Compute performance metrics for the given Y and Y_score without sklearn

In [70]:

```
import numpy as np
import pandas as pd
```

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 if y_score < 0.5 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/q/53603376/4084039), https://stackoverflow.com/a/39678975/4084039 (https://stackoverflow.com/a/39678975/4084039) Note: it should be numpy.trapz(tpr_array, fpr_array) not numpy.trapz(fpr_array, tpr_array)
4. Compute Accuracy Score

In [71]:

```
#Mountng the g-drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remo
unt, call drive.mount("/content/drive", force_remount=True).
```

In [74]:

```
# Read the data from ccsv into pandas dataframe
data_a = pd.read_csv('/content/drive/MyDrive/5_Performance_metrics/5_a.csv')
data_a.head(10)
```

Out[74]:

	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 [76]:

```
# creating y_prediction column which contains 1(if proba >= 0.5) else 0
y_pred = np.where(data_a['proba'] >= 0.5, 1, 0)
y_pred
```

Out[76]:

```
array([1, 1, 1, ..., 1, 1, 1])
```

In [77]:

```
#Compute confusion Matrix
def confusion(y, y_pred):
    fp = np.sum((y_pred == 1) & (y == 0))
    tp = np.sum((y_pred == 1) & (y == 1))
    fn = np.sum((y_pred == 0) & (y == 1))
    tn = np.sum((y_pred == 0) & (y == 0))
    confusion_matrix = [[tn, fn], [fp, tp]]
    return confusion_matrix, tn, fp, fn, tp

cm, tn, fp, fn, tp = confusion(data_a['y'], y_pred)
print (cm)
```

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

In [78]:

```
# calculate F1 score by calculating precision and recall
precision = tp / (tp + fp)
recall = tp / (fn + tp)
f1 = 2 * ((precision * recall)/(precision + recall))
f1
```

Out[78]:

```
0.9950248756218906
```

In [79]:

```
# Accuracy
accuracy_score = (tn + tp) / (tn + tp + fn + fp)
accuracy_score
```

Out[79]:

```
0.9900990099009901
```

In [80]:

```
# fpr and tpr calculation
def fpr_tpr_calculate(y, proba, thresholds):

  fpr = []
  tpr = []

  for threshold in thresholds:

      y_pred = np.where(proba >= threshold, 1, 0)
      fp = np.sum((y_pred == 1) & (y == 0))
      tp = np.sum((y_pred == 1) & (y == 1))
      fn = np.sum((y_pred == 0) & (y == 1))
      tn = np.sum((y_pred == 0) & (y == 0))

      fpr.append(fp / (fp + tn))
      tpr.append(tp / (tp + fn))

  return [fpr, tpr]

thresholds = data_a['proba']
thresholds = thresholds.sort_values(ascending=False)
result = fpr_tpr_calculate(data_a['y'], data_a['proba'], thresholds)
```

In [81]:

```
np.trapz(result[1], result[0])
```

Out[81]:

```
0.48829900000000004
```

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 if y_score < 0.5 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/q/53603376/4084039), https://stackoverflow.com/a/39678975/4084039 (https://stackoverflow.com/a/39678975/4084039)
4. Compute Accuracy Score

In [82]:

```
data_b = pd.read_csv('/content/drive/MyDrive/5_Performance_metrics/5_b.csv')
data_b.tail(10)
```

Out[82]:

	y	proba
10090	0.0	0.249445
10091	0.0	0.489342
10092	0.0	0.346930
10093	0.0	0.224109
10094	0.0	0.425500
10095	0.0	0.474401
10096	0.0	0.128403
10097	0.0	0.499331
10098	0.0	0.157616
10099	0.0	0.296618

In [83]:

```
y_pred_b = np.where(data_b['proba'] >= 0.5, 1, 0)
y_pred_b
```

Out[83]:

```
array([0, 0, 0, ..., 0, 0, 0])
```

In [84]:

```
#Compute confusion Matrix
def confusion(y, y_pred):
    fp = np.sum((y_pred == 1) & (y == 0))
    tp = np.sum((y_pred == 1) & (y == 1))
    fn = np.sum((y_pred == 0) & (y == 1))
    tn = np.sum((y_pred == 0) & (y == 0))
    confusion_matrix = [[tn, fn], [fp, tp]]
    return confusion_matrix, tn, fp, fn, tp

cm, tn, fp, fn, tp = confusion(data_b['y'], y_pred_b)
print (cm)
```

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

In [85]:

```
# calculate F1 score by calculating precision and recall
precision = tp / (tp + fp)
recall = tp / (fn + tp)
f1 = 2 * ((precision * recall)/(precision + recall))
f1
```

Out[85]:

```
0.2791878172588833
```

In [86]:

```
# Accuracy
accuracy_score = (tn + tp) / (tn + tp + fn + fp)
accuracy_score
```

Out[86]:

```
0.9718811881188119
```

In [87]:

```
# fpr tpr calculation
def fpr_tpr_calculate(y, proba, thresholds):

  fpr = []
  tpr = []

  for threshold in thresholds:

      y_pred = np.where(proba >= threshold, 1, 0)

      fp = np.sum((y_pred == 1) & (y == 0))
      tp = np.sum((y_pred == 1) & (y == 1))
      fn = np.sum((y_pred == 0) & (y == 1))
      tn = np.sum((y_pred == 0) & (y == 0))

      fpr.append(fp / (fp + tn))
      tpr.append(tp / (tp + fn))

  return [fpr, tpr]

thresholds = data_b['proba']
thresholds = thresholds.sort_values(ascending=False)
result = fpr_tpr_calculate(data_b['y'], data_b['proba'], thresholds)
```

In [88]:

```
np.trapz(result[1], result[0])
```

Out[88]:

```
0.9377570000000001
```

C. Compute the best threshold (similarly to ROC curve computation) of probability which gives lowest values of metric **A** for the given data **5_c.csv**

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 [89]:

```
data_c = pd.read_csv('/content/drive/MyDrive/5_Performance_metrics/5_c.csv')
data_c.tail(10)
```

Out[89]:

	y	prob
2842	1	0.234774
2843	1	0.645604
2844	1	0.804834
2845	1	0.632728
2846	1	0.575723
2847	1	0.491663
2848	1	0.292109
2849	1	0.659161
2850	1	0.456265
2851	1	0.659161

In [90]:

```
# Min Threshold calculation
def fn_fp_calculate(y, prob, thresholds):

  min_A = -1
  min_threshold = 0
  for threshold in thresholds:
      y_pred = np.where(prob >= threshold, 1, 0)
      fp = np.sum((y_pred == 1) & (y == 0))
      fn = np.sum((y_pred == 0) & (y == 1))
      A = (500 * fn) + (100 * fp)

      if min_A == -1:
        min_A = A
      elif min_A > A:
        min_A = A
        min_threshold = threshold

  return [min_A, min_threshold]

thresholds = data_c['prob']
thresholds = thresholds.sort_values(ascending=False)
min_A, thresh = fn_fp_calculate(data_c['y'], data_c['prob'], thresholds)
print ("Minimun A:", min_A, "at threshold:", thresh)
```

```
Minimun A: 141000 at threshold: 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 [91]:

```
data_d = pd.read_csv('/content/drive/MyDrive/5_Performance_metrics/5_d.csv')
data_d.tail(10)
```

Out[91]:

	y	pred
157190	91.0	111.0
157191	80.0	96.0
157192	76.0	85.0
157193	95.0	78.0
157194	82.0	89.0
157195	87.0	83.0
157196	97.0	86.0
157197	106.0	93.0
157198	105.0	101.0
157199	81.0	104.0

In [92]:

```
#Mean Square Error
np.square(np.subtract(data_d['y'], data_d['pred'])).mean()
```

Out[92]:

```
177.16569974554707
```

In [93]:

```
# MAPE
np.sum(np.abs(data_d['y'] - data_d['pred'])) / np.sum(data_d['y'])
```

Out[93]:

```
0.1291202994009687
```

In [94]:

```
# R^2 error
corr_matrix = np.corrcoef(data_d['y'], data_d['pred'])
corr = corr_matrix[0,1]
r_sq = corr**2
r_sq
```

Out[94]:

```
0.9563600409880488
```