Compute performance metrics for the given Y and Y_score without sklearn

In [26]:
```
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/39678975/4084039 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 [27]:
```
df_a=pd.read_csv('5_a.csv')
df_a.head(10)
copy_df_a=df_a.copy()
```

In [28]:
```
df_a.proba[df_a.proba>=0.5]=1 # replacing proba value greater than 0.5 as 1
df_a.proba[df_a.proba<0.5]=0 # replacing proba value less than 0.5 as 0
```

In [29]:
```
print(df_a['proba'].value_counts())# After replacing prob i.e predicted
print(df_a['y'].value_counts()) # Before predicting y values
```

```
1.0    10100
Name: proba, dtype: int64
1.0    10000
0.0      100
Name: y, dtype: int64
```

In [30]:
```
df_a.head(5)
```

Out[30]:

	y	proba
0	1.0	1.0
1	1.0	1.0
2	1.0	1.0
3	1.0	1.0
4	1.0	1.0

In [31]:
```
df_a=df_a.rename(columns={'proba':'Y_pred'},inplace=False)# renaming proba as y_pred
```

In [32]:
```
df_a.head(2)
```

Out[32]:

	y	Y_pred
0	1.0	1.0
1	1.0	1.0

In [33]:
```
# write your code here for task A
```

In [34]:
```
print(df_a['Y_pred'].value_counts())
print(df_a['y'].value_counts())
```

```
1.0    10100
Name: Y_pred, dtype: int64
1.0    10000
0.0      100
Name: y, dtype: int64
```

In [35]:
```
def Get_confusion_matrix(df_a): # Return a confusion matrix for a given dataset with y and y_pred columns
    True_possitive=0
    True_negative=0
    False_negative=0
    False_possitive=0
    for i in range(len(df_a)):
        y=df_a.loc[i,'y']
        Y_pred=df_a.loc[i,'Y_pred']
        if y==1 and Y_pred==1:
            True_possitive+=1
        elif y==1 and Y_pred==0:
            False_negative+=1
        elif y==0 and Y_pred==1:
            False_possitive+=1
        elif y==0 and Y_pred==0:
            True_negative+=1
    #print(True_possitive,True_negative,False_possitive,False_negative)
    confusion_matrix=[[True_negative,False_negative],
                     [False_possitive,True_possitive]]
    return confusion_matrix
```

In [36]:
```
confusion_matrix=Get_confusion_matrix(df_a)
print(confusion_matrix)
```

```
[[0, 0], [100, 10000]]
```

In [37]:
```
def f1_score(confusion_matrix):#Return F! function for a given confusion matrix
    precision=(confusion_matrix[1][1])/(confusion_matrix[1][0]+confusion_matrix[1][1])
    recall=(confusion_matrix[1][1])/(confusion_matrix[0][1]+confusion_matrix[1][1])
    f1_score=(2*precision*recall/(precision+recall))
    return f1_score
    #print(precision,recall)
```

In [38]:
```
f1_score(confusion_matrix)#F1 score for a given confusion matrix
```

Out[38]:
```
0.9950248756218906
```

In [39]:
```
def accuracy(df_a):#Take a dataset and return accuracy of that dataset , dataset should have colums with name y and y_pred
    total=0
    count=0
    for i in range(len(df_a)):
        y=df_a.loc[i,'y']
        Y_pred=df_a.loc[i,'Y_pred']
        total=total+1
        if y==Y_pred:
            count=count+1
    return count/total
```

In [40]:
```
print(accuracy(df_a))
```

```
0.9900990099009901
```

In [41]:
```
copy_df_a.head(10)
```

Out[41]:

	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
5	1.0	0.601600
6	1.0	0.666323
7	1.0	0.567012
8	1.0	0.650230
9	1.0	0.829346

In [42]:
```
final_df=copy_df_a.sort_values(by=['proba'],ascending=False,ignore_index=True)# dataset is sorted in descending order based on proba column
```

In [43]:
```
global x,y
x=[]
y=[]

def final_cal(temp_df,proa):#This function will return True possitive rate and False possitive rate for a given dataset
    new=[]
    for i in range(len(temp_df)):
        if temp_df.loc[i,'proba']<=proa:
            new.append(0)
        else:
            new.append(1)
    temp_df['new']=new
    temp_df=temp_df.rename(columns={'new':'Y_pred'},inplace=False)
    confusion_matrix=Get_confusion_matrix(temp_df)
    TPR=confusion_matrix[1][1]/(confusion_matrix[0][1]+confusion_matrix[1][1])
    FPR=confusion_matrix[0][1]/(confusion_matrix[0][1]+confusion_matrix[1][1])
    return TPR,FPR
```

In [44]:
```
for i in range(len(final_df)):
    proa=final_df.loc[i,'proba']
    if i%500==0 or i==0:
        print(i)
        print(proa)

