Title: STEEL INDUSTRY ENERGY CONSUMPTION PREDICTION

1. Objective:

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

2. Abstract:

Predictions of Energy Consumption for Industries gain an important place in energy management and control system, as there are dynamic and seasonal changes in the demand and supply of energy. This paper presents and discusses the predictive models for energy consumption of the steel industry

This article presents a machine learning model for predicting energy consumption in the steel industry, which aids in energy management, cost reduction, environmental regulation compliance, informed decision making for future energy investments, and contributes to sustainability. The dataset used for the prediction model comprises 11 attributes and 35,040 instances. The project focuses on the development of a model that accurately predicts energy consumption in the steel industry, leading to more sustainable and efficient practices.

Data used includes lagging and leading current reactive power, lagging and leading current power factor, carbon dioxide (tCO2) emission and load type. The study explores various machine learning algorithms like linear regression, decision tree, random forest, K-nearest neighbors, support vector machines, bagging and boosting to predict power consumption.

3. Steel Industry

3.1 Introduction to steel Industry:

The steel industry has grown from ancient times, when a few men may have operated, periodically, a small furnace producing 10 kilograms, to the modern integrated iron- and steelworks, with annual steel production of about 1 million tons.

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**.**

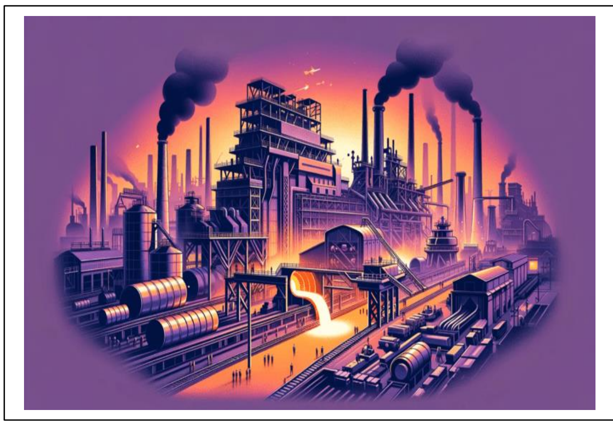
Steel companies are increasingly focusing on sustainability, including reducing carbon emissions and implementing eco-friendly production methods.

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

Due to the advancement of industrialization on a global scale and development of industry, the energy demand is elevated and is considered to be a major concern in national policy. Furthermore, economic growth and human development also add to the rapid growth of energy consumption. The Unregulated use of energy like over-consumption, poor infrastructure, and energy waste are the causes of such an outcome. Among the demanders of different sources of energy, Streimikiene forecasts a significant proportion of residential energy usage by 2030. As seen by Zuo, Energy consumptions in buildings cover 39 per cent of the total energy consumption of the United States

3.2 History

Steel was known in antiquity and was produced in bloomeries and crucibles.



The earliest known production of steel is seen in pieces of ironware excavated from an archaeological site in Anatolia (Kaman-Kalehoyuk) which are nearly 4,000 years old, dating from 1800 BC.

High-carbon steel was produced in Britain at Broxmouth Hillfort from 490–375 BC, and ultrahigh-carbon steel was produced in the Netherlands from the 2nd-4th centuries AD. The Roman author Horace identifies steel weapons such as the *falcata* in the Iberian Peninsula, while Noric steel was used by the Roman military.

There is evidence that carbon steel was made in Western Tanzania by the ancestors of the Haya people as early as 2,000 years ago by a complex process of "pre-heating" allowing temperatures inside a furnace to reach 1300 to 1400 °C.

3.3 Industry

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. In 1980, there were more than 500,000 U.S. steelworkers. By 2000, the number of steelworkers had fallen to 224,000.

The economic boom in China and India caused a massive increase in the demand for steel. Between 2000 and 2005, world steel demand increased by 6%. Since 2000, several Indian and Chinese steel firms have expanded to meet demand, such as Tata Steel (which bought Corus Group in 2007), Baosteel Group and Shagang Group. As of 2017, though, ArcelorMittal is the world's largest steel producer.

In 2021, it was estimated that around 7% of the global greenhouse gas emissions resulted from the steel industry. Reduction of these emissions are expected to come from a shift in the main production route using cokes, more recycling of steel and the application of carbon capture and storage or carbon capture and utilization technology.

3.4 Steel Industry in South Korea:

In South Korea, production industry has begun to evolve at an elevated pace since the 1990s and has become the primary pushing power with the fast economic development continuing in South Korea. In the 1990s, primary power usage grew at an annualized pace of 7.5%, which in the same era was greater than the annualized financial development level of 6.5%. This is due to strong development in energy-intensive sectors, including petrochemical sectors. The strong increase in industrial electricity consumption helped to boost the reduction of energy conversion, which further subverted energy intensity. The increase in energy production after 2009 significantly buffered the country against the global financial crisis but adversely affected the overall energy efficiency of the country. The energy consumption of the industries is impacted by several unstable variables, like industrial structure, level of technology, cost of energy, financial scale and national policy.

Research and industrial practice draw primary concern in forecasts related to energy resource and planning issues, which resulted due to the increasing issues of oil and coal shortages. Making fair use of by-product gasses in the steel industry demands scheduling operators to be aware of the quantity of real-time generation, usage and storage. The accurate prediction of these units of energy flow provides a useful guide for their planning and distribution.

The iron and steel industries are always energy-intensive, covering 10% of the full industry's energy consumption. Recently, with the rising energy resource shortage, the energy supply condition in the iron and steel industries has become highly challenging. Establishing an energy-saving plan is a common task that can be achieved in areas such as technological development, refurbishing of equipment and improving management. When oil prices increase, the cost of consuming energy is 10-20 times higher than that of the total production of the iron and steel industries. High energy consumption results in higher prices for the iron and steel products which lead to increased pollution and emissions.

To this end, several steps, such as improving the production framework and accelerating the improvement and advancement of power saving and discharge reducing techniques, are required to ensure efficient energy supply in the manufacturing industry in South Korea**.**

3.5 Uses:

Iron and steel are used widely in the construction of roads, railways, other infrastructure, appliances, and buildings. Most large modern structures, such as stadiums and skyscrapers, bridges, and airports, are supported by a steel skeleton. Even those with a concrete structure employ steel for reinforcing. It sees widespread use in major appliances and cars. Despite the growth in usage of aluminium, steel is still the main material for car bodies. Steel is used in a variety of other construction materials, such as bolts, nails and screws, and other household products and cooking utensils.

3.6 Applications of steel:

common applications include

- shipbuilding
- pipelines
- mining
- offshore construction
- aerospace
- white goods

4. Literature Review

4.1 Literature review- 1

K. Karthick, R. Dharmaprakash, 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 (RMSE) and Mean Absolute Percentage Error (MAPE) metrics, demonstrating its promise for accurate energy consumption prediction on both the training and test sets. The proposed model accurately predicts energy consumption for different load types, achieving impressive results on both the training set (RMSE=0.382, R 2=0.999, MAPE=1.139) and the test set (RMSE=1.073, R2=0.998, MAPE=1.142). These findings highlight the potential of CatBoost as a valuable tool for energy management and conservation, enabling organizations to make informed decisions, optimize resource allocation, and promote sustainability.

4.2 Literature review- 2

Sathishkumar V E:

Predictions of Energy Consumption for Industries gain an important place in energy management and control system, as there are dynamic and seasonal changes in the demand and supply of energy. This paper presents and discusses the predictive models for energy consumption of the steel industry. Data used includes lagging and leading current reactive power, lagging and leading current power factor, carbon dioxide (tCO2) emission and load type.

- 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

4.3 Literature review- 3

Myeongbae Lee & Yubin Kim:

This paper explores the potential for predicting energy consumption by data mining approaches. This study leads to the conclusion that RF is best in predicting the energy and GBM performance also equal to RF. So, RF and GBM are more suitable for predicting steel industry energy consumption prediction. A accurate long-term forecast of energy usage is one among the most critical problems for energy management and optimization in the steel industry. In the exploratory analysis the data analysis reveals thought-provoking results. This work aims to establish the best performing prediction algorithm to predict the hourly consumption of energy in the steel industry. The findings indicate that the RF model improves RMSE, MAE, and MAPE of predictions in consideration to other regression models considered in this research.

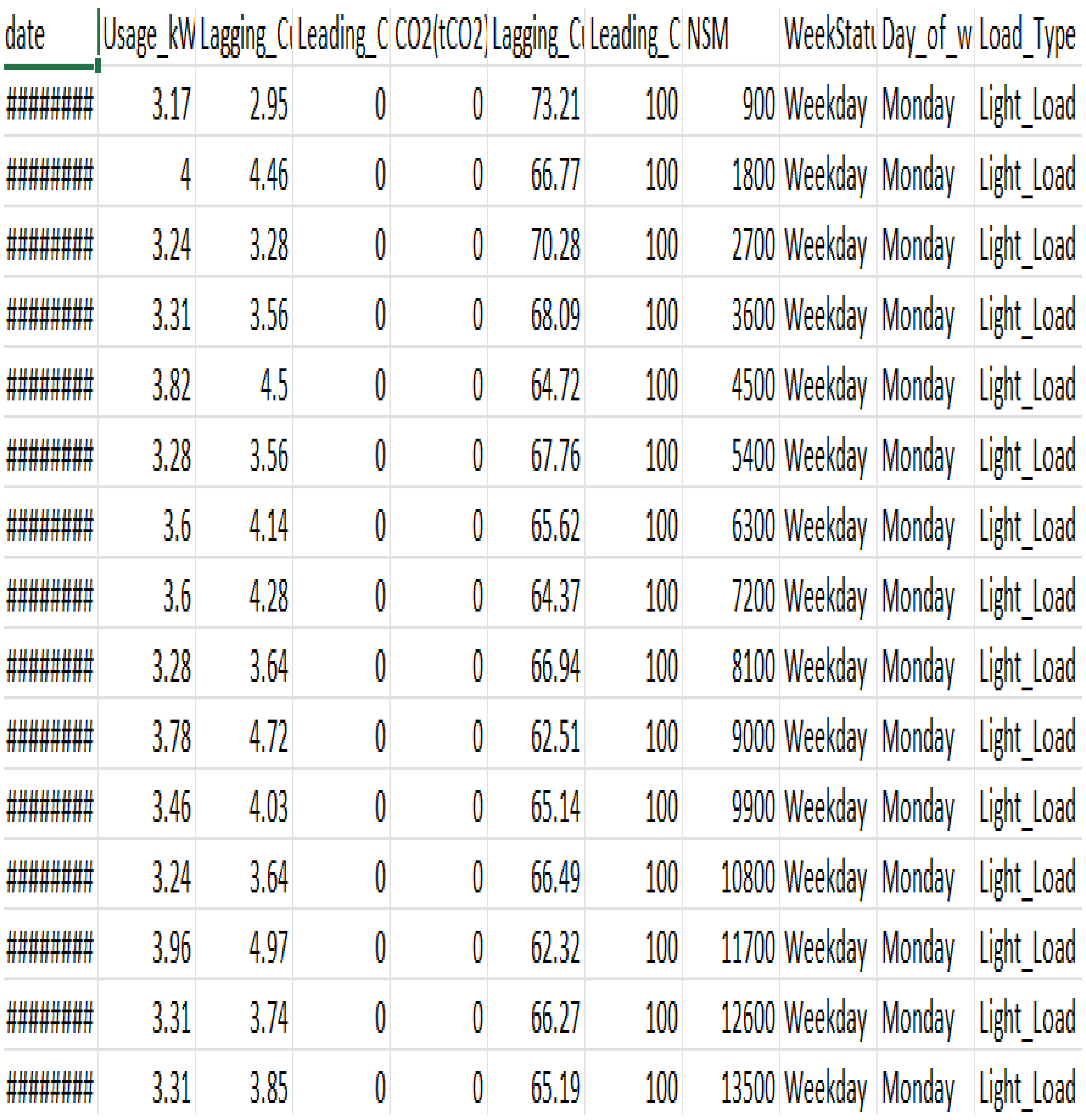
5. Dataset

5.1 Source of the data: The steel industry data is secondary dataset.

We collected this dataset of steel industry data.csv from UC Irvine.

5.2 Sample data:

This is the sample dataset of size 15



date	Usage_kW	Lagging_C	Leading_C	CO2(tCO2	Lagging_C	Leading_C	NSM	WeekStatu	Day_of_w	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
########	3.46	4.03	0	0	65.14	100	9900	Weekday	Monday	Light_Load
########	3.24	3.64	0	0	66.49	100	10800	Weekday	Monday	Light_Load
########	3.96	4.97	0	0	62.32	100	11700	Weekday	Monday	Light_Load
########	3.31	3.74	0	0	66.27	100	12600	Weekday	Monday	Light_Load
########	3.31	3.85	0	0	65.19	100	13500	Weekday	Monday	Light_Load

5.3 Data Description:

This dataset contains current usage, carbon-di-oxide emission and load types. It contains 11 variables namely

- **Usage_kVarh**
- **Lagging_Current_Reactive.Power_kVarh**
- **Leading_Current_Reactive.Power_kVarh**
- **Lagging_Current_Power_Factor_kVarh**
- **Leading_Current_Power_Factor_kVarh**
- **CO2(tCO2)**
- **NSM**
- **WeekStatus**
- **LoadType**
- **Day_of_week**

Variables description:

Lagging current reactive power occurs when the current waveform in an AC circuit lags behind the voltage waveform in phase.

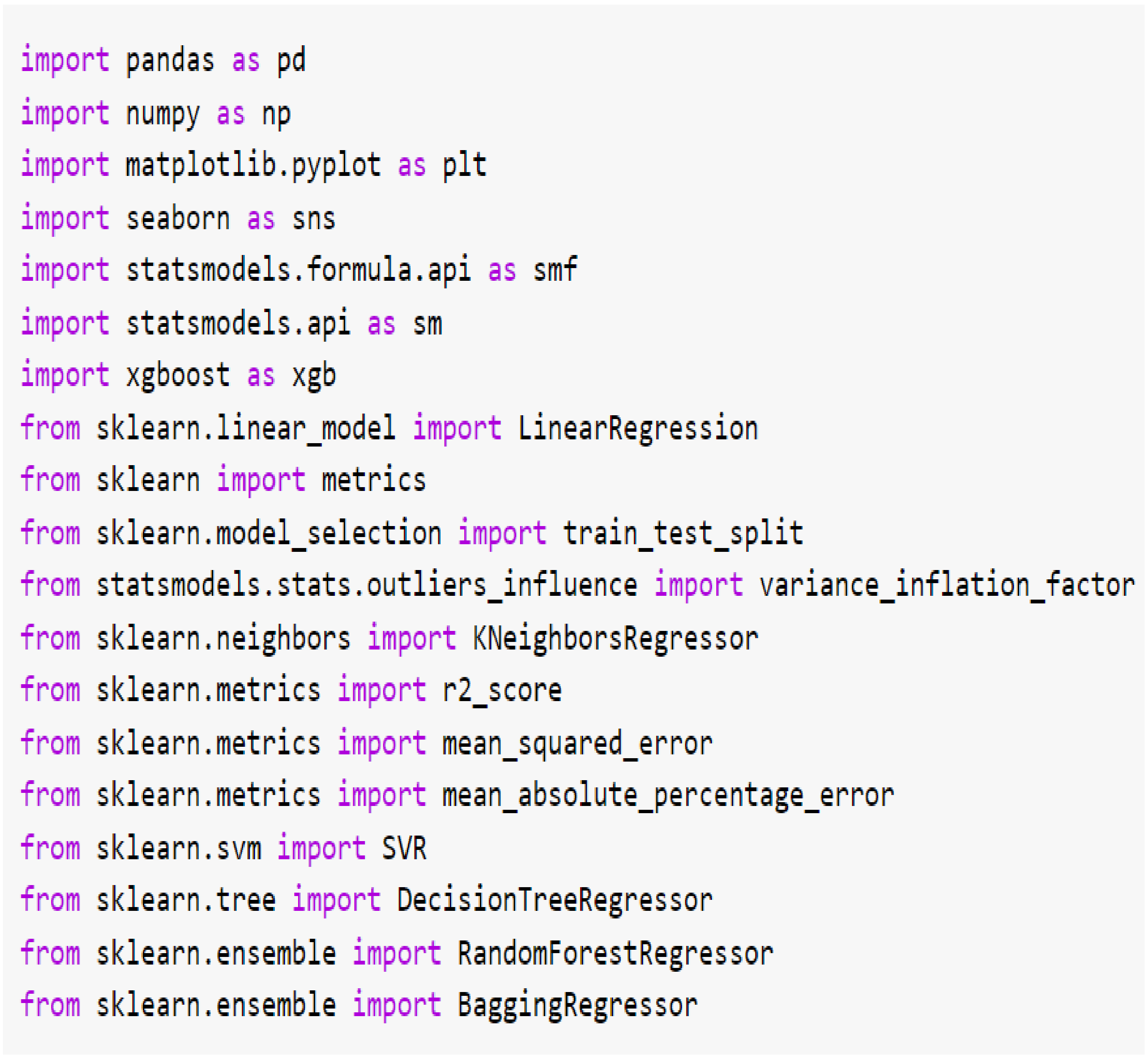
Leading reactive power happens when the current waveform in an electrical system leads the voltage waveform in phase

A lagging power factor occurs in a circuit when the current lags behind the voltage

A leading current power factor occurs when the current in a circuit leads the voltage.

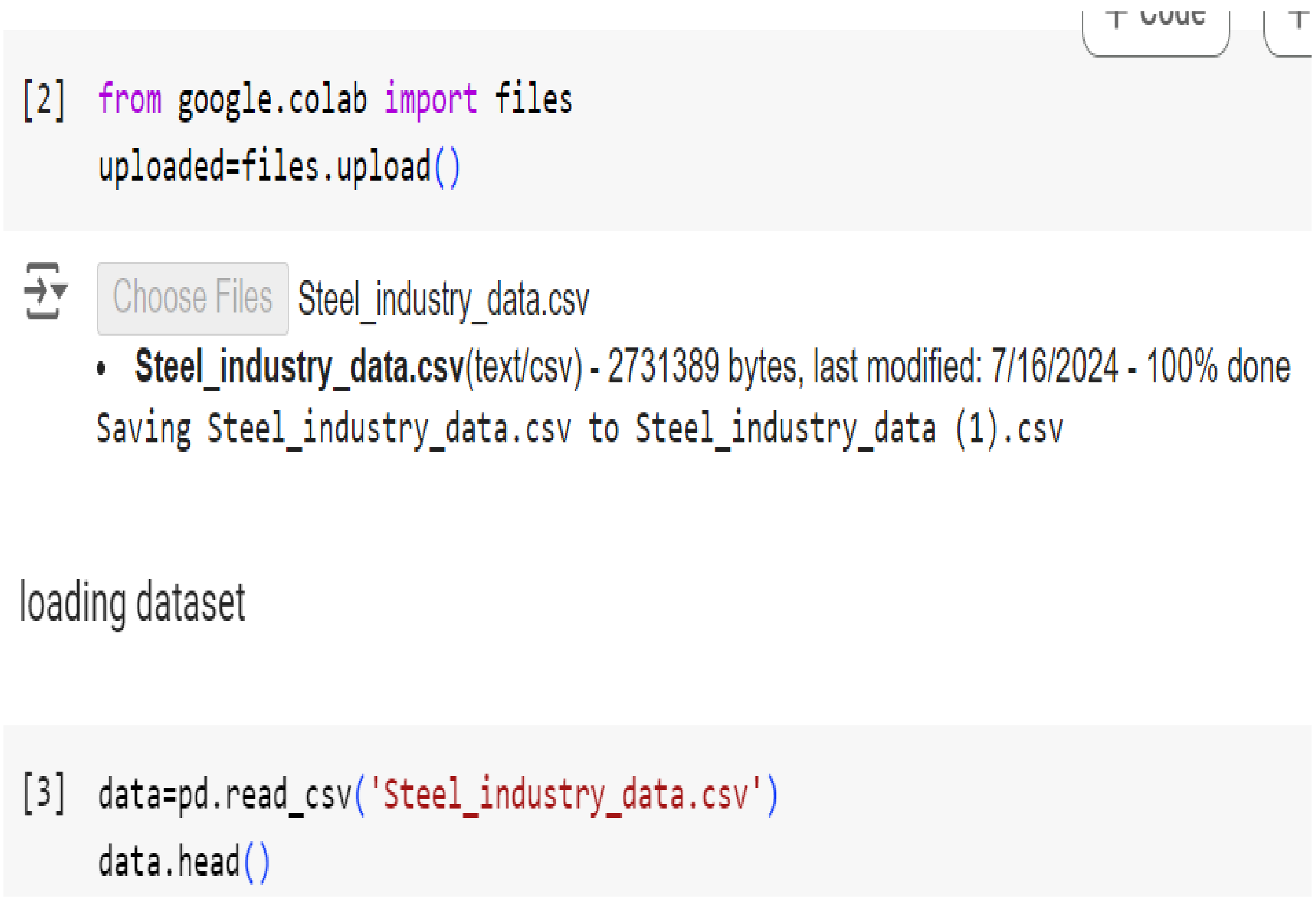
6. Dataset loading

Importing the libraries:



```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.formula.api as smf
import statsmodels.api as sm
import xgboost as xgb
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.model_selection import train_test_split
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_percentage_error
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import BaggingRegressor
```

Loading the Data Set:



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

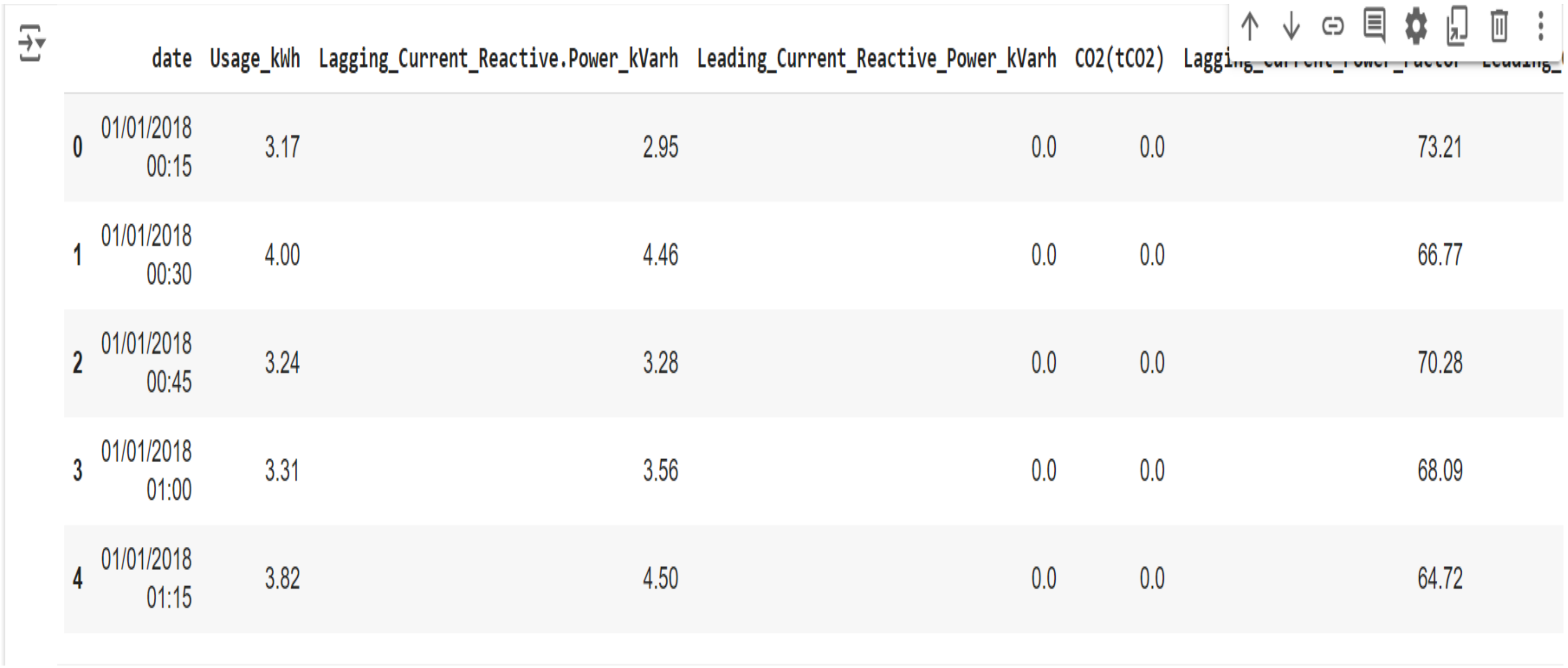
Choose Files Steel_industry_data.csv

- **Steel_industry_data.csv**(text/csv) - 2731389 bytes, last modified: 7/16/2024 - 100% done

```
Saving Steel_industry_data.csv to Steel_industry_data (1).csv
```

loading dataset

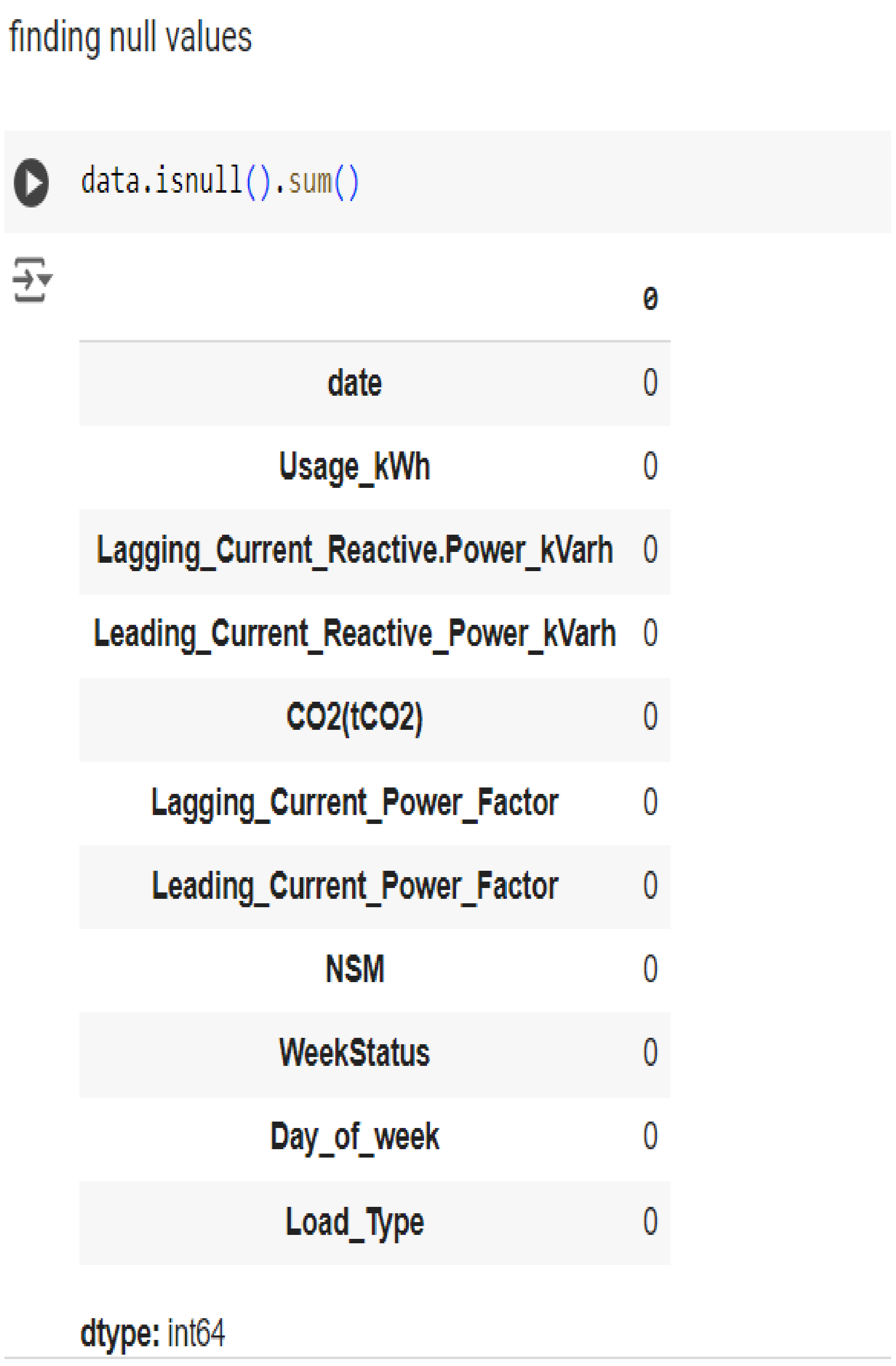
```
[3] data=pd.read_csv('Steel_industry_data.csv')
    data.head()
```



	date	Usage_kWh	Lagging_Current_Reactive.Power_kVarh	Leading_Current_Reactive_Power_kVarh	CO2(tCO2)	Lagging_Current_Power_Factor	Leading_
0	01/01/2018 00:15	3.17	2.95	0.0	0.0	73.21	
1	01/01/2018 00:30	4.00	4.46	0.0	0.0	66.77	
2	01/01/2018 00:45	3.24	3.28	0.0	0.0	70.28	
3	01/01/2018 01:00	3.31	3.56	0.0	0.0	68.09	
4	01/01/2018 01:15	3.82	4.50	0.0	0.0	64.72	

7. Data cleaning

7.1 Null Values:

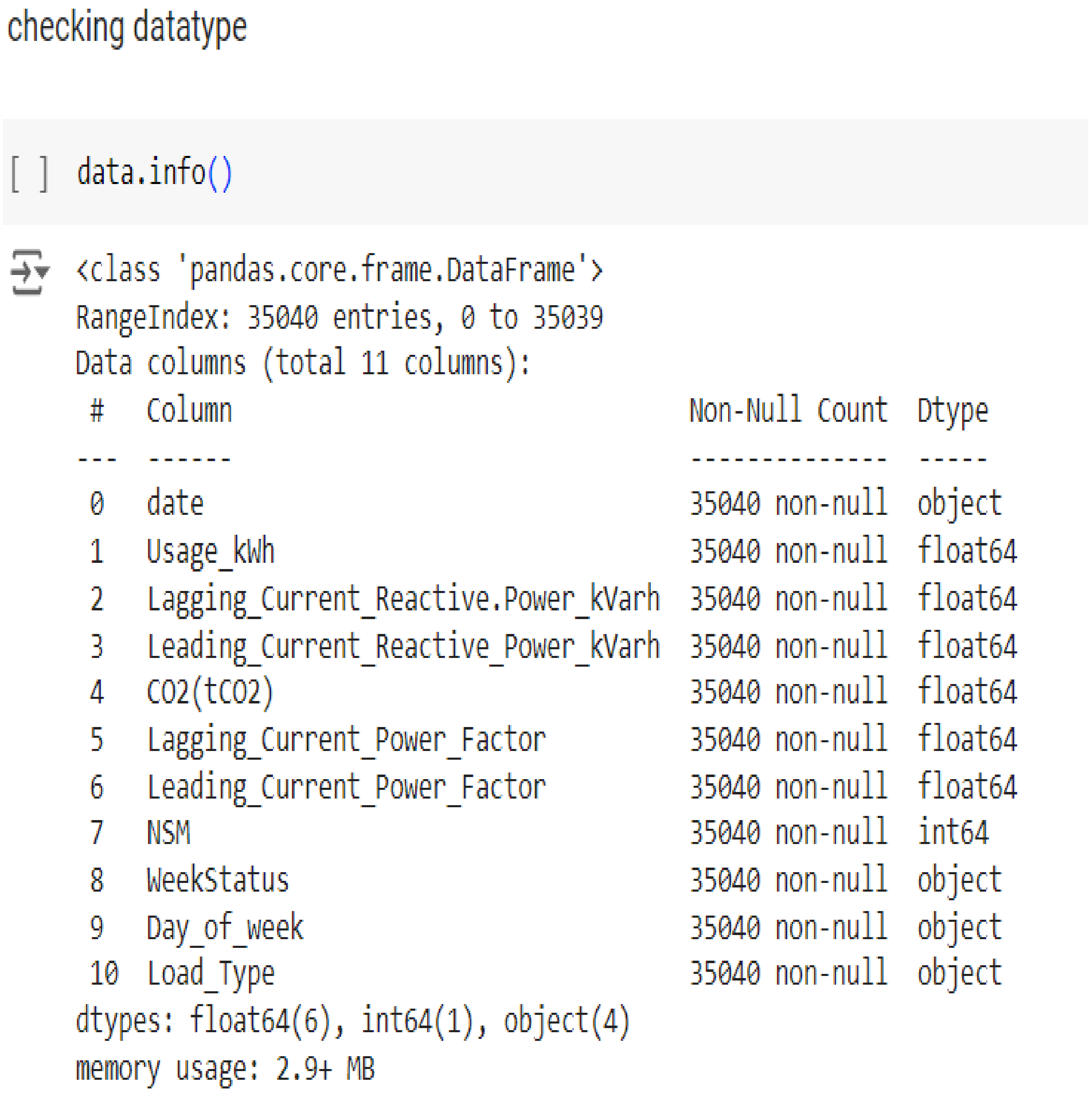


finding null values

```python
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
```

