A Mini-Project Report On

“Steel Industry Energy Consumption”

Submitted By

Sahil Birje

PRN No: 1132220178

Jyoti Pawar

PRN No: 1132220444

Mansi Pawar

PRN No: 1132220309

F.Y. M.Sc. (Data Science and Big Data Analytics)

School of Computer Science and Engineering Department of computer science and application

MIT – World Peace University

Pune – 411038

Academic Year 2022-2024

APRIL – 2023

Dr. Vishwanath Karad MIT WORLD PEACE UNIVERSITY, PUNE
SCHOOL OF COMPUTER SCIENCE

Certificate

This is to certify that
Mansi Pawar Prn No: 1132220309,

Of **M.Sc. (Data Science and Big Data Analytics)** successfully completed her Mini-Project in

Machine Learning

"Steel Industry Energy Consumption"

to our satisfaction and submitted the same during the academic year 2022- 2024

towards the partial fulfillment of degree of Master **of Science in Data Science and Big Data Analytics** of Dr. Vishwanath Karad MIT WORLD PEACE UNIVERSITY, PUNE

SCHOOL OF COMPUTER SCIENCE.

Dr. Shubhalaxmi Joshi	Prof. Surabhi Thatte	Prof. Sachin Bhoite
Associate Dean	Program Head	Professor
Faculty of Science	School of Computer Science	School of Computer Science
MITWPU	MITWPU	MITWPU

Acknowledgement

The satisfaction that accompanies that the successful completion of any task would be incomplete without the mention of people whose ceaseless cooperation made it possible, whose constant guidance and encouragement crown all efforts with success. We are grateful to our project guide Prof. Sachin Bhoite for the guidance, inspiration and constructive suggestions that help us in the preparation of this project. We also thank our colleagues who have helped in successful completion of the project.

Mansi Pawar

Prn No: 1132220309

Table of Contents

CHAPTER 1: INTRODUCTION

1.1 Domain of problem statement
1.2 Motivation
1.3 Problem statement

CHAPTER 2: LITERATURE SURVEY

CHAPTER 3: SOLUTION DESIGN

3.1 Solution Approach
3.2 Technology Stack
3.3 Design Model

CHAPTER 4: SOLUTION IMPLEMENTATION AND RESULT

4.1 Obtaining Data
4.2 EDA
4.3 Pre-processing
4.4 Machine Learning algorithms used
4.5 Results

CHAPTER 5: CONCLUSION AND FUTURE WORK

5.1 Conclusion
5.2 Future Work

CHAPTER 6: REFERENCES

1. INTRODUCTION

1.1 Domain of the problem statement

In today's era energy consumption has become a necessity. The environmental impact of the energy industry is significant, as energy and natural resource consumption are closely related.

Rapidly advancing technologies can potentially achieve a transition of energy generation, water and waste management, and food production towards better environmental and energy usage practices using methods of systems ecology and industrial ecology. An accurate prediction of energy demands could provide useful information to make decisions on energy generation and purchase. Furthermore, an accurate prediction would have a significant impact on preventing overloading and allowing an efficient energy storage. Hence, we can use machine learning models to predict the energy consumption.

In this mini project we will be predicting energy consumption of a small-scale steel industry.

1.2 Motivation

Energy consumption in the steel industry is a significant issue: The steel industry is one of the largest energy-consuming industries globally. By developing a machine learning model to predict energy consumption in the steel industry, it may be possible to identify areas for improvement and optimize energy usage to reduce costs and environmental impact. There is a large amount of data available: The steel industry generates a vast amount of data on energy consumption, production processes, and other relevant factors. This data can be used to train and test machine learning models to predict energy consumption. Machine learning can provide accurate predictions: By using machine learning algorithms, it may be possible to predict energy consumption in the steel industry accurately. This can help to optimize production processes and reduce energy usage, leading to cost savings and environmental benefits. Potential for wider applications: The techniques developed for predicting energy consumption in the steel industry can be applied to other industries, such as the automotive or construction industry, which also consume a large amount of energy.

1.3 Problem statement

To explore energy consumption prediction models for the steel industry to save and plan the future resources of energy which will be needed during the crises. And help the industry to optimize the resource consumption for energy demand.

2. LITERATURE SURVEY

Sr. No.	Paper Title	Publication Year	Author's Name	Outcome /Accuracy	Advantages	Limitations
1.	Industry Energy Consumption Prediction Using Data Mining Techniques	December 28, 2019	Sathish Kumar V E, Jongh Yun Lim, Myeongbae Lee, Kyeongryong Cho, Jangwoo Park, Changsun Shin, and Yongyun Cho* 1	Results indicate that the RF model enhances RMSE, MAE, MAPE and CV values of prediction relative to other regression models such as GLM, CART, SVM and KNN.	Additional features with the energy consumption of the industry can be explored in association with the products they produce, the geometry of the building, sort of equipment, etc. In the process of both the exploratory analysis and developing	Although the results show RF model performance is good, RF also suffers the instability problem in testing and training process. The performance of RF in testing phase is more than double of the error rate in the training phase.

					prediction models, the data analysis shows thought-provoking outcomes.	
2.	ENERGY ANALYSIS OF THE STEEL MAKING INDUSTRY	1 May 1998	MOUSA S. MOHSEN* AND BILAL A. AKASH	Heat losses occur along the line of production of the steel making industry. About 36% of total heat input is lost in the furnace. This is a recoverable heat which shouldnot be wasted. 17% of total heat input is lost in the	China's steel industry eliminated backward production capacity by 65 Mt in 2016 and by 55 Mt in 2017. The productivity utilization rate has increased from approximately 70% in 2015 to over 85% in 2017	**To reduce** its energy consumption and carbon emissions by eliminating backward production capacity (technological upgrading), implementing energy-saving technologies, increasing scrap consumption, and reducing the production of iron-

				crucible and mould. Some of it can be recovered or used in processing of steam. Over 26% of heat is rejected in the cooler.		making systems.

