STEEL INDUSTRY ENERGY CONSUMPTION PREDICTION

BHAVANS VIVEKANANDA COLLEGE OF SCIENCE, HUMANITIES AND COMMERCE

Presented by:-
Group 7

Priyanka Parthasarathy Sai Phaneendra Subhasish Choudhury Rishi Raj Singh

ABSTRACT

This project aims to build a cutting-edge predictive model that anticipates energy consumption patterns in the steel industry, paving the way for greener and more cost-effective operations. By leveraging predictive insights, the model empowers steel manufacturers to optimize energy use, reduce waste, and promote sustainable industrial practices.

OBJECTIVE

To develop a comprehensive machine learning framework to optimize energy consumption and reduce carbon emissions across the entire steel production process.

CONTENT

Introduction

- Steel plays an indispensable role in building economies, from towering skyscrapers to bridges and railways. The steel industry’s growth often mirrors a nation’s economic health and infrastructural progress

- With the rise of automation and digital innovation, the steel industry is evolving rapidly, enhancing production precision, boosting efficiency, and setting new standards for quality.

LITERATURE REVIEW

Literature Review 1:

Swan L.G. Ugursal

This 2009 paper in *Renewable and Sustainable Energy Reviews* examines methods for modeling energy consumption in the residential sector, categorizing various techniques and their applications.

The review covers statistical, engineering, and hybrid models, analyzing their effectiveness in predicting energy use patterns and supporting policy-making. The authors emphasize the importance of accurate modeling to address energy efficiency and inform sustainable energy strategies in residential settings.

Literature Review 2:

Karthick Dharmaprakash Sathya

This 2024 study leverages CatBoost regression to predict energy consumption in the steel industry, addressing the sector's need for sustainable energy management. CatBoost is selected for its capability to handle complex, non-linear relationships and large datasets, making it ideal for capturing intricate energy patterns in steel production.

Using historical data from steel plants, the model provides precise predictions of energy use across different production stages, helping industry stakeholders optimize energy efficiency and reduce environmental impact.

DATA PREPROCESSING

About the dataset

- The Steel Industry Energy Consumption Dataset from the UCI Machine Learning Repository provides a comprehensive dataset for studying energy consumption patterns within the steel industry. This dataset is crucial for developing predictive models that help understand and potentially reduce energy usage in one of the most energy-intensive manufacturing sectors.

- The dataset contains various operational parameters and environmental conditions that influence energy consumption in steel manufacturing. By analyzing these features, machine learning models can predict energy consumption with high accuracy, enabling data-driven decisions for energy management.

Data

Dataset: Our Dataset consists of 11 variable and 1800 records

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

DataVariables:

Categorical variables	Continuous variables
Date	Lagging_Current_Reactive.Power_kVarh
WeekStatus	Leading_Current_Reactive.Power_kVarh
LoadType	Lagging_Current_Power_Factor_kVarh
Day_of_week	Leading_Current_Power_Factor_kVarh
	CO2(tCO2)
	NSM
	Usage_kVarh

date	Usage_kWh	Lagging_Cu	Leadin	CO2(	Lagging	Leading	NSM	WeekStat	Day_of_	Load_Type
########	3.17	2.95	0	0	73.21	100	900	Weekday	Monday	Light_Load
########	4	4.46	0	0	66.77	100	1800	Weekday	Monday	Light_Load
########	3.24	3.28	0	0	70.28	100	2700	Weekday	Monday	Light_Load
########	3.31	3.56	0	0	68.09	100	3600	Weekday	Monday	Light_Load
########	3.82	4.5	0	0	64.72	100	4500	Weekday	Monday	Light_Load
########	3.28	3.56	0	0	67.76	100	5400	Weekday	Monday	Light_Load
########	3.6	4.14	0	0	65.62	100	6300	Weekday	Monday	Light_Load
########	3.6	4.28	0	0	64.37	100	7200	Weekday	Monday	Light_Load
########	3.28	3.64	0	0	66.94	100	8100	Weekday	Monday	Light_Load
########	3.78	4.72	0	0	62.51	100	9000	Weekday	Monday	Light_Load

DATA CLEANING

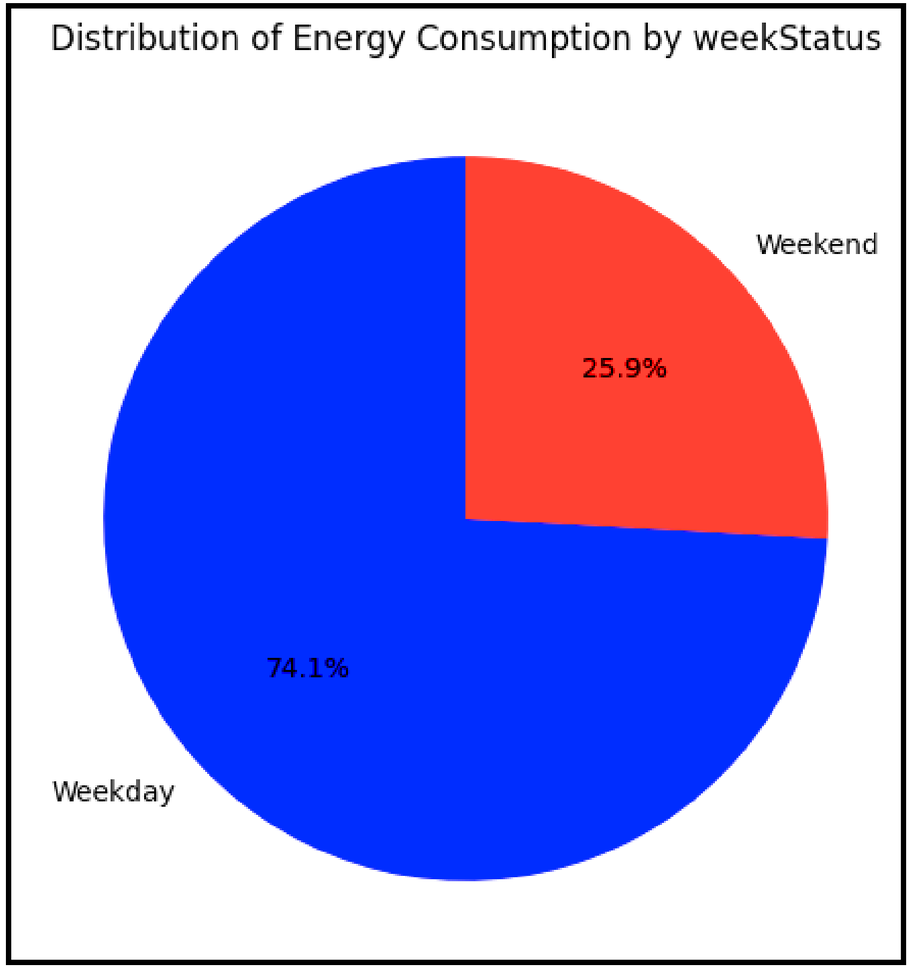
Removing Columns: To avoid confusion from duplicate timestamps on identical dates, we removed the 'date' column.

