STEEL INDUSTRY
ENERGY CONSUMPTION PREDICTION

AURORA'S DEGREE & PG COLLEGE | M.Sc. STATISTICS
K. Vaishnavi

Content



- Objective and Abstract - 2
- Introduction - 4
- Literature Review - 5
- Data Preprocessing - 8
- Exploratary data analysis - 12
- Algorithms - 21
- Summary - 29



Abstract

The project focuses on developing a model that accurately predicts energy consumption in the steel industry, leading to more sustainable and efficient practices.

The study explores machine learning algorithms like Linear Regression, Decision Tree, Random Forest, K-Nearest Neighbours, Support Vector Machines, Bagging, and Boosting to predict power consumption.

Objective

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

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

K. Karthik,
R. Dharmaprakash and
S. Sathya

- The CatBoost prediction algorithm was employed for energy consumption prediction, and hyperparameter optimization was performed using GridSearchCV with 5-fold cross-validation.

- The developed model has undergone a comparative analysis based on both Root Mean Squared Error and Mean Absolute Percentage Error metrics, demonstrating its promise for accurate energy consumption prediction on both the training and test sets.

https://r.search.yahoo.com/_ylt=AwrjbI0GPbxmJqcEHn1XNyoA;_ylu=Y29sbwNncTEEcG9zAzQEdnRpZAMEc2VjA3Ny/RV=2/RE=1724822023/RO=10/RU=https%3a%2f%2fwww.pubs2.ascee.org%2findex.php%2fIJRCS%2farticle%2fdownload%2f1234%2fpdf/RK=2/RS=Mv2_uR6XohqslVWnCKoUklLeGPw-

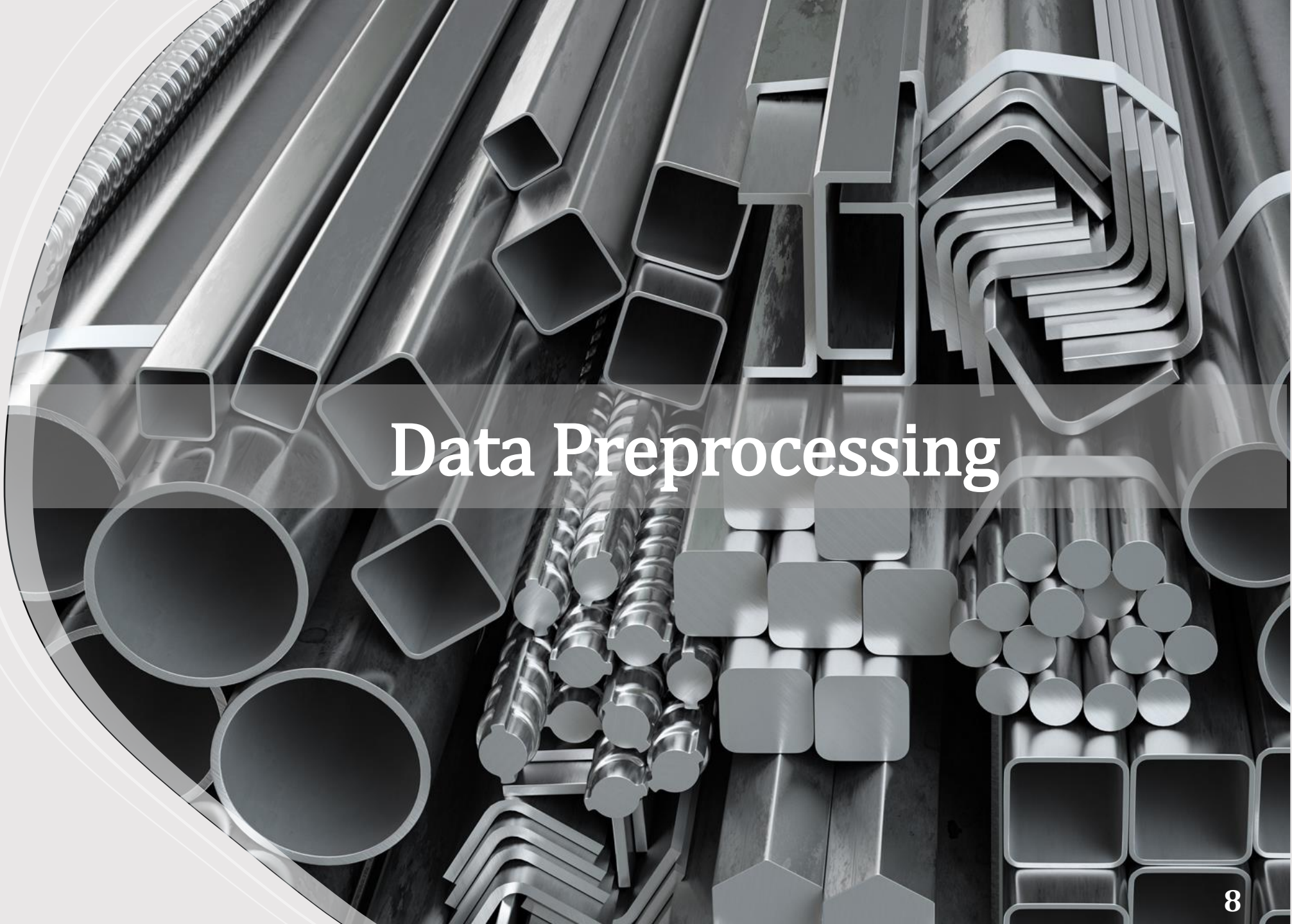
Literature Review - 2

Sathishkumar V E

- In the test set, four statistical models are trained and evaluated:
  (a) Linear regression (LR)
  (b) Support Vector Machine with radial kernel (SVM RBF)
  (c) Gradient Boosting Machine (GBM)
  (d) Random Forest (RF)

- Root Mean Squared Error (RMSE), Mean Absolute Error (MAE) and Mean Absolute Percentage Error (MAPE) are used to measure the prediction efficiency of regression designs. When using all the predictors, the best model RF can provide RMSE value 7.33 in the test set

https://r.search.yahoo.com/_ylt=AwrjbI0GPbxmJqcEG31XNyoA;_ylu=Y29sbwNncTEEcG9zAzEEdnRpZAMEc2VjA3Ny/RV=2/RE=1724822023/RO=10/RU=https%3a%2f%2fkoreascience.kr%2farticle%2fCFKO201924664107981.pdf/RK=2/RS=qFTz.64dMqygj1jNNmzGM8egfRw-



Data Preprocessing

Data

Source

Our dataset was collected from UC Irvine
https://archive.ics.uci.edu/dataset/851/steel+industry+energy+consumption

date	Usage_kW	Lagging_Cı	Leading_C	CO2(tCO2)	Lagging_Cı	Leading_C	NSM	WeekStatı	Day_of_w	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

Variables

The dataset used for the prediction model comprises 11 attributes and 35,040 instances.

Here, we predict the attribute named Usage_kVarh

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

Data Cleaning

- **Removing columns :**

  As there are same dates with different timestamps, we deleted the column named date from the data.

- Checked for missing values and unique values

- Enlabled and dummified the categorical variables

  --enlabled weekday and weekend from the variable WeekStatus to 0 and 1 respectively

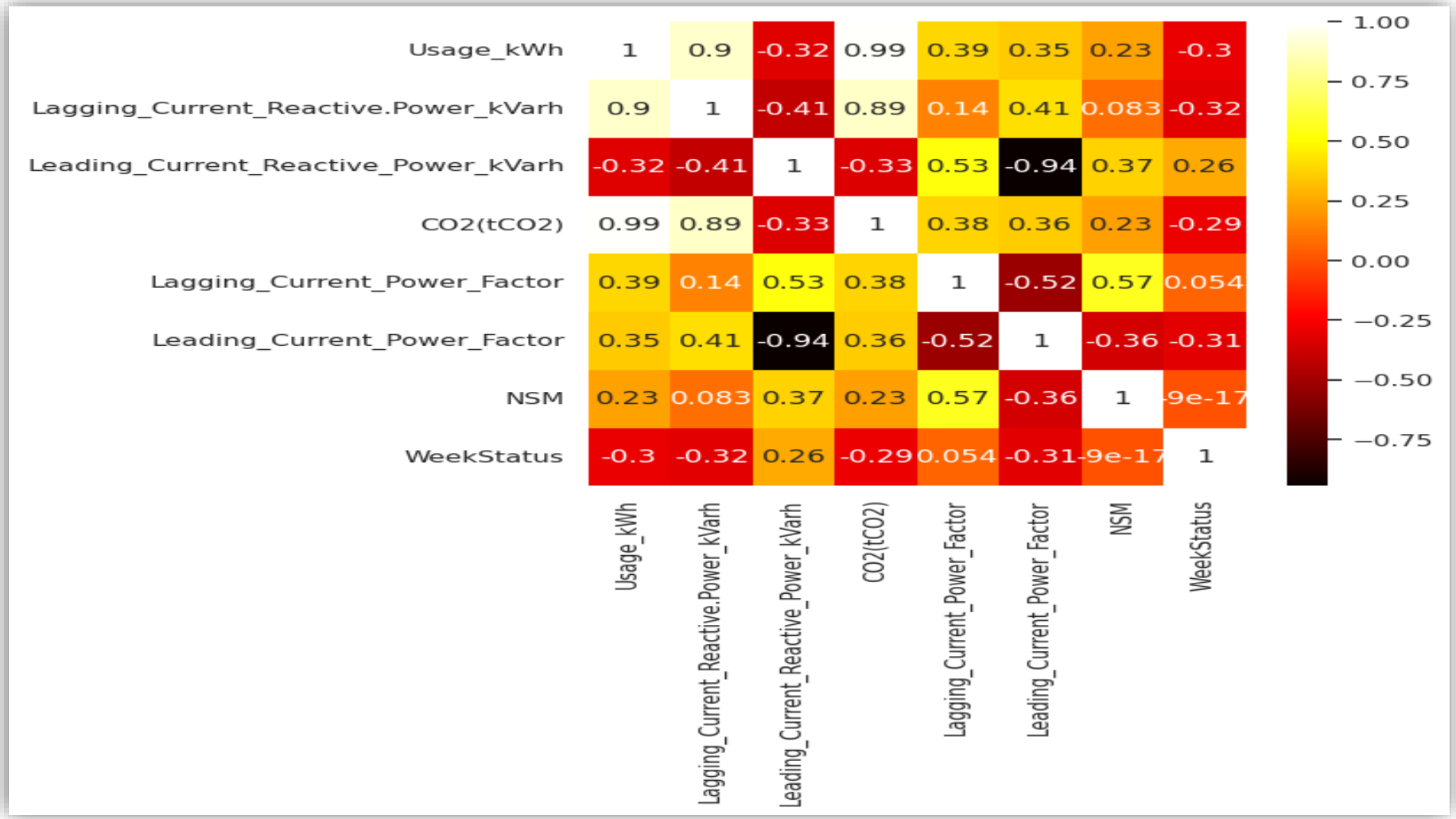
- To perform dummy variable encoding we divided the data into two sets
  - continuous data
  - categorical data

