Configuration Manual

MSc Research Project
MSc in Data Analytics

Srushti Prakash Ghadge
Student ID: x20234082

School of Computing
National College of Ireland

Supervisor: Dr. Catherine Mulwa

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Srushti Prakash Ghadge		
Student ID:	20234082		
Programme:	MSc in Data Analytics	**Year:**	2021-2022
Module:	Research Project		
Lecturer:	Dr. Catherine Mulwa		
Submission Due Date:	15-10-2022		
Project Title:	Electricity Theft Detection based on Machine Learning Algorithms: China		
Word Count:	1321	**Page Count:**	**16**

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Srushti Prakash Ghadge

Date: 15-08-2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Srushti Prakash Ghadge
Student ID: x20234082

1 Introduction

The setup handbook is a step-by-step manual that provides project direction for the project "Electricity Theft Detection Using Machine Learning Algorithms" that is given in a technical report. The goal of this report is to guide the reader through each phase and help them get the output and results they are looking for, which are then provided in a technical report. A variety of libraries, technologies, and software configurations are used to implement the complete project.

1.1 Project Overview

The aim of this project is to identify and detect electricity theft. The methods used are Boosting algorithm with CNN, Random Forest, K-mean clustering, and Decision Tree. The Boosting algorithm produced better results, which may aid in identifying consumers who are engaging in fraud.

2 Pre-requisites

Programming Language: Python.
Development Tools: Jupyter Notebook, Google Colab, Microsoft Excel.

MS Word and MS Excel from the Microsoft Office suite were utilized for data selection, viewing, and reporting. Python is the primary programming language employed in this study. Python 3.8.8 was utilized for this study, and it may be obtained for free from their official website. Python programming was done on the Anaconda platform, which can also be downloaded for free from their website. The Anaconda Navigator's Jupyter Notebook application was employed. Jupyter Notebooks include benefits such as quick implementation and ease of use.

3 Software Installation Guide

1. Install & Download the Anaconda Distribution.
2. Download the Anaconda Distribution package.
3. Installation of Anaconda
4. From Navigator, create an Anaconda from Navigator, create an Anaconda Launch Anaconda Navigator.
5. Establish a Setting for Jupyter Notebook.
6. Setup and Use of Jupyter Notebook

4 Project Implementation Guide

The project's implementation is discussed in this section. This part provides a brief explanation of all the codes, packages, and reasoning.

4.1 CNN with the Boosting algorithm

This is with the reference to the files 'XGBoost_china_data' present in the code artefact. The below screenshots contains the daily usage analysis with 70% training data. Please note that the same procedure has been followed for Daily power consumption data utilized for training with 80% of the total dataset, Monthly power consumption data utilized for training with 70% and 80 % of the total dataset in the files 'XGBoost_data80', 'XGBoost_china_dataM70', 'XGBoost_china_dataM80' respectively.

Steps followed for the data process and EDA

In Figure 1, the dataset is imported, as well as all the necessary libraries, and may be viewed as shown below.[1]

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly as pt
```

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

Figure 1: Load Libraries for CNN

Process Null Dataset

In Figure 2, Figure 3, Figure 4 the dataset for the model training is cleaned up of undesirable information. Look for missing or empty values. If so, remove it or replace it with zeros or the dataset's median; otherwise, carry on.

```
label_data = data['FLAG']
del data['FLAG']
del data['CONS_NO']
del label_data[1]
```

```
data
```

	2014/1/1	2014/1/10	2014/1/11	2014/1/12	2014/1/13	2014/1/14	2014/1/15	2014/1/16	2014/1/17	2014/1/18	...	2016/9/28	2016/9/29	2016/9/3	2016/9/30	20
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	10.12	9.96	16.92	7.60	
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	0.00	0.00	0.00	0.00	
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	6.50	9.99	11.78	18.59	
4	2.90	3.42	3.81	4.58	3.56	4.25	3.86	3.53	3.41	0.85	...	17.77	10.37	15.32	13.51	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	

Figure 2: Process Null Dataset

[1] https://github.com/henryRDlab/ElectricityTheftDetection

```
new_data = data.T
for cols in new_data.columns:
    a = new_data[cols].isnull().sum()
    if a>0:
        new_data[cols] = new_data[cols].fillna(new_data[cols].median())

print(new_data.isna().values.sum())
new_data = new_data.fillna(0)
```

Figure 3: Treat Null dataset

```
new_data.T
```

	2014/1/1	2014/1/10	2014/1/11	2014/1/12	2014/1/13	2014/1/14	2014/1/15	2014/1/16	2014/1/17	2014/1/18	...	2016/9/28	2016/9/29	2016/9/3	2016/9/30	20
0	9.770	9.770	9.770	9.770	9.770	9.770	9.770	9.770	9.770	9.770	...	10.12	9.96	16.92	7.60	
1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	...	0.00	0.00	0.00	0.00	
2	8.900	8.900	8.900	8.900	8.900	8.900	8.900	8.900	8.900	8.900	...	8.90	8.90	8.90	8.90	
3	12.735	12.735	12.735	12.735	12.735	12.735	12.735	12.735	12.735	12.735	...	6.50	9.99	11.78	18.59	
4	2.900	3.420	3.810	4.580	3.560	4.250	3.860	3.530	3.410	0.850	...	17.77	10.37	15.32	13.51	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	

Figure 4: Output after treating null dataset

In Figure 5, verified the timing of the dates. If so, use the date-time function to control the date-time line. Move on if not. Verify any continuous zero value in the consumer, including null. If so, it has no effect on customers.