3. SOLUTION DESIGN

3.1 Solution Approach

We have used Steel Energy Industry Consumption data from a South - Korean company. This dataset contains more than 35000 records. First, we have performed EDA on all dataset to understand the data if it contains any null value, missing value, any outliers, etc. After performing EDA we got the basic understanding of our dataset. Based on our observations we have considered some columns and those which were highly correlated were removed by performing label encoding. After performing all these task we have implemented certain machine learning algorithm to know the accuracy and precision of our data.

3.2 Technology Stack

We have used Jupyter notebook and python for EDA, pre-processing as well as for model building.

3.3 Designing model

On our steel industry dataset, we have applied SVM, Logistic regression, XGBoost, LGBM Classifier, Random forest, K-NN and K-fold algorithms. Out of these algorithms LGBM classifier gives highest accuracy.

Our goal is to classify load types such as maximum, medium load and low load depending on the other factors such as Usage kWh, NSM, Week Status, Leading current reactive power kVarh, Lagging current power factor, etc.

4. SOLUTION IMPLEMENTATION AND RESULT

4.1 Obtaining Data

The information gathered is from the DAEWOO Steel Co. Ltd in Gwangyang, South Korea. It produces several types of coils, steel plates, and iron plates. The information on electricity consumption is held in a cloud-based system. The information on energy consumption of the industry is stored on the website of the Korea Electric Power Corporation (pccs.kepco.go.kr), and the perspectives on daily, monthly, and annual data are calculated and shown.

We have collected data from online source i.e https://archive.ics.uci.edu/ml/datasets/Steel+Industry+Energy+Consumption+Dataset

Total Observation: 35040

The above data is collected from a smart small-scale steel industry in Korea.

4.2 EDA

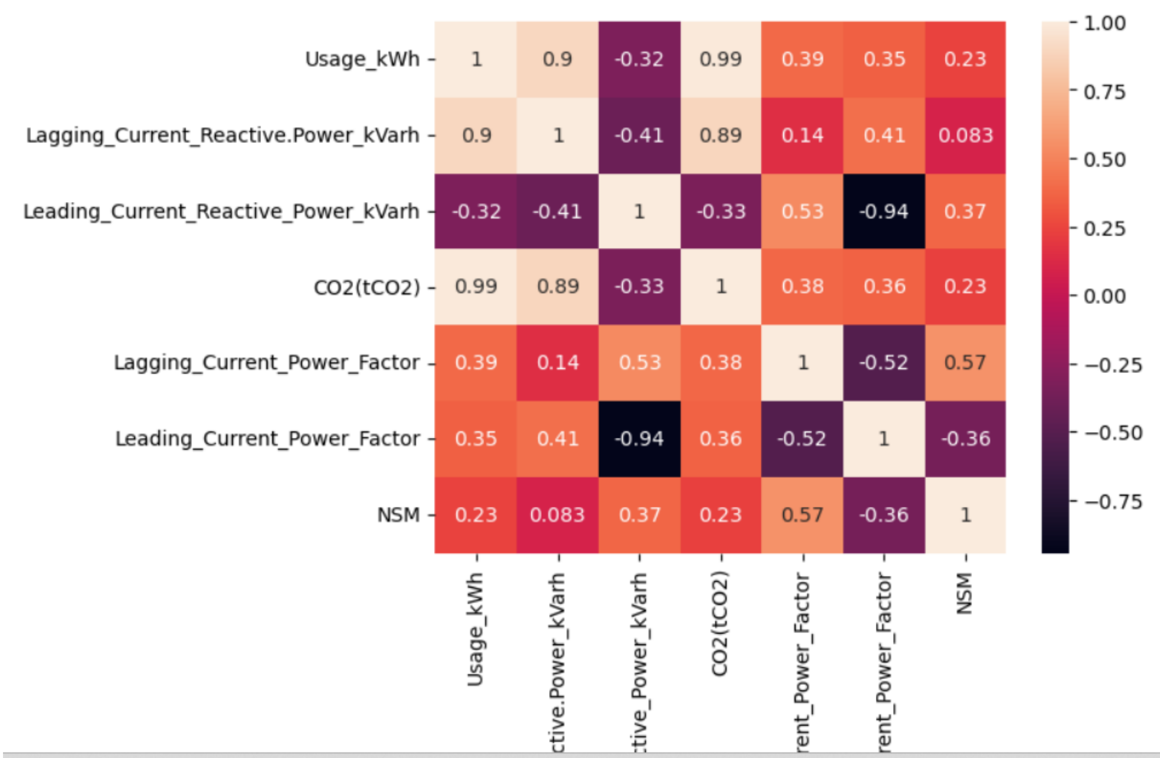
Using python and Jupyter notebook we have done EDA on steel industry dataset. We described the dataset and based on that performed further processing. Also plotted heat map to see correlation between the data points so that we could proceed for feature selection. Also plotted the distribution

```
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
```

```
sns.heatmap(correlation, xticklabels = correlation.columns, yticklabels=correlation.columns, annot = True)
```

```
<AxesSubplot:>
```



Overview | Alerts 17 | Reproduction

Dataset statistics

Number of variables	11
Number of observations	35040
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	10.9 MiB
Average record size in memory	324.8 B

