ELECTRICITY PRICE PREDICTION (PHASE 3)

Abstract:

The process for creating a predictive model that predicts future power costs using historical data and pertinent variables is described in this document. Offering a tool to help energy suppliers and customers make educated decisions about consumption and investment using suitable analytic methods is the goal.

Problem Description:

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.

3.1 Dataset Information:

The dataset contains historical electricity prices and relevant factors, including:

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

- **Date:** Timestamps for each data point.
- **Demand:** Electricity demand information.
- **Supply:** Electricity supply data.
- **Weather Conditions:** Weather-related factors that may impact electricity prices.
- **Economic Indicators:** Economic data that could influence electricity prices.

Dataset Columns:

1. DateTime
2. Holiday
3. HolidayFlag
4. DayofWeek
5. WeekOfYear
6. Day
7. Month
8. Year
9. PeriodOfDay
10. ForecastWindProduction
11. SystemLoadEA
12. SMPEA
13. ORKTemperature

14. ORKWindspeed
15. CO2Intensity
16. ActualWindProduction
17. SystemLoadEP2
18. SMPEP2

3.2 Loading the dataset:

1. Importing the datasets:

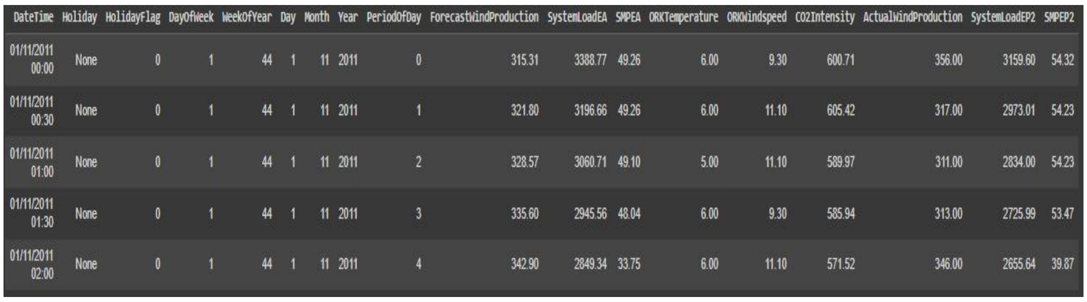
Import the dataset using a function in pandas called pd.read_csv.

```
data=pd.read_csv("Electricity.csv")
```

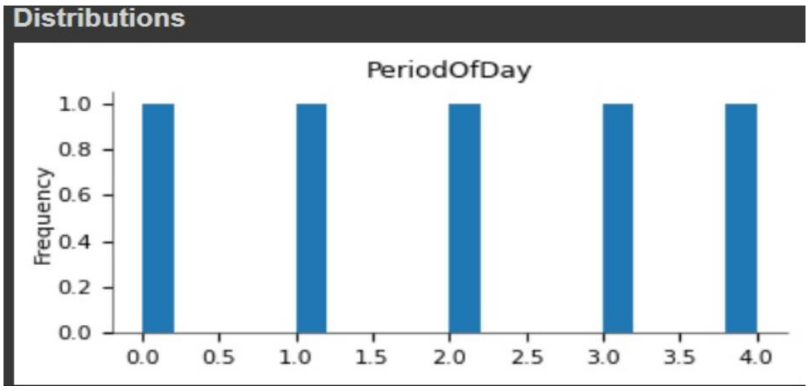
2. View of the dataset:

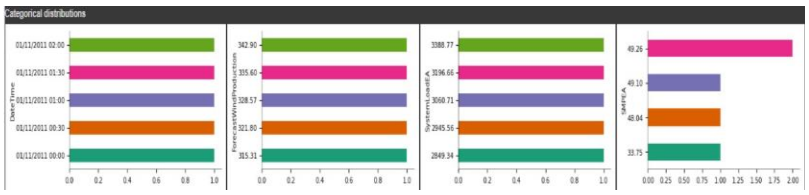
By using a special function called head(), we will be displaying the first 5 columns and if mentioned any number(n) within the parentheses, then 'n' rows of data will be printed.

```
data.head()
```



