```
#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


!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,appli
!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,appli
!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,appli

!unzip /content/falls.zip
!pip install seaborn --upgrade

import seaborn as sns

print(sns.__version__)
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 20021  100 20021    0     0  93120      0 --:--:-- --:--:-- --:--:-- 93120
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  6100  100  6100    0     0  28240      0 --:--:-- --:--:-- --:--:-- 28240
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 21.3M    0 21.3M    0     0  12.2M      0 --:--:--  0:00:01 --:--:-- 12.2M
Archive:  /content/falls.zip
   creating: falls/
  inflating: falls/equip_failures_test_set.csv
  inflating: falls/equip_failures_training_set.csv
Requirement already up-to-date: seaborn in /usr/local/lib/python3.6/dist-packages (0.11.0)
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: pandas>=0.23 in /usr/local/lib/python3.6/dist-packages (from seaborn) (1.1.2)
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: numpy>=1.15 in /usr/local/lib/python3.6/dist-packages (from seaborn) (1.18.5)
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: 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: python-dateutil>=2.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib>=2.2->seaborn) (2.8.1)
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: pytz>=2017.2 in /usr/local/lib/python3.6/dist-packages (from pandas>=0.23->seaborn) (2018.9)
Requirement already satisfied, skipping upgrade: six>=1.5 in /usr/local/lib/python3.6/dist-packages (from python-dateutil>=2.1->matplotlib>=2.2->seaborn) (1.15.0)
0.11.0
```

Loading scaler

```
#Loading Scaler

filename = '/content/scalar.pkl'


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

print(scaler)
```

```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

Loading varriables

```
#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)
```

```
high_nan_features =  ['sensor2_measure', 'sensor38_measure', 'sensor39_measure', 'sensor40_measure', 'sensor41_measure', 'sensor42_measure', 'sensor43_measure', 'sensor68_r
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',
```

Loading dataframe

```
#Loading dataframe
df = pd.read_csv("/content/falls/equip_failures_training_set.csv")

df.head()
```

```
df.head()
```

	id	target	sensor1_measure	sensor2_measure	sensor3_measure	sensor4_measure	sensor5_measure	sensor6_measure	sensor7_histogram_bin0	sensor7_histogram_bin1	sensor7_h
0	1	0	76698	na	2130706438	280	0	0	0	0	
1	2	0	33058	na	0	na	0	0	0	0	
2	3	0	41040	na	228	100	0	0	0	0	
3	4	0	12	0	70	66	0	10	0	0	
4	5	0	60874	na	1368	458	0	0	0	0	

5 rows × 172 columns

Replace na with np.nan

```
"""Instead of nan value we have na, so we will replace na with np.nan"""

df = df.replace('na', np.NaN)
df.head()
```

	id	target	sensor1_measure	sensor2_measure	sensor3_measure	sensor4_measure	sensor5_measure	sensor6_measure	sensor7_histogram_bin0	sensor7_histogram_bin1	sensor7_h
0	1	0	76698	NaN	2130706438	280	0	0	0	0	
1	2	0	33058	NaN	0	NaN	0	0	0	0	
2	3	0	41040	NaN	228	100	0	0	0	0	
3	4	0	12	0	70	66	0	10	0	0	
4	5	0	60874	NaN	1368	458	0	0	0	0	

5 rows × 172 columns

Change data-type of dataframe

```
"We could see that few coloumns are of int type, and other are of object type,So for using data we need to make them float data type"

df = df.astype("float32")
df.dtypes
```

```
id                          float32
target                      float32
sensor1_measure             float32
sensor2_measure             float32
sensor3_measure             float32
                             ...
sensor105_histogram_bin7    float32
sensor105_histogram_bin8    float32
sensor105_histogram_bin9    float32
sensor106_measure           float32
sensor107_measure           float32
Length: 172, dtype: object
```

Drop useless coloumn from feature