Variable types

Categorical	4
Numeric	7

date

Categorical

HIGH CARDINALITY UNIFORM UNIQUE

Distinct	35040
Distinct (%)	100.0%
Missing	0
Missing (%)	0.0%
Memory size	2.4 MiB

01/01/2018 0…	1
01/09/2018 0…	1
01/09/2018 0…	1
01/09/2018 0…	1
01/09/2018 0…	1
Other values …	35035

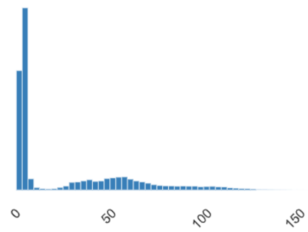
More details

Usage_kWh

Real number (ℝ)

Distinct	3343
Distinct (%)	9.5%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%
Mean	27.386892

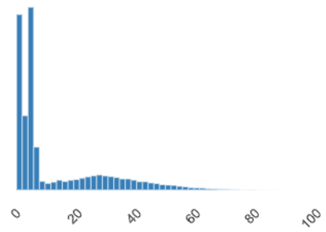
Minimum	0
Maximum	157.18
Zeros	1
Zeros (%)	< 0.1%
Negative	0
Negative (%)	0.0%
Memory size	273.9 KiB



Lagging_Current_Reactive.Power_kVarh
Real number (ℝ)

HIGH CORRELATION ZEROS

Distinct	1954	**Minimum**	0
Distinct (%)	5.6%	**Maximum**	96.91
Missing	0	**Zeros**	7194
Missing (%)	0.0%	**Zeros (%)**	20.5%
Infinite	0	**Negative**	0
Infinite (%)	0.0%	**Negative (%)**	0.0%
Mean	13.035384	**Memory size**	273.9 KiB

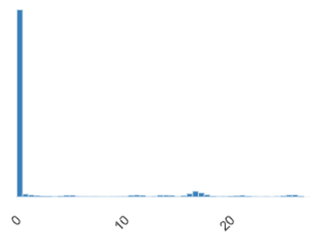


More details

Leading_Current_Reactive_Power_kVarh
Real number (ℝ)

HIGH CORRELATION ZEROS

Distinct	768	**Minimum**	0
Distinct (%)	2.2%	**Maximum**	27.76
Missing	0	**Zeros**	23610
Missing (%)	0.0%	**Zeros (%)**	67.4%
Infinite	0	**Negative**	0
Infinite (%)	0.0%	**Negative (%)**	0.0%
Mean	3.8709486	**Memory size**	273.9 KiB



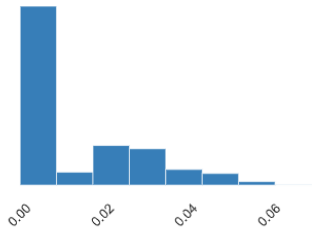
More details

CO2(tCO2)

Real number (ℝ)

HIGH CORRELATION ZEROS

Distinct	8	Minimum	0
Distinct (%)	< 0.1%	Maximum	0.07
Missing	0	Zeros	20990
Missing (%)	0.0%	Zeros (%)	59.9%
Infinite	0	Negative	0
Infinite (%)	0.0%	Negative (%)	0.0%
Mean	0.011524258	Memory size	273.9 KiB

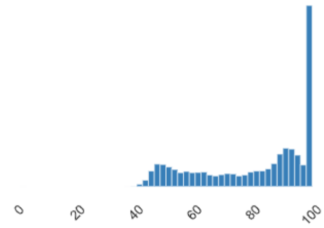


More details

Lagging_Current_Power_Factor

Real number (ℝ)

Distinct	5079	Minimum	0
Distinct (%)	14.5%	Maximum	100
Missing	0	Zeros	1
Missing (%)	0.0%	Zeros (%)	< 0.1%
Infinite	0	Negative	0
Infinite (%)	0.0%	Negative (%)	0.0%
Mean	80.578056	Memory size	273.9 KiB

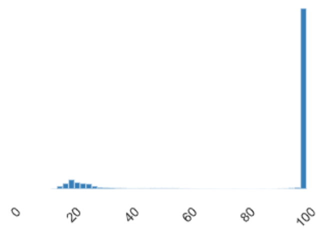


More details

Leading_Current_Power_Factor

Real number (ℝ)

Distinct	3366	Minimum	0
Distinct (%)	9.6%	Maximum	100
Missing	0	Zeros	1
Missing (%)	0.0%	Zeros (%)	< 0.1%
Infinite	0	Negative	0
Infinite (%)	0.0%	Negative (%)	0.0%
Mean	84.36787	Memory size	273.9 KiB



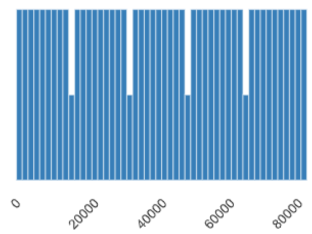
More details

NSM

Real number (ℝ)

HIGH CORRELATION ZEROS

Distinct	96	Minimum	0
Distinct (%)	0.3%	Maximum	85500
Missing	0	Zeros	365
Missing (%)	0.0%	Zeros (%)	1.0%
Infinite	0	Negative	0
Infinite (%)	0.0%	Negative (%)	0.0%
Mean	42750	Memory size	273.9 KiB