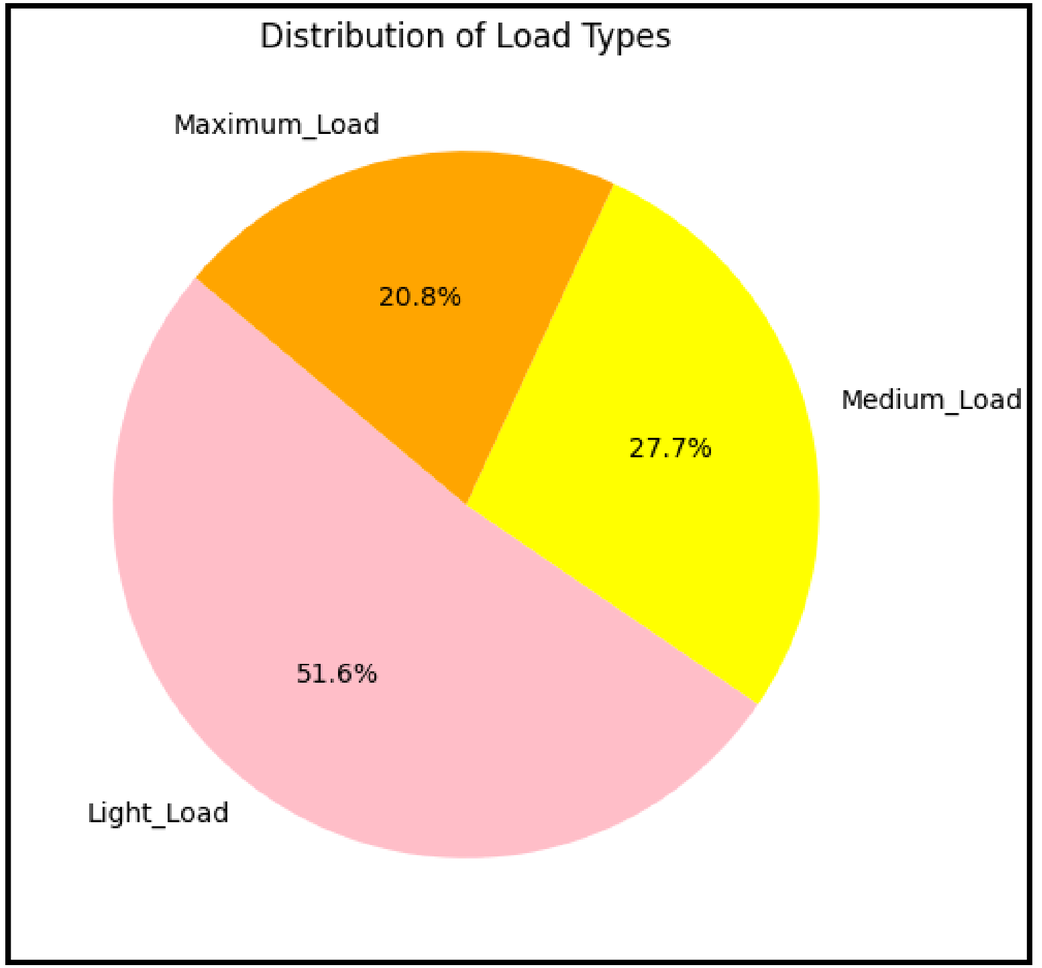
Quality Control: Scanned for any missing data and verified the uniqueness of values for a clean dataset.

Enabled and Dummified the categorical variables: Converted the 'WeekStatus' variable to binary format, with '0' representing weekdays and '1' for weekends.