- Renaming the columns names

Original variable names	Renamed variable names
Lagging_Current_Reactive.Power_kVarh	LaRP
Leading_Current_Reactive.Power_kVarh	LeRP
CO2(tCO2)	CO2
Lagging_Current_Power_Factor_kVarh	LaPF
Leading_Current_Power_Factor_kVarh	LePF
Maximum _Load	MaxL
Minimum_Load	MinL
Light_Load	LL
Usage_kVarh	Usage

Exploratory Data Analysis

Correlation matrix

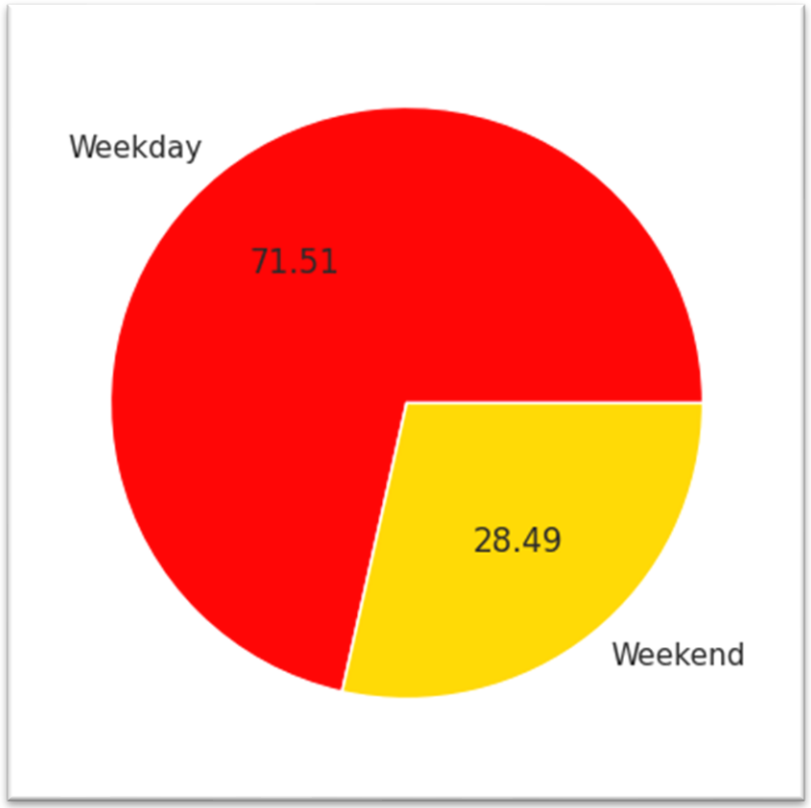


We can observe Usage and CO2 are most positively correlated

Next most positively correlated variables are Lagging current reactive power and usage

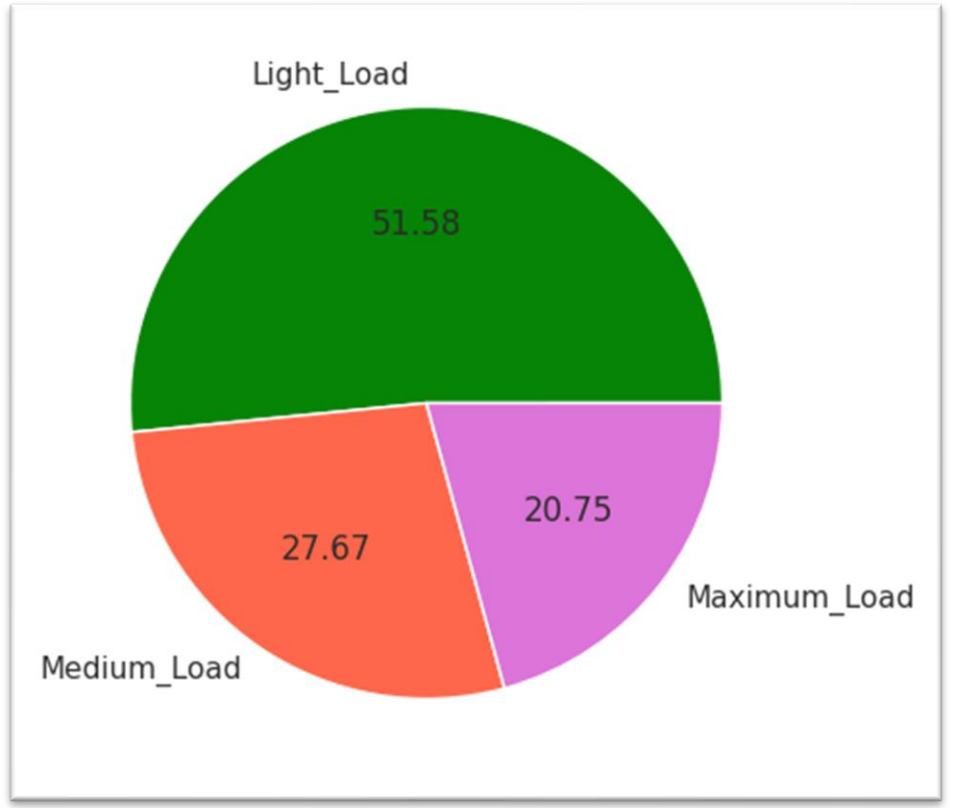
Pie chart

WeekStatus



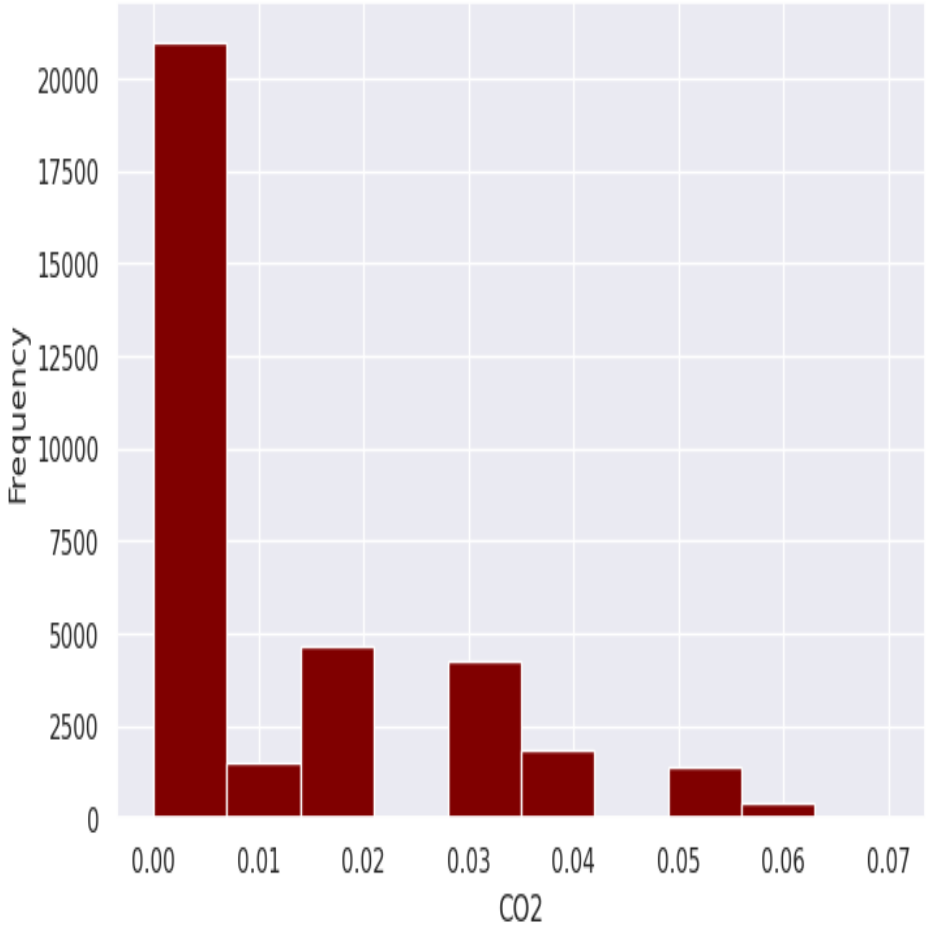
- Which gives the percentage of weekday and weekend