More details

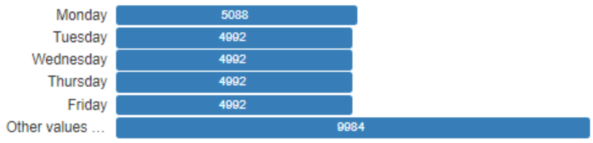
WeekStatus
Categorical

Distinct	2
Distinct (%)	< 0.1%
Missing	0
Missing (%)	0.0%
Memory size	2.1 MiB

More details

Day_of_week
Categorical

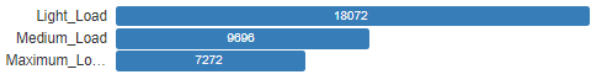
Distinct	7
Distinct (%)	< 0.1%
Missing	0
Missing (%)	0.0%
Memory size	2.1 MiB



More details

Load_Type
Categorical

Distinct	3
Distinct (%)	< 0.1%
Missing	0
Missing (%)	0.0%
Memory size	2.3 MiB



More details

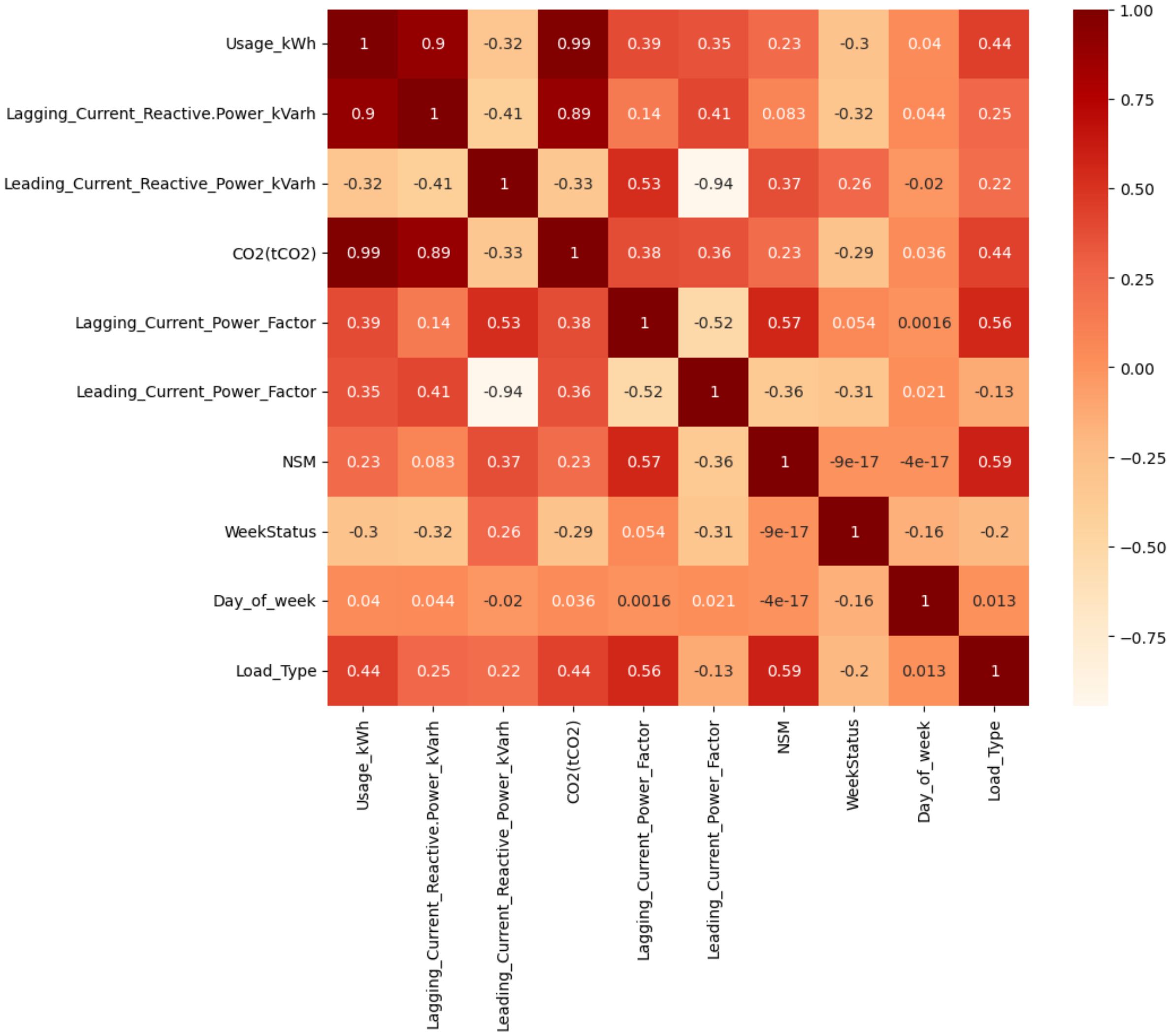
4.3 Pre-Processing

On our steel industry dataset we converted categorical values to numeric for further application. We used label-encoding technique.

```
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()
df.iloc[:,8] = label_encoder.fit_transform(df.iloc[:,8])
df.iloc[:,9] = label_encoder.fit_transform(df.iloc[:,9])
df.iloc[:,10] = label_encoder.fit_transform(df.iloc[:,10])
```

4.4 Machine Learning algorithm used

After the Dataset is pre-processed, it is then ready to feed to the Machine Learning Model. We have used XGB Classifier, LGBM Classifier, Random Forest Classifier, Logistic Regression, K Fold, SVM, K cross, K-NN. The model which performs the best will be used for deployment. We have used classifier models as the target variable is a categorical value. The features selected for prediction are Usage_kWh, Leading_Current_Reactive_Power_kVarh, Lagging_Current_Power_Factor, NSM, WeekStatus, Day_of_week . These features were selected based on correlation with target variable and within the features itself.