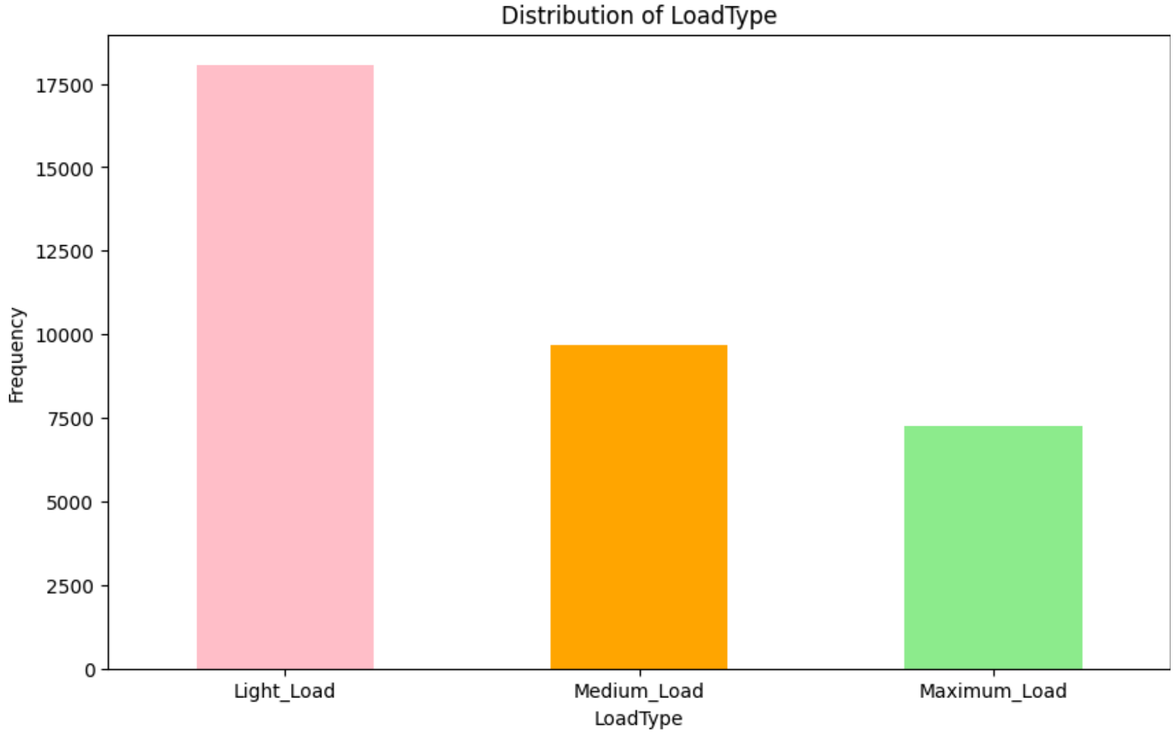
EXPLORATORY DATA ANALYSIS

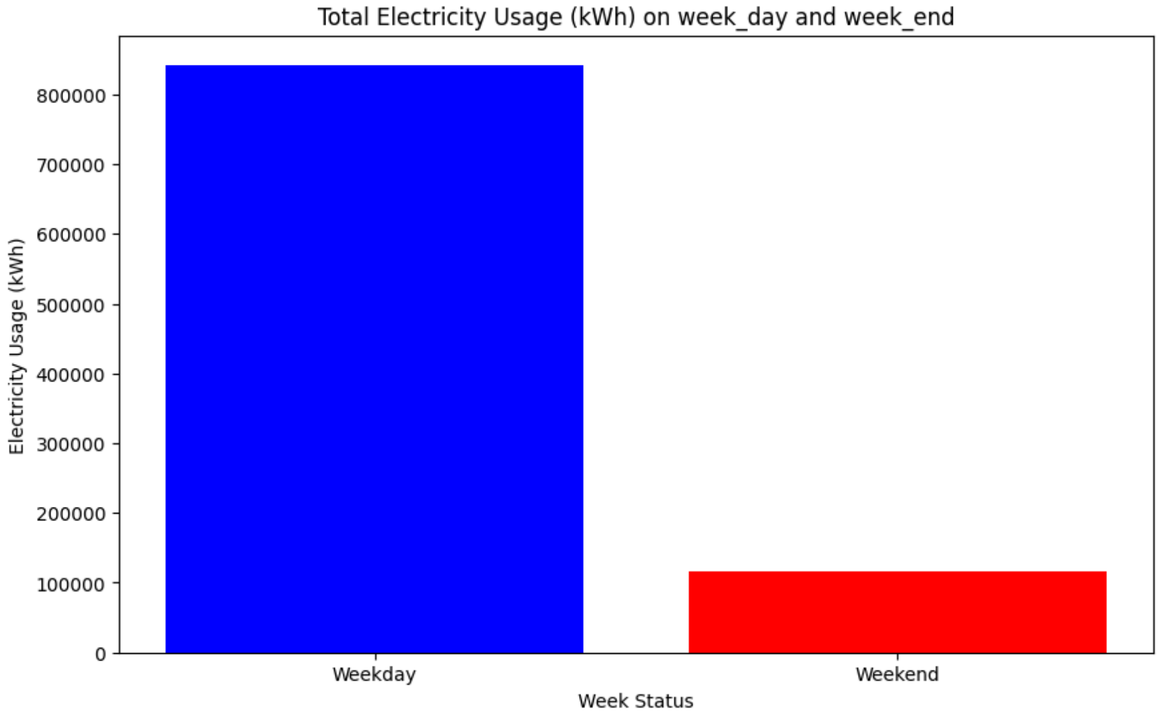
PIE CHARTS


Distribution of Energy Consumption by weekStatus
Weekend
25.9%
74.1%
Weekday

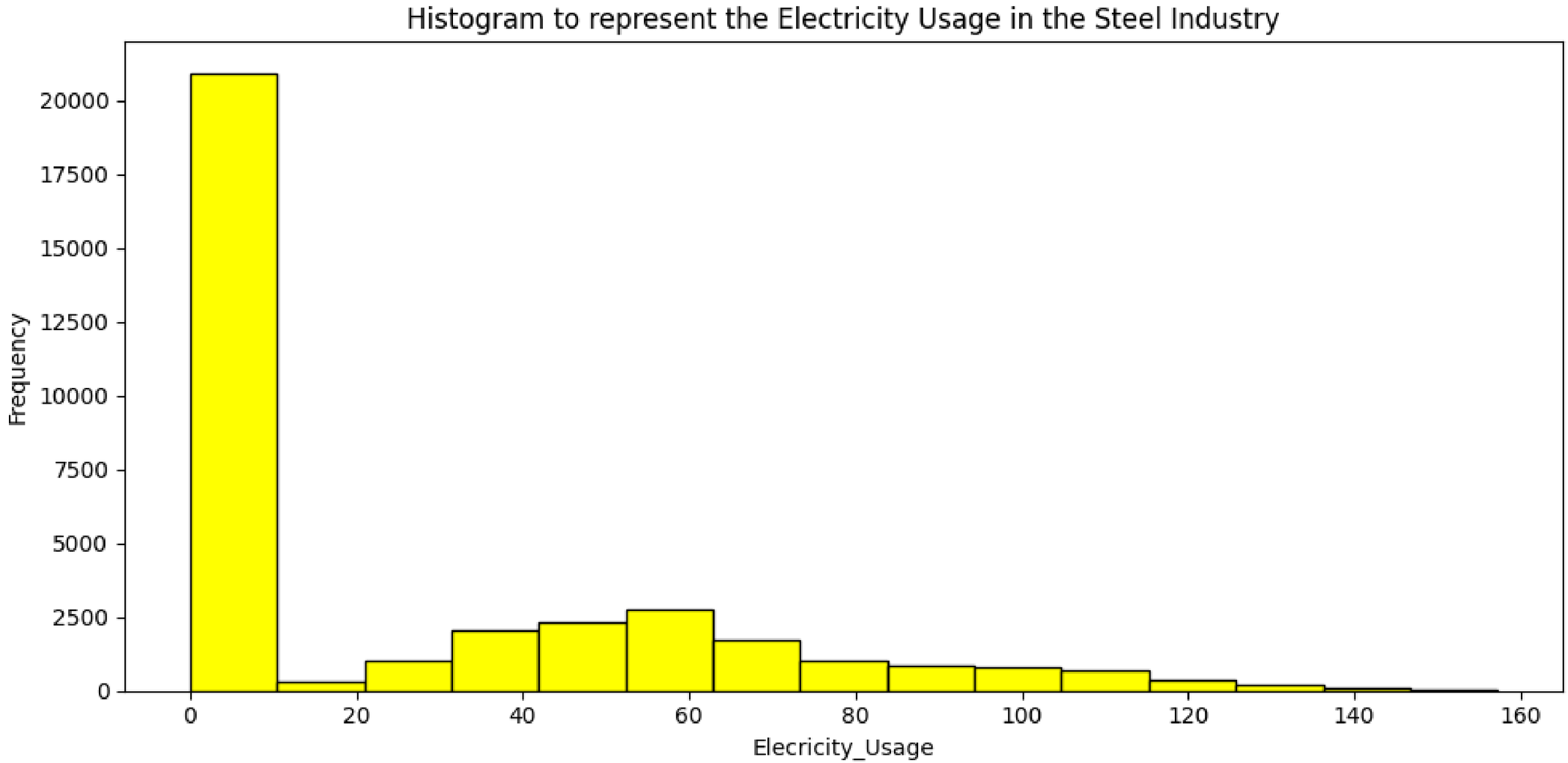

Distribution of Load Types
Maximum_Load
20.8%
Medium_Load
27.7%
51.6%
Light_Load

BAR GRAPH

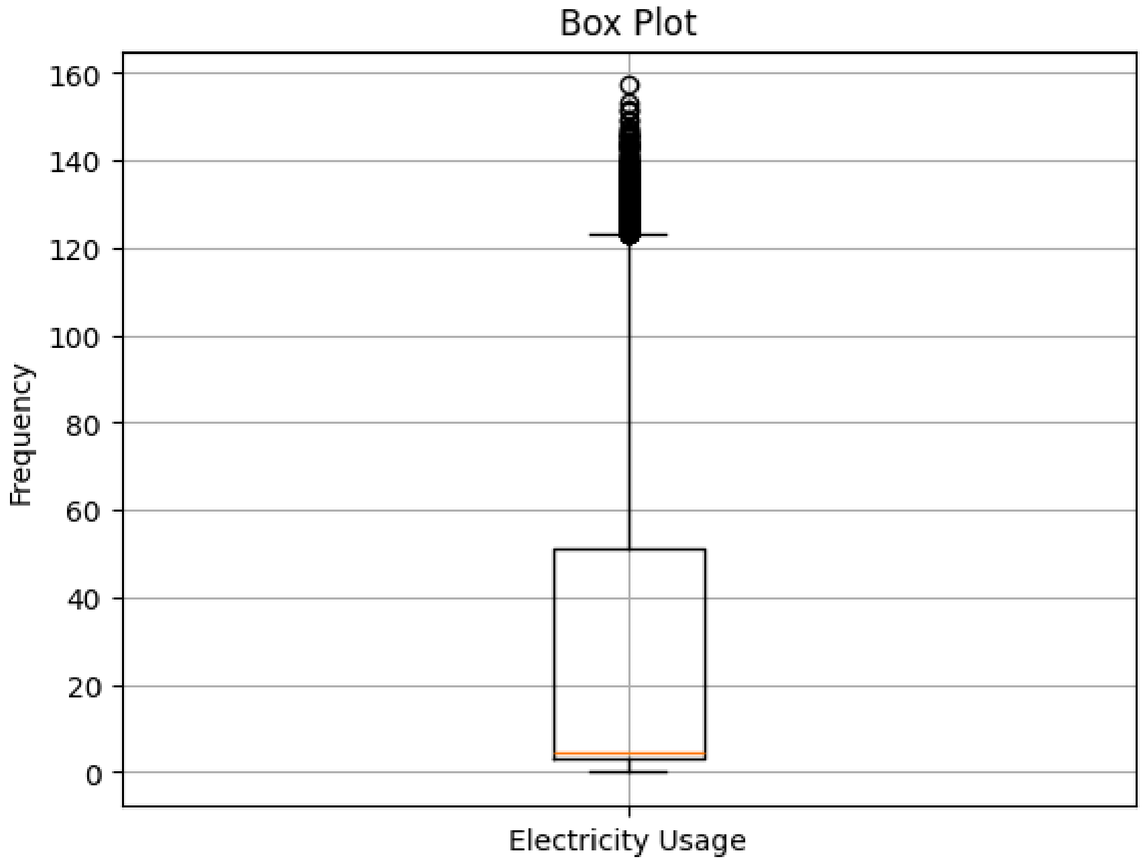

Distribution of LoadType
Frequency
17500
15000
12500
10000
7500
5000
2500
0
Light_Load
Medium_Load
Maximum_Load
LoadType

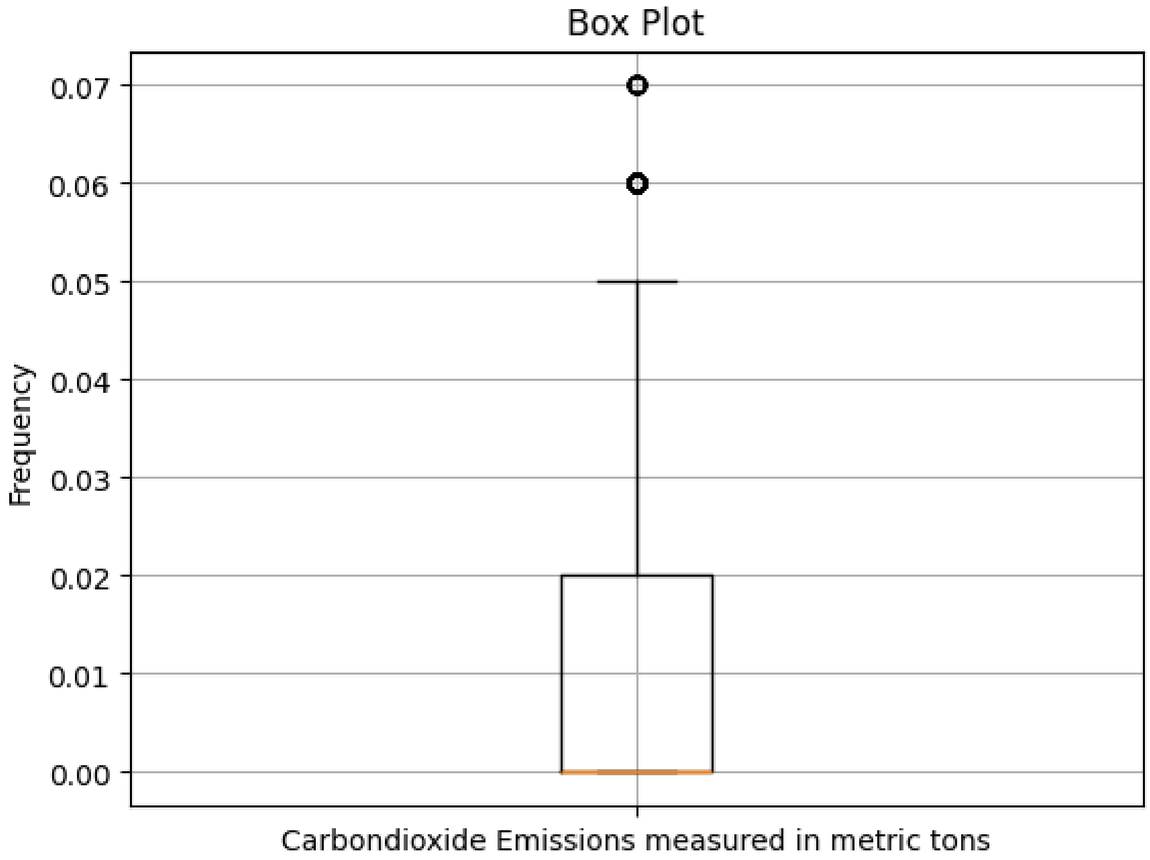

Total Electricity Usage (kWh) on week_day and week_end
Electricity Usage (kWh)
800000
700000
600000
500000
400000
300000
200000
100000
0
Weekday
Weekend
Week Status

