Muhammad Randa Yandika

Steel Industry Energy Consumption Prediction

linkedin.com/in/muhammad-randa-yandika

github.com/randayandika/

OUTLINE

Use Case Summary

Objective

- Get an insight into how much energy used and the factors that influence it.
- Create models to predict energy consumption with machine learning techniques.
- Find best model for predict energy consumption

Business Benefit

- Prediction result can use to improve energy efficiency and reduce costs in the steel industry in future

Outcome

- Get to know how energy used and the factors that influence it
- Making machine learning model to predict energy consumption
- Best Model to predict energy consumption

Data Understanding

About Data

Source:
https://archive.ics.uci.edu/ml/datasets/Steel+Industry+Energy+Consumption+Dataset
Number of Instances: 35040
Number of Attributes: 11

Attribute Information:

Data Variables	Type	Measurement
Industry Energy Consumption	Continuous	kWh
Lagging Current Reactive Power	Continuous	kVarh
Leading Current Reactive Power	Continuous	kVarh
tCO2(CO2)	Continuous	Ppm
Lagging Current Power Factor	Continuous	%
Leading Current Power Factor	Continuous	%
Number of Seconds from Midnight	Continuous	S
Week status	Categorical	(Weekend (0) or a Weekday(1))
Day of week	Categorical	Sunday, Monday …. Saturday
Load Type	Categorical	Light Load, Medium Load, Maximum Load

Data Information & Statistic Numerical

- From this information, we know this dataset have 11 columns with 35048 entries and data type from each column.
- In statistic data, we can get information like count, mean, std, min, max, etc. from each column in dataset

```
df.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
```

```
df.describe()
```

	Usage_kWh	Lagging_Current_Reactive.Power_kVarh	Leading_Current_Reactive_Power_kVarh	CO2(tCO2)	La
count	35040.000000	35040.000000	35040.000000	35040.000000	35
mean	27.386892	13.035384	3.870949	0.011524	80
std	33.444380	16.306000	7.424463	0.016151	18
min	0.000000	0.000000	0.000000	0.000000	0.
25%	3.200000	2.300000	0.000000	0.000000	63
50%	4.570000	5.000000	0.000000	0.000000	87
75%	51.237500	22.640000	2.090000	0.020000	99
max	157.180000	96.910000	27.760000	0.070000	10