```
"""id coloumn is just index, we don't need it , so we will drop it"""
df = df.drop(["id"],axis=1)
df.head()
```

	target	sensor1_measure	sensor2_measure	sensor3_measure	sensor4_measure	sensor5_measure	sensor6_measure	sensor7_histogram_bin0	sensor7_histogram_bin1	sensor7_hist
0	0.0	76698.0	NaN	2.130706e+09	280.0	0.0	0.0	0.0	0.0	
1	0.0	33058.0	NaN	0.000000e+00	NaN	0.0	0.0	0.0	0.0	
2	0.0	41040.0	NaN	2.280000e+02	100.0	0.0	0.0	0.0	0.0	
3	0.0	12.0	0.0	7.000000e+01	66.0	0.0	10.0	0.0	0.0	
4	0.0	60874.0	NaN	1.368000e+03	458.0	0.0	0.0	0.0	0.0	

5 rows × 171 columns

Train(D1) , validation(D2) and test(D_test) split

```
y = df["target"].tolist()
df_ = df.drop(["target"],axis=1)


D1 , D_test , y_1 , y_test = train_test_split( df_ , y , test_size=0.30, stratify = y ,  random_state=42)

D2 , D_test , y_2 , y_test = train_test_split( D_test , y_test , test_size=0.66, stratify = y_test ,  random_state=42)


print("train = ",D1.shape)
print("validation = ",D2.shape)
print("test = ",D_test.shape)
```

```
train =  (42000, 170)
validation =  (6120, 170)
test =  (11880, 170)
```

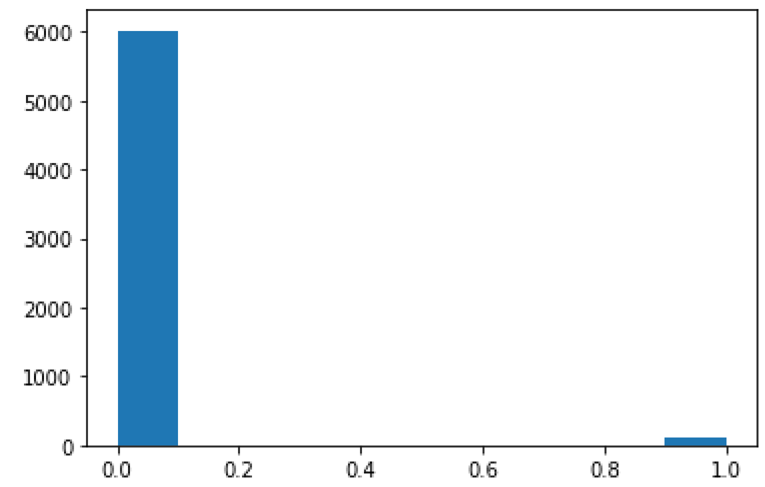
```
plt.hist(y_1)
```

```
(array([41300.,     0.,     0.,     0.,     0.,     0.,     0.,     0.,
            0.,   700.]),
 array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ]),
```

```
<a list of 10 Patch objects>)
```

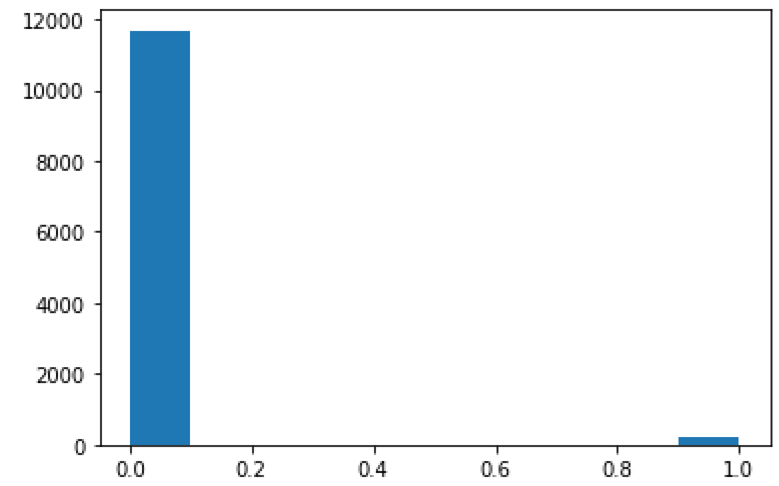

```
plt.hist(y_2)
```

```
(array([6018.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,
          102.]),
 array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ]),
 <a list of 10 Patch objects>)
```