    TPR,FPR=final_cal(final_df,proa)
    x.append(FPR)
    y.append(TPR)
```

```
0
0.8999653487823838
500
0.8791240623276378
1000
0.8590080612538507
1500
0.8373168946051712
2000
0.8159932794504626
2500
0.7949220935853838
3000
0.7743252112201882
3500
0.755482568935378
4000
0.7368709011091215
4500
0.7172746189553424
5000
0.6987738984053377
5500
0.6799145909531205
6000
0.661502140072926
6500
0.6415840864108876
7000
0.6225734860427463
7500
0.6032707857199351
8000
0.5819224876370167
8500
0.562993382011199
9000
0.5432465020287633
9500
0.5220242820927757
10000
0.5044086485903945
```

In [45]:
```
trapz=np.trapz(y,x)#Return area under the curve for the given points
print(trapz)
```

```
-0.499900005
```

B. Compute performance metrics for the given data '5_b.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_b.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/39678975/4084039
   Note- Make sure that you arrange your probability scores in descending order while calculating AUC
4. Compute Accuracy Score

In [46]:
```
df_b=pd.read_csv('5_b.csv')
df_b.head()
df_copy_b=df_b.copy()
```

In [47]:
```
# write your code here for task B
df_b.proba[df_b.proba>=0.5]=1
df_b.proba[df_b.proba<0.5]=0
```

In [48]:
```
print(df_b['proba'].value_counts())# Predicted points
print(df_b['y'].value_counts())# Actual given points
```

```
0.0    9806
1.0     294
Name: proba, dtype: int64
0.0    10000
1.0      100
Name: y, dtype: int64
```

In [49]:
```
#Getting confusion matrix from the above function Get_confusion_matrix
df_b=df_b.rename(columns={'proba':'Y_pred'},inplace=False)
confusion_matrix_b=Get_confusion_matrix(df_b)
print(confusion_matrix_b)
```

```
[[9761, 45], [239, 55]]
```

In [50]:
```
#Getting f1 score with above function f1_score
f1_score_b=f1_score(confusion_matrix_b)
print(f1_score_b)
```

```
0.2791878172588833
```

In [51]:
```
#Getting acuuracy with the above function accuracy
accuracy_b=accuracy(df_b)
print(accuracy_b)
```

```
0.9718811881188119
```

In [52]:
```
final_df_b=df_copy_b.drop_duplicates(subset=['proba'],inplace=False,keep='first')#Trying to remove duplicates , but no data found with duplicates
```

In [53]:
```
print(final_df_b['proba'].count())# after removing duplicates
df_b.count()# Before removing duplicates
final_df_b.sort_values(by=['proba'],ascending=False,ignore_index=True,inplace=True)# sort the data in descending order
final_df_b.head(10)
```

```
10100
```

Out[53]:

	y	proba
0	1.0	0.595294
1	1.0	0.594808
2	1.0	0.592198
3	1.0	0.590171
4	1.0	0.588718
5	1.0	0.585175
6	1.0	0.583235
7	1.0	0.582210
8	1.0	0.582020
9	1.0	0.581772

In [54]:
```
# using the function  final_cal to get the list of TPR and FPR values
global x_b,y_b
x_b=[]
y_b=[]

for i in range(len(final_df_b)):
    if i%500==0 or i==0:
        print(i)
        print(proa)
    proa=final_df_b.loc[i,'proba']
    TPR,FPR=final_cal(final_df_b,proa)
    x_b.append(FPR)
    y_b.append(TPR)
```

```
0
0.5000185949718864
500
0.4911047881811058
1000
0.4711579715048221
1500
0.4511207275572831
2000
0.432255453828537
2500
0.4120429070220512
3000
0.3911799922542589
3500
0.3704636605699682
4000
0.3496305891130574
4500
0.3283548098119311
5000
0.3051640086545476
5500
0.2848868658176385
6000
0.2642695877504534
6500
0.2435032622783316
7000
0.2236974633593647
7500
0.2037631365425095
8000
0.1830260951795065
8500
0.1641683911133339
9000
0.1440896229197282
9500
0.1242476577862734
10000
0.104052295804322
```

In [55]:
```
# AUC value which is the area under the AUC curve
trapz=np.trapz(x_b,y_b)
print(trapz)
```