HISTOGRAM

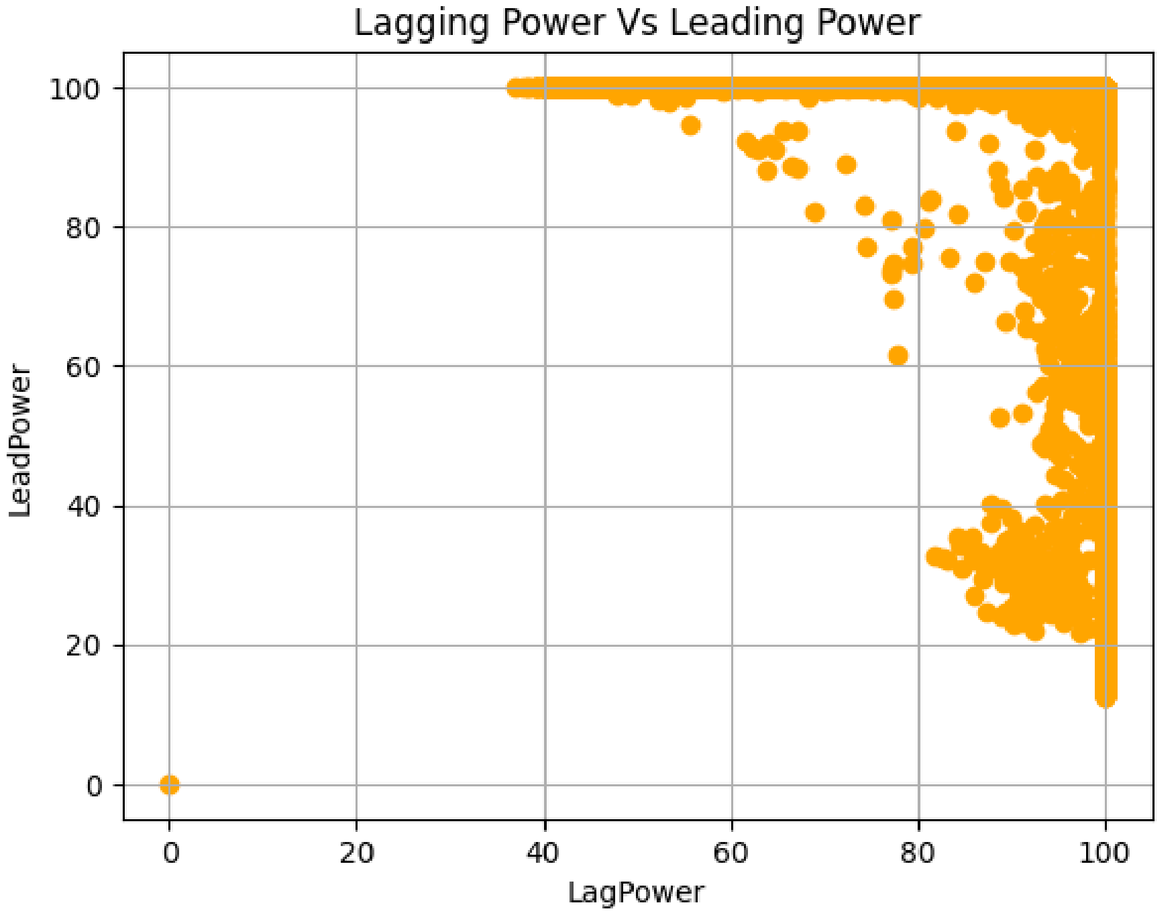

Histogram to represent the Electricity Usage in the Steel Industry
Frequency
20000
17500
15000
12500
10000
7500
5000
2500
0
0
20
40
60
80
100
120
140
160
Elecricity_Usage

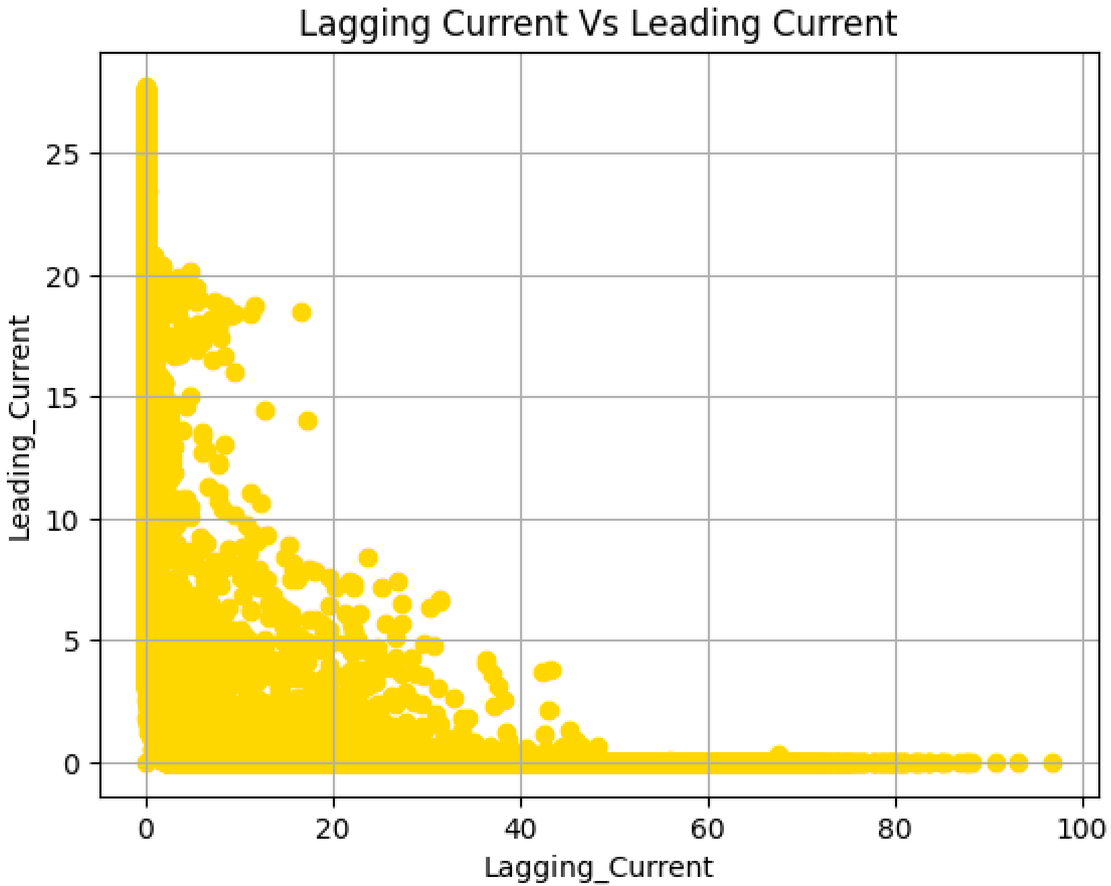
BOX PLOT


Box Plot
160
140
120
100
80
60
40
20
0
Frequency
Electricity Usage


Box Plot
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
Frequency
Carbondioxide Emissions measured in metric tons

SCATTER PLOT





MULTICOLLINEARITY CHECK

Variables with the greatest variance inflation factor (VIF > 2) were removed

	variables	VIF
0	Lagging_Current_Reactive.Power_kVarh	8.7
1	Leading_Current_Reactive_Power_kVarh	7.8
2	CO2(tCO2)	9.8
3	Lagging_Current_Power_Factor	37.9
4	Leading_Current_Power_Factor	21.7
5	NSM	6.0
6	WeekStatus	4.2

	variables	VIF
0	Lagging_Current_Reactive.Power_kVarh	8.4
1	Leading_Current_Reactive_Power_kVarh	2.3
2	CO2(tCO2)	7.9
3	Leading_Current_Power_Factor	5.6
4	NSM	5.4
5	WeekStatus	4.2

