Compute performance metrics for the given Y and Y_score

In [1]:
```
import numpy as np
import pandas as pd
# other than these two you should not import any other packages
```

A. Compute performance metrics for the given data '5_a.csv'

Note 1: in this data you can see number of positive points >> number of negatives points

Note 2: use pandas or numpy to read the data from **5_a.csv**

Note 3: you need to derive the class labels from given score

$y^{pred} = [0 \text{ if y\_score} < 0.5 \text{ else } 1]$

1. Compute Confusion Matrix
2. Compute F1 Score
3. Compute AUC Score, you need to compute different thresholds and for each threshold compute tpr,fpr and then use numpy.trapz(tpr_array, fpr_array) https://stackoverflow.com/q/53603376/4084039, https://stackoverflow.com/a/39878975/4084039 Note: it should be numpy.trapz(tpr_array, fpr_array) not numpy.trapz(fpr_array, tpr_array)

   Note- Make sure that you arrange your probability scores in descending order while calculating AUC
4. Compute Accuracy Score

In [3]:
```
df_a=pd.read_csv('5_a.csv')
df_a.head()
```

Out[3]:

	y	proba
0	1.0	0.637387
1	1.0	0.635165
2	1.0	0.766586
3	1.0	0.724564
4	1.0	0.889199

In [4]:
```
# write your code here for task A
df_a['y_predict']=df_a['proba'].apply(lambda x: 0 if x<= 0.5 else 1)
```

In [5]:
```
df_a.head()
```

Out[5]:

	y	proba	y_predict
0	1.0	0.637387	1
1	1.0	0.635165	1
2	1.0	0.766586	1
3	1.0	0.724564	1
4	1.0	0.889199	1

Confusion matrix:

In [6]:
```
def confusion_matrix(df):
    ctn = len(df[(df['y']==0) & (df['y_predict']==0)])
    ctp = len(df[(df['y']==1) & (df['y_predict']==1)])
    cfn = len(df[(df['y']==1) & (df['y_predict']==0)])
    cfp = len(df[(df['y']==0) & (df['y_predict']==1)])
    return ctn,ctp,cfn,cfp
```

In [7]:
```
ctn,ctp,cfn,cfp=confusion_matrix(df_a)
```

In [8]:
```
print("false negative :",cfn)
print("false positive :",cfp)
print("true positive :",ctp)
print("true negative :",ctn)
```

```
false negative : 0
false positive : 100
true positive : 10000
true negative : 0
```

In [9]:
```
def f1_score_and_accuracy(df):
    tn,tp,fn,fp = confusion_matrix(df)
    precision = (tp)/(tp+fp)
    recall = (tp)/(tp+fn)
    f1_score=(2*precision*recall)/(precision+recall)
    accuracy=(tp+tn)/(tp+tn+fp+fn)
    return f1_score,accuracy
```

In [10]:
```
def auc_score(data):
    tpr=[]
    fpr=[]
    sort= data.sort_values("proba",ascending=False)
    for i in range(0,len(sort)):
        sort['y_predict']=np.where(sort['proba']>=sort.iloc[i]['proba'],1,0)
        FN,FP,TN,TP=confusion_matrix(sort)
        fpr_rate=FP/(TN+FP)
        tpr_rate=TP/(TP+FN)
        tpr.append(tpr_rate)
        fpr.append(fpr_rate)
    c=np.trapz(tpr, fpr)
    return 1-c
```

In [11]:
```
gh=auc_score(df_a)
print(gh)
```

```
0.48829900000000004
```

Compute the performance metric of given dataset 5_b:

In [12]:
```
df_b=pd.read_csv('5_b.csv')
df_b.head()
```

Out[12]:

	y	proba
0	0.0	0.281035
1	0.0	0.465152
2	0.0	0.352793
3	0.0	0.157818
4	0.0	0.276648

In [13]:
```
df_b['y_predict']=df_b['proba'].apply(lambda x:0 if x<=0.5 else 1)
df_b.head()
```

Out[13]:

	y	proba	y_predict
0	0.0	0.281035	0
1	0.0	0.465152	0
2	0.0	0.352793	0
3	0.0	0.157818	0
4	0.0	0.276648	0

