In [160]:
```
import numpy as np
```

In [161]:
```
import pandas as pd
```

In [162]:
```
df=pd.read_csv('5_a.csv')
```

In [163]:
```
df.shape
```

Out[163]:
```
(10100, 2)
```

In [164]:
```
df['y'].value_counts()
```

Out[164]:
```
1.0    10000
0.0      100
Name: y, dtype: int64
```

In [165]:
```
df['proba'][0]
```

Out[165]:
```
0.6373866237658206
```

Function to modify the probabilities

In [166]:
```
def Change_prob(df):
    k=0
    for i in df['proba']:
        if (i > 0.5):
            df['proba'][k]=1.0
            k=k+1
        else :
            df['proba'][k]=0.0
            k=k+1
    df.rename(columns={'proba':'y_pred'},inplace=True)
    return df
```

In [167]:
```
Change_prob(df)
```

Out[167]:

	y	y_pred
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
5	1.0	1.0
6	1.0	1.0
7	1.0	1.0
8	1.0	1.0
9	1.0	1.0
10	1.0	1.0
11	1.0	1.0
12	1.0	1.0
13	1.0	1.0
14	1.0	1.0
15	1.0	1.0
16	1.0	1.0
17	1.0	1.0
18	1.0	1.0
19	1.0	1.0
20	1.0	1.0
21	1.0	1.0
22	1.0	1.0
23	1.0	1.0
24	1.0	1.0
25	1.0	1.0
26	1.0	1.0
27	1.0	1.0
28	1.0	1.0
29	1.0	1.0
...	...	...
10070	1.0	1.0
10071	1.0	1.0
10072	1.0	1.0
10073	1.0	1.0
10074	1.0	1.0
10075	1.0	1.0
10076	1.0	1.0
10077	1.0	1.0
10078	1.0	1.0
10079	1.0	1.0
10080	1.0	1.0
10081	1.0	1.0
10082	1.0	1.0
10083	1.0	1.0
10084	1.0	1.0
10085	1.0	1.0
10086	1.0	1.0
10087	1.0	1.0
10088	1.0	1.0
10089	1.0	1.0
10090	1.0	1.0
10091	1.0	1.0
10092	1.0	1.0
10093	1.0	1.0
10094	1.0	1.0
10095	1.0	1.0
10096	1.0	1.0
10097	1.0	1.0
10098	1.0	1.0
10099	1.0	1.0

10100 rows × 2 columns

In [168]:
```
df['y_pred'].value_counts()
```

Out[168]:
```
1.0    10100
Name: y_pred, dtype: int64
```

Task 1,2,4 in a single function Because the same can be repeated in the 2nd question

In [169]:
```
def confusion_matrix_Prec_recall_F1_acc(df):

    tp,tn,fp,fn=0,0,0,0
    tp = ((df['y']==1.0) & (df['y_pred']==1.0)).sum()
    fp = ((df['y']==0.0) & (df['y_pred']==1.0)).sum()
    fn = ((df['y']==1.0) & (df['y_pred']==0.0)).sum()
    tn = ((df['y']==0.0) & (df['y_pred']==0.0)).sum()
    print('confusion matrix :')
    print(np.array([tp,fp,fn,tn]))
    print('*'*25)

    prec=tp/(tp+fp)
    print('prec : {}'.format(prec))
    print('*'*25)

    recall=tp/(tp+fn)
    print('recall : {}'.format(recall))

    F1_score=2*prec*recall/(prec+recall)
    print('*'*25)
    print('F1_score  : {}'.format(F1_score))
    print('*'*25)

    accuracy=(tp+tn)/(tp+tn+fp+fn)
    print('accuracy {} '.format(accuracy))
    return tp,fp,fn,tn,prec,recall,F1_score,accuracy
```

In [170]:
```
confusion_matrix_Prec_recall_F1_acc(df)
```

```
confusion matrix :
[10000   100     0     0]
*************************
prec : 0.9900990099009901
*************************
recall : 1.0
*************************
F1_score  : 0.9950248756218906
*************************
accuracy 0.9900990099009901
```

Out[170]:
```
(10000,
 100,
 0,
 0,
 0.9900990099009901,
 1.0,
 0.9950248756218906,
 0.9900990099009901)
```

AUC

In [171]:
```
data=pd.read_csv('5_a.csv')
```

In [172]:
```
data.head()
```

Out[172]:

	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 [173]:
```
unique_prob = list(data.proba.unique())
```

In [174]:
```
len(unique_prob) # All the values are unique we need to have all treshold values
```

Out[174]:
```
10100
```

In [175]:
```
unique_prob.sort()
```

In [ ]:

In [176]:
```
# Observation: All the probabilities are greater than 0.5
```

In [ ]:

In [ ]:

In [177]:
```
from tqdm import tqdm
```

In [178]:
```
#Referred this video and followed mapped the example to code this - https://www.youtube.com/watch?v=A_ZKMsZ3f3o
def AUC(data):
    unique_prob = list(data.proba.unique())
    unique_prob.sort(reverse=True)
    tpr=[]
    fpr=[]

    for i in unique_prob: # The time complexity is O(n^2)
        y_new=[] # Assigning this here because for every new value of i we need to loop through all the values

        for j in data['proba']:
            if (j<i):
                y_new.append(0)
            else :
                y_new.append(1)


        data['y_new_pred']=y_new # marking this as a feature since we will be using y_new again

        tp = (((data['y'])==1) & ((data['y_new_pred'])==1)).sum()
        fp = (((data['y'])==0) & ((data['y_new_pred'])==1)).sum()
        fn = (((data['y'])==1) & ((data['y_new_pred'])==0)).sum()
        tn = (((data['y'])==0) & ((data['y_new_pred'])==0)).sum()

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


    tpr_plot = sorted(tpr)
    fpr_plot = sorted(fpr)
    AUC = np.trapz(tpr_plot,fpr_plot)

    print('AUC Score:{}'.format(AUC))
    print('tp :{}'.format(tp))
    print('fp :{}'.format(fp))
    print('tn :{}'.format(tn))
    print('fn :{}'.format(fn))
```

