Loading Libraries

```
#importing libraries

import matplotlib
import matplotlib.pyplot as plt
import random

import pandas as pd
import numpy as np

from tqdm import tqdm

from sklearn.decomposition import PCA
from sklearn.manifold import TSNE

from sklearn.model_selection import train_test_split

import pickle

!pip install seaborn --upgrade

import seaborn as sns

from sklearn.metrics import f1_score
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score


from sklearn.metrics import confusion_matrix

print(sns.__version__)
```

```
Requirement already up-to-date: seaborn in /usr/local/lib/python3.6/dist-packages (0.11.0)
Requirement already satisfied, skipping upgrade: pandas>=0.23 in /usr/local/lib/python3.6/dist-packages (from seaborn) (1.1.2)
Requirement already satisfied, skipping upgrade: numpy>=1.15 in /usr/local/lib/python3.6/dist-packages (from seaborn) (1.18.5)
Requirement already satisfied, skipping upgrade: scipy>=1.0 in /usr/local/lib/python3.6/dist-packages (from seaborn) (1.4.1)
Requirement already satisfied, skipping upgrade: matplotlib>=2.2 in /usr/local/lib/python3.6/dist-packages (from seaborn) (3.2.2)
Requirement already satisfied, skipping upgrade: python-dateutil>=2.7.3 in /usr/local/lib/python3.6/dist-packages (from pandas>=0.23->seaborn) (2.8.1)
Requirement already satisfied, skipping upgrade: pytz>=2017.2 in /usr/local/lib/python3.6/dist-packages (from pandas>=0.23->seaborn) (2018.9)
Requirement already satisfied, skipping upgrade: kiwisolver>=1.0.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib>=2.2->seaborn) (1.2.0)
Requirement already satisfied, skipping upgrade: cycler>=0.10 in /usr/local/lib/python3.6/dist-packages (from matplotlib>=2.2->seaborn) (0.10.0)
Requirement already satisfied, skipping upgrade: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib>=2.2->seaborn) (2.4.7)
Requirement already satisfied, skipping upgrade: six>=1.5 in /usr/local/lib/python3.6/dist-packages (from python-dateutil>=2.7.3->pandas>=0.23->seaborn) (1.15.0)
0.11.0
```

```
#Loading data
```