Load Type

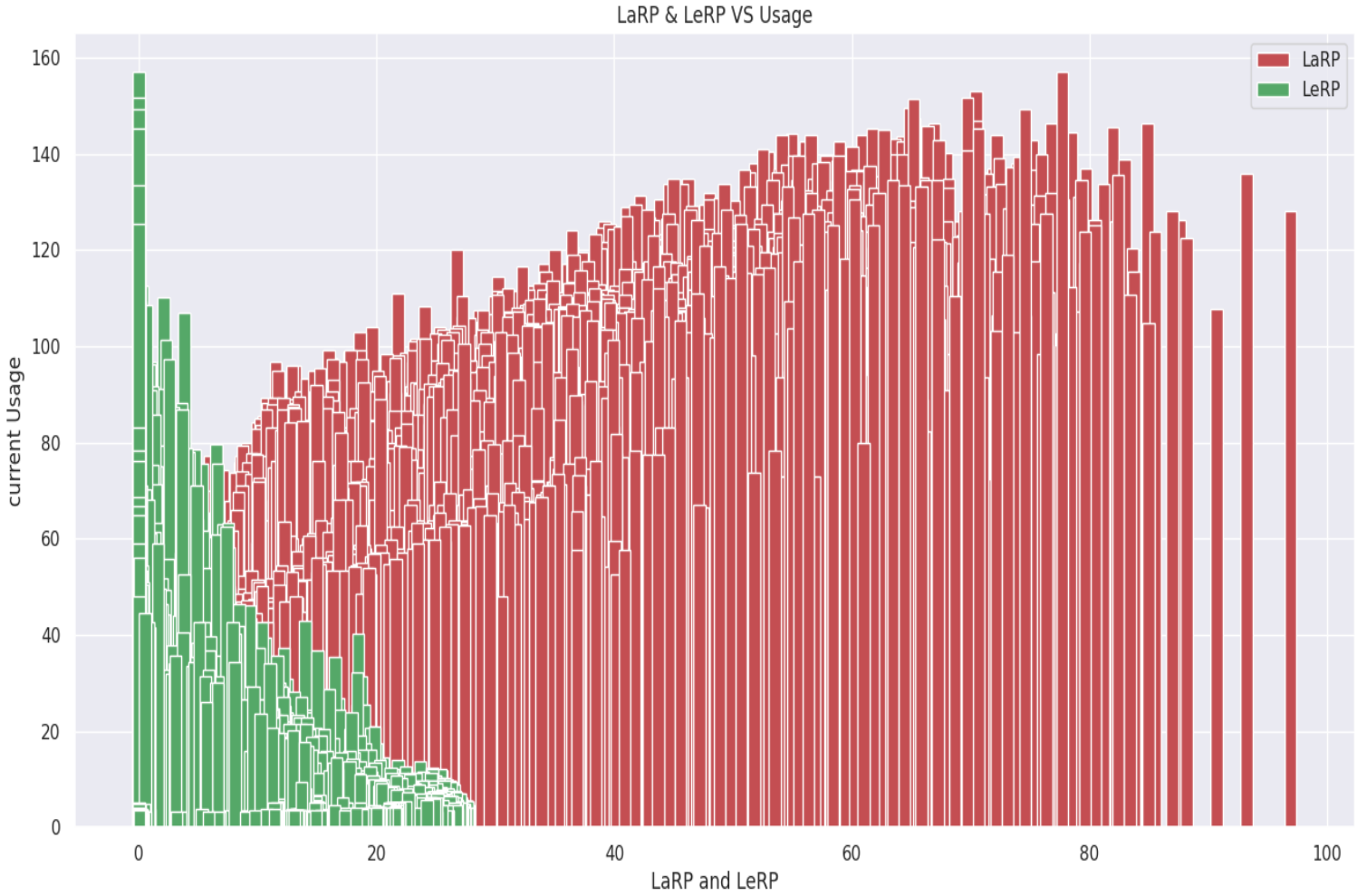


- Which gives the percentage of different factors in load type

Bar plots



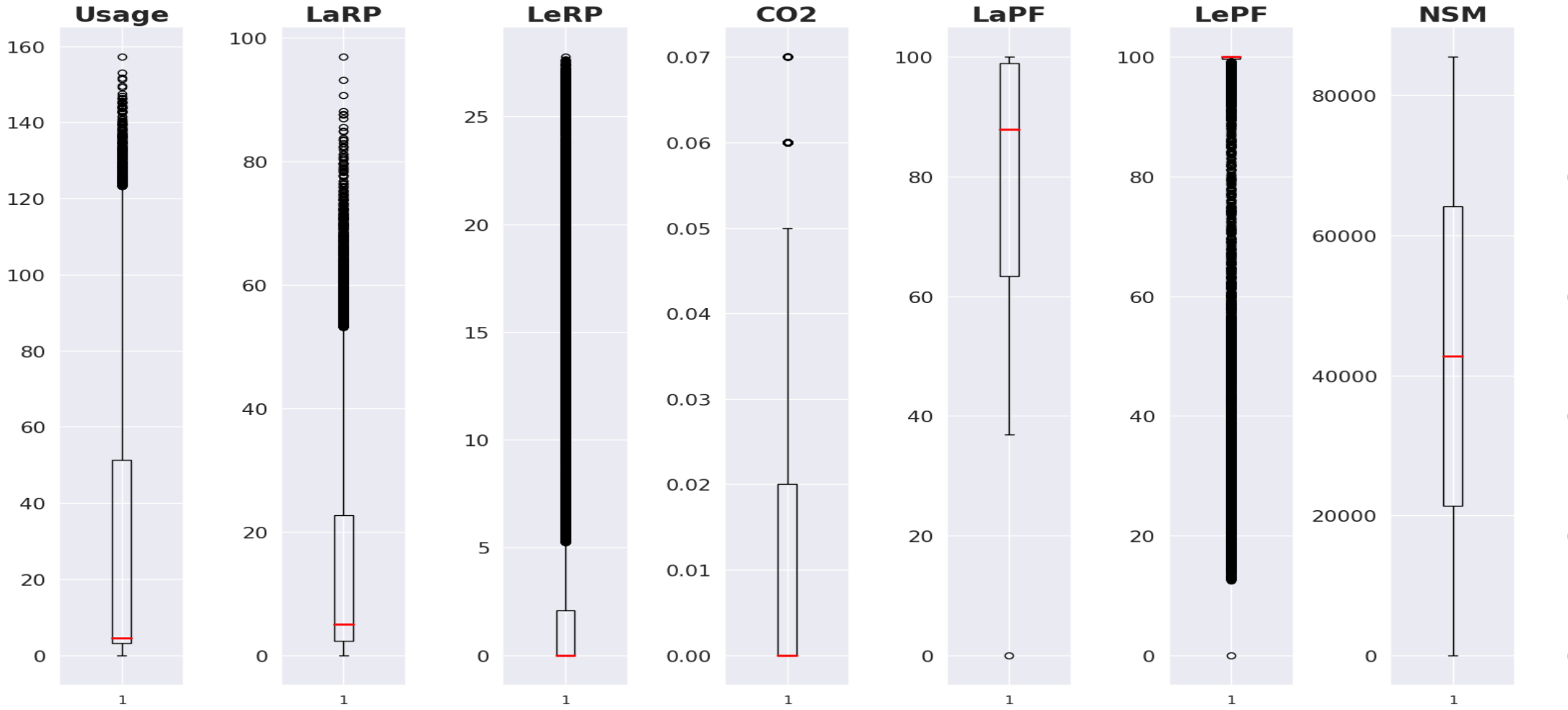
Histogram of CO2



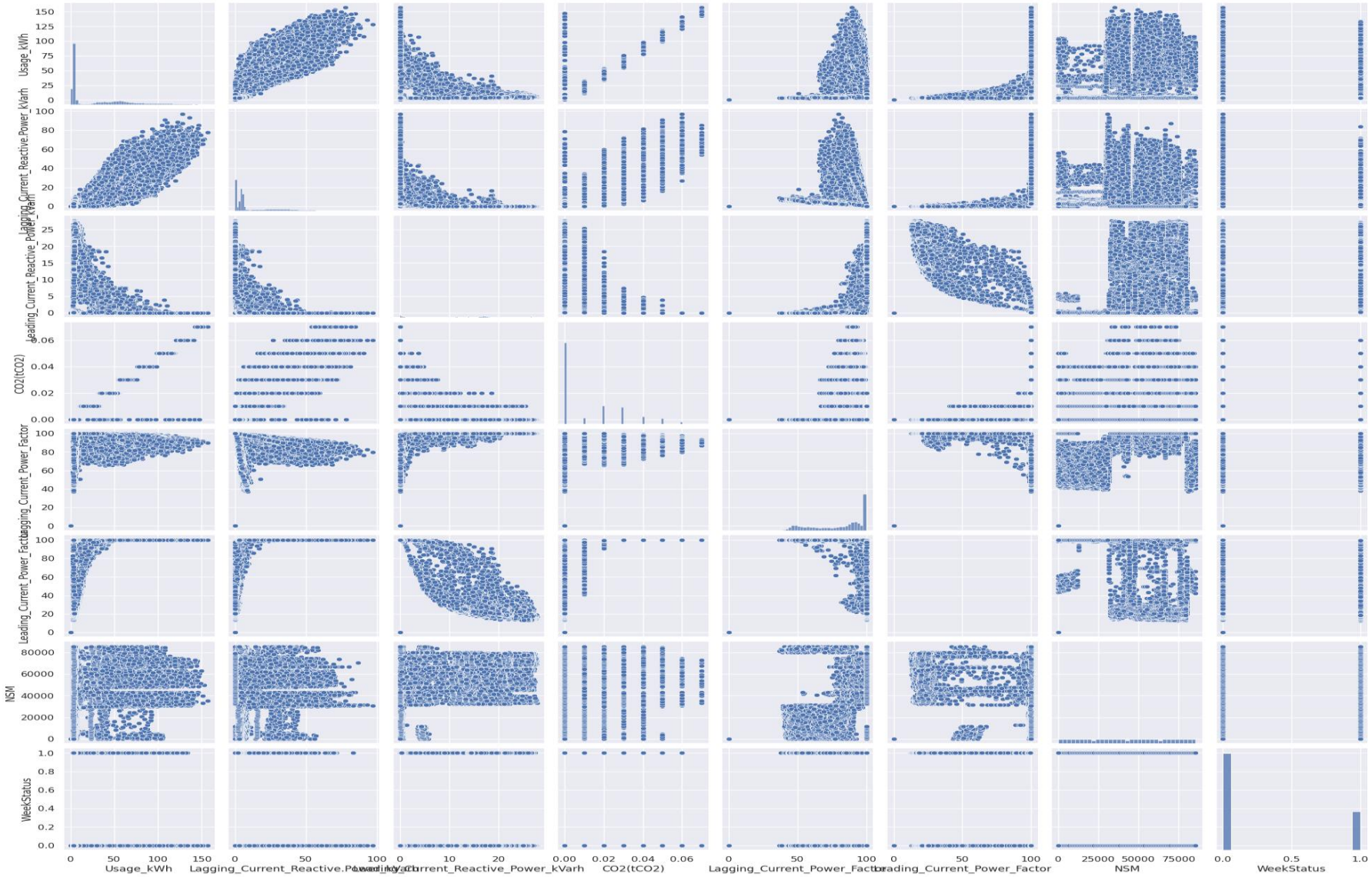
As usage increases, Lagging current reactive power increases and Leading current reactive power decreases

Box Plot

- Leading current power factor(LePF) consists of more outliers



Pair plot



- Lagging current reactive power is positively correlated with respect to Usage
- Leading current power factor is most negatively correlated with respect to leading current reactive power

Significance Test

- Fitting ordinary least squares model with all the features

OLS Regression Results

Dep. Variable:	Usage	R-squared:	0.918
Model:	OLS	Adj. R-squared:	0.918
Method:	Least Squares	F-statistic:	3.410e+04
Date:	Fri, 09 Aug 2024	Prob (F-statistic):	0.00
Time:	15:00:17	Log-Likelihood:	-90384.
No. Observations:	24528	AIC:	1.808e+05
Df Residuals:	24519	BIC:	1.809e+05
Df Model:	8		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
LaRP	1.4454	0.005	292.128	0.000	1.436	1.455
LeRP	0.0004	0.027	0.013	0.990	-0.053	0.054
LaPF	0.6185	0.005	114.336	0.000	0.608	0.629
LePF	0.2894	0.007	41.474	0.000	0.276	0.303
NSM	7.927e-06	3.42e-06	2.318	0.020	1.22e-06	1.46e-05
WeekStatus	1.3135	0.151	8.680	0.000	1.017	1.610
LL	-70.0792	0.875	-80.065	0.000	-71.795	-68.364
MaxL	-63.2756	1.009	-62.714	0.000	-65.253	-61.298
MinL	-61.9739	0.997	-62.184	0.000	-63.927	-60.020

Omnibus:	4005.600	Durbin-Watson:	2.000
Prob(Omnibus):	0.000	Jarque-Bera (JB):	12273.172
Skew:	0.851	Prob(JB):	0.00