DateTime	Holiday	HolidayFlag	DayOfWeek	WeekOfYear	Day	Month	Year	PeriodOfDay	ForecastWindProduction	SystemLoadEA	SMPEA	ORKTemperature	ORKWindspeed	CO2Intensity	ActualWindProduction	SystemLoadEP2	SMPEP2
01/11/2011 00:00	None	0	1	44	1	11	2011	0	315.31	3388.77	49.26	6.00	9.30	600.71	356.00	3159.60	54.32
01/11/2011 00:30	None	0	1	44	1	11	2011	1	321.80	3196.66	49.26	6.00	11.10	605.42	317.00	2973.01	54.23
01/11/2011 01:00	None	0	1	44	1	11	2011	2	328.57	3060.71	49.10	5.00	11.10	589.97	311.00	2834.00	54.23
01/11/2011 01:30	None	0	1	44	1	11	2011	3	335.60	2945.56	48.04	6.00	9.30	585.94	313.00	2725.99	53.47
01/11/2011 02:00	None	0	1	44	1	11	2011	4	342.90	2849.34	33.75	6.00	11.10	571.52	346.00	2655.64	39.87





3.Shape of a dataset:

The shape of the dataset is basically a representation of total rows and columns present in the dataset.

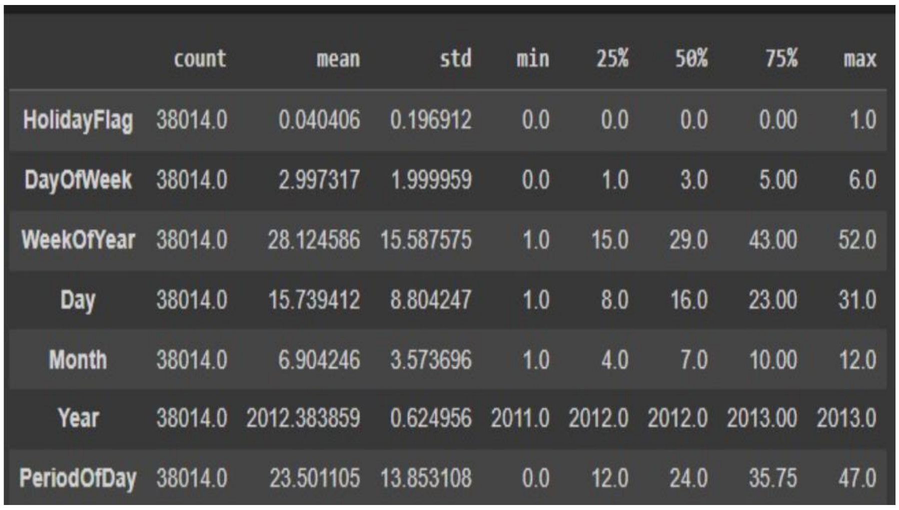
```
data.shape
```

```
(38014, 18)
```

4. Descriptive statistics of the data-sets:

In pandas, describe() function is used to view central tendency, mean, median, standard deviation, percentile & many other things to give you the idea about the data.