```
plt.hist(y_test)
```

```
(array([11682.,     0.,     0.,     0.,     0.,     0.,     0.,     0.,
            0.,   198.]),
 array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ]),
 <a list of 10 Patch objects>)
```



For each feature create new feature, that tells presence of nan, because nan values also contains some information

```
coloumns = D1.columns

#Train
for coloumn in tqdm(coloumns):
    D1[coloumn + "_nan"] = [1.0 if np.isnan(x) else 0.0 for x in D1[coloumn]]


#Validation
for coloumn in tqdm(coloumns):
    D2[coloumn + "_nan"] = [1.0 if np.isnan(x) else 0.0 for x in D2[coloumn]]


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

```
  0%|          | 0/170 [00:00<?, ?it/s]/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  """
100%|██████████| 170/170 [00:11<00:00, 14.81it/s]
100%|██████████| 170/170 [00:01<00:00, 89.26it/s]
100%|██████████| 170/170 [00:03<00:00, 50.33it/s]
```

Drop features with more than 50% nan values

```
#Train
D1 = D1.drop(high_nan_features,axis=1)

#Validation
D2 = D2.drop(high_nan_features,axis=1)

#Test
D_test = D_test.drop(high_nan_features,axis=1)
```

```
D1.head()
```

sensor1 measure sensor3 measure sensor4 measure sensor5 measure sensor6 measure sensor7 histogram bin0 sensor7 histogram bin1 sensor7 histogram bin2 sensor7 h

Replace nan with median of that coloumn , because values of each feature is either very low or very high, replacing nan with mean is not sensible at all

9933	18.0	32.0	30.0	0.0	0.0	0.0	0.0	0.0

```
#Train
D1 = D1.fillna(median)


#Validation
D2 = D2.fillna(median)                                                      #Here we are filling test nan values with, train median


#Test
D_test = D_test.fillna(median)                                              #Here we are filling test nan values with, train median



D_test.head()
```

	sensor1_measure	sensor3_measure	sensor4_measure	sensor5_measure	sensor6_measure	sensor7_histogram_bin0	sensor7_histogram_bin1	sensor7_histogram_bin2	sensor7_h
39062	40868.0	2.080000e+02	170.0	0.0	0.0	0.0	0.0	0.0	
20915	46282.0	2.130706e+09	2500.0	0.0	0.0	20614.0	96250.0	174206.0	
30989	39552.0	0.000000e+00	126.0	0.0	0.0	0.0	0.0	0.0	
6964	1436.0	6.800000e+01	66.0	0.0	0.0	0.0	0.0	0.0	
44837	33342.0	3.240000e+02	304.0	0.0	0.0	0.0	0.0	0.0	