Usage_kWh
Lagging_Current_Reactive.Power_kVarh
Leading_Current_Reactive_Power_kVarh
CO2(tCO2)
Lagging_Current_Power_Factor
Leading_Current_Power_Factor
NSM
WeekStatus
Day_of_week
Load_Type
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75

4.5 Results

XGB Classifier:

XGBoost (short for eXtreme Gradient Boosting) is a popular machine learning algorithm that is widely used for regression, classification, and ranking problems. The XGBoost algorithm works by building a series of decision trees, where each tree tries to correct the errors of the previous tree. It uses a gradient descent optimization algorithm to minimize a loss function, which measures the difference between the predicted and actual values.

The data was split into 75-25% for training data and testing data respectively, with the random state as 423

```
test_size = 0.25
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    Y,
                                                    test_size = test_size,
                                                    random_state = 423)
```

Random state was decided using the following code which would check the accuracy of the model for different random states n store the result in another dataframe for reference.

```
%%time

RS = []
Acc = []
for i in range(500,1500):
    rs = i
    X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = test_size, random_state = rs)
    model = XGBClassifier()
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    predictions = [round(value) for value in y_pred]
    accuracy = accuracy_score(y_test, predictions)
    accuracy *= 100
    Acc.append(accuracy)
    RS.append(rs)

best_accuracy_1 = pd.DataFrame(list(zip(RS,Acc)), columns = ['Random_State', 'Accuracy'])
```

```
best_accuracy_1 = best_accuracy_1.sort_values(by=['Accuracy'])
display(best_accuracy_1)
```

The model was fitted with default parameters.

```
model = XGBClassifier()
model.fit(X_train, y_train)
```

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              objective='multi:softprob', predictor=None, ...)
```

The accuracy for the model is 91.187, we have focused more on recall for class 2(High Load) as our goal is to help the industry manage its resources for optimized energy consumption and more resources are consumed during high load. XGB classifier provides us with 85% recall for High Load and 98%, 83% recall for light and medium load respectively.

```
y_pred = model.predict(X_test)
predictions = [round(value) for value in y_pred]
```

```
accuracy = accuracy_score(y_test, predictions)
print(accuracy * 100)
```

```
91.18721461187215
```

```
print(classification_report(y_pred,y_test))
```

```
              precision    recall  f1-score   support

           0       0.98      0.98      0.98      4543
           1       0.85      0.83      0.84      1863
           2       0.84      0.85      0.85      2354

    accuracy                           0.91      8760
   macro avg       0.89      0.89      0.89      8760
weighted avg       0.91      0.91      0.91      8760
```

LGBM Classifier:

LGBM (Light Gradient Boosting Machine) Classifier is a type of gradient boosting algorithm that is designed to be faster and more efficient than traditional gradient boosting algorithms. It is an implementation of the decision tree-based gradient boosting framework, similar to XGBoost.

LGBM Classifier works by building a series of decision trees, where each tree tries to correct the errors of the previous tree. However, LGBM uses a novel technique called 'leaf-wise' growth to construct the decision trees. This technique focuses on growing the tree by adding leaves that will lead to the greatest reduction in the loss function. This leads to faster and more efficient training compared to the traditional 'level-wise' growth approach.

The data was split into 75-25% for training data and testing data respectively, with the random state as 20

```
test_size = 0.25
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = test_size, random_state = 20 )
```

The model was fitted with default parameters.

```
import lightgbm as lgb
from lightgbm import LGBMClassifier

model = LGBMClassifier()
model.fit(X_train, y_train)
```

```
LGBMClassifier()
```

```
test_size = 0.25
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = test_size, random_state = 20 )
```

```
y_pred = model.predict(X_test)
predictions = [round(value) for value in y_pred]
```

```
accuracy = accuracy_score(y_test, predictions)
print(accuracy*100)
```

```
93.42465753424658
```

The accuracy for the model is 93.424, we have focused more on recall for class 2(High Load) as our goal is to help the industry manage its resources for optimized energy consumption and more resources are consumed during high load. LGBM classifier provides us with 90% recall for High Load and 98%, 86% recall for light and medium load respectively.

```
print(classification_report(y_pred,y_test))
```

```
              precision    recall  f1-score   support

           0       0.99      0.98      0.98      4565
           1       0.90      0.86      0.88      1908
           2       0.87      0.90      0.88      2287

    accuracy                           0.93      8760
   macro avg       0.92      0.91      0.91      8760
weighted avg       0.93      0.93      0.93      8760
```

Random Forest Classifier:

Random Forest Classifier is a popular machine learning algorithm that is widely used for classification and regression tasks. It is an ensemble learning method that combines multiple decision trees to make more accurate predictions.

Random Forest Classifier works by building multiple decision trees on random subsets of the training data and random subsets of the features. The decision trees are constructed using a random selection of features at each node to split the data. This helps to reduce the correlation between the trees and improve the overall performance.

The data was split into 75-25% for training data and testing data respectively, with the random state as 20

```
test_size = 0.25
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = test_size, random_state = 20 )
```

```
param_dist = {'n_estimators': randint(50,500),
              'max_depth': randint(1,20)}
```

```
rf = RandomForestClassifier()
```

```
rand_search = RandomizedSearchCV(rf,
                                 param_distributions = param_dist,
                                 n_iter=5,
                                 cv=5)
```

```
rand_search.fit(X_train, y_train)
```

The model was fitted by using the best hyparameters.

