Business problem

Stripper wells

A Stripper well is a low yield oil or gas well. These wells have low operational costs and have tax breaks which makes it attractive from a business point of view because of steady and low cost form of cash flow. Almost 80% of the oil wells in the US are Stripper wells.

The well has a lot of mechanical components and the breakdown is quite often compared to other wells. The breakdown occurs at surface level or down-hole level.

Project goal

The goal of the project is to predict whether the mechanical component has failed in the surface level or down-hole level. This information can be used to send the repair team to address the failure in either of the levels. Also, it is mentioned to find a way to minimize the costs associated with the failure.

In a business perspective, time is more valuable than small overhead costs. Therefore, it is very inefficient to send the repair team without knowing where the failure has occured. Therefore it becomes a neccessity to create an algorithm which predicts where the failure has occured.

The target values depend on large number of features. Since these many features when handled by a statistician will take a long time to predict target value, a machine learning algorithm is preffered since it saves time and money.

Dataset

The dataset is provided by ConocoPhilips. The dataset has 107 features which are taken from the sensors that collect a variety of information at surface and bottom levels.

Reading data

In [1]:

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

In [2]:

```
!cp '/content/gdrive/My Drive/falls.zip' '/content'
!unzip '/content/falls.zip'
```

```
Archive:  /content/falls.zip
   creating: falls/
  inflating: falls/equip_failures_test_set.csv
  inflating: falls/equip_failures_training_set.csv
```

In [3]:

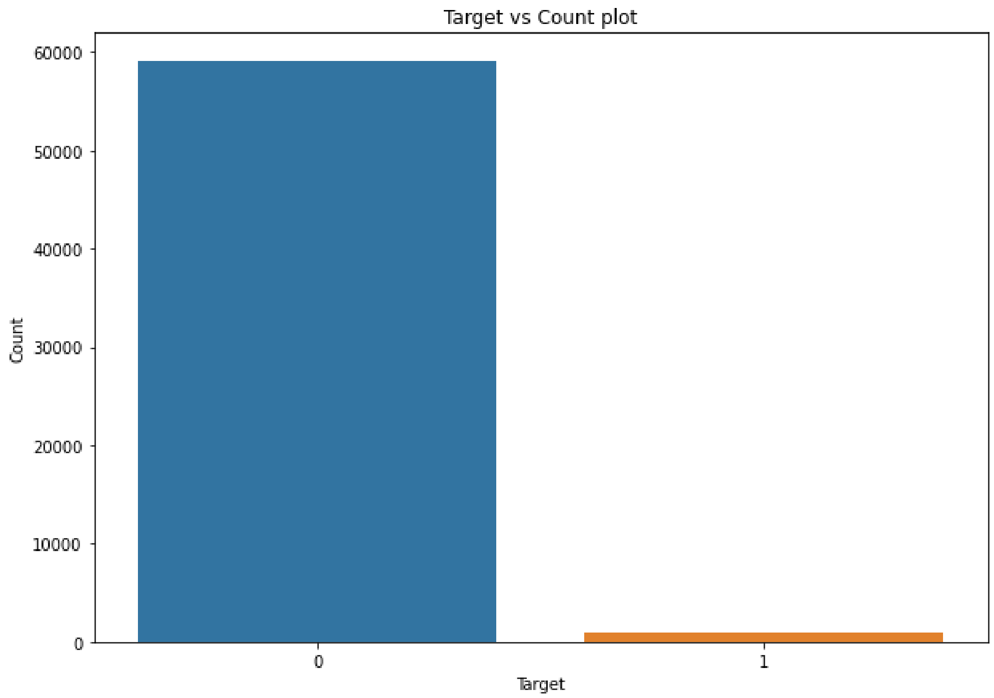
```
import pandas as pd

test = pd.read_csv('/content/falls/equip_failures_test_set.csv')
train = pd.read_csv('/content/falls/equip_failures_training_set.csv')
```

EDA

EDA

In [ ]:

```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10,7))
sns.countplot(x='target', data = train)
plt.title('Target vs Count plot')
plt.xlabel('Target')
plt.ylabel('Count')
plt.show()
```



Observation: The data is heavely imbalanced, therefore accuracy measure wont do good.

Since the above surface and down hole breakdown predictions have equal importance in a business perspective, both false positives and false negatives have equal weightage. (Since almost equal amount of time will be lost in either surface incase of a wrong prediction). Therefore, F1 score as performance metrics makes more sense

In [ ]:

```
train.head()
```

Out[ ]:

	id	target	sensor1_measure	sensor2_measure	sensor3_measure	sensor4_measure	sensor5_measure	sensor6_measure	sensor7_I
0	1	0	76698	na	2130706438	280	0	0	
1	2	0	33058	na	0	na	0	0	
2	3	0	41040	na	228	100	0	0	
3	4	0	12	0	70	66	0	10	
4	5	0	60874	na	1368	458	0	0	

5 rows × 172 columns

Observation: Train data has alot of na values. Since they are not np.nan (NaN) they need to be replaced by np.nan to make the features type float

In [ ]:

```
import numpy as np
```

```
train = train.replace('na',np.nan)

for col in train.columns:
  train[col] = train[col].astype(np.float) #converting to float because the data input na values we
re are of string type
```

In [ ]:

```
nan_0_to_25=[]
nan_25_to_50=[]
nan_50_to_75=[]
nan_75_to_100=[]

count = train.isnull().sum(axis=0).tolist()

for i in range(len(count)):
  sum1 = count[i]*100/len(train)
  if sum1 <= 25:
    nan_0_to_25.append(train.columns[i])
  elif sum1>25 and sum1<=50:
    nan_25_to_50.append(train.columns[i])
  elif sum1>50 and sum1<=75:
    nan_50_to_75.append(train.columns[i])
  elif sum1>75:
    nan_75_to_100.append(train.columns[i])

if len(nan_0_to_25)+len(nan_25_to_50)+len(nan_50_to_75)+len(nan_75_to_100) == len(train.columns):
  print('True')
```

```
True
```

In [ ]:

```
print('Features and NaN values observation:\n')
print('| missing value % |  count | ')
print('|      0-25       |  ',len(nan_0_to_25),' |')
print('|      25-50      |  ',len(nan_25_to_50),'   |')
print('|      50-75      |  ',len(nan_50_to_75),'   |')
print('|      75-100     |  ',len(nan_75_to_100),'   |')
```

```
Features and NaN values observation:

| missing value % |  count |
|      0-25       |   162  |
|      25-50      |   2    |
|      50-75      |   2    |
|      75-100     |   6    |
```

We will lose only a fraction of data if we delete features with missing values greater than 25% but first we will check if nan values are related to target variables.

In [ ]:

```
from tqdm import tqdm

y = train.target
hamming_distance = []

for i in tqdm(range(len(train.columns))):
    diff=0
    series = np.zeros(len(train))
    for j in range(len(train)):
      if np.isnan(train.iloc[j,i])==False:
        series[j]=1
      if series[j]-y[j]==0:
        diff+=1
    hamming_distance.append(diff)
```

```
100%|██████████| 172/172 [05:11<00:00,  1.81s/it]
```

Observing both max and min hamming distance because, if both nan substitute and target have same values, then hamming distance will be short. If they have different values, then distance will be large.

If the distance is short enough, close to 1000, then it is very valuable for the given data because it almost matches the target values.

If the distance is very large, close to 59000, then also it is very valuable because if we just interchange the nan subsitute values then it will become short and match the target values

In [ ]:

```
max(hamming_distance), min(hamming_distance)
```

Out[ ]:

```
(50084, 1000)
```

Observation:

1. Max hamming distance is almost 50000 which is garbage.
2. Min hamming distance is around 1000 which is very valuable for given data.

In [ ]:

```
'''filling nan values as 0 and rest 1 for features with less than 1500 hamming distance'''

save_cols=[]
for i in range(len(hamming_distance)):
  if hamming_distance[i]<1500:
    if i >1:
      save_cols.append(train.columns[i])
```

In [ ]:

```
save_cols
```

Out[ ]:

```
['sensor1_measure',
 'sensor45_measure',
 'sensor49_measure',
 'sensor59_measure',
 'sensor60_measure',
 'sensor61_measure']
```

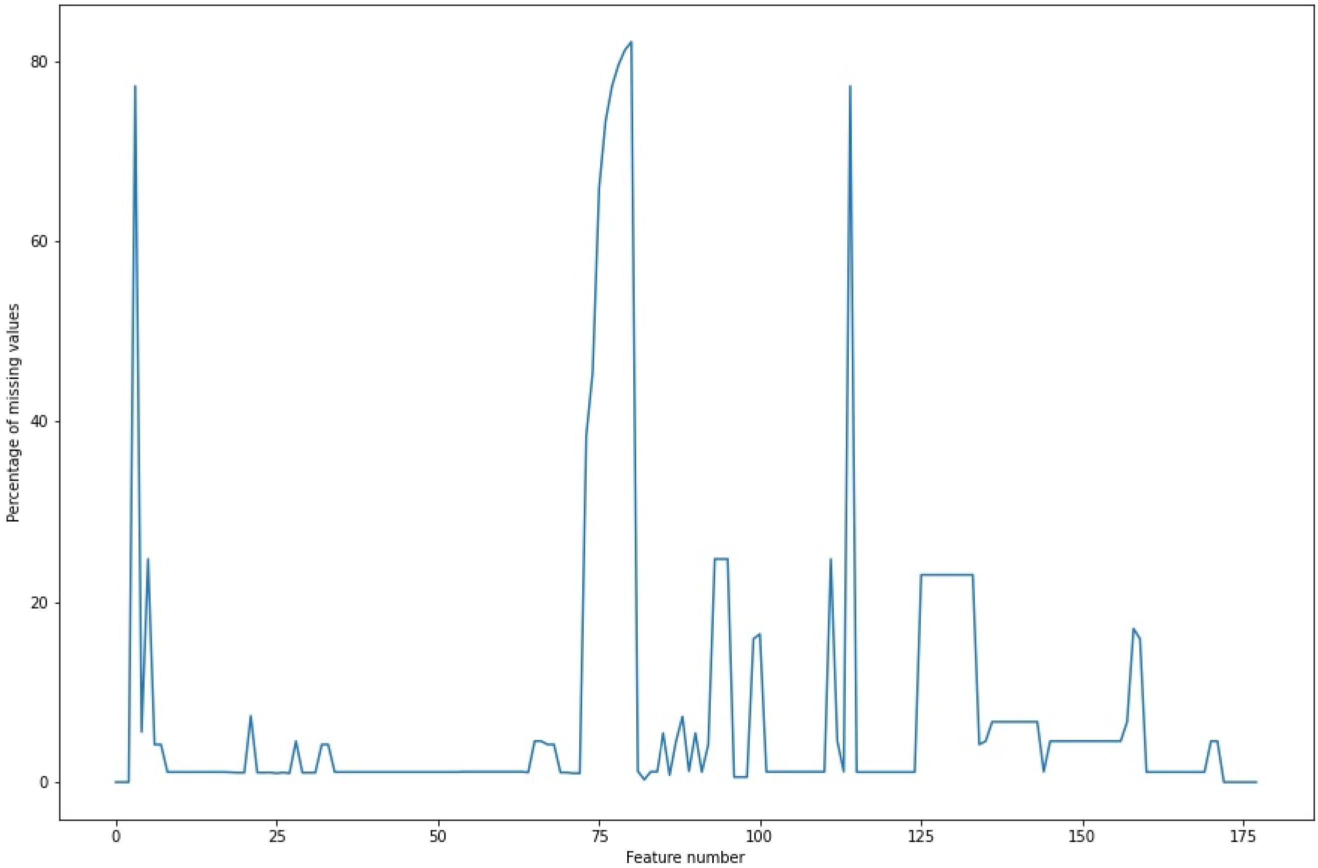
In [ ]:

```
'''Creating a new feature for those features whose nan values might be helpful in predictions'''

train['new_val1'] = np.where(train['sensor1_measure'].isnull(), 0, 1)
train['new_val2'] = np.where(train['sensor45_measure'].isnull(), 0, 1)
train['new_val3'] = np.where(train['sensor49_measure'].isnull(), 0, 1)
train['new_val4'] = np.where(train['sensor59_measure'].isnull(), 0, 1)
train['new_val5'] = np.where(train['sensor60_measure'].isnull(), 0, 1)
train['new_val6'] = np.where(train['sensor61_measure'].isnull(), 0, 1)
```

In [ ]:

```
'''plotting histogram of missing values for each feature'''

count = train.isnull().sum(axis=0).tolist()

for i in range(len(count)):
  count[i] = count[i]/600

col_names=[]
for i in range(178):
  col_names.append(i)

plt.figure(figsize=(15,10))
plt.plot(col_names,count)
```

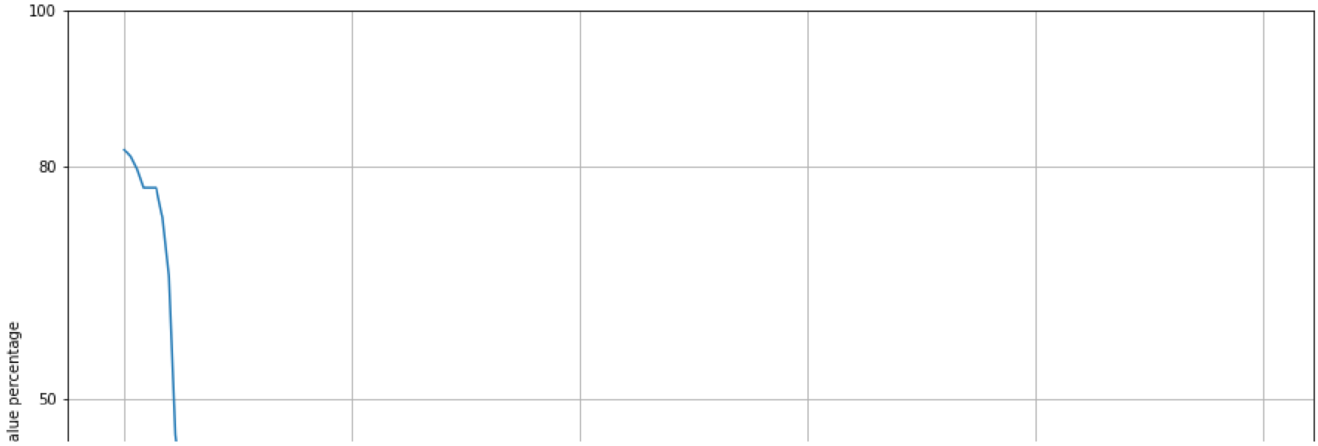
```
plt.xlabel('Feature number')
plt.ylabel('Percentage of missing values')
plt.show()
```

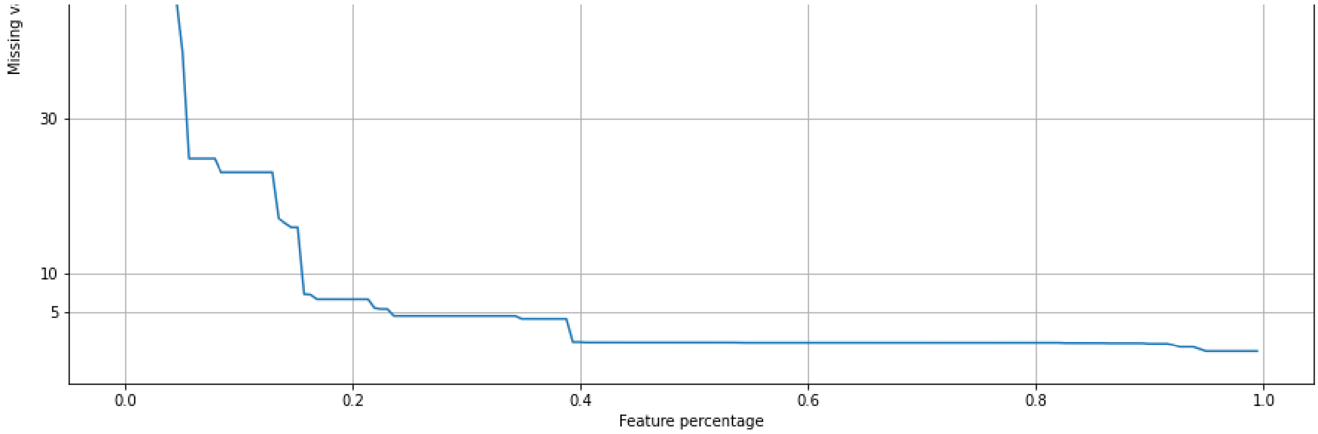


Observation: There is a lot of missing values in th data set.