7.2 Checking for the data type:



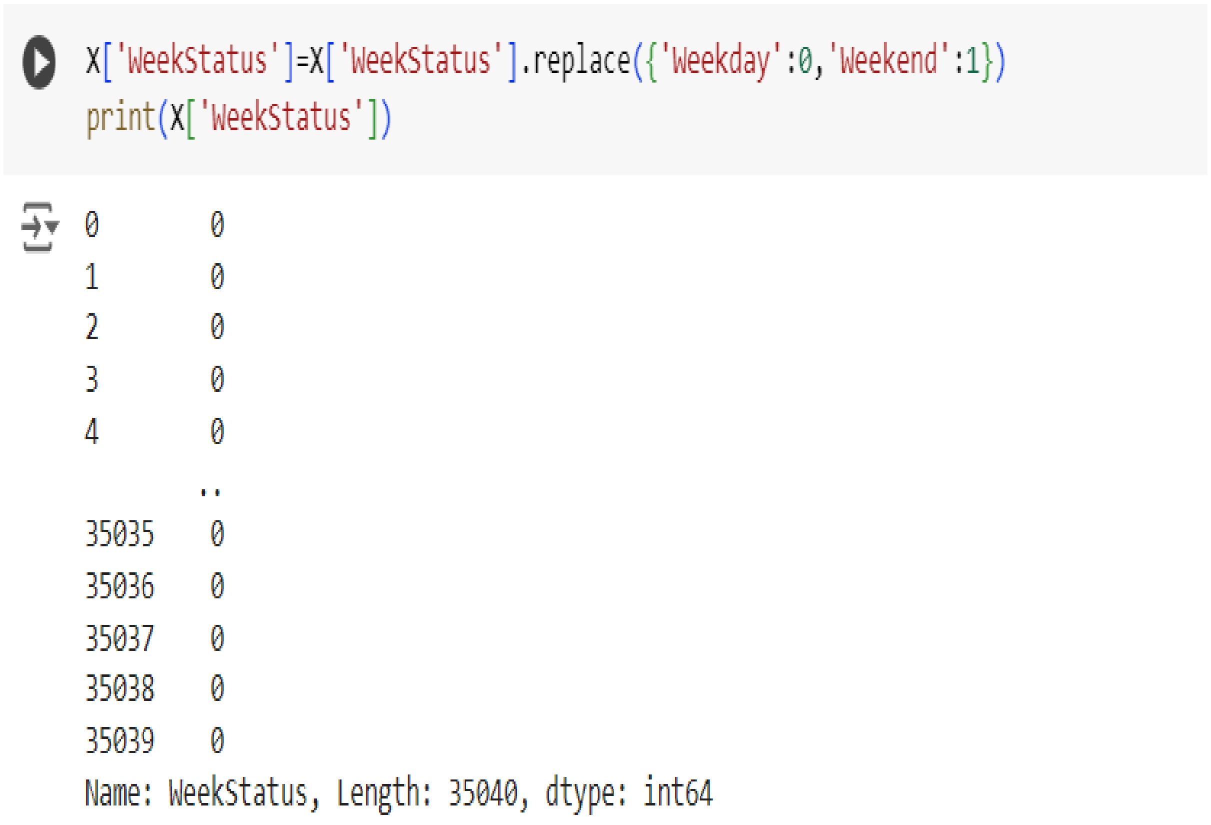
checking datatype

```
[ ] 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
```

7.3 Converting the categorical variables into continuous variables:

After removing date column, we enlabled the factors in the variable WeekStatus



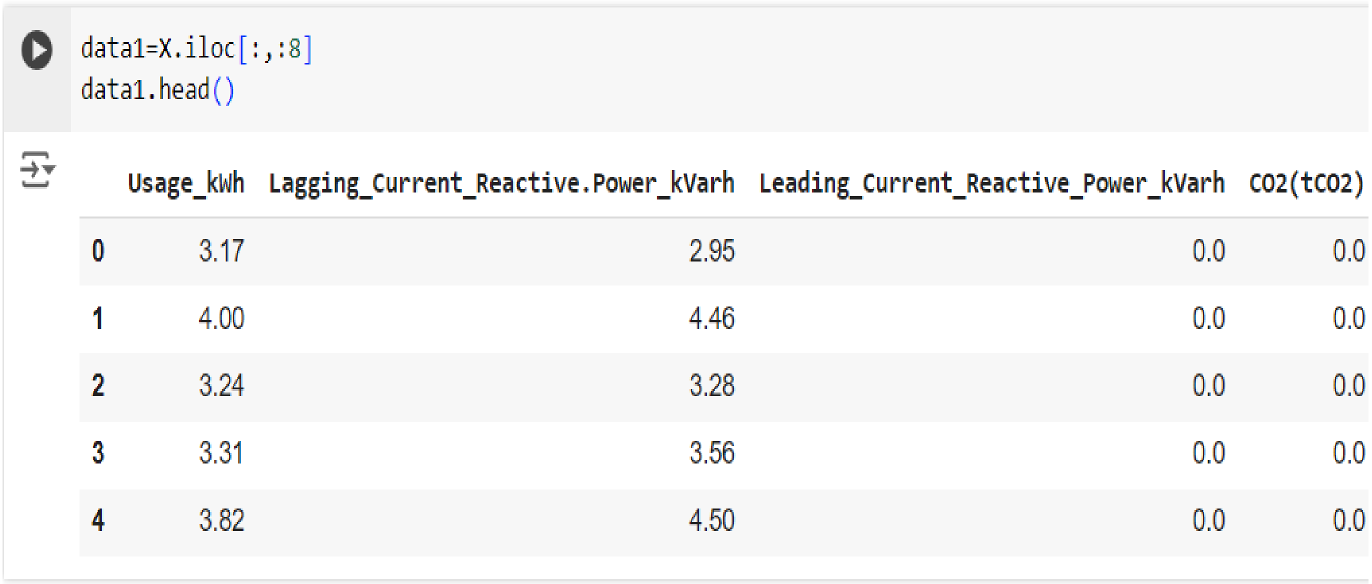
```python
X['WeekStatus']=X['WeekStatus'].replace({'Weekday':0,'Weekend':1})
print(X['WeekStatus'])
```

```
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
```

Dummy Variable encoding:

For dummy variable encoding we divided the dataset into 2 different datasets one with categorical after enlabling the variable WeekStatus and second data set with continuous variables.

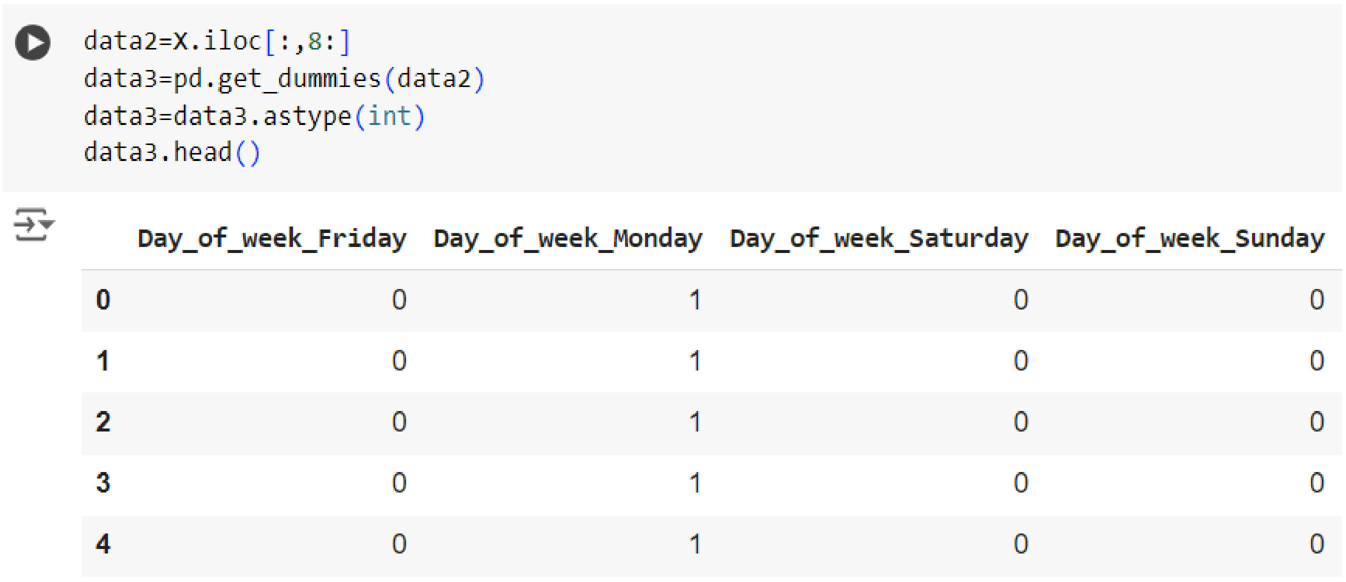
Data-1



```
data1=X.iloc[:,:8]
data1.head()
```

	Usage_kWh	Lagging_Current_Reactive.Power_kVarh	Leading_Current_Reactive_Power_kVarh	CO2(tCO2)
0	3.17	2.95	0.0	0.0
1	4.00	4.46	0.0	0.0
2	3.24	3.28	0.0	0.0
3	3.31	3.56	0.0	0.0
4	3.82	4.50	0.0	0.0

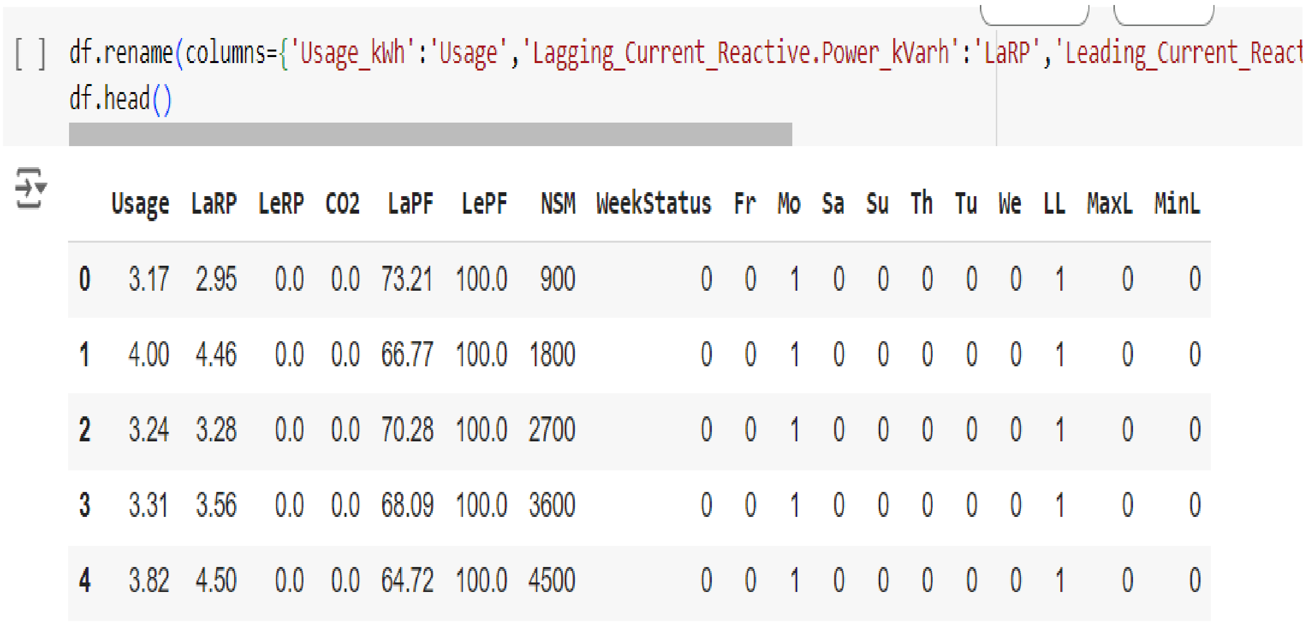
Data-2



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

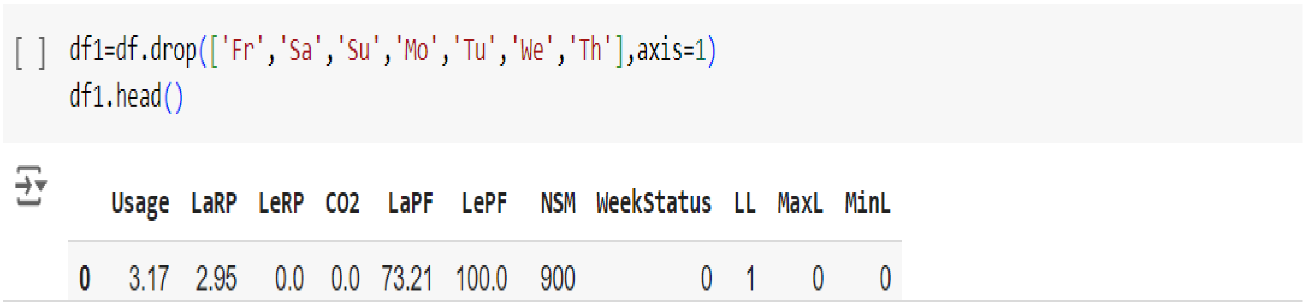
	Day_of_week_Friday	Day_of_week_Monday	Day_of_week_Saturday	Day_of_week_Sunday
0	0	1	0	0
1	0	1	0	0
2	0	1	0	0
3	0	1	0	0
4	0	1	0	0

7.4 Renaming the column names and removing day of week columns



```
[ ] df.rename(columns={'Usage_kWh':'Usage','Lagging_Current_Reactive.Power_kVarh':'LaRP','Leading_Current_React
    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
```

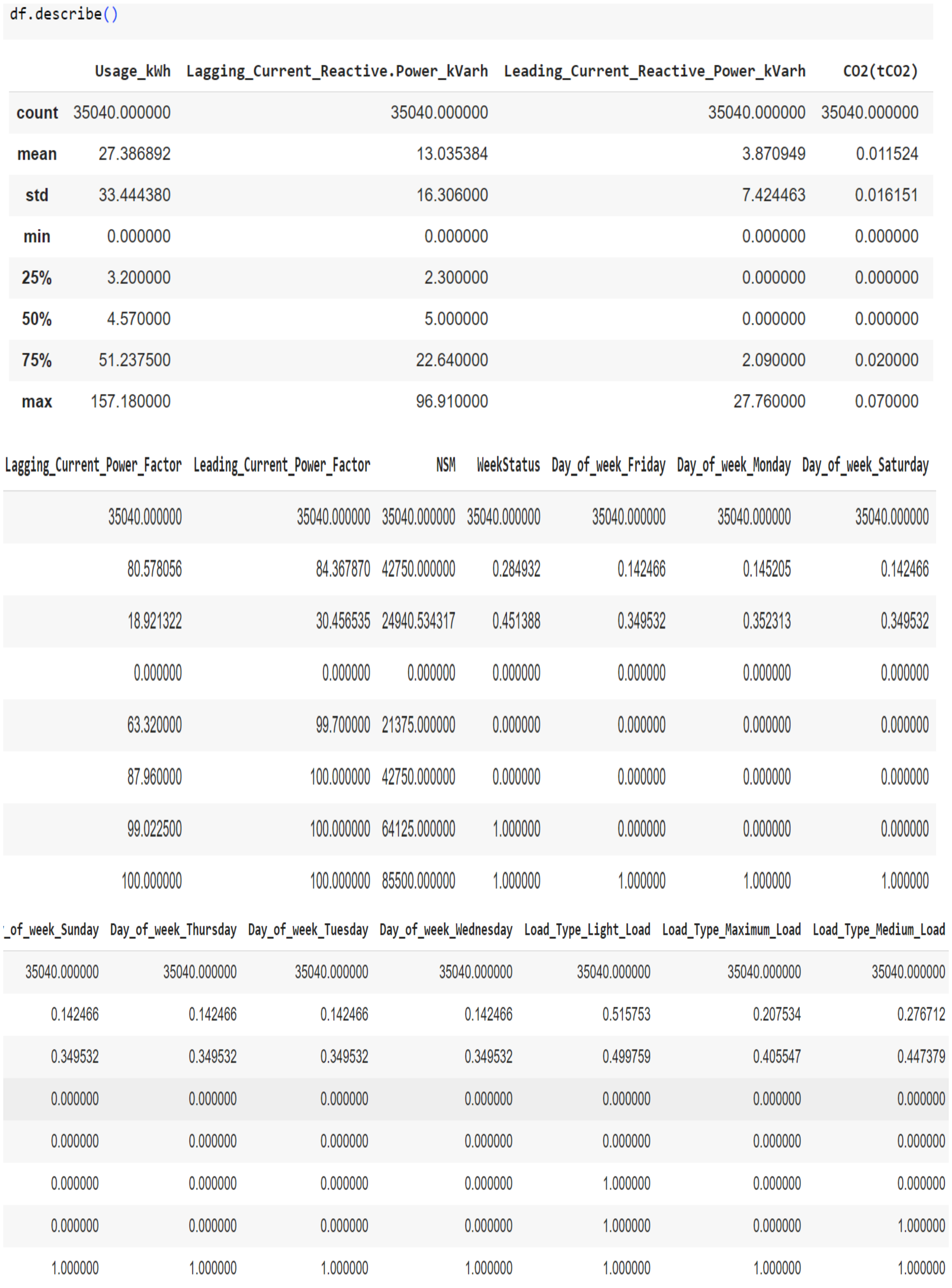


```
[ ] df1=df.drop(['Fr','Sa','Su','Mo','Tu','We','Th'],axis=1)
    df1.head()
```

```
   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
```

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

7.5 Data Analysis



```
df.describe()
```

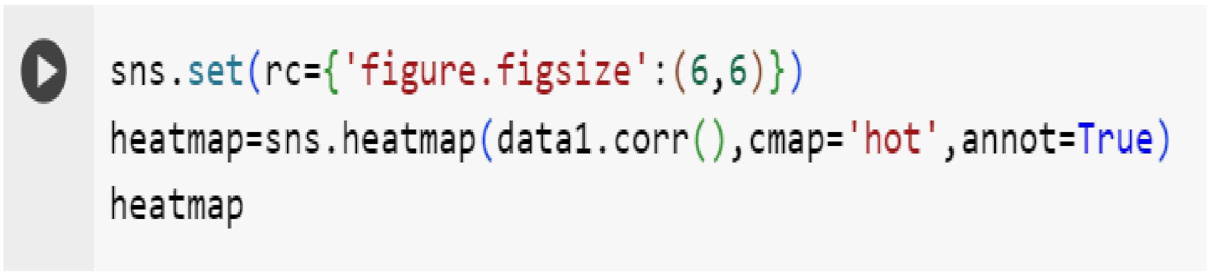
	Usage_kWh	Lagging_Current_Reactive.Power_kVarh	Leading_Current_Reactive_Power_kVarh	CO2(tCO2)
count	35040.000000	35040.000000	35040.000000	35040.000000
mean	27.386892	13.035384	3.870949	0.011524
std	33.444380	16.306000	7.424463	0.016151
min	0.000000	0.000000	0.000000	0.000000
25%	3.200000	2.300000	0.000000	0.000000
50%	4.570000	5.000000	0.000000	0.000000
75%	51.237500	22.640000	2.090000	0.020000
max	157.180000	96.910000	27.760000	0.070000

Lagging_Current_Power_Factor	Leading_Current_Power_Factor	NSM	WeekStatus	Day_of_week_Friday	Day_of_week_Monday	Day_of_week_Saturday
35040.000000	35040.000000	35040.000000	35040.000000	35040.000000	35040.000000	35040.000000
80.578056	84.367870	42750.000000	0.284932	0.142466	0.145205	0.142466
18.921322	30.456535	24940.534317	0.451388	0.349532	0.352313	0.349532
0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
63.320000	99.700000	21375.000000	0.000000	0.000000	0.000000	0.000000
87.960000	100.000000	42750.000000	0.000000	0.000000	0.000000	0.000000
99.022500	100.000000	64125.000000	1.000000	0.000000	0.000000	0.000000
100.000000	100.000000	85500.000000	1.000000	1.000000	1.000000	1.000000

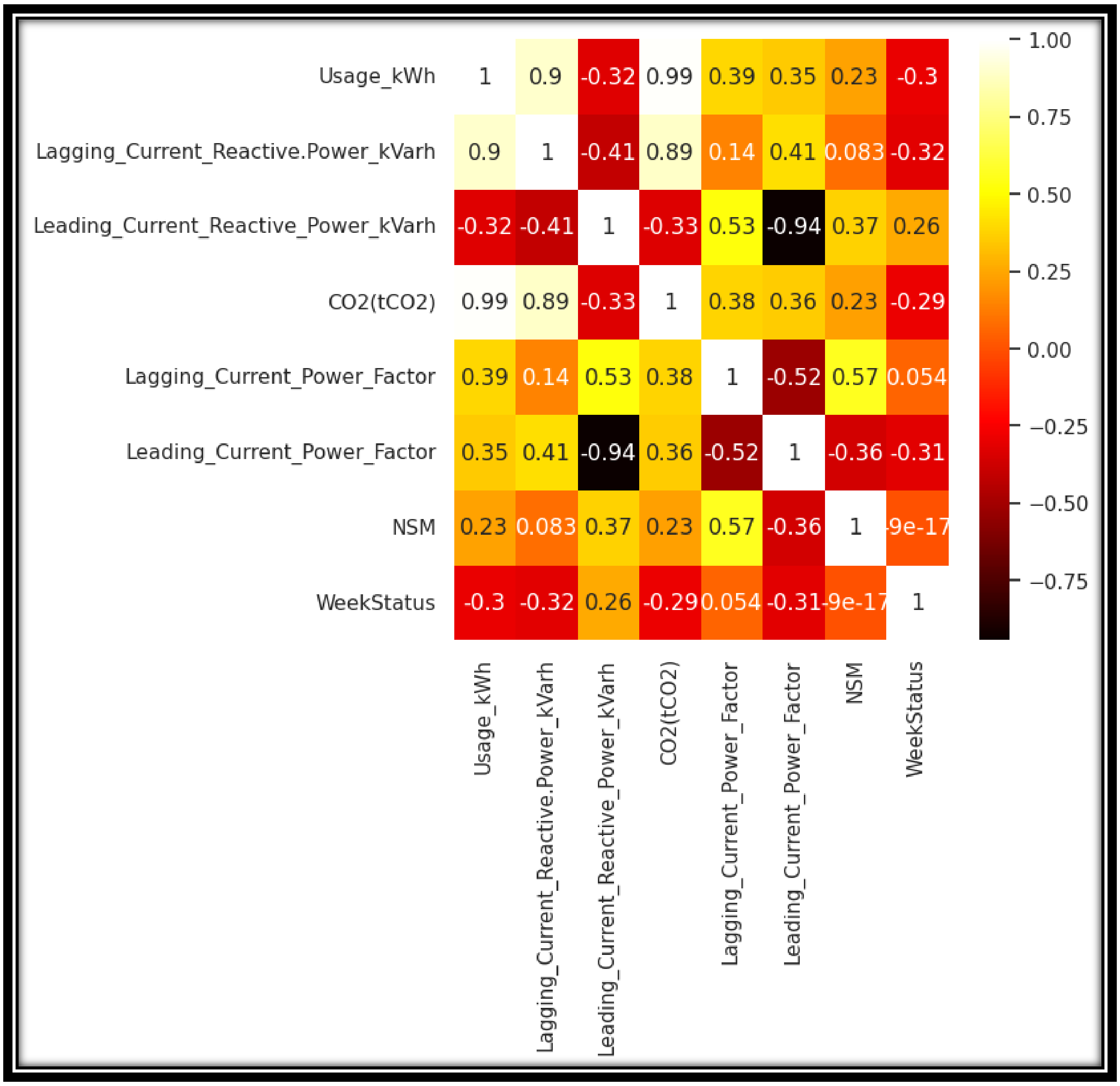
_of_week_Sunday	Day_of_week_Thursday	Day_of_week_Tuesday	Day_of_week_Wednesday	Load_Type_Light_Load	Load_Type_Maximum_Load	Load_Type_Medium_Load
35040.000000	35040.000000	35040.000000	35040.000000	35040.000000	35040.000000	35040.000000
0.142466	0.142466	0.142466	0.142466	0.515753	0.207534	0.276712
0.349532	0.349532	0.349532	0.349532	0.499759	0.405547	0.447379
0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	1.000000
1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

8 Exploratory Data Analysis:

8.1 Heatmap of correlation matrix:



```
sns.set(rc={'figure.figsize':(6,6)})
heatmap=sns.heatmap(data1.corr(),cmap='hot',annot=True)
heatmap
```

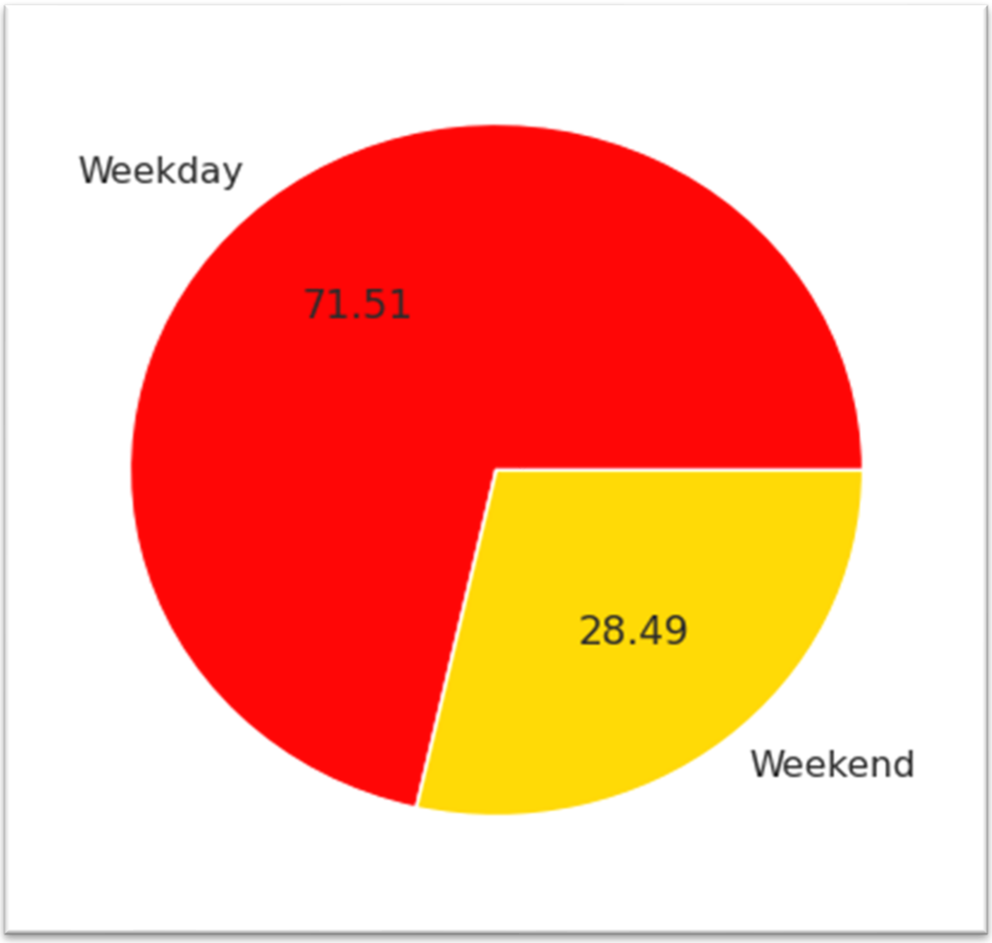


Here, we can observe CO2 and lagging current reactive power are most positively correlated with respect to usage and leading current power factor and reactive power are most negatively correlated.

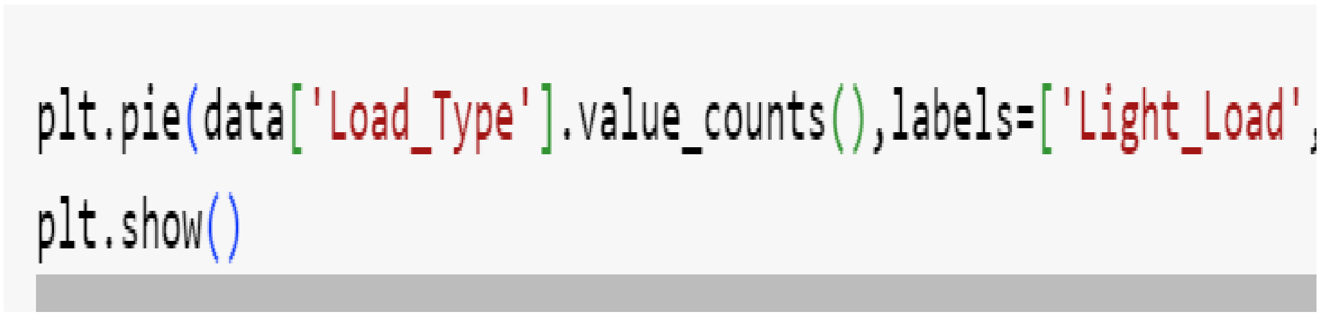
The values nearer to the 1 and -1 are said to be strongly correlated and values nearest to 0 are weakly correlated.