Loading data

```
#!curl --header 'Host: doc-0k-3c-docs.googleusercontent.com' --user-agent 'Mozilla/5.0 (X11; Linux x86_64; rv:81.0) Gecko/20100101 Firefox/81.0' --header 'Accept: text/html,appl
#!curl --header 'Host: doc-0g-3c-docs.googleusercontent.com' --user-agent 'Mozilla/5.0 (X11; Linux x86_64; rv:81.0) Gecko/20100101 Firefox/81.0' --header 'Accept: text/html,appl
#!curl --header 'Host: doc-0k-3c-docs.googleusercontent.com' --user-agent 'Mozilla/5.0 (X11; Linux x86_64; rv:81.0) Gecko/20100101 Firefox/81.0' --header 'Accept: text/html,appl
#!curl --header 'Host: doc-0o-3c-docs.googleusercontent.com' --user-agent 'Mozilla/5.0 (X11; Linux x86_64; rv:81.0) Gecko/20100101 Firefox/81.0' --header 'Accept: text/html,appl
#!curl --header 'Host: doc-08-3c-docs.googleusercontent.com' --user-agent 'Mozilla/5.0 (X11; Linux x86_64; rv:81.0) Gecko/20100101 Firefox/81.0' --header 'Accept: text/html,appl



#Loading Scaler
filename = '/content/scalar.pkl'

with open(filename, 'rb') as f:
    scaler = pickle.load(f)

print(scaler)

print("*"*50)

#Loading varriables
filename = '/content/varriables.pickle'

with open(filename, 'rb') as f:
    high_nan_features, median, time_based_sensor, bottom_n_features, useless_features = pickle.load(f)

print("high_nan_features = ",high_nan_features)
print("median = ",median)
print("time_based_sensor = ",time_based_sensor)
print("bottom_n_features = ",bottom_n_features)
print("useless_features = ",useless_features)

print("\n")
print("*"*500)
print("\n")

#Loading lables
filename = '/content/labels.pickle'

with open(filename, 'rb') as f:
    labels = pickle.load(f)

print("Lables = ",labels)

print("\n")
print("*"*500)
print("\n")

#Loading model
filename = '/content/Best_Model.sav'

with open(filename, 'rb') as f:
    best_model = pickle.load(f)

print("Best model = ",best_model)

print("\n")
```

```
print("*"*500)
print("\n")

#Loading test data

df = pd.read_csv("/content/Test.csv")
df.head()
```

```
StandardScaler(copy=True, with_mean=True, with_std=True)
**************************************************
high_nan_features =  ['sensor2_measure', 'sensor38_measure', 'sensor39_measure', 'sensor40_measure', 'sensor41_measure', 'sensor42_measure', 'sensor43_measure', 'sensor68_m
median =  sensor1_measure                 30755.0
sensor3_measure                   152.0
sensor4_measure                   126.0
sensor5_measure                     0.0
sensor6_measure                     0.0
                                  ...
sensor105_histogram_bin7_nan        0.0
sensor105_histogram_bin8_nan        0.0
sensor105_histogram_bin9_nan        0.0
sensor106_measure_nan               0.0
sensor107_measure_nan               0.0
Length: 332, dtype: float64
time_based_sensor =  [['sensor7_histogram_bin0', 'sensor7_histogram_bin1', 'sensor7_histogram_bin2', 'sensor7_histogram_bin3', 'sensor7_histogram_bin4', 'sensor7_histogram_
bottom_n_features =  {'sensor25_histogram_bin9': 0.000404517327265069, 'sensor4_measure': -0.0005747918610703464, 'sensor56_measure': -0.0005748712717671971, 'sensor5_measu
useless_features =  ['sensor7_histogram_bin5', 'sensor7_histogram_bin6', 'sensor8_measure', 'sensor12_measure', 'sensor14_measure', 'sensor15_measure', 'sensor16_measure',


******************************************************************************************************************************************************************************


Lables =  [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0


******************************************************************************************************************************************************************************


Best model =  StackingCVClassifier(classifiers=[DecisionTreeClassifier(ccp_alpha=0.0,
                                                        class_weight='balanced',
                                                        criterion='gini',
                                                        max_depth=30,
                                                        max_features=None,
                                                        max_leaf_nodes=None,
                                                        min_impurity_decrease=0.0,
                                                        min_impurity_split=None,
                                                        min_samples_leaf=1,
                                                        min_samples_split=30,
                                                        min_weight_fraction_leaf=0.0,
                                                        presort='deprecated',
                                                        random_state=None,
                                                        splitter='best'),
                                  XGBClassifie...
                                                  max_depth=50,
                                                  min_child_weight=1,
                                                  missing=nan,
                                                  n_estimators=3000, n_jobs=1,
                                                  nthread=None,
                                                  objective='binary:logistic',
                                                  random_state=0, reg_alpha=0,
                                                  reg_lambda=1,
                                                  scale_pos_weight=1,
                                                  seed=None, silent=None,
                                                  subsample=0.7, verbosity=1),
                     shuffle=True, store_train_meta_features=False,
                     stratify=True, use_clones=True,
                     use_features_in_secondary=True, use_probas=False,
                     verbose=0)

******************************************************************************************************************************************************************************

/usr/local/lib/python3.6/dist-packages/sklearn/externals/six.py:31: FutureWarning: The module is deprecated in version 0.21 and will be removed in version 0.23 since we've
  "(https://pypi.org/project/six/).", FutureWarning)
```

	id	sensor1_measure	sensor2_measure	sensor3_measure	sensor4_measure	sensor5_measure	sensor6_measure	sensor7_histogram_bin0	sensor7_histogram_bin1	sensor7_histog
0	51065	40966	na	190	130	0	0	0	0	
1	42771	32	na	0	na	0	0	0	0	
2	54788	868	2	64	40	0	0	0	0	
3	22822	52160	na	638	552	0	0	0	0	
4	29075	419854	na	na	na	na	na	0	0	

5 rows × 171 columns

Defining featurisation functions

```
def mean(a,b,c,d,e,f,g,h,i,j):
    list_ = [a,b,c,d,e,f,g,h,i,j]
    return np.mean(list_)


def min_(a,b,c,d,e,f,g,h,i,j):
    list_ = [a,b,c,d,e,f,g,h,i,j]
    return min(list_)


def max_(a,b,c,d,e,f,g,h,i,j):
    list_ = [a,b,c,d,e,f,g,h,i,j]
    return max(list_)
```