```
new_data.sort_index()
new_data.reset_index(inplace=True)
new_data['index'] = new_data['index'].astype('datetime64')
new_data.index =new_data['index']
del(new_data['index'])
del(new_data[1])
process_data = new_data.sort_index().T
print(process_data)
```

```
index  2014-01-01  2014-01-02  2014-01-03  2014-01-04  2014-01-05  2014-01-06  \
0           9.770       9.770       9.770       9.770       9.770       9.770
2           8.900       8.900       8.900       8.900       8.900       8.900
3          12.735      12.735      12.735      12.735      12.735      12.735
4           2.900       5.640       6.990       3.320       3.610       5.350
5           3.200       3.200       3.200       3.200       3.200       3.200
...           ...         ...         ...         ...         ...         ...
```

Figure 5: Sort index

```
nulls_removel = process_data.T
labs = label_data
for i in nulls:
  del nulls_removel[i]
  del labs[i]
  # print(i)
```

```
nulls_removel
```

index	0	2	3	4	5	6	7	8	9	10	...	42362	42363	42364	42365	42366	42367	42368	42369	42370	42371
2014-01-01	9.77	8.90	12.735	2.90	3.20	0.11	0.91	6.60	11.02	0.345	...	9.525	148.40	0.00	5.22	1.21	4.26	2.70	0.58	16.89	8.09
2014-01-02	9.77	8.90	12.735	5.64	3.20	0.11	1.16	6.60	7.92	0.345	...	9.525	159.86	0.00	5.04	1.21	4.26	0.00	1.16	15.15	8.09
2014-01-03	9.77	8.90	12.735	6.99	3.20	0.25	0.75	6.60	8.41	0.345	...	9.525	157.20	0.00	4.92	1.21	4.26	0.00	0.92	19.28	8.09
2014-01-04	9.77	8.90	12.735	3.32	3.20	0.27	1.30	6.60	9.66	0.345	...	9.525	104.80	0.00	4.88	1.21	4.26	5.72	0.98	17.19	8.09
2014-01-05	9.77	8.90	12.735	3.61	3.20	0.21	0.74	6.60	9.86	0.345	...	9.525	118.17	0.00	13.59	1.21	4.26	6.05	1.54	16.80	8.09
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

Figure 6: Deleting nulls

Data Pre-processing

In Figure 7, the train test split method is used to divide the dataset before performing an unbiased model and identifying overfitting or underfitting issues. Divide data using a machine learning library's train-test-split function Learn about the model selecting software. By using this strategy, the model's biases during the evaluation and validation process are reduced.

```
from numpy import random
def Data_split (features,lables):
  random_state = np.random.RandomState(32)
  features_tr,feature_tt,lables_tr,lables_tt = train_test_split(features,lables,
                                                               test_size=0.30, random_state=random_state)
  return features, feature_tt, lables, lables_tt
```

```
x_train,x_test, y_train, y_test = Data_split(scaled_data,labs)
```

Figure 7: Split data into train and test

Implementing CNN with XGBoost Algorithm

Here Figure 8, Figure 9, Figure 10 displays the implementation of CNN model and Figure 11 displays the implementation of XGBoost algorithm.

.

```
import tensorflow as tf
import keras
from keras import layers, models
from keras.layers.convolutional import Conv1D, MaxPooling1D
from keras.layers import Dense, Flatten, Dropout
```

```
#CNN model
CNNS_model= models.Sequential()
CNNS_model.add(Conv1D(filters=32, kernel_size=4,
                     activation='relu',
                     input_shape=(x_train.shape[1],1)))
CNNS_model.add(Conv1D(filters = 48, kernel_size=4,
                     activation='relu'))
CNNS_model.add(MaxPooling1D(pool_size=2))
CNNS_model.add(Conv1D(16, kernel_size=1,
                      activation='relu'))
CNNS_model.add(Conv1D(16, kernel_size=1,
                     activation='relu'))
CNNS_model.add(Dropout(0.15))
CNNS_model.add(Flatten())
CNNS_model.add(Dense(32, activation='relu'))
CNNS_model.add(Dense(2))

CNNS_model.compile(optimizer='adam',
                   loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                   metrics=["accuracy"])
CNNS_model.summary()
```

Figure 8: CNN Model

```
Model: "sequential"
```

```
CNNS_model.fit(x_train,y_train, batch_size=32,
          epochs=10, verbose=1,
          validation_data=(x_train,y_train), )
```