Kurtosis:	6.018	Cond. No.	1.33e+06

- Leading Current Reactive Power(LeRP) has Greatest P-value which is insignificant.

Significance Test

OLS Regression Results

Dep. Variable:	Usage	R-squared:	0.916
Model:	OLS	Adj. R-squared:	0.916
Method:	Least Squares	F-statistic:	1.628e+04
Date:	Fri, 09 Aug 2024	Prob (F-statistic):	0.00
Time:	15:00:17	Log-Likelihood:	-38902.
No. Observations:	10512	AIC:	7.782e+04
Df Residuals:	10504	BIC:	7.788e+04
Df Model:	7		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-50.4578	0.770	-65.510	0.000	-51.968	-48.948
LaRP	1.4327	0.008	190.578	0.000	1.418	1.447
LaPF	0.6372	0.008	75.537	0.000	0.621	0.654
LePF	0.2971	0.005	61.192	0.000	0.288	0.307
NSM	9.996e-06	5.34e-06	1.871	0.061	-4.74e-07	2.05e-05
WeekStatus	1.4742	0.233	6.316	0.000	1.017	1.932
LL	-21.6136	0.210	-102.925	0.000	-22.025	-21.202
MaxL	-14.7457	0.381	-38.658	0.000	-15.493	-13.998
MinL	-14.0984	0.337	-41.869	0.000	-14.758	-13.438

Omnibus:	1882.893	Durbin-Watson:	1.987
Prob(Omnibus):	0.000	Jarque-Bera (JB):	5773.426
Skew:	0.927	Prob(JB):	0.00
Kurtosis:	6.122	Cond. No.	7.58e+19

- After removing variable having greatest P-value (greater than 0.05)

❑ NSM has greatest P-value

Multi collinearity check

	variables	VIF
0	LaRP	1.7
1	LeRP	10.9
2	LaPF	2.8
3	LePF	11.9
4	NSM	1.9
5	WeekStatus	1.2
6	LL	104.2
7	MaxL	56.0
8	MinL	72.3

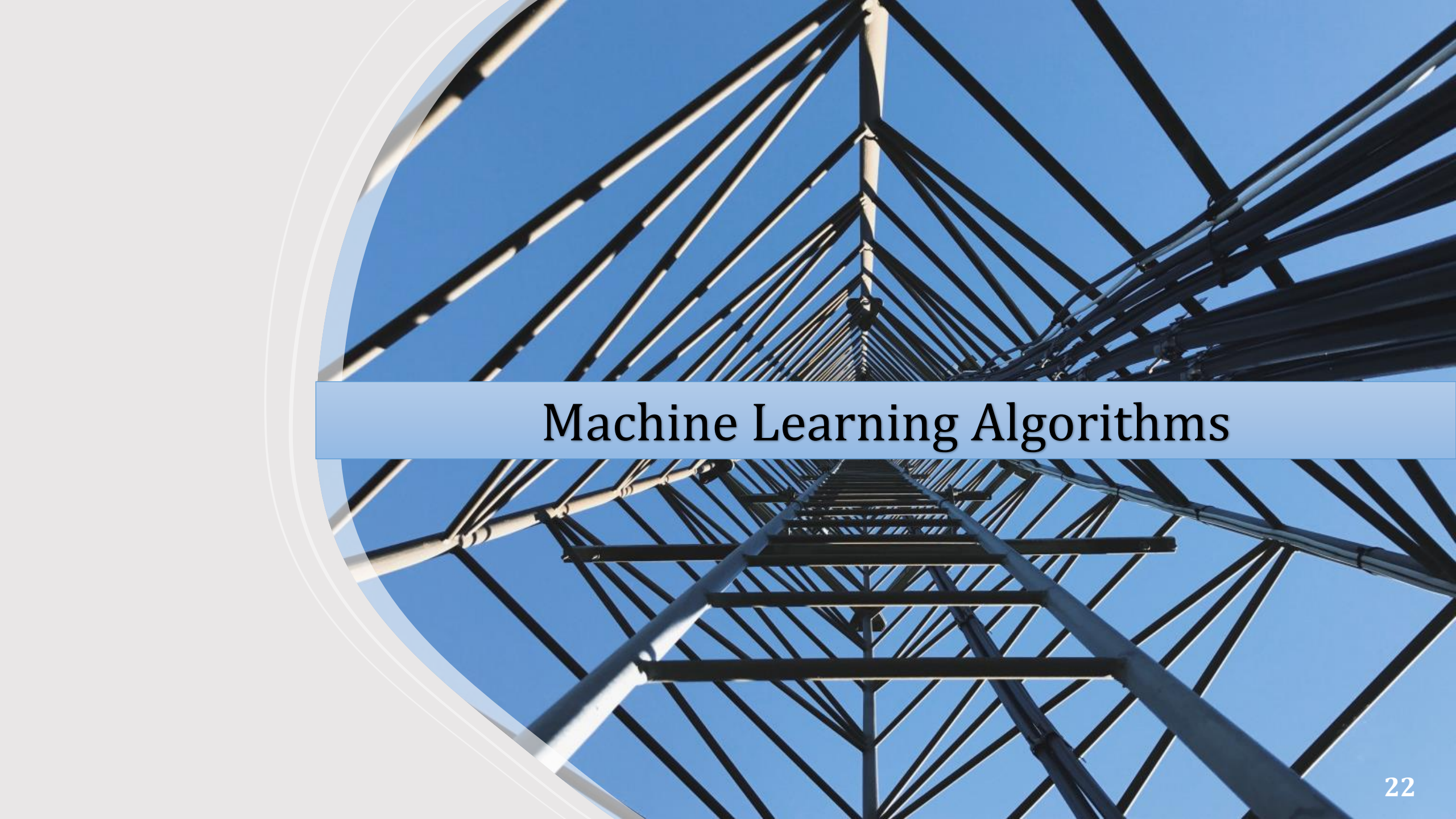
Light Load (LL) as removed

- Variables with the greatest variance inflation factor (VIF > 10) were removed

	variables	VIF
0	LaRP	2.8
1	LeRP	6.3
2	LaPF	33.1
3	LePF	15.8
4	NSM	7.2
5	WeekStatus	1.7
6	MaxL	2.4
7	MinL	2.7

Lagging current Power factor(LaPF) was removed

	variables	VIF
0	LaRP	2.6
1	LeRP	2.4
2	LePF	3.9
3	NSM	6.6
4	WeekStatus	1.6
5	MaxL	2.2
6	MinL	2.6



Machine Learning Algorithms

ML Algorithms

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

80:20 Train-Test Split

Algorithms	Model-1 (r2 score)	Model-2 (r2 score)	Model-3 (r2 score)	Model-1 (MAE)	Model-2 (MAE)	Model-3 (MAE)
Linear Regression	0.916	0.873	0.919	6.95	8.40	6.86
KNN	0.987	0.988	0.998	1.54	1.46	0.64
Decision Tree	0.894	0.894	0.895	5.88	5.86	5.94
Random Forest	0.999	0.999	0.999	0.20	0.19	0.39
Bagging	0.996	0.997	0.997	0.84	0.64	0.67
Boosting	0.995	0.996	0.996	1.21	1.22	1.22
SVM	0.907			0.196		

Model 1: *With all features* | ***Model 2:*** *After removing multi-collinear variables* | ***Model 3:*** *After removing insignificant variables from all the features*

75:25 Train-Test Split

Algorithms	Model-1 (r2 score)	Model-2 (r2 score)	Model-3 (r2 score)	Model-1 (MAE)	Model-2 (MAE)	Model-3 (MAE)
Linear Regression	0.917	0.870	0.915	6.83	8.36	6.97
KNN	0.984	0.984	0.997	1.75	1.76	0.66
Decision Tree	0.889	0.895	0.893	6.17	5.78	6.03
Random Forest	0.999	0.999	0.999	0.22	0.21	0.40
Bagging	0.996	0.997	0.997	0.87	0.70	0.68
Boosting	0.995	0.995	0.995	1.23	1.23	1.23
SVM	0.906			0.198		

Model 1: *With all features* | ***Model 2:*** *After removing multi-collinear variables* | ***Model 3:*** *After removing insignificant variables from all the features*

70:30 Train-Test Split

Algorithms	Model-1 (r2 score)	Model-2 (r2 score)	Model-3 (r2 score)	Model-1 (MAE)	Model-2 (MAE)	Model-3 (MAE)
Linear Regression	0.916	0.873	0.916	6.93	8.41	6.92
KNN	0.979	0.980	0.997	2.02	1.97	0.65
Decision Tree	0.895	0.895	0.894	5.98	5.93	5.85
Random Forest	0.999	0.999	0.999	0.21	0.23	0.41
Bagging	0.995	0.996	0.996	0.97	0.85	0.70
Boosting	0.995	0.998	0.996	1.25	1.21	1.21
SVM	0.906			0.196		

Model 1: *With all features* | ***Model 2:*** *After removing multi-collinear variables* | ***Model 3:*** *After removing insignificant variables from all the features*

80:20 Train-Test Split

Algorithms	Model-1 (r2 score)	Model-2 (r2 score)	Model-3 (r2 score)	Model-1 (MAE)	Model-2 (MAE)	Model-3 (MAE)
Linear Regression	0.917	0.874	0.918	6.86	8.32	6.93
KNN	0.973	0.978	0.997	2.29	2.09	0.66
Decision Tree	0.891	0.893	0.894	6.07	5.96	5.97
Random Forest	0.999	0.999	0.999	0.24	0.24	0.43
Bagging	0.996	0.996	0.997	1.04	0.85	0.75
Boosting	0.995	0.995	0.995	1.26	1.22	1.22
SVM	0.907			0.195		

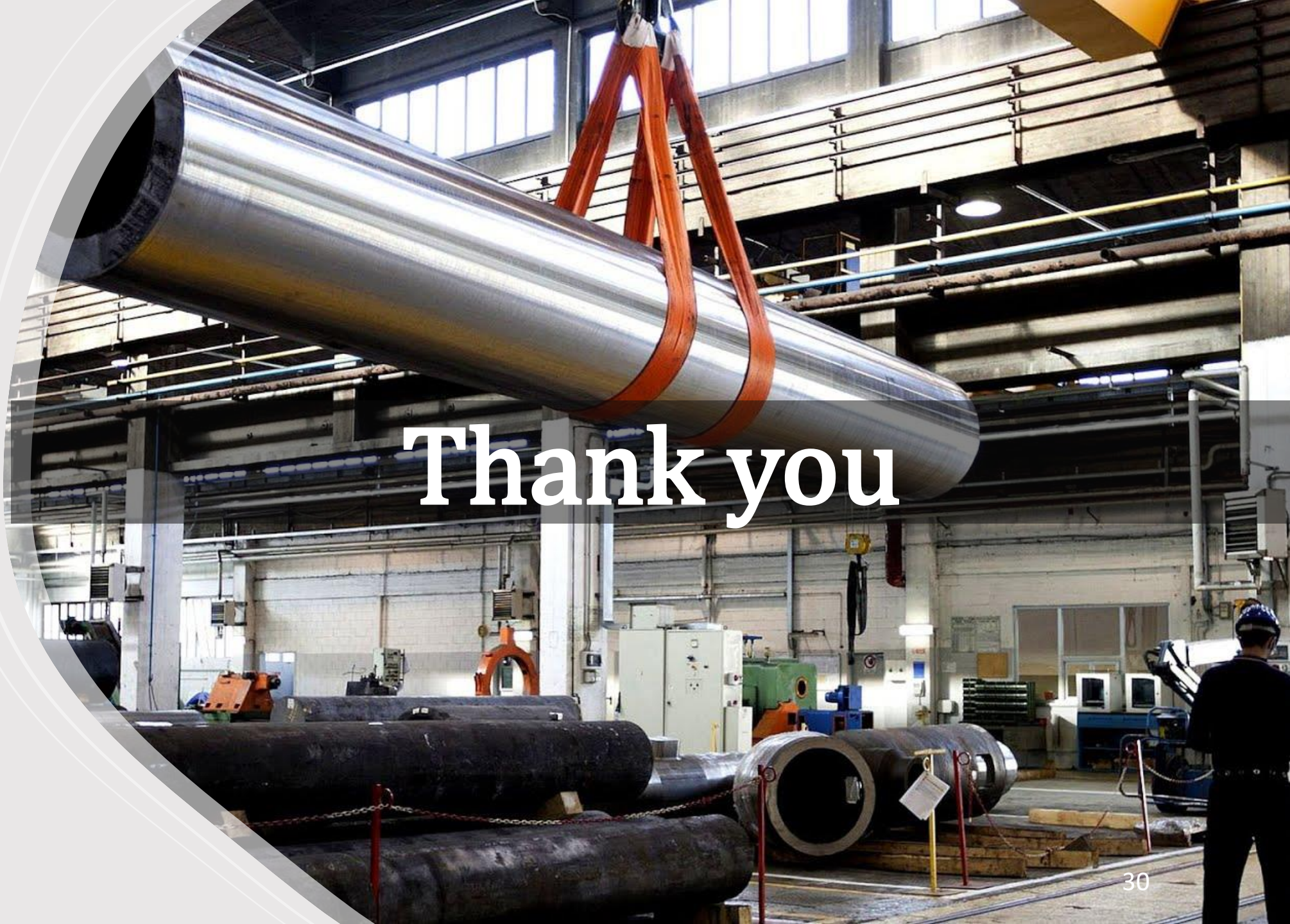
Model 1: *With all features* | ***Model 2:*** *After removing multi-collinear variables* | ***Model 3:*** *After removing insignificant variables from all the features*

Comparison of Algorithms

Algorithms	R- Square	MAE
Linear Regression	0.919	6.86
K- nearest numbers	0.998	6.64
Decision tree	0.895	5.78
Random forest	0.999	0.19
Bagging	0.997	0.64
Boosting	0.998	1.21
Support vector machine	0.907	0.195

Summary

- The purpose of this research is to determine the best performing machine learning techniques to predict the energy consumption in the steel industry.
- The Random Forest(RF) model improves Mean Absolute Error(MAE) and R-square of predictions in consideration to other regression models considered in this research



Thank you

Appendix

Loading the Dataset

loading dataset

```
[ ] data=pd.read_csv('Steel_industry_data.csv')
    data.head()
```

	date	Usage_kWh	Lagging_Current_Reactive.Power_kVarh	Leading_Current_Reactive_Power_kVarh	CO2(tCO2)	Lagging_Current_Power_Factor	Leading_Current_Power_Factor	NSM	WeekSt
0	01/01/2018 00:15	3.17	2.95	0.0	0.0	73.21	100.0	900	Wee
1	01/01/2018 00:30	4.00	4.46	0.0	0.0	66.77	100.0	1800	Wee
2	01/01/2018 00:45	3.24	3.28	0.0	0.0	70.28	100.0	2700	Wee
3	01/01/2018 01:00	3.31	3.56	0.0	0.0	68.09	100.0	3600	Wee
4	01/01/2018 01:15	3.82	4.50	0.0	0.0	64.72	100.0	4500	Wee