```
rand_search.fit(X_train, y_train)
```

```
RandomizedSearchCV(cv=5, estimator=RandomForestClassifier(), n_iter=5,
                   param_distributions={'max_depth': <scipy.stats._distn_infrastructure.rv_discrete_frozen object at 0x0000020D
16271460>,
                                        'n_estimators': <scipy.stats._distn_infrastructure.rv_discrete_frozen object at 0x00000
20D14ED0760>})
```

```
best_rf = rand_search.best_estimator_
print('Best hyperparameters:',  rand_search.best_params_)
```

```
Best hyperparameters: {'max_depth': 15, 'n_estimators': 107}
```

```
y_pred = best_rf.predict(X_test)
```

```
predictions = [round(value) for value in y_pred]
accuracy = accuracy_score(y_test, predictions)
print(accuracy*100)
```

```
91.13013698630137
```

The accuracy for the model is 91.131, we have focused more on recall for class 2(High Load) as our goal is to help the industry manage its resources for optimized energy consumption and more resources are consumed during high load. Random Forest Classifier provides us with 86% recall for High Load and 97%, 83% recall for light and medium load respectively.

```
print(classification_report(y_pred,y_test))
```

```
              precision    recall  f1-score   support

           0       0.98      0.97      0.97      4571
           1       0.86      0.83      0.84      1921
           2       0.82      0.86      0.84      2268

    accuracy                           0.91      8760
   macro avg       0.89      0.89      0.89      8760
weighted avg       0.91      0.91      0.91      8760
```

Logistic Regression:

We are using logistic regression to classify load types. The model uses a logistic function, which maps any input value to an output value between 0 and 1. This output value represents the probability of the binary outcome being 1. The logistic regression model estimates the coefficients of the independent variables that maximize the likelihood of the observed data given the model. After applying logistic regression our model is giving 0.72 as accuracy score.

The data was split into 75-25% for training data and testing data respectively, with the random state as 50.

```
In [15]: from sklearn.model_selection import train_test_split
         x_train, x_test, y_train , y_test = train_test_split(x, y, test_size=0.25, random_state=50)

In [16]: from sklearn.linear_model import LogisticRegression
         classifier=LogisticRegression(solver = "liblinear")

In [17]: from sklearn.model_selection import GridSearchCV
         parameter={'penalty':['l1','l2','elasticnet'],'C':[1,2,3,4,5]}

In [18]: classifier_regressor=GridSearchCV(classifier,param_grid=parameter,scoring='accuracy',cv=5)

In [19]: classifier_regressor.fit(x_train,y_train)
         Traceback (most recent call last):
           File "C:\Users\Admin\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py", line 680, in _fit_and_score
             estimator.fit(X_train, y_train, **fit_params)
           File "C:\Users\Admin\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py", line 1461, in fit
             solver = _check_solver(self.solver, self.penalty, self.dual)
           File "C:\Users\Admin\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py", line 457, in _check_solver
             raise ValueError(
         ValueError: Only 'saga' solver supports elasticnet penalty, got solver=liblinear.

           warnings.warn(some_fits_failed_message, FitFailedWarning)
         C:\Users\Admin\anaconda3\lib\site-packages\sklearn\model_selection\_search.py:969: UserWarning: One or more of the test score
         s are non-finite: [0.70901912 0.70414759        nan 0.70964762 0.70721242        nan
          0.70925478 0.71122028        nan 0.70972626 0.71711314        nan
          0.70949048 0.70996183        nan]
           warnings.warn(

Out[19]: GridSearchCV(cv=5, estimator=LogisticRegression(solver='liblinear'),
                      param_grid={'C': [1, 2, 3, 4, 5],
                                  'penalty': ['l1', 'l2', 'elasticnet']},

In [20]: print(classifier_regressor.best_params_)

         {'C': 4, 'penalty': 'l2'}

In [21]: print(classifier_regressor.best_score_)

         0.7171131417671234

In [22]: ###prediction
         y_pred=classifier_regressor.predict(x_test)

In [23]: ###accuracy score
         from sklearn.metrics import accuracy_score,classification_report

In [24]: score=accuracy_score(y_pred,y_test)
         print(score)

         0.7213578500707214

In [25]: print(classification_report(y_pred,y_test))

                       precision    recall  f1-score   support

                    0       0.73      0.77      0.75      2286
                    1       0.71      0.66      0.69      1956

             accuracy                           0.72      4242
            macro avg       0.72      0.72      0.72      4242
         weighted avg       0.72      0.72      0.72      4242
```

The accuracy for the model is 0.72, we have focused more on recall for class 2(High Load) as our goal is to help the industry manage its resources for optimized energy consumption and more resources are consumed during high load. Logistic Regression provides us with 66% recall for High Load and 77% recall for medium load respectively.

K-fold cross-validation:

It is a technique used to evaluate the performance of a machine learning model. The basic idea behind K-fold cross-validation is to split the dataset into K equally sized subsets, or "folds," and then use one-fold as the validation set while training the model on the remaining K-1 folds. This process is repeated K times, with each fold used as the validation set once, and the results are averaged across all the iterations.