```
Epoch 1/10
1259/1259 [==============================] - 140s 111ms/step - loss: 0.2770 - accuracy: 0.9106 - val_loss: 0.2536 - val_accurac
y: 0.9111
Epoch 2/10
1259/1259 [==============================] - 135s 107ms/step - loss: 0.2546 - accuracy: 0.9111 - val_loss: 0.2301 - val_accurac
y: 0.9111
Epoch 3/10
1259/1259 [==============================] - 152s 121ms/step - loss: 0.2362 - accuracy: 0.9129 - val_loss: 0.2127 - val_accurac
y: 0.9154
Epoch 4/10
1259/1259 [==============================] - 132s 105ms/step - loss: 0.2194 - accuracy: 0.9182 - val_loss: 0.1893 - val_accurac
y: 0.9256
Epoch 5/10
1259/1259 [==============================] - 152s 120ms/step - loss: 0.2010 - accuracy: 0.9244 - val_loss: 0.1752 - val_accurac
y: 0.9241
Epoch 6/10
1259/1259 [==============================] - 133s 105ms/step - loss: 0.1786 - accuracy: 0.9317 - val_loss: 0.1421 - val_accurac
y: 0.9497
Epoch 7/10
1259/1259 [==============================] - 136s 108ms/step - loss: 0.1607 - accuracy: 0.9389 - val_loss: 0.1184 - val_accurac
y: 0.9540
Epoch 8/10
1259/1259 [==============================] - 151s 120ms/step - loss: 0.1376 - accuracy: 0.9470 - val_loss: 0.0955 - val_accurac
y: 0.9667
Epoch 9/10
1259/1259 [==============================] - 135s 107ms/step - loss: 0.1175 - accuracy: 0.9554 - val_loss: 0.0741 - val_accurac
y: 0.9762
Epoch 10/10
1259/1259 [==============================] - 150s 119ms/step - loss: 0.1039 - accuracy: 0.9609 - val_loss: 0.0663 - val_accurac
y: 0.9804
```

```
from tensorflow.keras.models import load_model
from tensorflow.keras.models import Model
```

```
CNNS_model.save('cnn_sstructre_extractors.model')
```

```
INFO:tensorflow:Assets written to: cnn_sstructre_extractors.model/assets
```

```
prepred_model_weights = Model(inputs=CNNS_model.input, outputs = CNNS_model.get_layer('dense').output)
prepred_model_weights.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d_input (InputLayer)   [(None, 1034, 1)]         0

 conv1d (Conv1D)             (None, 1031, 32)          160

 conv1d_1 (Conv1D)           (None, 1028, 48)          6192

 max_pooling1d (MaxPooling1D  (None, 514, 48)          0
 )

 conv1d_2 (Conv1D)           (None, 514, 16)           784

 conv1d_3 (Conv1D)           (None, 514, 16)           272

 dropout (Dropout)           (None, 514, 16)           0

 flatten (Flatten)           (None, 8224)              0

 dense (Dense)               (None, 32)                263200

=================================================================
Total params: 270,608
Trainable params: 270,608
Non-trainable params: 0
```

Figure 10: Model Weight Summary

```
from xgboost import XGBClassifier
model_xg = XGBClassifier(scale_pos_weight = 9.5132)
model_xg.fit(Feature, y_train)
```

```
XGBClassifier(scale_pos_weight=9.5132)
```

```
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
y_preds_xg = model_xg.predict(prepred_model_weights.predict(x_test))
print(classification_report(y_test,y_preds_xg))
```

```
              precision    recall  f1-score   support

           0       1.00      0.96      0.98     10969
           1       0.70      0.97      0.81      1109

    accuracy                           0.96     12078
   macro avg       0.85      0.96      0.90     12078
weighted avg       0.97      0.96      0.96     12078
```

```
Tn,Fp, Fn, Tp = confusion_matrix(y_test, y_preds_xg).ravel()
print('\n True negative:',Tn,
     '\n False Negative:',Fn,
     '\n True Postive:', Tp,
     '\n False Positive:',Fp)
print('miss classification rate:',((Fp+Fn)/len(y_preds_xg))*100)
```

```
 True negative: 10509
 False Negative: 32
 True Postive: 1077
 False Positive: 460
miss classification rate: 4.0735221063089915
```

Figure 11: XGBoost implementation

4.2 Random Forest Model

This is with the reference to the file ‘Random Forest’ present in the code artifact.

Importing required libraries and datasets

In Figure 12, the dataset is imported, viewed, and all necessary libraries have been imported, as seen below.

```
import numpy as np # for the math and matrix operations
import pandas as pd # For the data loading into programm and data analysis
import matplotlib.pyplot as plt #for the ploting the diagrams and visulising the dataset
from scipy import stats #For the stastical data analysis
from scipy.stats import norm #For the normalisation of the dataset
import datetime #For the date and time opeations in the dataset

#Machine learning frame work
from sklearn import preprocessing #Load the data preprossesing tools from the sklean lib frame work
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, roc_auc_score #Load the classification repo
from sklearn.preprocessing import StandardScaler # load for the data normalizaion
scaler = StandardScaler() #store the standerd scaler into new variable
from sklearn.model_selection import train_test_split #Load the traning and testing data splitting data framwork
from imblearn.over_sampling import SMOTE# data class balancer
sm = SMOTE(random_state=2)
from sklearn.ensemble import RandomForestClassifier
```

Load dataset

```
# energy_data = pd.read_csv('/content/drive/MyDrive/Projects/Electric theft project/China_dataset/data.csv') #Call the data file
energy_data = pd.read_csv('data.csv')
```

Figure 12: Import necessary libraries for Random Forest

Pre-processing of Dataset

In pre-processing separate the labels into new variables and remove the unnecessary portions of the dataset. Process the null value using each consumer's median, and if any nulls are present in the small amount, replace them with 0. To properly format the time and date, process the dataset.

```
del energy_data['CONS_NO'] #Delete the Unnessary part from the dataset
Y_data = energy_data['FLAG'] #Seprating the lables and stored into new variable

#Process the null value with the median of the each consumer 2 year power consuption
del energy_data['FLAG']
new_energy = energy_data.T
for cols in new_energy.columns:
    a = new_energy[cols].isnull().sum()
    if a>0:
        new_energy[cols] = new_energy[cols].fillna(new_energy[cols].median())

# print('Null still aviable in the data',new_energy.isna().values.sum())
new_energy = new_energy.fillna(0)#If any null avialbe in the small amount then replace with 0

#Process dataset to set the correct format of the time and date
new_energy.sort_index()
new_energy.reset_index(inplace=True)
new_energy['index'] = new_energy['index'].astype('datetime64')
new_energy.index =new_energy['index']
del(new_energy['index'])
process_data = new_energy.sort_index().T
# print('Processed dataset',process_data)

#Call dataset in the daily powers, montly powers
daily_energy = process_data
daily_mean = new_energy.describe()
# print('\nStastical data discription on the \n',daily_mean)#Stastical dataset analysis

# Consumer monthly power consuption analysis
monthly_energy = process_data.T.resample('M').mean()#Store the Monthly power consluption in new veriable
monthly_mean = monthly_energy.describe()
# print('\n Stastical analysis of the Consumer Montly power consuption \n', monthly_mean)
```

Figure 13: Pre-processing the of dataset

Figure 14 displays the splitting of training and testing dataset fo Random Forest.

Create a set of training and testing dataset

```
#Split the traning and testing dataset
x_train, x_test, y_train, y_test = train_test_split(daily_energy, Y_data,
                                                     test_size=0.30, random_state = 32)#Daily power consption data

x_train_m,x_test_m,y_train_m,y_test_m = train_test_split(monthly_energy.T, Y_data,
                                                         test_size = 0.30, random_state = 32)# Split the Monthtly traning and te
```

Figure 14: Splitting training and testing dataset for Random Forest

Need to use techniques like SMOTE to increase the performance of the Random Forest algorithms because unbalanced data sets are frequently encountered in this practice. Figure **15** displays the function of SMOTE.

SMOTE

```
#Reguler energy data
sm_x_train, sm_y_train = sm.fit_resample(x_train, y_train)#Daily power consuption data processs with SMOT

sm_x_train_m,sm_y_train_m = sm.fit_resample(x_train_m,y_train_m)#montly power consption process with SMOT
```

Figure 15: SMOTE

Random Forest with Daily Regular Power Consumption

Figure 16 displays the Random forest implementation for Daily Power Consumption.

```
rf_clf_daily = RandomForestClassifier(random_state=80) # Asigne new variable to the random forest classifier
rf_clf_daily.fit(sm_x_train, sm_y_train)#Fit the dialy power data into the random forest model
y_pred = rf_clf_daily.predict(x_test) #Prdict the testing data

AUC = roc_auc_score(y_test, y_pred)
print('Model AUC score:',AUC)

randomforest_accuracy = accuracy_score(y_test, y_pred)*100 #random forest model accuracy
print('Model accuray:',randomforest_accuracy)
print('\nModel classification report\n',classification_report(y_test,y_pred)) #classification performance evolution
#Confusion matrix
T_n, F_p, F_n, T_p =confusion_matrix(y_test,y_pred).ravel()
#printing the consution matrix
print('\nTrue negative:',T_n,'\nFalse negative:',F_n,'\nTrue positive:',T_p,'\nFalse positive:',F_p)
#Miss classification rate
print('Model miss classficiaiton rate:',((F_p + F_n)/len(Y_data))*100)
```

```
Model AUC score: 0.5359010338624524
Model accuray: 91.4018250471995

Model classification report
               precision    recall  f1-score   support

           0       0.92      1.00      0.95     11586
           1       0.62      0.08      0.14      1126

    accuracy                           0.91     12712
   macro avg       0.77      0.54      0.55     12712
weighted avg       0.89      0.91      0.88     12712


True negative: 11533
False negative: 1040
True positive: 86
False positive: 53
Model miss classficiaiton rate: 2.5795336543000094
```

Figure 16: Random forest implementation for Daily Power Consumption

Random forest Model with the Monthly Power Consumption

Figure 17 displays the Random forest Model with the Monthly Power Consumption.