Null Values

finding null values

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

	0
date	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
Day_of_week	0
Load_Type	0

dtype: int64

Checking for the data type

checking datatype

```
[ ] data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 35040 entries, 0 to 35039
Data columns (total 11 columns):
 #   Column                                Non-Null Count  Dtype
---  ------                                --------------  -----
 0   date                                  35040 non-null  object
 1   Usage_kWh                             35040 non-null  float64
 2   Lagging_Current_Reactive.Power_kVarh  35040 non-null  float64
 3   Leading_Current_Reactive_Power_kVarh  35040 non-null  float64
 4   CO2(tCO2)                             35040 non-null  float64
 5   Lagging_Current_Power_Factor          35040 non-null  float64
 6   Leading_Current_Power_Factor          35040 non-null  float64
 7   NSM                                   35040 non-null  int64
 8   WeekStatus                            35040 non-null  object
 9   Day_of_week                           35040 non-null  object
 10  Load_Type                             35040 non-null  object
dtypes: float64(6), int64(1), object(4)
memory usage: 2.9+ MB
```

Enlabling the variable WeekStatus

```
X['WeekStatus']=X['WeekStatus'].replace({'Weekday':0,'Weekend':1})
print(X['WeekStatus'])
```

```
0        0
1        0
2        0
3        0
4        0
        ..
35035    0
35036    0
35037    0
35038    0
35039    0
Name: WeekStatus, Length: 35040, dtype: int64
```

Dividing the data set

Data-1

```
data1=X.iloc[:,:8]
data1.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
0	3.17	2.95	0.0	0.0	73.21	100.0	900	0
1	4.00	4.46	0.0	0.0	66.77	100.0	1800	0
2	3.24	3.28	0.0	0.0	70.28	100.0	2700	0
3	3.31	3.56	0.0	0.0	68.09	100.0	3600	0
4	3.82	4.50	0.0	0.0	64.72	100.0	4500	0

Data-2

```
data2=X.iloc[:,8:]
data3=pd.get_dummies(data2)
data3=data3.astype(int)
data3.head()
```

	Day_of_week_Friday	Day_of_week_Monday	Day_of_week_Saturday	Day_of_week_Sunday	Day_of_week_Thursday	Day_of_week_Tuesday	Day_of_week_Wednesday	Load_Type_Light_Load	Load_Type_M
0	0	1	0	0	0	0	0	1	
1	0	1	0	0	0	0	0	1	
2	0	1	0	0	0	0	0	1	
3	0	1	0	0	0	0	0	1	
4	0	1	0	0	0	0	0	1	

Renaming the column names and dropping the days from the data

```
[ ] df.rename(columns={'Usage_kWh':'Usage','Lagging_Current_Reactive.Power_kVarh':'LaRP','Leading_Current_Reactive_Power_kVarh':'LeRP','CO2(tCO2)':'CO2','Lagging_Current_Power_Factor':'
    df.head()
```

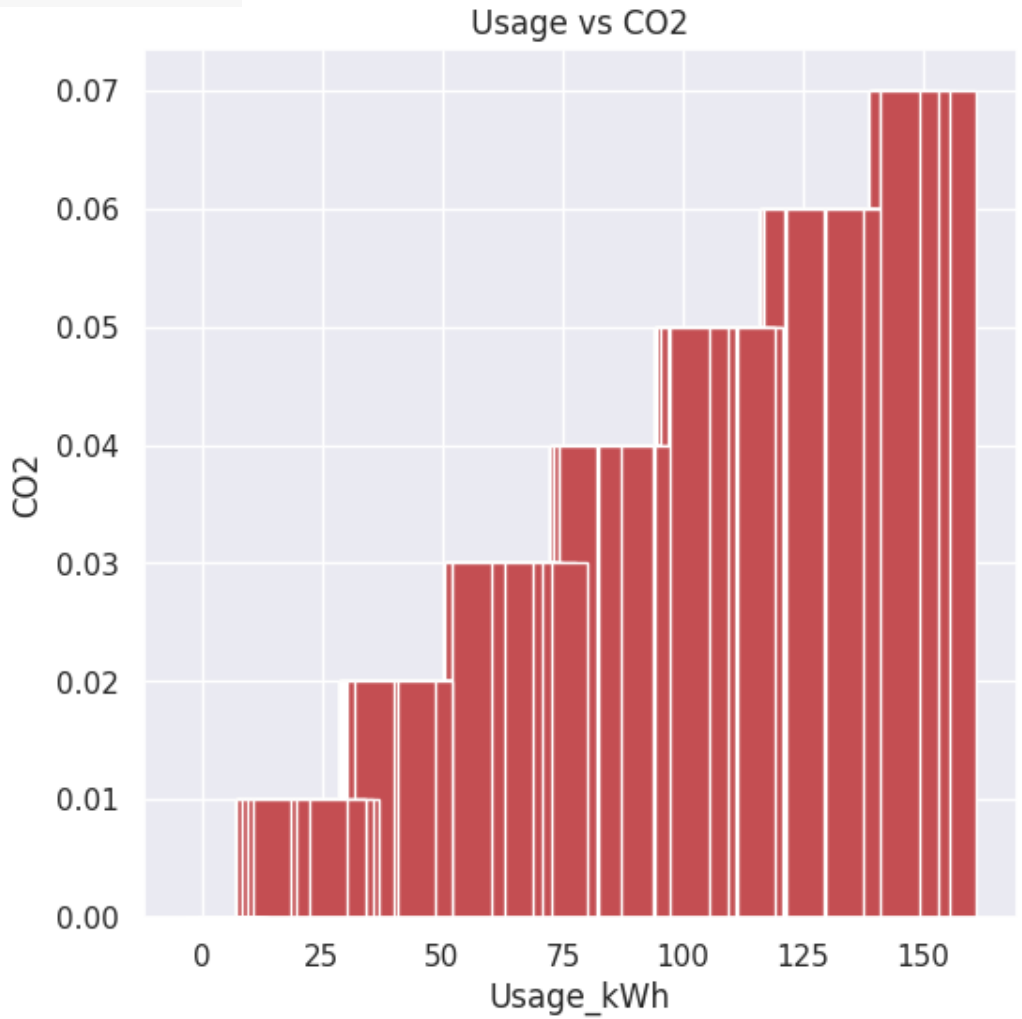
	Usage	LaRP	LeRP	CO2	LaPF	LePF	NSM	WeekStatus	Fr	Mo	Sa	Su	Th	Tu	We	LL	MaxL	MinL
0	3.17	2.95	0.0	0.0	73.21	100.0	900	0	0	1	0	0	0	0	0	1	0	0
1	4.00	4.46	0.0	0.0	66.77	100.0	1800	0	0	1	0	0	0	0	0	1	0	0
2	3.24	3.28	0.0	0.0	70.28	100.0	2700	0	0	1	0	0	0	0	0	1	0	0
3	3.31	3.56	0.0	0.0	68.09	100.0	3600	0	0	1	0	0	0	0	0	1	0	0
4	3.82	4.50	0.0	0.0	64.72	100.0	4500	0	0	1	0	0	0	0	0	1	0	0

```
[ ] df1=df.drop(['Fr','Sa','Su','Mo','Tu','We','Th'],axis=1)
    df1.head()
```

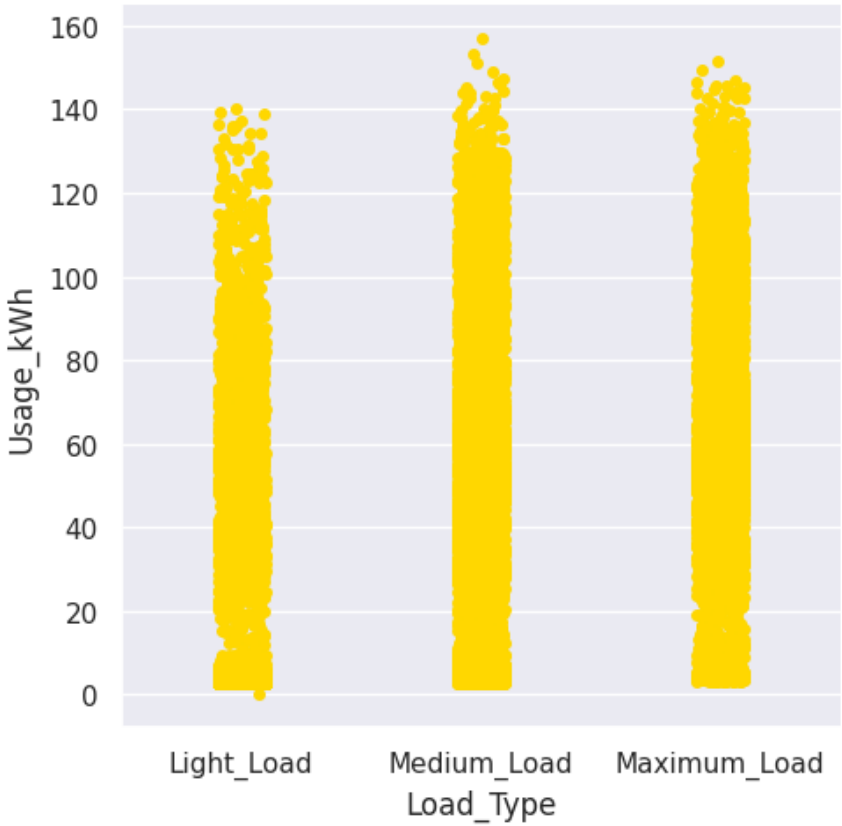
	Usage	LaRP	LeRP	CO2	LaPF	LePF	NSM	WeekStatus	LL	MaxL	MinL
0	3.17	2.95	0.0	0.0	73.21	100.0	900	0	1	0	0

Bar Plots

```
plt.hist(df['CO2'],color='maroon',bins=10)
plt.xlabel('CO2')
plt.ylabel('Frequency')
plt.show()
```




```
catplot=sns.catplot(x='Load_Type',y='Usage_
kWh',data=data,color='gold')
catplot.set_ylabels('Usage_kWh')
catplot.set_xlabels('Load_Type')
catplot
```



Multi collinearity check

```
[40] import warnings
     warnings.filterwarnings("ignore")
```

Loading...