Lagging_Current_Power_Factor was removed

	variables	VIF
0	Leading_Current_Reactive_Power_kVarh	2.3
1	CO2(tCO2)	2.1
2	Leading_Current_Power_Factor	5.3
3	NSM	5.2
4	WeekStatus	4.1

Lagging_Current_Reactive.Power_kVarh was removed

MULTICOLLINEARITY CHECK

Variables with the greatest variance inflation factor (VIF > 2) were removed

	variables	VIF
0	Leading_Current_Reactive_Power_kVarh	1.9
1	CO2(tCO2)	2.1
2	NSM	4.0
3	WeekStatus	2.4

	variables	VIF
0	Leading_Current_Reactive_Power_kVarh	1.1
1	CO2(tCO2)	1.7
2	WeekStatus	1.8

Leading_Current_Power_Factor was removed

NSM was removed

MACHINE LEARNING ALGORITHMS

ML ALGORITHMS

- Linear Regression
- K-Nearest Neighbors(KNN)
- Decision Tree
- Random Forest
- XG Boosting
- ADA Boosting
- Support Vector Machine(SVM)

80:20 Train-test split

Algorithms	Model 1 (r2 score)	Model 2 (r2 score)	Model 1 (MAE)	Model 2 (MAE)
Linear Regression	0.9819	0.9791	2.5235	2.7213
KNN	0.9838	0.9645	1.6800	3.2983
Decision Tree Regressor	0.9986	0.9797	0.4733	2.5469
Random Forest	0.9992	0.9792	0.2959	2.5794
XG Boosting	0.9989	0.9792	2.5373	2.5793
ADA Boosting	0.97	0.98	2.58	2.58
SVM	0.2635	0.1076	19.4694	22.4914

Model 1- Before VIF | Model 2- After VIF

75:25 Train-test split

Algorithms	Model 1 (r2 score)	Model 2 (r2 score)	Model 1 (MAE)	Model 2 (MAE)
Linear Regression	0.9829	0.9829	2.5162	2.5162
KNN	0.9831	0.9651	1.7508	3.3322
Decision Tree Regressor	0.9986	0.9812	0.4869	2.5276
Random Forest	0.9992	0.9808	0.3029	2.5515
XG Boosting	0.9990	0.9808	2.5793	2.5793
ADA Boosting	0.96	0.98	2.55	2.55
SVM	0.2713	0.1019	19.3498	22.6290

Model 1- Before VIF | Model 2- After VIF

70:30 Train-test split

Algorithms	Model 1 (r2 score)	Model 2 (r2 score)	Model 1 (MAE)	Model 2 (MAE)
Linear Regression	0.9831	0.9831	2.5476	2.5476
KNN	0.9820	0.9644	1.8261	3.3417
Decision Tree Regressor	0.9984	0.9819	0.5036	2.5155
Random Forest	0.9992	0.9819	0.3142	2.5155
XG Boosting	0.9990	0.9816	2.5793	2.5793
ADA Boosting	0.97	0.98	2.54	2.54
SVM	0.2726	0.0854	19.3205	22.6358

Model 1- Before VIF | Model 2- After VIF

60:40 Train-test split

Algorithms	Model 1 (r2 score)	Model 2 (r2 score)	Model 1 (MAE)	Model 2 (MAE)
Linear Regression	0.9828	0.9828	2.5506	2.5506
KNN	0.9793	0.9637	1.9805	3.5695
Decision Tree Regressor	0.9977	0.9811	0.5673	2.5198
Random Forest	0.9990	0.9811	0.3447	2.5198
XG Boosting	0.9990	0.9808	2.5793	2.5793
ADA Boosting	0.97	0.98	2.54	2.54
SVM	0.2719	0.0624	19.3505	22.7249

Model 1- Before VIF | Model 2- After VIF

Algorithms Comparison

80:20 Train - Test Split

Algorithms	r2 score	MAE
Linear Regression	0.9819	2.5235
K- nearest numbers	0.9838	1.6800
Decision tree Regressor	0.9986	0.4733
Random forest	0.9992	0.2959
XG Boosting	0.9989	2.5373
ADA Boosting	0.97	2.58
SVM	0.2635	19.4694

MODEL 1

70:30 Train - Test Split

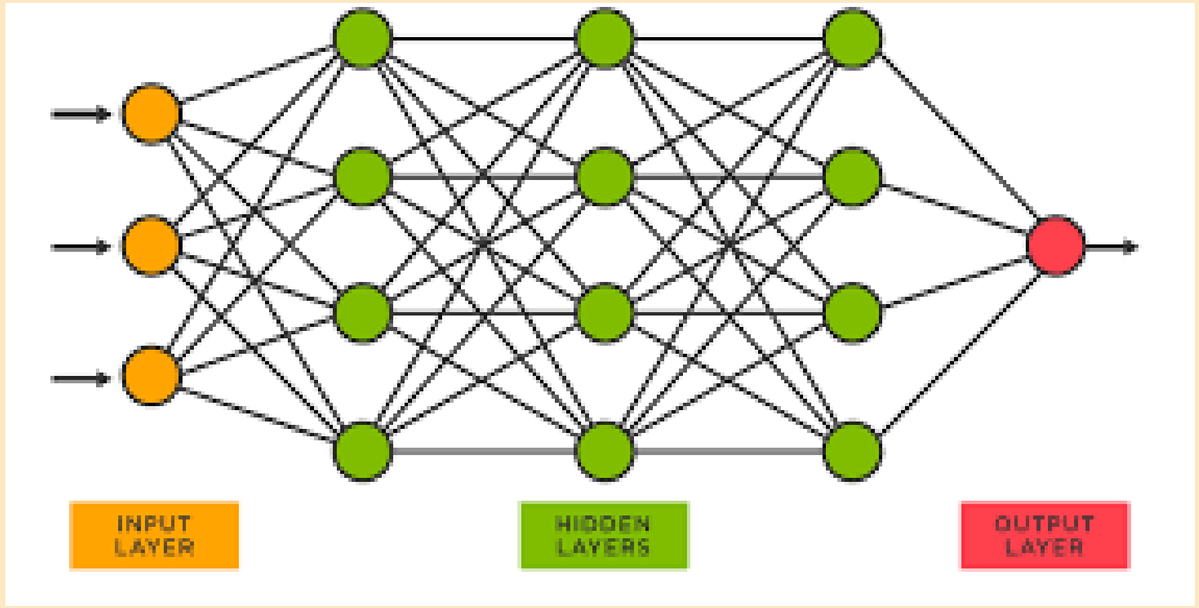
Algorithms	r2 score	MAE
Linear Regression	0.9831	2.5476
K- nearest numbers	0.9644	3.3417
Decision tree Regressor	0.9819	2.5155
Random forest	0.9819	2.5155
XG Boosting	0.9816	2.5793
ADA Boosting	0.9800	2.5400
SVM	0.0854	22.6358