Data Preparation

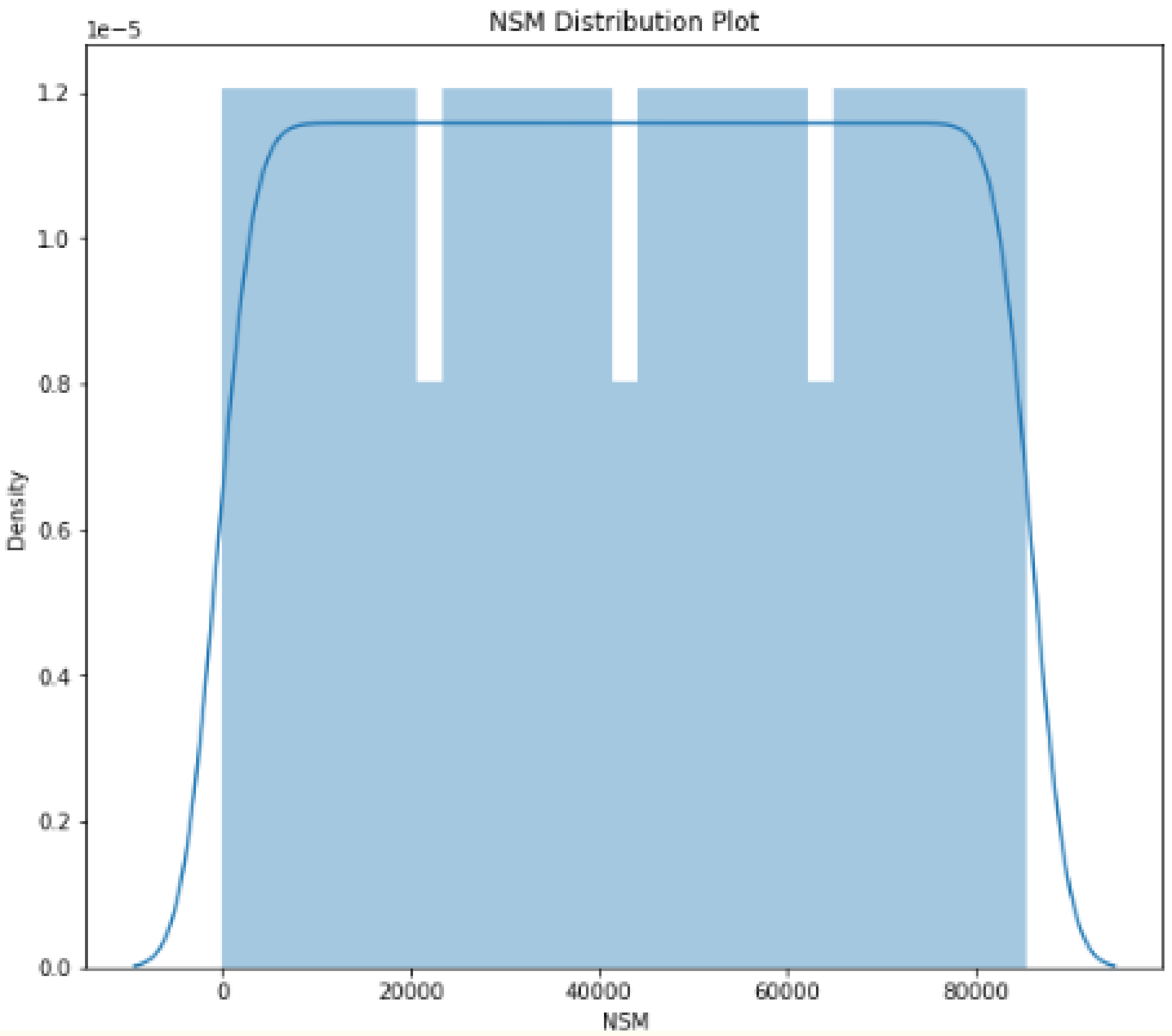
Packages and Libraries

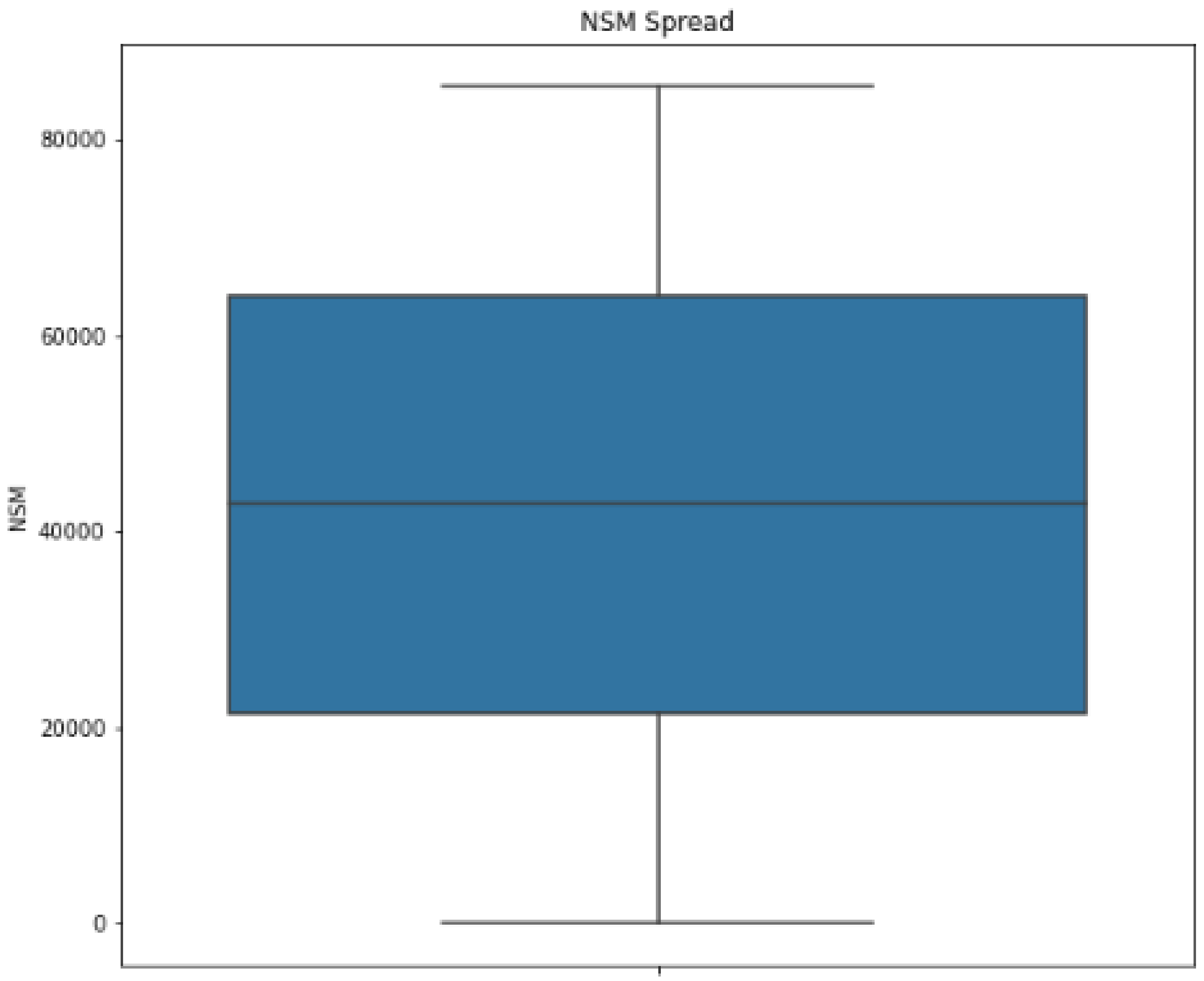
- This is packages and libraries we use in this project

```
import numpy as np
import pandas as pd
import plotly.express as px
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_percentage_error
```