8.2 Pie Charts:

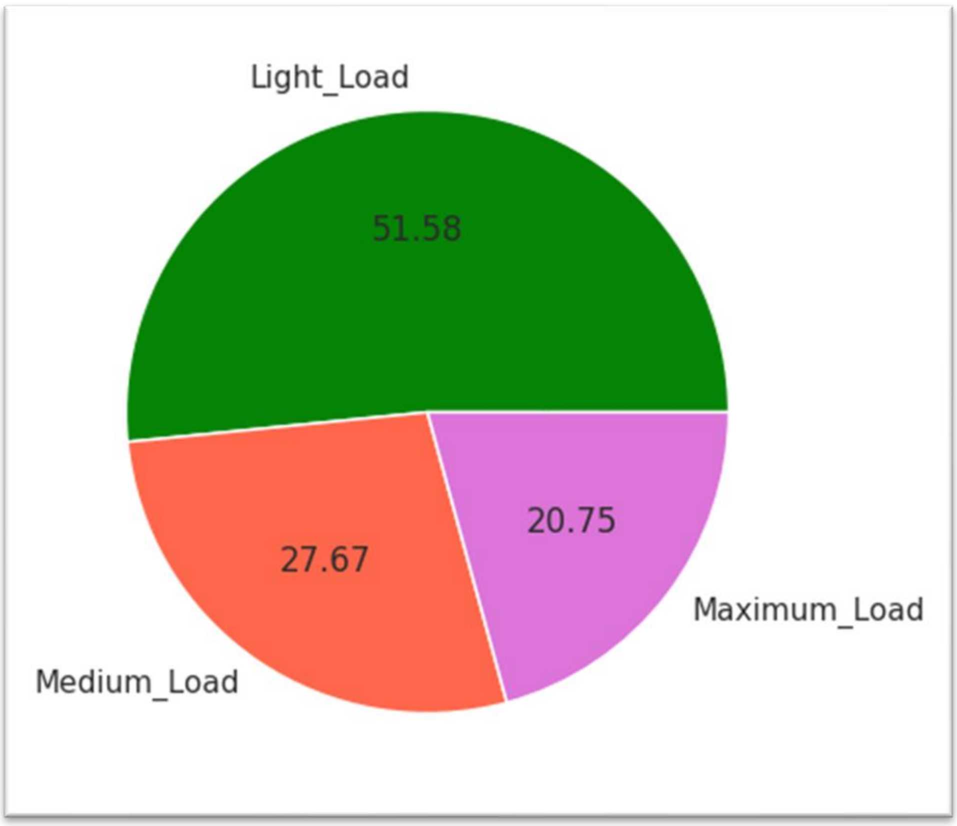
```
sns.set(rc={'figure.figsize':(8,5)})
colors=['red','gold']
plt.pie(df['WeekStatus'].value_counts(),labels=['Weekday','Weekend'],autopct=lambda x:np
plt.show()
```



This plot explains about the percentage of weekdays and weekends in the dataset. 71.51% of data consists of weekdays and 28.49% of data consists of weekends.



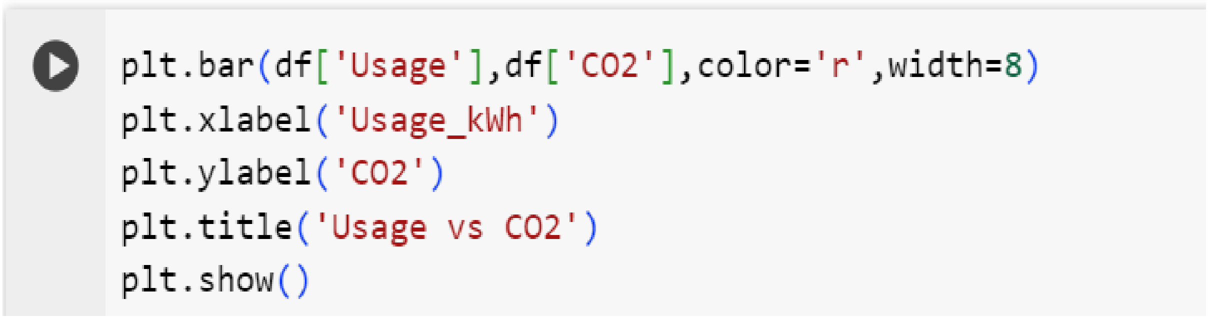
```
plt.pie(data['Load_Type'].value_counts(),labels=['Light_Load',
plt.show()
```



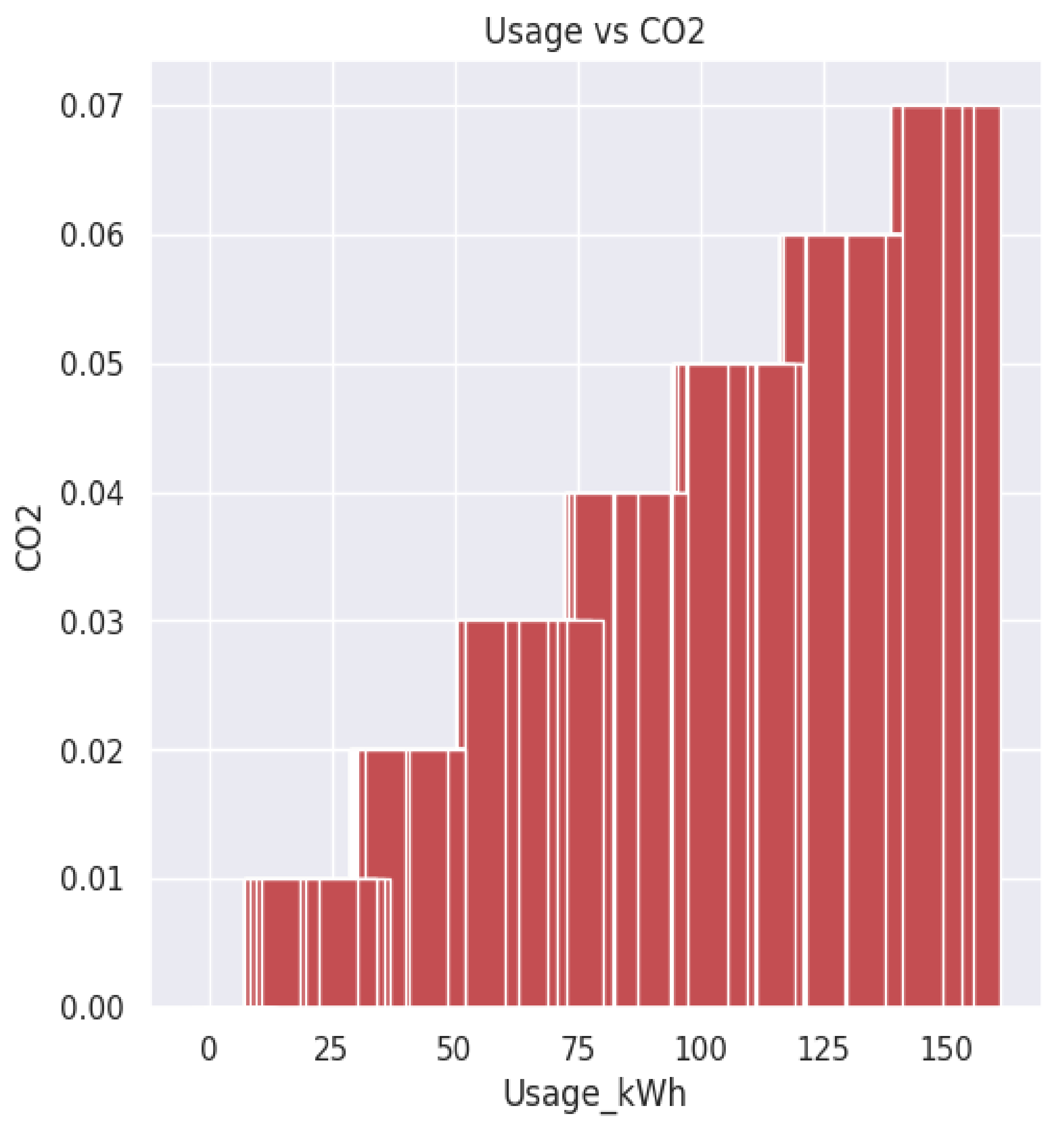
This plot explains about the percentage of load types.

- 51.58% of data consists of light load
- 27.67% of data consists of medium load
- 20.75% of data consists of maximum load

8.3 Bar Plots:



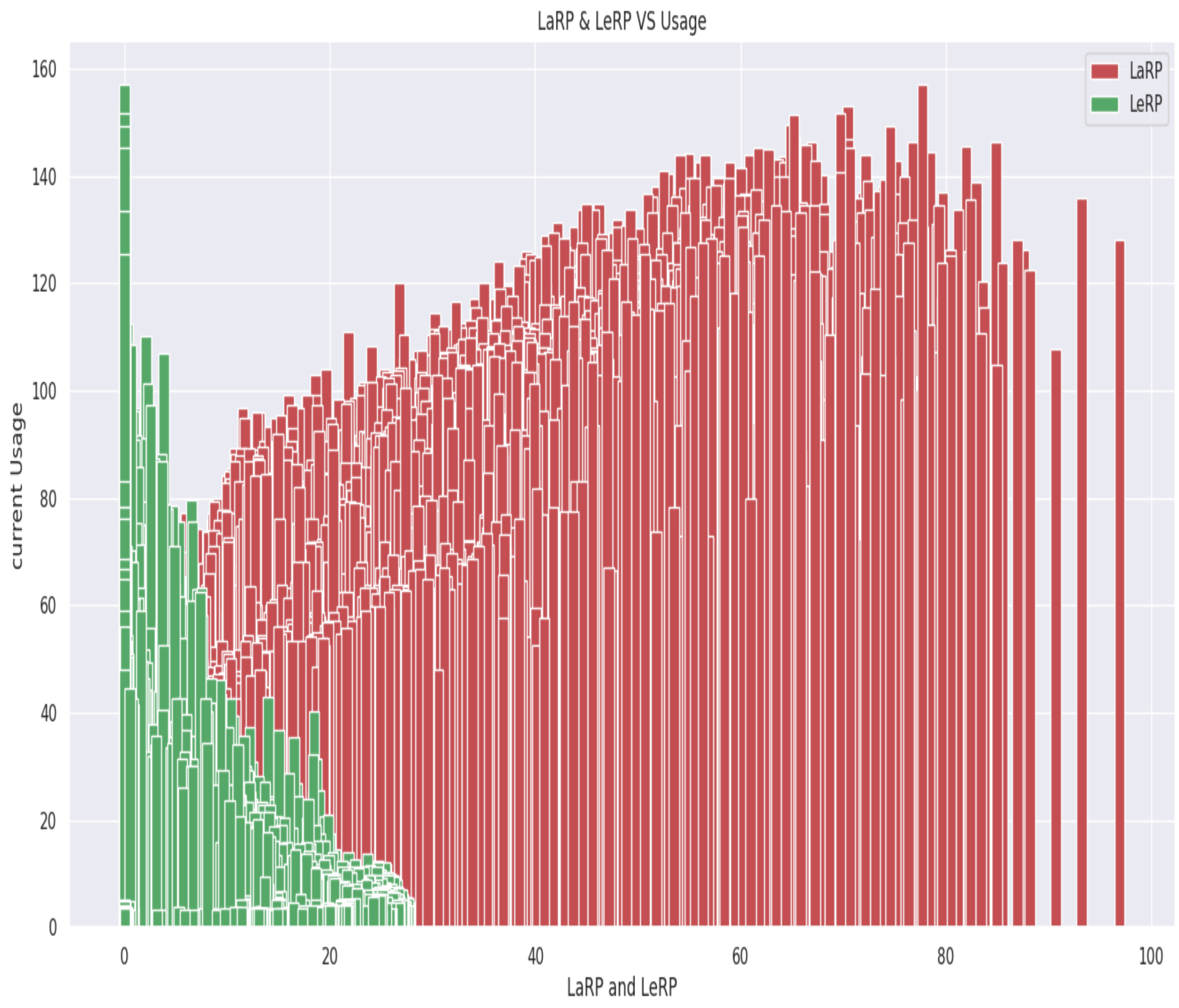
```
plt.bar(df['Usage'],df['CO2'],color='r',width=8)
plt.xlabel('Usage_kWh')
plt.ylabel('CO2')
plt.title('Usage vs CO2')
plt.show()
```



This plot explains, as usage increases CO2 emission also increases.

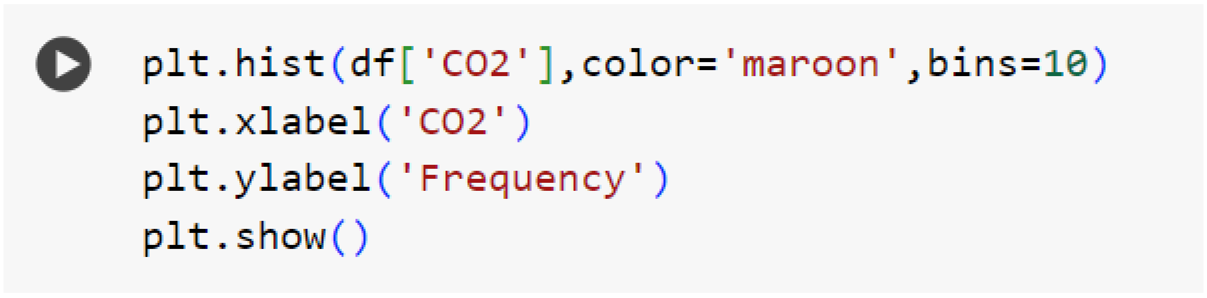
Lagging current reactive power, leading current reactive power vs Usage

```
sns.set(rc={'figure.figsize':(16,8)})
bar1=plt.bar(df['LaRP'],df['Usage'], color='r',width=1)
bar2=plt.bar(df['LeRP'],df['Usage'], color='g',width=1)
plt.xlabel('LaRP and LeRP')
plt.ylabel('current Usage')
plt.title('LaRP & LeRP VS Usage')
plt.legend((bar1,bar2),('LaRP','LeRP'))
plt.show()
```

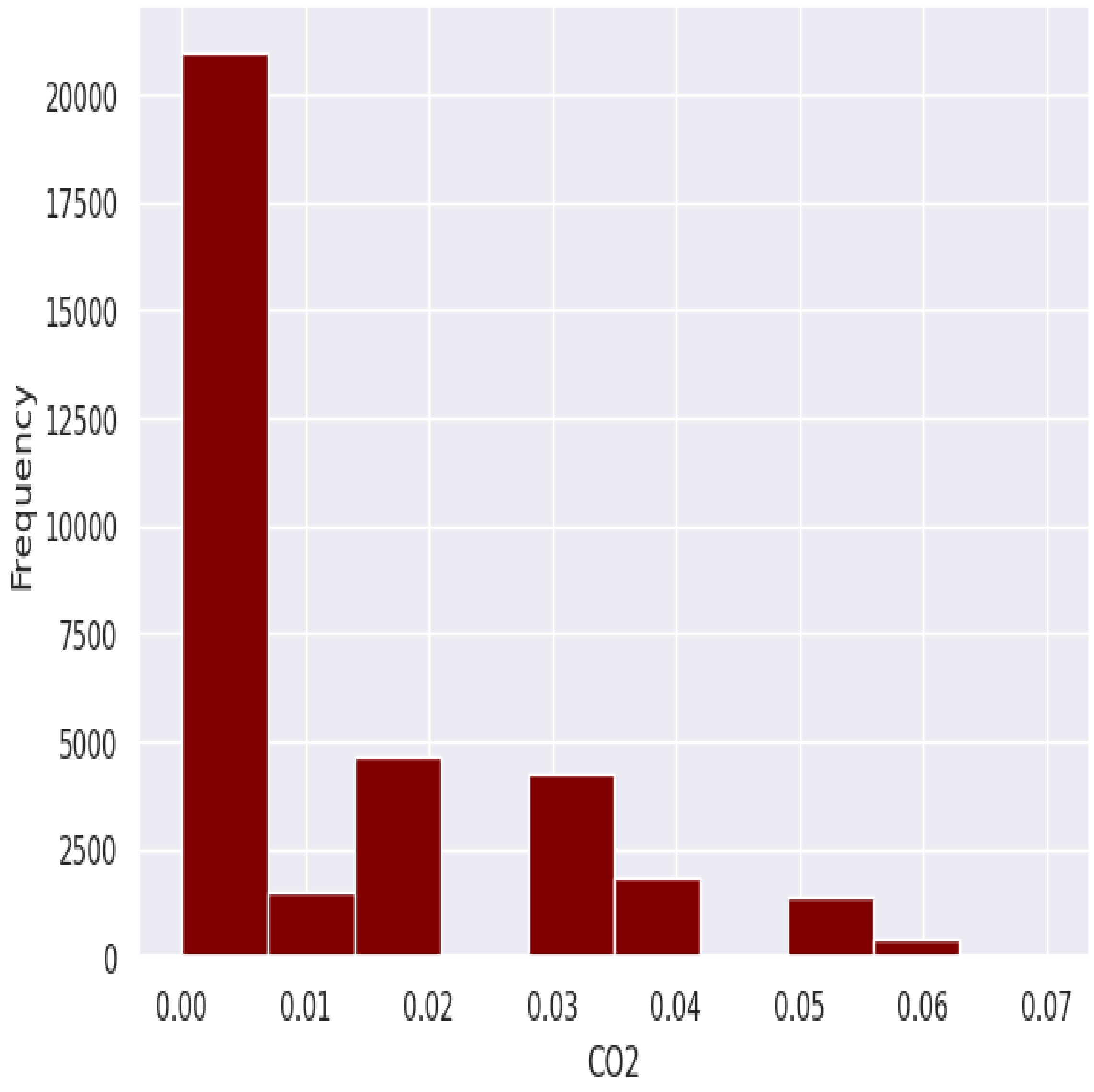


This plot explains as usage increases LeRP decreases and LaRP increases

8.4 Histogram



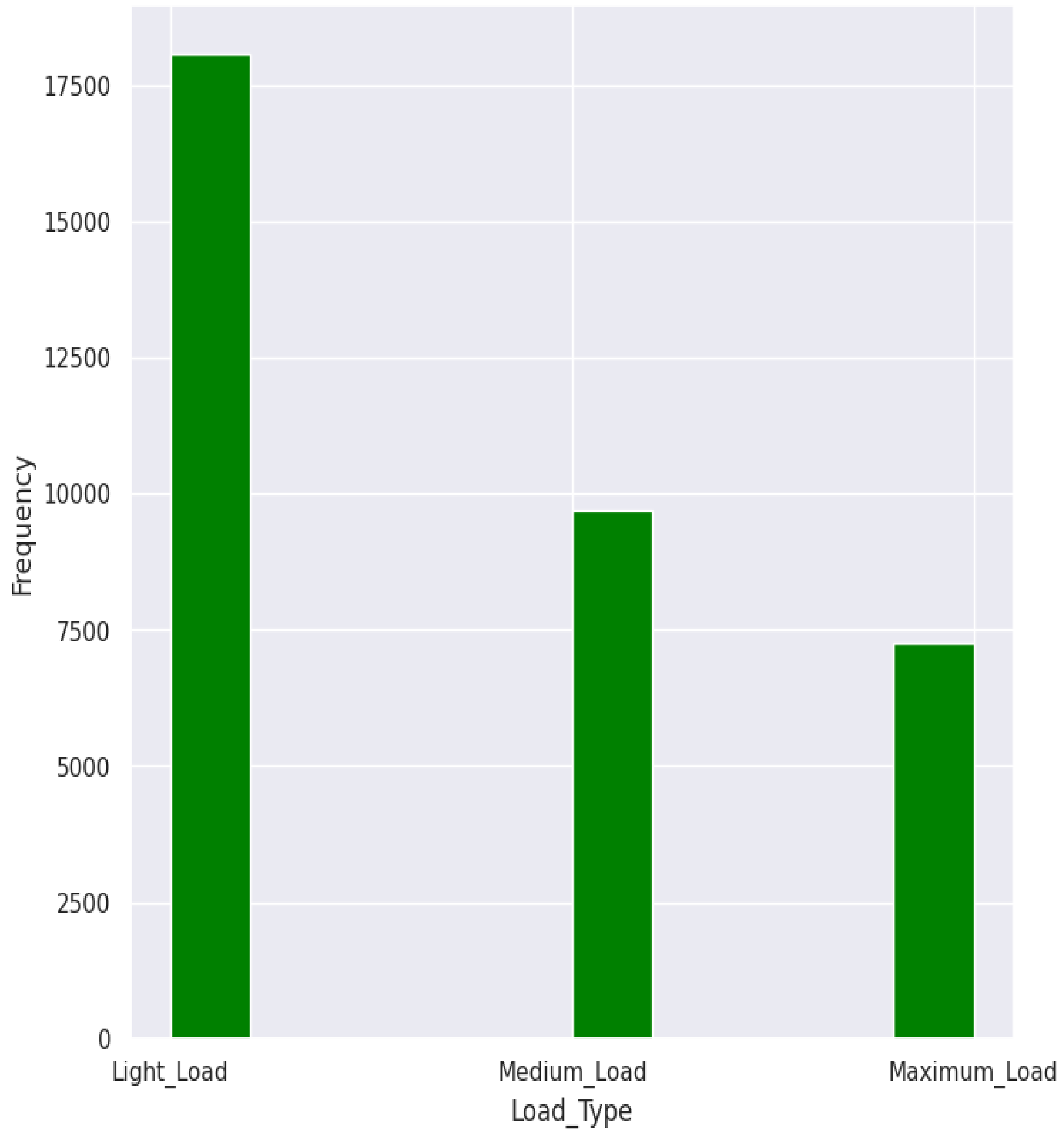
```
plt.hist(df['CO2'],color='maroon',bins=10)
plt.xlabel('CO2')
plt.ylabel('Frequency')
plt.show()
```



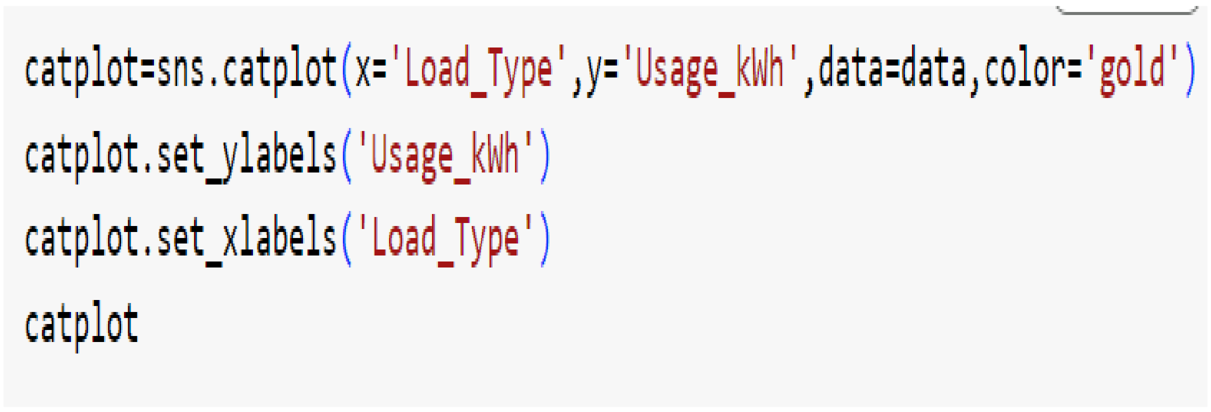
This plot gives the count of CO2 emission rate.

Histogram for loadtype

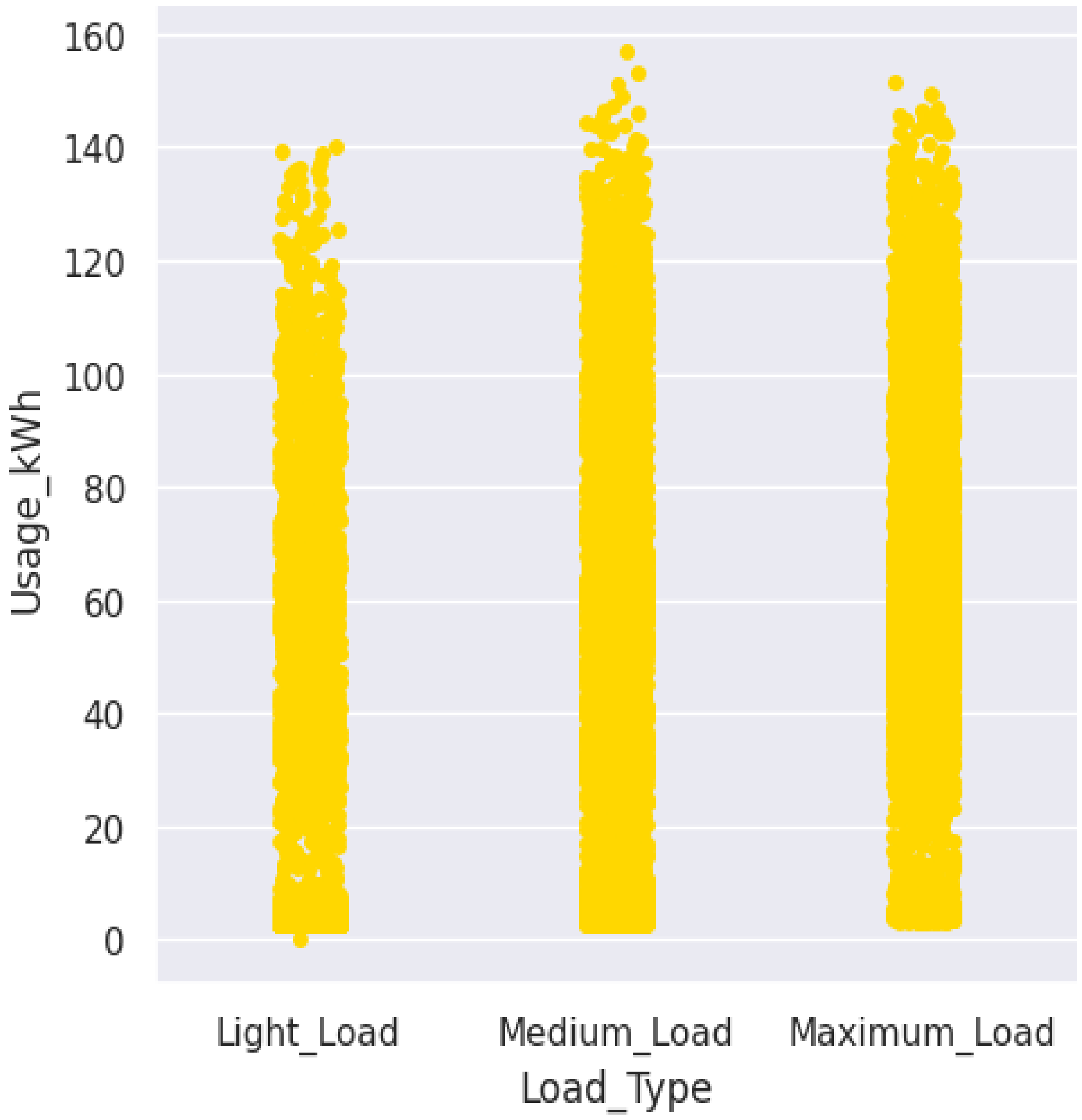
```
sns.set(rc={'figure.figsize':(8,8)})
plt.hist(data['Load_Type'],color='green')
plt.xlabel('Load_Type')
plt.ylabel('Frequency')
plt.show()
```



8.5 Categorical plot

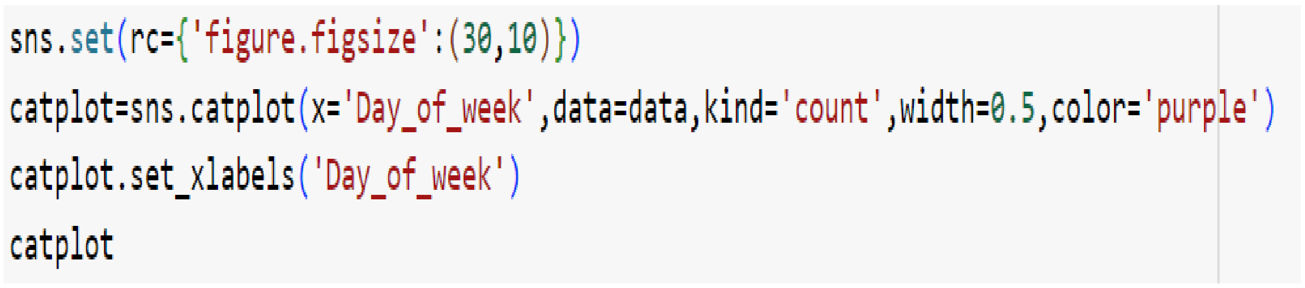


```
catplot=sns.catplot(x='Load_Type',y='Usage_kWh',data=data,color='gold')
catplot.set_ylabels('Usage_kWh')
catplot.set_xlabels('Load_Type')
catplot
```

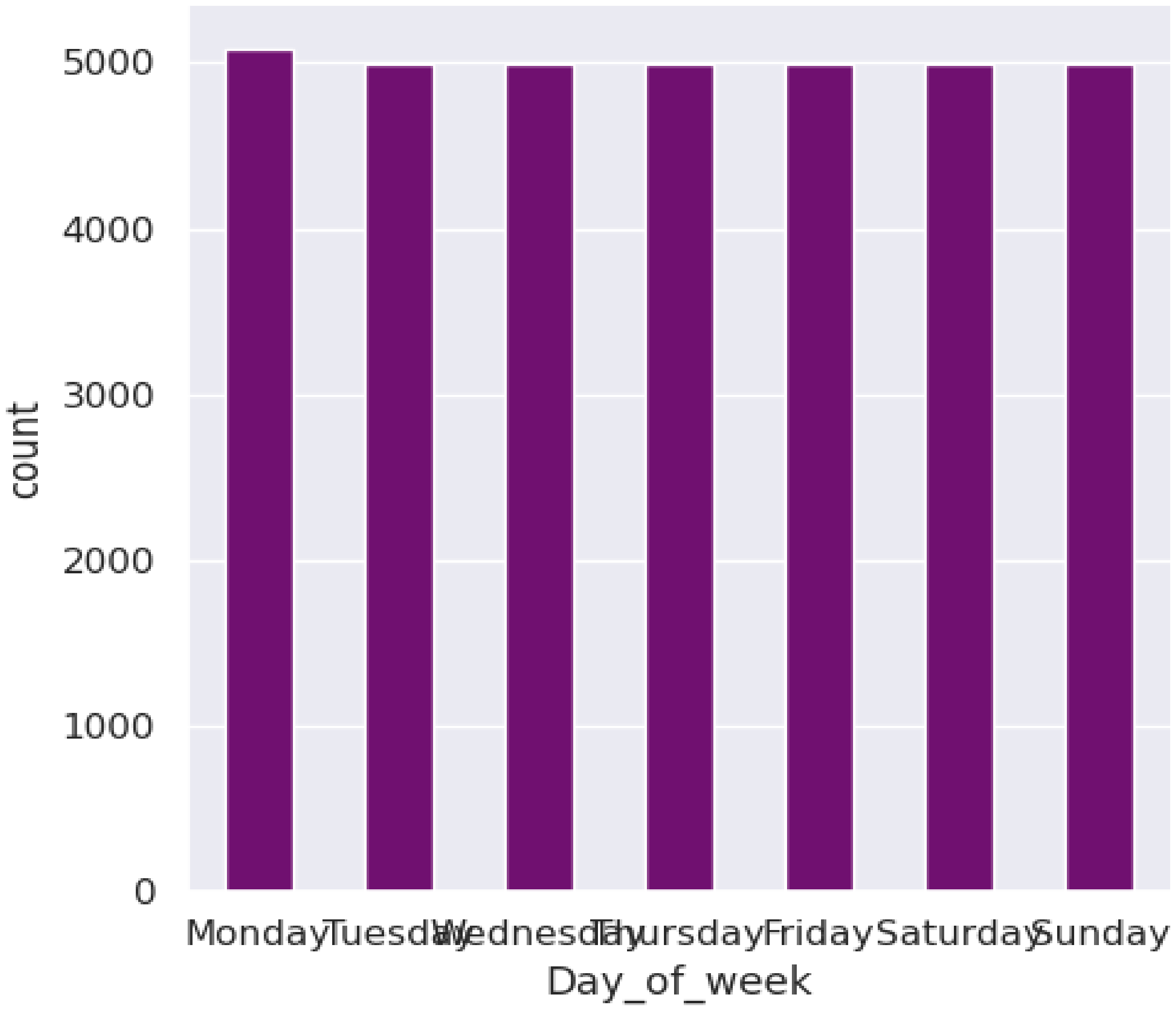


This plot gives the usage of the different load types.