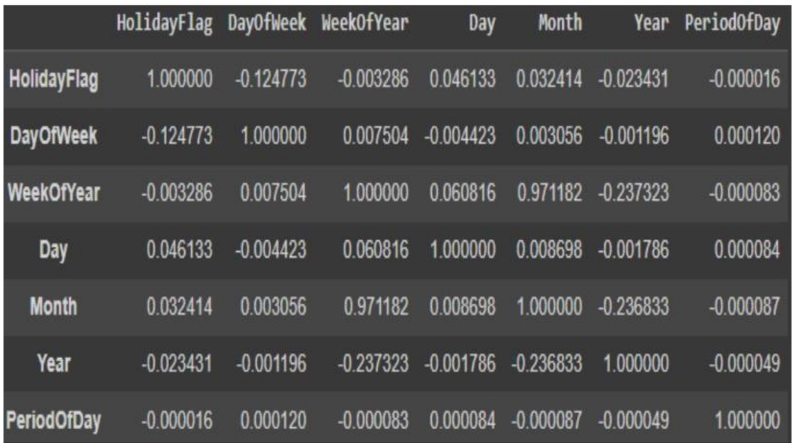
```
data.describe().transpose()
```



	count	mean	std	min	25%	50%	75%	max
HolidayFlag	38014.0	0.040406	0.196912	0.0	0.0	0.0	0.00	1.0
DayOfWeek	38014.0	2.997317	1.999959	0.0	1.0	3.0	5.00	6.0
WeekOfYear	38014.0	28.124586	15.587575	1.0	15.0	29.0	43.00	52.0
Day	38014.0	15.739412	8.804247	1.0	8.0	16.0	23.00	31.0
Month	38014.0	6.904246	3.573696	1.0	4.0	7.0	10.00	12.0
Year	38014.0	2012.383859	0.624956	2011.0	2012.0	2012.0	2013.00	2013.0
PeriodOfDay	38014.0	23.501105	13.853108	0.0	12.0	24.0	35.75	47.0

5. Checking about the correlation between features in a dataset:

pd.DataFrame.corr() calculates the correlation between features pairwise excluding null values.A correlation matrix is a table showing correlation coefficients between variables. Each cell in the table shows the correlation between two variables. A correlation matrix is used to summarize data, as an input into a more advanced analysis, and as a diagnostic for advanced analyses.



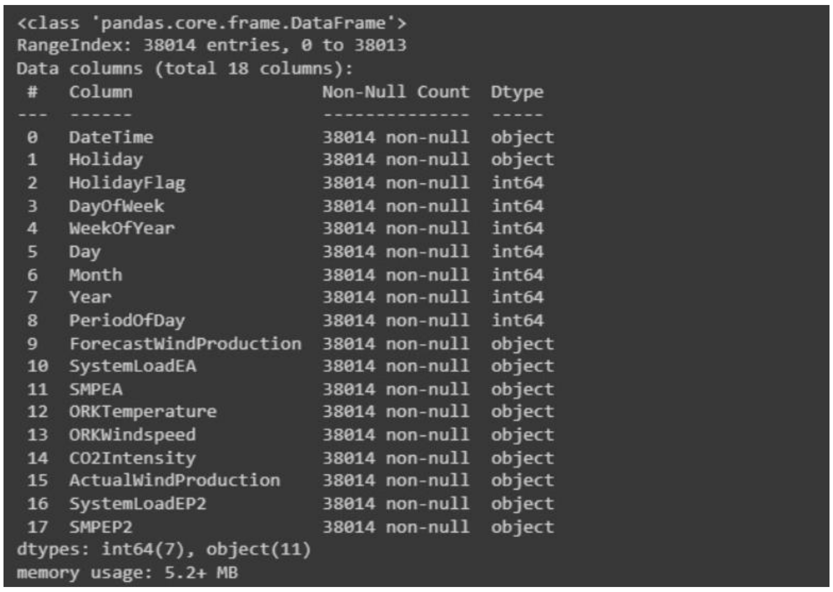
	HolidayFlag	DayOfWeek	WeekOfYear	Day	Month	Year	PeriodOfDay
HolidayFlag	1.000000	-0.124773	-0.003286	0.046133	0.032414	-0.023431	-0.000016
DayOfWeek	-0.124773	1.000000	0.007504	-0.004423	0.003056	-0.001196	0.000120
WeekOfYear	-0.003286	0.007504	1.000000	0.060816	0.971182	-0.237323	-0.000083
Day	0.046133	-0.004423	0.060816	1.000000	0.008698	-0.001786	0.000084
Month	0.032414	0.003056	0.971182	0.008698	1.000000	-0.236833	-0.000087
Year	-0.023431	-0.001196	-0.237323	-0.001786	-0.236833	1.000000	-0.000049
PeriodOfDay	-0.000016	0.000120	-0.000083	0.000084	-0.000087	-0.000049	1.000000

3.3 Analysis on Dataset:

6. Checking about data types and more information about the data:

pd.dataFrame.info() which returns the data type of each column present in the dataset. It tells about null and not null values present.

```
data.info()
```