In [14]:
```
ctn,ctp,cfn,cfp=confusion_matrix(df_b)
```

In [15]:
```
print("false negative :",cfn)
print("false positive :",cfp)
print("true positive :",ctp)
print("true negative :",ctn)
```

```
false negative : 45
false positive : 239
true positive : 55
true negative : 9761
```

In [16]:
```
f1_score,accuracy=f1_score_and_accuracy(df_b)
print("f1 score of dataset b :",f1_score)
print("accuracy of dataset b :",accuracy)
```

```
f1 score of dataset b : 0.2791878172588833
accuracy of dataset b : 0.9718011881188119
```

In [17]:
```
AUC=auc_score(df_b)
print(AUC)
```

```
0.937757
```

C. Compute the best threshold (similarly to ROC curve computation) of probability which gives lowest values of metric A for the given data

you will be predicting label of a data points like this: $y^{pred} = [0 \text{ if y\_score} < \text{threshold else } 1]$

$A = 500 \times \text{number of false negative} + 100 \times \text{numebr of false positive}$

Note 1: in this data you can see number of negative points > number of positive points

Note 2: use pandas or numpy to read the data from **5_c.csv**

In [18]:
```
df_c=pd.read_csv('5_c.csv')
df_c['y_predict']=df_c['prob'].apply(lambda x:0 if x<=0.5 else 1)
df_c.head()
```

Out[18]:

	y	prob	y_predict
0	0	0.458521	0
1	0	0.505037	1
2	0	0.418652	0
3	0	0.412057	0
4	0	0.375579	0

In [19]:
```
from tqdm import tqdm      # purpose of import is to just see progress
q={}
unique = list(df_c.prob)
unique.sort()

for i in tqdm(unique):
    df_c.loc[df_c['prob'] < i, 'y_pred'] = 0
    df_c.loc[df_c['prob'] > i, 'y_pred'] = 1

    FN = int(df_c[(df_c.y == 1) & (df_c.y_pred == 0)].count()[0])
    FP = int(df_c[(df_c.y == 0) & (df_c.y_pred == 1)].count()[0])

    A = (500 * FN) + (100 * FP)
    q[i] = A

minval = min(q.values())
print(minval)

for threshold in q:
    if q[threshold] == minval:
        print("The threshold is",threshold)
```

```
100%|██████████| 2852/2852 [01:06<00:00, 42.67it/s]
```

```
141000
The threshold is 0.2300390278970873
```

D. Compute performance metrics(for regression) for the given data 5_d.csv

Note 2: use pandas or numpy to read the data from **5_d.csv**

Note 1: 5_d.csv will having two columns Y and predicted_Y both are real valued features

1. Compute Mean Square Error
2. Compute MAPE: https://www.youtube.com/watch?v=ly6ztgIkUxk
3. Compute R^2 error: https://en.wikipedia.org/wiki/Coefficient_of_determination#Definitions

In [20]:
```
df_d=pd.read_csv('5_d.csv')
df_d.head()
```

Out[20]:

	y	pred
0	101.0	100.0
1	120.0	100.0
2	131.0	113.0
3	164.0	125.0
4	154.0	152.0

In [21]:
```
def regression_metric(dataset):
    n=len(dataset)
    dataset["abs_diff"]=dataset.apply(lambda x: abs(x['y']-x['pred']),axis=1)
    dataset['square']=dataset['abs_diff'].apply(lambda x: x**2)
    summ=dataset['square'].sum()
    MSE=summ/n
    MAPE=(dataset['abs_diff'].sum())/(dataset['y'].sum())
    simple_mean=dataset['y'].sum()/n
    ssres=dataset['square'].sum()
    dataset['total']=dataset.apply(lambda x: (x['y'] - simple_mean),axis=1)
    dataset['total']=dataset['total'].apply(lambda x: x**2)
    sstotal=dataset['total'].sum()
    Rsqrd=1-(ssres/sstotal)
    return MSE,MAPE,Rsqrd
```

In [22]:
```
MSE,MAPE,Rsqrd = regression_metric(df_d)
print("Mean squared error:",MSE)
print("Mean absolute percentage error:",MAPE)
print("R Squared error:",Rsqrd)
```

```
Mean squared error: 177.1656097454707
Mean absolute percentage error: 0.12912029940409687
R Squared error: 0.9563582786990937
```