5 rows × 332 columns

We have 100 simple sensor, and 7 time based sensor. Here we will extract min, max and mean from those time based sensors

```
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_)



#Train
for i in tqdm(range(0,len(time_based_sensor))):
    D1[time_based_sensor[i][0].split("_")[0] + "_mean"] = D1.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)


    D1[time_based_sensor[i][0].split("_")[0] + "_min"]  = D1.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)

    D1[time_based_sensor[i][0].split("_")[0] + "_max"]  = D1.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)




#Validation
for i in tqdm(range(0,len(time_based_sensor))):
    D2[time_based_sensor[i][0].split("_")[0] + "_mean"] = D2.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)


    D2[time_based_sensor[i][0].split("_")[0] + "_min"]  = D2.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)

    D2[time_based_sensor[i][0].split("_")[0] + "_max"]  = D2.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)




#Test
for i in tqdm(range(0,len(time_based_sensor))):
    D_test[time_based_sensor[i][0].split("_")[0] + "_mean"] = D_test.apply(lambda row : mean(row[time_based_sensor[i][0]] , row[time_based_sensor[i][1]] , row[time_based_sensor[
                                                                                     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)


    D_test[time_based_sensor[i][0].split("_")[0] + "_min"] = D_test.apply(lambda row : min_(row[time_based_sensor[i][0]] , row[time_based_sensor[i][1]] , row[time_based_sensor[i
                                                                                     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)

    D_test[time_based_sensor[i][0].split("_")[0] + "_max"] = D_test.apply(lambda row : max_(row[time_based_sensor[i][0]] , row[time_based_sensor[i][1]] , row[time_based_sensor[i
                                                                                     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[1][9]] ) , axis = 1)

D_test.head()
```

```
100%|██████████| 7/7 [00:43<00:00,  6.28s/it]
100%|██████████| 7/7 [00:06<00:00,  1.08it/s]
100%|██████████| 7/7 [00:12<00:00,  1.78s/it]
```

	sensor1_measure	sensor3_measure	sensor4_measure	sensor5_measure	sensor6_measure	sensor7_histogram_bin0	sensor7_histogram_bin1	sensor7_histogram_bin2	sensor7_hi
39062	40868.0	2.080000e+02	170.0	0.0	0.0	0.0	0.0	0.0	
20915	46282.0	2.130706e+09	2500.0	0.0	0.0	20614.0	96250.0	174206.0	
30989	39552.0	0.000000e+00	126.0	0.0	0.0	0.0	0.0	0.0	
6964	1436.0	6.800000e+01	66.0	0.0	0.0	0.0	0.0	0.0	
44837	33342.0	3.240000e+02	304.0	0.0	0.0	0.0	0.0	0.0	

5 rows × 353 columns

Removing all the features which are least correlated to our target

```
#Train
D1 = D1.drop(bottom_n_features.keys(),axis=1)

#Validation
D2 = D2.drop(bottom_n_features.keys(),axis=1)

#test
D_test = D_test.drop(bottom_n_features.keys(),axis=1)
```

Removing all the intercorrelated features

```
#Train
D1 = D1.drop(useless_features , axis=1)

#Validation
D2 = D2.drop(useless_features , axis=1)

#Test
D_test = D_test.drop(useless_features , axis=1)
```

```
print("train_size = ",D1.shape)
print("validation_size = ",D2.shape)
print("test_size = ",D_test.shape)
```

```
train_size =  (42000, 232)
validation_size =  (6120, 232)
test_size =  (11880, 232)
```

Feature Scaling

```
#Train
D1 = scaler.transform(D1)
```

```
#Train
D2 = scaler.transform(D2)
```

```
#Test
D_test = scaler.transform(D_test)
```

We will over sample our minority(down hole equip fail) using RandomOverSampler

```
from imblearn.over_sampling import RandomOverSampler
from collections import Counter
```

```
oversample = RandomOverSampler(sampling_strategy='minority')

# fit and apply the transform
D1_over , y_1_over = oversample.fit_resample(D1, y_1)
```

```
/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)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:144: FutureWarning: The sklearn.neighbors.base module is  deprecated in version 0.22 and will be remove
  warnings.warn(message, FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function safe_indexing is deprecated; safe_indexing is deprecated in version 0.22 and
  warnings.warn(msg, category=FutureWarning)
```

```
# summarize class distribution
print("distribution before oversampling = ",Counter(y_1))
print("distribution after  oversampling = ",Counter(y_1_over))
print("-"*50)
print("shape of X_train = ", D1.shape)
print("shape of y_train = ", len(y_1))
print("-"*50)
print("shape of X_train_over = ", D1_over.shape)
print("shape of y_train_over = ", y_1_over.shape)
```

```
distribution before oversampling =  Counter({0.0: 41300, 1.0: 700})
distribution after  oversampling =  Counter({0.0: 41300, 1.0: 41300})
--------------------------------------------------
```

```
shape of X_train =  (42000, 232)
shape of y_train =  42000
--------------------------------------------------
```