Final function 1

```
def final_fun_1(df):
    df = df.replace('na', np.NaN)
    df = df.astype("float32")
    df = df.drop(["id"],axis=1)

    coloumns = df.columns
```

```
    for coloumn in coloumns:
        df[coloumn + "_nan"] = [1.0 if np.isnan(x) else 0.0 for x in df[coloumn]]

    df = df.drop(high_nan_features,axis=1)

    df = df.fillna(median)


    for i in range(0,len(time_based_sensor)):
        df[time_based_sensor[i][0].split("_")[0] + "_mean"] = df.apply(lambda row : mean(row[time_based_sensor[i][0]] , row[time_based_sensor[i][1]] , row[time_based_sensor[i][2
                                                                                     row[time_based_sensor[i][3]] , row[time_based_sensor[i][4]] , row[time_based_sensor[i][5]
                                                                                     row[time_based_sensor[i][6]] , row[time_based_sensor[i][7]] , row[time_based_sensor[i][8]
                                                                                     row[time_based_sensor[i][9]] ) , axis = 1)


        df[time_based_sensor[i][0].split("_")[0] + "_min"]  = df.apply(lambda row : min_(row[time_based_sensor[i][0]] , row[time_based_sensor[i][1]] , row[time_based_sensor[i][2
                                                                                     row[time_based_sensor[i][3]] , row[time_based_sensor[i][4]] , row[time_based_sensor[i][5]
                                                                                     row[time_based_sensor[i][6]] , row[time_based_sensor[i][7]] , row[time_based_sensor[i][8]
                                                                                     row[time_based_sensor[i][9]] ) , axis = 1)

        df[time_based_sensor[i][0].split("_")[0] + "_max"]  = df.apply(lambda row : max_(row[time_based_sensor[i][0]] , row[time_based_sensor[i][1]] , row[time_based_sensor[i][2
                                                                                     row[time_based_sensor[i][3]] , row[time_based_sensor[i][4]] , row[time_based_sensor[i][5]
                                                                                     row[time_based_sensor[i][6]] , row[time_based_sensor[i][7]] , row[time_based_sensor[i][8]
                                                                                     row[time_based_sensor[i][9]] ) , axis = 1)


    df = df.drop(bottom_n_features.keys(),axis=1)
    df = df.drop(useless_features , axis=1)
    df = scaler.transform(df)

    return best_model.predict(df)

Predicted_Labels = final_fun_1(df)

print("Predicted_Labels = ",Predicted_Labels)
```

```
Predicted_Labels =  [0. 0. 0. ... 0. 0. 0.]
```

Final Function 2

```
def final_fun_2(df,labels):
    labels = np.array(labels)
    df = df.replace('na', np.NaN)
    df = df.astype("float32")
    df = df.drop(["id"],axis=1)

    coloumns = df.columns

    for coloumn in coloumns:
        df[coloumn + "_nan"] = [1.0 if np.isnan(x) else 0.0 for x in df[coloumn]]

    df = df.drop(high_nan_features,axis=1)

    df = df.fillna(median)

    for i in range(0,len(time_based_sensor)):
        df[time_based_sensor[i][0].split("_")[0] + "_mean"] = df.apply(lambda row : mean(row[time_based_sensor[i][0]] , row[time_based_sensor[i][1]] , row[time_based_sensor[i][2
                                                                                     row[time_based_sensor[i][3]] , row[time_based_sensor[i][4]] , row[time_based_sensor[i][5]
                                                                                     row[time_based_sensor[i][6]] , row[time_based_sensor[i][7]] , row[time_based_sensor[i][8]
                                                                                     row[time_based_sensor[i][9]] ) , axis = 1)


        df[time_based_sensor[i][0].split("_")[0] + "_min"]  = df.apply(lambda row : min_(row[time_based_sensor[i][0]] , row[time_based_sensor[i][1]] , row[time_based_sensor[i][2
                                                                                     row[time_based_sensor[i][3]] , row[time_based_sensor[i][4]] , row[time_based_sensor[i][5]
                                                                                     row[time_based_sensor[i][6]] , row[time_based_sensor[i][7]] , row[time_based_sensor[i][8]
                                                                                     row[time_based_sensor[i][9]] ) , axis = 1)

        df[time_based_sensor[i][0].split("_")[0] + "_max"]  = df.apply(lambda row : max_(row[time_based_sensor[i][0]] , row[time_based_sensor[i][1]] , row[time_based_sensor[i][2
                                                                                     row[time_based_sensor[i][3]] , row[time_based_sensor[i][4]] , row[time_based_sensor[i][5]
                                                                                     row[time_based_sensor[i][6]] , row[time_based_sensor[i][7]] , row[time_based_sensor[i][8]
                                                                                     row[time_based_sensor[i][9]] ) , axis = 1)


    df = df.drop(bottom_n_features.keys(),axis=1)
    df = df.drop(useless_features , axis=1)
    df = scaler.transform(df)



    return f1_score(labels , best_model.predict(df))

F1_Score = final_fun_2(df,labels)

print("F1 Score = ",F1_Score)
```

```
F1 Score =  0.8146214099216711
```