MODEL 2

r2 - Accuracy

MAE - Mean Absolute Error

Architecture of Artificial Neural Network

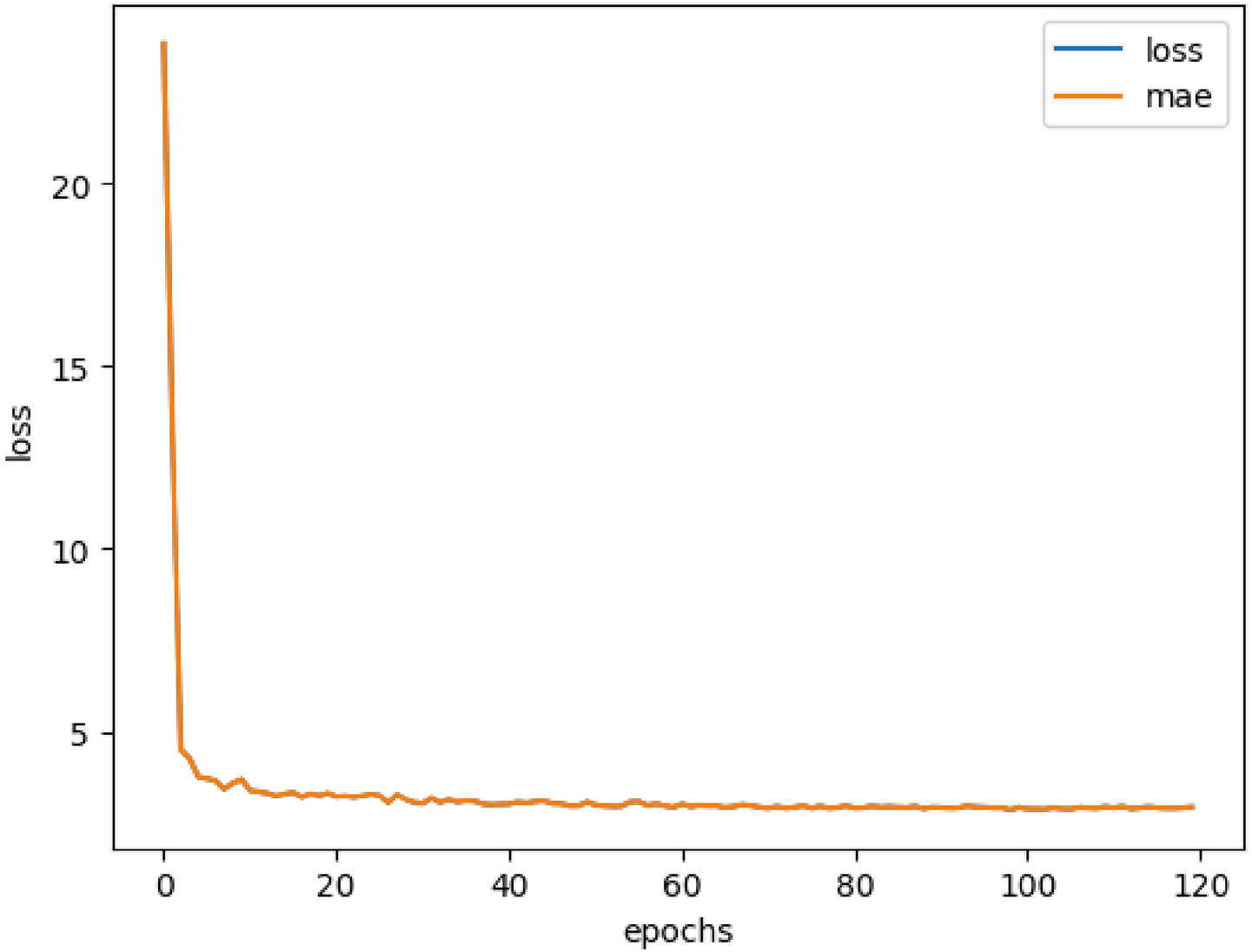


Before VIF

Train Test	Architecture	Optimizer	Epochs	MAE
80-20	24-23-22-21-20-10-1	Adam	120	8.5783
75-25	24-23-22-21-20-10-1	Adam	120	8.8187
70-30	24-23-22-21-20-10-1	Adam	120	7.6772
60-40	24-23-22-21-20-10-1	Adam	120	8.4429

After VIF

Train Test	Architecture	Optimizer	Epochs	MAE
80-20	24-23-22-21-20-10-1	Adam	120	2.8426
75-25	24-23-22-21-20-10-1	Adam	120	2.7961
70-30	24-23-22-21-20-10-1	Adam	120	2.7017
60-40	24-23-22-21-20-10-1	Adam	120	2.7023



Train-Test Split	**70-30**
Architecture	**24-23-22-21-20-10-1**
Optimizer	**Adam**
Epochs	**120**

SUMMARY

- This research seeks to evaluate various machine learning algorithms to identify the most accurate and efficient model for predicting energy consumption in the steel industry. By comparing performance metrics, the study aims to provide insights into the optimal approach for energy forecasting in this sector.

- For Model 1, the best fit model is Random Forest with the r2_score of 0.9992 and MAE 0.2959

- For Model 2, the best fit model is Linear Regression with the r2_score of 0.9831 and MAE 2.5476

KEY INSIGHTS

- **Data Preprocessing:** Cleaning datasets, removing high VIF variables, and encoding categorical data were critical steps for achieving reliable and accurate predictions.

- **Model Performance:** Decision Tree and Random Forest demonstrated strong capabilities in managing complex, high-dimensional datasets, proving their effectiveness for this application.

- **Sustainability Impact:** The predictive model enables steel manufacturers to optimize energy usage, reduce operational costs, and contribute to environmental sustainability.

FUTURE SCOPES

- **Energy Optimization:** Companies in the steel industry can use a regression model to identify ways to use energy more efficiently, saving costs and reducing waste.

- **Sustainability Goals:** The model can help industries lower their carbon footprint by suggesting cleaner energy sources or optimizing energy-intensive processes.

WORK DISTRIBUTION

Team member 1 Priyanka Parthasarathy	Collect information about Steel Industry & Literature Review
Team member 2 Rishi Raj Singh	Data Preprocessing
Team member 3 Subhasish Choudhury	Exploratory Data Analysis
Team member 4 Sai Phaneendra	Implement Machine Learning Algorithms

Collab Notebook

THANK YOU

Priyanka Parthasarathy

Rishi Raj Singh

Subhasish Choudhury

Sai Phaneendra

APPENDIX

Loading the Dataset

Loading Dataset

```
data= pd.read_csv('Steel_industry_data1.csv')
data
```

	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	Loa
0	01-01-2018 00:15	3.17	2.95	0.00	0.0	73.21	100.00	900	Weekday	Monday	Ligh
1	01-01-2018 00:30	4.00	4.46	0.00	0.0	66.77	100.00	1800	Weekday	Monday	Ligh
2	01-01-2018 00:45	3.24	3.28	0.00	0.0	70.28	100.00	2700	Weekday	Monday	Ligh
3	01-01-2018 01:00	3.31	3.56	0.00	0.0	68.09	100.00	3600	Weekday	Monday	Ligh
4	01-01-2018 01:15	3.82	4.50	0.00	0.0	64.72	100.00	4500	Weekday	Monday	Ligh

Null Value

Finding null value

```
[ ]  data.isna().sum()
```

	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

Checking for the data type

Checking datatype

```
[ ]  X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 35040 entries, 0 to 35039
Data columns (total 8 columns):
 #   Column                                Non-Null Count  Dtype
---  ------                                --------------  -----
 0   Lagging_Current_Reactive.Power_kVarh  35040 non-null  float64
 1   Leading_Current_Reactive_Power_kVarh  35040 non-null  float64
 2   CO2(tCO2)                             35040 non-null  float64
 3   Lagging_Current_Power_Factor          35040 non-null  float64
 4   Leading_Current_Power_Factor          35040 non-null  float64
 5   NSM                                   35040 non-null  int64
 6   WeekStatus                            35040 non-null  object
 7   Load_Type                             35040 non-null  object
dtypes: float64(5), int64(1), object(2)
memory usage: 2.1+ MB
```

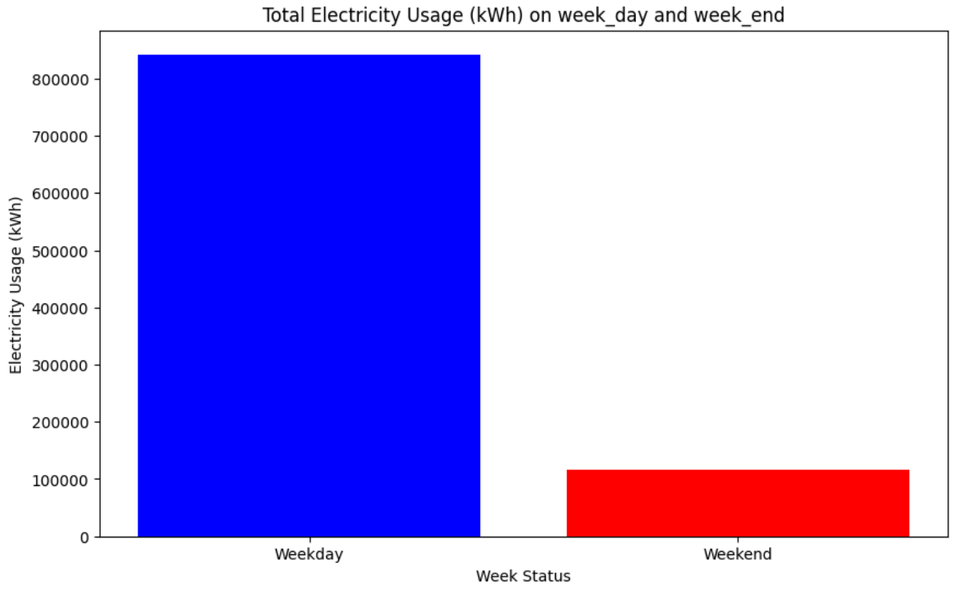
Enabling the variable WeekStatus

```
[ ] # Convert 'WeekStatus' to binary: 1 for Weekday, 0 for Weekend
    data['WeekStatus'] = data['WeekStatus'].apply(lambda x: 1 if x == 'Weekday' else 0)

    # Verify the changes
    data.head()
```