Exploratory Data Analysis
&
Data Visualization

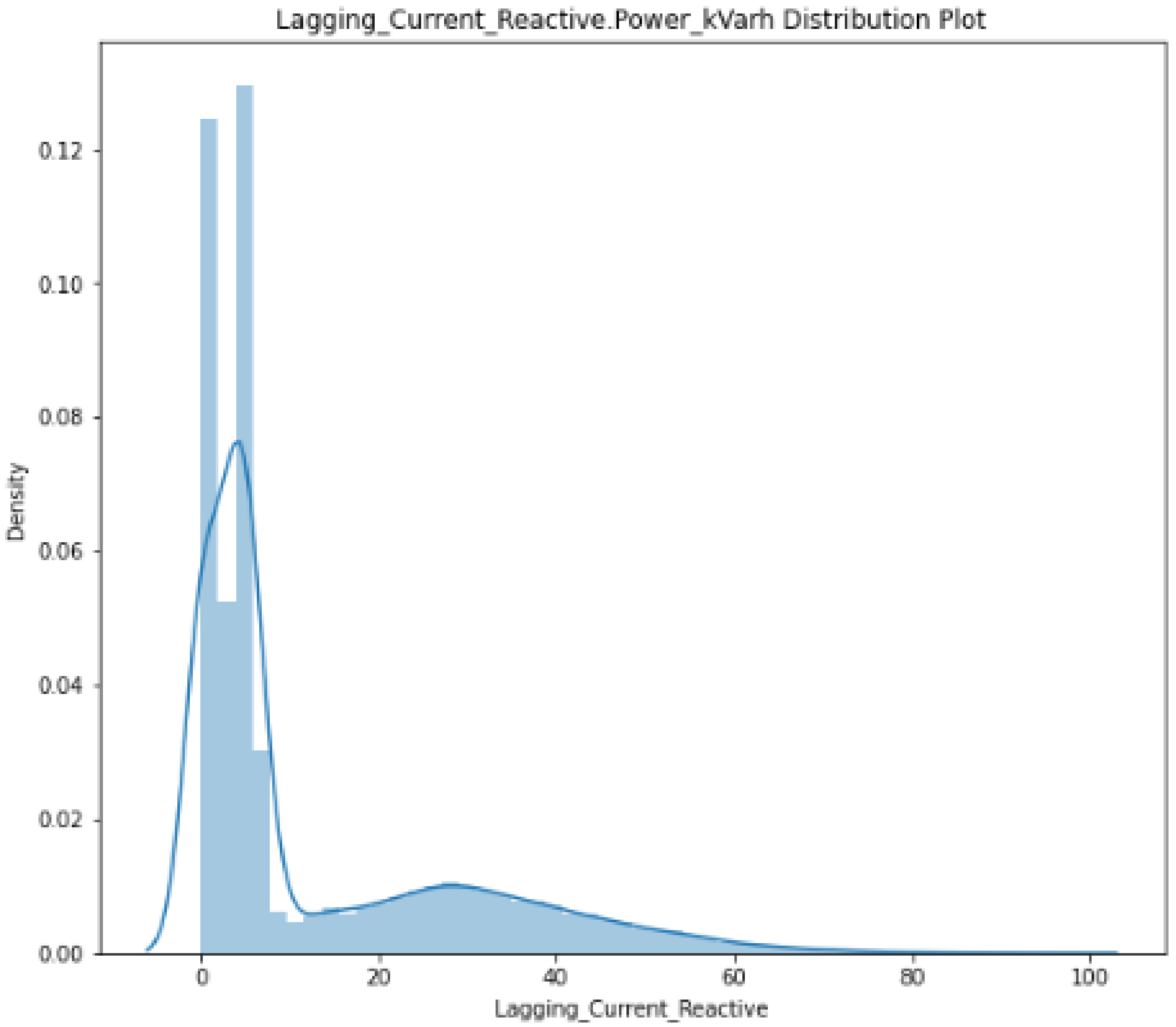
Distribution and Box Plot

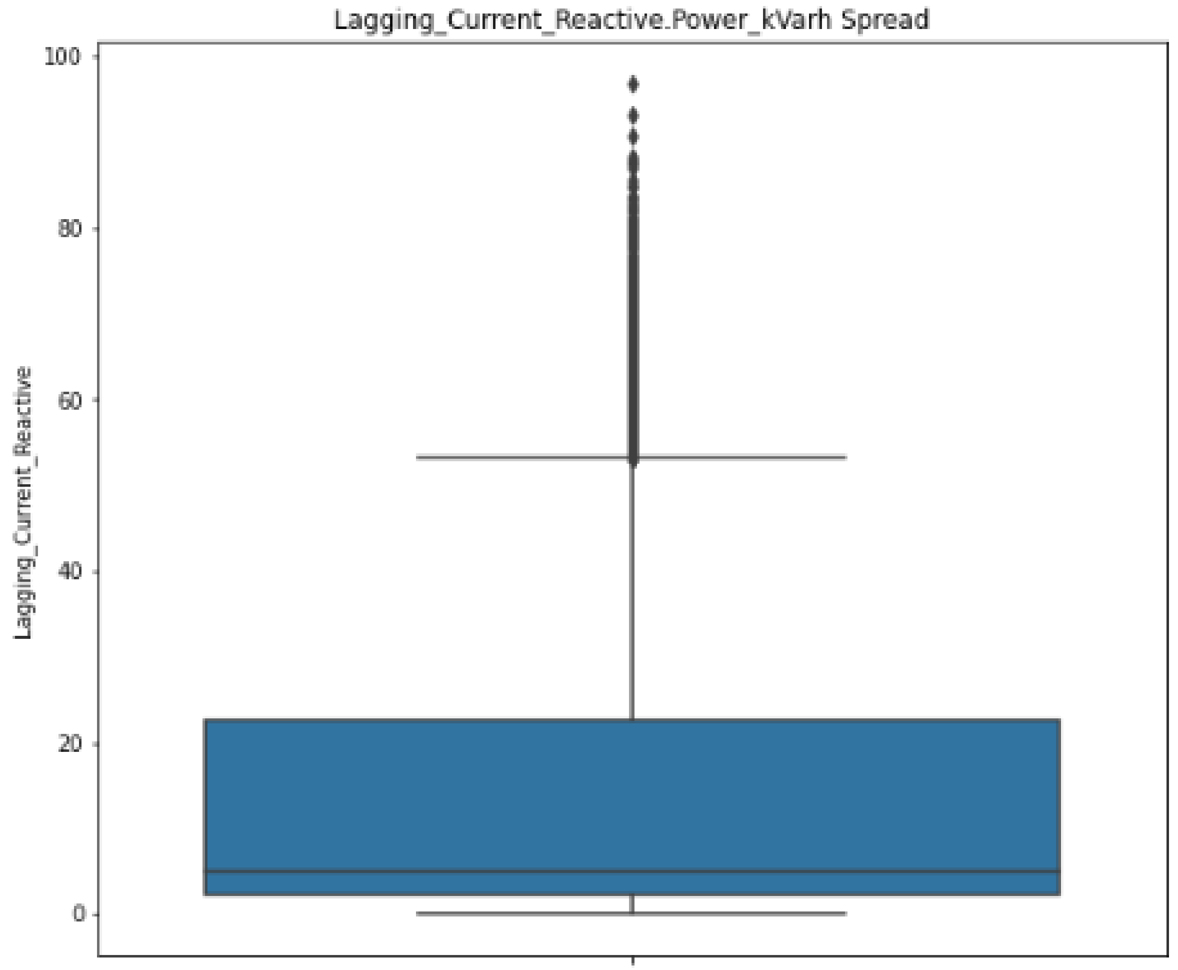




- Based on the graph above, we can see that 'NSM' are have stable density and don't have any ouliers

Distribution and Box Plot

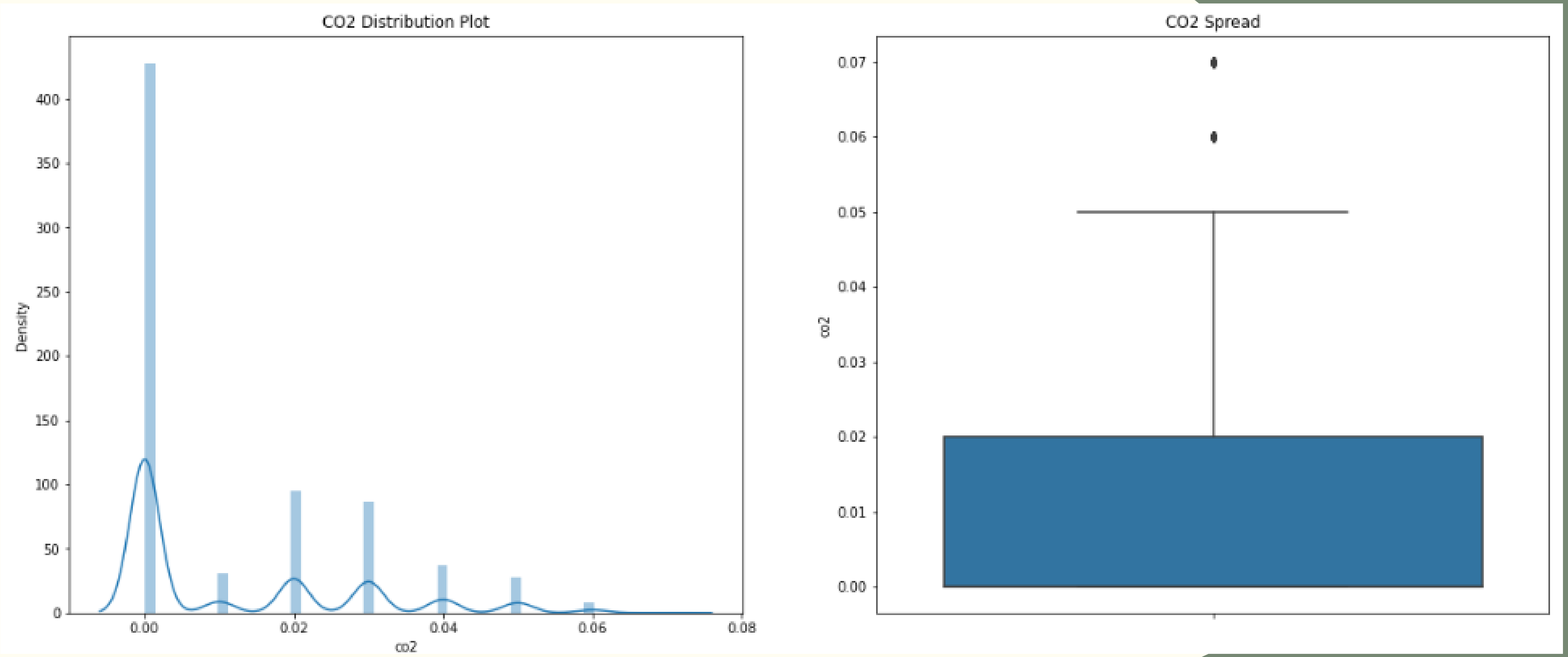




- Based on the graph above, we can see that feature are mostly or have the highest density at values around 0-10 kVarh, and have ouliers

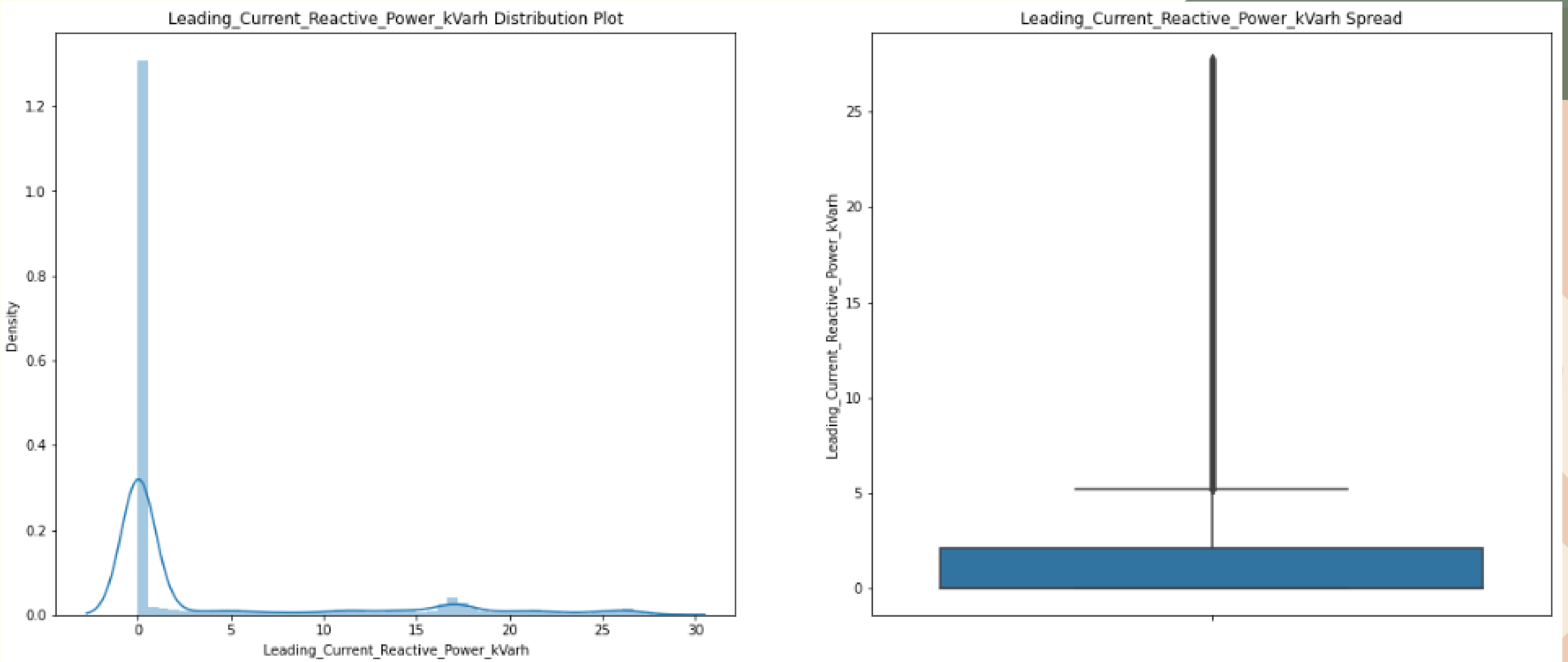
Distribution and Box Plot

- Based on the graph above, we can see that Co2 have the highest density at values around 0 Ppm, and have 2 outliers



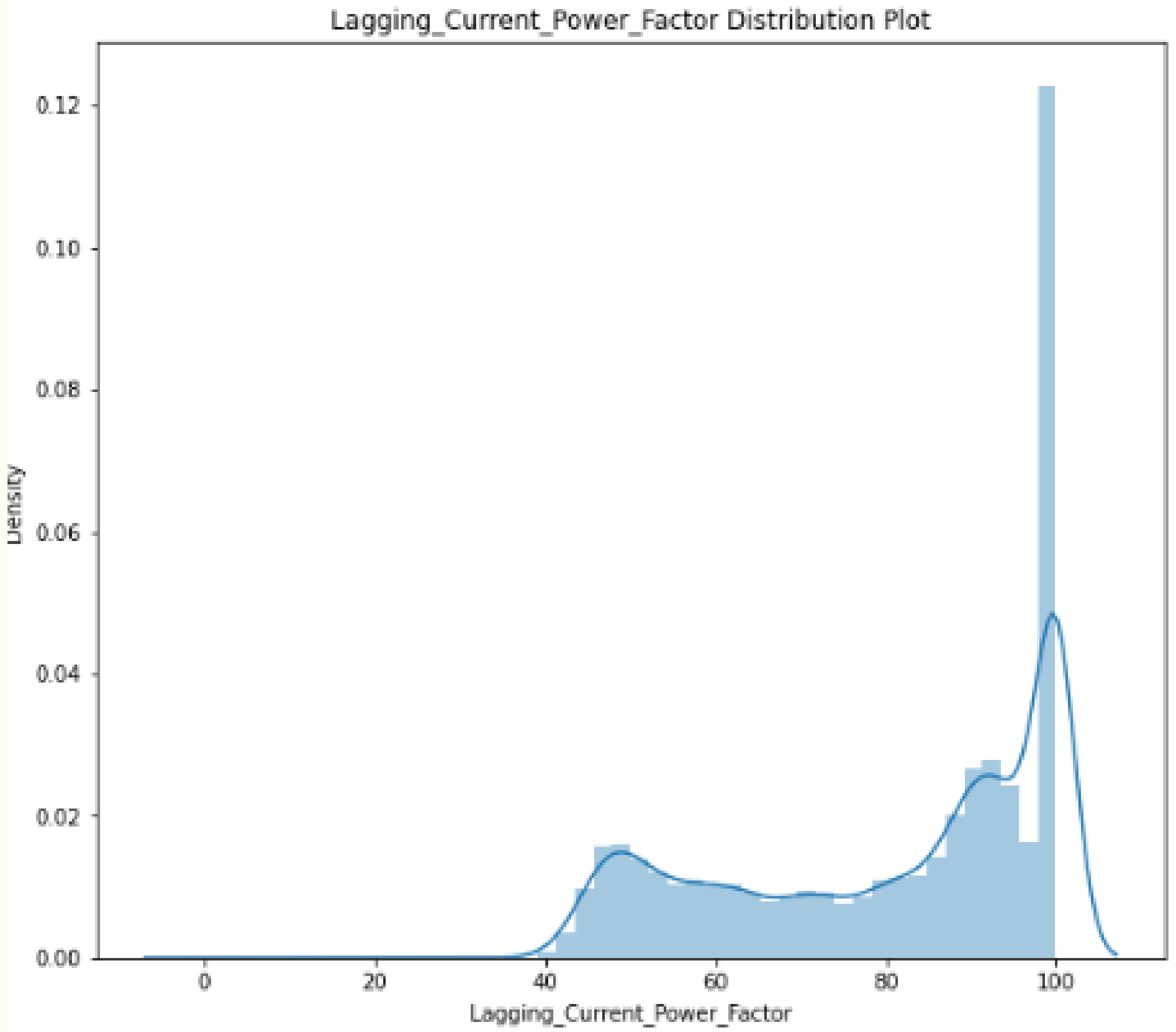
Distribution and Box Plot

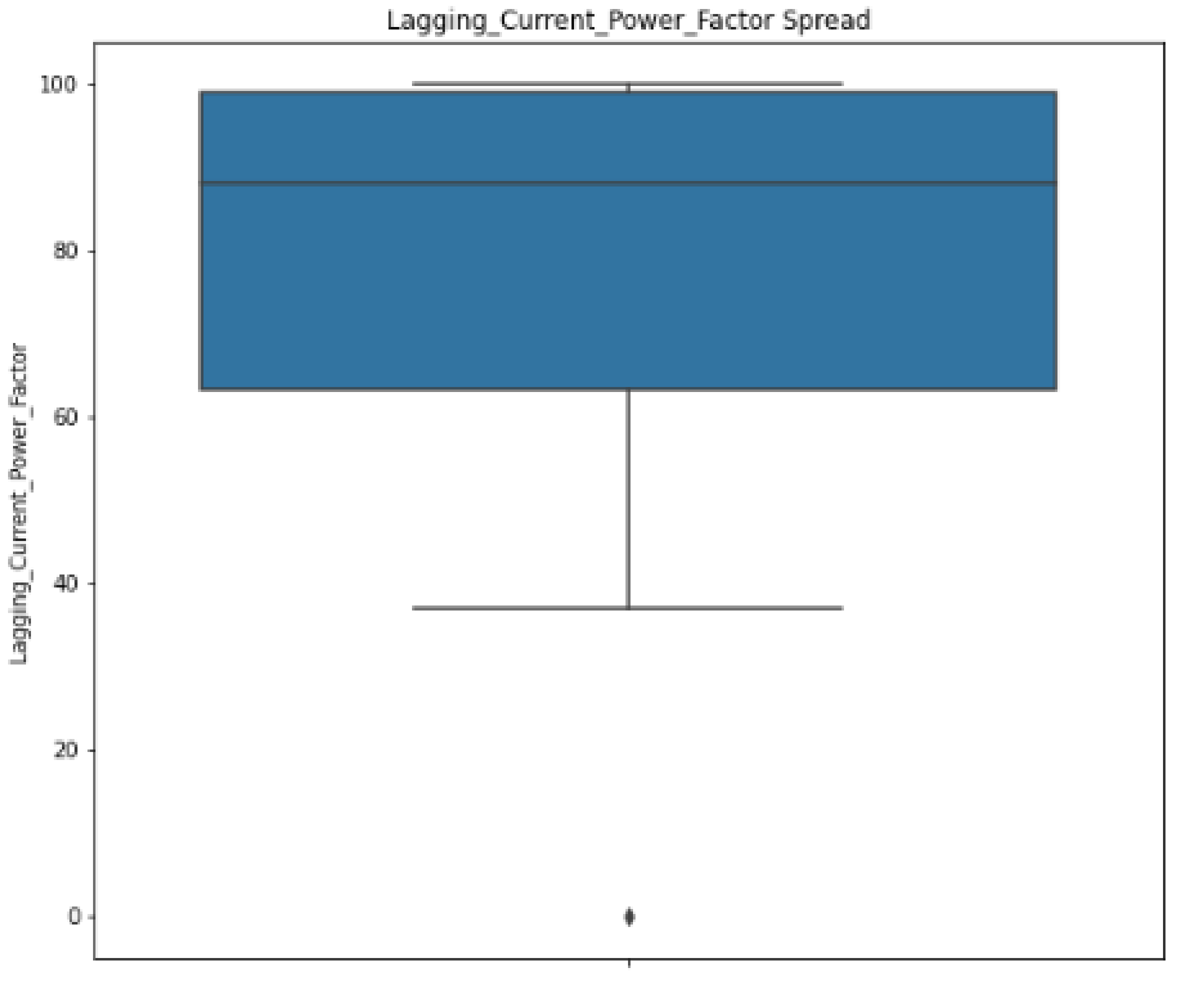
- Based on the graph above, we can see that feature have the highest density at values in 0 kVarh, and have ouliers