```
rf_clf_monthly = RandomForestClassifier(random_state=80) # Asigne new variable to the random forest classifier
rf_clf_monthly.fit(sm_x_train_m, sm_y_train_m)#Fit the dialy power data into the random forest model
y_pred = rf_clf_monthly.predict(x_test_m) #Prdict the testing data

AUC = roc_auc_score(y_test_m, y_pred)
print('Model AUC score:',AUC)

#Model accuracy and perfrormance on the testing dataset
randomforest_accuracy = accuracy_score(y_test_m, y_pred)*100 #random forest model accuracy
print('Model accuray:',randomforest_accuracy) #Printing the model accuracy

print(classification_report(y_test_m,y_pred)) #classification performance evolution
#Confusion matrix
T_n, F_p, F_n, T_p =confusion_matrix(y_test_m,y_pred).ravel()
#printing the consution matrix
print('\nTrue negative:',T_n,'\nFalse negative:',F_n,'\nTrue positive:',T_p,'\nFalse positive:',F_p)
#Miss classification rate
print('Model miss classficiaiton rate:',((F_p + F_n)/len(y_test_m))*100)
```

```
Model AUC score: 0.5344394180641241
Model accuray: 91.3546255506608
              precision    recall  f1-score   support

           0       0.92      1.00      0.95     11586
           1       0.60      0.07      0.13      1126

    accuracy                           0.91     12712
   macro avg       0.76      0.53      0.54     12712
weighted avg       0.89      0.91      0.88     12712


True negative: 11530
False negative: 1043
True positive: 83
False positive: 56
Model miss classficiaiton rate: 8.645374449339208
```

Figure 17: Random forest Model with the Monthly Power Consumption

4.3 K-Means

This is with the reference to the file ‘China_data_analysis_K’ present in the code artefact.

Importing required libraries

In Figure 18, all necessary libraries are imported along with the dataset which is read and stored in a data
frame.

Importing required libraries

```
import numpy as np # for the math and matrix operations
import pandas as pd # For the data loading into programm and data analysis
import matplotlib.pyplot as plt #for the ploting the diagrams and visulising the dataset
from scipy import stats #For the stastical data analysis
from scipy.stats import norm #For the normalisation of the dataset
import datetime #For the date and time opeations in the dataset
```

Import libraries for data preprocessing, K Means and other perfromace measuemnt tools (sklearn)

```
from sklearn.cluster import KMeans #Load the K Means algorithm frame work from the sklearn open source libraries
from sklearn import preprocessing #Load the data preprossesing tools from the sklean lib frame work
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, roc_auc_score #Load the classification repor
from sklearn.preprocessing import StandardScaler # load for the data normalizaion
scaler = StandardScaler() #store the standerd scaler into new variable
from sklearn.cluster import DBSCAN #Load the DBscan
```

Load the dataset from the local storage

```
# energy_data = pd.read_csv('/content/drive/MyDrive/Projects/Electric theft project/China_dataset/data.csv') #Call the data file
energy_data = pd.read_csv('data.csv')
```

Figure 18: Importing libraries for k-mean

Preprocessing of dataset

Initially, in Figure 19 the dataset imported is analyzed for structure. The Null values or NA values in the dataset is calculated. The unwanted columns are removed. The NA values are handled using median imputation.

Checking basic information of the data

```
print('The basic information on the dataset - Overall Info\n')
print(energy_data.info())
print('----------------------------------------')
print('\nThe inforamtion on the dataset - Attribute wise\n')
print(energy_data.info)
print('------------------------------------------')
print('\nNull Value Check\n')
print(energy_data.isnull().values.sum())
```

```
The basic information on the dataset - Overall Info

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42372 entries, 0 to 42371
Columns: 1036 entries, CONS_NO to 2016/9/9
dtypes: float64(1034), int64(1), object(1)
memory usage: 334.9+ MB
None
----------------------------------------

The inforamtion on the dataset - Attribute wise

<bound method DataFrame.info of                          CONS_NO  FLAG  2014/1/1  2014/1/10  2014/1/11  \
0      0387DD8A07E07FDA6271170F86AD9151     1       NaN        NaN        NaN
1      01D6177B5D4FFE0CABA9EF17DAFC2B84     1       NaN        NaN        NaN
2      4B75AC4F2D8434CFF62DB64D0BB43103     1       NaN        NaN        NaN
3      B32AC8CC6D5D805AC053557AB05F5343     1       NaN        NaN        NaN
4      EDFC78B07BA2908B3395C4EB2304665E     1      2.90       3.42       3.81
...                                 ...   ...       ...        ...        ...
42367  F1472871E1AFF49D4289564B6377D76C     0       NaN        NaN        NaN
42368  F3C8BBCD2DC26C1E0249DEEF6A4256B7     0      2.70       4.39       3.95
42369  A9A0FE83467A680FBFB0DBFC910DF227     0      0.58       0.84       1.61
42370  D9A6ADA018FA46A55D5438370456AA45     0     16.89      13.84      13.50
42371  F3406636BAD1E6E0826E8EDDC9A1BF00     0       NaN        NaN        NaN
```

Figure 19: Basic information check

```
       2014/1/12  2014/1/13  2014/1/14  2014/1/15  2014/1/16  ...  2016/9/28  \
0            NaN        NaN        NaN        NaN        NaN  ...      10.12   
1            NaN        NaN        NaN        NaN        NaN  ...       0.00   
2            NaN        NaN        NaN        NaN        NaN  ...        NaN   
3            NaN        NaN        NaN        NaN        NaN  ...       6.50   
4           4.58       3.56       4.25       3.86       3.53  ...      17.77   
...          ...        ...        ...        ...        ...  ...        ...   
42367        NaN        NaN        NaN        NaN        NaN  ...       4.25   
42368       0.00       0.00       0.00       0.00       0.00  ...       4.81   
42369       0.90       0.60       0.82       0.89       0.03  ...        NaN   
42370      14.60      14.46      12.34      15.37      17.01  ...      21.13   
42371        NaN        NaN        NaN        NaN        NaN  ...       2.80   