Making 3 dataset(Da,Db,Dc) for 3 models

```
def make_sample(data=D1_over ,label = y_1_over , size = 42000):
    random_index = np.random.randint(0 , len(data) , size)
    sampled_data  = []
    sampled_label = []

    for random in random_index:
       sampled_data.append(data[random])
       sampled_label.append(label[random])

    return np.array(sampled_data),np.array(sampled_label)


Da,ya = make_sample(data=D1_over ,label = y_1_over , size = 90000)
Db,yb = make_sample(data=D1_over ,label = y_1_over , size = 90000)
Dc,yc = make_sample(data=D1_over ,label = y_1_over , size = 90000)

print("*Distributon of data*")
print("ya = ",Counter(ya))
print("yb = ",Counter(yb))
print("yc = ",Counter(yc))
```

```
*Distributon of data*
ya =  Counter({1.0: 45216, 0.0: 44784})
yb =  Counter({0.0: 45067, 1.0: 44933})
yc =  Counter({1.0: 45096, 0.0: 44904})
```

Making 3 models

Decision Tree

```
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
import seaborn as sns

from sklearn.tree import DecisionTreeClassifier

def plot_confusion_matrix(test_y, predict_y,lables):
    C = confusion_matrix(test_y, predict_y)
    print("Number of misclassified points ",(len(test_y)-np.trace(C))/len(test_y)*100)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predicted class j

    A =(((C.T)/(C.sum(axis=1))).T)
    #divid each element of the confusion matrix with the sum of elements in that column

    # C = [[1, 2],
    #     [3, 4]]
    # C.T = [[1, 3],
    #        [2, 4]]
    # C.sum(axis = 1)  axis=0 corresonds to columns and axis=1 corresponds to rows in two diamensional array
    # C.sum(axix =1) = [[3, 7]]
    # ((C.T)/(C.sum(axis=1))) = [[1/3, 3/7]
    #                            [2/3, 4/7]]

    # ((C.T)/(C.sum(axis=1))).T = [[1/3, 2/3]
    #                            [3/7, 4/7]]
    # sum of row elements = 1

    B =(C/C.sum(axis=0))
    #divid each element of the confusion matrix with the sum of elements in that row
    # C = [[1, 2],
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds to rows in two diamensional array
    # C.sum(axix =0) = [[4, 6]]
    # (C/C.sum(axis=0)) = [[1/4, 2/6],
    #                       [3/4, 4/6]]

    labels = lables
    cmap=sns.light_palette("green")
    # representing A in heatmap format
    print("-"*50, "Confusion matrix", "-"*50)
    plt.figure(figsize=(10,5))
    sns.heatmap(C, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

    print("-"*50, "Precision matrix", "-"*50)
    plt.figure(figsize=(10,5))
    sns.heatmap(B, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
    print("Sum of columns in precision matrix",B.sum(axis=0))

    # representing B in heatmap format
    print("-"*50, "Recall matrix"    , "-"*50)
    plt.figure(figsize=(10,5))
    sns.heatmap(A, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
    print("Sum of rows in precision matrix",A.sum(axis=1))
```

```
DT_best = DecisionTreeClassifier(class_weight="balanced" , max_depth=30 , min_samples_split = 30)
DT_best.fit(Da,ya)

print("Train F1 Score = ",f1_score(ya , DT_best.predict(Da)))
print("Validation F1 Score = ",f1_score(y_2 , DT_best.predict(D2)))
```

```
Train F1 Score =  0.9975291210730673
Validation F1 Score =  0.5949820788530467
```