In [ ]:

```
'''plotting missing values in decreasing order to get percentile plot'''

perc_vals = []
for i in col_names:
  perc_vals.append(i/178)

sorted_counts = sorted(count,reverse=True)

plt.figure(figsize=(15,10))
plt.plot(perc_vals,sorted_counts)
plt.yticks([5,10,30,50,80,100])
plt.xlabel('Feature percentage')
plt.ylabel('Missing value percentage')
plt.grid()
plt.show()
```



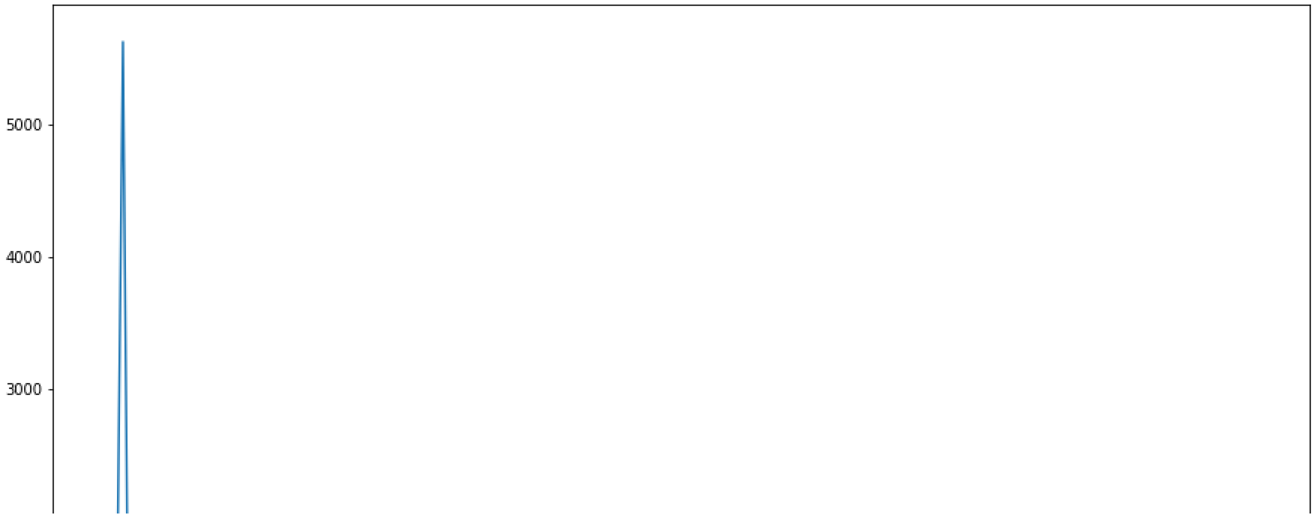


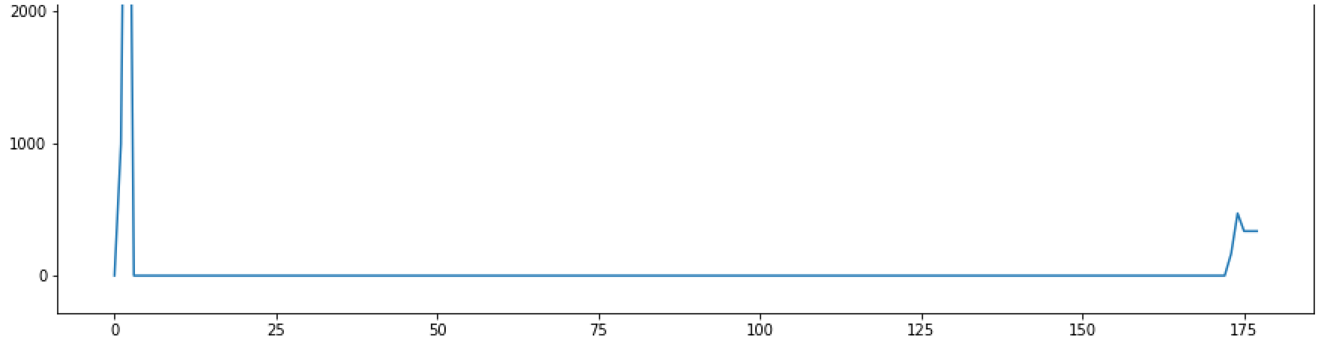
Observation: Almost 40% of features having more than 5% missing values.

In [ ]:

```
'''checking for outliers'''

for c in train.columns:
    train[c] = train[c].astype(np.float)

out_count = []

for i in tqdm(train.columns):
  Q25 = np.percentile(train[i],25)
  Q75 = np.percentile(train[i],75)
  IQR = Q75-Q25
  IQR = IQR*1.5
  UL = Q75+IQR
  LL = Q25-IQR
  out=0
  for i in train[i]:
    if i>UL or i<LL:
      out+=1
  out_count.append(out)

out_cols = []

for i in range(len(out_count)):
  if out_count[i]>0 and i>1:
    out_cols.append((train.columns[i],out_count[i]))

plt.figure(figsize=(15,10))
plt.plot(col_names,out_count)
plt.show()

out_cols
```

```
100%|██████████| 178/178 [00:04<00:00, 43.05it/s]
```





Out[ ]:

```
[('sensor1_measure', 5627),
 ('new_val2', 167),
 ('new_val3', 473),
 ('new_val4', 338),
 ('new_val5', 338),
 ('new_val6', 338)]
```

Observation: Only one feature has outlier. It is sensor1_measure with 5627 outliers. Remaining features are features have majority 0 values because they are new features.

In [ ]:

```
'''removing sensor1 because of too many outliers and also removing id'''

train = train.drop(columns=['sensor1_measure','id'],axis=1)
```

In [ ]:

```
train.head()
```

Out[ ]:

	target	sensor2_measure	sensor3_measure	sensor4_measure	sensor5_measure	sensor6_measure	sensor7_histogram_bin0	sensor
0	0.0	NaN	2.130706e+09	280.0	0.0	0.0	0.0	
1	0.0	NaN	0.000000e+00	NaN	0.0	0.0	0.0	
2	0.0	NaN	2.280000e+02	100.0	0.0	0.0	0.0	
3	0.0	0.0	7.000000e+01	66.0	0.0	10.0	0.0	
4	0.0	NaN	1.368000e+03	458.0	0.0	0.0	0.0	

5 rows × 176 columns

In [ ]:

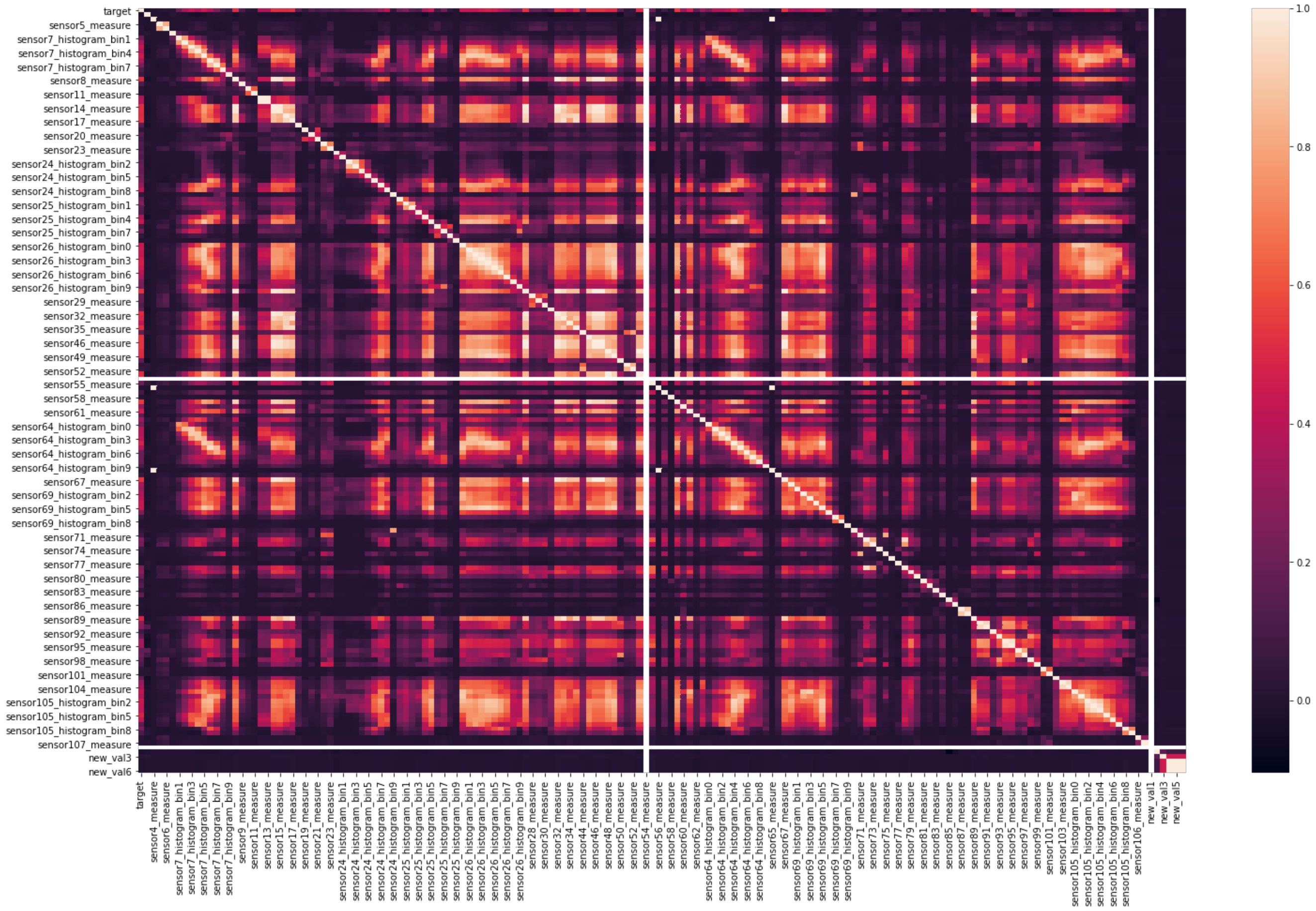
```
'''Removing features with nan values greater than threshold'''

def nan_cols(df,val,threshold=None,col_names=None):
  if val==1:
    perc = []
    for i in range(len(df.columns)):
      nans = df.iloc[:,i].isnull().sum()
      perc.append(nans/len(df))
    df_names=[]
    for i in range(len(perc)):
      if perc[i]>threshold:
        df_names.append(df.columns[i])
    for i in df_names:
      df = df.drop(columns=i,axis=1)
    return df

train = nan_cols(train,1,threshold=0.25)
```

In [ ]:

```
'''imputing mean values so that heatmap does not have NaN values.'''

train = train.iloc[:,:].fillna(train.mean())
```

In [ ]:

```
'''Checking correlation among features by plotting heatmap'''

import seaborn as sns

mat = train.corr()

plt.figure(figsize=(25,15))
sns.heatmap(mat)
plt.show()
```



Observation: Heatmap has bright points which means that there are many highly correlated features

In [ ]:

```
'''Removing features with correlation greater than given threshold'''

threshold=0.8

print('The number of columns in dataset before removing correlated features:',len(train.columns))

indeces=[]
for i in range(len(mat)):
  for j in range(len(mat)):
    if i!=0:
      if j!=0:
        if i!=j:
          if abs(mat.iloc[i,j])>threshold:
            indeces.append((i,j))
save=[]
delete=[]
for i in indeces:
  if i[0] in save:
```

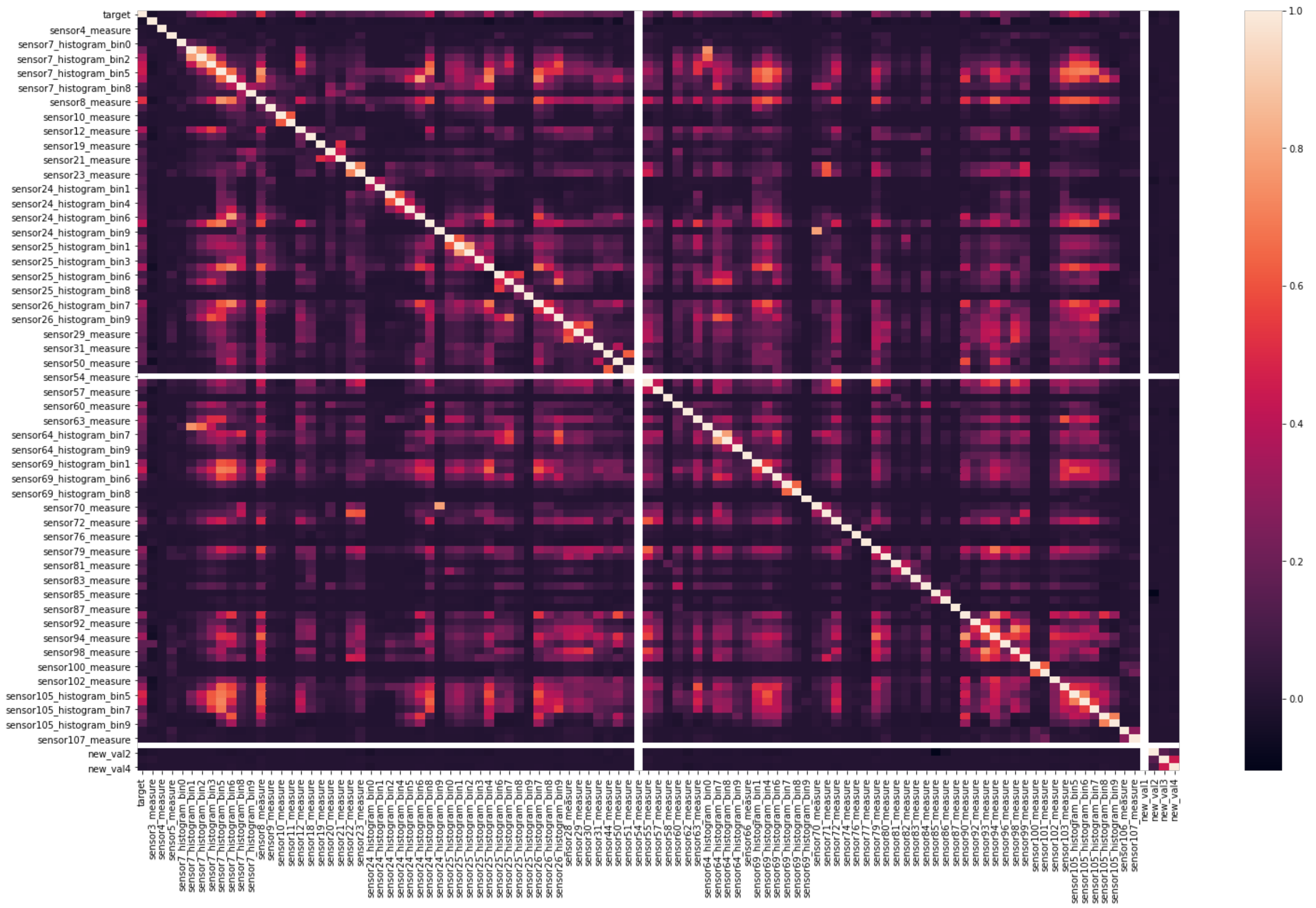
```
    if i[1] not in delete:
      delete.append(i[1])
  elif i[0] in delete:
    if i[1] not in save:
      if i[1] not in delete:
        delete.append(i[1])
  elif i[1] in delete:
    delete.append(i[0])
  else:
    save.append(i[0])
    delete.append(i[1])
names=[]
for i in delete:
  names.append(train.columns[i])
indeces=[]
for i in names:
  train = train.drop(columns=i,axis=1)

print('The number of columns in dataset after removing correlated features:',len(train.columns))
```

```
The number of columns in dataset before removing correlated features: 166
The number of columns in dataset after removing correlated features: 105
```

Observation: The number of dimensions have been reduced from 170 to 104 (target)

In [ ]:

```
'''Plotting heatmap after removing highly correlated features'''

mat2 = train.corr()
plt.figure(figsize=(25,15))
sns.heatmap(mat2)
plt.show()
```



Observation: There is an abnormal feature in the dataset because of two white lines (one horizontal and one vertical) in the heatmap. If it was correlation then it wouldve been removed. The feature have NaN values.