```
In [8]: from sklearn.linear_model import LogisticRegression
        from sklearn.svm import SVC
        from sklearn.ensemble import RandomForestClassifier
        import numpy as np
        from sklearn.datasets import load_digits
        import matplotlib.pyplot as plt
        digits = load_digits()
        from sklearn.model_selection import train_test_split
        X_train, X_test, y_train, y_test = train_test_split(digits.data,digits.target,test_size=0.3)
```

```
In [9]: lr = LogisticRegression(solver='liblinear',multi_class='ovr')
        lr.fit(X_train, y_train)
        lr.score(X_test, y_test)

Out[9]: 0.9648148148148148
```

```
In [10]: svm = SVC(gamma='auto')
         svm.fit(X_train, y_train)
         svm.score(X_test, y_test)

Out[10]: 0.35185185185185186
```

```
In [11]: rf = RandomForestClassifier(n_estimators=40)
         rf.fit(X_train, y_train)
         rf.score(X_test, y_test)

Out[11]: 0.975925925925926
```

```
In [12]: from sklearn.model_selection import cross_val_score
         #Set LogisticRegression, CV =3

         score_lr=cross_val_score(LogisticRegression(solver='liblinear',multi_class='ovr'), digits.data, digits.target,cv=3)
         print(score_lr)
         print("Avg :",np.average(score_lr))

         [0.89482471 0.95325543 0.90984975]
         Avg : 0.9193099610461881
```

```
In [13]: #Set SVM and CV=3

         score_svm =cross_val_score(SVC(gamma='auto'), digits.data, digits.target,cv=3)
         print(score_svm)
         print("Avg :",np.average(score_svm))

         [0.38063439 0.41068447 0.51252087]
         Avg : 0.4346132442960489
```

```
In [14]: #Set Random Forest and CV=3

         score_rf=cross_val_score(RandomForestClassifier(n_estimators=40),digits.data, digits.target,cv=3)
         print(score_rf)
         print("Avg :",np.average(score_rf))

         [0.93656093 0.96327212 0.92153589]
         Avg : 0.9404563160823595
```

```
In [22]: from sklearn.model_selection import cross_val_score
         from sklearn.neighbors import KNeighborsClassifier
         knn = KNeighborsClassifier(n_neighbors=5)
         scores = cross_val_score(knn, x, y, cv=10, scoring='accuracy')
         print(scores.mean())
```

```
C:\Users\Admin\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other reduction func
tions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, th
is behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is taken will b
e eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\Users\Admin\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other reduction func
tions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, th
is behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is taken will b
e eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\Users\Admin\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other reduction func
tions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, th
is behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is taken will b
e eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\Users\Admin\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other reduction func
tions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, th
is behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is taken will b
e eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\Users\Admin\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other reduction func
tions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, th
is behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is taken will b
e eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\Users\Admin\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other reduction func
tions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, th
is behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is taken will b
e eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\Users\Admin\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other reduction func
tions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, th
is behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is taken will b
e eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\Users\Admin\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other reduction func
tions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, th
is behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is taken will b
e eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\Users\Admin\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other reduction func
tions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, th
is behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is taken will b
e eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

```
0.8525684931506848
```

```
C:\Users\Admin\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other reduction func
tions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, th
is behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is taken will b
e eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

```
In [23]: k_range = list(range(1, 25))
         k_scores = []
         for k in k_range:
             knn = KNeighborsClassifier(n_neighbors=k)
             scores = cross_val_score(knn, x, y, cv=10, scoring='accuracy')
             k_scores.append(scores.mean())
         print(k_scores)
```

```
ing.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\Users\Admin\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other reduction
functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.
11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is
taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warn
ing.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

```
[0.8393835616438355, 0.8453767123287672, 0.850142694063927, 0.8529965753424656, 0.8525684931506848, 0.8558504566210046, 0.8
547659817351597, 0.8567066210045663, 0.8563356164383562, 0.8568207762557078, 0.856392694063927, 0.8561929223744291, 0.85510
84474885844, 0.8568778538812785, 0.8554223744292238, 0.8551369863013699, 0.8541095890410959, 0.8553652968036529, 0.85488013
69863014, 0.8552511415525116, 0.8555650684931507, 0.8556792237442922, 0.8555365296803652, 0.8564497716894977]
```

As seen in the above screenshot of the algorithm used we can see the Average of logistic regression as 0.91 , Average of SVM as 0.43, Average of Random Forest as 0.94 and the KNN Classifier Accuracy as 0.85 .

K-NN:

KNN is a machine learning algorithm used for classification and regression problems. It works by finding the k-nearest neighbors of a given data point in the feature space and using their class or output values to predict the class or output value of the new data point. KNN requires a labeled dataset, a distance metric, and a value for k.

```
In [8]: from sklearn.model_selection import train_test_split

In [9]: X_train, X_test, y_train, y_test = train_test_split(x,y,test_size=0.30,random_state=0)

In [10]: from sklearn.neighbors import KNeighborsClassifier

In [11]: knn = KNeighborsClassifier(n_neighbors=1)

In [12]: knn.fit(X_train,y_train)

Out[12]: KNeighborsClassifier
         KNeighborsClassifier(n_neighbors=1)

In [13]: pred = knn.predict(X_test)

In [15]: print(confusion_matrix(y_test,pred))

[[5196   48  181]
 [  50 1614  496]
 [ 208  424 2295]]
```

Row 1 (high load type): The model predicted that 5196 instances were of high load type, and this was correct for 5196 of them. However, it incorrectly predicted 48 instances as medium load type and 181 instances as low load type.

Row 2 (medium load type): The model predicted that 1614 instances were of medium load type, and this was correct for 1614 of them. However, it incorrectly predicted 50 instances as high load type and 496 instances as low load type.

Row 3 (low load type): The model predicted that 2295 instances were of low load type, and this was correct for 2295 of them. However, it incorrectly predicted 208 instances as high load type and 424 instances as medium load type.

```
In [16]: print(classification_report(y_test,pred))

              precision    recall  f1-score   support

           0       0.95      0.96      0.96      5425
           1       0.77      0.75      0.76      2160
           2       0.77      0.78      0.78      2927

    accuracy                           0.87     10512
   macro avg       0.83      0.83      0.83     10512
weighted avg       0.87      0.87      0.87     10512
```

The accuracy for the model is 0.87, we have focused more on recall for class 2 i.e high load as our goal is to help the industry manage its resources for optimized energy consumption and more resources are consumed during high load. KNN provides us with 78%(high load) and 75%(medium load) recall for class 2 and class 1 respectively.