Gradient_Boost_1

```
import xgboost as xgb

GB1 = xgb.XGBClassifier(max_depth=100,learning_rate=0.12,n_estimators=2000,colsample_bytree=0.4,subsample=0.4)

GB1.fit(Db,yb)

print("Train F1 Score = ",f1_score(yb , GB1.predict(Db)))
print("Validation F1 Score = ",f1_score(y_2 , GB1.predict(D2)))
```

```
Train F1 Score =  1.0
Validation F1 Score =  0.7631578947368423
```

Gradient_Boost_2

```
GB2 = xgb.XGBClassifier(max_depth=60,learning_rate=0.1,n_estimators=3000,colsample_bytree=0.7,subsample=0.7)

GB2.fit(Dc,yc)

print("Train F1 Score = ",f1_score(yc , GB2.predict(Dc)))
print("Validation F1 Score = ",f1_score(y_2 , GB2.predict(D2)))
```

```
Train F1 Score =  1.0
Validation F1 Score =  0.7911111111111111
```

Stacking Classifier

```
y_1 = np.array(y_1)
y_2 = np.array(y_2)
```

```
from sklearn.linear_model import LogisticRegression
from mlxtend.classifier import StackingCVClassifier
```

```
GB = xgb.XGBClassifier(max_depth=50,learning_rate=0.1,n_estimators=3000,colsample_bytree=0.7,subsample=0.7)

SC = StackingCVClassifier(classifiers=[DT_best,GB1,GB2],
                            use_probas=False,
                            use_features_in_secondary=True,
                            meta_classifier=GB)

SC.fit(D1,y_1)

print("Train F1 Score = ",f1_score(y_1 , SC.predict(D1)))
print("Validation F1 Score = ",f1_score(y_2 , SC.predict(D2)))
```

```
Train F1 Score =  1.0
Validation F1 Score =  0.8125
```

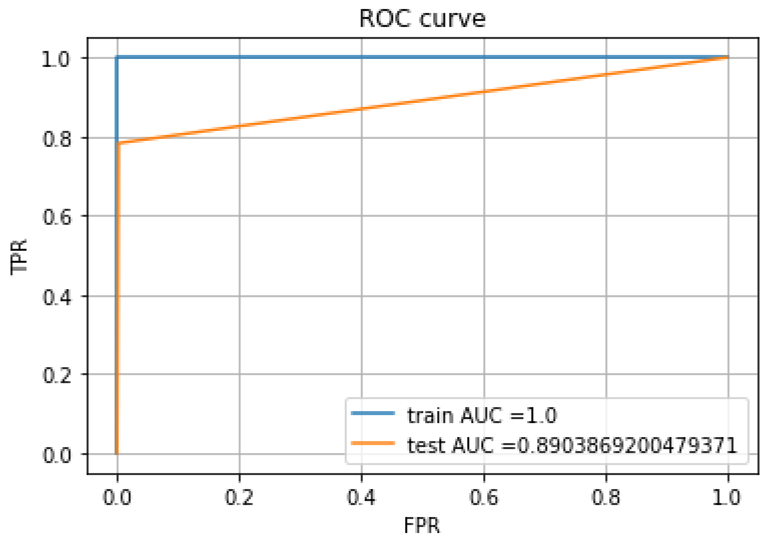
```
print("Test F1 Score = ",f1_score(y_test , SC.predict(D_test)))
```

```
Test F1 Score =  0.8222811671087533
```

AUC Score

```
train_fpr, train_tpr, tr_thresholds = roc_curve(y_1, SC.predict(D1))
test_fpr, test_tpr, te_thresholds = roc_curve(y_test , SC.predict(D_test))

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC curve")
plt.grid()
plt.show()
```



Precision

```
print("Train precision Score = ",precision_score(y_1 , SC.predict(D1)))
print("Test precision Score = ",precision_score(y_test , SC.predict(D_test)))
```

```
Train precision Score =  1.0
Test precision Score =  0.8659217877094972
```

Recall

Recall

```
print("Train recall Score = ",recall_score(y_1 , SC.predict(D1)))
print("Test recall Score = ",recall_score(y_test , SC.predict(D_test)))
```