In [ ]:

```
null_columns=mat2.columns[mat2.isnull().sum()>2] #Since all features will have atleast one NaN val
ue because of one or more NaN feature
null_columns
```

Out[ ]:

```
Index(['sensor54_measure', 'new_val1'], dtype='object')
```

In [ ]:

```
train.sensor54_measure
```

Out[ ]:

```
0        1209600.0
1        1209600.0
2        1209600.0
3        1209600.0
4        1209600.0
           ...
59995    1209600.0
59996    1209600.0
59997    1209600.0
59998    1209600.0
59999    1209600.0
Name: sensor54_measure, Length: 60000, dtype: float64
```

In [ ]:

```
train.new_val1
```

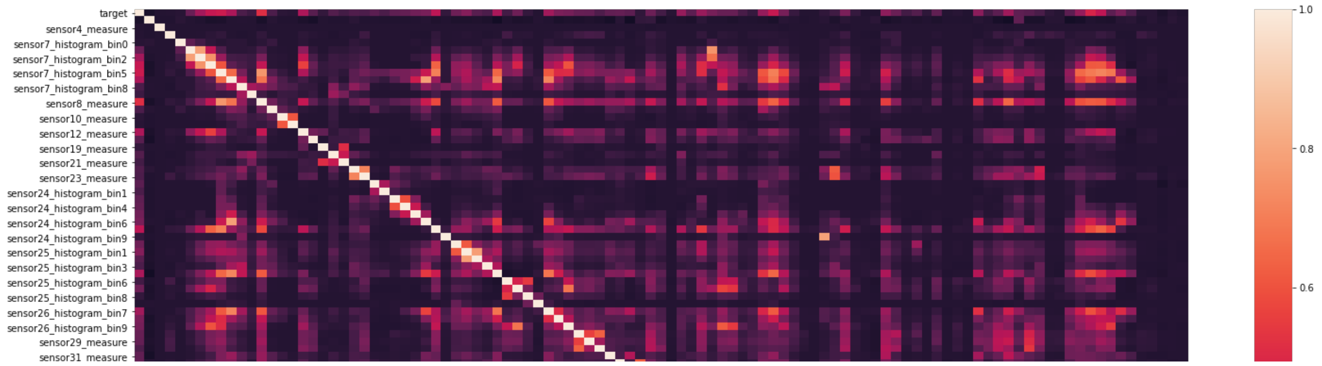
Out[ ]:

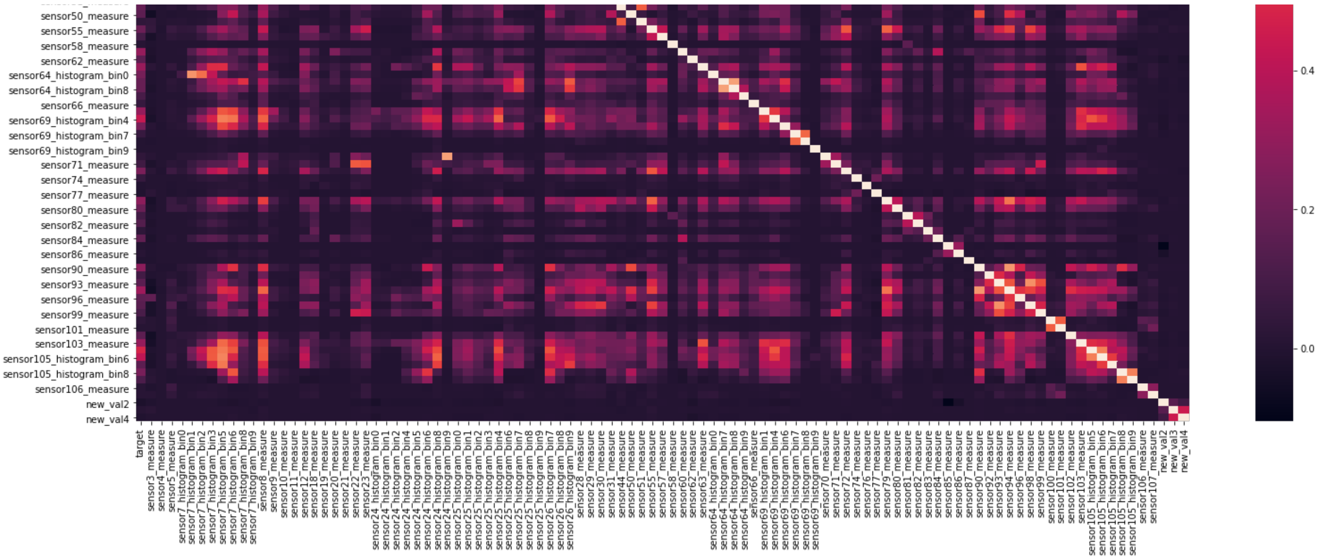
```
0        1.0
1        1.0
2        1.0
3        1.0
4        1.0
        ...
59995    1.0
59996    1.0
59997    1.0
59998    1.0
59999    1.0
Name: new_val1, Length: 60000, dtype: float64
```

Observation: The NaN values in correlation matrix was because the sensor54_measure had only one value. This is useless and therefore removing this feature.

In [ ]:

```
train = train.drop(columns=['sensor54_measure','new_val1'],axis=1)
mat3 = train.corr()
plt.figure(figsize=(25,15))
sns.heatmap(mat3)
plt.show()
```





Observation: Heatmap looks much cleaner now since highly correlated features are removed

Feature engineering

In [4]:

```
import pandas as pd

test = pd.read_csv('/content/falls/equip_failures_test_set.csv')
train = pd.read_csv('/content/falls/equip_failures_training_set.csv')
```

In [5]:

```
import numpy as np

train = train.replace('na',np.nan)
```

In [6]:

```
train['new_val2'] = np.where(train['sensor45_measure'].isnull(), 0, 1)
train['new_val3'] = np.where(train['sensor49_measure'].isnull(), 0, 1)
train['new_val4'] = np.where(train['sensor59_measure'].isnull(), 0, 1)
train['new_val5'] = np.where(train['sensor60_measure'].isnull(), 0, 1)
train['new_val6'] = np.where(train['sensor61_measure'].isnull(), 0, 1)
```

In [7]:

```
'''Splitting data for validation purpose and avoid data leakage while feature engineering'''

from sklearn.model_selection import train_test_split

y=train.target
train=train.drop(columns=['target','sensor54_measure','id'],axis=1)

X_train, X_test, y_train, y_test = train_test_split(train, y, test_size=0.2, random_state=42, strat
ify=y)
```

In [8]:

```
'''converting data into float and doing mean imputation for columns less than 10% missing values''
'

def impute(df):
  for col in df.columns:
    df[col] = df[col].astype(np.float)
    df[col] = df[col].fillna(df[col].mean())
  return df
```

```
X_train = impute(X_train)
X_test = impute(X_test)
```

```
/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
  """
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:6: 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
```

In [9]:

```
'''Removing features with nan values greater than threshold'''

def nan_cols(df,val,threshold=None,col_names=None):
  if val==1:
    perc = []
    for i in range(len(df.columns)):
      nans = df.iloc[:,i].isnull().sum()
      perc.append(nans/len(df))
    df_names=[]
    for i in range(len(perc)):
      if perc[i]>threshold:
        df_names.append(df.columns[i])
    for i in df_names:
      df = df.drop(columns=i,axis=1)
    return df,df_names
  else:
    for i in col_names:
      df = df.drop(columns=i,axis=1)
    return df

X_train_,cols = nan_cols(X_train,1,threshold=0.25)
X_test_ = nan_cols(X_test,0,col_names=cols)

'''Making sure nan_cols gave correct output'''

if X_train.columns.all()==X_test.columns.all():
  print('yes')

print(len(X_train.columns)-len(X_train_.columns),'columns removed from X_train')
print(len(X_test.columns)-len(X_test_.columns),'columns removed from X_test')

X_train = X_train_
X_test = X_test_
```

```
yes
0 columns removed from X_train
0 columns removed from X_test
```

In [10]:

```
'''Creating a function that will remove features with correlation greater than given threshold'''

def corr_check(train_,mat=None,threshold=None,val=0,colss=None):
  if val==1:
    indeces=[]
    for i in range(len(mat)):
      for j in range(len(mat)):
        if i!=0:
          if j!=0:
            if i!=j:
              if abs(mat.iloc[i,j])>threshold:
                indeces.append((i,j))
    save=[]
    delete=[]
```

```
    for i in indeces:
      if i[0] in save:
        if i[1] not in delete:
          delete.append(i[1])
      elif i[0] in delete:
        if i[1] not in save:
          if i[1] not in delete:
            delete.append(i[1])
      elif i[1] in delete:
        delete.append(i[0])
      else:
        save.append(i[0])
        delete.append(i[1])
    names=[]
    for i in delete:
      names.append(train_.columns[i])
    indeces=[]
    for i in names:
      train_ = train_.drop(columns=i,axis=1)
    return train_,names
  else:
    for i in colss:
      train_ = train_.drop(columns=i,axis=1)
    return train_

corr = X_train.corr()

X_train_,col_names = corr_check(X_train,mat=corr,threshold=0.8,val=1)
X_test_ = corr_check(X_test,colss=col_names)

print(len(X_train.columns)-len(X_train_.columns),'features removed from X_train')
print(len(X_test.columns)-len(X_test_.columns),'features removed from X_test')

X_train = X_train_
X_test = X_test_
```

```
69 features removed from X_train
69 features removed from X_test
```

In [11]:

```
print('Shape of X_train after preprocessing:',X_train.shape)
print('Shape of X_test after preprocessing:',X_test.shape)
```

```
Shape of X_train after preprocessing: (48000, 105)
Shape of X_test after preprocessing: (12000, 105)
```

In [12]:

```
'''Scaling features so that it can be directly used in any model'''

from sklearn import preprocessing

def scale(df,val=0,fit=None):
  if val==1:
    x = df.values
    min_max_scaler = preprocessing.MinMaxScaler()
    x_scaled = min_max_scaler.fit_transform(x)
    fit = min_max_scaler.fit(x)
    df = pd.DataFrame(x_scaled)
    return df, min_max_scaler
  else:
    x=df.values
    x_scaled = fit_scaled.transform(x)
    df = pd.DataFrame(x_scaled)
    return df

X_train,fit_scaled = scale(X_train,val=1)
X_test = scale(X_test,fit=fit_scaled)
```

In [13]:

```
from sklearn.metrics import confusion_matrix

#from reference book assignment part
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    print("Number of misclassified points",(len(test_y)-np.trace(C)),'out of',len(test),'points')
    print('Precision:',round(metrics.precision_score(test_y,predict_y),3))
    print('Recall:',round(metrics.recall_score(test_y,predict_y),3))
    # 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 = [1,2]
    cmap=sns.light_palette("green")
    # representing A in heatmap format
    print("-"*50, "Confusion matrix", "-"*50)
    plt.figure(figsize=(10,6))
    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=(10,6))
    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=(10,6))
    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))
```

EDA and Feature Engineering summary

EDA:

1. The data is heavily imbalanced. 59:1 imbalance.
2. 40% of the features have more than 10% missing values.
3. There are only 8 features with more than 25% missing data.
4. Many features are correlated to each other.
5. There is a feature sensor54_measure which has only one value.

6. sensor1_measure has many outliers. No other features have any outliers.

FE:

1. Features with more than 25% missing values have been removed because it wont cause any information loss. (Less than 0.5% of data is lost).
2. Created 5 new useful features from features with nan values that have relation with target variables.
3. Removed features with greater than 0.8 correlation
4. Feature sensor54_measure is removed because it has only 1 value
5. The data is scaled so that it can be used on any model.

Modelling

Random Forest

In [ ]:

```
'''Hypertuning the Random Forest model using GridSearchCV'''

from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics

RF = RandomForestClassifier(n_jobs=-1,class_weight={0:1,1:59})
param_grid = { 'n_estimators': [200,500,1000,2000],'max_depth': [5,10,20]}
CV_RF = GridSearchCV(estimator=RF, param_grid=param_grid, cv= 5, scoring='f1',verbose=1)

CV_RF.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  60 out of  60 | elapsed: 80.3min finished
```

Out[ ]:

```
GridSearchCV(cv=5, error_score=nan,
             estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                              class_weight={0: 1, 1: 59},
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              max_samples=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators=100, n_jobs=-1,
                                              oob_score=False,
                                              random_state=None, verbose=0,
                                              warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': [5, 10, 20],
                         'n_estimators': [200, 500, 1000, 2000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='f1', verbose=1)
```

In [ ]:

```
'''Getting best hyper parameters'''

CV_RF.best_params_
```

Out[ ]:

```
{'max_depth': 20, 'n_estimators': 1000}
```

In [14]:

```python
'''Fitting the hyper tuned Random Forest model on train set'''

from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics

RF_final = RandomForestClassifier(max_depth=20,n_estimators=1000)
RF_final.fit(X_train,y_train)
```

Out[14]:

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=20, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=1000,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [27]:

```python
'''Validating the model with test set using F1 score'''

y_pred_RF = RF_final.predict(X_test)
f1=metrics.f1_score(y_pred_RF,y_test)
print('The F1 score for validation set is:',round(f1,3))
```

```
The F1 score for validation set is: 0.783
```

In [28]:

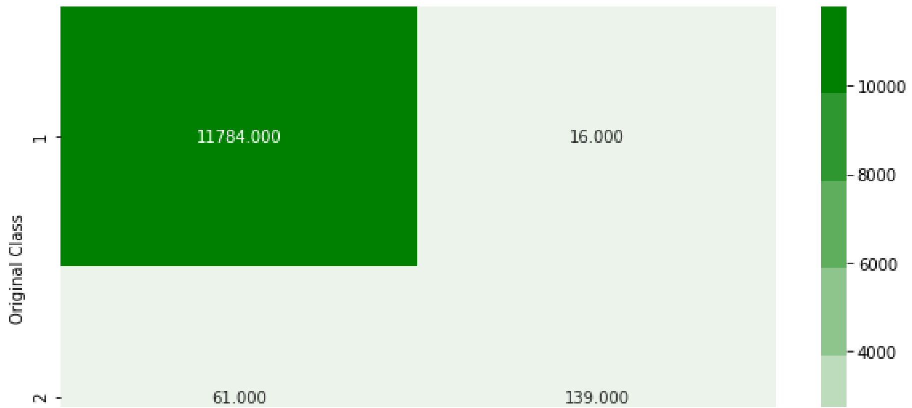
```python
'''Validating the model with test set using F1 score'''

y_pred_RF = RF_final.predict(X_test)
f1=metrics.f1_score(y_pred_RF,y_test,average='macro')
print('The Macro F1 score for validation set is:',round(f1,3))
```

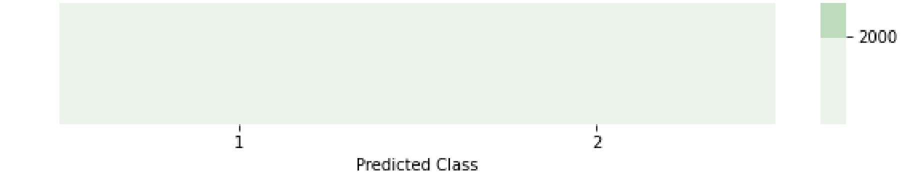
```
The Macro F1 score for validation set is: 0.89
```

In [26]:

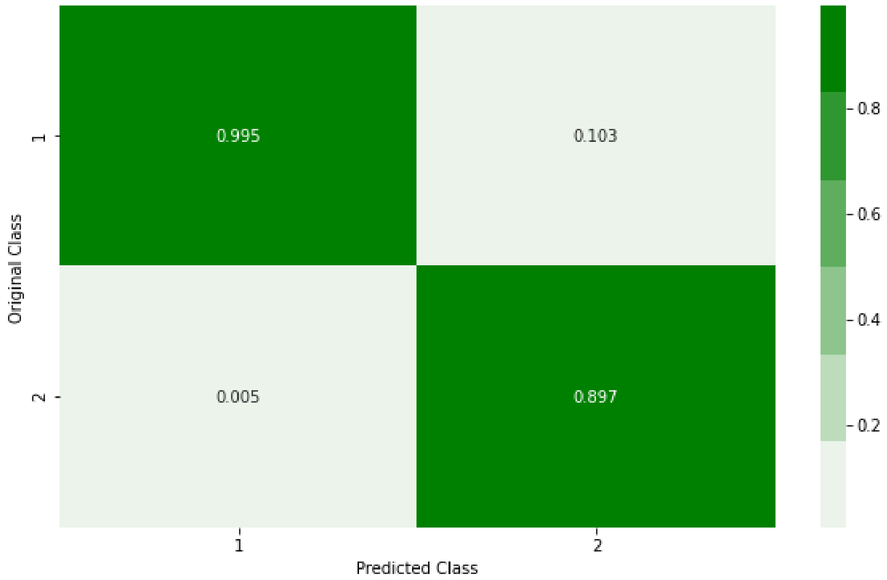
```python
import matplotlib.pyplot as plt
import seaborn as sns

plot_confusion_matrix(y_test,y_pred_RF)
```

```
Number of misclassified points 77 out of 16001 points
Precision: 0.897
Recall: 0.695
-------------------------------------------------- Confusion matrix ------------------------------
----------------
```

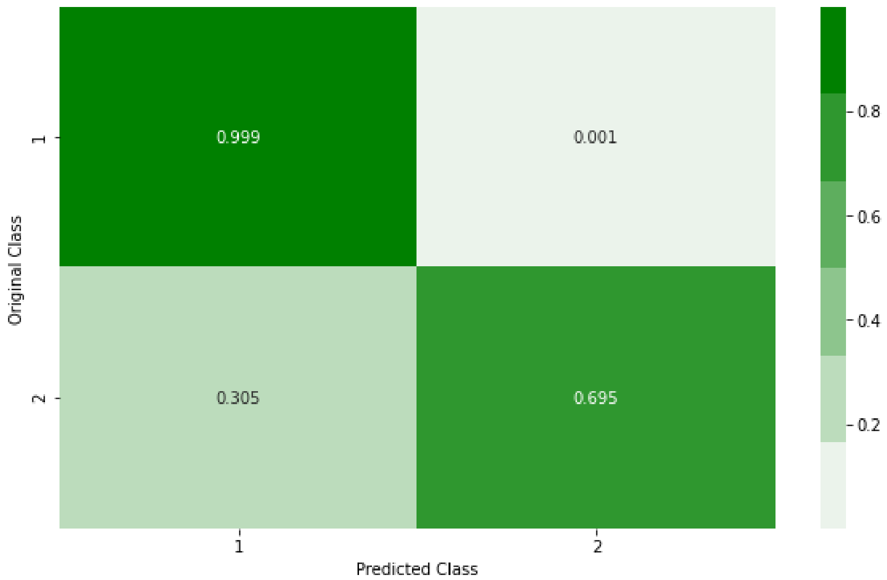





```
-------------------------------------------------- Precision matrix -------------------------------
----------------
```




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




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

XGBoost

In [30]:

```
import xgboost as xgb
from xgboost.sklearn import XGBClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn import metrics
```

In [ ]:

```
'''Hyper parameter tuning using RandomizedSearchCV'''

param_grid = {
    'max_depth': [10,20],
    'learning_rate': [0.1],
    'subsample': [0.5, 1],
    'colsample_bytree': [0.7,1],
    'colsample_bylevel': [0.7, 1.0],
    'min_child_weight': [3.0, 5.0],
    'gamma': [1.0],
    'reg_lambda': [ 1.0],
    'n_estimators': [100,200,500,1000,2000]}

xgb_model = xgb.XGBClassifier(n_jobs=-1,scale_pos_weight=59)

randomized_search = RandomizedSearchCV(xgb_model, param_grid, n_iter=30,
                        n_jobs=-1, cv=5,
                        scoring='f1', random_state=42)

randomized_search.fit(X_train, y_train)
```

Out[ ]:

```
RandomizedSearchCV(cv=5, error_score=nan,
                   estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                           colsample_bylevel=1,
                                           colsample_bynode=1,
                                           colsample_bytree=1, gamma=0,
                                           learning_rate=0.1, max_delta_step=0,
                                           max_depth=3, min_child_weight=1,
                                           missing=None, n_estimators=100,
                                           n_jobs=-1, nthread=None,
                                           objective='binary:logistic',
                                           random_state=0, reg_alpha=0,
                                           reg_lambda=1, s...
                   iid='deprecated', n_iter=30, n_jobs=-1,
                   param_distributions={'colsample_bylevel': [0.7, 1.0],
                                        'colsample_bytree': [0.7, 1],
                                        'gamma': [1.0], 'learning_rate': [0.1],
                                        'max_depth': [10, 20],
                                        'min_child_weight': [3.0, 5.0],
                                        'n_estimators': [100, 200, 500, 1000,
                                                         2000],
                                        'reg_lambda': [1.0],
                                        'subsample': [0.5, 1]},
                   pre_dispatch='2*n_jobs', random_state=42, refit=True,
                   return_train_score=False, scoring='f1', verbose=0)
```

In [ ]:

```
'''getting the best parameters'''

randomized_search.best_params_
```

Out[ ]:

```
{'colsample_bylevel': 0.7,
 'colsample_bytree': 0.7,
 'gamma': 1.0,
 'learning_rate': 0.1,
 'max_depth': 10,
 'min_child_weight': 5.0,
 'n_estimators': 1000,
 'reg_lambda': 1.0,
 'subsample': 0.5}
```

In [31]:

```
xgb_model = xgb.XGBClassifier(n_jobs=-1,scale_pos_weight=59, colsample_bylevel=0.7,colsample_bytree
=0.7,gamma=1.0,learning_rate=0.1,max_depth=10,min_child_weight=5.0,n_estimators=1000,reg_lambda=1.0
,subsample=0.5)
xgb_model.fit(X_train, y_train)
```

```
xgb_model.fit(x_train, y_train)
```

Out[31]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=0.7,
              colsample_bynode=1, colsample_bytree=0.7, gamma=1.0,
              learning_rate=0.1, max_delta_step=0, max_depth=10,
              min_child_weight=5.0, missing=None, n_estimators=1000, n_jobs=-1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1.0, scale_pos_weight=59, seed=None,
              silent=None, subsample=0.5, verbosity=1)
```

In [35]:

```
y_pred_lr = xgb_model.predict(X_test)
f1=metrics.f1_score(y_pred_lr,y_test)
round(f1,3)
```

Out[35]:

```
0.837
```

In [36]:

```
y_pred_lr = xgb_model.predict(X_test)
f1=metrics.f1_score(y_pred_lr,y_test,average='macro')
round(f1,3)
```

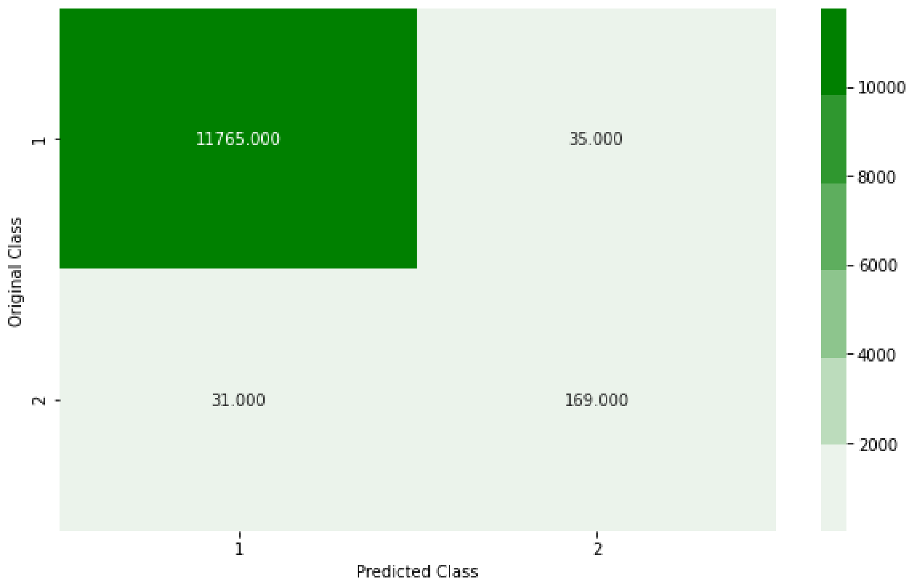
Out[36]:

```
0.917
```

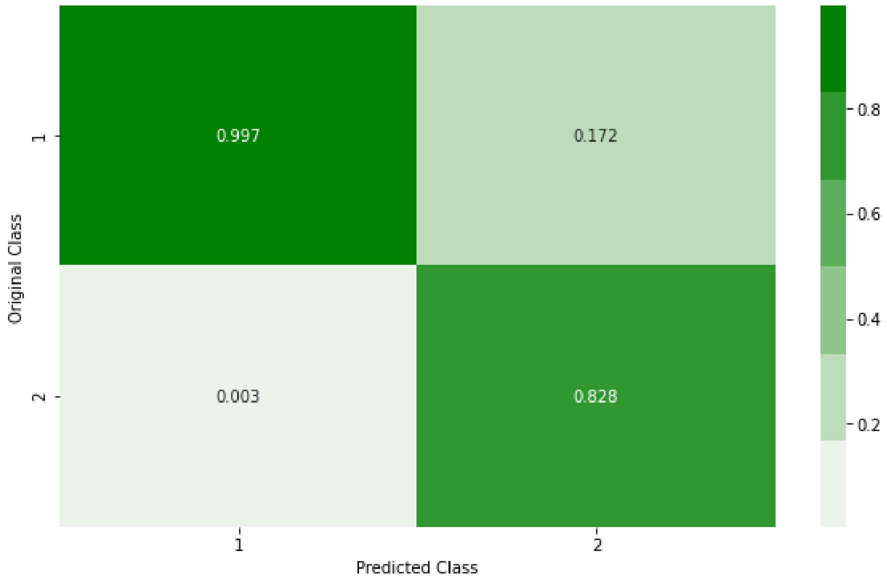
In [34]:

```
'''plotting results'''

import seaborn as sns
import matplotlib.pyplot as plt

plot_confusion_matrix(y_test,y_pred_lr)
```

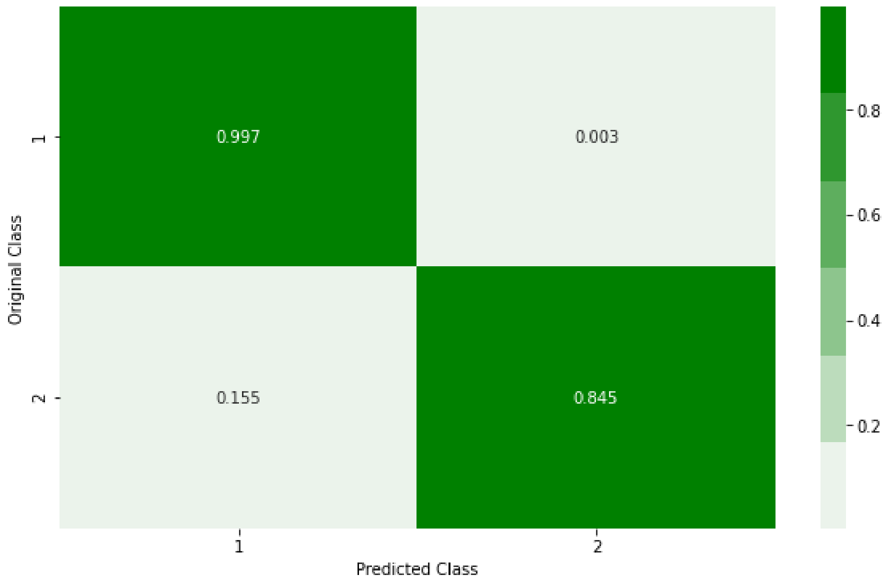
```
Number of misclassified points 66 out of 16001 points
Precision: 0.828
Recall: 0.845
-------------------------------------------------- Confusion matrix ------------------------------
----------------
```




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




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




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

AdaBoost

In [37]:

```
'''Hypertuning the Random Forest model using GridSearchCV'''

from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import AdaBoostClassifier
from sklearn import metrics
from sklearn.tree import DecisionTreeClassifier
```

In [ ]:

```
Ada = AdaBoostClassifier(DecisionTreeClassifier(max_depth=20,class_weight={0:1,1:59}))
param_grid = { 'n_estimators': [200,500,1000]}
CV_Ada = GridSearchCV(estimator=Ada, param_grid=param_grid, cv= 3, scoring='f1',verbose=5)

CV_Ada.fit(X_train,y_train)
```

```
Fitting 3 folds for each of 3 candidates, totalling 9 fits
[CV] n_estimators=200 ................................................

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

[CV] .................... n_estimators=200, score=0.768, total= 5.7min
[CV] n_estimators=200 ................................................

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  5.7min remaining:    0.0s

[CV] .................... n_estimators=200, score=0.743, total= 5.6min
[CV] n_estimators=200 ................................................

[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed: 11.3min remaining:    0.0s

[CV] .................... n_estimators=200, score=0.785, total= 6.2min
[CV] n_estimators=500 ................................................

[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed: 17.5min remaining:    0.0s

[CV] .................... n_estimators=500, score=0.771, total=12.3min
[CV] n_estimators=500 ................................................

[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed: 29.9min remaining:    0.0s

[CV] .................... n_estimators=500, score=0.743, total=12.2min
[CV] n_estimators=500 ................................................
[CV] .................... n_estimators=500, score=0.782, total=12.9min
[CV] n_estimators=1000 ...............................................
[CV] ................... n_estimators=1000, score=0.783, total=18.6min
[CV] n_estimators=1000 ...............................................
[CV] ................... n_estimators=1000, score=0.733, total=17.8min
[CV] n_estimators=1000 ...............................................
[CV] ................... n_estimators=1000, score=0.777, total=18.4min

[Parallel(n_jobs=1)]: Done   9 out of   9 | elapsed: 109.8min finished
```

Out[ ]:

```
GridSearchCV(cv=3, error_score=nan,
             estimator=AdaBoostClassifier(algorithm='SAMME.R',
                                          base_estimator=DecisionTreeClassifier(ccp_alpha=0.0,
                                                                                class_weight={0: 1,
                                                                                              1: 59
                                                                                criterion='gini',
                                                                                max_depth=20,
                                                                                max_features=None,
                                                                                max_leaf_nodes=None

min_impurity_decrease=0.0,

min_impurity_split=None,
                                                                                min_samples_leaf=1,
                                                                                min_samples_split=2

min_weight_fraction_leaf=0.0,
                                                                                presort='deprecated
                                                                                random_state=None,
                                                                                splitter='best'),
                                          learning_rate=1.0, n_estimators=50,
                                          random_state=None),
             iid='deprecated', n_jobs=None,
             param_grid={'n_estimators': [200, 500, 1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='f1', verbose=5)
```

In [ ]:

```
CV_Ada.best_params_
```

Out[ ]:

```
{'n_estimators': 500}
```

In [38]:

```
Ada = AdaBoostClassifier(DecisionTreeClassifier(max_depth=20,class_weight={0:1,1:59}), n_estimators
=500)
Ada.fit(X_train,y_train)
```

Out[38]:

```
AdaBoostClassifier(algorithm='SAMME.R',
                   base_estimator=DecisionTreeClassifier(ccp_alpha=0.0,
                                                         class_weight={0: 1,
                                                                       1: 59},
                                                         criterion='gini',
                                                         max_depth=20,
                                                         max_features=None,
                                                         max_leaf_nodes=None,
                                                         min_impurity_decrease=0.0,
                                                         min_impurity_split=None,
                                                         min_samples_leaf=1,
                                                         min_samples_split=2,
                                                         min_weight_fraction_leaf=0.0,
                                                         presort='deprecated',
                                                         random_state=None,
                                                         splitter='best'),
                   learning_rate=1.0, n_estimators=500, random_state=None)
```

In [39]:

```
y_pred_lr = Ada.predict(X_test)
f1=metrics.f1_score(y_pred_lr,y_test)
round(f1,3)
```

Out[39]:

```
0.784
```

In [40]:

```
y_pred_lr = Ada.predict(X_test)
f1=metrics.f1_score(y_pred_lr,y_test,average='macro')
round(f1,3)
```

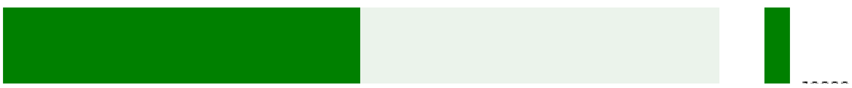
Out[40]:

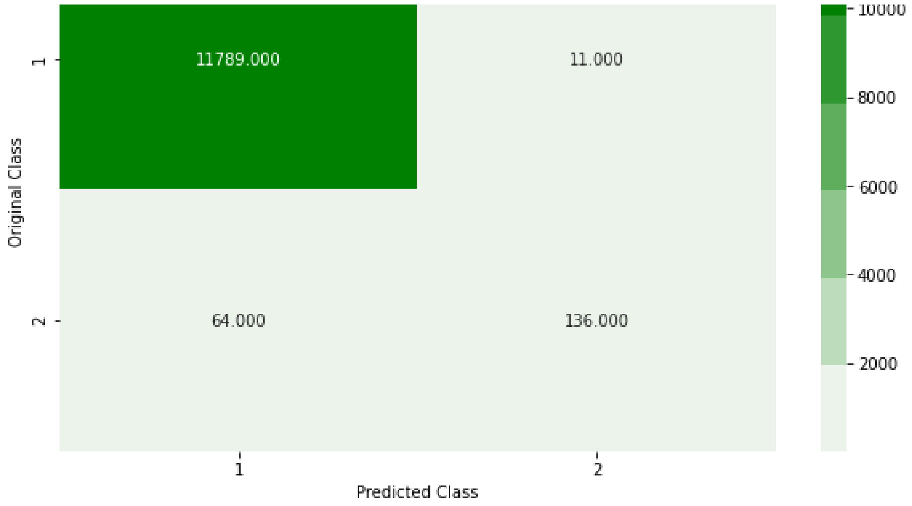
```
0.89
```

In [41]:

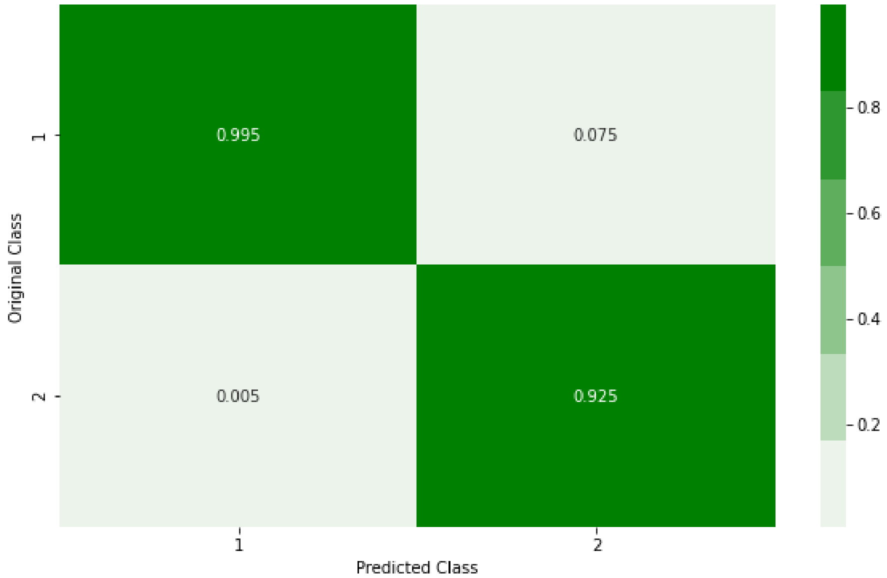
```
'''plotting results'''

import seaborn as sns
import matplotlib.pyplot as plt

plot_confusion_matrix(y_test,y_pred_lr)
```

```
Number of misclassified points 75 out of 16001 points
Precision: 0.925
Recall: 0.68
--------------------------------------------------- Confusion matrix -------------------------------
----------------
```

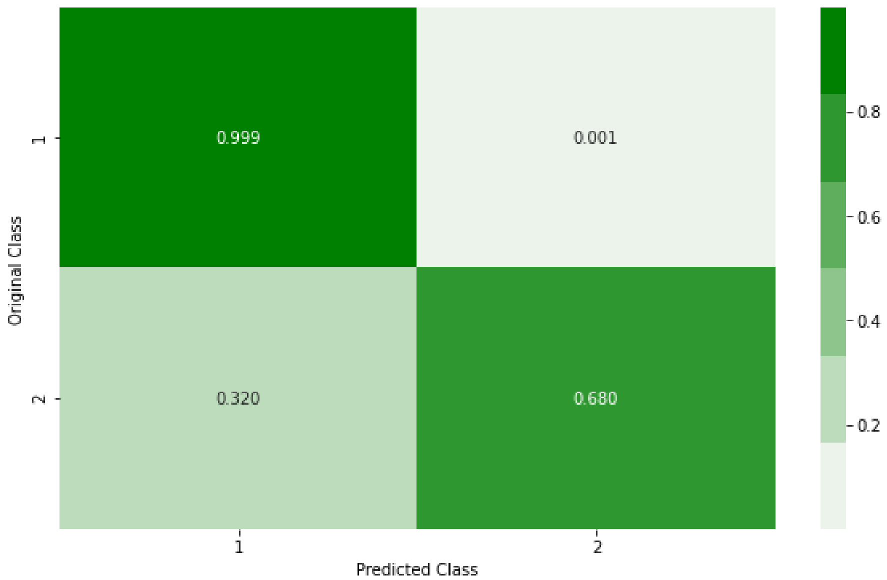





```
-------------------------------------------------- Precision matrix ------------------------------
----------------
```




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




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

Decision Tree

In [ ]:

```
DTC = DecisionTreeClassifier(class_weight={0:1,1:59})
param_grid = {'max_depth':[2,5,10,20,50,100,200]}
CV_DTC = GridSearchCV(estimator=DTC, param_grid=param_grid, cv= 5, scoring='f1',verbose=5)
```

In [ ]:

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

```
Fitting 5 folds for each of 7 candidates, totalling 35 fits
[CV] max_depth=2 .......................................................
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
[CV] ........................ max_depth=2, score=0.229, total=   0.5s
[CV] max_depth=2 .......................................................
```

```
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.5s remaining:    0.0s
```

```
[CV] ........................ max_depth=2, score=0.239, total=   0.5s
[CV] max_depth=2 .......................................................
```

```
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    1.0s remaining:    0.0s
```

```
[CV] ........................ max_depth=2, score=0.230, total=   0.5s
[CV] max_depth=2 .......................................................
```

```
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    1.5s remaining:    0.0s
```

```
[CV] ........................ max_depth=2, score=0.250, total=   0.6s
[CV] max_depth=2 .......................................................
```

```
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    2.1s remaining:    0.0s
```

```
[CV] ........................ max_depth=2, score=0.241, total=   0.5s
[CV] max_depth=5 .......................................................
[CV] ........................ max_depth=5, score=0.345, total=   1.2s
[CV] max_depth=5 .......................................................
[CV] ........................ max_depth=5, score=0.398, total=   1.2s
[CV] max_depth=5 .......................................................
[CV] ........................ max_depth=5, score=0.402, total=   1.2s
[CV] max_depth=5 .......................................................
[CV] ........................ max_depth=5, score=0.408, total=   1.2s
[CV] max_depth=5 .......................................................
[CV] ........................ max_depth=5, score=0.368, total=   1.2s
[CV] max_depth=10 ......................................................
[CV] ....................... max_depth=10, score=0.454, total=   2.2s
[CV] max_depth=10 ......................................................
[CV] ....................... max_depth=10, score=0.529, total=   2.2s
[CV] max_depth=10 ......................................................
[CV] ....................... max_depth=10, score=0.485, total=   2.0s
[CV] max_depth=10 ......................................................
[CV] ....................... max_depth=10, score=0.531, total=   2.2s
[CV] max_depth=10 ......................................................
[CV] ....................... max_depth=10, score=0.483, total=   2.2s
[CV] max_depth=20 ......................................................
[CV] ....................... max_depth=20, score=0.490, total=   2.3s
[CV] max_depth=20 ......................................................
[CV] ....................... max_depth=20, score=0.573, total=   2.9s
[CV] max_depth=20 ......................................................
[CV] ....................... max_depth=20, score=0.536, total=   2.1s
[CV] max_depth=20 ......................................................
[CV] ....................... max_depth=20, score=0.585, total=   2.2s
```

```
[CV] max_depth=20 ....................................................
[CV] ....................... max_depth=20, score=0.559, total=   2.9s
[CV] max_depth=50 ....................................................
[CV] ....................... max_depth=50, score=0.544, total=   2.3s
[CV] max_depth=50 ....................................................
[CV] ....................... max_depth=50, score=0.629, total=   2.9s
[CV] max_depth=50 ....................................................
[CV] ....................... max_depth=50, score=0.606, total=   2.1s
[CV] max_depth=50 ....................................................
[CV] ....................... max_depth=50, score=0.627, total=   2.3s
[CV] max_depth=50 ....................................................
[CV] ....................... max_depth=50, score=0.624, total=   2.9s
[CV] max_depth=100 ...................................................
[CV] ...................... max_depth=100, score=0.523, total=   2.3s
[CV] max_depth=100 ...................................................
[CV] ...................... max_depth=100, score=0.632, total=   2.9s
[CV] max_depth=100 ...................................................
[CV] ...................... max_depth=100, score=0.600, total=   2.2s
[CV] max_depth=100 ...................................................
[CV] ...................... max_depth=100, score=0.658, total=   2.3s
[CV] max_depth=100 ...................................................
[CV] ...................... max_depth=100, score=0.649, total=   2.9s
[CV] max_depth=200 ...................................................
[CV] ...................... max_depth=200, score=0.558, total=   2.3s
[CV] max_depth=200 ...................................................
[CV] ...................... max_depth=200, score=0.622, total=   2.9s
[CV] max_depth=200 ...................................................
[CV] ...................... max_depth=200, score=0.575, total=   2.2s
[CV] max_depth=200 ...................................................
[CV] ...................... max_depth=200, score=0.652, total=   2.3s
[CV] max_depth=200 ...................................................
[CV] ...................... max_depth=200, score=0.638, total=   2.9s
```

```
[Parallel(n_jobs=1)]: Done  35 out of  35 | elapsed:  1.2min finished
```

Out[ ]:

```
GridSearchCV(cv=5, error_score=nan,
             estimator=DecisionTreeClassifier(ccp_alpha=0.0,
                                              class_weight={0: 1, 1: 59},
                                              criterion='gini', max_depth=None,
                                              max_features=None,
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              presort='deprecated',
                                              random_state=None,
                                              splitter='best'),
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': [2, 5, 10, 20, 50, 100, 200]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='f1', verbose=5)
```

In [ ]:

```
CV_DTC.best_params_
```

Out[ ]:

```
{'max_depth': 100}
```

In [42]:

```
DTC = DecisionTreeClassifier(class_weight={0:1,1:59},max_depth=100)
DTC.fit(X_train,y_train)
```

Out[42]:

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight={0: 1, 1: 59},
                       criterion='gini', max_depth=100, max_features=None,
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
```

```
                   max_leaf_nodes=None, min_impurity_decrease=0.0,
                   min_impurity_split=None, min_samples_leaf=1,
                   min_samples_split=2, min_weight_fraction_leaf=0.0,
                   presort='deprecated', random_state=None,
                   splitter='best')
```

In [43]:

```
y_pred_lr = DTC.predict(X_test)
f1=metrics.f1_score(y_pred_lr,y_test)
round(f1,3)
```

Out[43]:

```
0.612
```

In [44]:

```
y_pred_lr = DTC.predict(X_test)
f1=metrics.f1_score(y_pred_lr,y_test,average='macro')
round(f1,3)
```

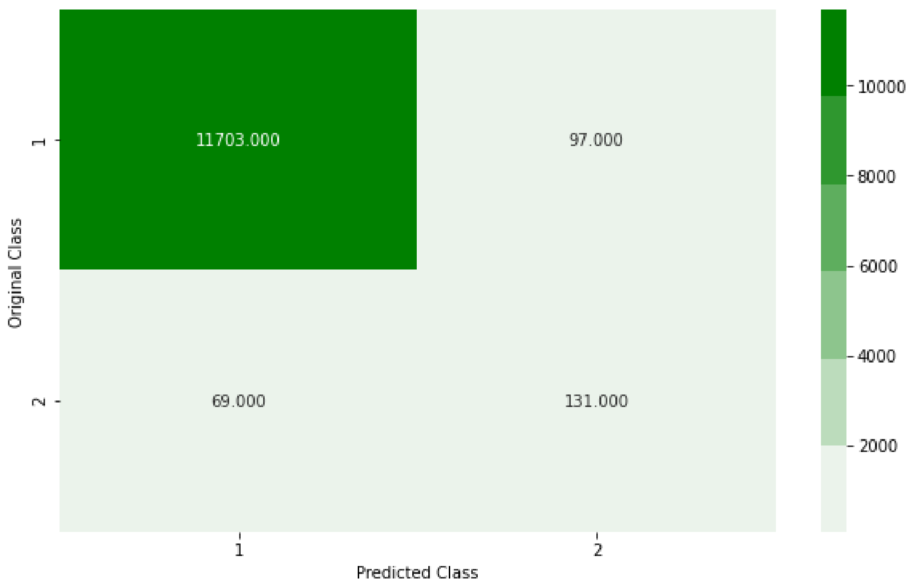
Out[44]:

```
0.803
```

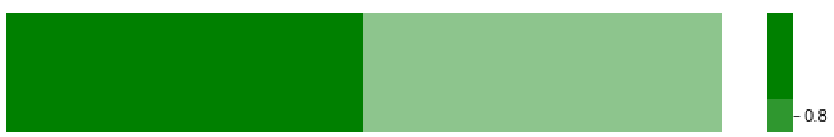
In [45]:

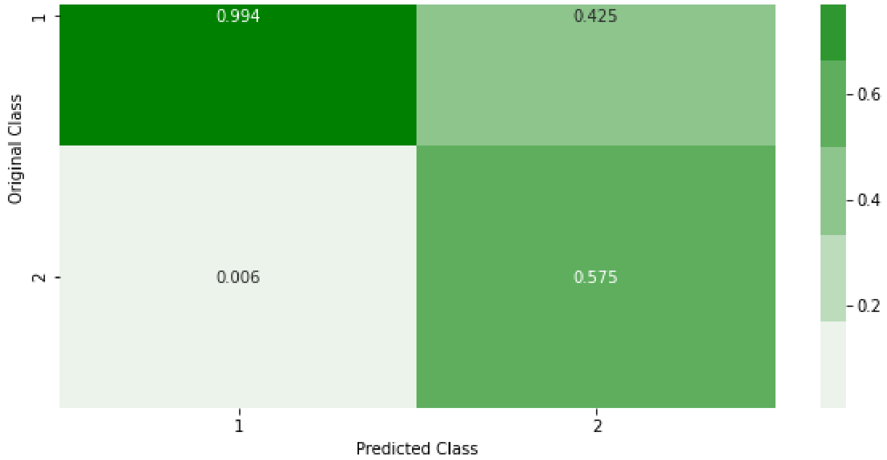
```
'''plotting results'''

import seaborn as sns
import matplotlib.pyplot as plt

plot_confusion_matrix(y_test,y_pred_lr)
```

```
Number of misclassified points 166 out of 16001 points
Precision: 0.575
Recall: 0.655
---------------------------------------------------- Confusion matrix ------------------------------
----------------
```

