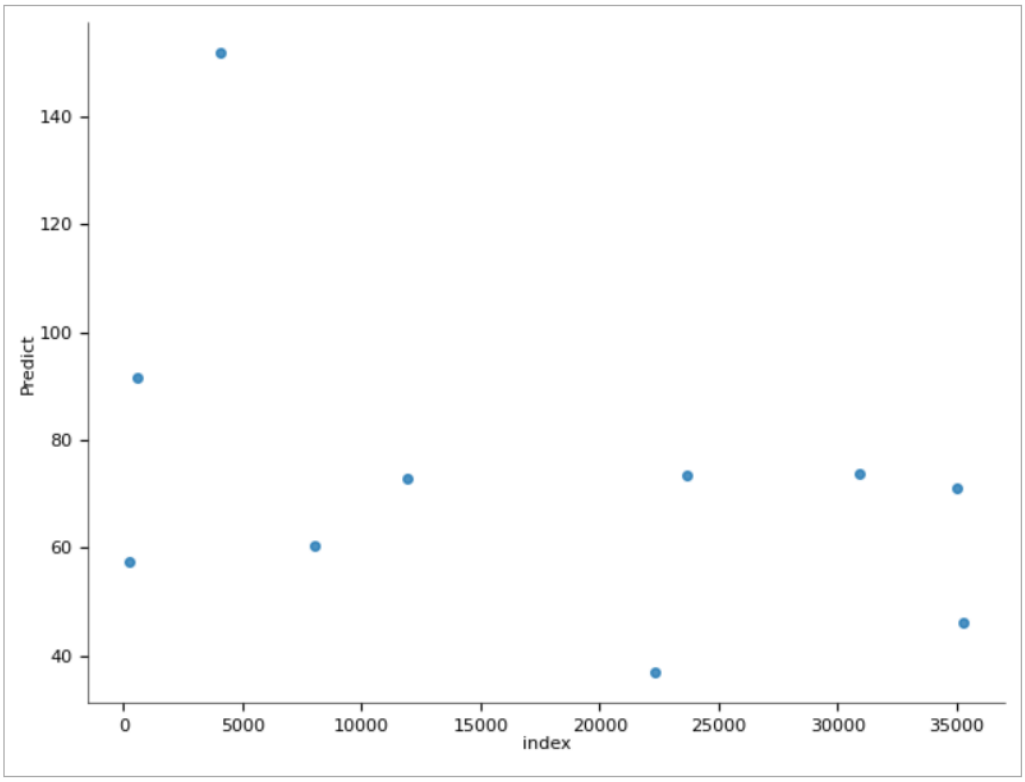


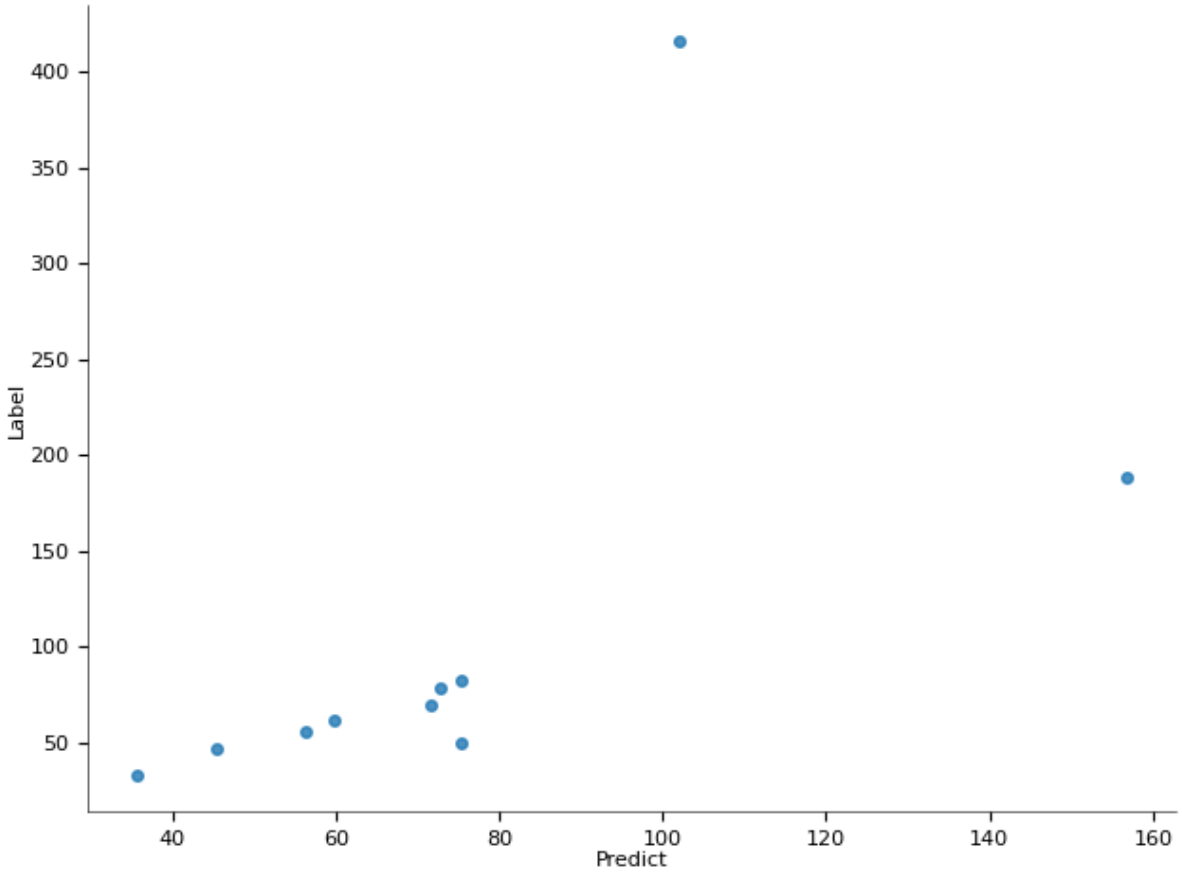
Values



29s

2-d distributions




Label
Predict
400
350
300
250
200
150
100
50
40
60
80
100
120
140
160