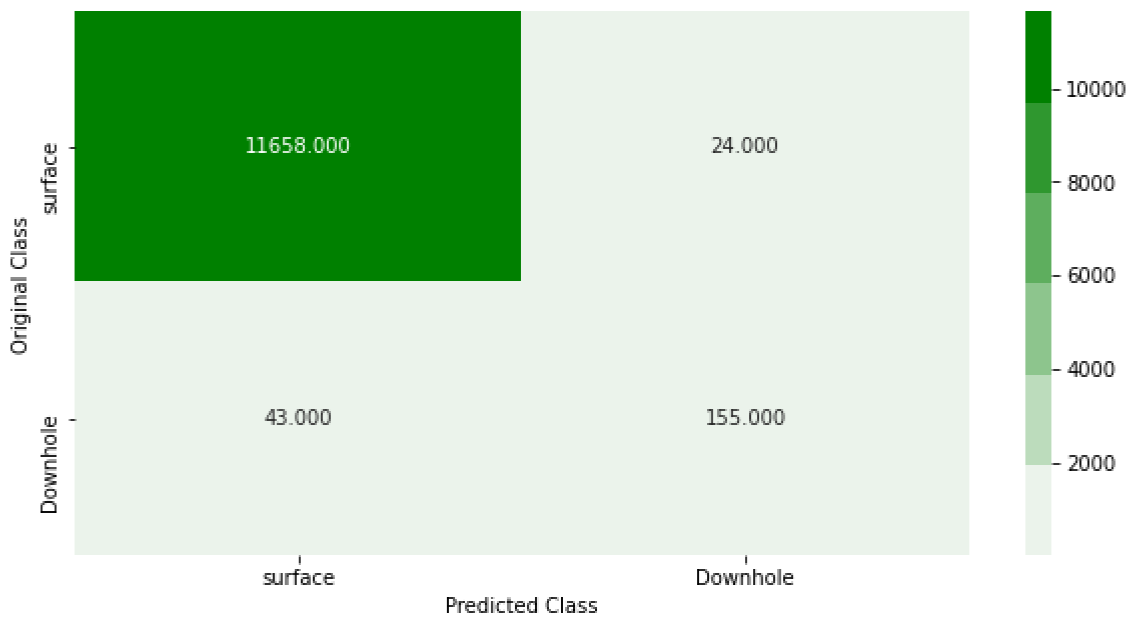
```
Train recall Score =  1.0
Test recall Score =  0.7828282828282829
```

Confusion matrix test

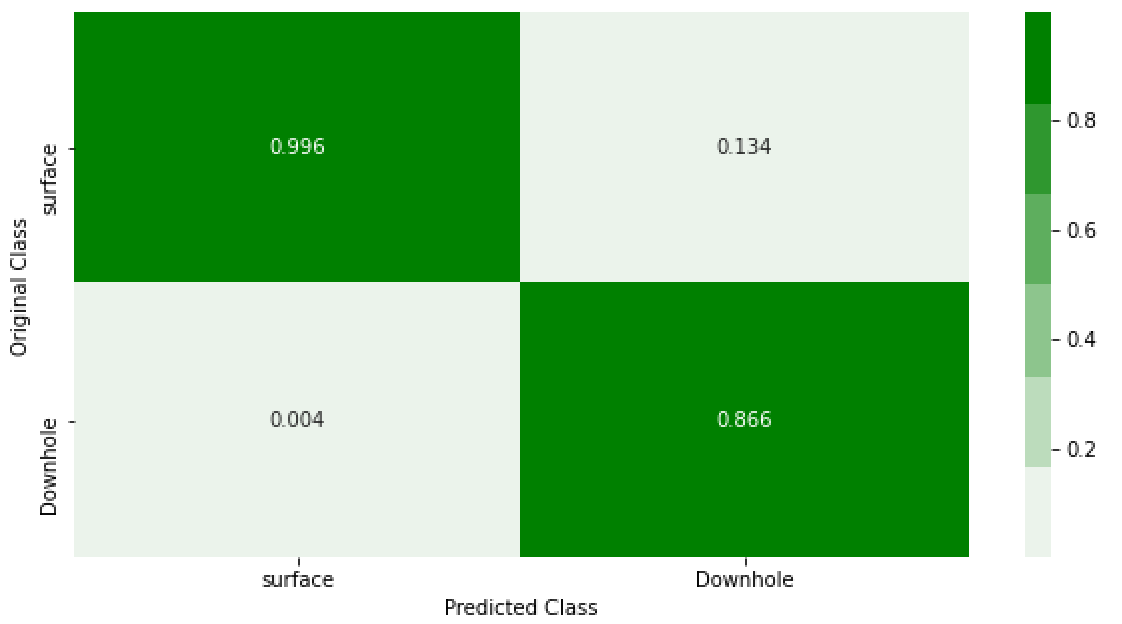
```
lables = ["surface" , "Downhole"]

plot_confusion_matrix(y_test, SC.predict(D_test) , lables)
```