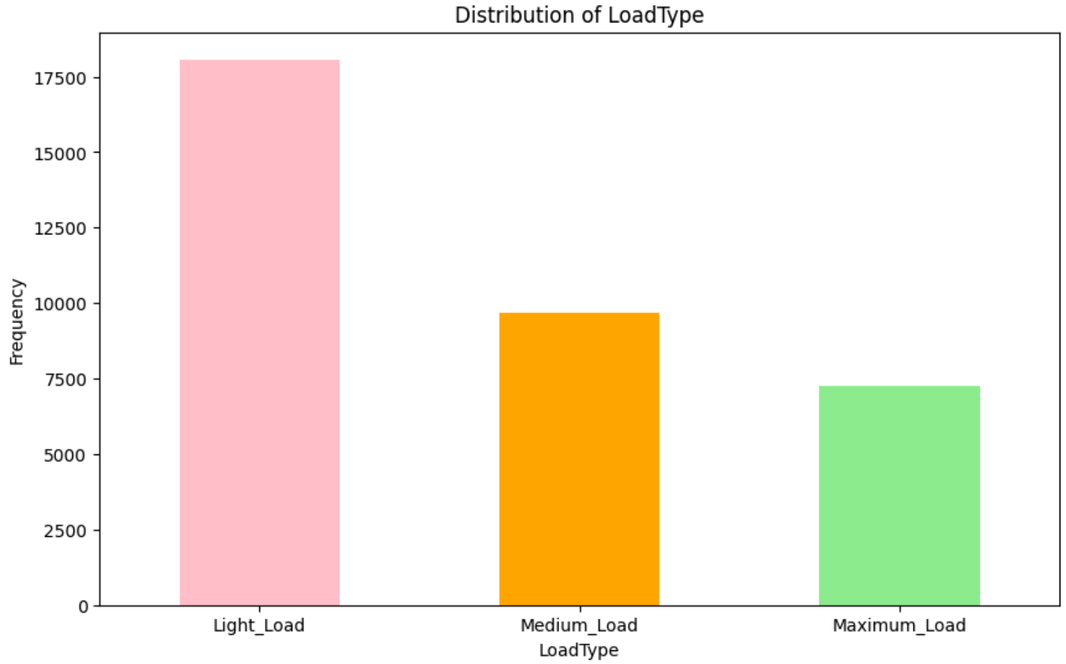
	Usage_kWh	Lagging_Current_Reactive.Power_kVarh	Leading_Current_Reactive_Power_kVarh	CO2(tCO2)	Lagging_Current_Power_Factor	Leading_Current_Power_Factor	NSM	WeekStatus	Load_Type
0	3.17	2.95	0.0	0.0	73.21	100.0	900	1	Light_Load
1	4.00	4.46	0.0	0.0	66.77	100.0	1800	1	Light_Load
2	3.24	3.28	0.0	0.0	70.28	100.0	2700	1	Light_Load
3	3.31	3.56	0.0	0.0	68.09	100.0	3600	1	Light_Load
4	3.82	4.50	0.0	0.0	64.72	100.0	4500	1	Light_Load

Bar Plot

```
plt.figure(figsize=(10, 6))
plt.bar(total_usage.index, total_usage.values, color=['blue','red'])

plt.title('Total Electricity Usage (kWh) on week_day and week_end')
plt.xlabel('Week Status')
plt.ylabel('Electricity Usage (kWh)')

plt.show()
```




```
plt.figure(figsize=(10, 6))
status_counts.plot(kind='bar', color=['pink','orange','lightgreen'])
plt.title('Distribution of LoadType')
plt.xlabel('LoadType')
plt.ylabel('Frequency')
plt.xticks(rotation=0)

plt.show()
```



Multi collinearity check

```python
# 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)
```

	variables	VIF
0	Lagging_Current_Reactive.Power_kVarh	8.7
1	Leading_Current_Reactive_Power_kVarh	7.8
2	CO2(tCO2)	9.8
3	Lagging_Current_Power_Factor	37.9
4	Leading_Current_Power_Factor	21.7
5	NSM	6.0
6	WeekStatus	4.2

Linear Regression

```
import pandas as pd
import seaborn as sns
import statsmodels.formula.api as smf
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.model_selection import train_test_split
import numpy as np
```

```
# 20-80
lm2 = LinearRegression()
lm2.fit(X_train1, y_train1)
y_pred1 = lm2.predict(X_test1)
print((metrics.r2_score(y_test1, y_pred1)))
print((metrics.mean_squared_error(y_test1, y_pred1)))
print((metrics.mean_absolute_error(y_test1,y_pred1)))
```

```
0.9819580238142336
20.06012520183277
2.5235996706297033
```

Before applying VIF

```
[ ] X_nomulti = data[['Lagging_Current_Power_Factor','Lagging_Current_Reactive.Power_kVarh','Leading_Current_Power_Factor','NSM']]
    y = data.Usage_kWh
    X_train_nomulti, X_test_nomulti, y_train_nomulti, y_test_nomulti = train_test_split(X_nomulti, y, random_state=1)
    lm2 = LinearRegression()
    lm2.fit(X_train_nomulti, y_train_nomulti)
    y_pred = lm2.predict(X_test_nomulti)
    print(np.sqrt(metrics.mean_squared_error(y_test_nomulti, y_pred)))
```

```
9.896204310822169
```

```
[ ] # 20-80
    lm2 = LinearRegression()
    lm2.fit(X_train1_nomulti, y_train1_nomulti)
    y_pred1 = lm2.predict(X_test1_nomulti)
    print((metrics.r2_score(y_test1_nomulti, y_pred1)))
    print((metrics.mean_squared_error(y_test1_nomulti, y_pred1)))
    print((metrics.mean_absolute_error(y_test1_nomulti,y_pred1)))
```

```
0.9791367081851473
23.19702906261984
2.721318808493265
```

After applying VIF

K Nearest Neighbors

80-20 Ratio

```
[ ]  model=KNeighborsRegressor(n_neighbors=5)
     model.fit(X_train1,y_train1)
```

```
KNeighborsRegressor
KNeighborsRegressor()
```

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

```
[ 3.888  3.874  3.636 ...  3.636 58.448  3.352]
```

```
     df=pd.DataFrame({'Predicted':y_pred1,'Actual':y_test1})
     print(df)
```

```
        Predicted  Actual
5760        3.888    3.20
14294       3.874    3.78
35035       3.636    3.85
30292      47.570   43.81
31651      64.568   65.34
...           ...     ...
6873      103.420   96.62
25118      47.686   40.61
6907        3.636    3.85
6185       58.448   60.80
27936       3.352    3.60
```

Before applying VIF

80-20 Ratio

```
[ ]  model=KNeighborsRegressor(n_neighbors=5)
     model.fit(X_train1_nomulti,y_train1_nomulti)
```

```
KNeighborsRegressor
KNeighborsRegressor()
```

```
[ ]  y_pred1 = model.predict(X_test1_nomulti)
     print(y_pred1)
```

```
[ 3.564  5.874  3.564 ...  3.564 62.254  3.564]
```

```
     df=pd.DataFrame({'Predicted':y_pred1,'Actual':y_test1_nomulti})
     print(df)
```

```
        Predicted  Actual
5760        3.564    3.20
14294       5.874    3.78
35035       3.564    3.85
30292      35.716   43.81
31651      62.254   65.34
...           ...     ...
6873       87.970   96.62
25118      45.036   40.61
6907        3.564    3.85
6185       62.254   60.80
```

After applying VIF

Decision Tree Regressor

80-20 Ratio

```
[ ] clf = DecisionTreeRegressor()
    clf1 = clf.fit(X_train1,y_train1)
    print(clf1)
```

```
DecisionTreeRegressor()
```

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

```
[ 3.2   3.78  3.85 ...  3.85 61.13  3.56]
```

```
[ ] print("R-squared:", metrics.r2_score(y_test1, y_pred1))
```

```
R-squared: 0.9986623426592398
```

```
[ ] print("Mean Squared Error:", metrics.mean_squared_error(y_test1, y_pred1))
```

```
Mean Squared Error: 1.4872857305936071
```

```
print("Mean Absolute Error:", metrics.mean_absolute_error(y_test1, y_pred1))
```

```
Mean Absolute Error: 0.47431221461187223
```