Distribution and Box Plot

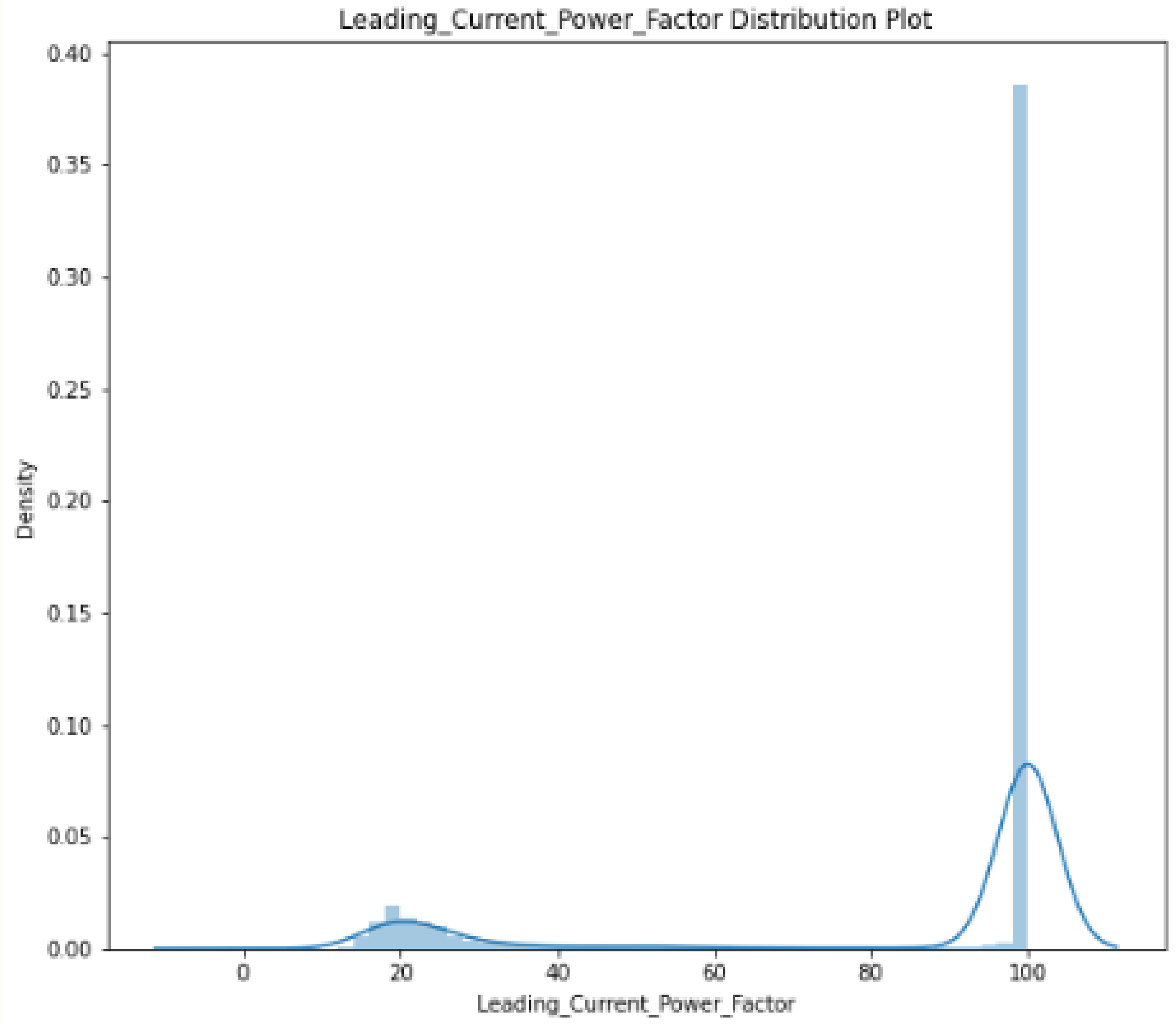
- Based on the graph above, we can see that feature are mostly or have the highest density at values 80-100% and have ouliers





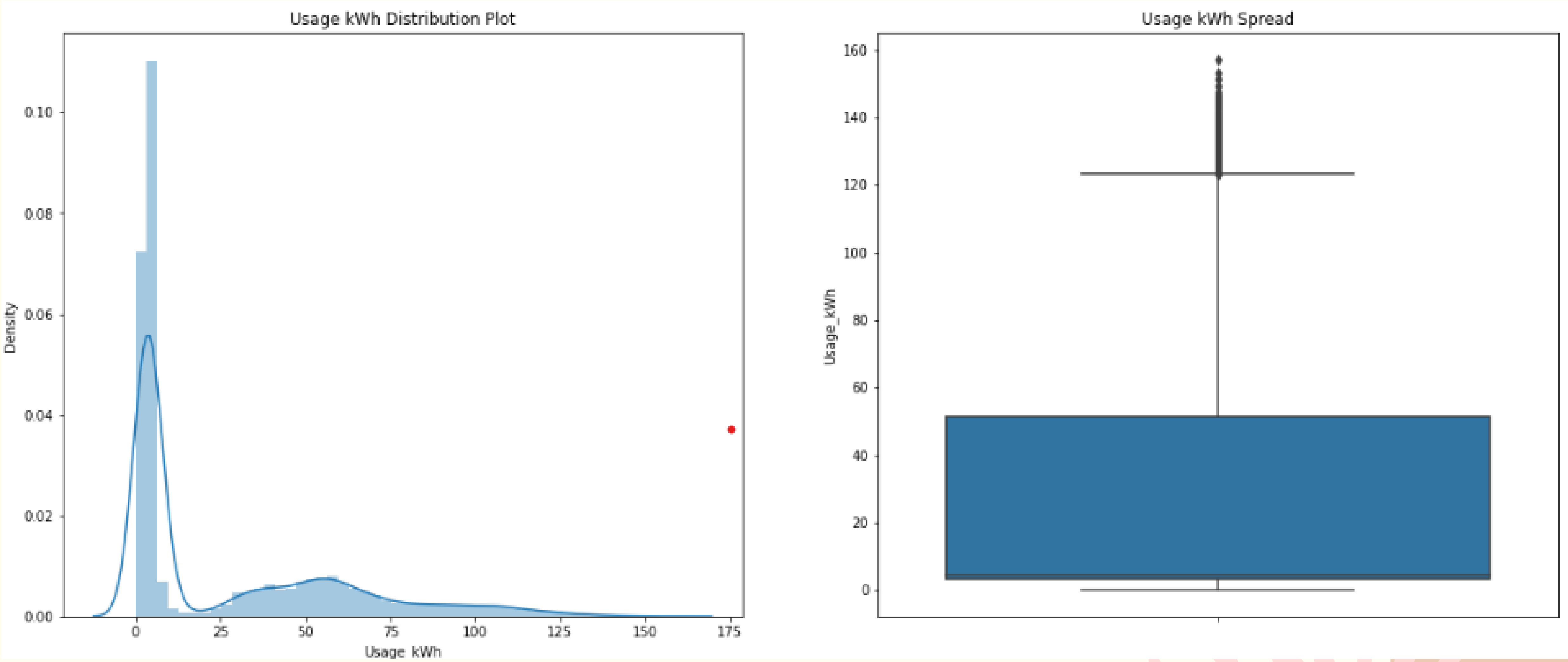
Distribution and Box Plot

- Based on the graph above, we can see that feature have the highest density at values 100%, and have ouliers