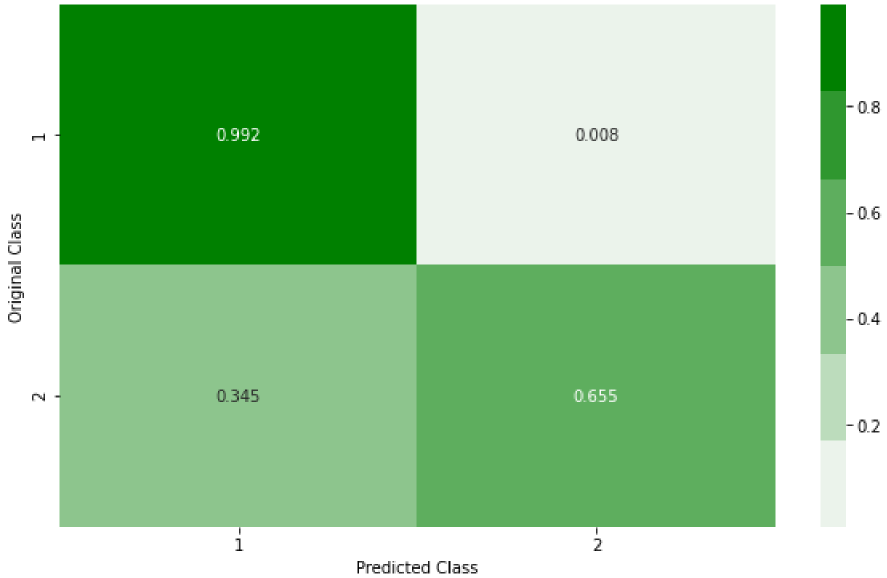



```
---------------------------------------------------- Precision matrix ------------------------------
----------------
```






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




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

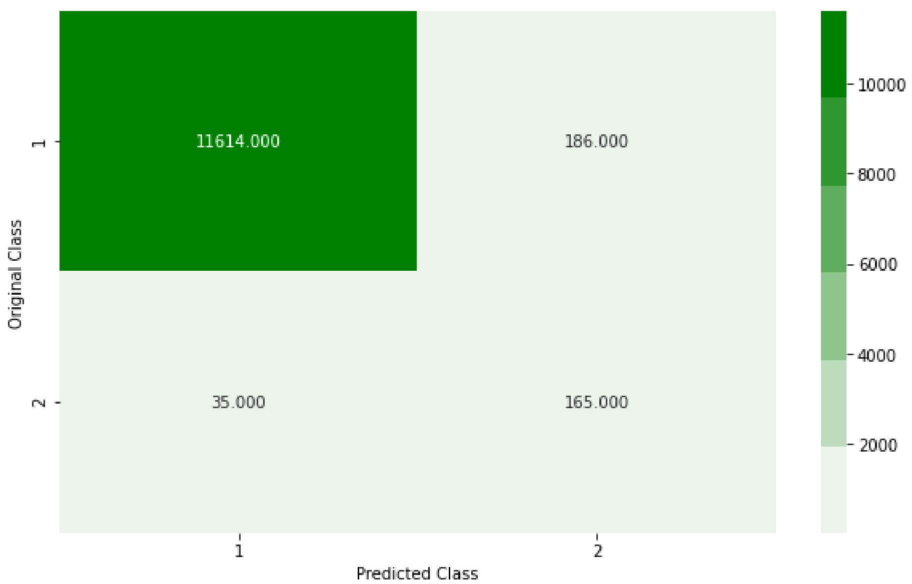
Custom ensemble 1

In [46]:

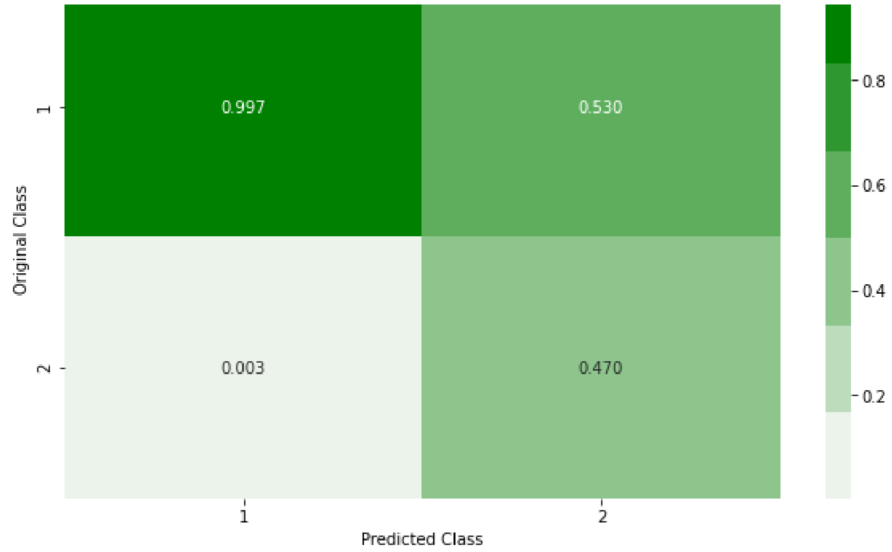
```
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
import xgboost as xgb
from sklearn.ensemble import AdaBoostClassifier
from xgboost.sklearn import XGBClassifier
from sklearn.tree import DecisionTreeClassifier

X1, X2, y1, y2 = train_test_split(X_train, y_train, test_size=0.5, random_state=42, stratify=y_trai
n)

_, d1, _, dy1 = train_test_split(X1, y1, test_size=0.25, stratify=y1)
_, d2, _, dy2 = train_test_split(X1, y1, test_size=0.25, stratify=y1)
_, d3, _, dy3 = train_test_split(X1, y1, test_size=0.25, stratify=y1)
_, d4, _, dy4 = train_test_split(X1, y1, test_size=0.25, stratify=y1)

RF_final = RandomForestClassifier(max_depth=20,n_estimators=1000,class_weight={0:1,1:59})
RF_final.fit(d1,dy1)
xgb_model = xgb.XGBClassifier(n_jobs=-1,scale_pos_weight=59, colsample_bylevel=0.7,colsample_bytree
=1,gamma=1.0,learning_rate=0.1,max_depth=20,min_child_weight=5.0,n_estimators=1000,reg_lambda=1.0,s
ubsample=0.5)
xgb_model.fit(d2, dy2)
AdaB = AdaBoostClassifier(n_estimators=500, random_state=0)
```

```
AdaB.fit(d3, dy3)
DTC = DecisionTreeClassifier(class_weight={0:1,1:59},max_depth=100)
DTC.fit(d4, dy4)

pred1 = RF_final.predict(X2)
pred2 = xgb_model.predict(X2)
pred3 = AdaB.predict(X2)
pred4 = DTC.predict(X2)

df_list = [pred1,pred2,pred3,pred4]

df = pd.DataFrame(df_list)
df = df.transpose()

xgb_model_F = xgb.XGBClassifier(n_jobs=-1,scale_pos_weight=59, colsample_bylevel=0.7,colsample_bytr
ee=1,gamma=1.0,learning_rate=0.1,max_depth=20,min_child_weight=5.0,n_estimators=1000,reg_lambda=1.0
,subsample=0.5)
xgb_model_F.fit(df, y2)

pred1 = RF_final.predict(X_test)
pred2 = xgb_model.predict(X_test)
pred3 = AdaB.predict(X_test)
pred4 = DTC.predict(X_test)

df_list = [pred1,pred2,pred3,pred4]

df_test = pd.DataFrame(df_list)
df_test = df_test.transpose()

test_pred = xgb_model_F.predict(df_test)

f1=metrics.f1_score(test_pred,y_test,average='macro')
print('The Macro F1 score for validation set is:',f1,'\n')

import matplotlib.pyplot as plt
import seaborn as sns

plot_confusion_matrix(y_test,test_pred)
```

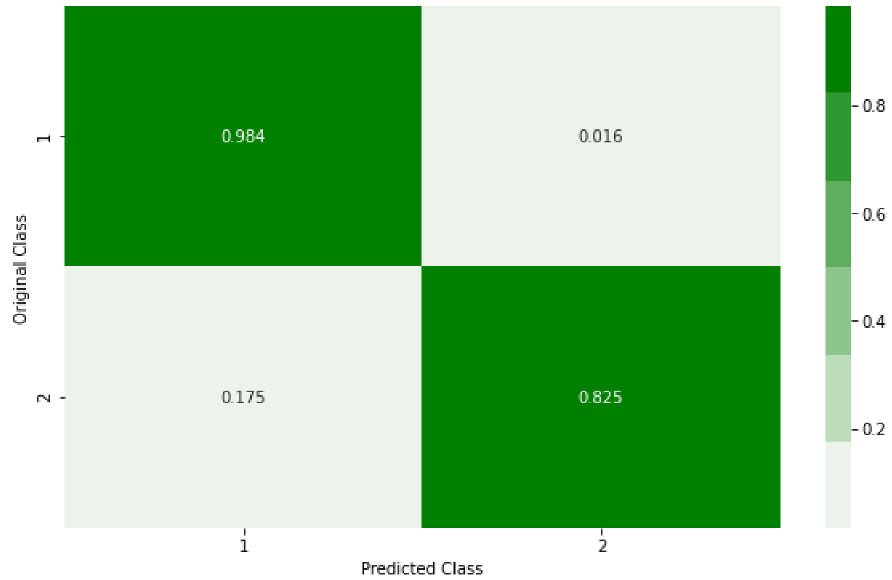
```
The Macro F1 score for validation set is: 0.7947431809187936

Number of misclassified points 221 out of 16001 points
Precision: 0.47
Recall: 0.825
---------------------------------------------------- Confusion matrix ------------------------------------------------
```




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




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




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

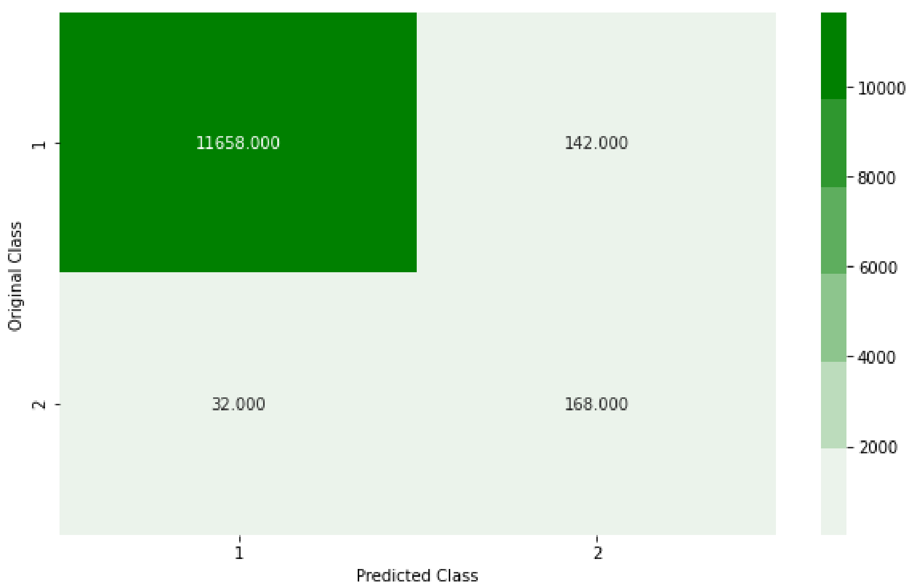
Custom ensemble 2

In [47]:

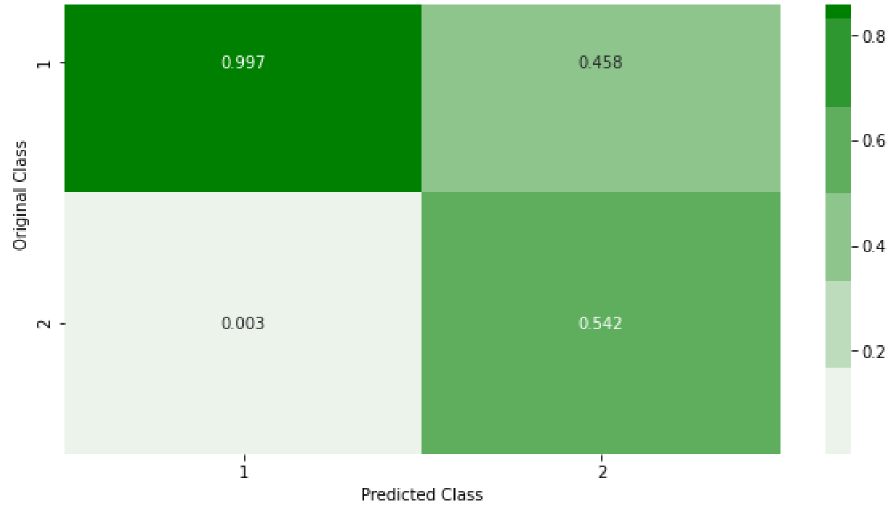
```
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
import xgboost as xgb
from sklearn.ensemble import AdaBoostClassifier
from xgboost.sklearn import XGBClassifier
from sklearn.tree import DecisionTreeClassifier

X1, X2, y1, y2 = train_test_split(X_train, y_train, test_size=0.5, random_state=42, stratify=y_trai
n)

RF_final = RandomForestClassifier(max_depth=20,n_estimators=1000,class_weight={0:1,1:59})
RF_final.fit(X1,y1)
xgb_model = xgb.XGBClassifier(n_jobs=-1,scale_pos_weight=59, colsample_bylevel=0.7,colsample_bytree
=1,gamma=1.0,learning_rate=0.1,max_depth=20,min_child_weight=5.0,n_estimators=1000,reg_lambda=1.0,s
ubsample=0.5)
xgb_model.fit(X1,y1)
AdaB = AdaBoostClassifier(n_estimators=500, random_state=0)
AdaB.fit(X1,y1)
DTC = DecisionTreeClassifier(class_weight={0:1,1:59},max_depth=100)
```

```
DTC = DecisionTreeClassifier(class_weight={0:1,1:59},max_depth=100)
DTC.fit(X1,y1)

pred1 = RF_final.predict(X2)
pred2 = xgb_model.predict(X2)
pred3 = AdaB.predict(X2)
pred4 = DTC.predict(X2)

df_list = [pred1,pred2,pred3,pred4]

df = pd.DataFrame(df_list)
df = df.transpose()

xgb_model_F = xgb.XGBClassifier(n_jobs=-1,scale_pos_weight=59, colsample_bylevel=0.7,colsample_bytr
ee=1,gamma=1.0,learning_rate=0.1,max_depth=20,min_child_weight=5.0,n_estimators=1000,reg_lambda=1.0
,subsample=0.5)
xgb_model_F.fit(df, y2)

pred1 = RF_final.predict(X_test)
pred2 = xgb_model.predict(X_test)
pred3 = AdaB.predict(X_test)
pred4 = DTC.predict(X_test)

df_list = [pred1,pred2,pred3,pred4]

df_test = pd.DataFrame(df_list)
df_test = df_test.transpose()

test_pred = xgb_model_F.predict(df_test)

f1=metrics.f1_score(test_pred,y_test,average='macro')
print('The Macro F1 score for validation set is:',f1,'\n')

import matplotlib.pyplot as plt
import seaborn as sns

plot_confusion_matrix(y_test,test_pred)
```

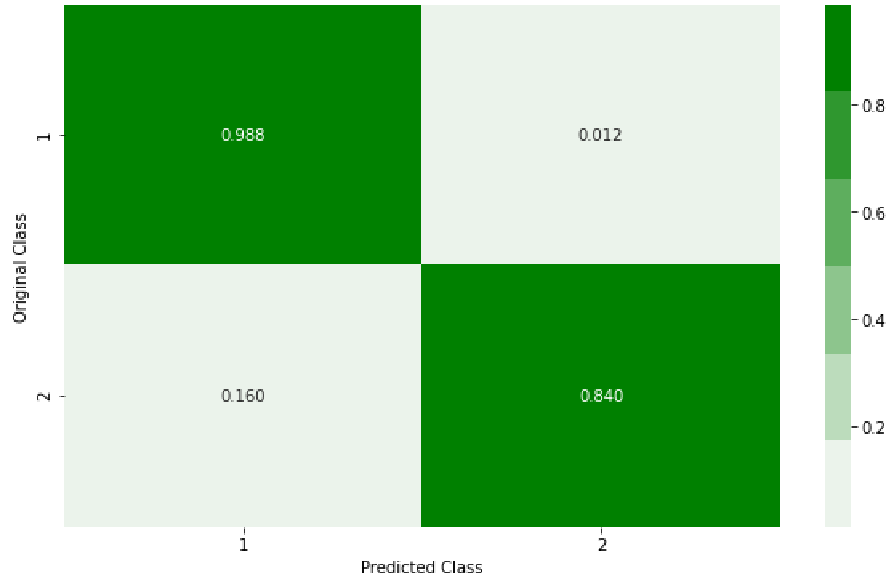
```
The Macro F1 score for validation set is: 0.8257080610021788

Number of misclassified points 174 out of 16001 points
Precision: 0.542
Recall: 0.84
--------------------------------------------------- Confusion matrix -------------------------------
----------------
```