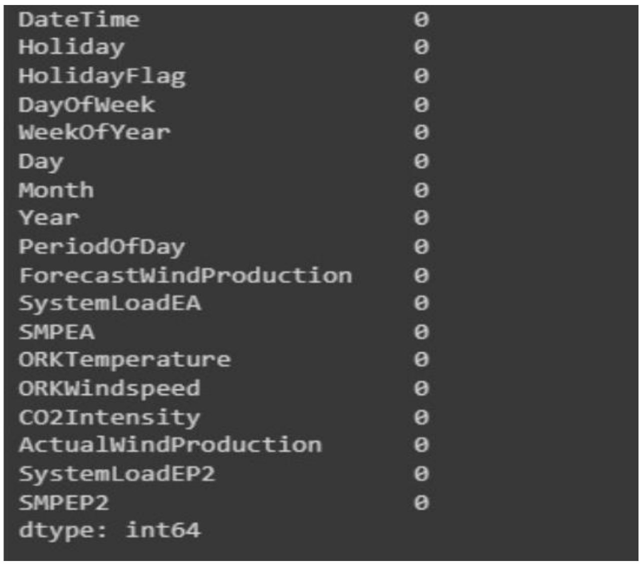


```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 38014 entries, 0 to 38013
Data columns (total 18 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   DateTime                38014 non-null  object
 1   Holiday                 38014 non-null  object
 2   HolidayFlag             38014 non-null  int64
 3   DayOfWeek               38014 non-null  int64
 4   WeekOfYear              38014 non-null  int64
 5   Day                     38014 non-null  int64
 6   Month                   38014 non-null  int64
 7   Year                    38014 non-null  int64
 8   PeriodOfDay             38014 non-null  int64
 9   ForecastWindProduction  38014 non-null  object
 10  SystemLoadEA            38014 non-null  object
 11  SMPEA                   38014 non-null  object
 12  ORKTemperature          38014 non-null  object
 13  ORKWindspeed            38014 non-null  object
 14  CO2Intensity            38014 non-null  object
 15  ActualWindProduction    38014 non-null  object
 16  SystemLoadEP2           38014 non-null  object
 17  SMPEP2                  38014 non-null  object
dtypes: int64(7), object(11)
memory usage: 5.2+ MB
```

7. Checking about missing values in the data:

Missing values in the data can be checked by using isnull() function present in pandas.

```
data.isnull().sum()
```



```
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
dtype: int64
```

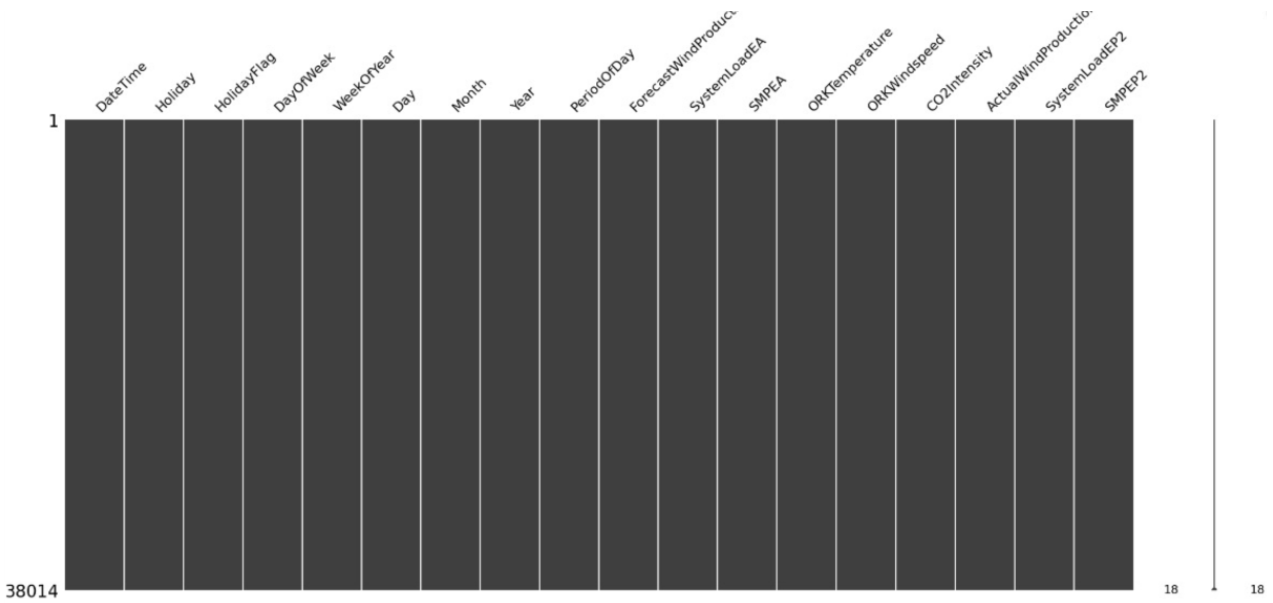
Visualize missing values (NaN) values using Missingno Library:

Missingno library offers a very nice way to visualize the distribution of NaN values. Missingno is a Python library and compatible with Pandas.

Matrix :

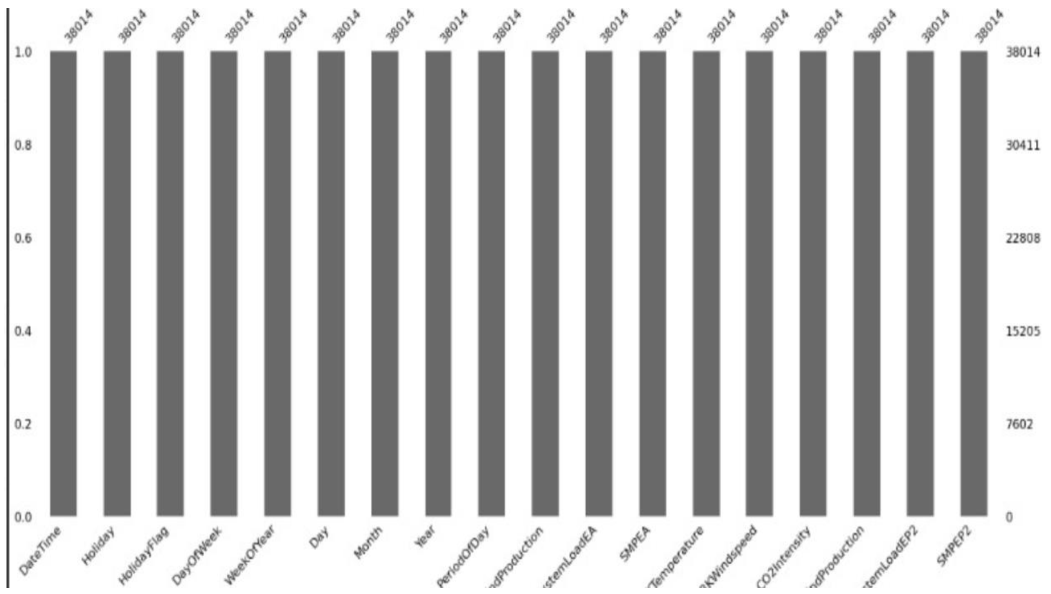
Matrix can quickly find the pattern of missingness in the dataset.

```
import missingno as msno
msno.matrix(data)
```



Bar Chart :

This bar chart gives you an idea about how many missing values are there in each column.



1.Dropping data:

dropna() function is used to remove missing values from the dataset

```
data = data.dropna()
```

2.Replacing with the Previous value – forward fill: In some cases, imputing the values with the previous value instead of the mean, mode, or median is more appropriate. This is called forward fill. It is mostly used in time series data.

```
data = data.fillna(method = "ffill")
```

3.Replacing with the Next value – backward fill: In backward fill, the missing value is imputed using the next value

```
data = data.fillna(method = "bfill")
```

4.Interpolation: Missing values can also be imputed using interpolation. Pandas' interpolate method can be used to replace the missing values with different interpolation methods like 'polynomial,' 'linear,' and 'quadratic.' The default method is 'linear.'

```
data = data.interpolate()
```

Feature Scaling:

Feature scaling is a data pre-processing technique used to transform the values of features or variables in a dataset to a similar scale. The purpose is to ensure that all features contribute equally to the model and to avoid the domination of features with larger values. Feature scaling becomes necessary when dealing with datasets containing features that have different ranges, units of measurement, or orders of magnitude. In such cases, the variation in feature values can lead to biased model performance or difficulties during the learning process There are several common techniques for feature scaling, including standardization, normalization, and min-max scaling. These methods adjust the feature values while preserving their relative relationships and distributions. Tree-based algorithms are fairly insensitive to the scale of the features. A decision tree only splits a node based on a single feature. The decision tree splits a node on a feature that increases the homogeneity of the node. Other features do not influence this split on a feature. So, the remaining features have virtually no effect on the split. This is what makes them invariant to the scale of the features!

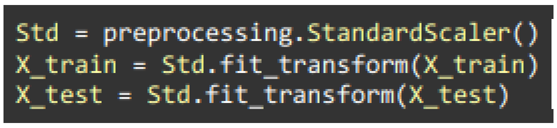
1. Normalization: Normalization is a scaling technique in which values are shifted and rescaled so that they end up ranging between 0 and 1. It is also known as Min-Max scaling. Normalization Equation Xmax and Xmin are the maximum and the minimum values of the feature

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

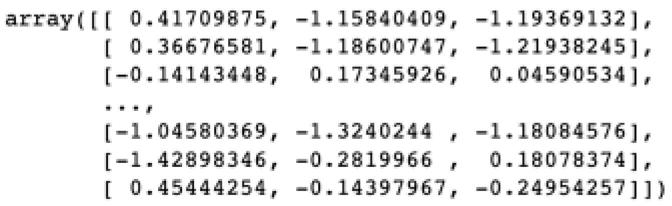
2. Standardization: Standardization is another scaling method where the values are centered around the mean with a unit standard deviation.This means that the mean of the attribute becomes zero, and the resultant distribution has a unit standard deviation. Standardization equation Normalization also helps to reduce the impact of outliers and improve the accuracy and stability of statistical models

$$X' = \frac{X - \mu}{\sigma}$$

Standardization:

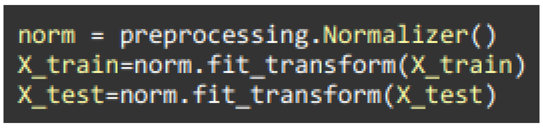


```
Std = preprocessing.StandardScaler()
X_train = Std.fit_transform(X_train)
X_test = Std.fit_transform(X_test)
```

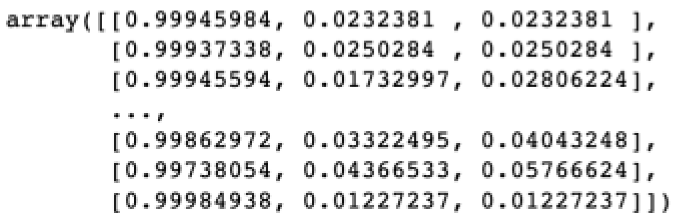


```
array([[ 0.41709875, -1.15840409, -1.19369132],
       [ 0.36676581, -1.18600747, -1.21938245],
       [-0.14143448,  0.17345926,  0.04590534],
       ...,
       [-1.04580369, -1.3240244 , -1.18084576],
       [-1.42898346, -0.2819966 ,  0.18078374],
       [ 0.45444254, -0.14397967, -0.24954257]])
```

Normalization:



```
norm = preprocessing.Normalizer()
X_train=norm.fit_transform(X_train)
X_test=norm.fit_transform(X_test)
```



```
array([[0.99945984, 0.0232381 , 0.0232381 ],
       [0.99937338, 0.0250284 , 0.0250284 ],
       [0.99945594, 0.01732997, 0.02806224],
       ...,
       [0.99862972, 0.03322495, 0.04043248],
       [0.99738054, 0.04366533, 0.05766624],
       [0.99984938, 0.01227237, 0.01227237]])
```

Conclusion:

In conclusion, the rigorous pre-processing of the dataset, sourced from Kaggle, has rendered it well-suited for the subsequent stages of model development and analysis. Addressing correlations, providing detailed feature descriptions, inspecting data properties, handling missing values, and applying feature scaling techniques were pivotal steps in ensuring data quality and reliability. The attention to detail and rigor applied in the pre-processing phase will ultimately contribute to the success and accuracy of our machine learning endeavours.