```
x=df1.drop(['Usage','CO2'],axis=1)
y=df1['Usage']
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.30,random_state=0)
```

```
[42] from statsmodels.stats.outliers_influence import variance_inflation_factor
     def calc_vif(x):
        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_train)
```

	variables	VIF
0	LaRP	1.7
1	LeRP	10.9
2	LaPF	2.8

Ordinary least squares

```
[45] import statsmodels.api as sm
     x_train_constant=sm.add_constant(x_train)
     model=sm.OLS(y_train, x_train).fit()
     print(model.summary())
```

Assigning constants to their respective variables

```
[50] feature=['LaRP','LeRP','LaPF','LePF','NSM','WeekStatus','LL']
     x_1=df1[feature]
     y_1=df1.Usage
     lm1=LinearRegression()
     lm1.fit(x_1,y_1)
     print(lm1.intercept_)
     print(lm1.coef_)
```

Linear Regression

```
[34] import statsmodels.formula.api as smf
     from sklearn.linear_model import LinearRegression
     from sklearn import metrics
     from sklearn.model_selection import train_test_split
     from sklearn.metrics import mean_absolute_error
     from sklearn.metrics import mean_absolute_percentage_error
     from sklearn.metrics import r2_score
```

```
x=df[['L Loading... ','LePF','LaPF','NSM','WeekStatus','LL']]
y=df['Usage']
x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.3,test_size=0.70)
lm=LinearRegression()
lm.fit(x_train,y_train)
y_pred=lm.predict(x_test)
print(y_pred)
print("MSE:",np.sqrt(metrics.mean_squared_error(y_test,y_pred)))
print("MAE:",mean_absolute_error(y_test,y_pred))
print("MAPE:",mean_absolute_percentage_error(y_test,y_pred))
print("R2:",r2_score(y_test,y_pred))
```

```
[ -2.85037583  -1.1076037  109.61232984 ...   7.22512842   5.26807267
   3.76242493]
MSE: 9.617956563957458
MAE: 6.929594887925694
MAPE: 0.8556369773437488
R2: 0.9171861091252882
```

```
[61] print(lm2.score(x_test_1,y_test_1))

     print(lm.score(x_test,y_test))
```

```
0.8721025142304913
0.916042677572706
```

KNN

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

before multicollinearity

```
[63] model_k1=KNeighborsRegressor(n_neighbors=5)
     model_k1.fit(x_train,y_train)
     y_pred_k1=model_k1.predict(x_test)
     y_pred_k1
     knn=pd.DataFrame({'Predicted':y_pred_k1,'Actual':y_test})
     print(knn)
```

K-Nearest Neighbors

Decision tree regressor

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

```
split_ratios = [0.25, 0.2, 0.3, 0.4]

for ratio in split_ratios:
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=ratio, random_state=1)
    DT_model = DecisionTreeRegressor()
    DT_model.fit(x_train, y_train)
    Dt_y_pred = DT_model.predict(x_test)
    DT_MAE = mean_absolute_error(y_test, Dt_y_pred)
    DT_R2 = r2_score(y_test, Dt_y_pred)
    print(f"Split Ratio: {ratio}, MAE: {DT_MAE}, R Score: {DT_R2}")
```

```
Split Ratio: 0.25, MAE: 0.45835273972602736, R Score: 0.9988441810908901
Split Ratio: 0.2, MAE: 0.4331678082191781, R Score: 0.9989471889594841
Split Ratio: 0.3, MAE: 0.4859151445966514, R Score: 0.9985369299931104
Split Ratio: 0.4, MAE: 0.5228510273972603, R Score: 0.9982600719463054
```

```
split_ratios = [0.25, 0.2, 0.3, 0.4]

for ratio in split_ratios:
    x_train_1, x_test_1, y_train_1, y_test_1 = train_test_split(x_1, y_1, test_size=ratio, random_state=1)
    DT_model = De Loading... Regressor()
    DT_model.fit(x_train_1, y_train_1)
    Dt_y_pred = DT_model.predict(x_test_1)
    DT_MAE = mean_absolute_error(y_test_1, Dt_y_pred)
    DT_R2 = r2_score(y_test_1, Dt_y_pred)
    print(f"Split Ratio: {ratio}, MAE: {DT_MAE}, R Score: {DT_R2}")
```

```
Split Ratio: 0.25, MAE: 0.44680365296803654, R Score: 0.9988787544978057
Split Ratio: 0.2, MAE: 0.4269848744292237, R Score: 0.9990616098032482
Split Ratio: 0.3, MAE: 0.47457952815829524, R Score: 0.9985833453735368
Split Ratio: 0.4, MAE: 0.5202953767123287, R Score: 0.9982090216949071
```

Decision tree regressor

```
split_ratios = [0.25, 0.2, 0.3, 0.4]

for ratio i Loading... tios:
    x_train_2, x_test_2, y_train_2, y_test_2 = train_test_split(x_is, y_is, test_size=ratio, random_state=1)
    DT_model = DecisionTreeRegressor()
    DT_model.fit(x_train_2, y_train_2)
    Dt_y_pred = DT_model.predict(x_test_2)
    DT_MAE = mean_absolute_error(y_test_2, Dt_y_pred)
    DT_R2 = r2_score(y_test_2, Dt_y_pred)
    print(f"Split Ratio: {ratio}, MAE: {DT_MAE}, R Score: {DT_R2}")
```

```
Split Ratio: 0.25, MAE: 0.622840378343118, R Score: 0.9982393388241929
Split Ratio: 0.2, MAE: 0.6173303707327681, R Score: 0.998227691248836
Split Ratio: 0.3, MAE: 0.6475983184750308, R Score: 0.9980403244628812
Split Ratio: 0.4, MAE: 0.6665822200931363, R Score: 0.9980880618400884
```

Random forest regressor

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

```
split_ratios = [0.25, 0.2, 0.3, 0.4]

for ratio in split_ratios:
    x_train_1,x_test_1, y_train_1, y_test_1 = train_test_split(x_1, y_1, test_size=ratio, random_state=1)
    rf_model = RandomForestRegressor()
    rf_model.fit(x_train_1, y_train_1)
    Rf_y_pred = rf_model.predict(x_test_1)
    RF_mae = mean_absolute_error(y_test_1, Rf_y_pred)
    RF_R2 = r2_score(y_test_1, Rf_y_pred)
    print(f"Split Ratio: {ratio},MAE: {RF_mae}, R Score: {RF_R2}")
```

```
Split Ratio: 0.25,MAE: 0.21671069634703186, R Score: 0.9995936579850507
Split Ratio: 0.2,MAE: 0.2134381135844748, R Score: 0.9995944227735726
Split Ratio: 0.3,MAE: 0.23187499048706192, R Score: 0.9994694301735966
Split Ratio: 0.4,MAE: 0.2532447203196343, R Score: 0.9993578099208046
```

```python
split_ratios = [0.25, 0.2, 0.3, 0.4]

for ratio in split_ratios:
    x_train,x_test, y_train, y_test = train_test_split(x, y, test_size=ratio, random_state=1)
    rf_model = RandomForestRegressor()
    rf_model.fit(x_train, y_train)
    Rf_y_pred = rf_model.predict(x_test)
    RF_mae = mean_absolute_error(y_test, Rf_y_pred)
    RF_R2 = r2_score(y_test, Rf_y_pred)
    print(f"Split Ratio: {ratio},MAE: {RF_mae}, R Score: {RF_R2}")
```

```
Split Ratio: 0.25,MAE: 0.2190894863013695, R Score: 0.9995807309513717
Split Ratio: 0.2,MAE: 0.21734531963470266, R Score: 0.9995677084327599
Split Ratio: 0.3,MAE: 0.23442719748858412, R Score: 0.9994642375231478
Split Ratio: 0.4,MAE: 0.25518837756849266, R Score: 0.999369590150777
```

Random forest regressor

```python
split_ratios = [0.25, 0.2, 0.3, 0.4]

for ratio in split_ratios:
    x_train_2,x_test_2, y_train_2, y_test_2 = train_test_split(x_is, y_is, test_size=ratio, random_state=1)
    rf_model = RandomForestRegressor()
    rf_model.fit(x_train_2, y_train_2)
    Rf_y_pred = rf_model.predict(x_test_2)
    RF_mae = mean_absolute_error(y_test_2, Rf_y_pred)
    RF_R2 = r2_score(y_test_2, Rf_y_pred)
    print(f"Split Ratio: {ratio}, MAE: {RF_mae}, R Score: {RF_R2}")
```