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




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




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

Custom ensemble 3:

In [ ]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
import xgboost as xgb
from sklearn.ensemble import AdaBoostClassifier
from xgboost.sklearn import XGBClassifier
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt
import seaborn as sns

def custom_ensemble(df_train,dfy_train,df_test,dfy_test,n_estimators,val):
  X1, X2, y1, y2 = train_test_split(df_train, dfy_train, test_size=0.5, random_state=42, stratify=d
fy_train)
  preds=[]
  pred_test=[]
  for i in range(n_estimators):
    _, d1, _, dy1 = train_test_split(X1, y1, test_size=0.25, stratify=y1,random_state=i)
    DTC = DecisionTreeClassifier(max_depth=val)
    DTC.fit(d1, dy1)
    pred_test.append(DTC.predict(X_test))
    preds.append(DTC.predict(X2))
```

```
  new_df = pd.DataFrame(preds)
  new_df_test = pd.DataFrame(pred_test)
  new_df = new_df.transpose()
  new_df_test = new_df_test.transpose()
  xgb_model = xgb.XGBClassifier(n_jobs=-1,scale_pos_weight=59, colsample_bylevel=0.7,colsample_bytr
ee=1,gamma=1.0,learning_rate=0.1,max_depth=20,min_child_weight=5.0,n_estimators=500,reg_lambda=1.0,
subsample=0.5)
  xgb_model.fit(new_df, y2)
  y_pred = xgb_model.predict(new_df_test)
  f1=metrics.f1_score(y_pred,dfy_test)
  return f1

val_list=[50,100,200,500,1000,1500,2000,3000,5000]
f1_list=[]
for k in val_list:
  f1_list.append(round(custom_ensemble(X_train,y_train,X_test,y_test,4,val=k),3))

from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Depth", "F1 score"]
for i in range(len(val_list)):
  x.add_row([val_list[i],f1_list[i]])
print(x)
```

```
+-------+----------+
| Depth | F1 score |
+-------+----------+
|   50  |  0.566   |
|  100  |   0.54   |
|  200  |   0.55   |
|  500  |  0.538   |
|  1000 |   0.55   |
|  1500 |   0.53   |
|  2000 |  0.535   |
|  3000 |  0.526   |
|  5000 |  0.524   |
+-------+----------+
```

Observation: Best result seen for max_depth 50

In [ ]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
import xgboost as xgb
from sklearn.ensemble import AdaBoostClassifier
from xgboost.sklearn import XGBClassifier
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt
import seaborn as sns

def custom_ensemble(df_train,dfy_train,df_test,dfy_test,n_estimators):
  X1, X2, y1, y2 = train_test_split(df_train, dfy_train, test_size=0.5, random_state=42, stratify=d
fy_train)
  preds=[]
  pred_test=[]
  for i in range(n_estimators):
    _, d1, _, dy1 = train_test_split(X1, y1, test_size=0.25, stratify=y1,random_state=i)
    DTC = DecisionTreeClassifier(max_depth=50)
    DTC.fit(d1, dy1)
    pred_test.append(DTC.predict(X_test))
    preds.append(DTC.predict(X2))
  new_df = pd.DataFrame(preds)
  new_df_test = pd.DataFrame(pred_test)
  new_df = new_df.transpose()
  new_df_test = new_df_test.transpose()
  xgb_model = xgb.XGBClassifier(n_jobs=-1,scale_pos_weight=59, colsample_bylevel=0.7,colsample_bytr
ee=1,gamma=1.0,learning_rate=0.1,max_depth=20,min_child_weight=5.0,n_estimators=500,reg_lambda=1.0,
subsample=0.5)
  xgb_model.fit(new_df, y2)
  y_pred = xgb_model.predict(new_df_test)
  f1=metrics.f1_score(y_pred,dfy_test)
  return f1

val_list=[3,4,5,6,8,10]
```

```
val_list=[3,4,5,6,8,10]
f1_list=[]
for k in val_list:
  f1_list.append(round(custom_ensemble(X_train,y_train,X_test,y_test,k),3))

from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["n_estimators", "F1 score"]
for i in range(len(val_list)):
  x.add_row([val_list[i],f1_list[i]])
print(x)
```

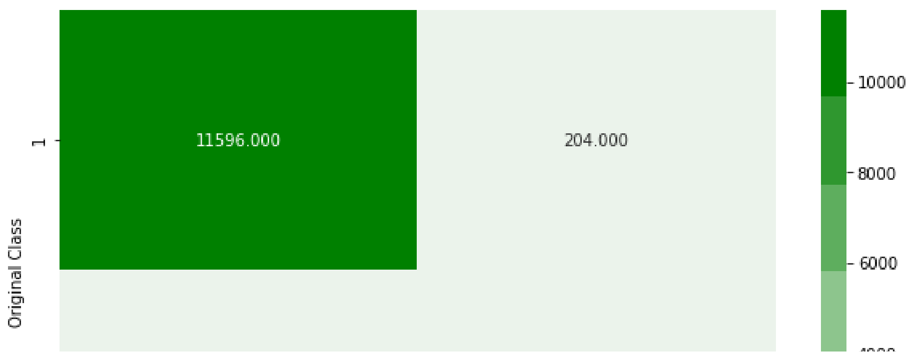
```
+--------------+----------+
| n_estimators | F1 score |
+--------------+----------+
|      3       |  0.567   |
|      4       |  0.541   |
|      5       |  0.524   |
|      6       |  0.492   |
|      8       |  0.466   |
|      10      |  0.503   |
+--------------+----------+
```

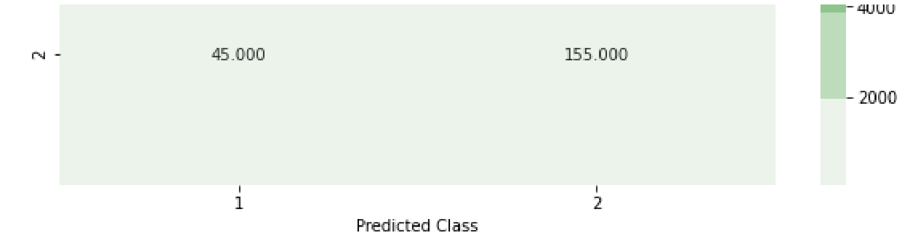
Observation: Best result seen for n_estimators=3

In [48]:

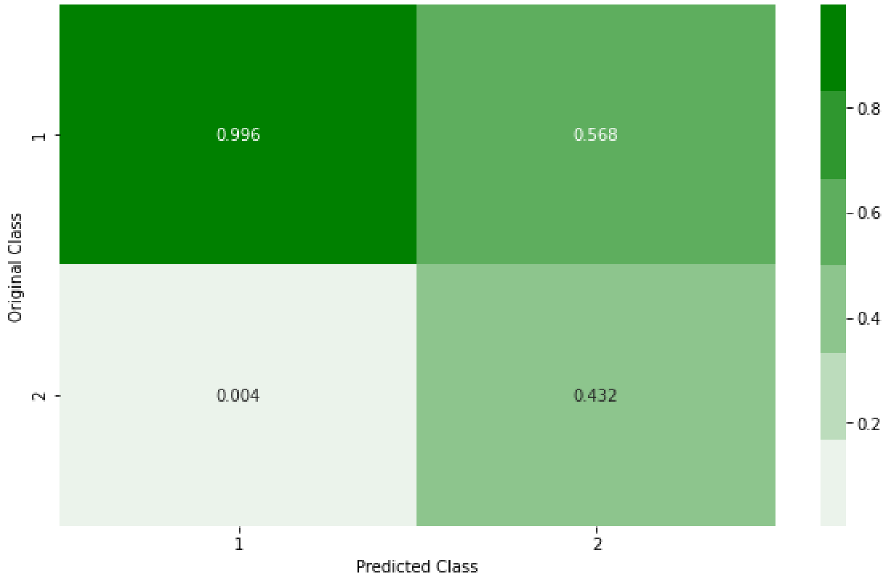
```
def custom_ensemble(df_train,dfy_train,df_test,dfy_test,n_estimators):
  X1, X2, y1, y2 = train_test_split(df_train, dfy_train, test_size=0.5, random_state=42, stratify=d
fy_train)
  preds=[]
  pred_test=[]
  for i in range(n_estimators):
    _, d1, _, dy1 = train_test_split(X1, y1, test_size=0.25, stratify=y1,random_state=i)
    DTC = DecisionTreeClassifier(max_depth=50)
    DTC.fit(d1, dy1)
    pred_test.append(DTC.predict(X_test))
    preds.append(DTC.predict(X2))
  new_df = pd.DataFrame(preds)
  new_df_test = pd.DataFrame(pred_test)
  new_df = new_df.transpose()
  new_df_test = new_df_test.transpose()
  xgb_model = xgb.XGBClassifier(n_jobs=-1,scale_pos_weight=59, colsample_bylevel=0.7,colsample_bytr
ee=1,gamma=1.0,learning_rate=0.1,max_depth=20,min_child_weight=5.0,n_estimators=500,reg_lambda=1.0,
subsample=0.5)
  xgb_model.fit(new_df, y2)
  y_pred = xgb_model.predict(new_df_test)
  f1=metrics.f1_score(test_pred,y_test,average='macro')
  print('The Macro F1 score for validation set is:',f1,'\n')
  plot_confusion_matrix(dfy_test,y_pred)

custom_ensemble(X_train,y_train,X_test,y_test,3)
```

```
The Macro F1 score for validation set is: 0.8257080610021788

Number of misclassified points 249 out of 16001 points
Precision: 0.432
Recall: 0.775
---------------------------------------------------- Confusion matrix --------------------------------
----------------
```

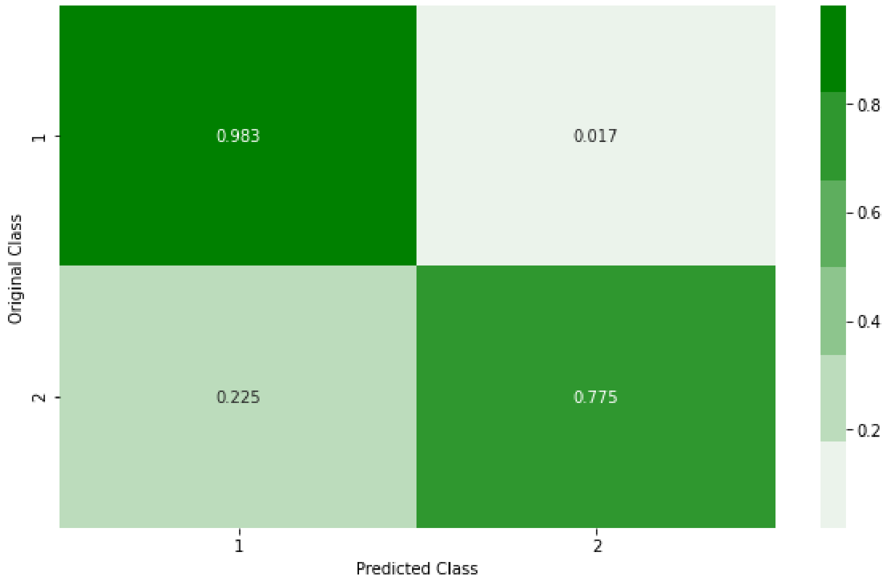





```
---------------------------------------------------- Precision matrix ----------------------------------------------------
```




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




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

ANN

In [118]:

```
from tensorflow.keras.layers import Dense, Conv2D, Conv1D, Flatten, Dropout, MaxPooling2D, MaxPool1D, Concatenate, LSTM
import tensorflow as tf
```

```
import tensorflow.keras.backend as K
from tensorflow.keras.callbacks import Callback,ModelCheckpoint
from sklearn.metrics import auc
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense,Input,Conv2D,MaxPool2D,Activation,Dropout,Flatten,
concatenate
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Embedding
from tensorflow.keras.regularizers import l2
import numpy as np
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

In [119]:

```
tf.keras.backend.clear_session()
from tensorflow.keras import regularizers
from tensorflow.keras.layers import BatchNormalization

%load_ext tensorboard

input = Input(shape=len(X_train.columns))
dense1 = Dense(50,activation='relu')(input)
output = Dense(2,activation='sigmoid')(dense1)

model = Model(inputs=input, outputs=output)
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

In [120]:

```
import tensorflow_addons as tfa

adam = tf.keras.optimizers.Adam(lr=0.01)
model.compile(optimizer=adam, loss='binary_crossentropy',metrics=[tfa.metrics.F1Score(num_classes=
2,average='macro')])
```

In [121]:

```
neg, pos = np.bincount(y_train)
total = neg + pos
print('Total: {}\n    Positive: {} ({:.2f}% of total)\n'.format(
    total, pos, 100 * pos / total))
```

```
Total: 48000
    Positive: 800 (1.67% of total)
```

In [122]:

```
weight_for_0 = (1 / neg)*(total)/2.0
weight_for_1 = (1 / pos)*(total)/2.0

class_weight = {0: weight_for_0, 1: weight_for_1}

print('Weight for class 0: {:.2f}'.format(weight_for_0))
print('Weight for class 1: {:.2f}'.format(weight_for_1))
```

```
Weight for class 0: 0.51
Weight for class 1: 30.00
```

In [123]:

```
y_train1 = tf.keras.utils.to_categorical(y_train, 2)
y_test1 = tf.keras.utils.to_categorical(y_test, 2)
```

In [124]:

```
X_train1 = np.asarray(X_train)
X_test1 = np.asarray(X_test)
```

In [125]:

```
callback_list = []
log_dir="Model-1-logs"
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,histogram_freq=1, write_graph
=True,write_grads=True)
callback_list.append(tensorboard_callback)
filepath="Model-1-weights.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_loss', verbose=0, save_best_only=True,
mode='auto')
callback_list.append(checkpoint)
```

```
WARNING:tensorflow:`write_grads` will be ignored in TensorFlow 2.0 for the `TensorBoard` Callback.
```

In [126]:

```
model.fit(X_train1, y_train1,batch_size=50, epochs=30, validation_data=(X_test1, y_test1),callbacks
=callback_list,class_weight=class_weight)
```

```
Epoch 1/30
  1/960 [..............................] - ETA: 0s - loss: 0.7623 - f1_score:
0.4845WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/tensorflow/python/ops/summary_ops_v2.py:1277: stop (from
tensorflow.python.eager.profiler) is deprecated and will be removed after 2020-07-01.
Instructions for updating:
use `tf.profiler.experimental.stop` instead.
  2/960 [..............................] - ETA: 30s - loss: 0.7698 - f1_score:
0.3865WARNING:tensorflow:Callbacks method `on_train_batch_end` is slow compared to the batch time
(batch time: 0.0109s vs `on_train_batch_end` time: 0.0525s). Check your callbacks.
960/960 [==============================] - 5s 5ms/step - loss: 0.2214 - f1_score: 0.6461 - val_los
s: 0.1929 - val_f1_score: 0.6839
Epoch 2/30
960/960 [==============================] - 4s 5ms/step - loss: 0.1722 - f1_score: 0.6747 - val_los
s: 0.1270 - val_f1_score: 0.7061
Epoch 3/30
960/960 [==============================] - 4s 5ms/step - loss: 0.1680 - f1_score: 0.6777 - val_los
s: 0.1246 - val_f1_score: 0.7094
Epoch 4/30
960/960 [==============================] - 4s 5ms/step - loss: 0.1564 - f1_score: 0.6858 - val_los
s: 0.1283 - val_f1_score: 0.6905
Epoch 5/30
960/960 [==============================] - 4s 5ms/step - loss: 0.1493 - f1_score: 0.6859 - val_los
s: 0.1198 - val_f1_score: 0.7015
Epoch 6/30
960/960 [==============================] - 4s 5ms/step - loss: 0.1397 - f1_score: 0.6919 - val_los
s: 0.1042 - val_f1_score: 0.7240
Epoch 7/30
960/960 [==============================] - 4s 5ms/step - loss: 0.1409 - f1_score: 0.6944 - val_los
s: 0.1399 - val_f1_score: 0.6904
Epoch 8/30
960/960 [==============================] - 4s 5ms/step - loss: 0.1339 - f1_score: 0.7066 - val_los
s: 0.1078 - val_f1_score: 0.7240
Epoch 9/30
960/960 [==============================] - 4s 5ms/step - loss: 0.1393 - f1_score: 0.6980 - val_los
s: 0.0940 - val_f1_score: 0.7275
Epoch 10/30
960/960 [==============================] - 4s 5ms/step - loss: 0.1314 - f1_score: 0.6969 - val_los
s: 0.1124 - val_f1_score: 0.7083
Epoch 11/30
960/960 [==============================] - 4s 5ms/step - loss: 0.1317 - f1_score: 0.7100 - val_los
s: 0.1237 - val_f1_score: 0.7019
Epoch 12/30
960/960 [==============================] - 5s 5ms/step - loss: 0.1275 - f1_score: 0.7049 - val_los
s: 0.0807 - val_f1_score: 0.7491
Epoch 13/30
960/960 [==============================] - 5s 5ms/step - loss: 0.1245 - f1_score: 0.7086 - val_los
s: 0.1027 - val_f1_score: 0.7224
Epoch 14/30
960/960 [==============================] - 4s 5ms/step - loss: 0.1337 - f1_score: 0.7091 - val_los
s: 0.1343 - val_f1_score: 0.6973
Epoch 15/30
```

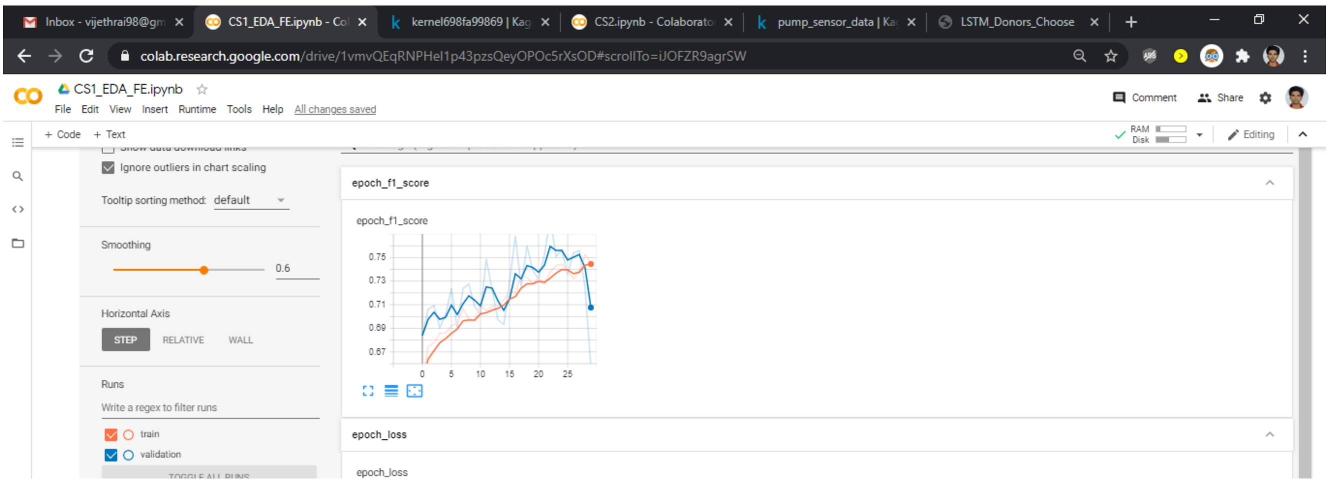
```
Epoch 15/30
960/960 [==============================] - 4s 5ms/step - loss: 0.1227 - f1_score: 0.7140 - val_los
s: 0.1338 - val_f1_score: 0.6932
Epoch 16/30
960/960 [==============================] - 4s 5ms/step - loss: 0.1176 - f1_score: 0.7222 - val_los
s: 0.0991 - val_f1_score: 0.7288
Epoch 17/30
960/960 [==============================] - 4s 5ms/step - loss: 0.1223 - f1_score: 0.7202 - val_los
s: 0.0924 - val_f1_score: 0.7685
Epoch 18/30
960/960 [==============================] - 4s 5ms/step - loss: 0.1237 - f1_score: 0.7349 - val_los
s: 0.0959 - val_f1_score: 0.7255
Epoch 19/30
960/960 [==============================] - 4s 4ms/step - loss: 0.1135 - f1_score: 0.7331 - val_los
s: 0.0817 - val_f1_score: 0.7600
Epoch 20/30
960/960 [==============================] - 4s 5ms/step - loss: 0.1162 - f1_score: 0.7280 - val_los
s: 0.0950 - val_f1_score: 0.7392
Epoch 21/30
960/960 [==============================] - 4s 5ms/step - loss: 0.1161 - f1_score: 0.7329 - val_los
s: 0.0995 - val_f1_score: 0.7317
Epoch 22/30
960/960 [==============================] - 4s 5ms/step - loss: 0.1118 - f1_score: 0.7278 - val_los
s: 0.0896 - val_f1_score: 0.7521
Epoch 23/30
960/960 [==============================] - 4s 4ms/step - loss: 0.1124 - f1_score: 0.7379 - val_los
s: 0.0718 - val_f1_score: 0.7836
Epoch 24/30
960/960 [==============================] - 4s 5ms/step - loss: 0.1113 - f1_score: 0.7432 - val_los
s: 0.0946 - val_f1_score: 0.7507
Epoch 25/30
960/960 [==============================] - 4s 5ms/step - loss: 0.1073 - f1_score: 0.7444 - val_los
s: 0.0825 - val_f1_score: 0.7564
Epoch 26/30
960/960 [==============================] - 4s 5ms/step - loss: 0.1117 - f1_score: 0.7389 - val_los
s: 0.1173 - val_f1_score: 0.7356
Epoch 27/30
960/960 [==============================] - 4s 5ms/step - loss: 0.1075 - f1_score: 0.7320 - val_los
s: 0.0891 - val_f1_score: 0.7538
Epoch 28/30
960/960 [==============================] - 4s 5ms/step - loss: 0.1093 - f1_score: 0.7393 - val_los
s: 0.0855 - val_f1_score: 0.7560
Epoch 29/30
960/960 [==============================] - 4s 5ms/step - loss: 0.1040 - f1_score: 0.7524 - val_los
s: 0.1104 - val_f1_score: 0.7245
Epoch 30/30
960/960 [==============================] - 4s 4ms/step - loss: 0.1058 - f1_score: 0.7461 - val_los
s: 0.2427 - val_f1_score: 0.6571
```

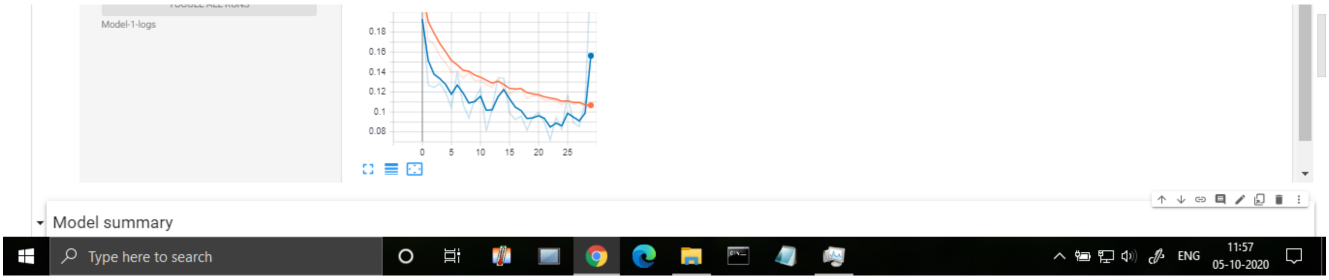
Out[126]:

```
<tensorflow.python.keras.callbacks.History at 0x7fbad3b9c080>
```

In [127]:

```
%tensorboard --logdir 'Model-1-logs'
```





In [132]:

```
model.load_weights('/content/Model-1-weights.hdf5')
```

In [135]:

```
loss,f1 = model.evaluate(X_test1,  y_test1, verbose=2)
print("Restored model, f1 score: {:5.2f}".format(f1))
```

```
375/375 - 1s - loss: 0.0718 - f1_score: 0.7836
Restored model, f1 score:  0.78
```

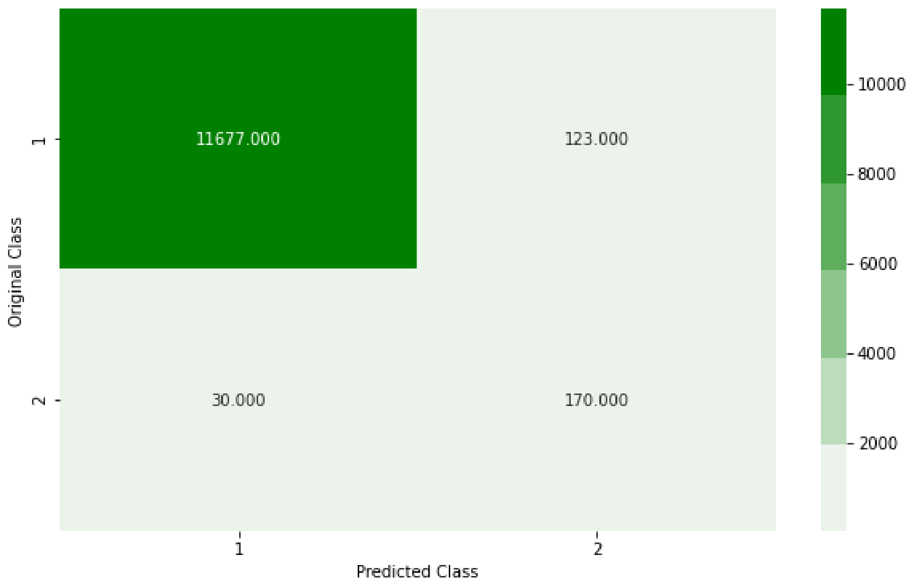
In [155]:

```
yy=np.zeros(len(y_test))
for i in range(len(y_pred_lr)):
  if y_pred_lr[i][1]>0.8:
    yy[i]=1
```

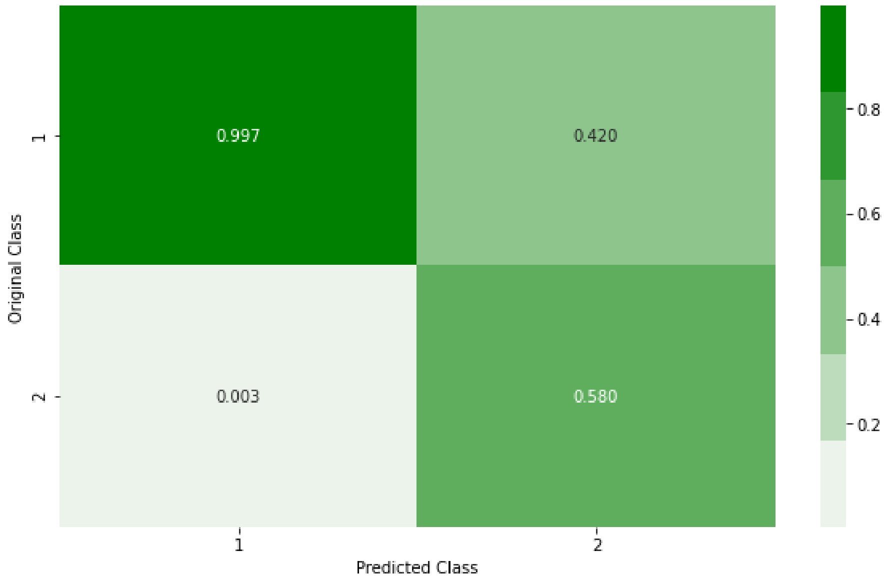
In [156]:

```
'''plotting results'''

import seaborn as sns
import matplotlib.pyplot as plt

plot_confusion_matrix(y_test,yy)
```

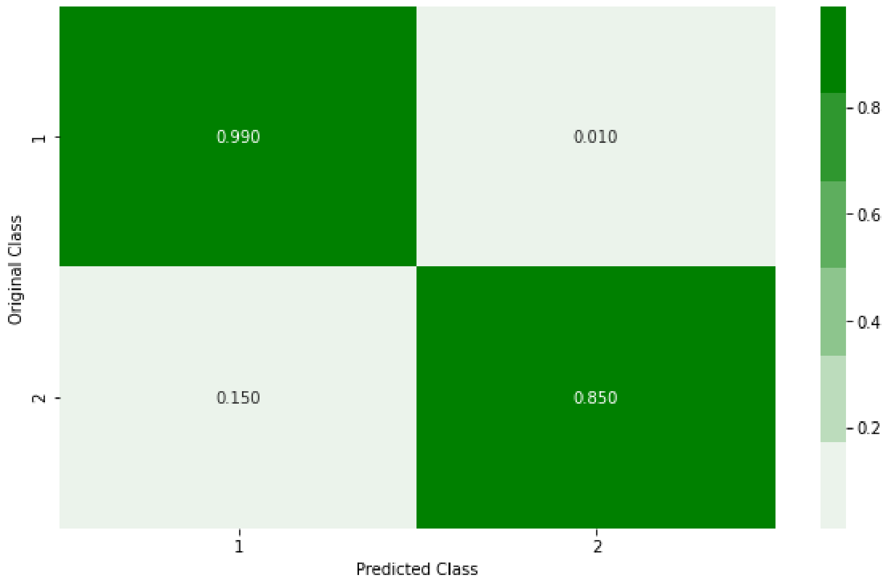
```
Number of misclassified points 153 out of 16001 points
Precision: 0.58
Recall: 0.85
-------------------------------------------------- Confusion matrix -------------------------------
----------------
```




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




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




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

Model summary

Random Forest:

1. The model is hypertuned for n_estimators and max_depth
2. Macro F1 score = 0.89
3. Precision = 0.897
4. Recall = 0.695
5. 79 points are misclassified out of 16001 points in validation set

XGBoost:

1. The model is hypertuned.
2. Macro F1 score = 0.917
3. Precision = 0.828
4. Recall = 0.845
5. 66 points are misclassified out of 16001 points in validation set.

AdaBoost:

1. The model is hypertuned
2. Macro F1 score = 0.89
3. Precision = 0.925
4. Recall = 0.68
5. 75 points are misclassified out of 16001 points

Decision Tree:

1. The model is hypertuned
2. Macro F1 score = 0.803
3. Precion = 0.575
4. Recall = 0.655
5. 166 points are misclassified out of 16001 points

Custom ensemble 1:

1. The previously hypertuned models are used as base models and combined together by sampling with replacement of 25% of X1 data which is derived from 50% of X_train. Then another model is trained on the prediction of these models for X2 which is the other 50% of X_train.
2. Macro F1 score = 0.795
3. Precision = 0.47
4. Recall = 0.825
5. 221 points are misclassified out of 16001 points

Custom ensemble 2:

1. The previsouly hyypertuned models are used and combined together. Each model is trained on X1 which is 50% of X_train. Then another model is trained on the prediction of these models for X2 which is the other 50% of X_train.
2. Macro F1 score = 0.826
3. Precision = 0.542
4. Recall = 0.84
5. 174 points are misclassified out of 16001 points.

Custom ensemble 3:

1. The model is hypertuned
2. Macro F1 score = 0.826
3. Precision = 0.432
4. Recall = 0.775
5. 249 points out of 16001 points are misclassified

ANN:

1. The model is run for 30 epochs
2. Macro F1 score = 0.7836
3. Loss = 0.0718
4. Precision = 0.58
5. Recall = 0.85
6. 153 points of 16001 points are misclassified

XGBoost gives better results hence it is chosen as the final model