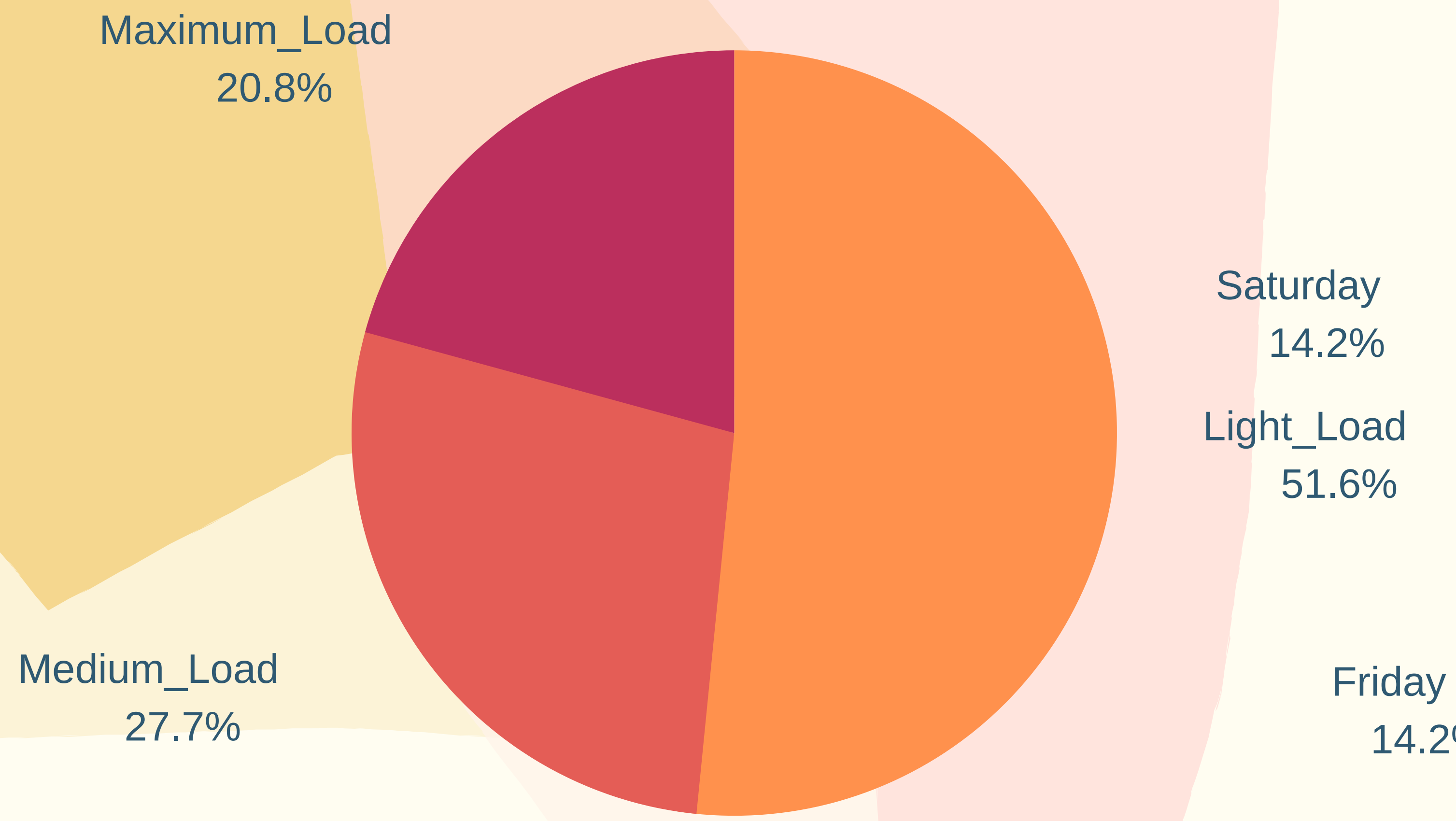
Distribution and Box Plot

- Based on the graph above, we can see that label feature are mostly or have the highest density at values around 0-10 kWh and have ouliers