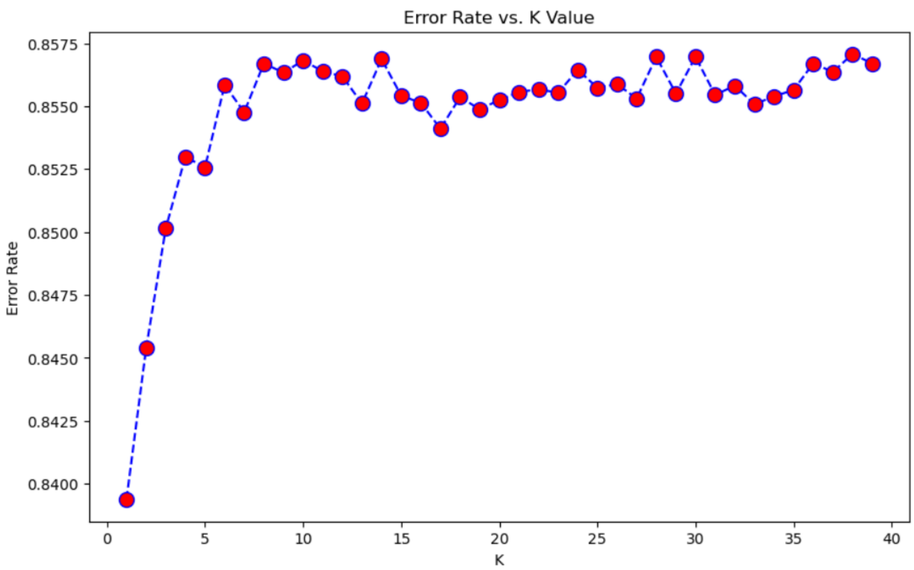
```
accuracy_rate = []

for i in range(1,40):

    knn = KNeighborsClassifier(n_neighbors=i)
    score=cross_val_score(knn,x,y,cv=10)
    accuracy_rate.append(score.mean())
```

```
error_rate = []
for i in range(1,40):

    knn = KNeighborsClassifier(n_neighbors=i)
    score=cross_val_score(knn,x,y,cv=10)
    error_rate.append(1-score.mean())
```

```
error_rate = []
for i in range(1,40):

    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,y_train)
    pred_i = knn.predict(X_test)
    error_rate.append(np.mean(pred_i != y_test))
```

```
plt.figure(figsize=(10,6))
plt.plot(range(1,40),accuracy_rate,color='blue', linestyle='dashed', marker='o
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```

```
Out[20]: Text(0, 0.5, 'Error Rate')
```



Above is the plot for Error rate Vs K value. Here we can see that the datapoints are not fluctuating more from 10 onwards.

In [21]:
```python
# FIRST A QUICK COMPARISON TO OUR ORIGINAL K=1
knn = KNeighborsClassifier(n_neighbors=1)

knn.fit(X_train,y_train)
pred = knn.predict(X_test)

print('WITH K=1')
print('\n')
print(confusion_matrix(y_test,pred))
print('\n')
print(classification_report(y_test,pred))
```

```
WITH K=1


[[5196   48  181]
 [  50 1614  496]
 [ 208  424 2295]]


              precision    recall  f1-score   support

           0       0.95      0.96      0.96      5425
           1       0.77      0.75      0.76      2160
           2       0.77      0.78      0.78      2927

    accuracy                           0.87     10512
   macro avg       0.83      0.83      0.83     10512
weighted avg       0.87      0.87      0.87     10512
```

In [25]:
```python
# NOW WITH K=10
knn = KNeighborsClassifier(n_neighbors=23)

knn.fit(X_train,y_train)
pred = knn.predict(X_test)

print('WITH K=10')
print('\n')
print(confusion_matrix(y_test,pred))
print('\n')
print(classification_report(y_test,pred))
```

```
WITH K=10


[[5205   31  189]
 [  59 1741  360]
 [ 205  456 2266]]


              precision    recall  f1-score   support

           0       0.95      0.96      0.96      5425
           1       0.78      0.81      0.79      2160
           2       0.80      0.77      0.79      2927

    accuracy                           0.88     10512
   macro avg       0.85      0.85      0.85     10512
weighted avg       0.88      0.88      0.88     10512
```

1. First we have chosen k=1, we get 78%(high load) and 75%(medium load) recall for class 2 and class 1 respectively.
2. For k=10, we get 77%(high load) and 81%(medium load) recall for class 2 and class 1 respectively.

5. CONCLUSION AND FUTURE WORK

5.1 Conclusion

After observing the accuracy and performance metrics of the all the models, it is concluded that LGBM Classifier is the best suited model for the task as it gives an accuracy of 93.424% and good recall for class 2(high load) and remaining two classes as well.

5.2 Future Work

We currently have collected data from only once source i.e. Korean Power Plant, in future we can get data from more plants located over various regions and owned by different companies or from different domains to get more scenarios and find more patterns

6 REFERENCES

- https://www.kaggle.com/ •
- https://archive.ics.uci.edu/ml/datasets/Steel+Industry+Energy+Consumption+Dataset
- https://gvpress.com/journals/IJEIC/vol11_no1/2.pdf
- https://koreascience.kr/article/JAKO202015762902359.page
- https://gvpress.com/journals/IJEIC/vol11_no1/vol11_no1_2020_02.html