```
Split Ratio: 0.25, MAE: 0.40385184418661285, R Score: 0.9990758733517252
Split Ratio: 0.2, MAE: 0.4049003793089591, R Score: 0.9990864131905693
Split Ratio: 0.3, MAE: 0.4152956853850844, R Score: 0.9990326814826408
Split Ratio: 0.4, MAE: 0.4308729950146092, R Score: 0.998950457615234
```

Bagging Regressor

```
[84] from sklearn.ensemble import BaggingRegressor
```

```
base_model_2 = DecisionTreeRegressor(max_features = "sqrt")
```

```
[86] split_ratios = [0.25, 0.2, 0.3, 0.4]
     base_model_2 = DecisionTreeRegressor(max_features = "sqrt")

     for ratio in split_ratios:
         x_train_2, x_test_2, y_train_2, y_test_2 = train_test_split(x_is, y_is, test_size=ratio, random_state=1)
         ba_model_2 = BaggingRegressor(base_model_2, n_estimators = 100, random_state = 1)
         ba_model_2.fit(x_train_2, y_train_2)
         Ba_y_pred_2 = ba_model_2.predict(x_test_2)
         BA_MAE_2 = mean_absolute_error(y_test_2, Ba_y_pred_2)
         BA_R2_2 = r2_score(y_test_2, Ba_y_pred_2)
         print(f"Split Ratio: {ratio}, MAE: {BA_MAE_2}, R Score: {BA_R2_2}")
```

```
Split Ratio: 0.25, MAE: 0.544223614047841, R Score: 0.9985305569168326
Split Ratio: 0.2, MAE: 0.545021704211197, R Score: 0.998507459516208
Split Ratio: 0.3, MAE: 0.5676244393481168, R Score: 0.9983778104466098
Split Ratio: 0.4, MAE: 0.5977291713796768, R Score: 0.9981344791470439
```

Bagging Regressor

```
split_ratios = [0.25, 0.2, 0.3, 0.4]
base_model_2 = DecisionTreeRegressor(max_features = "sqrt")
for ratio in split_ratios:
    x_train_1, x_test_1, y_train_1, y_test_1 = train_test_split(x_1, y_1, test_size=ratio, random_state=1)
    ba_model_2 = BaggingRegressor(base_model_2, n_estimators = 100, random_state = 1)
    ba_model_2.fit(x_train_1, y_train_1)
    y_pred_2 = ba_model_2.predict(x_test_1)
    BA_MAE_2 = mean_absolute_error(y_test_1, y_pred_2)
    BA_R2_2 = r2_score(y_test_1, y_pred_2)
    print(f"Split Ratio: {ratio}, MAE: {BA_MAE_2}, R Score: {BA_R2_2}")
```

```
Split Ratio: 0.25, MAE: 0.6785845928462709, R Score: 0.9975424579670079
Split Ratio: 0.2, MAE: 0.6666314783105022, R Score: 0.9975052883789308
Split Ratio: 0.3, MAE: 0.7320911339421613, R Score: 0.9970349890988559
Split Ratio: 0.4, MAE: 0.7667859565258753, R Score: 0.9966671014840396
```

```
split_ratios = [0.25, 0.2, 0.3, 0.4]

for ratio in split_ratios:
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=ratio, random_state=1)
    ba_model_2 = BaggingRegressor(base_model_2, n_estimators = 100, random_state = 1)
    ba_model_2.fit(x_train, y_train)
    y_pred_2 = ba_model_2.predict(x_test)
    BA_MAE_2 = mean_absolute_error(y_test, y_pred_2)
    BA_R2_2 = r2_score(y_test, y_pred_2)
    print(f"Split Ratio: {ratio}, MAE: {BA_MAE_2}, R Score: {BA_R2_2}")
```

```
Split Ratio: 0.25, MAE: 0.6916379566210044, R Score: 0.9974728663052979
Split Ratio: 0.2, MAE: 0.6802139554794518, R Score: 0.9975243108360045
Split Ratio: 0.3, MAE: 0.752752749238964, R Score: 0.9969104941630762
Split Ratio: 0.4, MAE: 0.7813861515410958, R Score: 0.9966128434311846
```

```
from sklearn.ensemble import GradientBoostingRegressor
```

```
[90] split_ratios = [0.25, 0.2, 0.3, 0.4]

     for ratio in split_ratios:
         x_train_2,x_test_2, y_train_2, y_test_2 = train_test_split(x_is, y_is, test_size=ratio, random_state=1)
         gf_model = GradientBoostingRegressor()
         gf_model.fit(x_train_2, y_train_2)
         Gf_y_pred = gf_model.predict(x_test_2)
         GF_mae = mean_absolute_error(y_test_2, Gf_y_pred)
         GF_R2 = r2_score(y_test_2, Gf_y_pred)
         print(f"Split Ratio: {ratio}, MAE: {GF_mae}, R Score: {GF_R2}")
```

```
Split Ratio: 0.25, MAE: 1.2328671479743718, R Score: 0.9959984996490892
Split Ratio: 0.2, MAE: 1.2352871657031563, R Score: 0.9958495160767151
Split Ratio: 0.3, MAE: 1.2114896906799077, R Score: 0.9959862528898511
Split Ratio: 0.4, MAE: 1.2246988730675172, R Score: 0.9958141737561723
```

Boosting Regressor

```
[91] split_ratios = [0.25, 0.2, 0.3, 0.4]

     for ratio in split_ratios:
         x_train_1,x_test_1, y_train_1, y_test_1 = train_test_split(x_1, y_1, test_size=ratio, random_state=1)
         gf_model = GradientBoostingRegressor()
         gf_model.fit(x_train_1, y_train_1)
         Gf_y_pred = gf_model.predict(x_test_1)
         GF_mae = mean_absolute_error(y_test_1, Gf_y_pred)
         GF_R2 = r2_score(y_test_1, Gf_y_pred)
         print(f"Split Ratio: {ratio}, MAE: {GF_mae}, R Score: {GF_R2}")
```

```
Split Ratio: 0.25, MAE: 1.219925950826706, R Score: 0.9958108002618445
Split Ratio: 0.2, MAE: 1.2097804693764909, R Score: 0.9958507843900986
Split Ratio: 0.3, MAE: 1.2048704764269205, R Score: 0.9958174402708889
Split Ratio: 0.4, MAE: 1.210402113364092, R Score: 0.9957398572129694
```

```
split_ratios = [0.25, 0.2, 0.3, 0.4]

for ratio in split_ratios:
    x_train,x_test, y_train, y_test = train_test_split(x, y, test_size=ratio, random_state=1)
    gf_model = GradientBoostingRegressor()
    gf_model.fit(x_train, y_train)
    Gf_y_pred = gf_model.predict(x_test)
    GF_mae = mean_absolute_error(y_test, Gf_y_pred)
    GF_R2 = r2_score(y_test, Gf_y_pred)
    print(f"Split Ratio: {ratio}, MAE: {GF_mae}, R Score: {GF_R2}")
```

```
Split Ratio: 0.25, MAE: 1.211245561008604, R Score: 0.9959262928209743
Split Ratio: 0.2, MAE: 1.2357187319674579, R Score: 0.9957246213633338
Split Ratio: 0.3, MAE: 1.2067620262821845, R Score: 0.9958738108439852
Split Ratio: 0.4, MAE: 1.2180435108117371, R Score: 0.9956436870515977
```

Standardizing the data

```
[93] from sklearn.preprocessing import StandardScaler
```

```
scaler=StandardScaler()
scaler.fit(df1)
std=scaler.transform(df1)
std
```

```
array([[-0.72410486, -0.61851634, -0.52138505, ...,  0.96897421,
        -0.51174621, -0.61852709],
       [-0.69928718, -0.52591107, -0.52138505, ...,  0.96897421,
        -0.51174621, -0.61852709],
       [-0.72201181, -0.59827811, -0.52138505, ...,  0.96897421,
        -0.51174621, -0.61852709],
       ...,
       [-0.70586536, -0.60502418, -0.51195662, ...,  0.96897421,
        -0.51174621, -0.61852709],
       [-0.70586536, -0.61177026, -0.50656895, ...,  0.96897421,
        -0.51174621, -0.61852709],
       [-0.70915445, -0.61422338, -0.51195662, ...,  0.96897421,
        -0.51174621, -0.61852709]])
```

Support vector regressor

```
[96] svr=SVR(kernel='linear')
```

```
[97] x_2=x_std.drop(['Usage','CO2'], axis=1)
     y_2=x_std.Usage
```

```
split_ratios = [0.25, 0.2, 0.3, 0.4]
svr=SVR(kernel='linear')

for ratio in split_ratios:
    x_train_3,x_test_3, y_train_3, y_test_3 = train_test_split(x_2, y_2, test_size=ratio, random_state=1)
    svr.fit(x_train_3, y_train_3)
    y_pred = svr.predict(x_test_3)
    mae = mean_absolute_error(y_test_3, y_pred)
    R2 = r2_score(y_test_3, y_pred)
    print(f"Split Ratio: {ratio}, MAE: {mae}, R Score: {R2}")
```

```
Split Ratio: 0.25, MAE: 0.1981836972595499, R Score: 0.9129520285482899
Split Ratio: 0.2, MAE: 0.19741579559588832, R Score: 0.9137366969581622
Split Ratio: 0.3, MAE: 0.2000915681292742, R Score: 0.9106783623045374
Split Ratio: 0.4, MAE: 0.19958803720511423, R Score: 0.9110495411177141
```