```
0.5
```

In [ ]:

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

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

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 [56]:
```
df_c=pd.read_csv('5_c.csv')
df_c.rename(columns={'prob':'proba'},inplace=True)
df_copy_c=df_c.copy()# making a copy of the dataset to use in feature requirement
df_c.head()
```

Out[56]:

	y	proba
0	0	0.458521
1	0	0.505037
2	0	0.418652
3	0	0.412057
4	0	0.375579

In [57]:
```
 # write your code for task C
df_c.proba[df_c.proba>=0.5]=1
df_c.proba[df_c.proba<0.5]=0
```

```
<ipython-input-57-40262a55a381>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_c.proba[df_c.proba>=0.5]=1
<ipython-input-57-40262a55a381>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_c.proba[df_c.proba<0.5]=0
```

In [58]:
```
print(df_c['proba'].value_counts())# Predicted points
print(df_c['y'].value_counts())# Actual given points
```

```
0.0    2099
1.0     753
Name: proba, dtype: int64
0    1805
1    1047
Name: y, dtype: int64
```

In [59]:
```
#Getting confusion matrix with the function Get_confusion_matrix
df_c=df_c.rename(columns={'proba':'Y_pred'},inplace=False)
confusion_matrix_c=Get_confusion_matrix(df_c)
print(confusion_matrix_c)
```

```
[[1637, 462], [168, 585]]
```

In [60]:
```
def find_A(temp_df,proa):# Return A value for a given dataset and thershold value
    new=[]
    count=0
    for i in range(len(temp_df)):
        if temp_df.loc[i,'proba']<=proa:
            new.append(0)
        else:
            new.append(1)
    temp_df['new']=new
    temp_df=temp_df.rename(columns={'new':'Y_pred'},inplace=False)
    confusion_matrix=Get_confusion_matrix(temp_df)
    False_positive=confusion_matrix[1][0]
    False_Negative=confusion_matrix[0][1]
    A=500*False_Negative + 100*False_positive
    return A
```

In [61]:
```
c_calulated_values={}#USed to store all the A values in the form of dict with thershold, A as keyvalue pair
final_df_c=df_copy_c.drop_duplicates(subset=['proba'],keep='first',ignore_index=True)
final_df_c.sort_values(by=['proba'],ascending=False,ignore_index=True,inplace=True)
for i in range(len(final_df_c)):
    proa=final_df_c.loc[i,'proba']
    if i%500==0 or i==0:
        print(i)
        print(proa)
    A=find_A(final_df_c,proa)
    c_calulated_values[proa]=A
```

```
<ipython-input-61-057247ecd039>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  final_df_c.sort_values(by=['proba'],ascending=False,ignore_index=True,inplace=True)
<ipython-input-60-fc22342ff1c1>:9: 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
  temp_df['new']=new
0
0.9577467989277196
500
0.5684922297258634
1000
0.4258294327033533
1500
0.3135492302275993
2000
0.217318328449855
2500
0.1269036644213909
```

In [62]:
```
def get_min_value(dct):# Get a value of thershold such that A is minimum
    min_term=max(dct.values())
    req=-1
    for i,j in dct.items():
        if j<min_term:
            min_term=j
            req=i
    return req
```

In [63]:
```
print(get_min_value(c_calulated_values))
```

```
0.2501259844850849
```

In [ ]:

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 [64]:
```
df_d=pd.read_csv('5_d.csv')
df_d.head()
```

Out[64]:

	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 [65]:
```
 # write your code for task 5d
MSE=0
for i in range(len(df_d)):
    y=df_d.loc[i,'y']
    y_pred=df_d.loc[i,'pred']
    sub=(y-y_pred)*(y-y_pred)
    MSE=MSE+sub
MSE=MSE/len(df_d)
print(MSE)# mean square value
```

```
177.16569974554707
```

In [66]:
```
#mean Absolute percentage errror
e=0
a=0
for i in range(len(df_d)):
    y=df_d.loc[i,'y']
    y_pred=df_d.loc[i,'pred']
    temp=abs(y_pred-y)
    e=e+temp
    a=a+y
MAPE=e/a
print(MAPE)
print(MAPE*100)# MAPE value less than 30 is always acceptable
```

```
0.1291202994009687
12.91202994009687
```

In [67]:
```
#R squared values
RR=0
avg_y=sum(df_d['y'])/len(df_d['y'])
sum_of_squares=0
sum_of_resude=0
for i in range(len(df_d)):
    y=df_d.loc[i,'y']
    y_pred=df_d.loc[i,'pred']
    sum_of_squares=sum_of_squares+((y-avg_y)**2)
    sum_of_resude=sum_of_resude + (y-y_pred)**2

RR=(1 - (sum_of_resude/sum_of_squares))**2
print(RR)# RR value near to 1 is best model we have got around 0.91
```

```
0.9146211572362986
```

In [ ]: