Predicting Energy Consumption Using Machine Learning

STEEL INDUSTRY

GROUP NUMBER 8

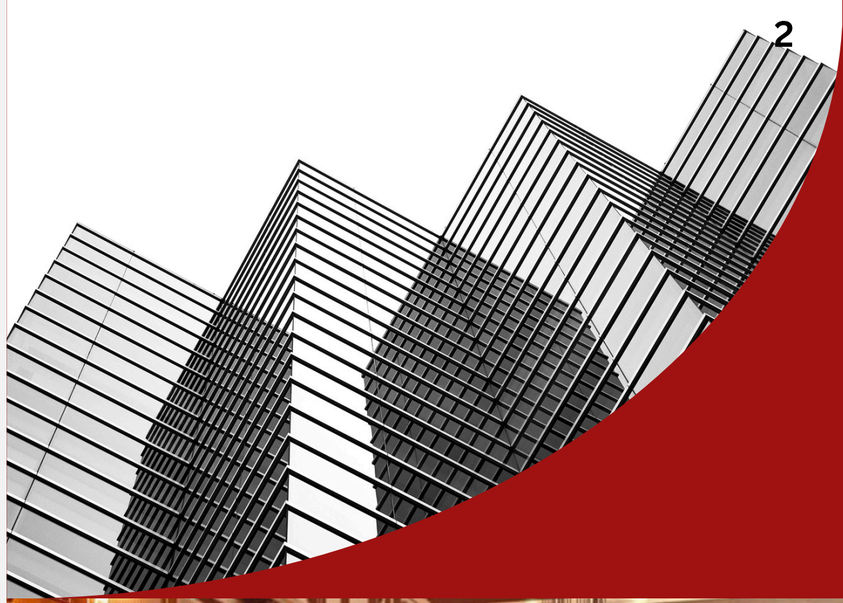
Bhavan’s Vivekananda College

R.SREE VARDHINI TULASI | PULIGADDA VENKATA KRISHNA KIREETI

PULIGADDA VENKATA SAI KIRAN | RAMIDI HARISHWAR REDDY

Abstract

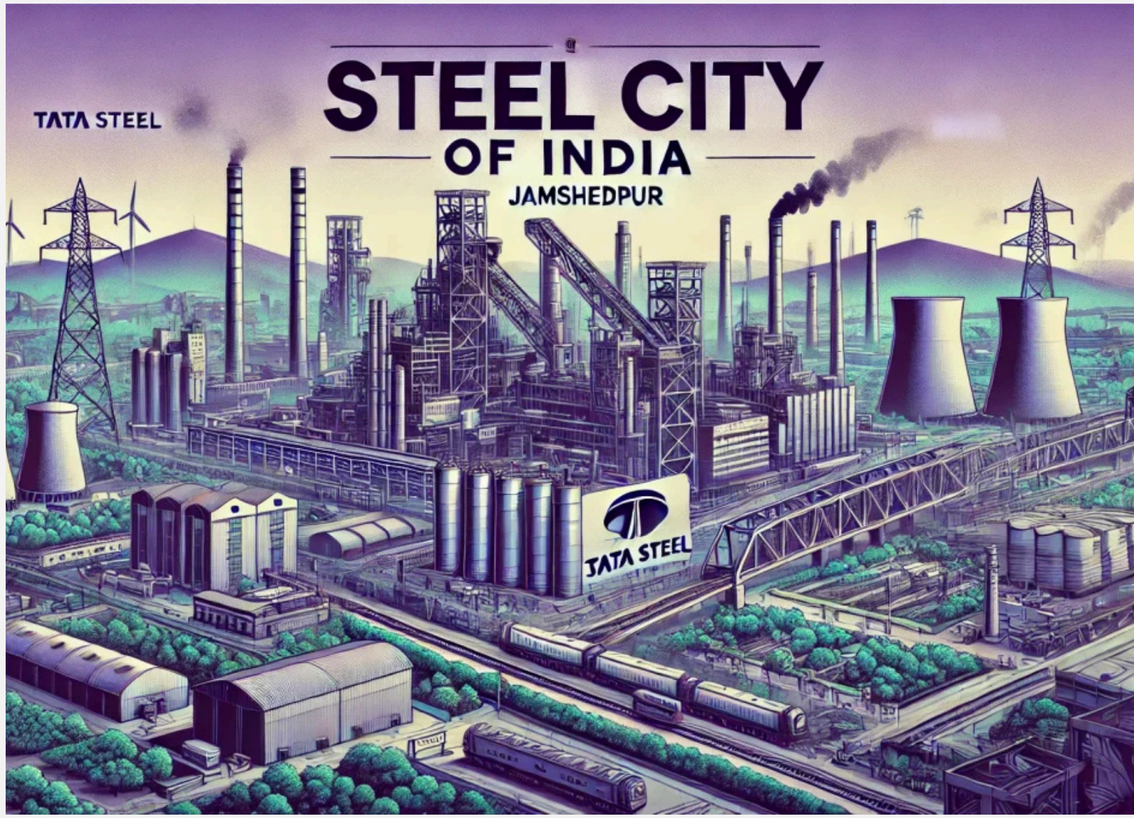
This study explores the application of machine learning techniques using Python to analyze and predict the energy consumption within the steel industry. By leveraging libraries such as Pandas, NumPy, and Scikit-learn, we developed predictive model for the steel industry, specifically targeting the prediction of energy consumption.



Objective

To find the suitable machine learning model to predict the energy consumption in steel industry for implementing eco-friendly production methods.

Content

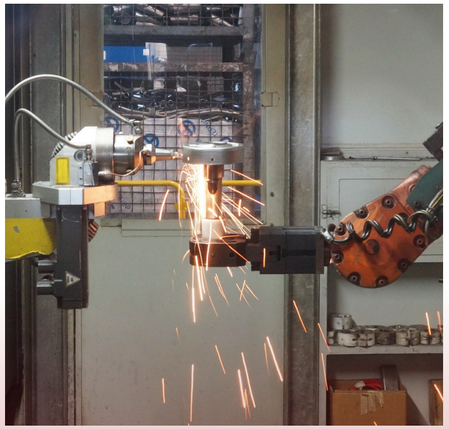


Introduction

- The steel industry is often considered an indicator of economic progress, because of the critical role played by steel in infrastructural and overall economic development.

- Technological advancements: Automation and digitalization are transforming steel manufacturing, leading to improved efficiency and quality.

Literature Review



Literature review - 1

Authors:
Danish Irfan
Sivaraj Varadharajan
Shahina Mateen
Syed Md. Mobasshir

“Study of Growth of Steel, Steel Infrastructure and Steel Industries in India” (2023)

This study explores the growth of India's steel industry, which contributes around 2% to GDP, focusing on steel production, demand, and consumption trends. It highlights key developments such as the establishment of TISCO and SAIL, helping India become the world's second-largest steel producer with 111 MT of crude steel production.[1]

Literature review - 2

Authors:
Neelam Sharma
Viswanathan N. Nurni
Vilas Tathavadkar
Somnath Basu

“A review on the generation of solid wastes and their utilization in Indian steel industries”
(2017)

As India becomes a major metal producer, the steel industry faces significant challenges in solid waste management, generating 1.6 tons of waste per ton of steel produced. This paper addresses the issues related to waste generation, including Blast Furnace slag and steel-making slag, and efforts towards recycling and utilization.

Data Pre-Processing

Data

Dataset : Our Dataset consists of 11 variables and 35040 records

Source : https://archive.ics.uci.edu/dataset/851/steel+industry+energy+consumption

***Variables*:**

- Continuous variables-
  1. Date
  2. Usage_kWh - Industry Energy Consumption
  3. Lagging Current reactive power kVarh
  4. Leading Current reactive power kVarh
  5. CO2(tCO2)
  6. Lagging Current power factor
  7. Leading Current Power factor
  8. Number of Seconds from midnight
- Categorical variables-
  1. WeekStatus
  2. Day_of_week
  3. Load_Type

late	Usage_kWh	Lagging_Current_Reactive.Power_kVarh	Leading_Current_Reactive_Power_kVarh	CO2(tCO2)	Lagging_Current_Power_Factor	Leading_Current_Power_Factor	NSM	WeekStatus	Day_of_week	Load_Type
01-01-2018 00:15	3.17	2.95	0	0	73.21	100	900	Weekday	Monday	Light_Load
01-01-2018 00:30	4	4.46	0	0	66.77	100	1800	Weekday	Monday	Light_Load
01-01-2018 00:45	3.24	3.28	0	0	70.28	100	2700	Weekday	Monday	Light_Load
01-01-2018 01:00	3.31	3.56	0	0	68.09	100	3600	Weekday	Monday	Light_Load
01-01-2018 01:15	3.82	4.5	0	0	64.72	100	4500	Weekday	Monday	Light_Load
01-01-2018 01:30	3.28	3.56	0	0	67.76	100	5400	Weekday	Monday	Light_Load
01-01-2018 01:45	3.6	4.14	0	0	65.62	100	6300	Weekday	Monday	Light_Load
01-01-2018 02:00	3.6	4.28	0	0	64.37	100	7200	Weekday	Monday	Light_Load
01-01-2018 02:15	3.28	3.64	0	0	66.94	100	8100	Weekday	Monday	Light_Load
01-01-2018 02:30	3.78	4.72	0	0	62.51	100	9000	Weekday	Monday	Light_Load
01-01-2018 02:45	3.46	4.03	0	0	65.14	100	9900	Weekday	Monday	Light_Load
01-01-2018 03:00	3.24	3.64	0	0	66.49	100	10800	Weekday	Monday	Light_Load
01-01-2018 03:15	3.96	4.97	0	0	62.32	100	11700	Weekday	Monday	Light_Load
01-01-2018 03:30	3.31	3.74	0	0	66.27	100	12600	Weekday	Monday	Light_Load
01-01-2018 03:45	3.31	3.85	0	0	65.19	100	13500	Weekday	Monday	Light_Load
01-01-2018 04:00	3.89	5	0	0	61.4	100	14400	Weekday	Monday	Light_Load
01-01-2018 04:15	3.28	3.82	0	0	65.14	100	15300	Weekday	Monday	Light_Load
01-01-2018 04:30	3.56	4.28	0	0	63.95	100	16200	Weekday	Monday	Light_Load
01-01-2018 04:45	3.74	4.54	0	0	63.58	100	17100	Weekday	Monday	Light_Load
01-01-2018 05:00	3.31	3.6	0	0	67.68	100	18000	Weekday	Monday	Light_Load
01-01-2018 05:15	3.56	4.07	0	0	65.84	100	18900	Weekday	Monday	Light_Load
01-01-2018 05:30	3.56	4.1	0	0	65.56	100	19800	Weekday	Monday	Light_Load
01-01-2018 05:45	3.28	3.49	0	0	68.48	100	20700	Weekday	Monday	Light_Load
01-01-2018 06:00	3.78	4.32	0	0	65.85	100	21600	Weekday	Monday	Light_Load
01-01-2018 06:15	3.35	3.64	0	0	67.72	100	22500	Weekday	Monday	Light_Load
01-01-2018 06:30	3.24	3.35	0	0	69.52	100	23400	Weekday	Monday	Light_Load
01-01-2018 06:45	3.89	4.46	0	0	65.73	100	24300	Weekday	Monday	Light_Load
01-01-2018 07:00	3.31	3.53	0	0	68.4	100	25200	Weekday	Monday	Light_Load
01-01-2018 07:15	3.28	3.49	0	0	68.48	100	26100	Weekday	Monday	Light_Load

Data Cleaning :

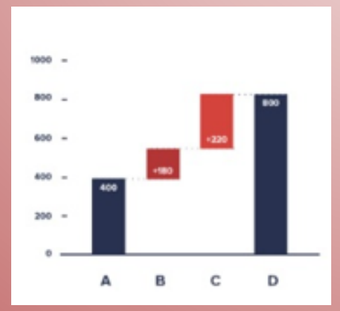
- The "day of week" and “data” column has been removed from the dataset as it is unnecessary for further analysis.

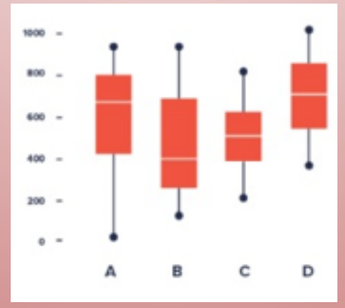
- Examined the dataset for both missing and unique values, it was found to be complete with no missing entries.

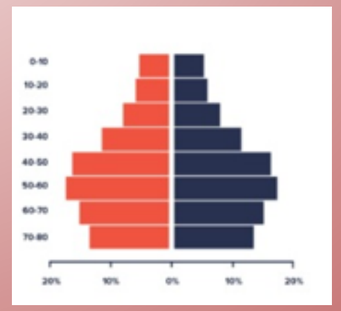
- Encoded and dummified the categorical variables using OneHotcoding.

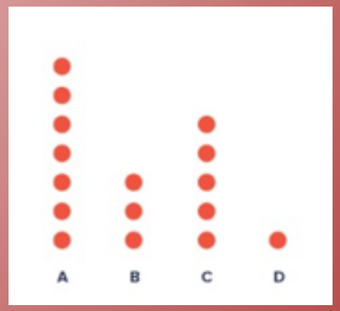
#	Column	Non-Null Count	Dtype
0	Usage_kWh	35040 non-null	float64
1	Lagging_Current_Reactive.Power_kVarh	35040 non-null	float64
2	Leading_Current_Reactive_Power_kVarh	35040 non-null	float64
3	CO2(tCO2)	35040 non-null	float64
4	Lagging_Current_Power_Factor	35040 non-null	float64
5	Leading_Current_Power_Factor	35040 non-null	float64
6	NSM	35040 non-null	int64
7	WeekStatus_Weekend	35040 non-null	int64
8	Load_Type_Maximum_Load	35040 non-null	int64
9	Load_Type_Medium_Load	35040 non-null	int64

Exploratory Data Analysis



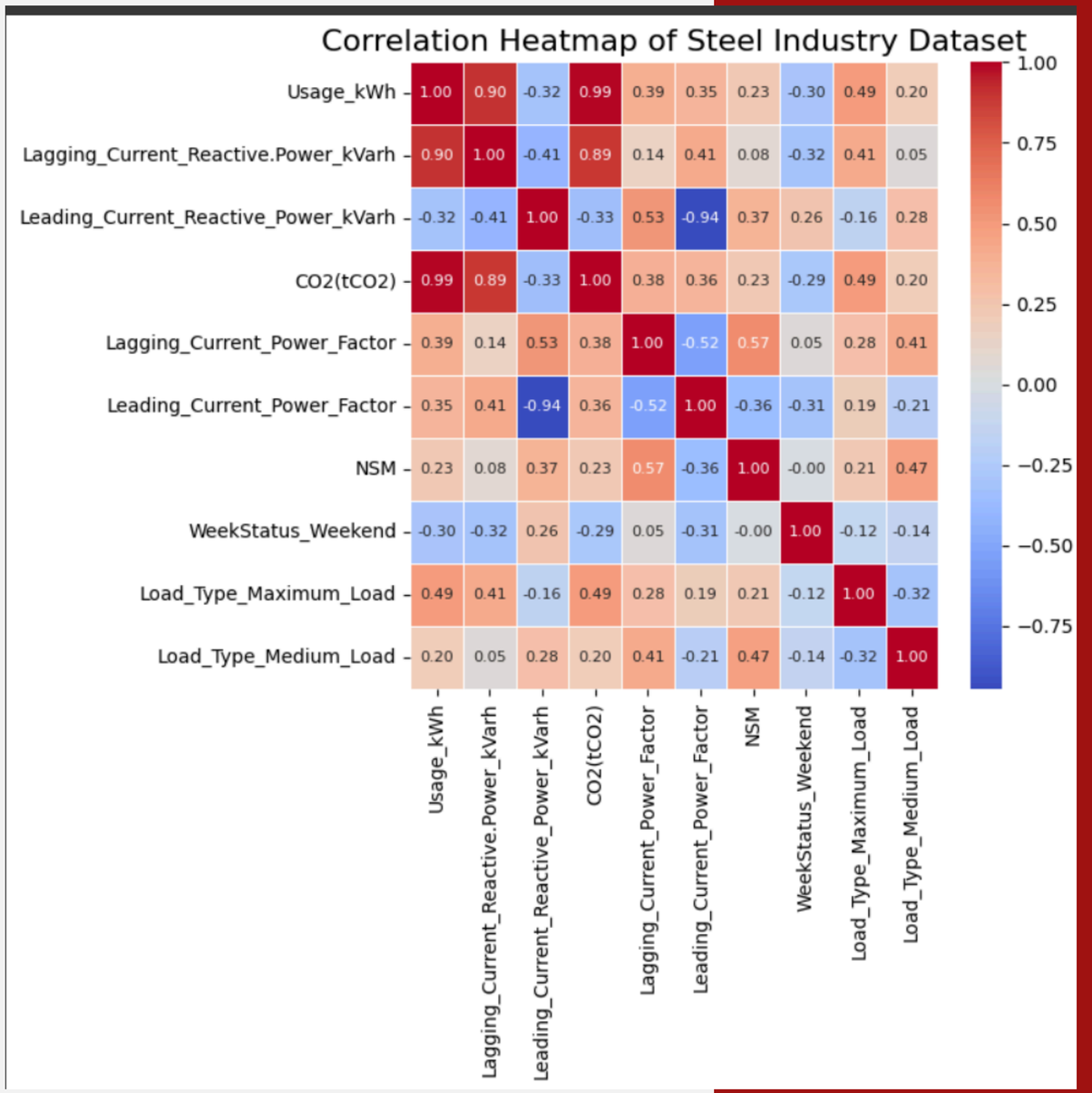




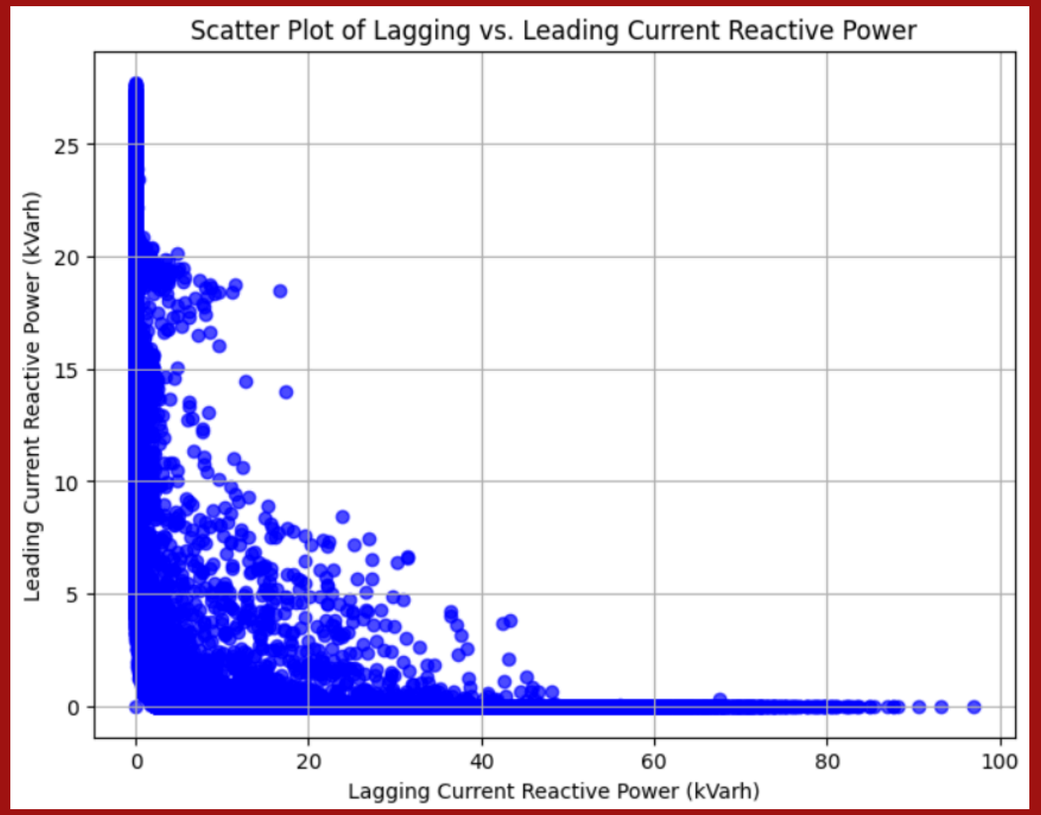


Correlation Matrix

- **Red Shades:** Positive correlation (closer to +1). This means that as one variable increases, the other tends to increase as well.
- **Blue Shades:** Negative correlation (closer to -1). This indicates that as one variable increases, the other tends to decrease.
- **White/Light Shades:** Weak or no correlation (around 0).

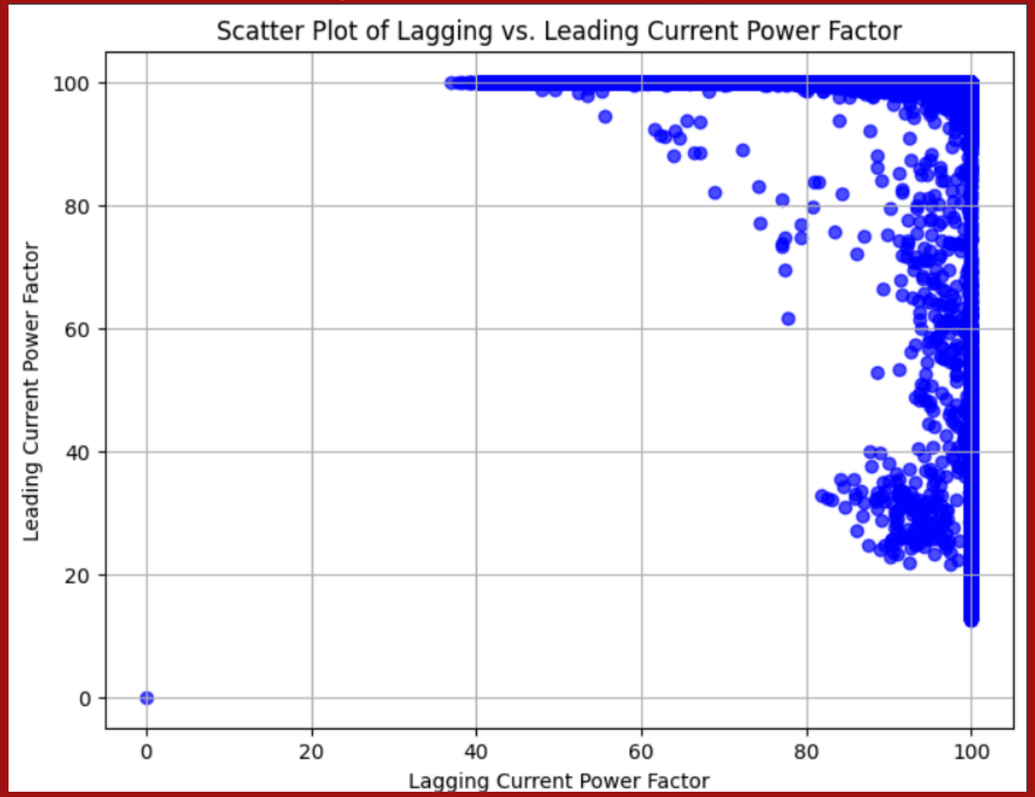


Scatter Plots



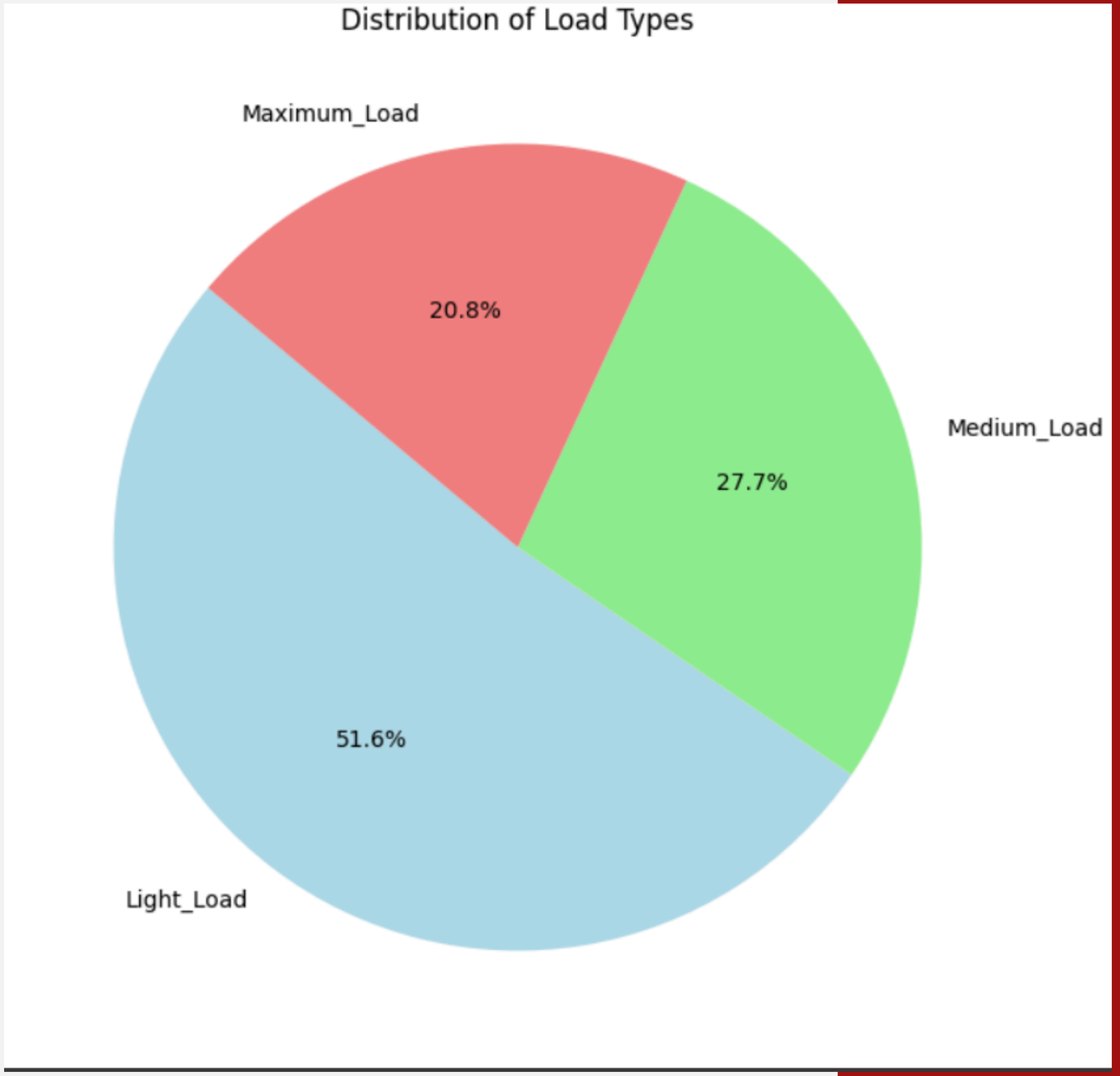
The scatter plot reveals a negative correlation where data points are concentrated at lower ranges.

The scatter plot indicates a dense cluster of high values for both lagging and leading power factors at the upper right, showing a nonlinear pattern.

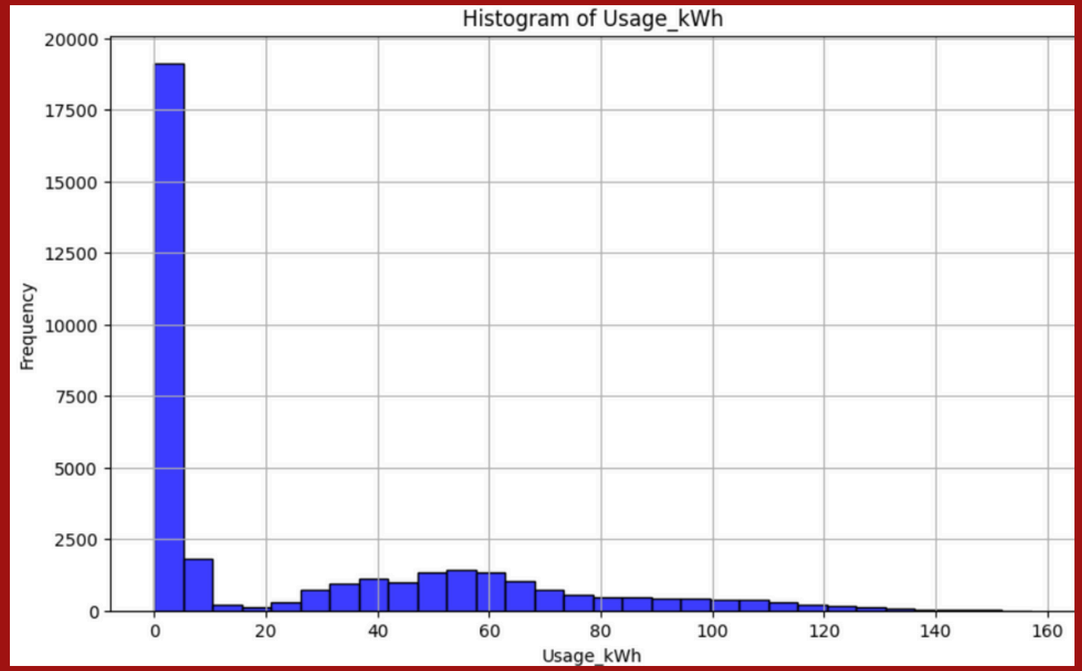


Pie chart

Light Load is the most prevalent, representing over half of the total. Medium Load follows, making up a little over a quarter. Maximum Load is the least common, accounting for about one-fifth of the distribution.

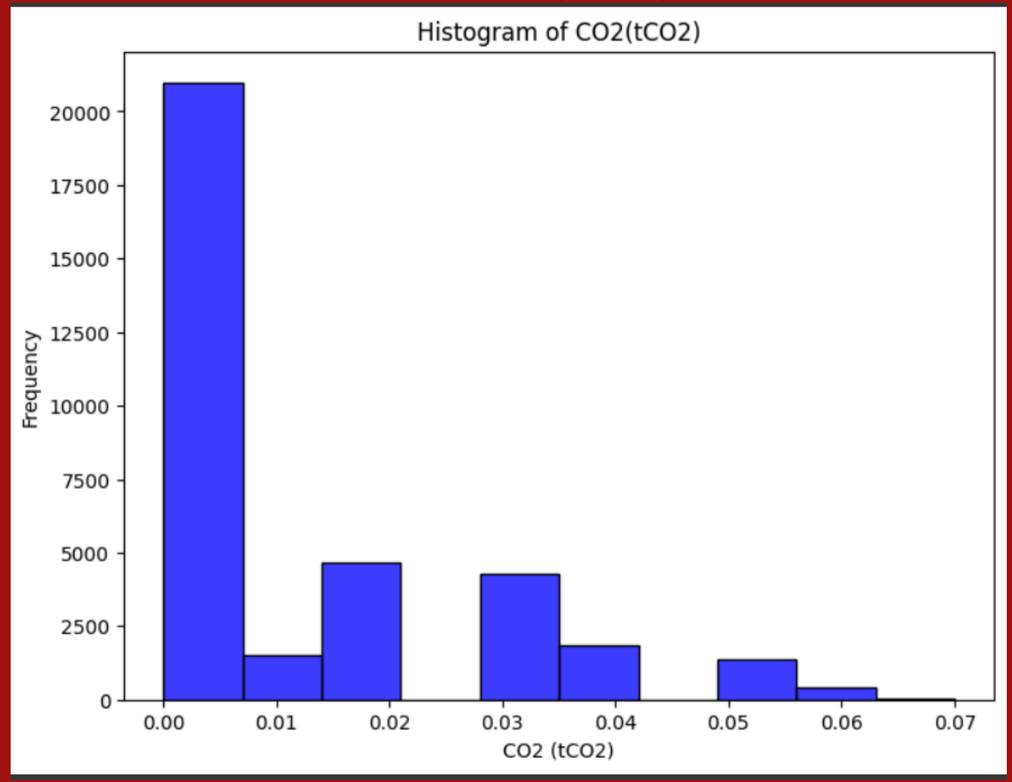


Histograms

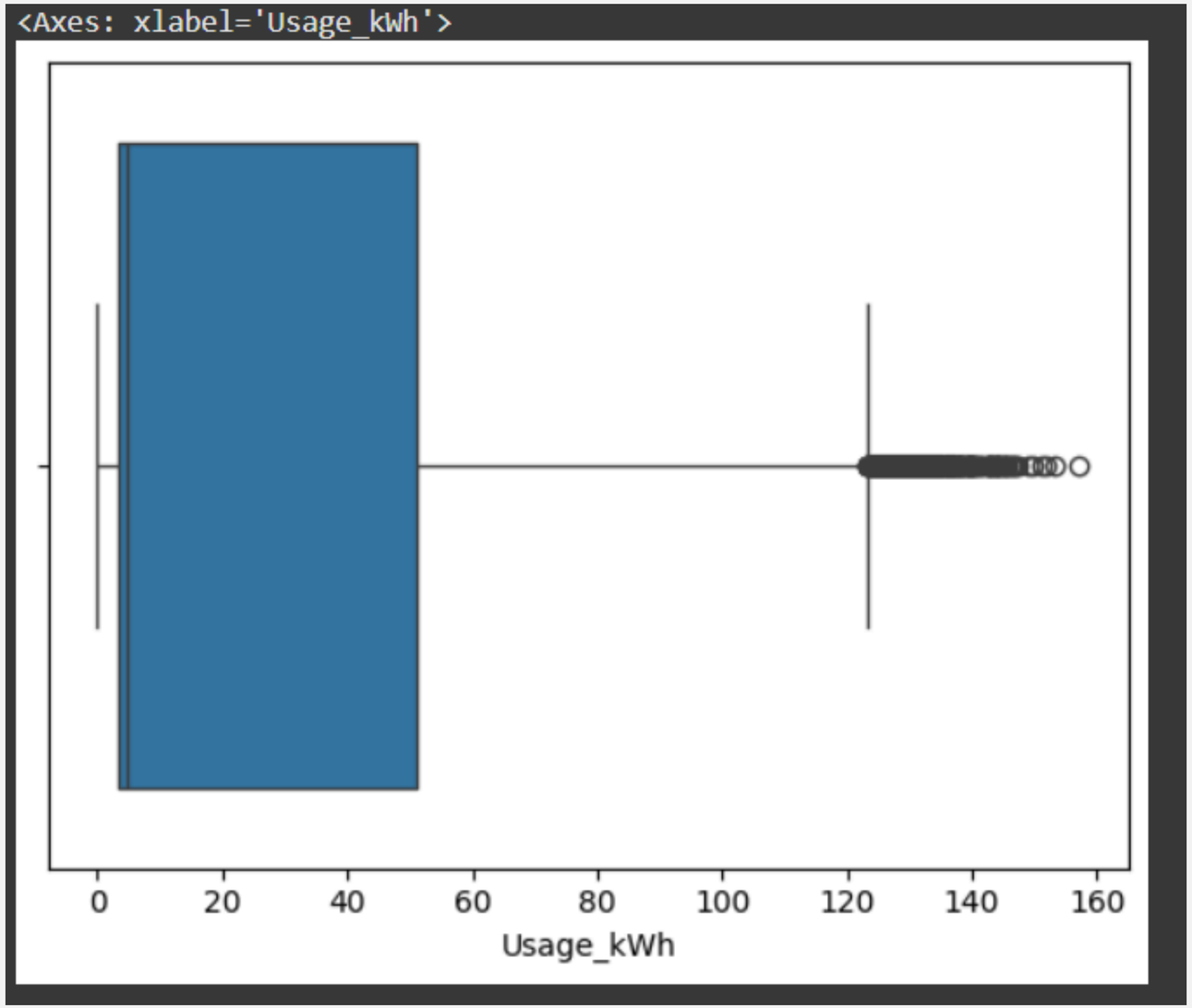


The data shows right skewness, indicating that most users have low energy usage while a smaller group has significantly higher consumption.

The histogram of CO2 emissions shows a right-skewed distribution, indicating that most entities emit minimal CO2, with only a few cases having higher emissions.



Box-Plot



This boxplot shows the distribution of the Usage_kWh variable. This suggests a positively skewed distribution. Most electricity usage is around 20 kWh, with values typically ranging from 0 to 40 kWh.

Significance Test

OLS Regression Results

Dep. Variable:	Usage_kWh	R-squared:	0.980
Model:	OLS	Adj. R-squared:	0.980
Method:	Least Squares	F-statistic:	1.946e+05
Date:	Thu, 07 Nov 2024	Prob (F-statistic):	0.00
Time:	17:54:12	Log-Likelihood:	-1.0382e+05
No. Observations:	35040	AIC:	2.077e+05
Df Residuals:	35030	BIC:	2.077e+05
Df Model:	9		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-10.8814	0.394	-27.632	0.000	-11.653	-10.110
Lagging_Current_Reactive.Power_kVarh	0.3096	0.004	78.792	0.000	0.302	0.317
Leading_Current_Reactive_Power_kVarh	0.0685	0.011	6.181	0.000	0.047	0.090
CO2(tCO2)	1665.8013	4.954	336.266	0.000	1656.092	1675.511
Lagging_Current_Power_Factor	0.1158	0.003	43.641	0.000	0.111	0.121
Leading_Current_Power_Factor	0.0572	0.003	19.736	0.000	0.052	0.063
NSM	-2.735e-06	1.39e-06	-1.961	0.050	-5.47e-06	-1.64e-09
WeekStatus_Weekend	0.0255	0.062	0.413	0.680	-0.095	0.146
Load_Type_Maximum_Load	1.2477	0.097	12.825	0.000	1.057	1.438
Load_Type_Medium_Load	1.6893	0.091	18.609	0.000	1.511	1.867

Omnibus:	41848.428	Durbin-Watson:	1.004
Prob(Omnibus):	0.000	Jarque-Bera (JB):	22400660.545
Skew:	5.821	Prob(JB):	0.00
Kurtosis:	126.318	Cond. No.	9.81e+06

The significance test results for all the features indicate that most features have statistically significant coefficients (p-value < 0.05), except for NSM and Week Status weekend, which are not significant contributors to the model.

Significance Test

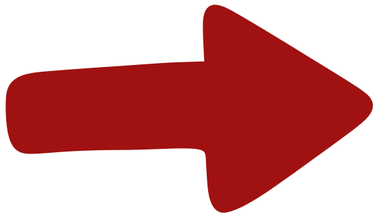
```
Columns in X_new: Index(['const', 'Lagging_Current_Reactive.Power_kVarh',
       'Leading_Current_Reactive_Power_kVarh', 'Leading_Current_Power_Factor',
       'WeekStatus_Weekend', 'Load_Type_Maximum_Load',
       'Load_Type_Medium_Load'],
      dtype='object')
                            OLS Regression Results
==============================================================================
Dep. Variable:              Usage_kWh   R-squared:                       0.872
Model:                            OLS   Adj. R-squared:                  0.872
Method:                 Least Squares   F-statistic:                 3.980e+04
Date:                Thu, 07 Nov 2024   Prob (F-statistic):               0.00
Time:                        17:54:18   Log-Likelihood:            -1.3668e+05
No. Observations:               35040   AIC:                         2.734e+05
Df Residuals:                   35033   BIC:                         2.734e+05
Df Model:                           6
Covariance Type:            nonrobust
=========================================================================================================
                                            coef    std err          t      P>|t|      [0.025      0.975]
---------------------------------------------------------------------------------------------------------
const                                    -6.5160      0.672     -9.694      0.000      -7.833      -5.199
Lagging_Current_Reactive.Power_kVarh      1.5707      0.005    312.718      0.000       1.561       1.581
Leading_Current_Reactive_Power_kVarh     -0.1104      0.028     -3.928      0.000      -0.166      -0.055
Leading_Current_Power_Factor              0.0410      0.007      6.129      0.000       0.028       0.054
WeekStatus_Weekend                        2.5132      0.157     16.036      0.000       2.206       2.820
Load_Type_Maximum_Load                   20.4376      0.193    106.052      0.000      20.060      20.815
Load_Type_Medium_Load                    19.6420      0.173    113.744      0.000      19.304      19.980
==============================================================================
Omnibus:                     4362.959   Durbin-Watson:                   0.411
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            11812.918
Skew:                           0.694   Prob(JB):                         0.00
Kurtosis:                       5.483   Cond. No.                         960.
==============================================================================
```

By using VIF feature selection, the model shows an R-squared value of 0.872, meaning 87.2% of the variance is explained by the selected features.
The p-value of 0.00 indicate that the model is statistically significant.

Multicollinearity Check

Variables with the greatest variance inflation factor
(VIF > 3) were removed

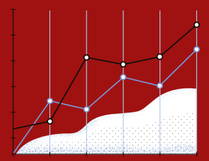
	variables	VIF
0	Lagging_Current_Reactive.Power_kVarh	9.3
1	Leading_Current_Reactive_Power_kVarh	7.9
2	CO2(tCO2)	12.4
3	Lagging_Current_Power_Factor	39.3
4	Leading_Current_Power_Factor	19.5
5	NSM	7.3
6	WeekStatus_Weekend	1.7
7	Load_Type_Maximum_Load	2.8
8	Load_Type_Medium_Load	3.3



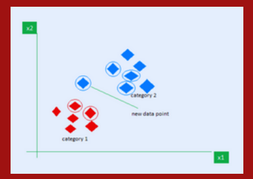
	variables	VIF
0	Lagging_Current_Reactive.Power_kVarh	2.6
1	Leading_Current_Reactive_Power_kVarh	1.8
2	Leading_Current_Power_Factor	2.8
3	WeekStatus_Weekend	1.6
4	Load_Type_Maximum_Load	1.8
5	Load_Type_Medium_Load	2.0

Machine Learning Algorithms

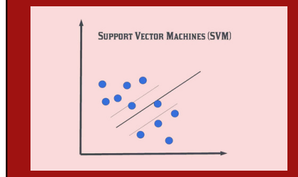
ML Algorithms



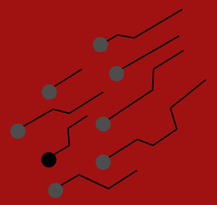
Linear Regression



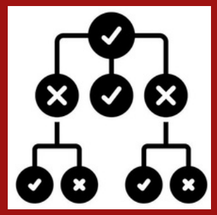
K-Nearest Neighbors (KNN)



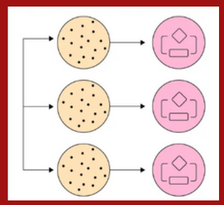
Support Vector Machine (SVM)



ANN



Decision Tree



Random Forest

Boosting

70:30 Train-Test split

Algorithms	Model-1 (r2)	Model-2 (r2)	Model-1 (MAE)	Model-2 (MAE)
Linear Regression	0.983	0.869	1.602	2.916
KNN	0.982	0.901	1.836	5.264
SVM	0.273	0.812	19.351	8.308
Decision Tree	0.998	0.870	0.519	5.665
Random Forest	0.999	0.896	0.324	5.167
XGBoost	0.999	0.917	0.480	4.748
AdaBoost	0.959	0.869	6.009	9.175
ANN	0.863	0.839	7.615	8.255

80:20 Train-Test split

Algorithms	Model-1 (r2)	Model-2 (r2)	Model-1 (MAE)	Model-2 (MAE)
Linear Regression	0.982	0.872	1.594	2.907
KNN	0.984	0.905	1.700	5.151
SVM	0.264	0.815	19.498	8.257
Decision Tree	0.999	0.880	0.484	5.429
Random Forest	0.999	0.903	0.305	4.994
XGBoost	0.999	0.921	0.473	4.652
AdaBoost	0.960	0.876	5.811	8.325
ANN	0.869	0.861	7.529	8.188

60:40 Train-Test split

Algorithms	Model-1 (r2)	Model-2 (r2)	Model-1 (MAE)	Model-2 (MAE)
Linear Regression	0.983	0.870	1.605	2.914
KNN	0.979	0.899	1.986	5.327
SVM	0.272	0.812	19.379	8.323
Decision Tree	0.998	0.869	0.584	5.729
Random Forest	0.999	0.895	0.349	5.231
XGBoost	0.999	0.918	0.479	4.721
AdaBoost	0.971	0.867	5.116	9.214
ANN	0.770	0.857	11.021	8.244

75:25 Train-Test split

Algorithms	Model-1 (r2)	Model-2 (r2)	Model-1 (MAE)	Model-2 (MAE)
Linear Regression	0.983	0.872	1.592	2.904
KNN	0.983	0.903	1.767	5.194
SVM	0.271	0.817	19.380	8.236
Decision Tree	0.999	0.876	0.496	5.514
Random Forest	0.999	0.899	0.309	5.079
XGBoost	0.999	0.919	0.473	4.674
AdaBoost	0.971	0.875	5.134	8.036
ANN	0.868	0.852	8.734	8.080

Algorithm Comparison

for the 60:40 ratio

Algorithms	r2 score	MAE
Linear Regression	0.982	1.592
KNN	0.984	1.700
SVM	0.815	8.236
Decision Tree	0.999	0.484
Random Forest	0.999	0.305
XGBoost	0.999	0.473
AdaBoost	0.960	5.116
ANN	0.869	7.529

Algorithm Comparison

for the 75:25 ratio

Algorithms	r2 score	MAE
Linear Regression	0.872	2.904
KNN	0.903	5.194
SVM	0.817	8.236
Decision Tree	0.876	5.514
Random Forest	0.899	5.079
XGBoost	0.919	4.674
AdaBoost	0.875	8.036
ANN	0.852	8.080

Summary

Summary

The primary objective of this research is to identify the most effective machine learning techniques for predicting energy consumption within the steel industry.

Among the regression models evaluated, the Random Forest (RF) model stands out by significantly enhancing the Mean Absolute Error (MAE) and R2R^2 score, surpassing the performance of other models considered in this study.

Linear Regression outperforms in terms of MAE in Model-2, the Random Forest model stands out as the highest and best-performing model overall, showcasing its superior ability to handle complex relationships and deliver better predictions.

Work Distribution

PULIGADDA VENKATA SAI KIRAN	Collect information about Steel Industry Energy Consumption & Literature Review
PULIGADDA VENKATA KRISHNA KIREETI	Data Preprocessing
RAMIDI HARISHWAR REDDY	Exploratory Data Analysis
R.SREE VARDHINI TULASI	Implement Machine Learning Algorithms

THANK YOU

Done by:

RAMIDI HARISHWAR REDDY

PULIGADDA VENKATA SAI KIRAN

PULIGADDA VENKATA KRISHNA KIREETI

R.SREE VARDHINI TULASI

APPENDIX



```
[1] from google.colab import files
    uploaded = files.upload()
```

```
Choose Files Steel_industry_data.csv
• Steel_industry_data.csv(text/csv) - 2731389 bytes, last modified: 8/25/2024 - 100% done
Saving Steel_industry_data.csv to Steel_industry_data.csv
```

```
[2] import numpy as np
    import pandas as pd
    data= pd.read_csv("/content/Steel_industry_data.csv")
```

```
[3] data.shape
```

```
(35040, 11)
```

```
[4] print(data.head())
```

```
                 date  Usage_kWh  Lagging_Current_Reactive.Power_kVarh  \
0  01/01/2018 00:15       3.17                                 2.95
1  01/01/2018 00:30       4.00                                 4.46
2  01/01/2018 00:45       3.24                                 3.28
3  01/01/2018 01:00       3.31                                 3.56
4  01/01/2018 01:15       3.82                                 4.50

   Leading_Current_Reactive_Power_kVarh  CO2(tCO2)  \
0                                   0.0        0.0
```

```
[5] data.columns #no. of columns
```

```
Index(['date', 'Usage_kWh', 'Lagging_Current_Reactive.Power_kVarh',
       'Leading_Current_Reactive_Power_kVarh', 'CO2(tCO2)',
       'Lagging_Current_Power_Factor', 'Leading_Current_Power_Factor', 'NSM',
       'WeekStatus', 'Day_of_week', 'Load_Type'],
      dtype='object')
```

```
[6] data = data.drop(['date'], axis=1)
    data = data.drop(['Day_of_week'], axis=1)
```

```
[7] print(data.head())
```

3.Checking of null values

```
[10] data.isna().sum() #no. of null values
```

	0
Usage_kWh	0
Lagging_Current_Reactive.Power_kVarh	0
Leading_Current_Reactive_Power_kVarh	0
CO2(tCO2)	0
Lagging_Current_Power_Factor	0
Leading_Current_Power_Factor	0
NSM	0
WeekStatus	0
Load_Type	0

dtype: int64

5.Finding out unique values for each column and counting them

```
[12] data.shape[1]
```

```
9
```

```
[13] for i in range(data.shape[1]):   #The for i in range(data.shape[1]): loop iterates over the columns of a DataFrame or a 2D a
       print(i)
```

```
0
1
2
3
4
5
6
7
8
```

```
[14] # Iterate over each column in the dataset and print unique values along with their value counts
     for i in range(data.shape[1]):
       print(data.iloc[:,i].unique())
       print(data.iloc[:,i].value_counts())
```

```
100.00    7474
99.99      152
99.98       87
70.71       81
```

7.Converting categorical into numercial variables

One-Hot encoding

```
# List of categorical columns
categorical_columns = ['WeekStatus', 'Load_Type']

# Apply dummy variable encoding (one-hot encoding with drop_first=True)
data_encoded = pd.get_dummies(data, columns=categorical_columns, drop_first=True)

# Convert 'yes' and 'no' to 1 and 0
cols_to_convert = data_encoded.columns.difference(['CO2(tCO2)','Usage_kWh', 'Lagging_Current_Reactive.Power_kVarh',
       'Leading_Current_Reactive_Power_kVarh','Lagging_Current_Power_Factor', 'Leading_Current_Power_Factor'])
data_encoded[cols_to_convert] = data_encoded[cols_to_convert].replace({'yes': 1, 'no': 0})

# Convert only relevant boolean or object columns to integers
cols_to_convert = data_encoded.columns.difference(['CO2(tCO2)','Usage_kWh', 'Lagging_Current_Reactive.Power_kVarh',
       'Leading_Current_Reactive_Power_kVarh','Lagging_Current_Power_Factor', 'Leading_Current_Power_Factor'])
data_encoded[cols_to_convert] = data_encoded[cols_to_convert].astype(int)

# Display the first few rows of the dummy variable encoded dataset
print(data_encoded.head())
```

```
   Usage_kWh  Lagging_Current_Reactive.Power_kVarh  \
0       3.17                                 2.95
1       4.00                                 4.46
2       3.24                                 3.28
3       3.31                                 3.56
4       3.82                                 4.50
```

Independent and dependent :

```
[22] X=data_encoded.drop(['Usage_kWh'],axis=1)

     print(X)
     y = data_encoded['Usage_kWh']
     print(y)
```

```
35035                                 4.86
35036                                 3.74
35037                                 3.17
35038                                 3.06
35039                                 3.02

       Leading_Current_Reactive_Power_kVarh  CO2(tCO2)  \
0                                      0.00        0.0
1                                      0.00        0.0
2                                      0.00        0.0
3                                      0.00        0.0
4                                      0.00        0.0
...                                     ...        ...
35035                                  0.00        0.0
35036                                  0.00        0.0
35037                                  0.07        0.0
```

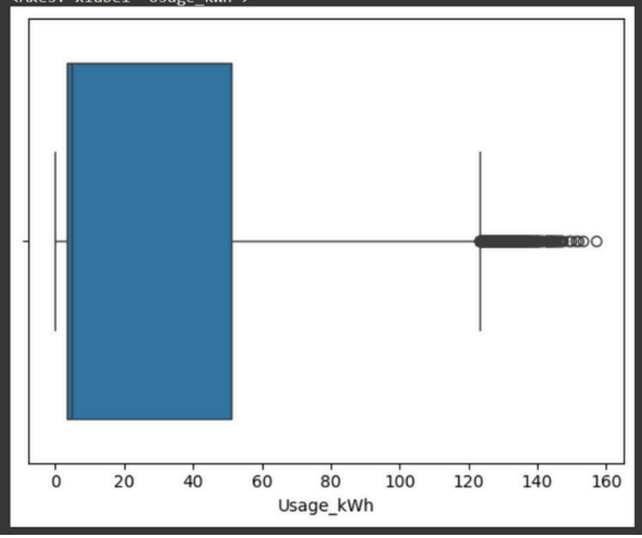
Heatmap : : Heatmaps are used to highlight areas of interest or intensity within a dataset. They are particularly useful for identifying trends, correlations, and outliers.

```
[24] # Calculate the correlation matrix
     corr_matrix = data_encoded.corr()

     # Set a larger plot size
     plt.figure(figsize=(8, 8))
     # Create the heatmap with smaller annotation font size
     sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5,
                 annot_kws={"size": 8})
     #heatmap - coorelation by using an heatmap
     #annot=True: Annotates each cell with the correlation coefficient.
     #cmap='coolwarm': Uses the 'coolwarm' colormap to color the heatmap.
     #fmt='.2f': Formats the annotation text to two decimal places.
     #linewidths=0.5: Sets the width of the lines that divide the cells.
     #annot_kws={"size": 8}: Sets the font size of the annotations to 8


     # Add a title with increased font size for better visibility
     plt.title('Correlation Heatmap of Steel Industry Dataset', fontsize=16)

     # Show the plot with tight layout for better spacing
     plt.tight_layout()   #Adjusts the padding between and around subplots to minimiz
     plt.show()
```

Box plot : A box plot (also known as a box-and-whisker plot) is a graphical representation used to display the distribution of a dataset. It provides a visual summary of several important aspects of the data, including its central tendency, variability, and skewness.

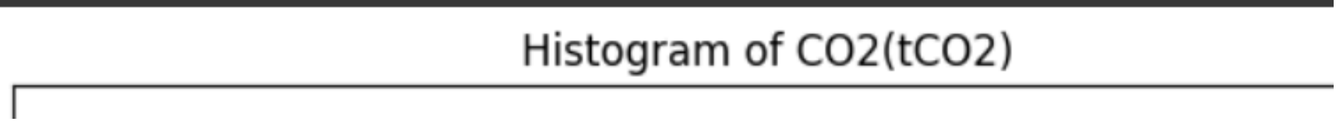
```
[25] sns.boxplot(x=data_encoded['Usage_kWh'])
```

```
<Axes: xlabel='Usage_kWh'>
```



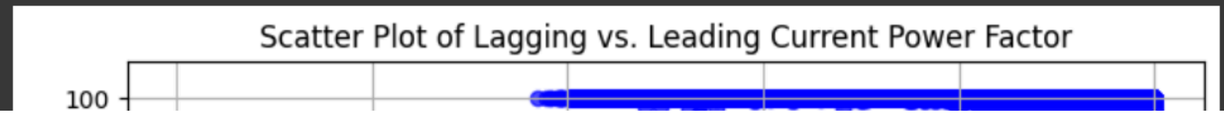
```
[26] # create the histogram using seaborn
     plt.figure(figsize=(10, 6))
     sns.histplot(y, bins=30, kde=False, color='blue')
     #bins parameter specifies the number of bins (intervals) into which the data is divided.
     #kde=False means that the Kernel Density Estimate plot will not be included in the histogram.
     plt.xlabel('Usage_kWh')
     plt.ylabel('Frequency')
     plt.title(f'Histogram of Usage_kWh')
     plt.grid(True)
     plt.show()
```

```
[27] # Assuming your dataset is stored in 'df'
     plt.figure(figsize=(8, 6))
     sns.histplot(data_encoded['CO2(tCO2)'], kde=False, bins=10, color='blue')
     plt.title('Histogram of CO2(tCO2)')
     plt.xlabel('CO2 (tCO2)')
     plt.ylabel('Frequency')
     plt.show()
```



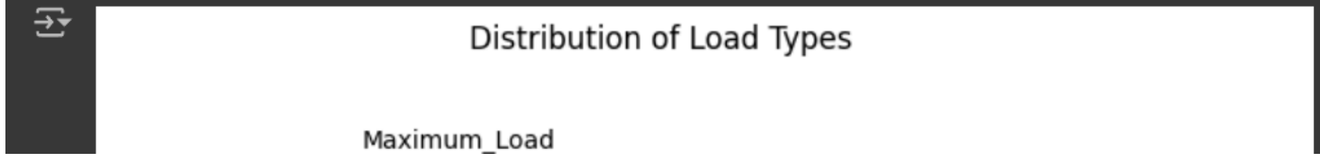
```
[28] # Create a scatter plot
     plt.figure(figsize=(8, 6))
     plt.scatter(data['Lagging_Current_Reactive.Power_kVarh'], #Fixed typo in the column name from "Lagging_Current_Reactive_Power
                 data['Leading_Current_Reactive_Power_kVarh'],
                 color='blue',
                 alpha=0.7)

     # Add titles and labels
     plt.title('Scatter Plot of Lagging vs. Leading Current Reactive Power')
     plt.xlabel('Lagging Current Reactive Power (kVarh)')
     plt.ylabel('Leading Current Reactive Power (kVarh)')
     plt.grid(True)

     # Show the plot
     plt.show()
```

```
[29] # Plotting the scatter plot
     plt.figure(figsize=(8,6))
     plt.scatter(data['Lagging_Current_Power_Factor'], data['Leading_Current_Power_Factor'], color='blue', alpha=0.7)

     # Adding title and labels
     plt.title('Scatter Plot of Lagging vs. Leading Current Power Factor')
     plt.xlabel('Lagging Current Power Factor')
     plt.ylabel('Leading Current Power Factor')

     # Displaying the plot
     plt.grid(True)
     plt.show()
```



Pie-chart : A pie chart is a circular graph divided into slices to illustrate numerical proportions.

```
[30] # Count the occurrences of each Load_Type
     load_counts = data['Load_Type'].value_counts()

     # Create the pie chart
     plt.figure(figsize=(8, 8))
     plt.pie(load_counts, labels=load_counts.index, autopct='%1.1f%%', startangle=140, colors=['lightblue', 'lightgreen', 'lightc

     # Add a title
     plt.title('Distribution of Load Types')

     # Show the plot
     plt.show()
```



splitting in different ratios

```
[32] from sklearn.model_selection import train_test_split
     X_train1, X_test1, y_train1, y_test1 = train_test_split(X, y, test_size=0.30, train_size=0.70, random_state=1)
     X_train2, X_test2, y_train2, y_test2 = train_test_split(X, y, test_size=0.20, train_size=0.80, random_state=1)
     X_train3, X_test3, y_train3, y_test3 = train_test_split(X, y, test_size=0.40, train_size=0.60 , random_state=1)
     X_train4, X_test4, y_train4, y_test4 = train_test_split(X, y, test_size=0.25, train_size=0.75 , random_state=1)
```

Splliting the dataset after the feature selection

```
[73] from sklearn.model_selection import train_test_split
     X_train5, X_test5, y_train5, y_test5 = train_test_split(X_new, y, test_size=0.30, train_size=0.70, random_state=1)
     X_train6, X_test6, y_train6, y_test6 = train_test_split(X_new, y, test_size=0.20, train_size=0.80, random_state=1)
     X_train7, X_test7, y_train7, y_test7 = train_test_split(X_new, y, test_size=0.40, train_size=0.60, random_state=1)
     X_train8, X_test8, y_train8, y_test8 = train_test_split(X_new, y, test_size=0.25, train_size=0.75, random_state=1)
```

Linear regression

Feature selection

+ Code + Text

[65]
```
# create X and y

X=data_encoded.drop(['Usage_kWh'],axis=1)
y = data_encoded['Usage_kWh']
```

[66]
```
# Import library for VIF
from statsmodels.stats.outliers_influence import variance_inflation_factor

def calc_vif(X):

    # Calculating VIF
    vif = pd.DataFrame()
    vif["variables"] = X.columns
    vif["VIF"] = [variance_inflation_factor(X.values, i).round(1) for i in range(X.shape[1])]

    return(vif)

calc_vif(X)
```

Linear Regression before feature selection

```
[ ]  # imports
     import pandas as pd
     import seaborn as sns
     from sklearn.linear_model import LinearRegression
     from sklearn import metrics
     import numpy as np
```

```
[ ]  #70 - 30

     lm2 = LinearRegression()
     lm2.fit(X_train1, y_train1)
     y_pred1 = lm2.predict(X_test1)
     print(np.sqrt(metrics.mean_absolute_error(y_test1, y_pred1)))
```

```
1.6024151506252082
```

```
[ ]  #evaluarion metric
     from sklearn.metrics import r2_score
     r2_score(y_test1,y_pred1)
```

```
0.9832672446990494
```

KNN

70 - 30 Ratio

```
[ ] from sklearn.neighbors import KNeighborsRegressor
```

```
[ ] model=KNeighborsRegressor(n_neighbors=5)
```

```
[ ] model.fit(X_train1, y_train1)
```

```
KNeighborsRegressor
KNeighborsRegressor()
```

```
[ ] y_pred1 = model.predict(X_test1)
    y_pred1
```

```
array([3.888, 3.874, 3.636, ..., 3.89 , 5.016, 3.904])
```

```
[ ] knn = pd.DataFrame({'Predicted':y_pred1,'Actual':y_test1})
    knn
```

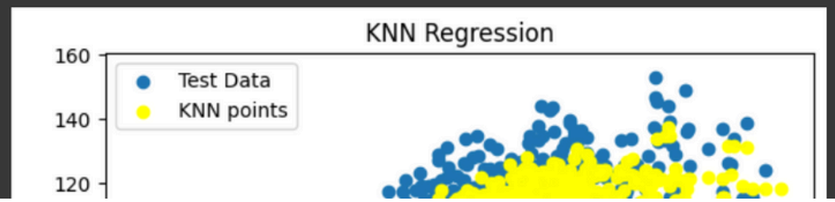
```
[ ] #evaluarion metric
    from sklearn.metrics import r2_score
    r2_score(y_test1,y_pred1)
```

```
0.981859390094929
```

```
[ ] from sklearn import metrics
    metrics.mean_absolute_error(y_test1,y_pred1)
```

```
1.8363369482496195
```

```
[ ] plt.scatter(X_test1['Lagging_Current_Reactive.Power_kVarh'], y_test1, label='Test Data')
    plt.scatter(X_test1['Lagging_Current_Reactive.Power_kVarh'], y_pred1, color='yellow', linewidth=1, label='KNN points')
    plt.xlabel('Lagging_Current_Reactive.Power_kVarh')
    plt.ylabel('Usage_kWh')
    plt.title('KNN Regression')
    plt.legend()
    plt.show()
```



SVM

```
[ ] from sklearn.svm import SVR
```

```
[ ] model = SVR(kernel='rbf')
```

```
[ ] model.fit(X_train1, y_train1)
```

```
SVR
SVR()
```

```
[ ] y_pred1 = model.predict(X_test1)
    y_pred1
```

```
array([ 5.59112244, 12.39636655,  3.05005709, ..., -0.36030282,
       28.12271563, 35.48718502])
```

```
[ ] svm = pd.DataFrame({'Predicted':y_pred1,'Actual':y_test1})
    svm
```

	Predicted	Actual
5760	5.591122	3.20
14294	12.396367	3.78
35035	3.050057	3.85
30292	47.265328	43.81

```
[ ] from sklearn.metrics import r2_score
    r2_score(y_test1,y_pred1)
```

```
0.2726758720886666
```

```
[ ] from sklearn import metrics
    metrics.mean_absolute_error(y_test1,y_pred1)
```

```
19.350823673657654
```

```
[ ] plt.scatter(X_test1['Lagging_Current_Reactive.Power_kVarh'], y_test1, label='Test Data')
    plt.scatter(X_test1['Lagging_Current_Reactive.Power_kVarh'], y_pred1, color='yellow', linewidth=1, label='SVM points')
    plt.xlabel('Lagging_Current_Reactive.Power_kVarh')
    plt.ylabel('Usage_kWh')
    plt.title('SVM Regression')
    plt.legend()
    plt.show()
```

DECISION TREE

```
[ ] from sklearn.tree import DecisionTreeRegressor
    from sklearn import metrics
```

Loading...

```
    clf = DecisionTreeRegressor()

    # Train Decision Tree Classifer
    clf = clf.fit(X_train1,y_train1)
```

```
[ ] y_pred1 = clf.predict(X_test1)
    y_pred1
```

```
array([3.2 , 3.78, 3.82, ..., 3.96, 4.97, 5.33])
```

```
[ ] dr = pd.DataFrame({'Predicted':y_pred1,'Actual':y_test1})
    dr
```

	Predicted	Actual
5760	3.20	3.20

10512 rows × 2 columns

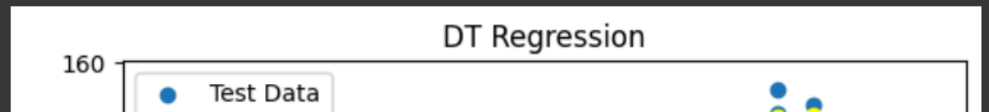
```
[ ] #Evaluation Metric
    from sklearn.metrics import r2_score
    r2_score(y_test1,y_pred1)
```

```
0.9983517328917512
```

```
[ ] from sklearn import metrics
    metrics.mean_absolute_error(y_test1,y_pred1)
```

```
0.5188984018264841
```

```
[ ] plt.scatter(X_test1['Lagging_Current_Reactive.Power_kVarh'], y_test1, label='Test Data')
    plt.scatter(X_test1['Lagging_Current_Reactive.Power_kVarh'], y_pred1, color='yellow', linewidth=1, label='DT points'
    plt.xlabel('Lagging_Current_Reactive.Power_kVarh')
    plt.ylabel('Usage_kWh')
    plt.title('DT Regression')
    plt.legend()
    plt.show()
```



RANDOM FOREST

70 - 30 ratio

```
[ ] from sklearn.ensemble import RandomForestRegressor
```

```
[ ] rf = RandomForestRegressor()
```

```
[ ] rf.fit(X_train1, y_train1)
```

```
RandomForestRegressor
RandomForestRegressor()
```

```
[ ] y_pred1 = rf.predict(X_test1)
    y_pred1
```

```
array([3.2053, 3.7739, 3.8314, ..., 3.9536, 4.9343, 5.3429])
```

```
[ ] r = pd.DataFrame({'Predicted':y_pred1,'Actual':y_test1})
    r
```

	Predicted	Actual

```
[ ] #Evaluation Metric
    from sklearn.metrics import r2_score
    r2_score(y_test1,y_pred1)
```

```
0.999164923679201
```

```
[ ] from sklearn import metrics
    metrics.mean_absolute_error(y_test1,y_pred1)
```

```
0.3244233066971081
```

```
[ ] plt.scatter(X_test1['Lagging_Current_Reactive.Power_kVarh'], y_test1, label='Test Data')
    plt.scatter(X_test1['Lagging_Current_Reactive.Power_kVarh'], y_pred1, color='yellow', linewidth=1, label='RF points')
    plt.xlabel('Lagging_Current_Reactive.Power_kVarh')
    plt.ylabel('Usage_kWh')
    plt.title('RF Regression')
    plt.legend()
    plt.show()
```

XG BOOST

XG **boost** 70 - 30 ratio

```
[ ]  import xgboost as xgb
```

```
[ ]  model = xgb.XGBRegressor()

     train_model = model.fit(X_train1, y_train1)
```

```
[ ]  y_pred1 = train_model.predict(X_test1)
     y_pred1
```

```
array([3.2505913, 4.0345516, 3.4650652, ..., 3.8175178, 4.951982 ,
       5.1443043], dtype=float32)
```

```
[ ]  xg = pd.DataFrame({'Predicted':y_pred1,'Actual':y_test1})
     xg
```

	Predicted	Actual
5760	3.250591	3.20
14294	4.034552	3.78

```
[ ]  #Evaluation Metric
     from sklearn.metrics import r2_score
     r2_score(y_test1,y_pred1)
```

```
0.9990195746151693
```

```
[ ]  from sklearn import metrics
     metrics.mean_absolute_error(y_test1,y_pred1)
```

```
0.48032607259997134
```

```
[ ]  plt.scatter(X_test1['Lagging_Current_Reactive.Power_kVarh'], y_test1, label='Test Data')
     plt.scatter(X_test1['Lagging_Current_Reactive.Power_kVarh'], y_pred1, color='yellow', linewidth=1, label='XG points')
     plt.xlabel('Lagging_Current_Reactive.Power_kVarh')
     plt.ylabel('Usage_kWh')
     plt.title('XG Regression')
     plt.legend()
     plt.show()
```

ADA BOOST

```
[ ]  from sklearn.ensemble import AdaBoostRegressor

     # Initialize the AdaBoost classifier
     model = AdaBoostRegressor(n_estimators=50, random_state=42)

     # Fit the model on the training data
     model.fit(X_train1, y_train1)

     # Predict on the test data
     y_pred1 = model.predict(X_test1)
```

```
[ ]  ab = pd.DataFrame({'Predicted':y_pred1,'Actual':y_test1})
     ab
```

	Predicted	Actual
5760	8.853814	3.20
14294	8.853814	3.78
35035	12.083785	3.85

10512 rows × 2 columns

```
[ ]  #Evaluation Metric
     from sklearn.metrics import r2_score
     r2_score(y_test1,y_pred1)
```

```
0.9595932754780265
```

```
[ ]  from sklearn import metrics
     metrics.mean_absolute_error(y_test1,y_pred1)
```

```
6.008528524048208
```

```
[ ]  plt.scatter(X_test1['Lagging_Current_Reactive.Power_kVarh'], y_test1, label='Test Data')
     plt.scatter(X_test1['Lagging_Current_Reactive.Power_kVarh'], y_pred1, color='yellow', linewidth=1, label='ADA points')
     plt.xlabel('Lagging_Current_Reactive.Power_kVarh')
     plt.ylabel('Usage_kWh')
     plt.title('ADA Regression')
     plt.legend()
     plt.show()
```

```
[35] import tensorflow as tf
     import matplotlib.pyplot as plt
```

70 - 30 ratio

```
tf.random.set_seed(42)

# STEP1: Creating the model

model= tf.keras.Sequential([
                            tf.keras.layers.Dense(18),

                            tf.keras.layers.Dense(15),
 tf.keras.layers.Dense(12),
                            tf.keras.layers.Dense(6),
                            tf.keras.layers.Dense(3),
                            tf.keras.layers.Dense(1)

])

# STEP2: Compiling the model

model.compile(loss= tf.keras.losses.mae,
              optimizer= tf.keras.optimizers.Adam(),
              metrics= ["mae"])

# STEP3: Fit the model

history= model.fit(X_train1, y_train1, epochs= 50, verbose=1)
```

```
[37] model.evaluate(X_test1, y_test1)
```

```
329/329 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 7.6153 - mae: 7.6153
[7.6728515625, 7.6728515625]
```

```
[38] y_pred1 = model.predict(X_test1)
     y_pred1 = y_pred1.flatten()
```

```
329/329 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step
```

```
[39] from sklearn.metrics import r2_score
     r2 = r2_score(y_test1, y_pred1)
     print(f'R² Score: {r2}')
```

```
R² Score: 0.8635913007591117
```

```
[40] ann = pd.DataFrame({'Predicted':y_pred1,'Actual':y_test1})
     ann
```

	Predicted	Actual
5760	3.584916	3.20
14294	9.570083	3.78

```
[41] model.summary();
```

Model: "sequential"

Layer (type)	Output Shape
dense (Dense)	(None, 18)
dense_1 (Dense)	(None, 15)
dense_2 (Dense)	(None, 12)
dense_3 (Dense)	(None, 6)
dense_4 (Dense)	(None, 3)
dense_5 (Dense)	(None, 1)

Total params: 2,282 (8.92 KB)
Trainable params: 760 (2.97 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 1,522 (5.95 KB)

```
[42] pd.DataFrame(history.history).plot()
     plt.ylabel("loss")
     plt.xlabel("epochs")
```

ANN