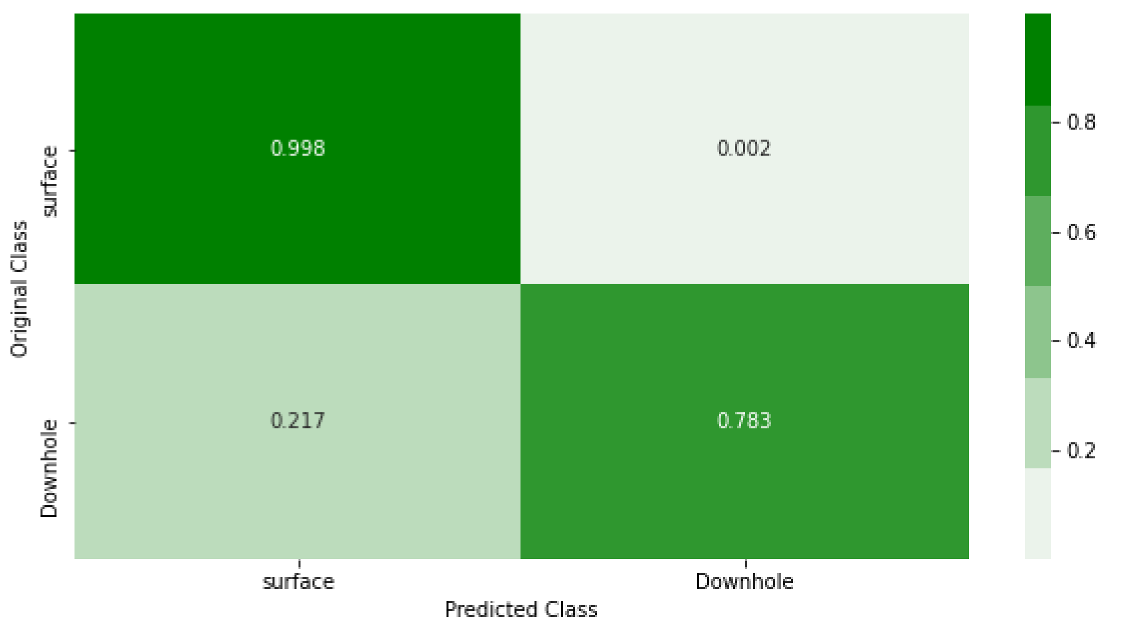
```
Number of misclassified points  0.563973063973064
-------------------------------------------------- Confusion matrix --------------------------------------------------
```




```
-------------------------------------------------- Precision matrix --------------------------------------------------
```




```
Sum of columns in precision matrix [1. 1.]
-------------------------------------------------- Recall matrix --------------------------------------------------
```




```
Sum of rows in precision matrix [1. 1.]
```

Result

- Just using 70% of total data, we mannaged to get F1 score of 0.82. Which is pretty good