Before applying VIF

80-20 Ratio

```
[ ] clf = DecisionTreeRegressor()
    clf1 = clf.fit(X_train1_nomulti,y_train1_nomulti)
    print(clf1)
```

```
DecisionTreeRegressor()
```

```
[ ] y_pred1 = clf.predict(X_test1_nomulti)
    print(y_pred1)
```

```
[ 3.83838392  4.04333333  3.83838392 ...  3.83838392 63.39701377
  3.83838392]
```

```
[ ] print("R-squared:", metrics.r2_score(y_test1_nomulti, y_pred1))
```

```
R-squared: 0.9797766339876377
```

```
[ ] print("Mean Squared Error:", metrics.mean_squared_error(y_test1_nomulti, y_pred1))
```

```
Mean Squared Error: 22.48552209765839
```

```
print("Mean Absolute Error:", metrics.mean_absolute_error(y_test1_nomulti, y_pred1))
```

```
Mean Absolute Error: 2.5461480831521244
```

After applying VIF

Random Forest

80-20 Ratio

```
[ ]  print(X_train1.shape)
     print(y_train1.shape)
     print(X_test1.shape)
     print(y_test1.shape)
```

```
(28032, 7)
(28032,)
(7008, 7)
(7008,)
```

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

```
RandomForestRegressor
RandomForestRegressor()
```

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

```
[ 3.2213   3.7727   3.8416 ...   3.8416 60.5589   3.6028]
```

```
[ ]  print("R-squared:", metrics.r2_score(y_test1, y_pred1))
```

```
R-squared: 0.9992750696705234
```

Before applying VIF

80-20 Ratio

```
[ ]  print(X_train1_nomulti.shape)
     print(y_train1_nomulti.shape)
     print(X_test1_nomulti.shape)
     print(y_test1_nomulti.shape)
```

```
(28032, 3)
(28032,)
(7008, 3)
(7008,)
```

```
[ ]  rf = RandomForestRegressor()
     rf.fit(X_train1_nomulti, y_train1_nomulti)
```

```
RandomForestRegressor
RandomForestRegressor()
```

```
[ ]  y_pred1 = clf.predict(X_test1_nomulti)
     print(y_pred1)
```

```
[ 3.85836716  4.23        3.85836716 ...  3.85836716 63.45416576
  3.85836716]
```

```
print("R-squared:", metrics.r2_score(y_test1_nomulti, y_pred1))
```

```
R-squared: 0.9792584380301215
```

After applying VIF

XG Boosting

80-20 Ratio

```
[ ]  model = xgb.XGBRegressor()
     model = model.fit(X_train1, y_train1)
```

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

```
[ 3.1953146  3.784911   3.6262646 ...  3.6262646 60.08243    3.5062823]
```

```
mse = mean_squared_error(y_test1, y_pred1)
r2 = r2_score(y_test1, y_pred1)

print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")
print(f"Mean Absolute Error: {mae}")
```

```
Mean Squared Error: 1.1463911143874774
R-squared: 0.998968941570541
```

Before applying VIF

80-20 Ratio

```
[ ]  model = xgb.XGBRegressor()
     model = model.fit(X_train1_nomulti, y_train1_nomulti)
```

```
[ ]  y_pred1 = clf.predict(X_test1_nomulti)
     print(y_pred1)
```

```
[ 3.85836716  4.23        3.85836716 ...  3.85836716 63.45416576
  3.85836716]
```

```
[ ]  mse = mean_squared_error(y_test1_nomulti, y_pred1)
     r2 = r2_score(y_test1_nomulti, y_pred1)

     print(f"Mean Squared Error: {mse}")
     print(f"R-squared: {r2}")
     print(f"Mean Absolute Error: {mae}")
```

```
Mean Squared Error: 23.061682695578966
R-squared: 0.9792584380301215
Mean Absolute Error: 6.645378972266709
```

After applying VIF

ADA Boosting

80-20 Ratio

```
[ ]  model = AdaBoostRegressor(n_estimators=50, random_state=42)
     model.fit(X_train1, y_train1)
```

```
AdaBoostRegressor(random_state=42)
```

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

```
[ 7.78265423  9.03034891 10.30705614 ... 10.30705614 63.75161043
 10.30705614]
```

```
     r2 = r2_score(y_test1, y_pred1)
     print(f"R-squared: {r2:.2f}")

     mse = mean_squared_error(y_test1, y_pred1)
     print(f"Mean Squared Error: {mse:.2f}")

     mae=mean_absolute_error(y_test1, y_pred1)
     print(f"Mean Absolute Error: {mae:.2f}")
```

```
R-squared: 0.97
Mean Squared Error: 35.62
Mean Absolute Error: 5.38
```

Before applying VIF

80-20 Ratio

```
[ ]  model = AdaBoostRegressor(n_estimators=50, random_state=42)
     model.fit(X_train1_nomulti, y_train1_nomulti)
```

```
[ ]  y_pred1 = clf.predict(X_test1_nomulti)
     print(y_pred1)
```

```
[ ]  r2 = r2_score(y_test1_nomulti, y_pred1)
     print(f"R-squared: {r2:.2f}")

     mse = mean_squared_error(y_test1_nomulti, y_pred1)
     print(f"Mean Squared Error: {mse:.2f}")

     mae = mean_absolute_error(y_test1_nomulti, y_pred1)
     print(f"Mean Absolute Error: {mae:.2f}")
```

```
R-squared: 0.98
Mean Squared Error: 23.06
Mean Absolute Error: 2.58
```

After applying VIF

Support Vector Machine

80-20 Ratio

```
[ ]  model = SVR(kernel='rbf')
     model.fit(X_train1, y_train1)
```

```
SVR()
```

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

```
[ 5.5593862  12.07253019  3.1405723  ...  3.1405723  34.52717529
  5.55886705]
```

```
     df = pd.DataFrame({'Predicted':y_pred1,'Actual':y_test1})
     print(df)
```

```
       Predicted  Actual
5760    5.559386    3.20
14294  12.072530    3.78
35035   3.140572    3.85
30292  48.141869   43.81
31651  45.291080   65.34
...          ...     ...
6873   50.357576   96.62
25118  49.390866   40.61
6907    3.140572    3.85
6185   34.527175   60.80
```

Before applying VIF

80-20 Ratio

```
[ ]  model = SVR(kernel='rbf')
     model.fit(X_train1_nomulti, y_train1_nomulti)
```

```
SVR()
```

```
[ ]  y_pred1 = model.predict(X_test1_nomulti)
     print(y_pred1)
```

```
[27.81462235  4.59836468 27.81462235 ... 27.81462235 28.10126744
 27.81462235]
```

```
     df=pd.DataFrame({'Predicted':y_pred1,'Actual':y_test1_nomulti})
     print(df)
```

```
       Predicted  Actual
5760   27.814622    3.20
14294   4.598365    3.78
35035  27.814622    3.85
30292  32.184537   43.81
31651  28.101267   65.34
...          ...     ...
6873   28.196663   96.62
25118  28.005795   40.61
6907   27.814622    3.85
```

After applying VIF

Artificial Neural Network

```
[ ]  import tensorflow as tf
     import pandas as pd
     import matplotlib.pyplot as plt
```

80-20 Ratio

```
tf.random.set_seed(42)

# STEP1: Creating the model

model= tf.keras.Sequential([
                            tf.keras.layers.Dense(24),

                            tf.keras.layers.Dense(23),

                            tf.keras.layers.Dense(22),

                            tf.keras.layers.Dense(21),

                            tf.keras.layers.Dense(20),

                            tf.keras.layers.Dense(10),

                            tf.keras.layers.Dense(1)
])

# STEP2: Compiling the model   # optimizer can be SGD, Adam
# Set the learning rate when creating the Adam optimizer
optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)

model.compile(loss= tf.keras.losses.mae,
              optimizer= optimizer,  # Use the optimizer with the specified learning rate
              metrics= ["mae"])

# STEP3: Fit the model
# Remove learning_rate from fit() arguments
history= model.fit(X_train1, y_train1, epochs= 120, verbose=1)
```

Before applying VIF


```
[ ]  import tensorflow as tf
     import pandas as pd
     import matplotlib.pyplot as plt
```

80-20 Ratio

```
tf.random.set_seed(42)

# STEP1: Creating the model

model= tf.keras.Sequential([
                            tf.keras.layers.Dense(24),

                            tf.keras.layers.Dense(23),

                            tf.keras.layers.Dense(22),

                            tf.keras.layers.Dense(21),

                            tf.keras.layers.Dense(20),

                            tf.keras.layers.Dense(10),

                            tf.keras.layers.Dense(1)
])

# STEP2: Compiling the model   # optimizer can be SGD, Adam
# Set the learning rate when creating the Adam optimizer
optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)

model.compile(loss= tf.keras.losses.mae,
              optimizer= optimizer,  # Use the optimizer with the specified learning rate
              metrics= ["mae"])

# STEP3: Fit the model
# Remove learning_rate from fit() arguments
history= model.fit(X_train1_nomulti, y_train1_nomulti, epochs= 120, verbose=1)
```


After applying VIF