       2016/9/29  2016/9/3  2016/9/30  2016/9/4  2016/9/5  2016/9/6  2016/9/7  \
0           9.96     16.92       7.60     27.22     18.05     26.47     18.75   
1           0.00      0.00       0.00      0.00      0.00      0.00      0.00   
2            NaN       NaN        NaN       NaN       NaN       NaN       NaN   
3           9.99     11.78      18.59     26.80     18.57     14.59     12.82   
4          10.37     15.32      13.51     12.23     14.68     16.35     18.14   
...          ...       ...        ...       ...       ...       ...       ...   
42367       3.56      3.38       4.39      3.72      3.77      3.96      3.64   
42368       4.87      4.48       3.67      3.31      4.58      3.33      3.19   
42369       0.66      2.92       2.36      3.86      4.28      3.37      6.67   
42370      13.75     22.61      18.83     25.52     18.11     19.31     17.48   
42371       4.45      9.80       5.11     16.69     12.04      9.90      8.23   

       2016/9/8  2016/9/9  
0         17.84     14.92  
1          0.00      0.00  
2           NaN       NaN  
3         19.37     15.92  
4         18.41     17.31  
...         ...       ...  
42367      3.40      4.38  
42368      4.57      4.00  
42369      2.44      1.15  
```

Figure 20: Output data of basic information

Processing the data based on the above information

1) Remove the consumer ID information from the dataset

```
del energy_data['CONS_NO']
```

Process the null values

2) from the above stastical analysis we have found the missing or the NA values to be around 11233528 which is around the 25% of the total dataset.

This missing vlaues are due to the faulty meters or possible human error while collecting and documenting the data.

Generally, In the case of missing values or NA values we just remove it and move on, however we cannot ignore this as we are missing 25% of the data, hence we handle this using median imputation.

```
Y_data = energy_data['FLAG'] #Seprating the lables and stored into new variable

#Process the null value with the median of the each consumer 2 year power consuption
del energy_data['FLAG']
new_energy = energy_data.T
for cols in new_energy.columns:
    a = new_energy[cols].isnull().sum()
    if a>0:
        new_energy[cols] = new_energy[cols].fillna(new_energy[cols].median())

print(new_energy.isna().values.sum())
new_energy = new_energy.fillna(0)
```

```
5170
```

Figure 21: Processing of null value

Process the date and time sequence

```
new_energy.sort_index()
new_energy.reset_index(inplace=True)
new_energy['index'] = new_energy['index'].astype('datetime64')
new_energy.index =new_energy['index']
del(new_energy['index'])
process_data = new_energy.sort_index().T
print(process_data)
```

```
index  2014-01-01  2014-01-02  2014-01-03  2014-01-04  2014-01-05  2014-01-06  \
0           9.770       9.770       9.770       9.770       9.770       9.770
1           0.000       0.000       0.000       0.000       0.000       0.000
2           8.900       8.900       8.900       8.900       8.900       8.900
3          12.735      12.735      12.735      12.735      12.735      12.735
4           2.900       5.640       6.990       3.320       3.610       5.350
...           ...         ...         ...         ...         ...         ...
42367       4.260       4.260       4.260       4.260       4.260       4.260
42368       2.700       0.000       0.000       5.720       6.050       5.810
42369       0.580       1.160       0.920       0.980       1.540       1.380
42370      16.890      15.150      19.280      17.190      16.800      17.480
42371       8.090       8.090       8.090       8.090       8.090       8.090
```

Figure 22: Process data and time sequence

Data Preparation

In Figure 23, The data is prepared for applying the K-means model. Data preparation steps include scaling of the data. Data here is aggregated into daily data and monthly data. Both these aggregated data are scaled.

Prepare the dataset for the K-Means model and test the model

```
#Regular power data at various intervals
daily_energy_array = np.array(process_data) #simple daily power consumption data of each consumer
monthly_energy_array = np.array(monthly_energy.T) #average power consumption of each consumer on monthly basis

# Standard and mean power data at various interval
daily_mean_array = np.array(daily_mean.loc[['mean','std']].T)#Mean and standard deavation of the daily power consumption
montly_mean_array = np.array(monthly_mean.loc[['mean','std']].T)#Mean and STD of the montly avg power consumption

#Scale the daily energy dataset
scale_daily_energy = scaler.fit_transform(daily_energy_array.T)
scale_daily_energy = scale_daily_energy.T#Daily energy consption dataset

#Scale the montly energy dataset
scale_monthly_energy = scaler.fit_transform(monthly_energy_array.T)
scale_monthly_energy = scale_monthly_energy.T#Daily energy consption dataset