Categorical plot for Day of week



```
sns.set(rc={'figure.figsize':(30,10)})
catplot=sns.catplot(x='Day_of_week',data=data,kind='count',width=0.5,color='purple')
catplot.set_xlabels('Day_of_week')
catplot
```

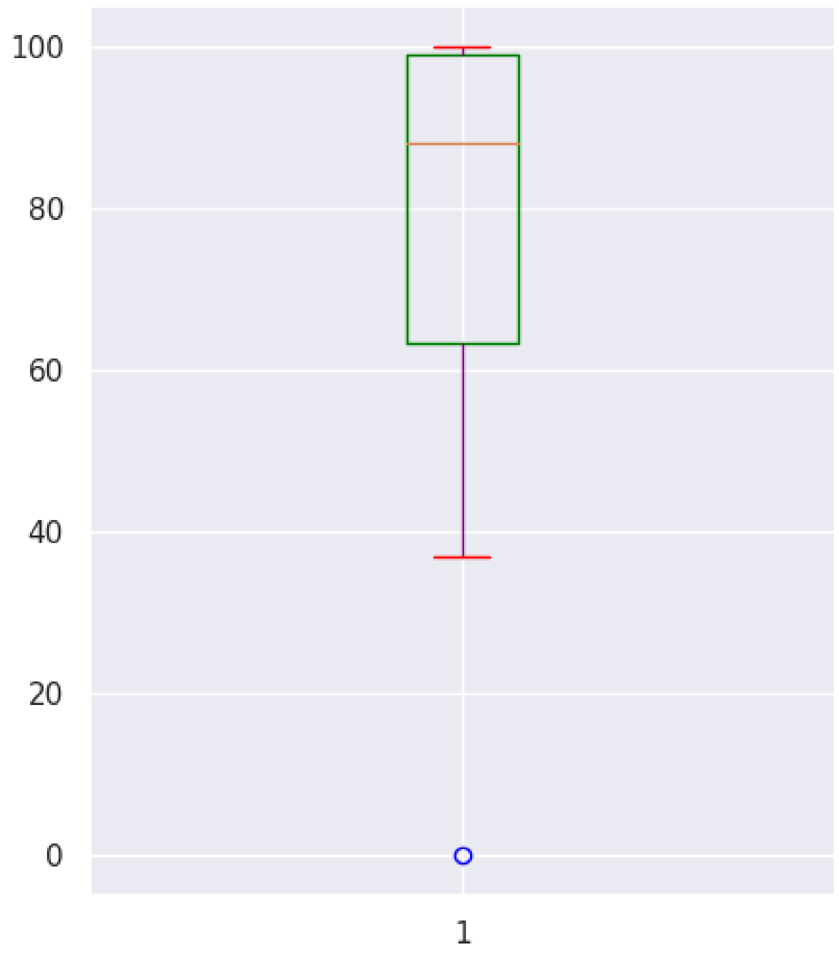


Here, we observe that Monday was repeated more number of times than remaining days.

8.4 Box Plot:

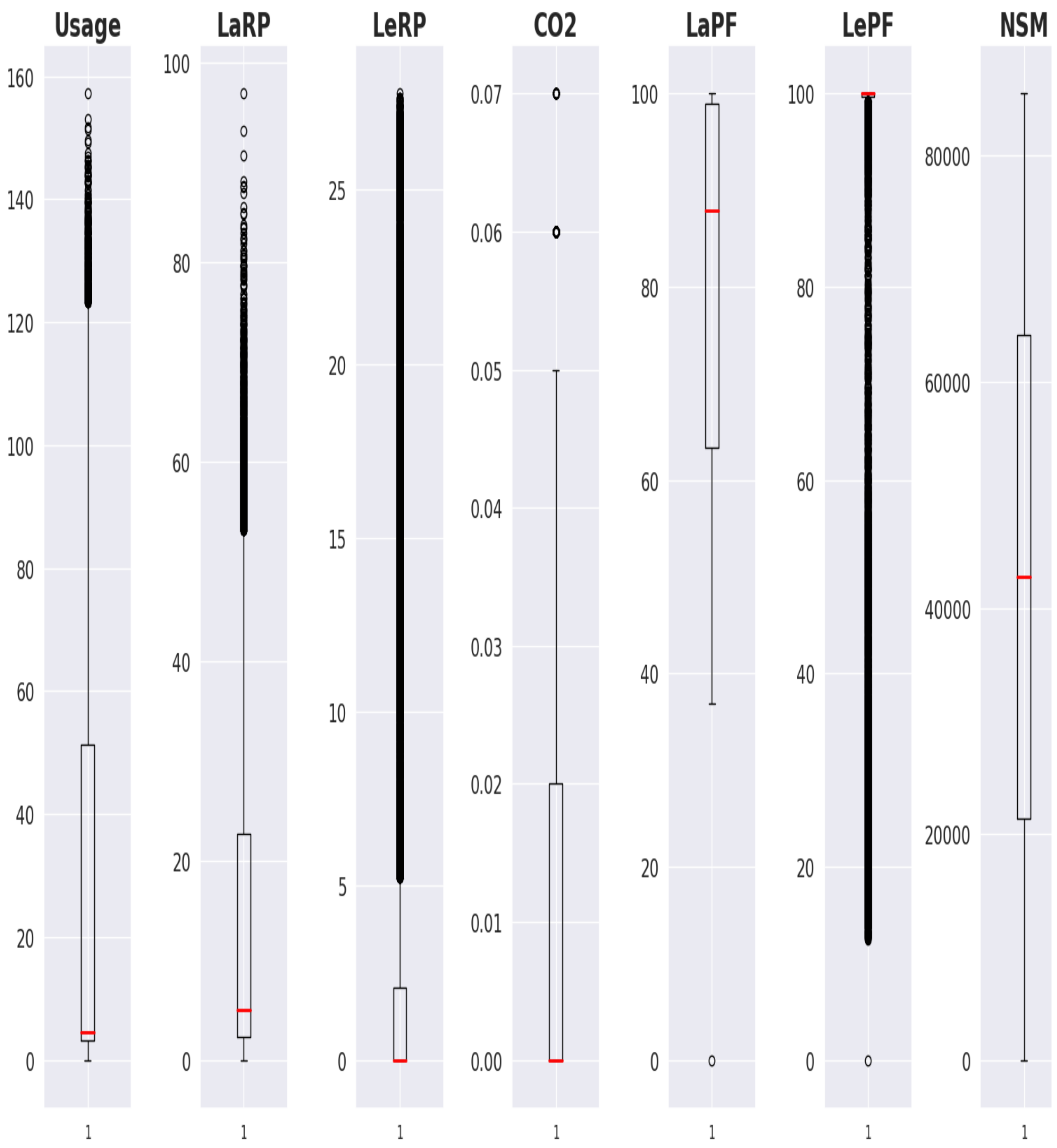
Boxplot for lagging current power factor

```
sns.set(rc={'figure.figsize':(5,6)})
plt.boxplot(df.iloc[:,4],boxprops=dict(color='green'),capprops=dict(color='red'))
plt.show()
```



```
fig, axs=plt.subplots(1, len(df.columns),figsize=(40,10))
for i, ax in enumerate(axs.flat):
  ax.boxplot(df.iloc[:,i],medianprops=dict(color='red',linewidth=2))
  ax.set_title(df.columns[i],fontsize=20,fontweight='bold')
  ax.tick_params(axis='y',labelsize=15)

plt.tight_layout()
plt.show()
```

Box plots for all continuous variables

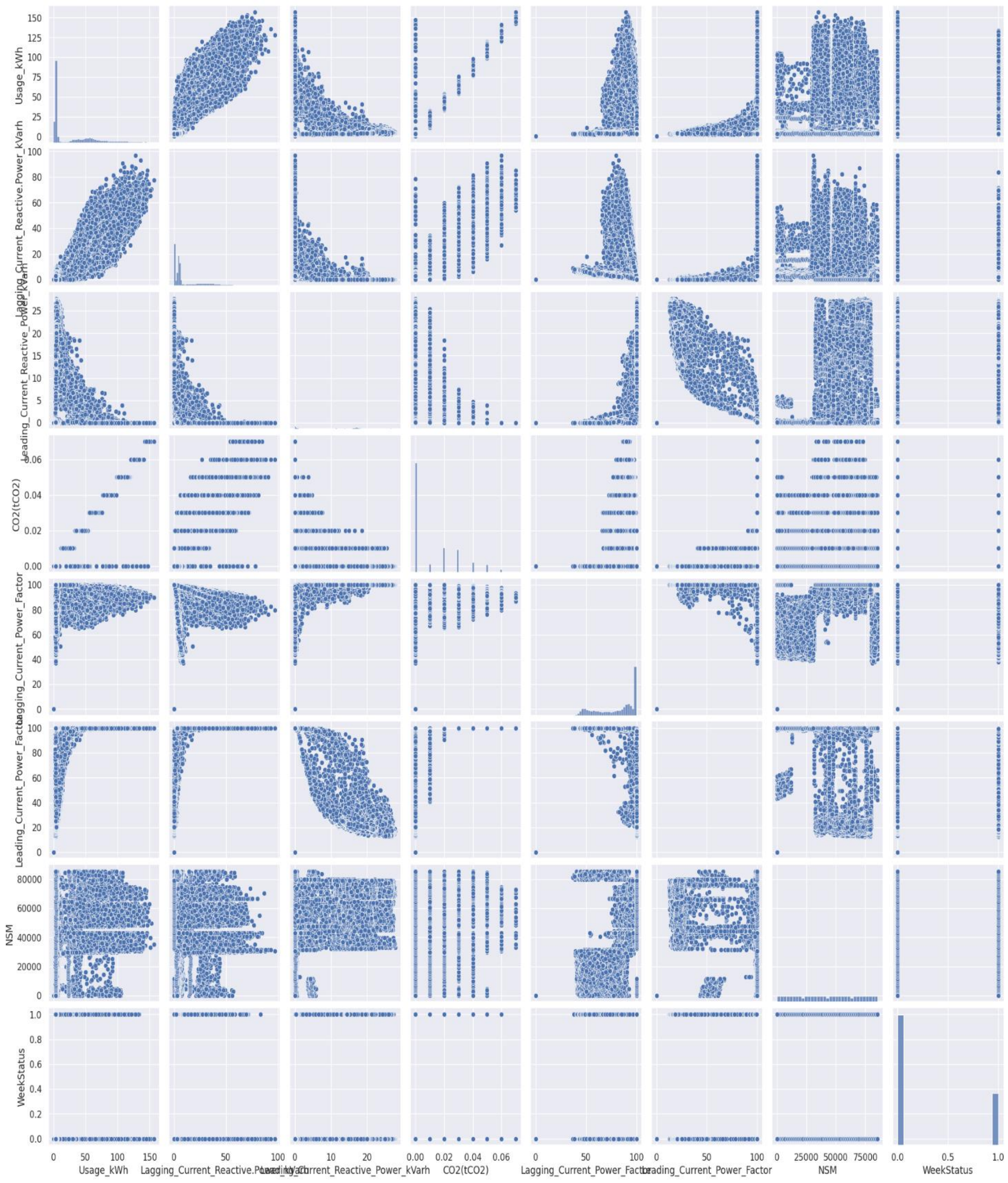


Here, we can observe LePF contains a greater number of outliers.

Next greater number of outliers contains in leading current reactive power.

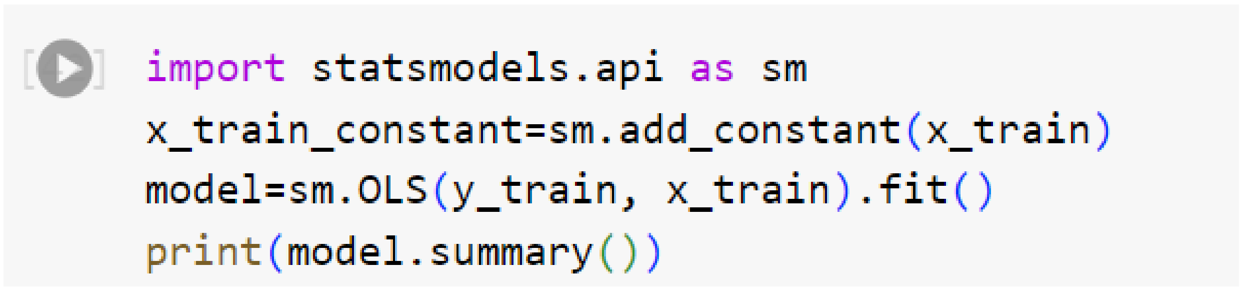
8.5 Pair Plot:

```
[28]  sns.pairplot(X,diag_kind='hist')
```



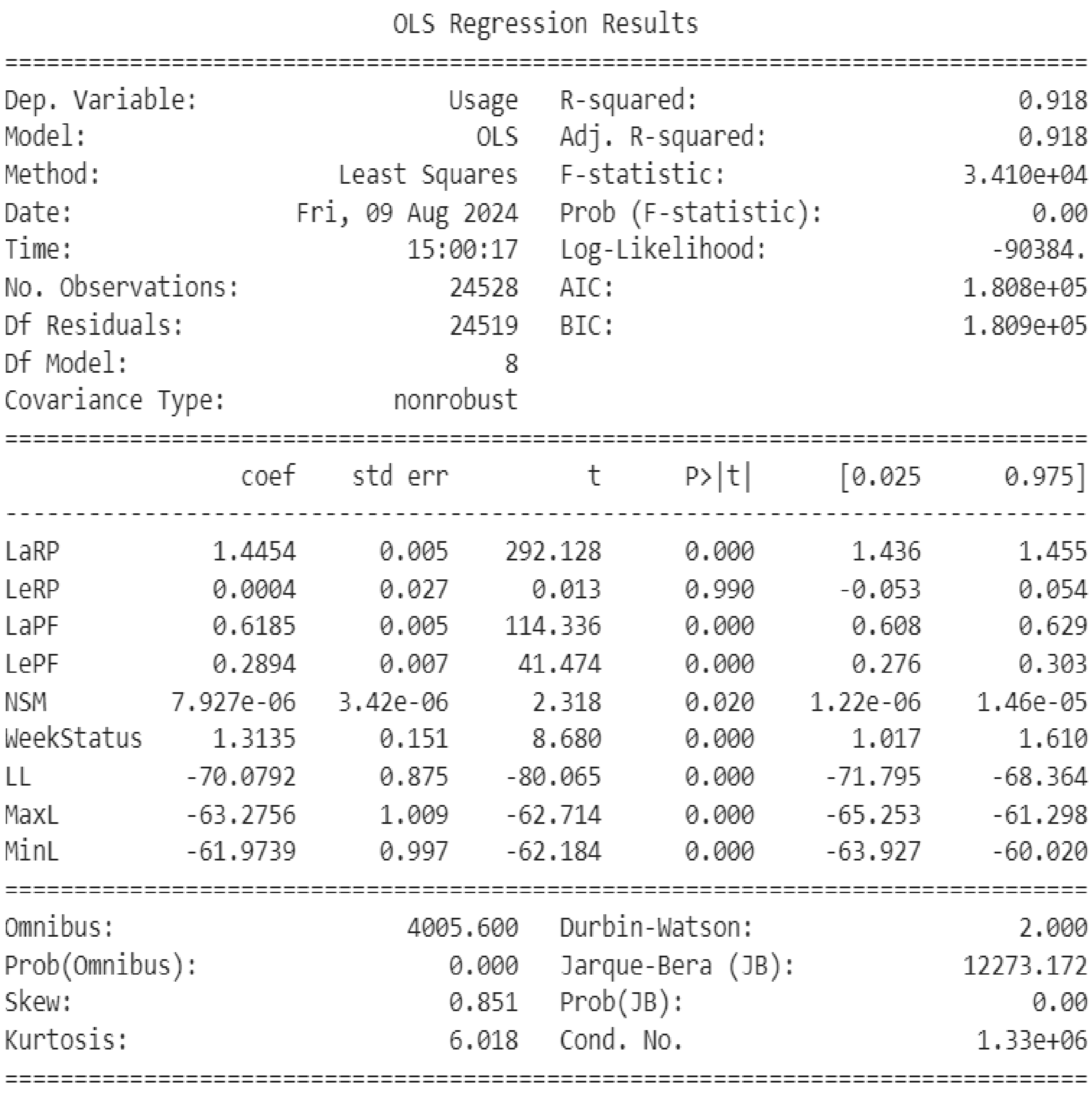
9. Significance Test:

Fitting ordinary least squares method:



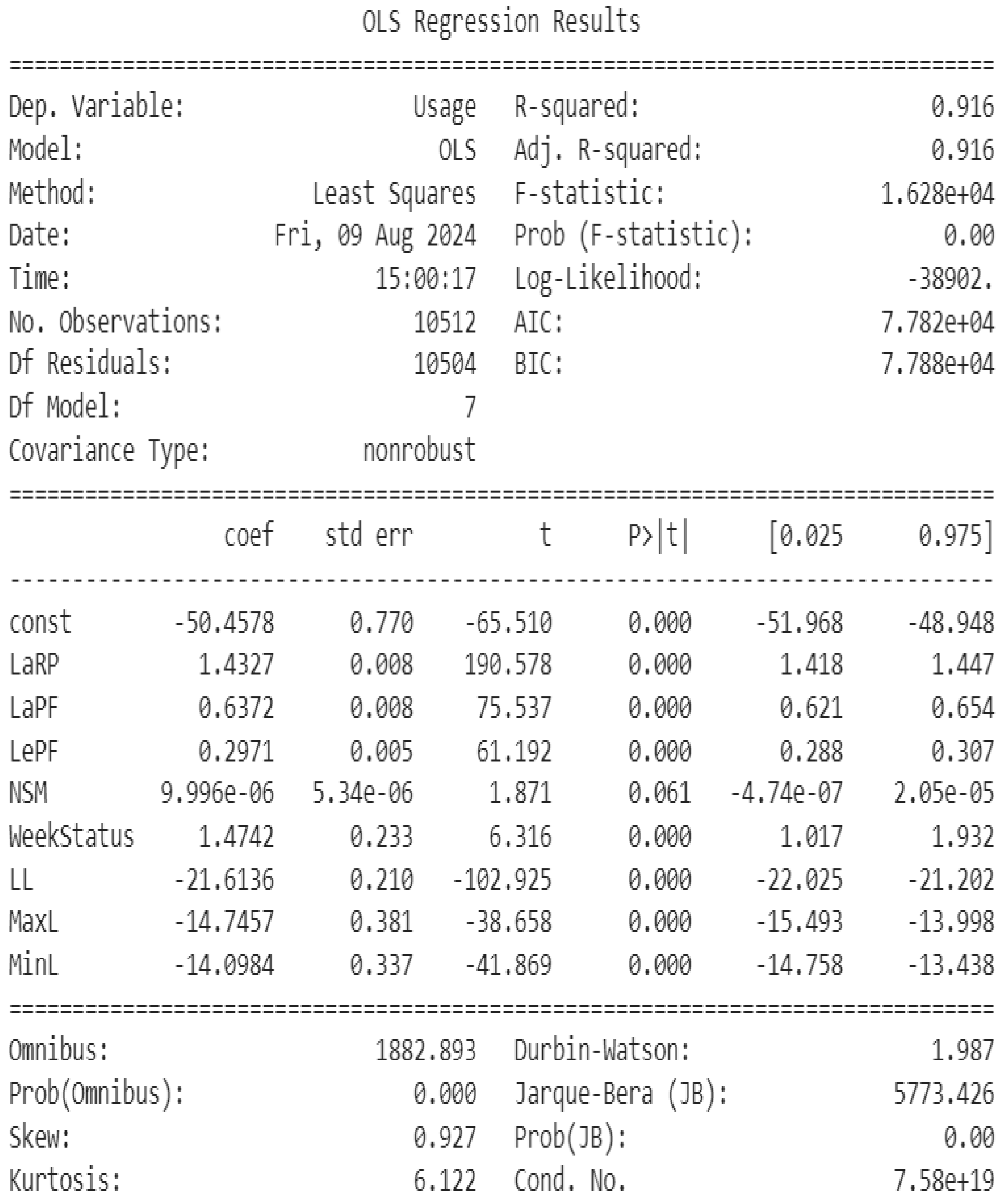
```
import statsmodels.api as sm
x_train_constant=sm.add_constant(x_train)
model=sm.OLS(y_train, x_train).fit()
print(model.summary())
```

The variables which have the P-value greater than 0.05 are said to be insignificant variable and less than 0.05 are considered to be significant variable.



```
                            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
==============================================================================
```

Here we can observe **Leading current reactive power (LeRP)** have greatest **P-value which is greater than 0.05**. We call it as **insignificant variable,** so we have removed that variable and again fitted the OLS model after **removing LeRP**.

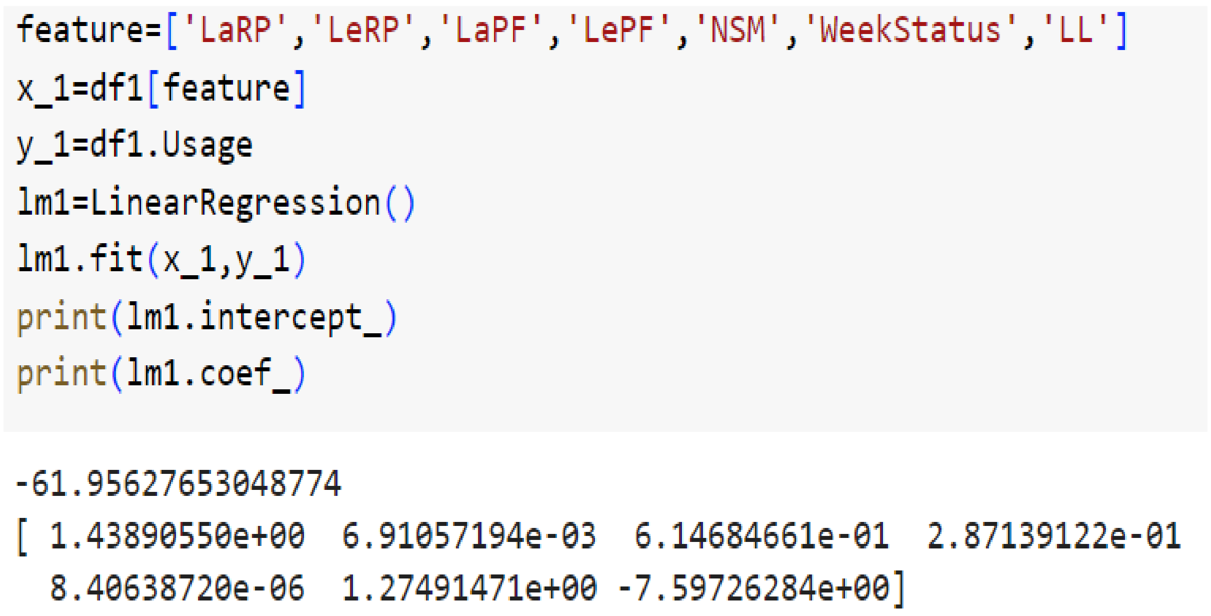


```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Usage   R-squared:                       0.916
Model:                            OLS   Adj. R-squared:                  0.916
Method:                 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
```

Here, we can observe the variable NSM has greater P-value we removed that variable and performed the significance test and got all the variables with significant values.

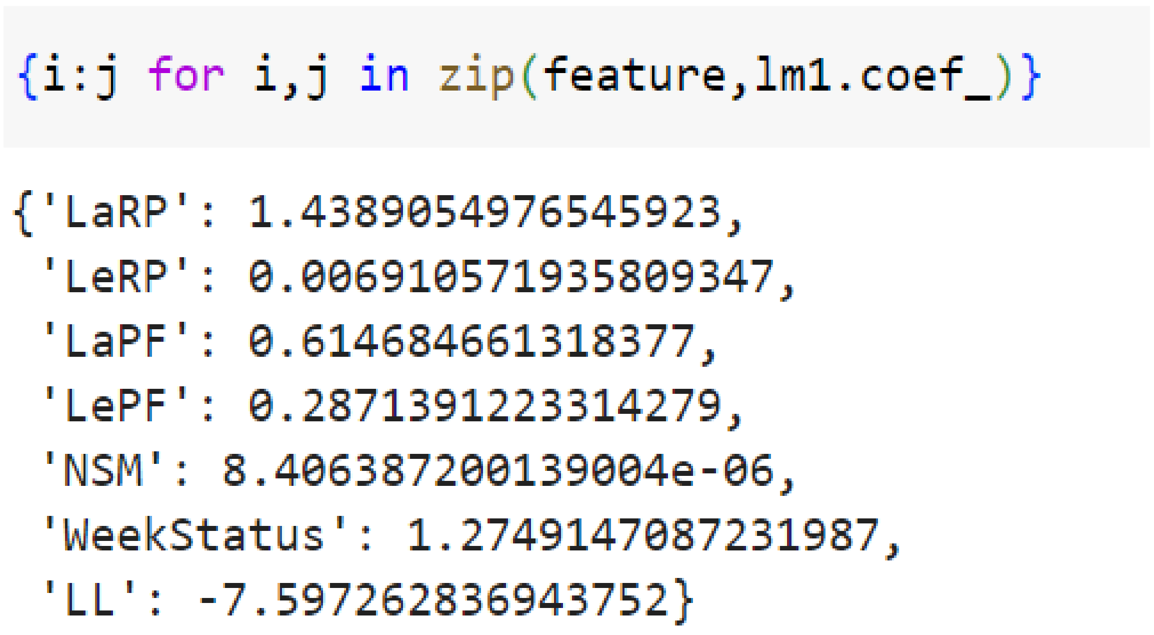
Assigning the coefficients with respect to their variables:

Model -1 (with all the features except CO2):



```
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_)
```

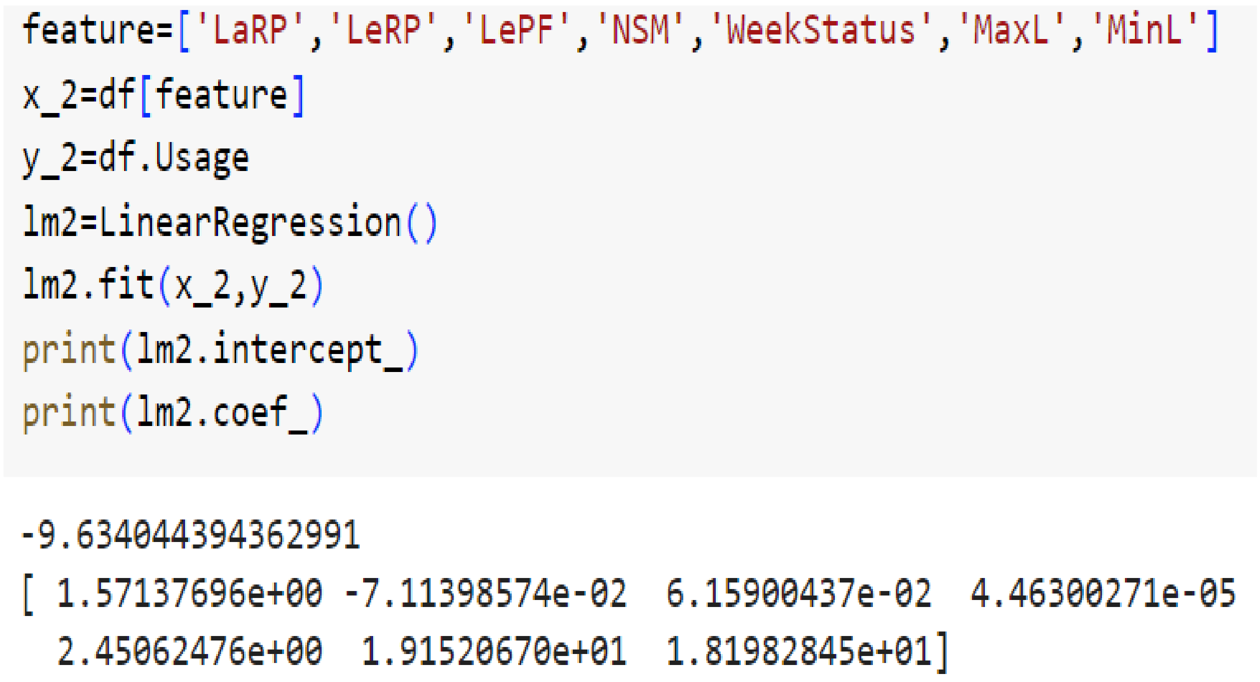
```
-61.95627653048774
[ 1.43890550e+00  6.91057194e-03  6.14684661e-01  2.87139122e-01
  8.40638720e-06  1.27491471e+00 -7.59726284e+00]
```



```
{i:j for i,j in zip(feature,lm1.coef_)}
```

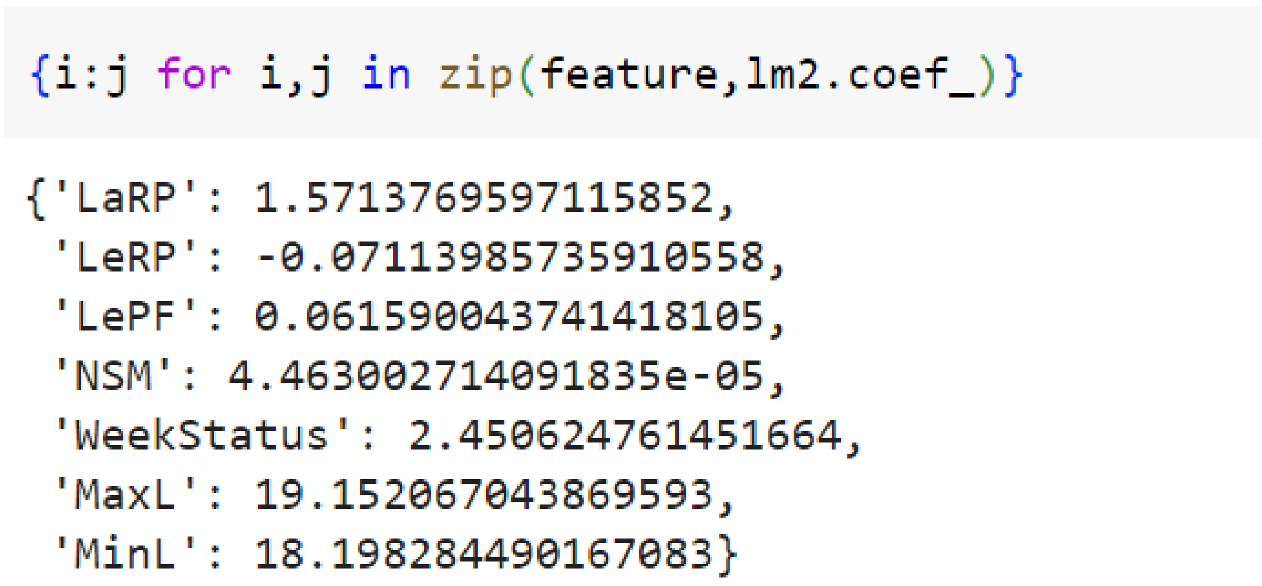
```
{'LaRP': 1.4389054976545923,
 'LeRP': 0.006910571935809347,
 'LaPF': 0.61468466131837,
 'LePF': 0.287139123314279,
 'NSM': 8.406387200139004e-06,
 'WeekStatus': 1.274914708723198,
 'LL': -7.59726283694375}
```

Model – 2 (without multicollinear variables):



```
feature=['LaRP','LeRP','LePF','NSM','WeekStatus','MaxL','MinL']
x_2=df[feature]
y_2=df.Usage
lm2=LinearRegression()
lm2.fit(x_2,y_2)
print(lm2.intercept_)
print(lm2.coef_)
```

```
-9.634044394362991
[ 1.57137696e+00 -7.11398574e-02  6.15900437e-02  4.46300271e-05
  2.45062476e+00  1.91520670e+01  1.81982845e+01]
```



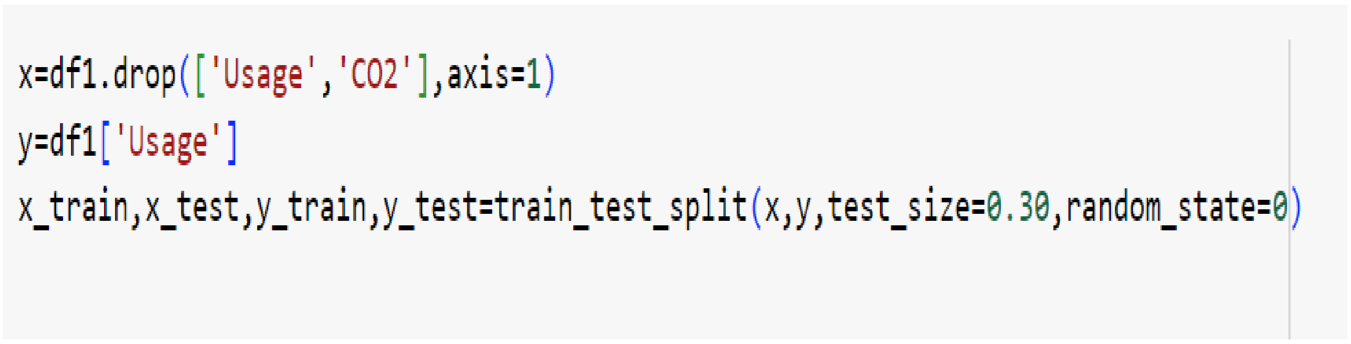
```
{i:j for i,j in zip(feature,lm2.coef_)}
```

```
{'LaRP': 1.571376959711585 2,
 'LeRP': -0.07113985735910558,
 'LePF': 0.06159004374141810 5,
 'NSM': 4.463002714091835e-05,
 'WeekStatus': 2.450624761451664,
 'MaxL': 19.15206704386959 3,
 'MinL': 18.19828449016708 3}
```

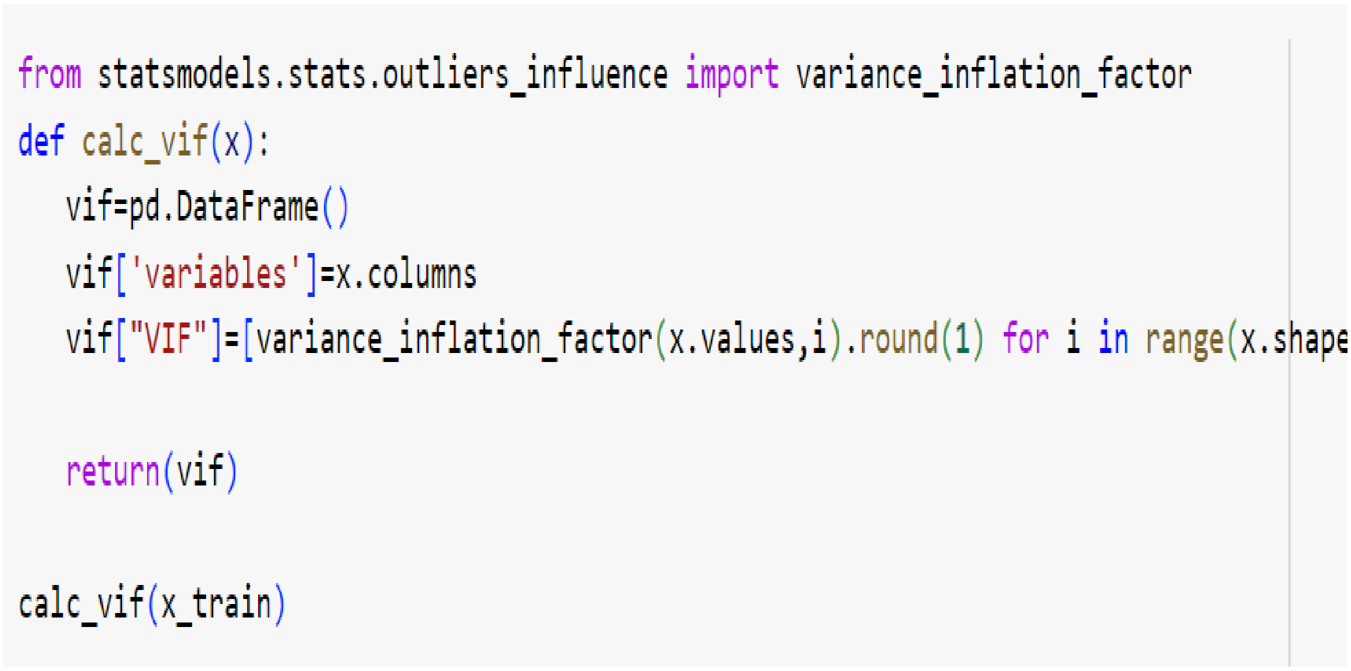
10. Multi collinearity check:

Multicollinearity is a statistical concept that describes a situation where two or more independent variables in a model are highly correlated with each other, making it difficult to determine the effect of each variable on the dependent variable. This correlation can lead to unreliable and unstable estimates of regression coefficients.

When the dataset contains any multicollinear variable we may not get accurate results we have to remove those attributes. So, we use variance inflation factor to remove those variables from our dataset.

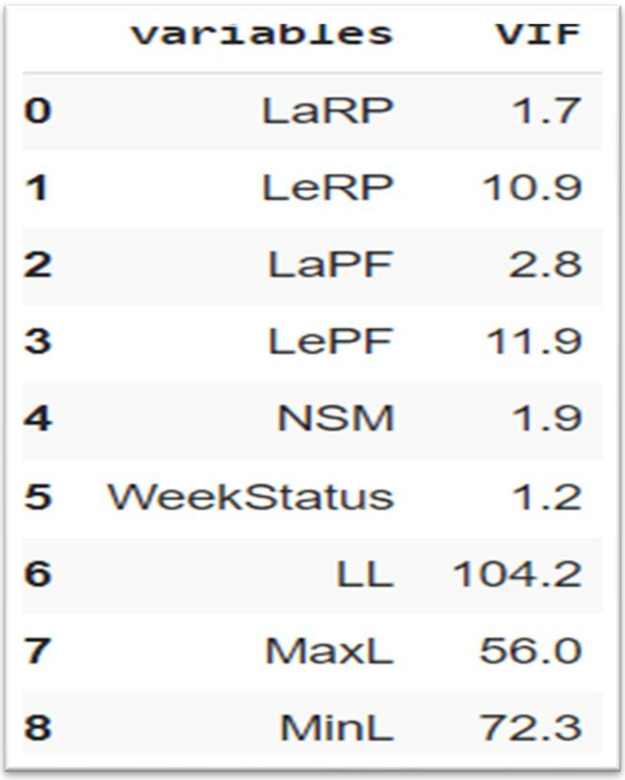


```
x=df1.drop(['Usage','CO2'],axis=1)
y=df1['Usage']
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.30,random_state=0)
```



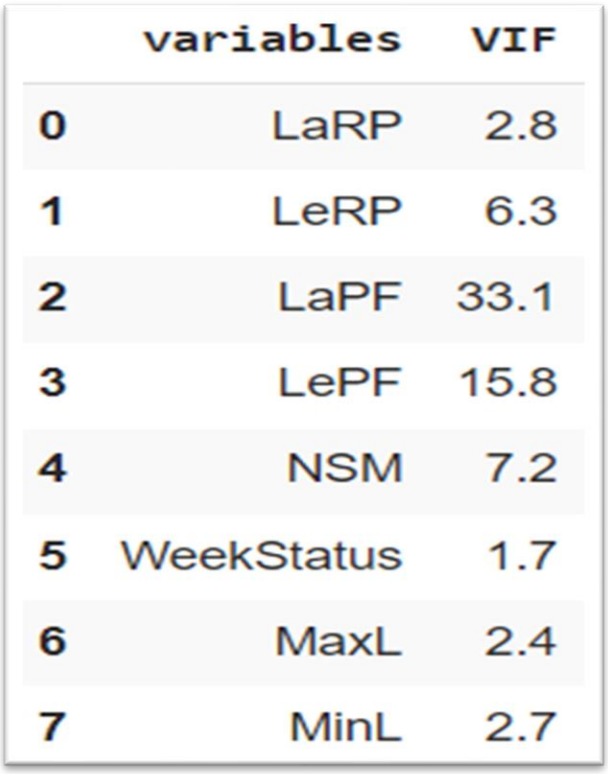
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
def calc_vif(x):
    vif=pd.DataFrame()
    vif['variables']=x.columns
    vif["VIF"]=[variance_inflation_factor(x.values,i).round(1) for i in range(x.shape

    return(vif)

calc_vif(x_train)
```

Here the variables with the highest variance inflation factor (VIF >10) will be removed.



	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

The variable LL was **removed**

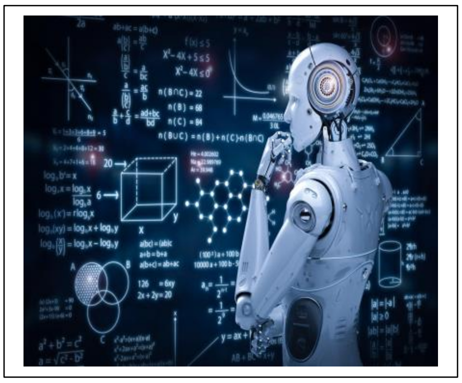


	variables	VIF
0	LaRP	2.8
1	LeRP	6.3
2	LaPF	33.1
3	LePF	15.8
4	NSM	7.2
5	WeekStatus	1.7
6	MaxL	2.4
7	MinL	2.7

The variable LaPF was removed

11. Methodology:

We used different machine learning algorithms for analysis.



Which includes linear regression, decision tree, random forest, K-nearest neighbors, support vector machines, bagging and boosting to predict power consumption.

11.1 Multiple Linear Regression (MLR):

Multiple linear regression refers to a statistical technique that uses two or more independent variables to predict the outcome of a dependent variable. The technique enables analysts to determine the variation of the model and the relative contribution of each independent variable in the total variance.

Assumptions

- Level of measurement
- Sample size
- Normality
- Linearity
- Homoscedasticity
- Multicollinearity
- Multi-variate outliers
- Normality of residuals

Types of Multiple Linear Regression (MLR):

There are several types of Multiple Linear Regression (MLR), including

- Standard Multiple Linear Regression (MLR)
- Hierarchical Multiple Linear Regression (MLR)
- Forward Multiple Linear Regression (MLR)
- Backward Multiple Linear Regression (MLR)
- Stepwise Multiple Linear Regression (MLR)

Applications

We can use multiple regression to investigate the influence of various factors on health outcomes, such as blood pressure, cholesterol, or diabetes.

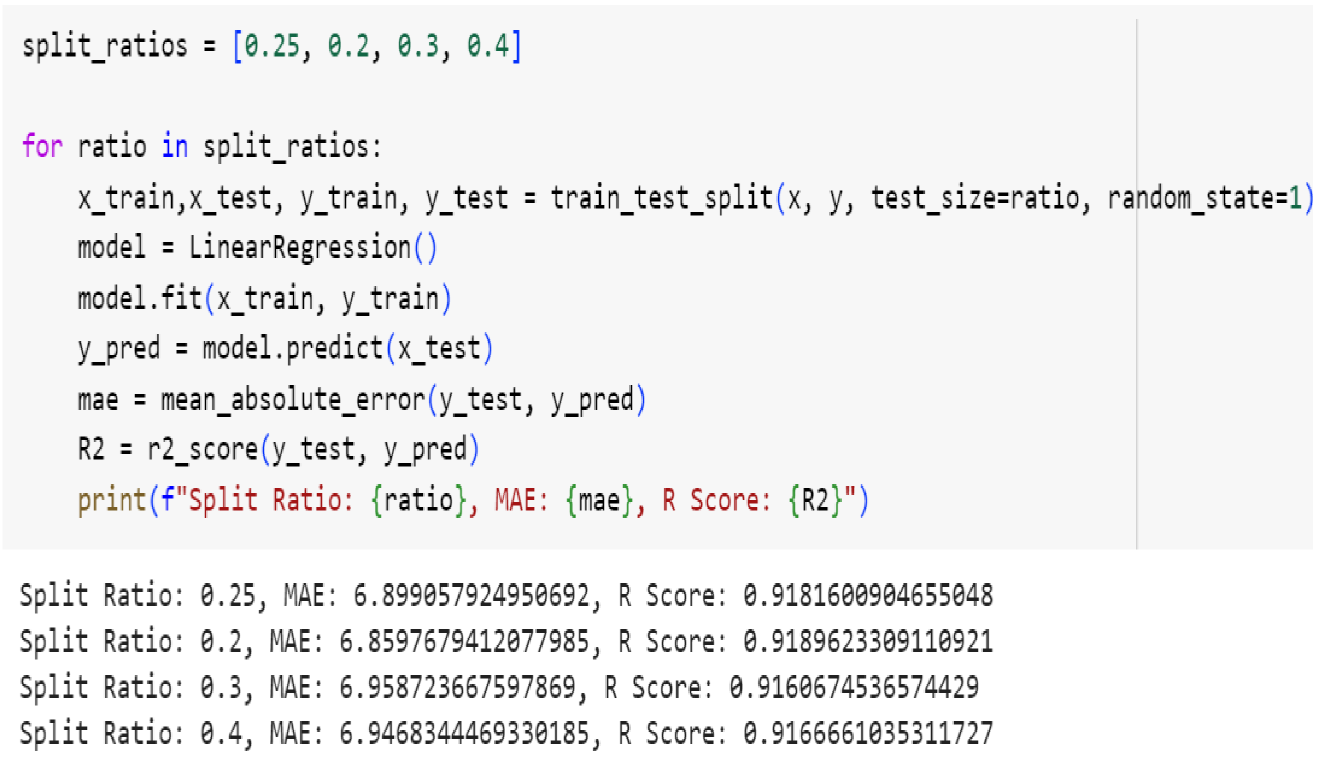
For example, you can create a model that predicts blood pressure based on variables such as age, gender, weight, diet, exercise, and medication

Steps to implement Linear Regression in Python

1. Step 1: Import the required python packages. ...
2. Step 2: Load the dataset. ...
3. Step 3: Data analysis. ...
4. Step 4: Split the dataset into dependent/independent variables. ...
5. Step 4: Split data into Train/Test sets. ...
6. Step 5: Train the regression model. ...
7. Step 6: Predict the result

Computation

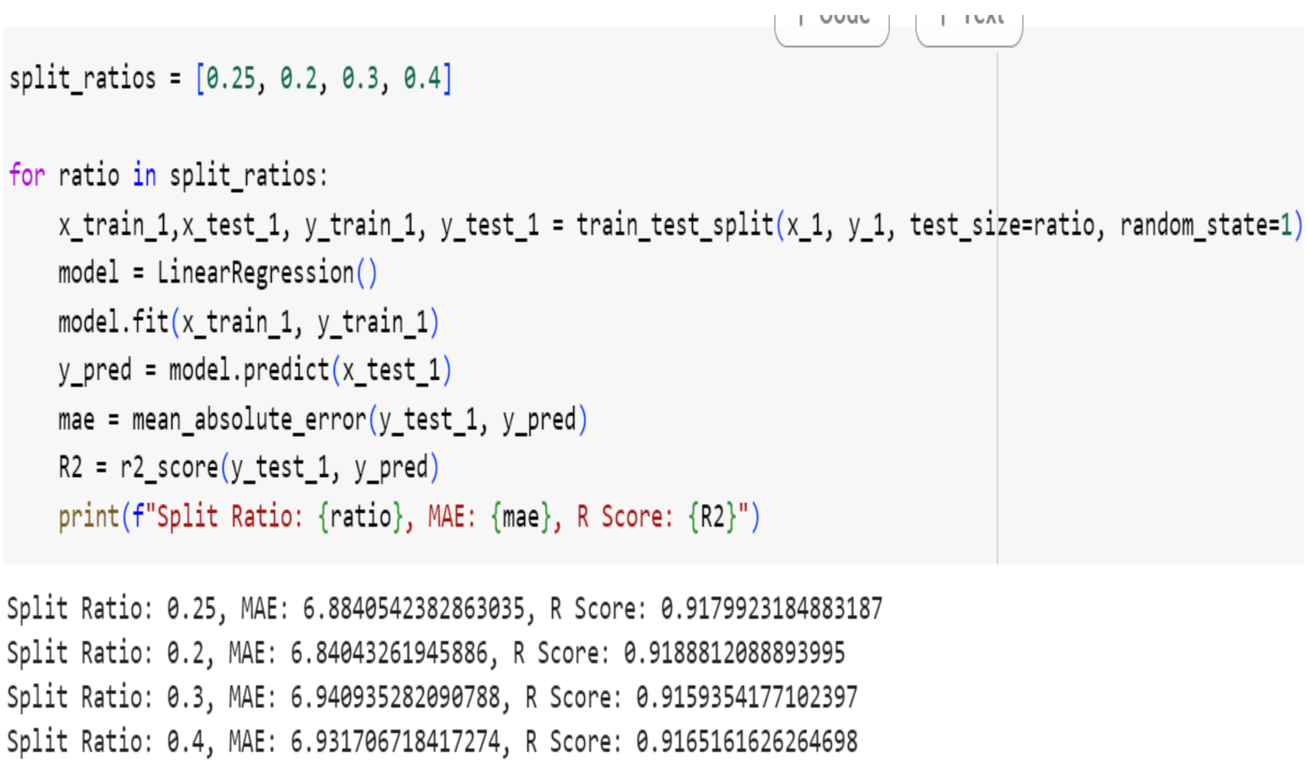
Model 1(with all the independent features except CO2):



```
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)
    model = LinearRegression()
    model.fit(x_train, y_train)
    y_pred = model.predict(x_test)
    mae = mean_absolute_error(y_test, y_pred)
    R2 = r2_score(y_test, y_pred)
    print(f"Split Ratio: {ratio}, MAE: {mae}, R Score: {R2}")
```

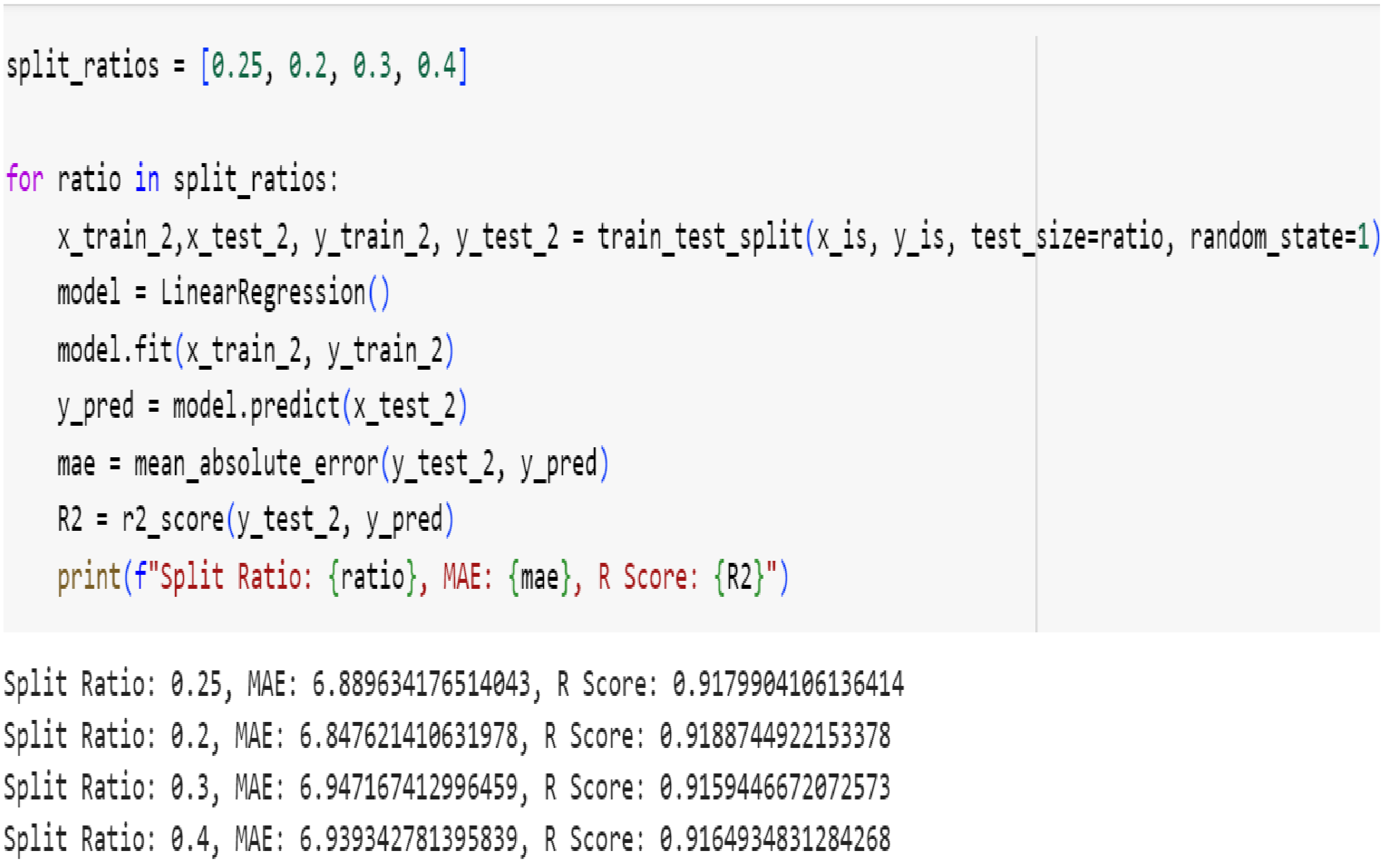
```
Split Ratio: 0.25, MAE: 6.899057924950692, R Score: 0.9181600904655048
Split Ratio: 0.2, MAE: 6.8597679412077985, R Score: 0.9189623309110921
Split Ratio: 0.3, MAE: 6.958723667597869, R Score: 0.9160674536574429
Split Ratio: 0.4, MAE: 6.9468344469330185, R Score: 0.9166661035311727
```

Model 2(without multicollinear variables):


```
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)
    model = LinearRegression()
    model.fit(x_train_1, y_train_1)
    y_pred = model.predict(x_test_1)
    mae = mean_absolute_error(y_test_1, y_pred)
    R2 = r2_score(y_test_1, y_pred)
    print(f"Split Ratio: {ratio}, MAE: {mae}, R Score: {R2}")
```

```
Split Ratio: 0.25, MAE: 6.8840542382863035, R Score: 0.9179923184883187
Split Ratio: 0.2, MAE: 6.84043261945886, R Score: 0.9188812088893995
Split Ratio: 0.3, MAE: 6.940935282090788, R Score: 0.9159354177102397
Split Ratio: 0.4, MAE: 6.931706718417274, R Score: 0.9165161626264698
```

Model 3(without insignificant variables):

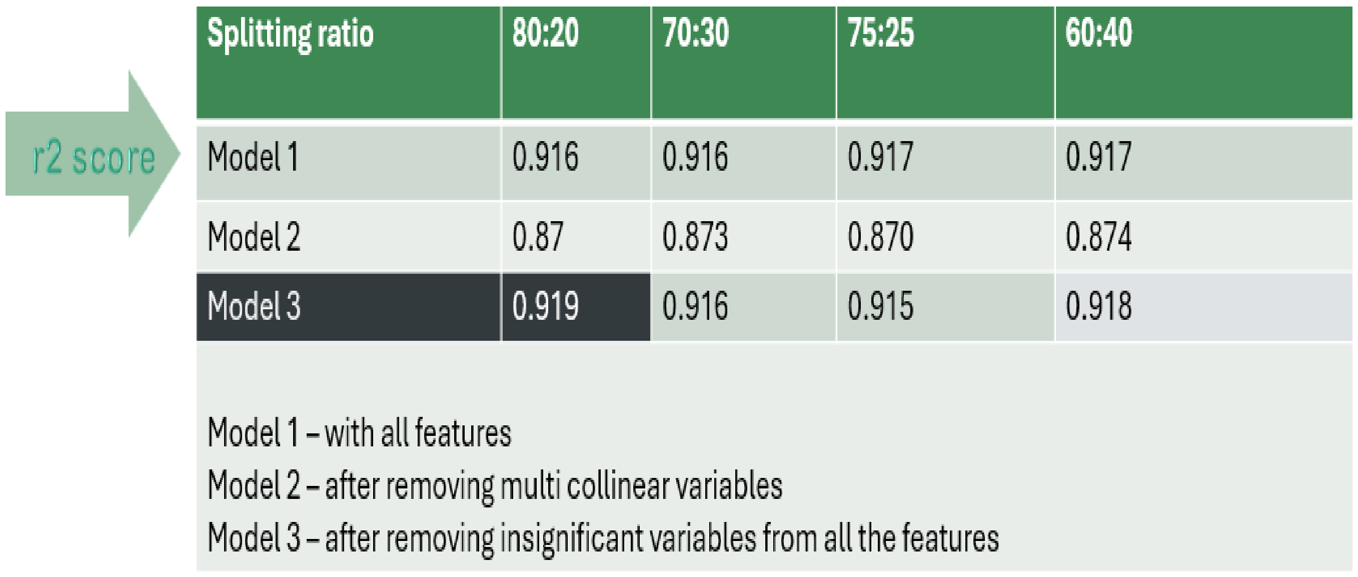

```
split_ratios = [0.25, 0.2, 0.3, 0.4]

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)
    model = LinearRegression()
    model.fit(x_train_2, y_train_2)
    y_pred = model.predict(x_test_2)
    mae = mean_absolute_error(y_test_2, y_pred)
    R2 = r2_score(y_test_2, y_pred)
    print(f"Split Ratio: {ratio}, MAE: {mae}, R Score: {R2}")
```

```
Split Ratio: 0.25, MAE: 6.889634176514043, R Score: 0.9179904106136414
Split Ratio: 0.2, MAE: 6.847621410631978, R Score: 0.9188744922153378
Split Ratio: 0.3, MAE: 6.947167412996459, R Score: 0.9159446672072573
Split Ratio: 0.4, MAE: 6.939342781395839, R Score: 0.9164934831284268
```

Analysis of a algorithm using R-square and Mean Absolute Error (MAE):

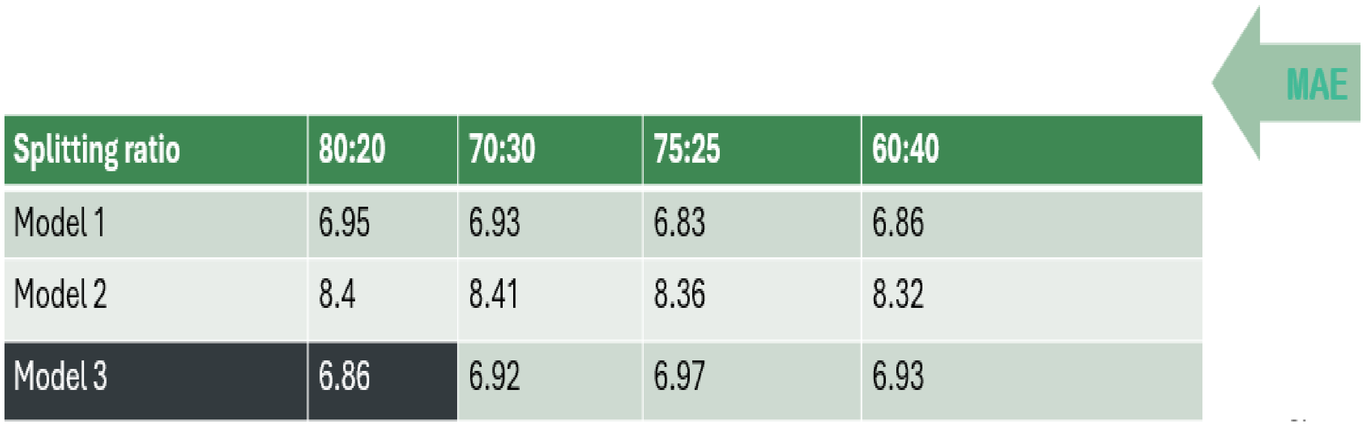
The model with greatest R- square and minimum MAE value is considered to be a best fit.

Here, model 3 with r2 score 0.919 and MAE value 6.86 with splitting ratio 80:20 is considered to be a best fit



Splitting ratio	80:20	70:30	75:25	60:40
Model 1	0.916	0.916	0.917	0.917
Model 2	0.87	0.873	0.870	0.874
Model 3	0.919	0.916	0.915	0.918

Model 1 – with all features
Model 2 – after removing multi collinear variables
Model 3 – after removing insignificant variables from all the features



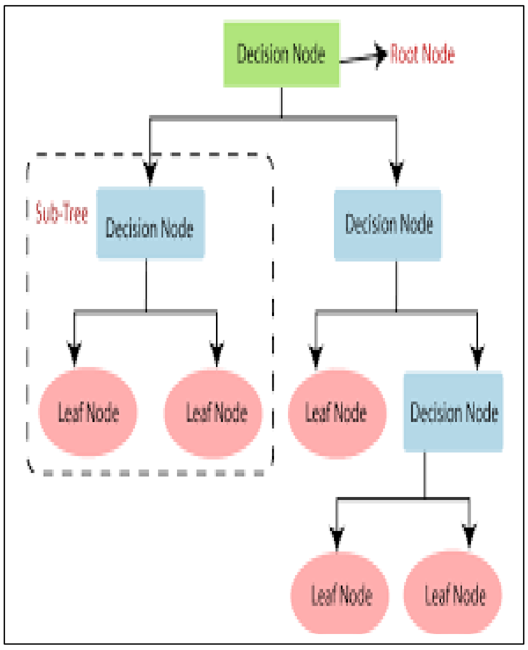
Splitting ratio	80:20	70:30	75:25	60:40
Model 1	6.95	6.93	6.83	6.86
Model 2	8.4	8.41	8.36	8.32
Model 3	6.86	6.92	6.97	6.93

11.2 Decision Tree:

Decision tree regression observes features of an object and trains a model in the structure of a tree to predict data in the future to produce meaningful continuous output. Continuous output means that the output/result is not discrete, i.e., it is not represented just by a discrete, known set of numbers or values.

Advantages

- Simple to understand and interpret
- Able to handle both numerical and categorical data
- Requires little data preparation
- Uses a white box or open-box[3] model
- Possible to validate a model using statistical tests.
- Performs well with large datasets.



Disadvantages of Decision Trees

- Prone to Overfitting.
- Unstable to Changes in the Data.
- Unstable to Noise.
- Non-Continuous.

Limitations

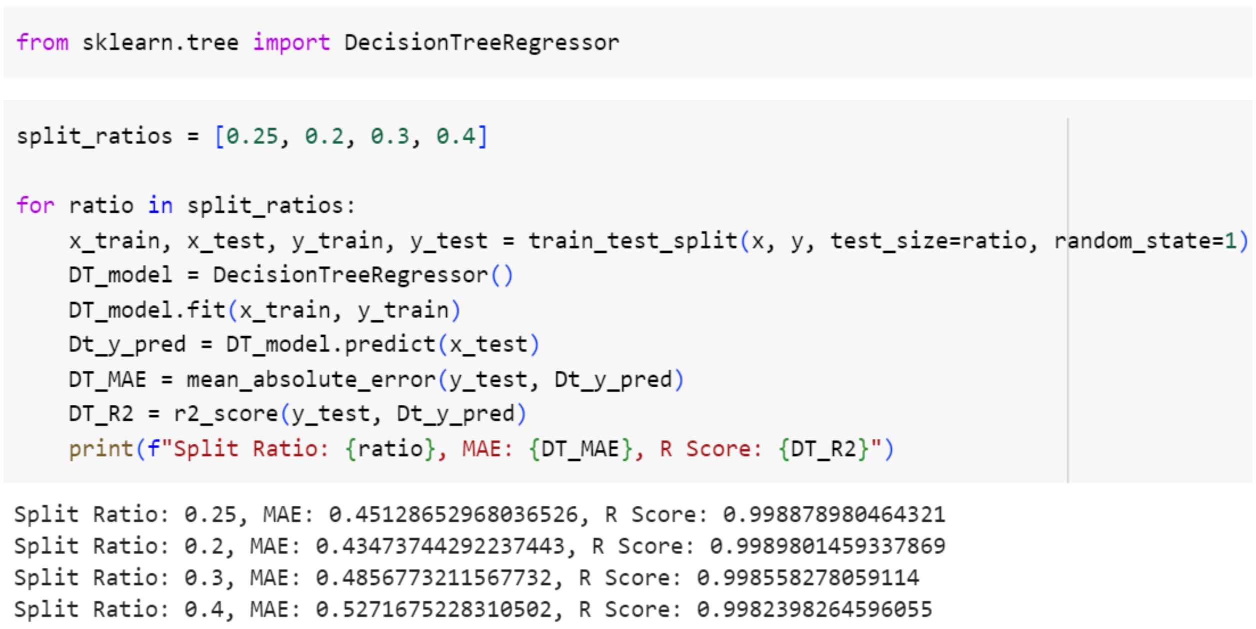
- Trees can be very non-robust. A small change in the training data can result in a large change in the tree and consequently the final predictions.
- Decision-tree learners can create over-complex trees that do not generalize well from the training data. (This is known as overfitting.) Mechanisms such as pruning are necessary to avoid this problem (with the exception of some algorithms such as the Conditional Inference approach, that does not require pruning).
- The average depth of the tree that is defined by the number of nodes or tests till classification is not guaranteed to be minimal or small under various splitting criteria.
- For data including categorical variables with different numbers of levels, information gain in decision trees is biased in favor of attributes with more levels. To counter this problem, instead of choosing the attribute with highest information gain, one can choose the attribute with the highest information gain ratio among the attributes whose information gain is greater than the mean information gain. This biases the decision tree against considering attributes with many distinct values, while not giving an unfair advantage to attributes with very low information gain. Alternatively, the issue of biased predictor selection can be avoided by the Conditional Inference approach, a two-stage approach, or adaptive leave-one-out feature selection.

Steps to implement decision tree in python:

- Step 1: Import the Necessary Libraries. ...
- Step 2: Load the Dataset. ...
- Step 3: Split the Data into Training and Test Sets. ...
- Step 4: Build the Decision Tree Model. ...
- Step 5: Make Predictions and Evaluate the Model. ...
- Step 6: Visualize the Decision Tree

Computation

Model 1(with all the independent features except CO2):

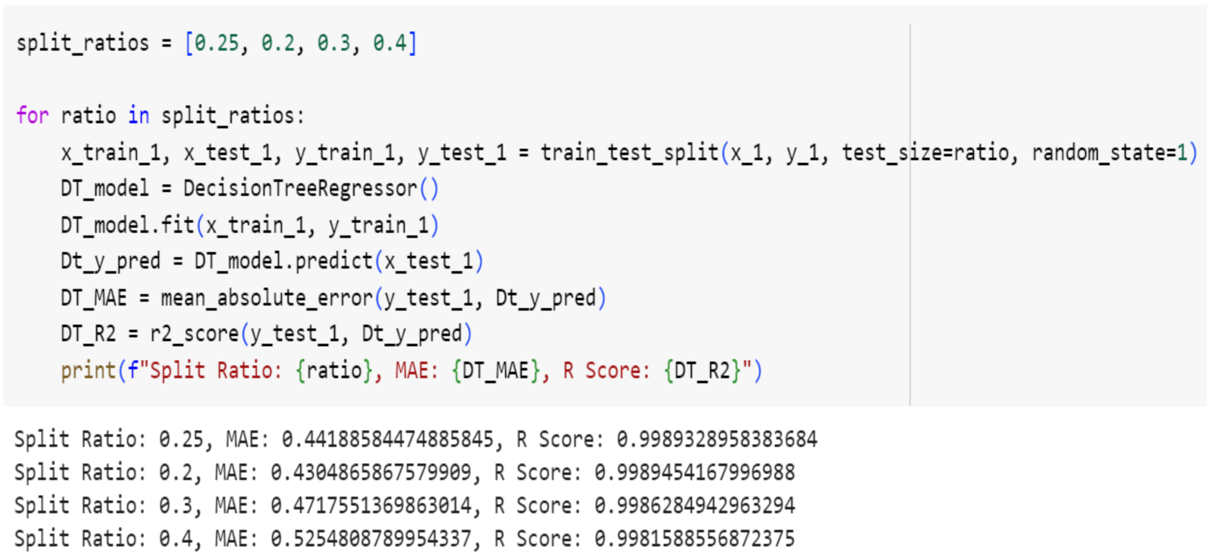


```python
from sklearn.tree import DecisionTreeRegressor
```

```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)
    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: 0.45128652968036526, R Score: 0.998878980464321
Split Ratio: 0.2, MAE: 0.43473744292237443, R Score: 0.9989801459337869
Split Ratio: 0.3, MAE: 0.4856773211567732, R Score: 0.998558278059114
Split Ratio: 0.4, MAE: 0.5271675228310502, R Score: 0.9982398264596055
```

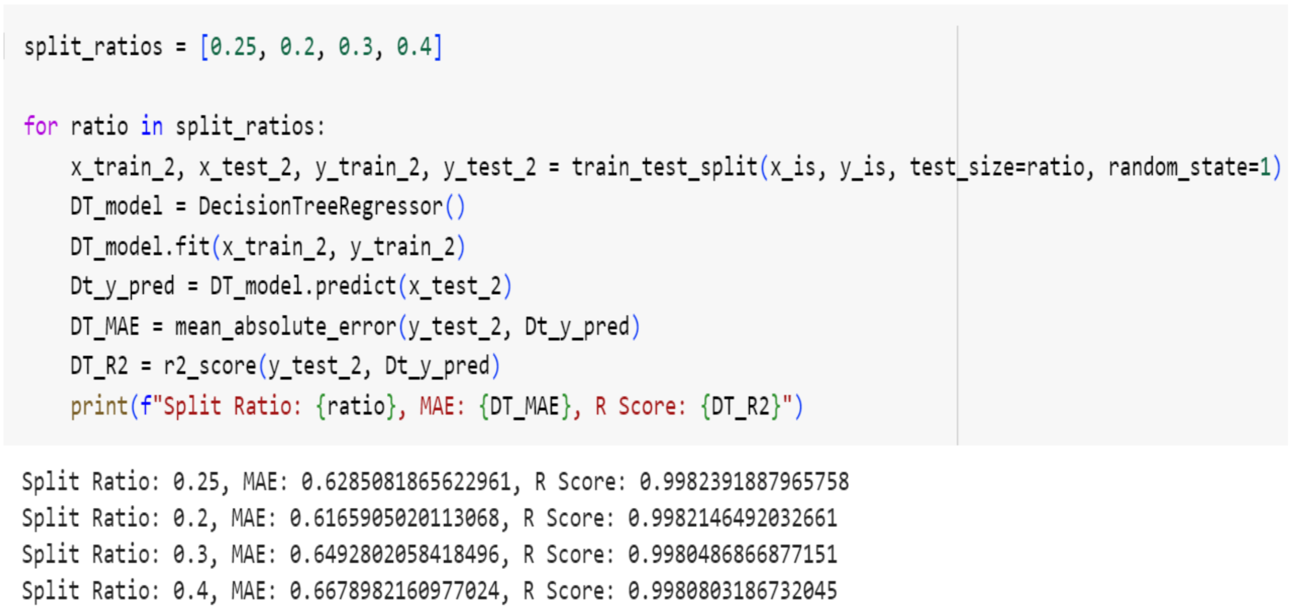
Model 2(without multicollinear variables):



```python
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 = DecisionTreeRegressor()
    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: 0.44188584474885845, R Score: 0.9989328958383684
Split Ratio: 0.2, MAE: 0.4304865867579909, R Score: 0.9989454167996988
Split Ratio: 0.3, MAE: 0.4717551369863014, R Score: 0.9986284942963294
Split Ratio: 0.4, MAE: 0.5254808789954337, R Score: 0.9981588556872375
```

Model 3(without insignificant variables):

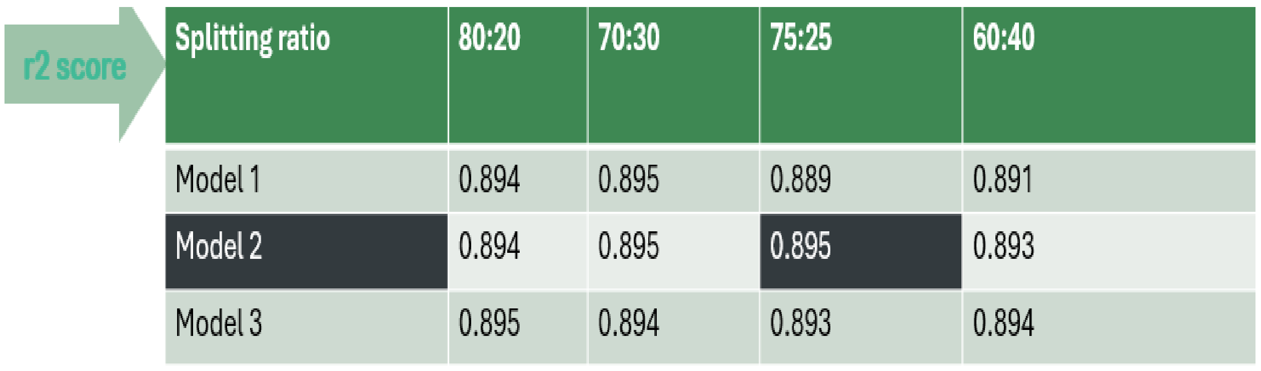


```
split_ratios = [0.25, 0.2, 0.3, 0.4]

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)
    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}")
```

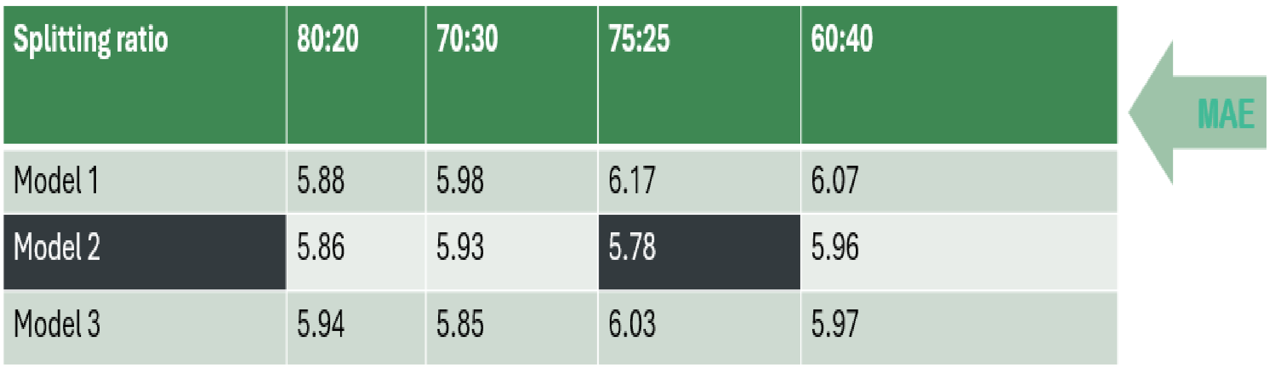
```
Split Ratio: 0.25, MAE: 0.6285081865622961, R Score: 0.9982391887965758
Split Ratio: 0.2, MAE: 0.6165905020113068, R Score: 0.9982146492032661
Split Ratio: 0.3, MAE: 0.6492802058418496, R Score: 0.9980486866877151
Split Ratio: 0.4, MAE: 0.6678982160977024, R Score: 0.9980803186732045
```

Analysis of a algorithm using R- square and Mean Absolute Error (MAE):

Here, model 2 with R- square 0.895 and MAE value 5.78 with splitting ratio 75:25 is considered to be a best fit



Splitting ratio	80:20	70:30	75:25	60:40
Model 1	0.894	0.895	0.889	0.891
Model 2	0.894	0.895	0.895	0.893
Model 3	0.895	0.894	0.893	0.894



Splitting ratio	80:20	70:30	75:25	60:40
Model 1	5.88	5.98	6.17	6.07
Model 2	5.86	5.93	5.78	5.96
Model 3	5.94	5.85	6.03	5.97

11.3 Random Forest:

Random Forest Regression in machine learning is an ensemble technique capable of performing both regression and classification tasks with the use of multiple decision trees and a technique called Bootstrap and Aggregation, commonly known as bagging. The basic idea behind this is to combine multiple decision trees in determining the final output rather than relying on individual decision trees.

Random Forest has multiple decision trees as base learning models. We randomly perform row sampling and feature sampling from the dataset forming sample datasets for every model. This part is called Bootstrap.

Properties

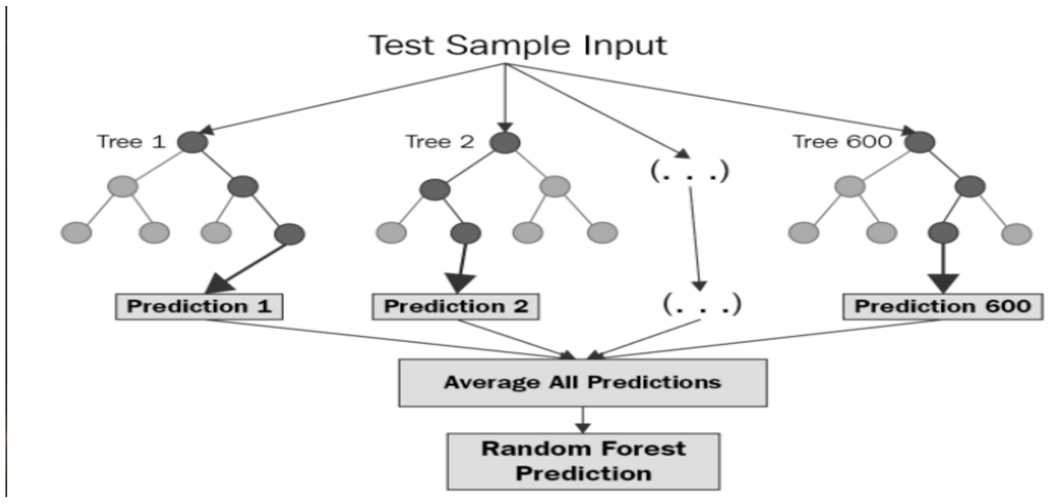
- Variable importance
- Permutation Importance
- Mean Decrease in Impurity Feature Importance
- Relationship to nearest neighbors

Advantages of Random Forest

- Random forests are highly accurate and powerful.
- Random forests are easy to use, interpret and visualize.
- The training time of a random forest is fast compared to many other techniques.
- They are resistant to overfitting and can be used to estimate missing data

Disadvantages of Random Forest

- Random forests are prone to overfitting if the data contains a large number of features.
- Random forests are very slow in making predictions when compared to other algorithms.
- They are not suitable for real-time applications as they require the entire dataset to be stored in memory.

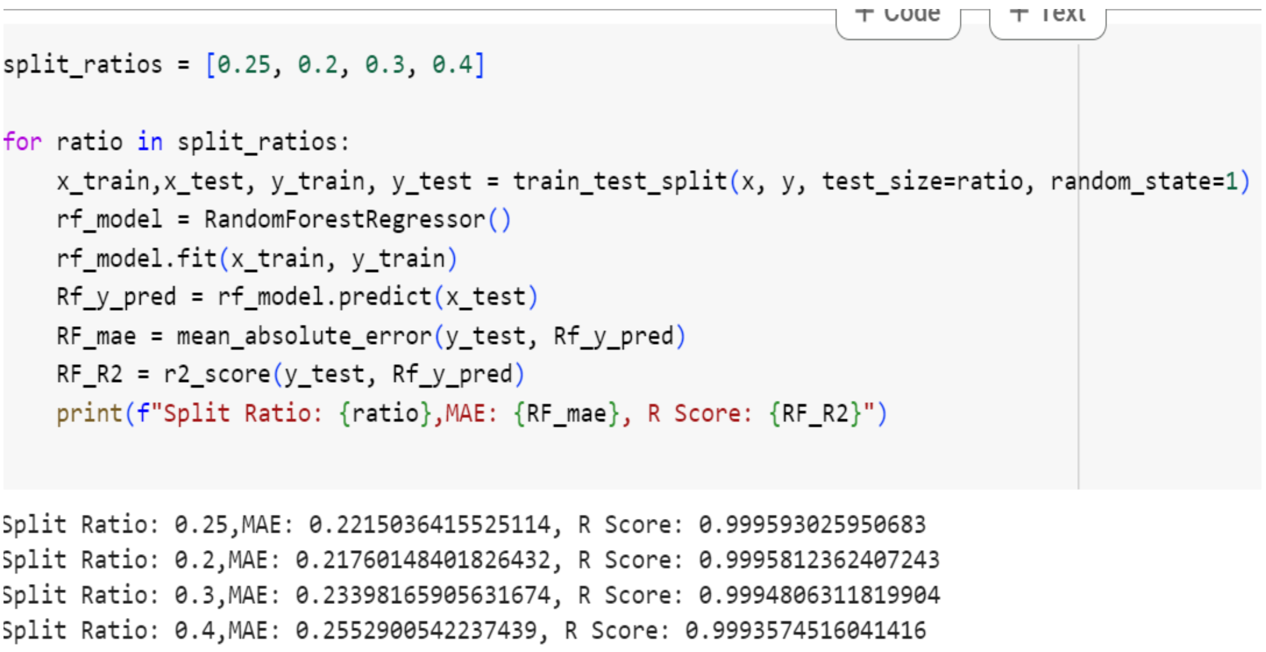


Step to implement random forest in python:

1. Step 1: Identify your dependent (y) and independent variables (X) ...
2. Step 2: Split the dataset into the Training set and Test set. ...
3. Step 3: Training the Random Forest Regression model on the whole dataset. ...
4. Step 4: Predicting the Test set results.

Computation

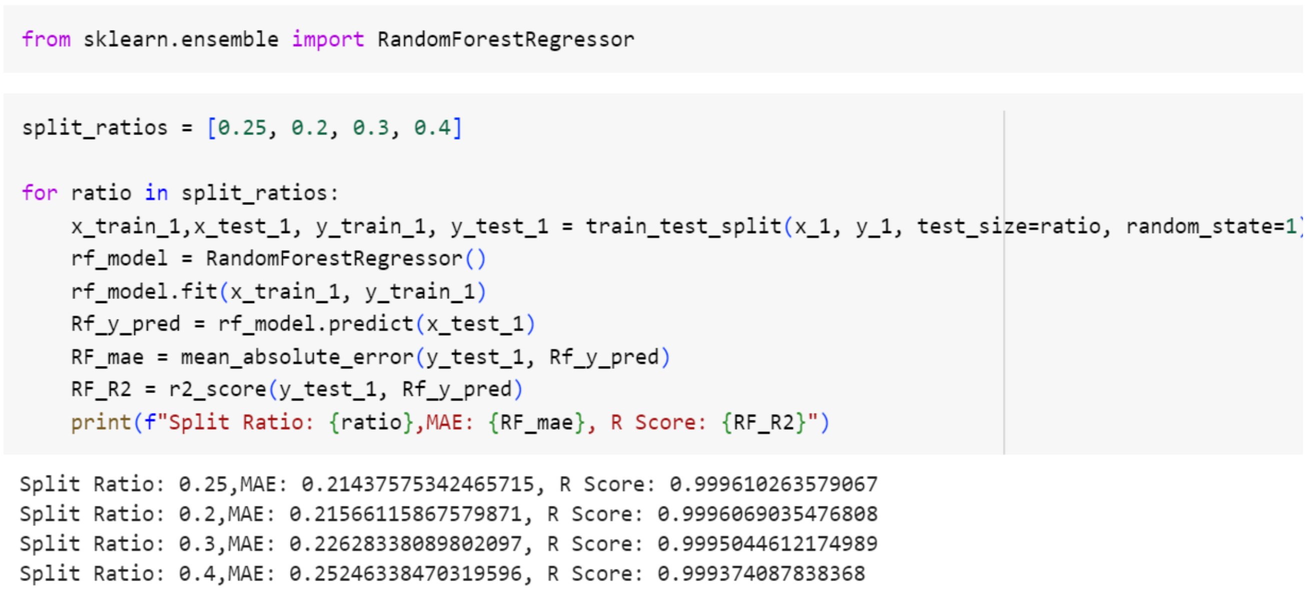
Model 1(with all the independent features except CO2):



```
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: 0.2215036415525114, R Score: 0.999593025950683
Split Ratio: 0.2,MAE: 0.21760148401826432, R Score: 0.9995812362407243
Split Ratio: 0.3,MAE: 0.23398165905631674, R Score: 0.9994806311819904
Split Ratio: 0.4,MAE: 0.2552900542237439, R Score: 0.9993574516041416
```

Model 2(without multicollinear variables):

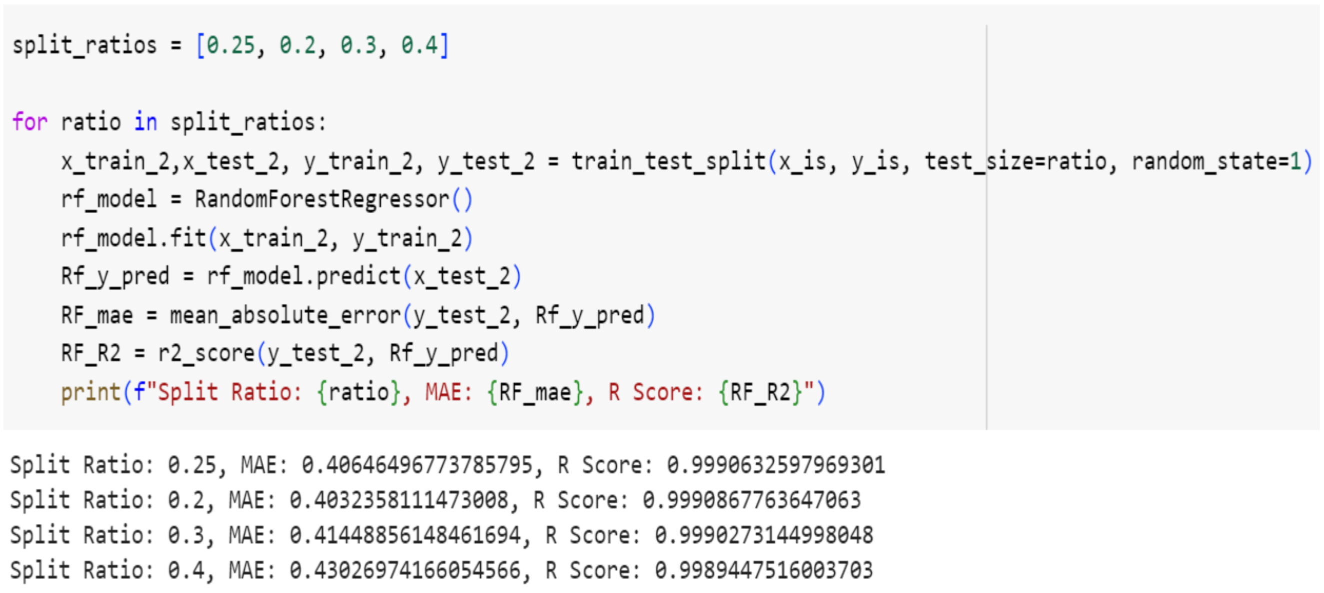


```
from sklearn.ensemble import RandomForestRegressor
```

```
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: 0.21437575342465715, R Score: 0.999610263579067
Split Ratio: 0.2,MAE: 0.21566115867579871, R Score: 0.9996069035476808
Split Ratio: 0.3,MAE: 0.22628338089802097, R Score: 0.9995044612174989
Split Ratio: 0.4,MAE: 0.25246338470319596, R Score: 0.999374087838368
```

Model 3(without insignificant variables):

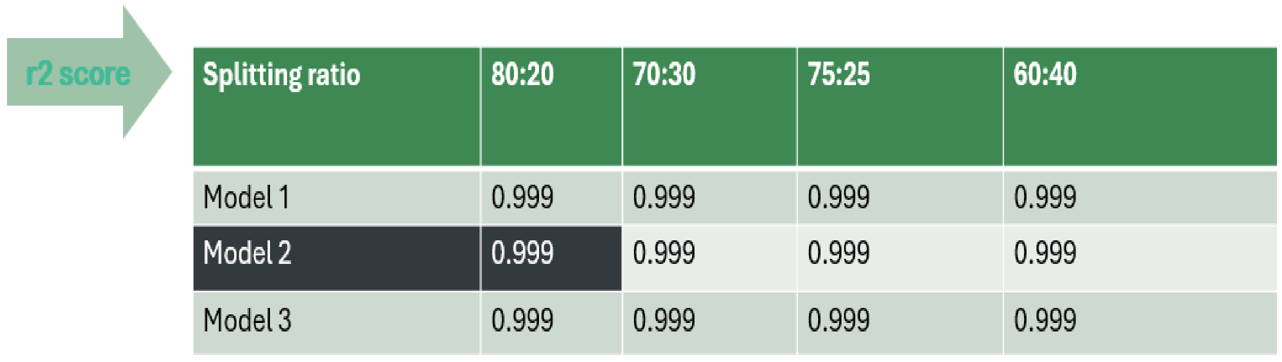


```
split_ratios = [0.25, 0.2, 0.3, 0.4]

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: 0.40646496773785795, R Score: 0.9990632597969301
Split Ratio: 0.2, MAE: 0.4032358111473008, R Score: 0.9990867763647063
Split Ratio: 0.3, MAE: 0.41448856148461694, R Score: 0.9990273144998048
Split Ratio: 0.4, MAE: 0.43026974166054566, R Score: 0.9989447516003703
```

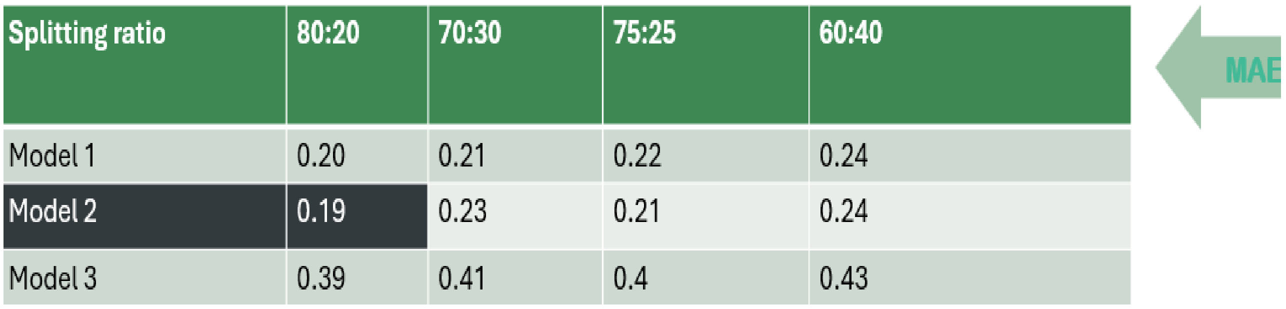
Analysis of a algorithm using R- square and Mean Absolute Error (MAE):

Here, model 2 with R- square 0.999 and MAE value 0.19 with splitting ratio 80:20 is considered to be a best fit



r2 score

Splitting ratio	80:20	70:30	75:25	60:40
Model 1	0.999	0.999	0.999	0.999
Model 2	0.999	0.999	0.999	0.999
Model 3	0.999	0.999	0.999	0.999



MAE

Splitting ratio	80:20	70:30	75:25	60:40
Model 1	0.20	0.21	0.22	0.24
Model 2	0.19	0.23	0.21	0.24
Model 3	0.39	0.41	0.4	0.43

11.4 K- Nearest Neighbors:

The *k*-nearest neighbors algorithm (*k*-NN) is a non-parametric supervised learning method first developed by Evelyn Fix and Joseph Hodges in 1951,[1] and later expanded by Thomas Cover.

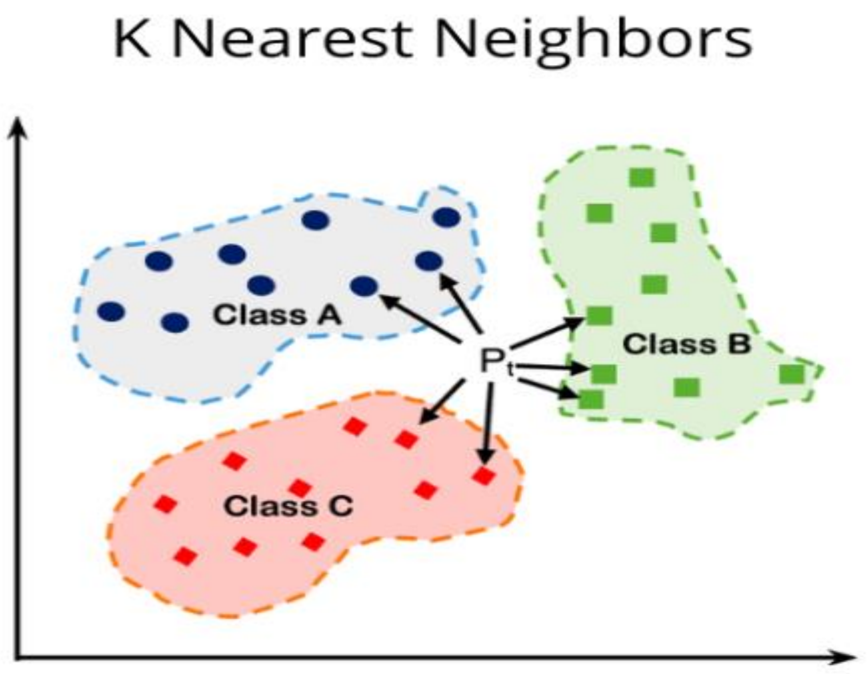
The input consists of the *k* closest training examples in a data set. In *k-NN regression*, the output is the property value for the object. This value is the average of the values of *k* nearest neighbors. If $k = 1$, then the output is simply assigned to the value of that single nearest neighbor.

Advantages:

- Easy to use: KNN is intuitive and simple to implement.
- Quick calculation time: KNN has a quick calculation time.
- No training time: KNN doesn't require an explicit training step, so all the work happens during prediction.
- Evolves with new data: As new data is added to the dataset, the prediction is adjusted without retraining a new model.
- Versatile: KNN is versatile and can learn non-linear decision boundaries when used for classification and regression.

Disadvantages:

A disadvantage of the KNN algorithm is that it does not create a generalized separable model. There is no summary equations or trees that can be produced by the training process that can be quickly applied to new records. Instead, KNN simply uses the training data itself to perform prediction.



Steps to implement KNN in python:

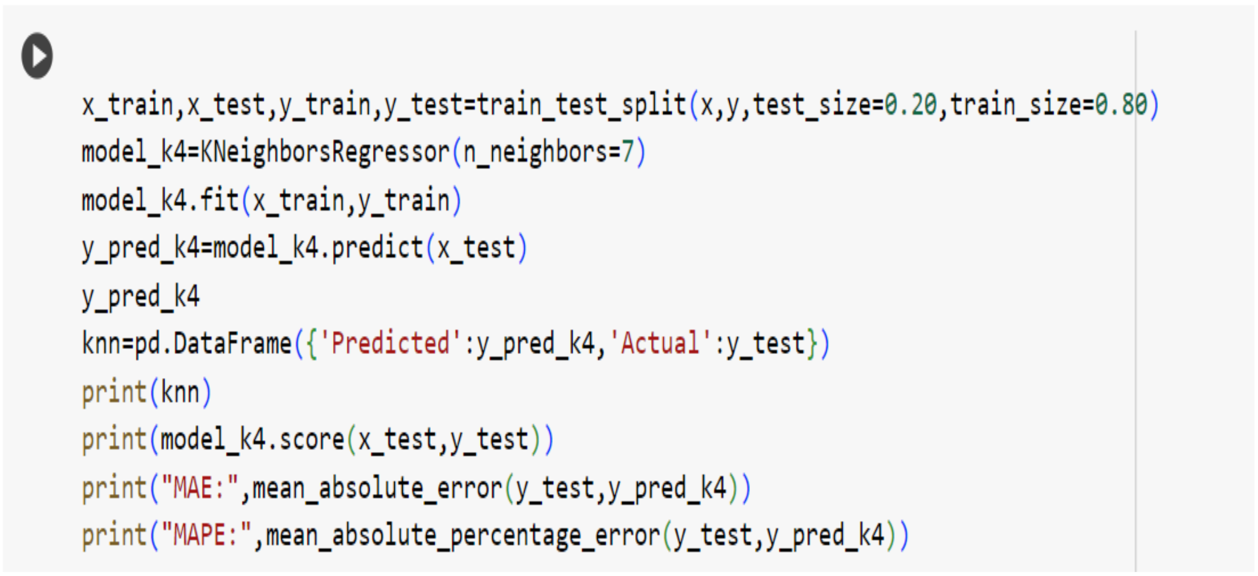
Step-1: Load the data.

Step-2: Initialize the value of k.

Step-3: For getting the predicted class, iterate from 1 to total number of training data points. Calculate the distance between test data and each row of training dataset.

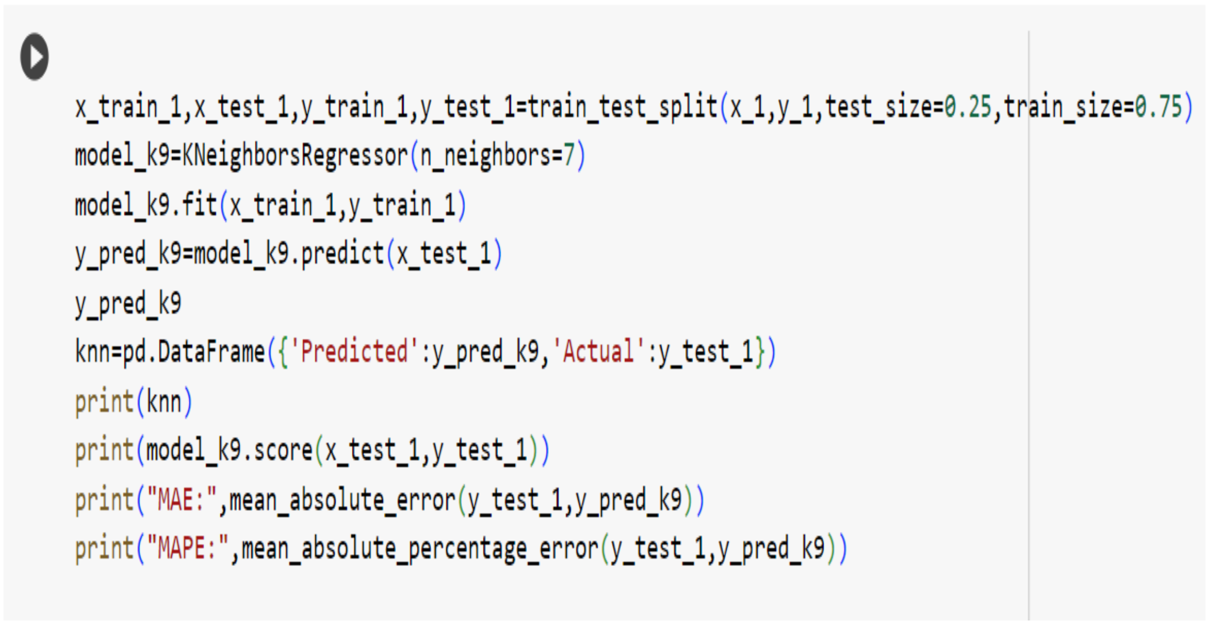
Computation

Model 1(with all the independent features except CO2):



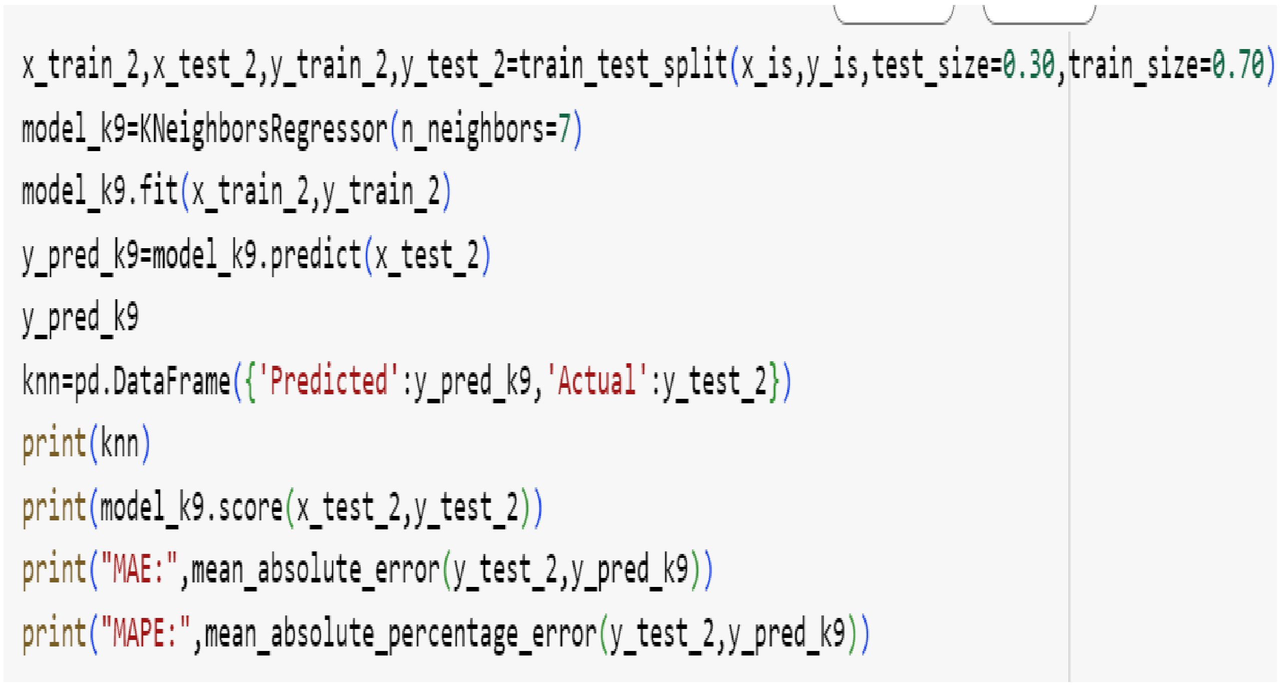
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.20,train_size=0.80)
model_k4=KNeighborsRegressor(n_neighbors=7)
model_k4.fit(x_train,y_train)
y_pred_k4=model_k4.predict(x_test)
y_pred_k4
knn=pd.DataFrame({'Predicted':y_pred_k4,'Actual':y_test})
print(knn)
print(model_k4.score(x_test,y_test))
print("MAE:",mean_absolute_error(y_test,y_pred_k4))
print("MAPE:",mean_absolute_percentage_error(y_test,y_pred_k4))
```

Model 2(without multicollinear variables):



```
x_train_1,x_test_1,y_train_1,y_test_1=train_test_split(x_1,y_1,test_size=0.25,train_size=0.75)
model_k9=KNeighborsRegressor(n_neighbors=7)
model_k9.fit(x_train_1,y_train_1)
y_pred_k9=model_k9.predict(x_test_1)
y_pred_k9
knn=pd.DataFrame({'Predicted':y_pred_k9,'Actual':y_test_1})
print(knn)
print(model_k9.score(x_test_1,y_test_1))
print("MAE:",mean_absolute_error(y_test_1,y_pred_k9))
print("MAPE:",mean_absolute_percentage_error(y_test_1,y_pred_k9))
```

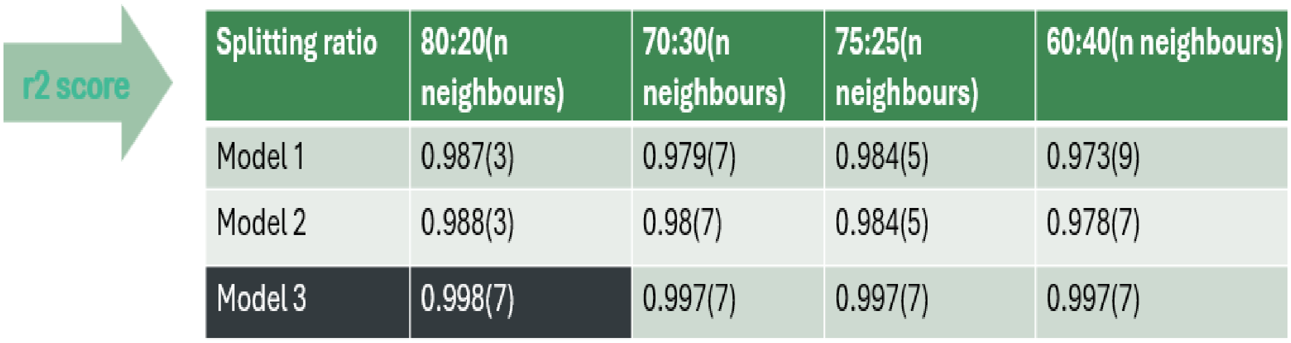
Model 3(without insignificant variables):



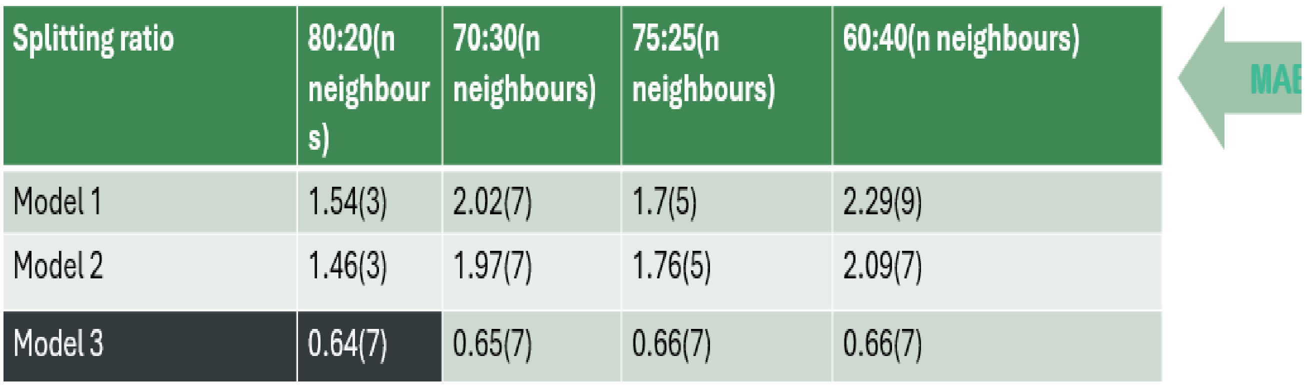
```
x_train_2,x_test_2,y_train_2,y_test_2=train_test_split(x_is,y_is,test_size=0.30,train_size=0.70)
model_k9=KNeighborsRegressor(n_neighbors=7)
model_k9.fit(x_train_2,y_train_2)
y_pred_k9=model_k9.predict(x_test_2)
y_pred_k9
knn=pd.DataFrame({'Predicted':y_pred_k9,'Actual':y_test_2})
print(knn)
print(model_k9.score(x_test_2,y_test_2))
print("MAE:",mean_absolute_error(y_test_2,y_pred_k9))
print("MAPE:",mean_absolute_percentage_error(y_test_2,y_pred_k9))
```

Analysis of a algorithm using R- square and Mean Absolute Error (MAE):

Here, model 3 with R-square 0.998 and MAE value 0.64 with splitting ratio 80:20 is considered to be a best fit.



Splitting ratio	80:20(n neighbours)	70:30(n neighbours)	75:25(n neighbours)	60:40(n neighbours)
Model 1	0.987(3)	0.979(7)	0.984(5)	0.973(9)
Model 2	0.988(3)	0.98(7)	0.984(5)	0.978(7)
Model 3	0.998(7)	0.997(7)	0.997(7)	0.997(7)



Splitting ratio	80:20(n neighbours)	70:30(n neighbours)	75:25(n neighbours)	60:40(n neighbours)
Model 1	1.54(3)	2.02(7)	1.7(5)	2.29(9)
Model 2	1.46(3)	1.97(7)	1.76(5)	2.09(7)
Model 3	0.64(7)	0.65(7)	0.66(7)	0.66(7)

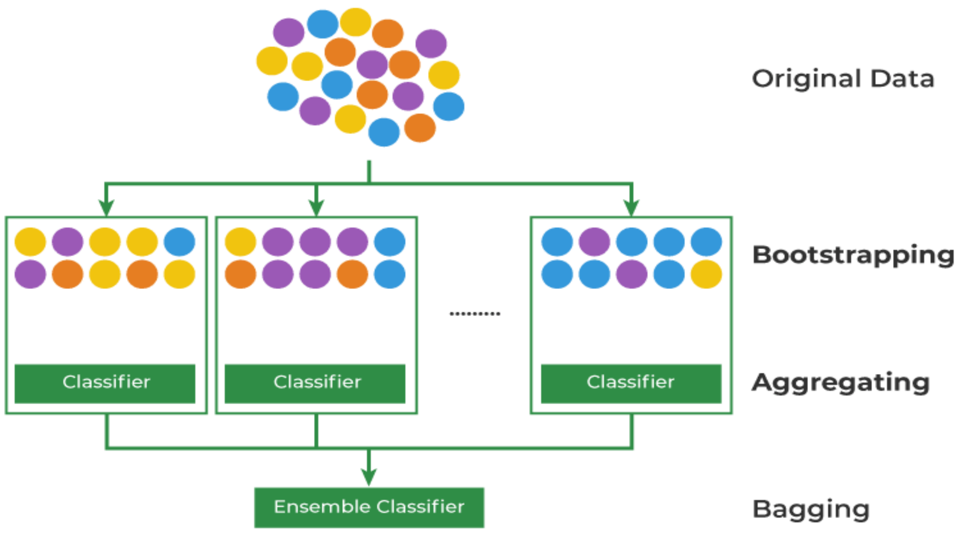
11.5 Bagging Regressor:

Bagging, also known as bootstrap aggregation, is the ensemble learning method that is commonly used to reduce variance within a noisy data set. In bagging, a random sample of data in a training set is selected with replacement—meaning that the individual data points can be chosen more than once.

Bagging involves training multiple models independently on random subsets of the data, and aggregating their predictions through voting or averaging.

Application of the Bagging:

- **IT:** Bagging can also improve the precision and accuracy of IT structures, together with network intrusion detection structures.
- **Environment**: Ensemble techniques, together with Bagging, were carried out inside the area of far-flung sensing. This study indicates how it has been used to map the styles of wetlands inside a coastal landscape
- **Finance:** Bagging has also been leveraged with deep gaining knowledge of models within the finance enterprise, automating essential tasks, along with fraud detection, credit risk reviews, and option pricing issues.
- **Healthcare:** used for various bioinformatics issues, including gene and protein selection, to perceive a selected trait of interest. More significantly, this study mainly delves into its use to expect the onset of diabetes based on various threat predictors.
- **Easier for implementation:** Python libraries, including scikit-examine (sklearn), make it easy to mix the predictions of base beginners or estimators to enhance model performance. Their documentation outlines the available modules you can leverage for your model optimization
- **Variance reduction:** The Bagging can reduce the variance inside a getting to know set of rules which is especially helpful with excessive-dimensional facts, where missing values can result in better conflict, making it more liable to overfitting and stopping correct generalization to new datasets



Advantages of Bagging

Bagging is beneficial because it reduces variance and helps prevent overfitting by combining predictions from multiple models trained on different subsets of the data. This ensemble approach often improves generalization and robustness, especially for complex models.

Disadvantages of Bagging

- It may result in high bias if it is not modeled properly and thus may result in underfitting.
- Since we must use multiple models, it becomes computationally expensive and may not be suitable in various use cases

Steps to implement bagging in python:

Step 1: Multiple subsets are made from the original information set with identical tuples, deciding on observations with replacement.

Step 2: A base model is created on all subsets.

Step 3: Every version is found in parallel with each training set and unbiased.

Step 4: The very last predictions are determined by combining the forecasts from all models.

Computation

Model 1(with all the independent features except CO2):

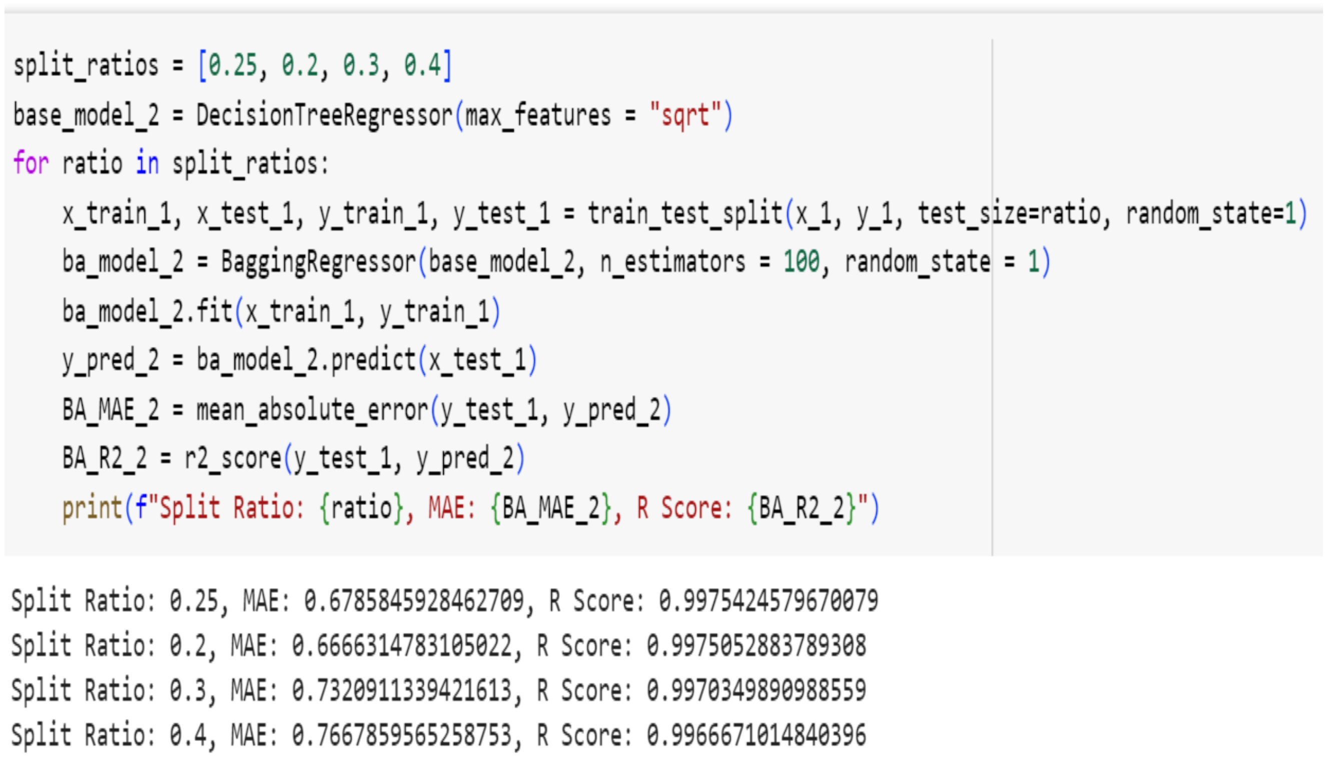
```
from sklearn.ensemble import BaggingRegressor
```

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

```
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: 0.7527527492389645, R Score: 0.9969104941630762
Split Ratio: 0.4, MAE: 0.7813861515410958, R Score: 0.9966128434311846
```

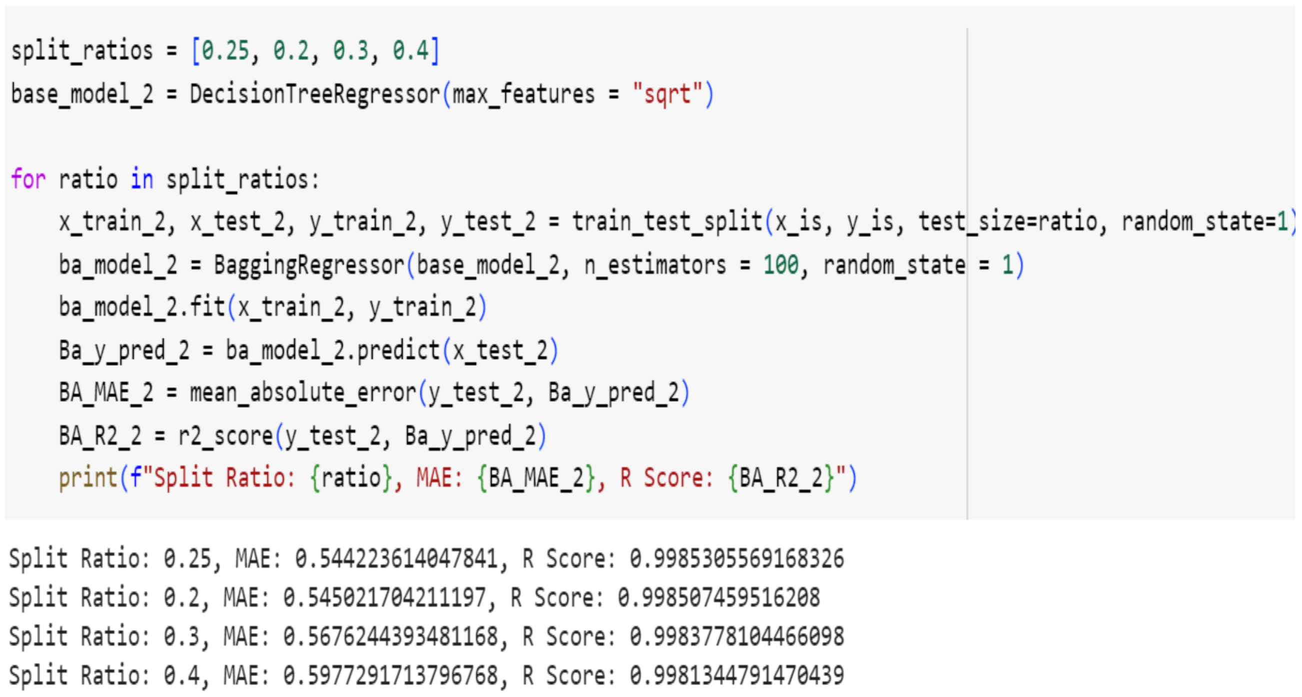
Model 2(without multicollinear variables):



```
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
```

Model 3(without insignificant variables):

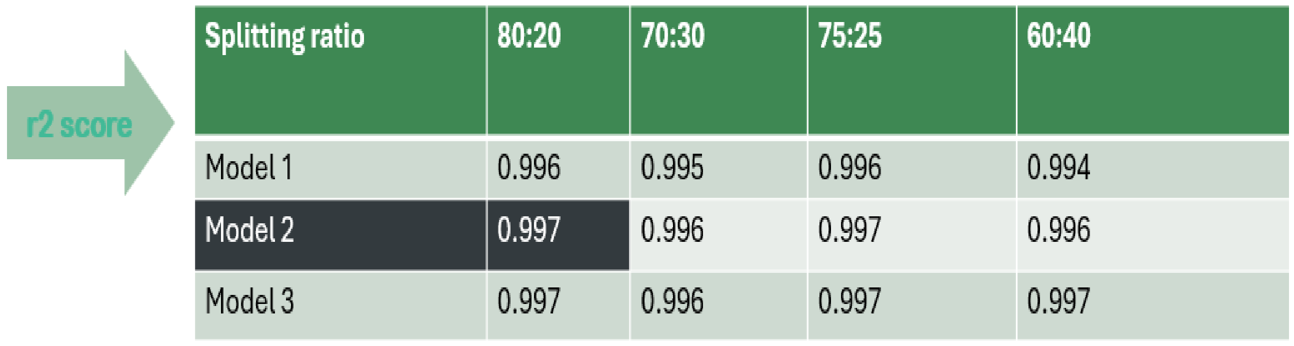


```
split_ratios = [0.25, 0.2, 0.3, 0.4]
base_model_2 = DecisionTreeRegressor(max_features = "sqrt")

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
```

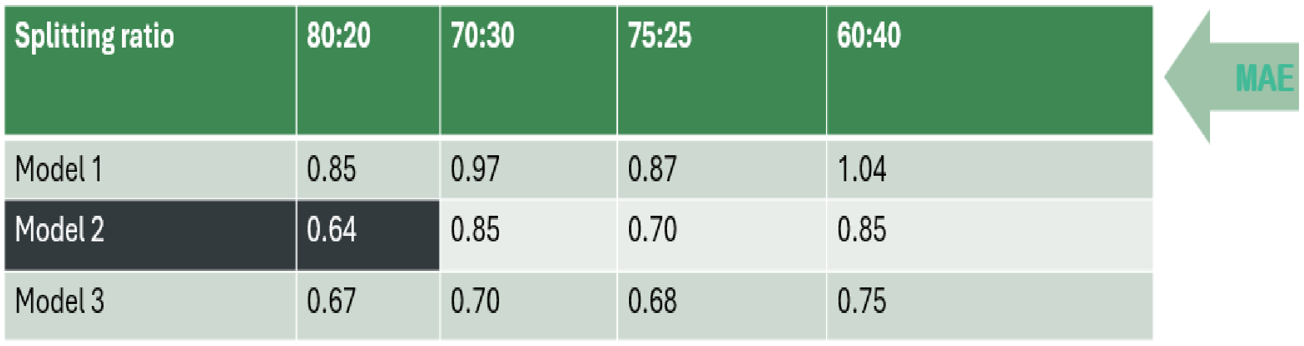
Analysis of a algorithm using R- square and Mean Absolute Error (MAE):

Here, model 2 with R- square 0.997 and MAE value 0.64 with splitting ratio 80:20 is considered to be a best fit.



r2 score

Splitting ratio	80:20	70:30	75:25	60:40
Model 1	0.996	0.995	0.996	0.994
Model 2	0.997	0.996	0.997	0.996
Model 3	0.997	0.996	0.997	0.997



Splitting ratio	80:20	70:30	75:25	60:40
Model 1	0.85	0.97	0.87	1.04
Model 2	0.64	0.85	0.70	0.85
Model 3	0.67	0.70	0.68	0.75

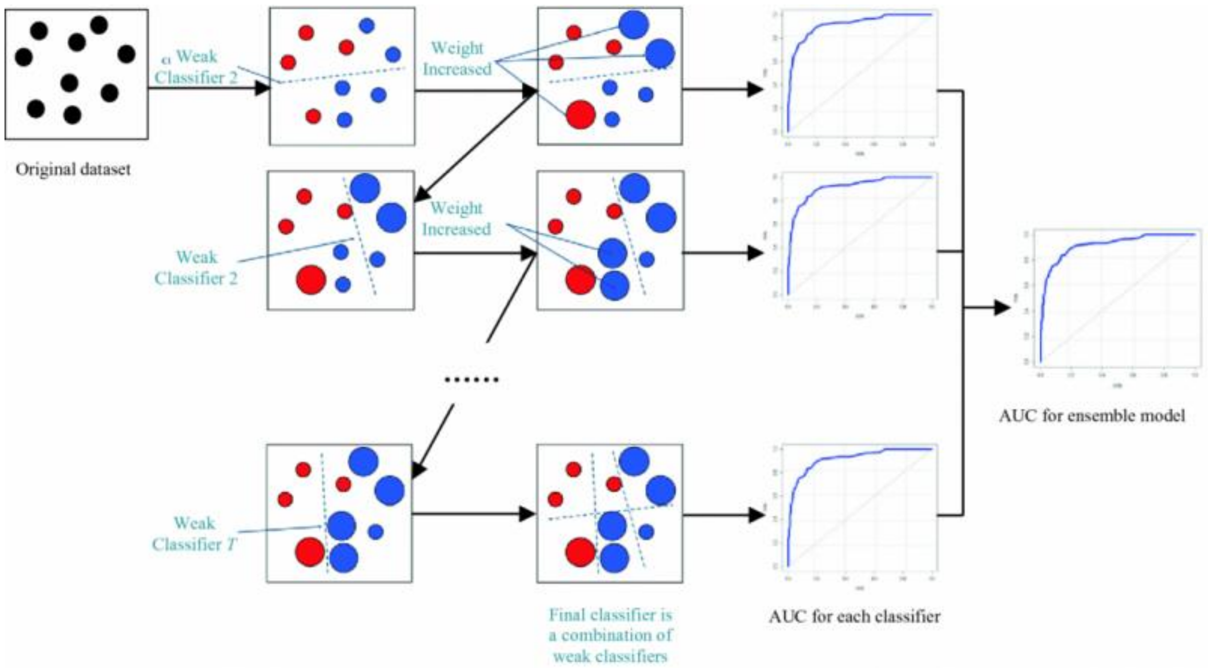
MAE

11.6 Boosting Regressor:

Boosting is an ensemble meta-algorithm for primarily reducing bias, variance. While boosting is not algorithmically constrained, most boosting algorithms consist of iteratively learning weak classifiers with respect to a distribution and adding them to a final strong classifier. When they are added, they are weighted in a way that is related to the weak learners' accuracy. After a weak learner is added, the data weights are readjusted, known as "re-weighting". Misclassified input data gain a higher weight and examples that are classified correctly lose weight. Thus, future weak learners focus more on the examples that previous weak learners misclassified.

Uses

Gradient boosting can be used in the field of learning to rank. The commercial web search engines Yahoo and Yandex use variants of gradient boosting in their machine-learned ranking engines. Gradient boosting is also utilized in High Energy Physics in data analysis. At the Large Hadron Collider (LHC), variants of gradient boosting Deep Neural Networks (DNN) were successful in reproducing the results of non-machine learning methods of analysis on datasets used to discover the Higgs boson.[16] Gradient boosting decision tree was also applied in earth and geological studies – for example quality evaluation of sandstone reservoir.



Advantages of Boosting

- Significant Improvement in Accuracy: Boosting can significantly enhance the accuracy of weak models by iteratively refining their predictions.
- Effective Handling of Class Imbalance: Boosting addresses class imbalance by assigning higher weights to misclassified instances.

Disadvantages of Boosting

- It is sensitive to outliers since every classifier is obliged to fix the errors in the predecessors. Thus, the method is too dependent on outliers. Another disadvantage is that the method is almost impossible to scale up.

Steps to implement gradient boosting in python:

Step-1: Import the necessary libraries

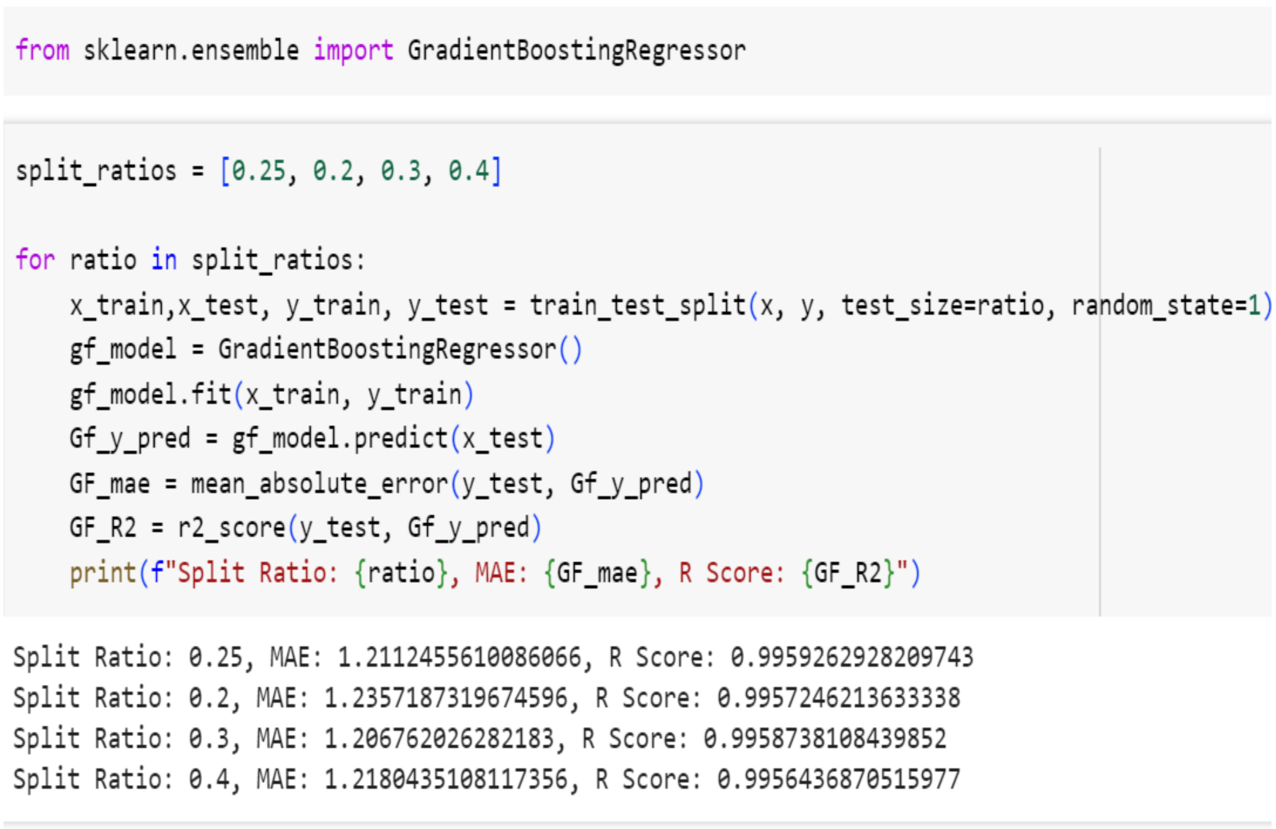
Step-2: Load the digit dataset and split it into train and test.

Step-3: Instantiate Gradient Boosting regressor and fit the model.

Step-4: Predict the test set and compute the accuracy score

Computation

Model 1(with all the features except CO2):

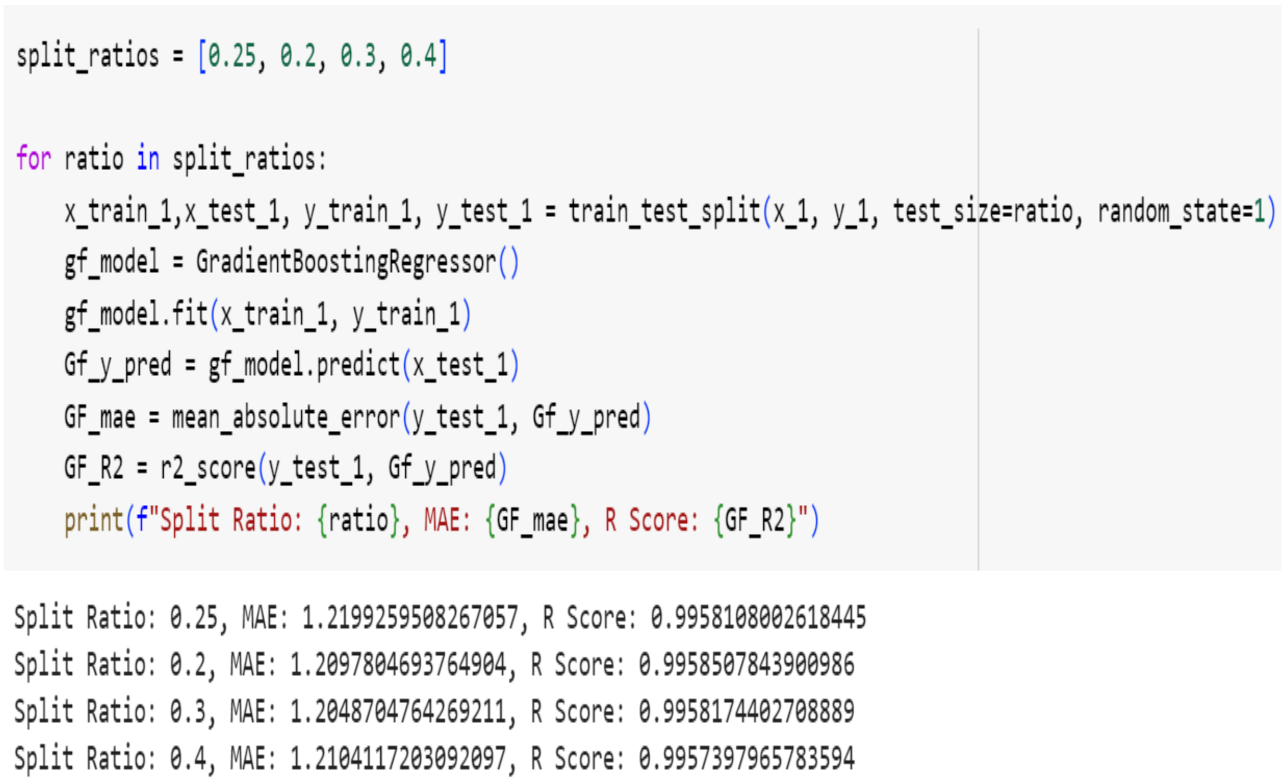


```python
from sklearn.ensemble import GradientBoostingRegressor
```

```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)
    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: 1.2112455610086066, R Score: 0.9959262928209743
Split Ratio: 0.2, MAE: 1.2357187319674596, R Score: 0.9957246213633338
Split Ratio: 0.3, MAE: 1.206762026282183, R Score: 0.9958738108439852
Split Ratio: 0.4, MAE: 1.2180435108117356, R Score: 0.9956436870515977
```

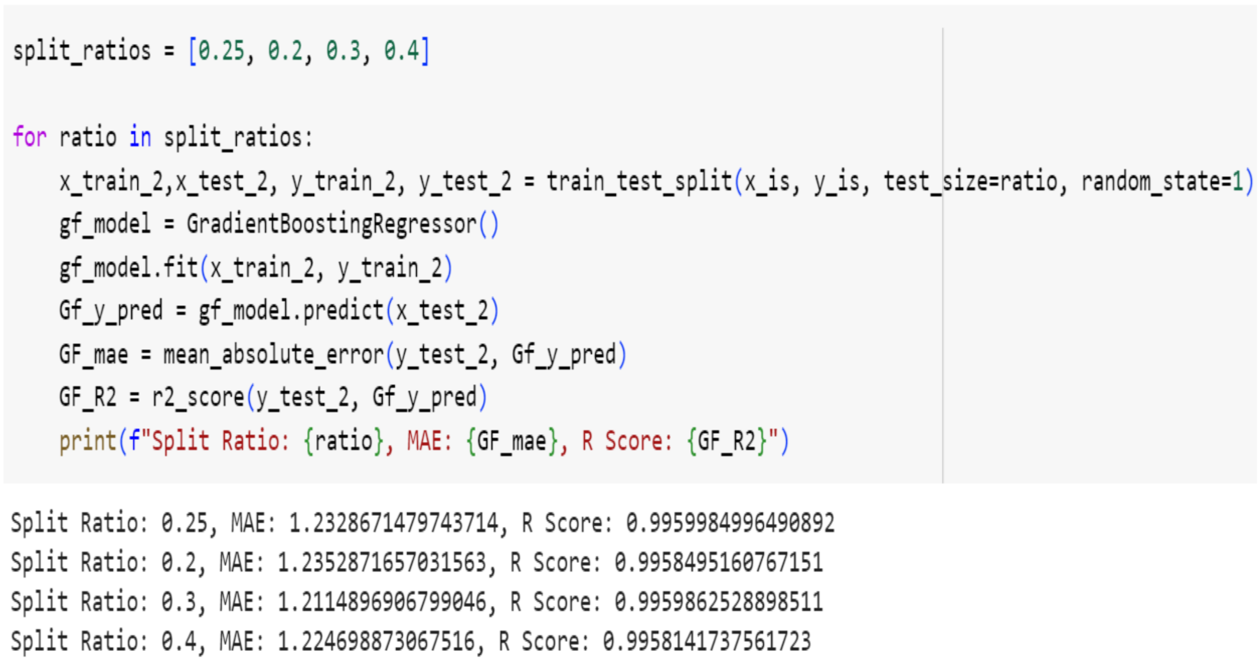
Model 2(without multicollinear variables):



```
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: 1.2199259508267057, R Score: 0.9958108002618445
Split Ratio: 0.2, MAE: 1.2097804693764904, R Score: 0.9958507843900986
Split Ratio: 0.3, MAE: 1.2048704764269211, R Score: 0.9958174402708889
Split Ratio: 0.4, MAE: 1.2104117203092097, R Score: 0.9957397965783594
```

Model 3(without insignificant variable):

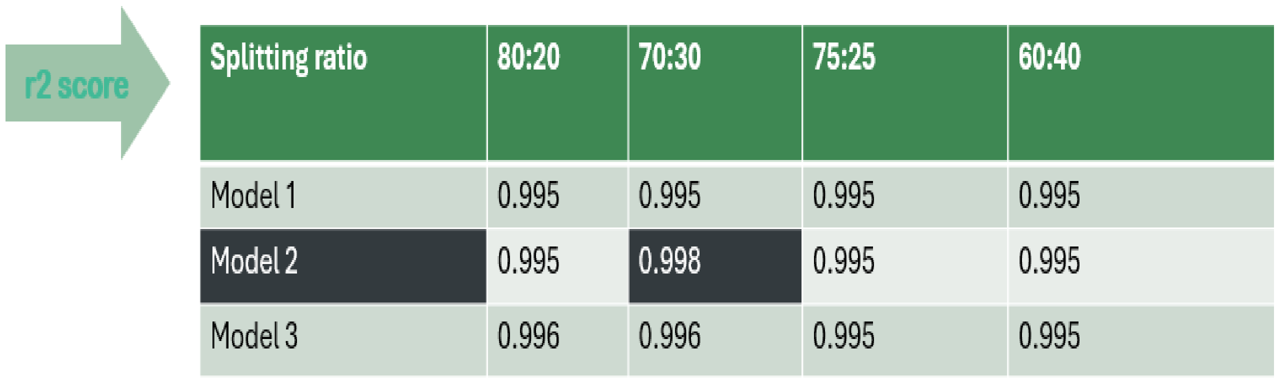


```
split_ratios = [0.25, 0.2, 0.3, 0.4]

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}")
```

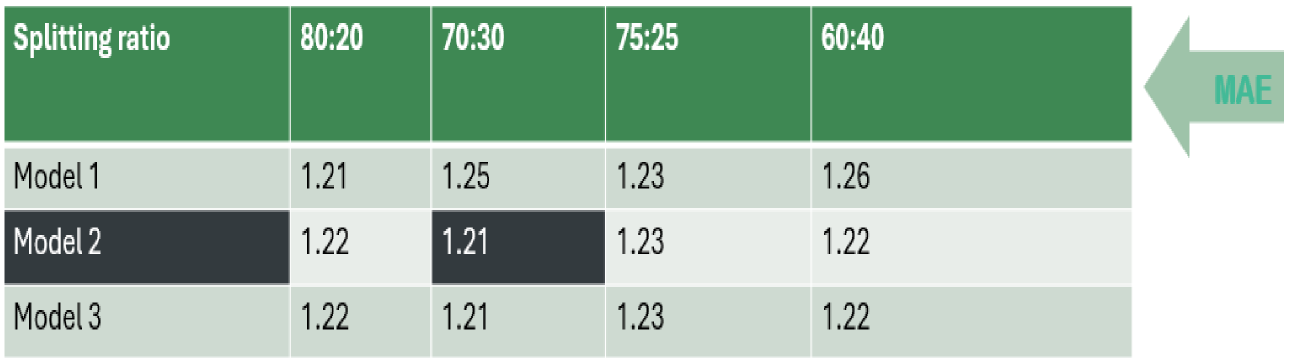
```
Split Ratio: 0.25, MAE: 1.2328671479743714, R Score: 0.9959984996490892
Split Ratio: 0.2, MAE: 1.2352871657031563, R Score: 0.9958495160767151
Split Ratio: 0.3, MAE: 1.2114896906799046, R Score: 0.9959862528898511
Split Ratio: 0.4, MAE: 1.224698873067516, R Score: 0.9958141737561723
```

Analysis of a algorithm using r2 score and Mean Absolute Error (MAE):

Here, model 2 with r2 score 0.998 and MAE value 1.21 with splitting ratio 70:30 is considered to be a best fit.



Splitting ratio	80:20	70:30	75:25	60:40
Model 1	0.995	0.995	0.995	0.995
Model 2	0.995	0.998	0.995	0.995
Model 3	0.996	0.996	0.995	0.995



Splitting ratio	80:20	70:30	75:25	60:40
Model 1	1.21	1.25	1.23	1.26
Model 2	1.22	1.21	1.23	1.22
Model 3	1.22	1.21	1.23	1.22

11.7 Support Vector Machines:

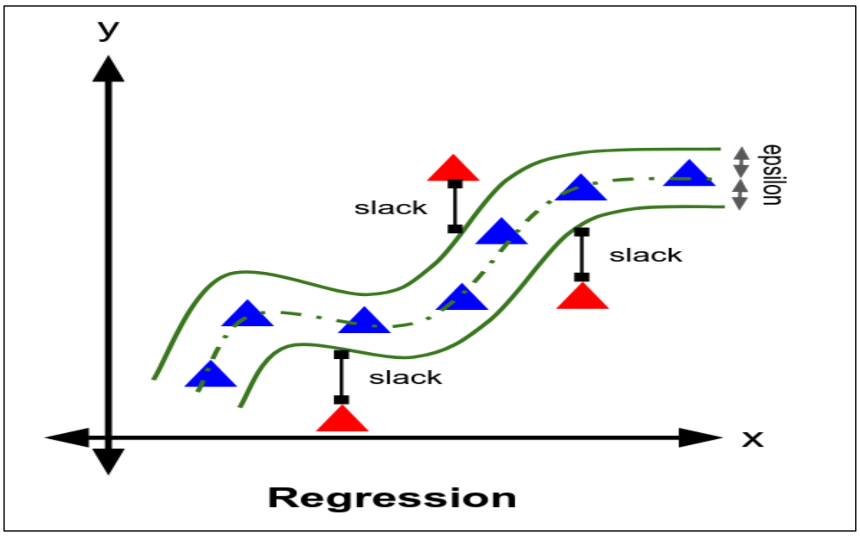
Support vector regression (SVR) is a type of support vector machine (SVM) that is used for regression tasks. It tries to find a function that best predicts the continuous output value for a given input value.

SVR can use both linear and non-linear kernels. A linear kernel is a simple dot product between two input vectors, while a non-linear kernel is a more complex function that can capture more intricate patterns in the data. The choice of kernel depends on the data's characteristics and the task's complexity.

Applications

SVMs can be used to solve various real-world problems:

- SVMs are helpful in text and hypertext categorization, as their application can significantly reduce the need for labeled training instances in both the standard inductive and transductive settings. Some methods for shallow semantic parsing are based on support vector machines.
- Classification of images can also be performed using SVMs. Experimental results show that SVMs achieve significantly higher search accuracy than traditional query refinement schemes after just three to four rounds of relevance feedback. This is also true for image segmentation systems, including those using a modified version SVM that uses the privileged approach as suggested by Vapnik.
- Classification of satellite data like SAR data using supervised SVM. Hand-written characters can be recognized using SVM.
- The SVM algorithm has been widely applied in the biological and other sciences. They have been used to classify proteins with up to 90% of the compounds classified correctly. Permutation tests based on SVM weights have been suggested as a mechanism for interpretation of SVM models. Support vector machine weights have also been used to interpret SVM models in the past. Posthoc interpretation of support vector machine models in order to identify features used by the model to make predictions is a relatively new area of research with special significance in the biological sciences.



Advantages

Support Vector Machine (SVM) has many advantages, including:

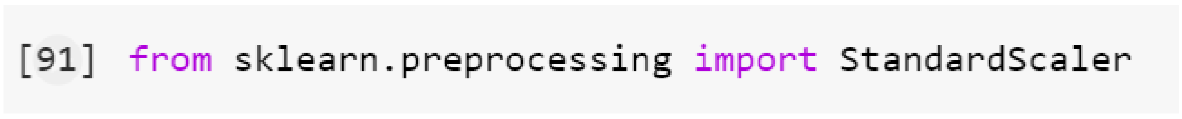
- Linear data: SVM works well when data is linear.
- High dimensions: SVM is more effective in high dimensions, even when the number of dimensions is greater than the number of samples.
- Kernel trick: SVM can solve complex problems using the kernel trick

SVM Disadvantages

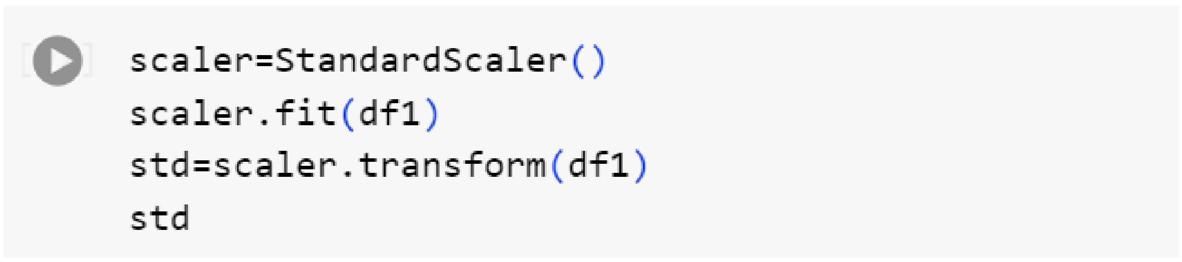
Long training time for large datasets. Difficult to understand and interpret the final model, variable weights and individual impact. Since the final model is not so easy to see, we can not do small calibrations to the model hence its tough to incorporate our business logic.

For support vector regressor as we did not get the output for the original data we have **standardized the data** and then performed the SVM task.

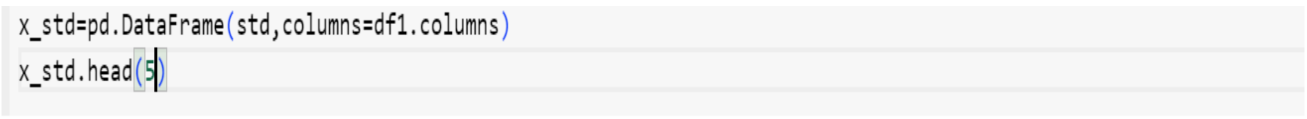
Standardizing the data:



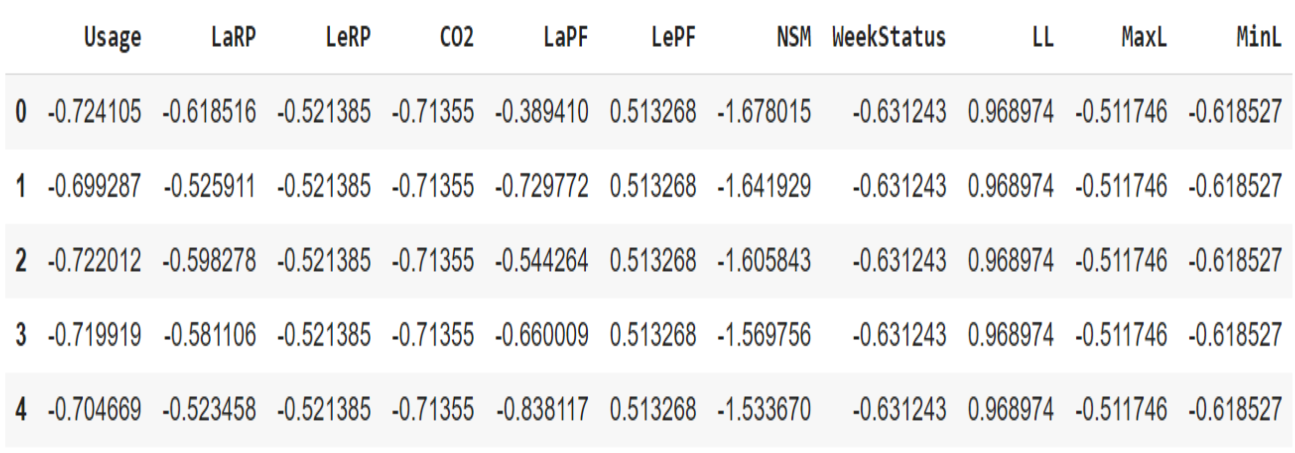
```
[91] from sklearn.preprocessing import StandardScaler
```



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

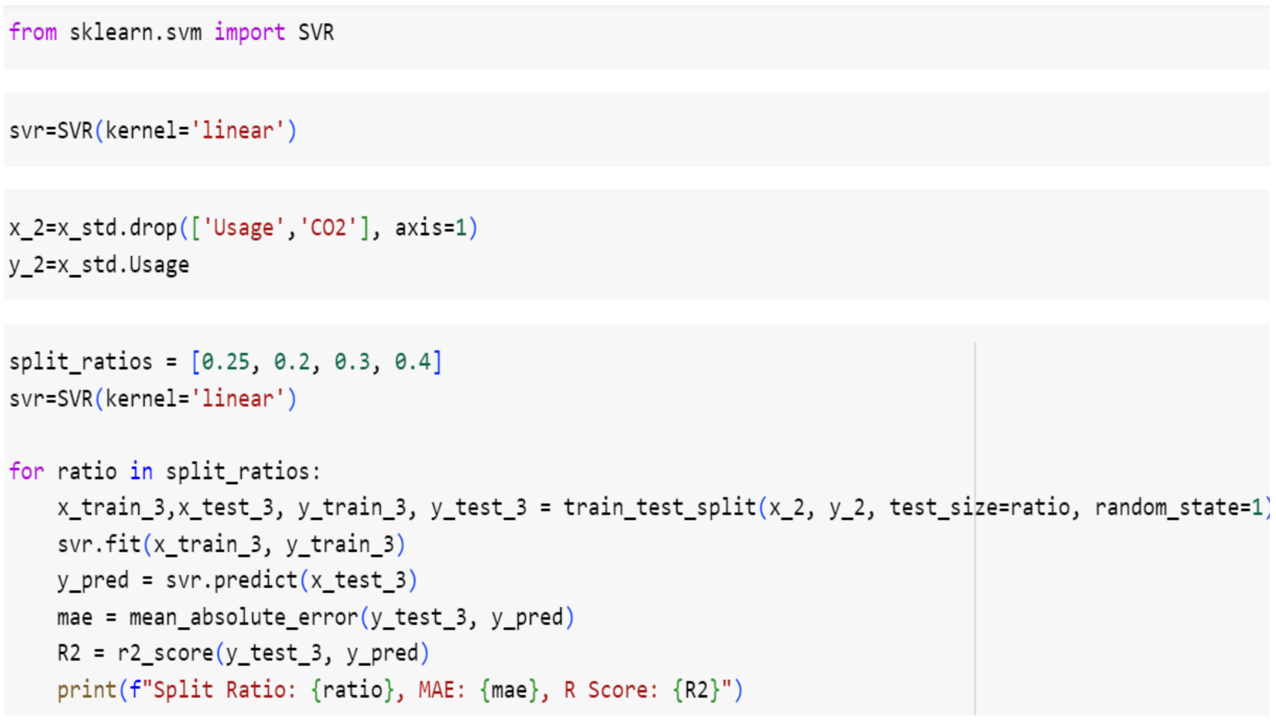


```
x_std=pd.DataFrame(std,columns=df1.columns)
x_std.head(5)
```



	Usage	LaRP	LeRP	CO2	LaPF	LePF	NSM	WeekStatus	LL	MaxL	MinL
0	-0.724105	-0.618516	-0.521385	-0.71355	-0.389410	0.513268	-1.678015	-0.631243	0.968974	-0.511746	-0.618527
1	-0.699287	-0.525911	-0.521385	-0.71355	-0.729772	0.513268	-1.641929	-0.631243	0.968974	-0.511746	-0.618527
2	-0.722012	-0.598278	-0.521385	-0.71355	-0.544264	0.513268	-1.605843	-0.631243	0.968974	-0.511746	-0.618527
3	-0.719919	-0.581106	-0.521385	-0.71355	-0.660009	0.513268	-1.569756	-0.631243	0.968974	-0.511746	-0.618527
4	-0.704669	-0.523458	-0.521385	-0.71355	-0.838117	0.513268	-1.533670	-0.631243	0.968974	-0.511746	-0.618527

Support vector regressor:

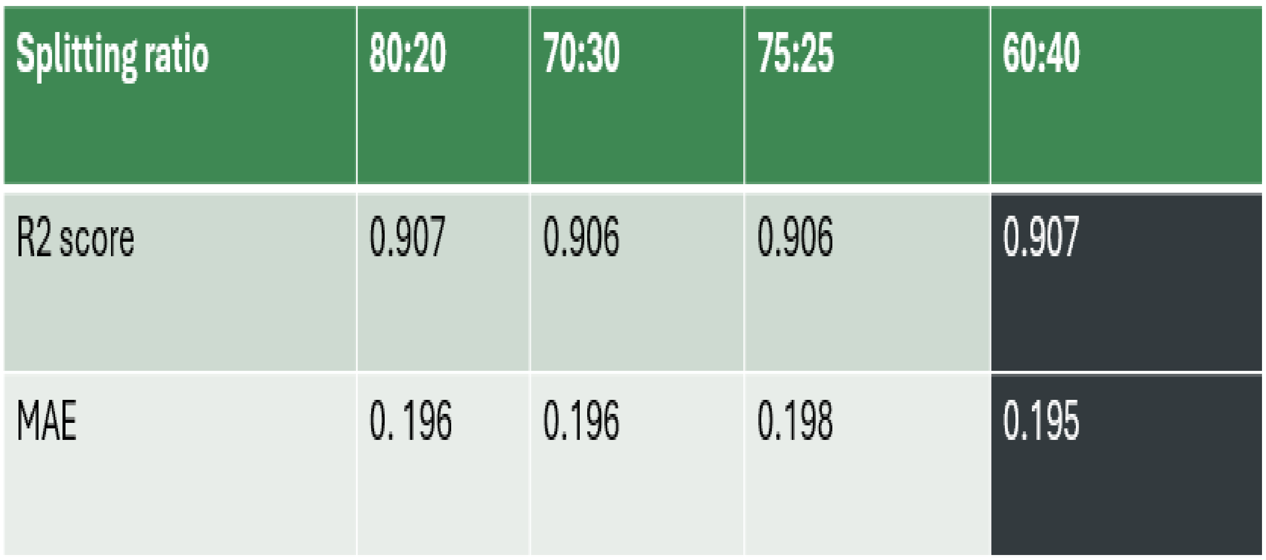


```
from sklearn.svm import SVR
```

```
svr=SVR(kernel='linear')
```

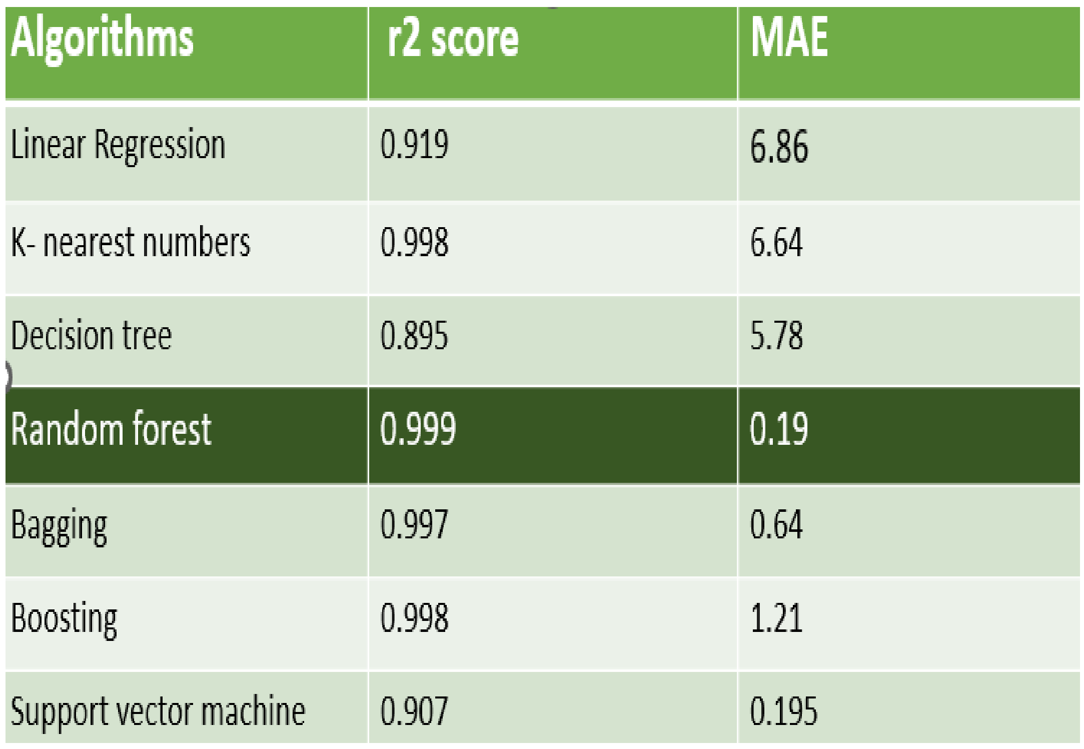
```
x_2=x_std.drop(['Usage','CO2'], axis=1)
y_2=x_std.Usage
```

```
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}")
```



Splitting ratio	80:20	70:30	75:25	60:40
R2 score	0.907	0.906	0.906	0.907
MAE	0. 196	0.196	0.198	0.195

12 Comparison of Algorithms:



Algorithms	r2 score	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

13 Conclusion:

- One of the most important issues for energy management and optimization in the steel industry is a precise long-term prediction of energy consumption. The data analysis shows thought-provoking outcomes in the exploratory analysis.
- The purpose of this research is to determine the best performing machine learning techniques to predict the energy consumption in the steel industry.
- The findings indicate that the RF model improves MAE and r2 score of predictions in consideration to other regression models considered in this research.

14 References:

Dataset

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

Steel industry

https://en.wikipedia.org/wiki/Steel

Steel industry in South Korea

https://www.kosa.or.kr/sub/eng/introduction/sub02.jsp

Literature review

- 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-

- 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-

Machine learning algorithms

- https://www.geeksforgeeks.org/support-vector-regression-svr-using-linear-and-non-linear-kernels-in-scikit-learn/
- https://en.wikipedia.org/wiki/Boosting_(machine_learning)
- https://www.geeksforgeeks.org/random-forest-regression-in-python/
- https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm#:~:text=In%20k%2DNN%20regression%2C%20the%20k%2DNN%20algorithm%20is,the%20inverse%20of%20their%20distance.
- https://www.ibm.com/topics/bagging#:~:text=IBM-,What%20is%20bagging%3F,be%20chosen%20more%20than%20once.
- https://en.wikipedia.org/wiki/Decision_tree_learning
- https://www.investopedia.com/terms/m/mlr.asp