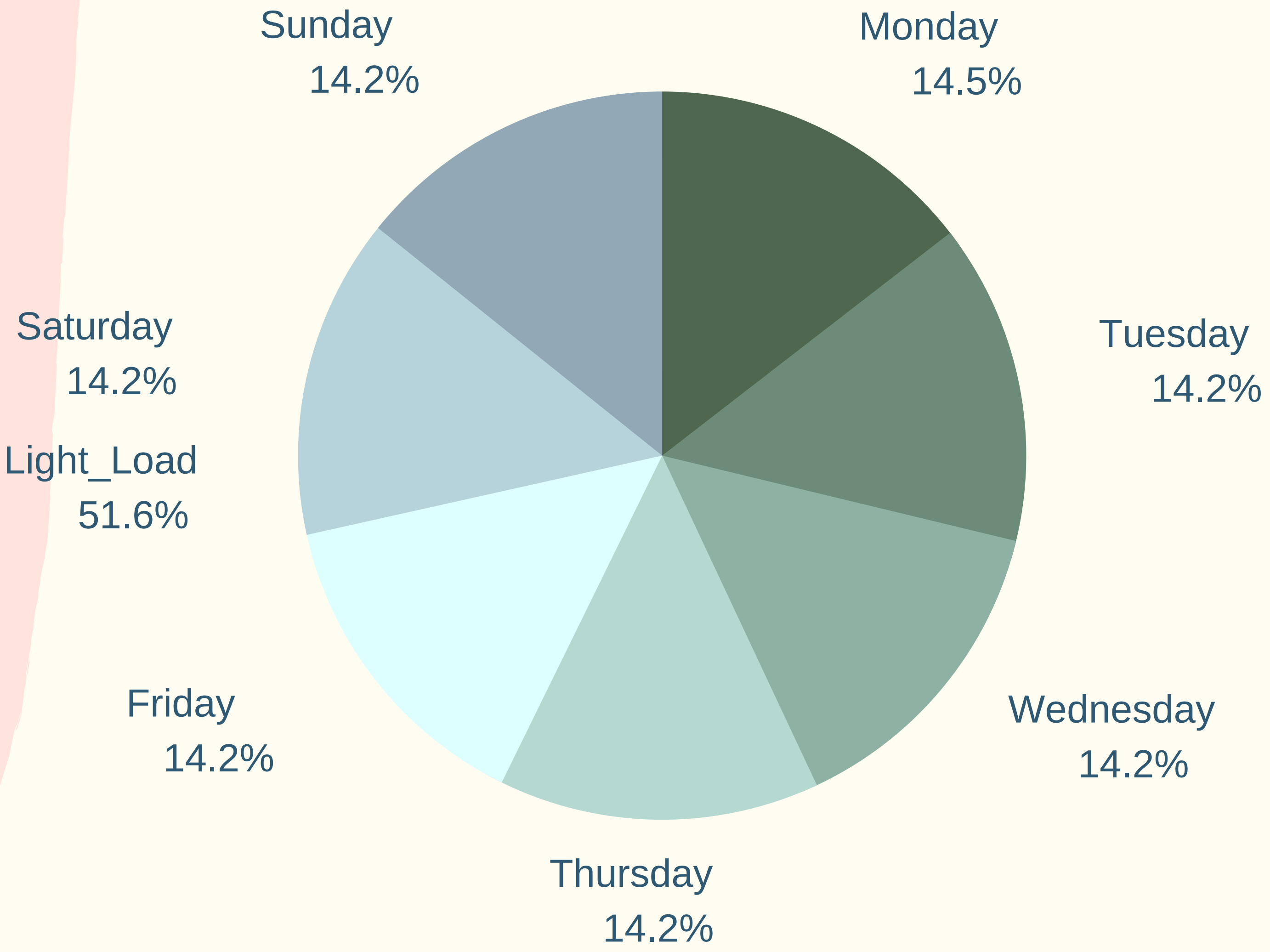
CATEGORICAL FEATURE PIE CHART

REQUIRED CAR PARKING SPACE



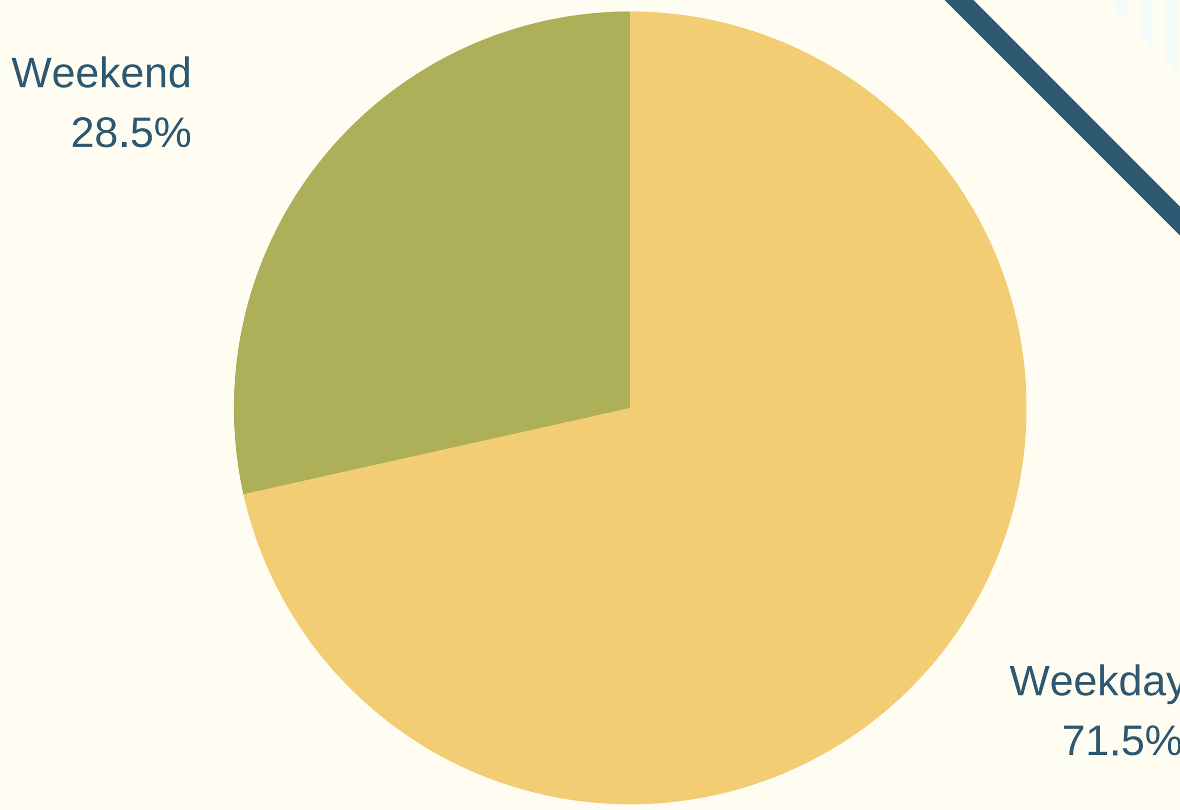
Light_Load	18072
Medium_Load	9696
Maximum_Load	7272

REQUIRED CAR PARKING SPACE



Monday	5088	Friday	4992
Tuesday	4992	Saturday	4992
Wednesday	4992	Sunday	4992
Thursday	4992		

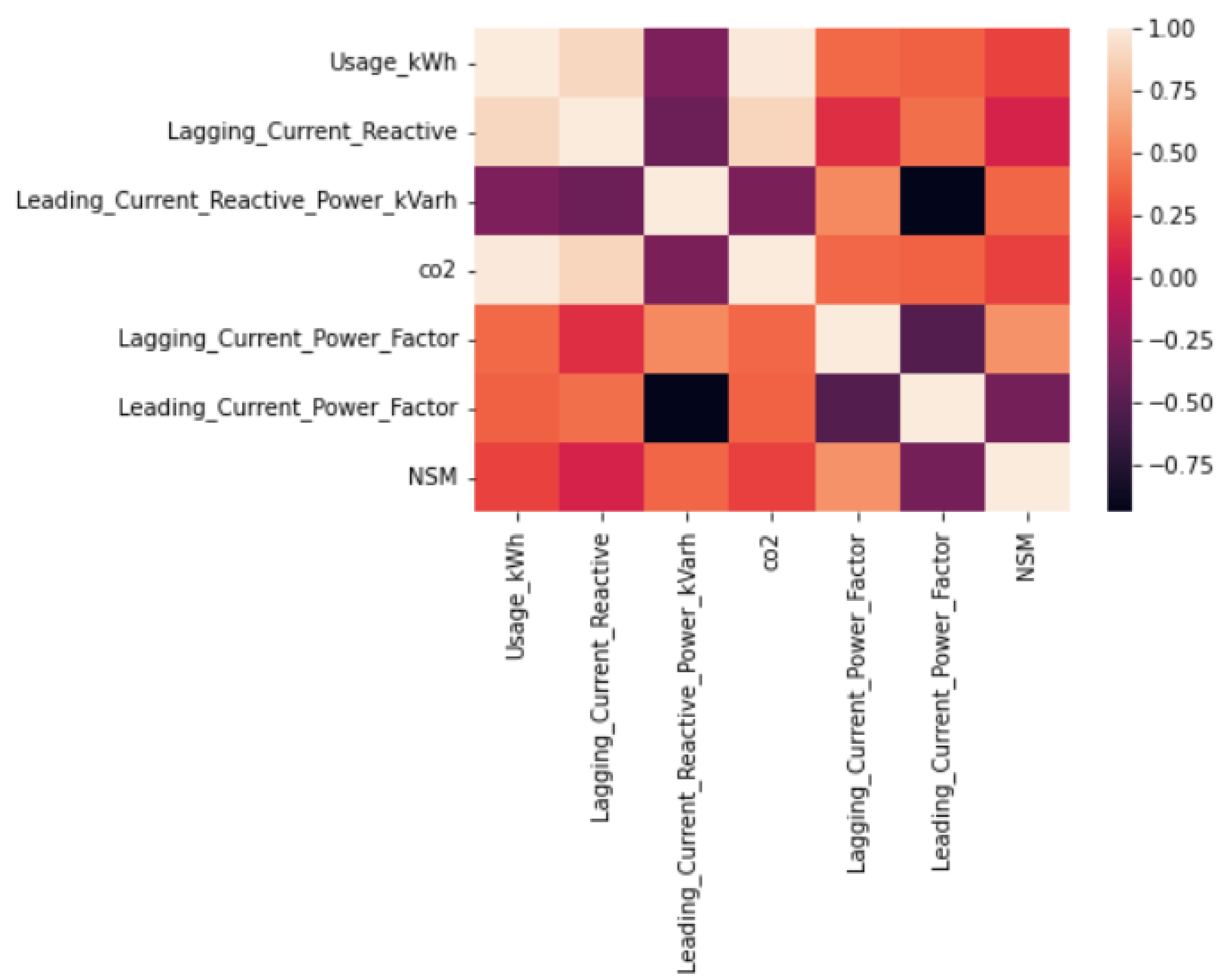
REQUIRED CAR PARKING SPACE



Weekday	25056
Weekend	9984

Heatmap

- **Correlation heatmaps are a type of plot that visualize the strength of relationships between variables.**



Data Preprocessing

Data Preprocessing

In this step,

- **Check null in dataset.**
- **Check count of unique value.**
- **Check duplicated**
- **Split between categorical data and numerical data,**

```
numerical =  df.select_dtypes(include=[np.number])
numerical.columns
```

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

```
categorical = df.select_dtypes(exclude=[np.number])
categorical.columns
```

```
Index(['date', 'WeekStatus', 'Day_of_week', 'Load_Type'], dtype='object')
```

```
# Determine count of unique values for each col
df.nunique()
```

```
date                                    35040
Usage_kWh                                3343
Lagging_Current_Reactive                 1954
Leading_Current_Reactive_Power_kVarh      768
co2                                         8
Lagging_Current_Power_Factor             5079
Leading_Current_Power_Factor             3366
NSM                                        96
WeekStatus                                  2
Day_of_week                                 7
Load_Type                                   3
dtype: int64
```

```
# Checking if any rows are missing any data
df.isnull().sum()
```

```
date                                    0
Usage_kWh                               0
Lagging_Current_Reactive                0
Leading_Current_Reactive_Power_kVarh    0
co2                                     0
Lagging_Current_Power_Factor            0
Leading_Current_Power_Factor            0
NSM                                     0
WeekStatus                              0
Day_of_week                             0
Load_Type                               0
dtype: int64
```