# #The array input to the ML model
# print('Daily power energy array:\n', daily_energy_array)
# print('\n weekly power energy array:\n', weekly_energy_array)
# print('\n montly power energy array:\n', monthly_energy_array)
# #mean data
# print('\n Dayily mean power energy array:\n', daily_mean_array)
# print('\n weekly mean power energy array:\n', weekly_mean_array)
# print('\n montly mean power energy array:\n', montly_mean_array)
# print('\n anaulay mean power energy array:\n', anual_mean_array)
```

Figure 23: Preparation of data for k-Mean

Model Building – K means

Figure 23 includes K-means applying on regular power consumption dataset on daily readings as well as monthly readings.

K-means model with the various regular power consumption dataset

```
#Prediction model for the daily power consuption
K_means_dialy = KMeans(n_clusters = 2, random_state=342).fit(daily_energy_array)

AUC = roc_auc_score(Y_data, K_means_dialy.labels_)
print('Model AUC score:',AUC)

KMeans_accuracy = accuracy_score(Y_data, K_means_dialy.labels_)*100 # model accuracy
print('KMeans model for daily power consumption based  clustering accuracy', KMeans_accuracy)
#Model clustering perfromance
print('KMeans Model Classification Report')
print(classification_report(Y_data, K_means_dialy.labels_))

#Confusion matrix
T_n, F_p, F_n, T_p =confusion_matrix(Y_data, K_means_dialy.labels_).ravel()
#printing the consution matrix
print('\nTrue negative:',T_n,'\nFalse negative:',F_n,'\nTrue positive:',T_p,'\nFalse positive:',F_p)
#Misclassification rate
print('Model misclassficiaiton rate:',((F_p + F_n)/len(K_means_dialy.labels_))*100)
```

```
Model AUC score: 0.49998709910467787
KMeans model for daily power consumption based  clustering accuracy 91.46606249409987
KMeans Model Classification Report
              precision    recall  f1-score   support

           0       0.91      1.00      0.96     38757
           1       0.00      0.00      0.00      3615

    accuracy                           0.91     42372
   macro avg       0.46      0.50      0.48     42372
weighted avg       0.84      0.91      0.87     42372


True negative: 38756
False negative: 3615
True positive: 0
False positive: 1
Model miss classficiaiton rate: 8.533937505900123
```

Figure 23: K mean for daily power consumption data

```
#Prediction model for the monthly power consuption
K_means_monthly = KMeans(n_clusters = 2, random_state=435).fit(monthly_energy_array)

AUC = roc_auc_score(Y_data, K_means_monthly.labels_)
print('Model AUC score:',AUC)

KMeans_accuracy = accuracy_score(Y_data, K_means_monthly.labels_)*100 # model accuracy
print('KMeans model for monthly power consuption based  clustering accuracy', KMeans_accuracy)
#Model clusring perfromance
print('KMeans Model Classification Report')
print(classification_report(Y_data, K_means_monthly.labels_))

#Confusion matrix
T_n, F_p, F_n, T_p =confusion_matrix(Y_data, K_means_monthly.labels_).ravel()
#printing the consution matrix
print('\nTrue negative:',T_n,'\nFalse negative:',F_n,'\nTrue positive:',T_p,'\nFalse positive:',F_p)
#Misclassification rate
print('Model misclassficiaiton rate:',((F_p + F_n)/len(K_means_monthly.labels_))*100)
```

```
Model AUC score: 0.5001383125864454
KMeans model for monthly power consuption based  clustering accuracy 91.47078259227793
KMeans Model Classification Report
              precision    recall  f1-score   support

           0       0.91      1.00      0.96     38757
           1       1.00      0.00      0.00      3615

    accuracy                           0.91     42372
   macro avg       0.96      0.50      0.48     42372
weighted avg       0.92      0.91      0.87     42372


True negative: 38757
False negative: 3614
True positive: 1
False positive: 0
Model misclassficiaiton rate: 8.529217407722081
```

Figure 24: K mean for Monthly consumption data

4.4 Decision Tree Model

This is with the reference to the file ‘Decision_tree’ present in the code artefact

Importing required libraries and datasets.

In Figure 25, the dataset is imported, viewed, and all necessary libraries have been imported, as seen below.

```
import numpy as np # for the math and matrix operations
import pandas as pd # For the data loading into programm and data analysis
import matplotlib.pyplot as plt #for the ploting the diagrams and visulising the dataset
from scipy import stats #For the stastical data analysis
from scipy.stats import norm #For the normalisation of the dataset
import datetime #For the date and time opeations in the dataset

#Machine learning frame work
from sklearn import preprocessing #Load the data preprossesing tools from the sklean lib frame work
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, roc_auc_score #Load the classification re
from sklearn.preprocessing import StandardScaler # Load for the data normalizaion
scaler = StandardScaler() #store the standerd scaler into new variable
from sklearn.model_selection import train_test_split #Load the traning and testing data splitting data framwork
from imblearn.over_sampling import SMOTE# data class balancer
sm = SMOTE(random_state=42)
from sklearn.tree import DecisionTreeClassifier
```

Figure 25: Import libraries for Decision Tree

Pre-processing of Dataset

In Figure 26 pre-processing separates the labels into new variables and remove the unnecessary portions of the dataset. Process the null value using each consumer's median, and if any nulls are present in the small amount, replace them with 0. To properly format the time and date, process the dataset.