Data Preprocessing

In this step,

- **Encoding process to change the categorical feature to numerical feature using One Hot Encoding,**
- **Set date as index**

```
categorical1 = ['WeekStatus', 'Day_of_week', 'Load_Type']
```

```
for cat in categorical1:
    onehots = pd.get_dummies(df[cat], prefix=cat)
    df = df.join(onehots)
```

```
df = df.set_index('date', append=False)
```

```
df_clean = df.drop(['WeekStatus', 'Day_of_week', 'Load_Type'],axis=1)
df_clean.sample(5)
```

Data Preprocessing

Other Preprocessing,

- Remove oulier data in every single column,
- Rename some of the columns so that there are no errors in the next step.

```
print(f'Jumlah Baris Sebelum Outlier Dihapus: {len(df)}')
filtered_entries = np.array([True] * len(df))
for col in['Lagging_Current_Reactive',
       'Leading_Current_Reactive_Power_kVarh', 'co2',
       'Lagging_Current_Power_Factor', 'Leading_Current_Power_Factor']:

    q1=df[col].quantile(0.25)
    q3=df[col].quantile(0.75)
    iqr=q3-q1

    min_IQR = q1 - (1.5 * iqr)
    max_IQR = q3 + (1.5 * iqr)

    filtered_entries=((df[col]>=min_IQR) & (df[col]<=max_IQR)) & filtered_entries
    df=df[filtered_entries]

print(f'Jumlah Baris Sebelum Outlier Dihapus: {len(df)}')
```

```
Jumlah Baris Sebelum Outlier Dihapus: 35040
Jumlah Baris Sebelum Outlier Dihapus: 23371
```

```
df.rename(columns = {'Lagging_Current_Reactive.Power_kVarh':'Lagging_Current_Reactive'}, i
nplace = True)
df.rename(columns = {'CO2(tCO2)':'co2'}, inplace = True)
```

Modelling & Evaluation

Split data train and test

We split data with ratio 80:20

```
X = df_clean.drop(columns='Usage_kWh')
y = df_clean['Usage_kWh']
```

```
from sklearn.model_selection import train_test_split,cross_validate
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
```

Data Shape

We look shape of data train and data test.

```
print(f'X_train Shape: {(X_train.shape)}')
print(f'y_train Shape: {(y_train.shape)}')
print(f'X_test Shape: {(X_test.shape)}')
print(f'y_test Shape: {(y_test.shape)}')
```

```
X_train Shape: (18696, 18)
y_train Shape: (18696,)
X_test Shape: (4675, 18)
y_test Shape: (4675,)
```

LINEAR REGRESSION

Model Building and Model Training Process

```
from sklearn.linear_model import LinearRegression
regr = LinearRegression()
regr.fit(X_train, y_train)
print(regr.score(X_test, y_test))
```

```
0.978956291618086
```

```
y_pred=regr.predict(X_test)
result = pd.DataFrame({'Actual':y_test,'Prediction':y_pred})
result.head(5)
```

date	Actual	Prediction
03/09/2018 02:00	2.74	2.559534
12/07/2018 07:00	2.99	2.482301
09/11/2018 16:00	57.38	66.415498
18/12/2018 09:30	78.30	86.497118
18/09/2018 06:45	2.88	2.616703

Model Evaluation

```
Data Train Performance Results:
MAE training set 2.61
MSE training set 19.57
RMSE training set 4.42
MAPE training set 0.13


Data Test Performance Results:
MAE test set 2.66
MSE test set 24.41
RMSE test set 4.94
MAPE test set 0.12
```

LASSO

Model Building and Model Training Process

```
from sklearn.linear_model import Lasso

lasso = Lasso(alpha=1)
lasso.fit(X_train, y_train)
print(lasso.score(X_test, y_test))
```

```
0.9020938968174589
```

```
lasso_pred=lasso.predict(X_test)
lasso_result = pd.DataFrame({'Actual':y_test,'Prediction':lasso_pred})
lasso_result.head(5)
```

date	Actual	Prediction
03/09/2018 02:00	2.74	-1.935266
12/07/2018 07:00	2.99	-0.811733
09/11/2018 16:00	57.38	68.822357
18/12/2018 09:30	78.30	81.556939
18/09/2018 06:45	2.88	-3.138150

Model Evaluation

```
Data Train Performance Results:
MAE training set 7.8
MSE training set 112.92
RMSE training set 10.63
MAPE training set 0.9


Data Test Performance Results:
MAE test set 7.87
MSE test set 113.57
RMSE test set 10.66
MAPE test set 0.89
```

RIDGE

Model Building and Model Training Process

```
from sklearn.linear_model import Ridge

ridge = Ridge(alpha = 1)
ridge.fit(X_train, y_train)
ridge.score(X_test, y_test)
```

0.9514734479801025

```
ridge_pred=ridge.predict(X_test)
ridge_result = pd.DataFrame({'Actual':y_test,'Prediction':ridge_pred})
ridge_result.head(5)
```

date	Actual	Prediction
03/09/2018 02:00	2.74	0.294328
12/07/2018 07:00	2.99	-0.143427
09/11/2018 16:00	57.38	69.722400
18/12/2018 09:30	78.30	85.997670
18/09/2018 06:45	2.88	-0.956048

Model Evaluation

```
Data Train Performance Results:
MAE training set 5.33
MSE training set 56.82
RMSE training set 7.54
MAPE training set 0.53


Data Test Performance Results:
MAE test set 5.31
MSE test set 56.29
RMSE test set 7.5
MAPE test set 0.52
```

RANDOM FOREST

Model Building and Model Training Process

```
from sklearn.ensemble import RandomForestRegressor
forest = RandomForestRegressor(n_estimators = 1000,
                               criterion = 'mse',
                               random_state = 1,
                               n_jobs = -1)
forest.fit(X_train,y_train)
```

```
rf_result = pd.DataFrame({'Actual':y_test,'Prediction':forest_test_pred})
rf_result.head(5)
```

date	Actual	Prediction
03/09/2018 02:00	2.74	2.73993
12/07/2018 07:00	2.99	2.99172
09/11/2018 16:00	57.38	57.36993
18/12/2018 09:30	78.30	78.77248
18/09/2018 06:45	2.88	2.87940

Model Evaluation

```
Data Train Performance Results:
MAE training set 0.1
MSE training set 0.1
RMSE training set 0.31
MAPE training set 0.0


Data Test Performance Results:
MAE test set 0.27
MSE test set 0.75
RMSE test set 0.87
MAPE test set 0.01
```

SVR

Model Building and Model Training Process

```
from sklearn.svm import SVR
regressor = SVR(kernel = 'rbf')
regressor.fit(X_train,y_train)
```

```
SVR()
```

```
svr_result = pd.DataFrame({'Actual':y_test,'Prediction':svr_test})
svr_result.head(5)
```

date	Actual	Prediction
03/09/2018 02:00	2.74	2.414686
12/07/2018 07:00	2.99	14.259152
09/11/2018 16:00	57.38	68.973425
18/12/2018 09:30	78.30	34.298847
18/09/2018 06:45	2.88	12.675467

Model Evaluation

```
Data Train Performance Results:
MAE training set 12.67
MSE training set 415.28
RMSE training set 20.38
MAPE training set 0.87


Data Test Performance Results:
MAE test set 12.83
MSE test set 426.12
RMSE test set 20.64
MAPE test set 0.86
```

Conclusion

In regression model, to determine the accuracy of performance of an algorithm, we can take it from the values that have small errors. In other words, the smaller the value of the error generated, the closer the value or distance between the actual value and the prediction value.

MSE, RMSE, or MAE are better be used to compare performance between different regression models. Random Forest Regressor provided the best results than other model

Thank You !

linkedin.com/in/muhammad-randa-yandika
github.com/randayandika/