```python
del energy_data['CONS_NO'] #Delet the Unnessary part from the dataset
Y_data = energy_data['FLAG'] #Seprating the lables and stored into new variable

#Process the null value with the median of the each consumer 2 year power consuption
del energy_data['FLAG']
new_energy = energy_data.T
for cols in new_energy.columns:
    a = new_energy[cols].isnull().sum()
    if a>0:
        new_energy[cols] = new_energy[cols].fillna(new_energy[cols].median())

# print('Null still aviable in the data',new_energy.isna().values.sum())
new_energy = new_energy.fillna(0)#If any null avialbe in the small amount then replace with 0

#Process dataset to set the correct format of the time and date
new_energy.sort_index()
new_energy.reset_index(inplace=True)
new_energy['index'] = new_energy['index'].astype('datetime64')
new_energy.index =new_energy['index']
del(new_energy['index'])
process_data = new_energy.sort_index().T
# print('Processed dataset',process_data)

#Call dataset in the daily powerspower consumption
daily_energy = process_data
daily_mean = new_energy.describe()
# print('\nStastical data discription on the \n',daily_mean)#Stastical dataset analysis


# Consumer monthly power consuption analysis
monthly_energy = process_data.T.resample('M').mean()#Store the Monthly power consluption in new veriable
monthly_mean = monthly_energy.describe()
```

Figure 26: Pre-processing for Decision Tree

Split the training and testing dataset

```python
#Split the traning and testing dataset
x_train, x_test, y_train, y_test = train_test_split(daily_energy, Y_data,
                                                    test_size=0.30, random_state = 32)#Daily power consption data


x_train_m,x_test_m,y_train_m,y_test_m = train_test_split(monthly_energy.T, Y_data,
                                                         test_size = 0.30, random_state = 32)# Split the Monthtly traning and
```

Figure 27: Split the training and testing dataset

Need to use techniques like SMOTE to increase the performance of the Random Forest algorithms because unbalanced data sets are frequently encountered in this practice. Figure 28 displays the SMOTE function.

SMOTE

```python
#Applying smote
sm_x_train, sm_y_train = sm.fit_resample(x_train, y_train)#Daily power consuption data processs with SMOTE

sm_x_train_m,sm_y_train_m = sm.fit_resample(x_train_m,y_train_m)#montly power consption process with SMOTE
```

Figure 28: SMOTE

Decision Tree for Daily Regular Power Consumption

Figure 29 displays the Decision Tree for Daily Power Consumption.

Decision Tree Implementation for Daily Regular Power Consumption

```
# Decision tree with entropy for daily
Dt_clf_entropy_daily = DecisionTreeClassifier(
            criterion = "entropy", random_state = 100,
            max_depth = 10, min_samples_leaf = 5)

    # Performing training
Dt_clf_entropy_daily.fit(sm_x_train, sm_y_train)
y_pred = Dt_clf_entropy_daily.predict(x_test)

AUC = roc_auc_score(y_test, y_pred)
print('Model AUC score:',AUC)

#Model accuracy and perfromance on the testing dataset
decisionTree_accuracy = accuracy_score(y_test, y_pred)*100 #random forest model accuracy
print('Model accuray:',decisionTree_accuracy) #Printing the model accuracy

# model classification report
print(classification_report(y_test,y_pred)) #classification performance evolution
#Confusion matrix
T_n, F_p, F_n, T_p =confusion_matrix(y_test,y_pred).ravel()
#printing the consution matrix
print('\nTrue negative:',T_n,'\nFalse negative:',F_n,'\nTrue positive:',T_p,'\nFalse positive:',F_p)
#Misclassification rate
print('Model misclassficiaiton rate:',((F_p + F_n)/len(y_test))*100)
```

```
Model AUC score: 0.5978066104770902
Model accuray: 57.890182504719945
```

Figure 29: Decision Tree for Daily Power Consumption

Decision Tree for Monthly Power Consumption

Figure 30 displays the Decision Tree for Monthly Power Consumption.

```
# Decision tree with entropy for monthly
Dt_clf_entropy_monthly = DecisionTreeClassifier(
            criterion = "entropy", random_state = 100,
            max_depth = 10, min_samples_leaf = 5)

    # Performing training
Dt_clf_entropy_monthly.fit(sm_x_train_m, sm_y_train_m)
y_pred_monthly = Dt_clf_entropy_monthly.predict(x_test_m)

AUC = roc_auc_score(y_test_m, y_pred_monthly)
print('Model AUC score:',AUC)

#Model accuracy and perfromance on the testing dataset
randomforest_accuracy = accuracy_score(y_test_m, y_pred_monthly)*100 #random forest model accuracy
print('Model accuray:',randomforest_accuracy) #Printing the model accuracy

# model classification report
print(classification_report(y_test_m,y_pred_monthly)) #classification performance evolution
#Confusion matrix
T_n, F_p, F_n, T_p =confusion_matrix(y_test_m,y_pred_monthly).ravel()
#printing the consution matrix
print('\nTrue negative:',T_n,'\nFalse negative:',F_n,'\nTrue positive:',T_p,'\nFalse positive:',F_p)
#Misclassification rate
print('Model misclassficiaiton rate:',((F_p + F_n)/len(y_test_m))*100)
```

```
Model AUC score: 0.6260369975523224
Model accuray: 68.29767149150409
```

Figure 30: Decision Tree for Monthly Power Consumption