In [179]:
```
data=pd.read_csv('5_a.csv')
```

In [180]:
```
AUC(data)
```

```
AUC Score:0.48829900000000004
tp :10000
fp :100
tn :0
fn :0
```

B part

In [181]:
```
df1=pd.read_csv('5_b.csv')
```

In [182]:
```
Change_prob(df1)
```

Out[182]:

	y	y_pred
0	0.0	0.0
1	0.0	0.0
2	0.0	0.0
3	0.0	0.0
4	0.0	0.0
5	0.0	0.0
6	0.0	0.0
7	0.0	0.0
8	0.0	0.0
9	0.0	0.0
10	0.0	0.0
11	0.0	0.0
12	0.0	0.0
13	0.0	0.0
14	0.0	0.0
15	0.0	0.0
16	0.0	0.0
17	0.0	0.0
18	0.0	0.0
19	0.0	0.0
20	0.0	0.0
21	0.0	0.0
22	0.0	0.0
23	0.0	0.0
24	0.0	0.0
25	0.0	0.0
26	0.0	0.0
27	0.0	0.0
28	0.0	0.0
29	0.0	0.0
...	...	...
10070	0.0	0.0
10071	0.0	0.0
10072	0.0	0.0
10073	0.0	0.0
10074	0.0	0.0
10075	0.0	0.0
10076	0.0	0.0
10077	0.0	1.0
10078	0.0	0.0
10079	0.0	0.0
10080	0.0	0.0
10081	0.0	0.0
10082	0.0	0.0
10083	0.0	0.0
10084	0.0	0.0
10085	0.0	0.0
10086	0.0	0.0
10087	1.0	0.0
10088	0.0	0.0
10089	0.0	0.0
10090	0.0	0.0
10091	0.0	0.0
10092	0.0	0.0
10093	0.0	0.0
10094	0.0	0.0
10095	0.0	0.0
10096	0.0	0.0
10097	0.0	0.0
10098	0.0	0.0
10099	0.0	0.0

10100 rows × 2 columns

In [183]:
```
confusion_matrix_Prec_recall_F1_acc(df1)
```

```
confusion matrix :
[  55  239   45 9761]
*************************
prec : 0.1870748299319728
*************************
recall : 0.55
*************************
F1_score  : 0.2791878172588833
*************************
accuracy 0.9718811881188119
```

Out[183]:
```
(55,
 239,
 45,
 9761,
 0.1870748299319728,
 0.55,
 0.2791878172588833,
 0.9718811881188119)
```

In [184]:
```
data1=pd.read_csv('5_b.csv')
```

In [185]:
```
AUC(data1)
```

```
AUC Score:0.9377570000000001
tp :100
fp :10000
tn :0
fn :0
```

C part

In [186]:
```
data2=pd.read_csv('5_c.csv')
```

In [187]:
```
data2.head()
```

Out[187]:

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

In [188]:
```
data2=pd.read_csv('5_c.csv')
unique_prob = list(data2.prob.unique())
unique_prob.sort(reverse=True)

res=[]
for i in unique_prob: # The time complexity is O(n^2)
    y_new=[] # Assigning this here because for every new value of i we need to loop through all the values

    for j in data2['prob']:
        if (j<=i):
            y_new.append(0)
        else :
            y_new.append(1)


    data2['y_new_pred']=y_new # marking this as a feature since we will be using y_new again


    fp = (((data2['y'])==0) & ((data2['y_new_pred'])==1)).sum()
    fn = (((data2['y'])==1) & ((data2['y_new_pred'])==0)).sum()
    res.append((500*fn) +(100*fp)) # just a modification to AUC function.



uniq=unique_prob[res.index(min(res))]

A=min(res)
print('threshold {}'.format(uniq))
print('min value  {}'.format(A))
```

```
threshold 0.22987164436159915
min value  141000
```

D Task

In [189]:
```
data3=pd.read_csv('5_d.csv')
```

In [190]:
```
data3.head()
```

Out[190]:

	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 [191]:
```
def MeanSquareErorr(data3):
    length=len(data3)
    mse=0
    for i in range(length):
        mse = (mse + (data3['y'][i] - data3['pred'][i])**2) # In every Iteration it loops through each value of y and pred
        mse=(1/length) * ( mse )
    return mse
```

In [192]:
```
def mape(data3):
    length=len(data3)
    numerator=0
    denominator=0
    for i in range(len(data3)):
        numerator+= abs(data3['y'][i] - data3['pred'][i])
        denominator+= (data3['y'][i])
    mape = (numerator / denominator) * 100
    return mape
```

In [193]:
```
def rsquare(data3):
    length=len(data3)
    ss_res=0
    ss_total=0
    mean=0
    mean =np.mean(data3['y'])
    for i in range(length):
        ss_total += (data3['y'][i] - mean)**2
        ss_res += (data3['y'][i] - data3['pred'][i])**2
    R_square = 1 -(ss_res/ss_total)

    return R_square
```

In [194]:
```
MeanSquareErorr(data3)
```

Out[194]:
```
0.0033651405966158357
```

In [195]:
```
mape(data3)
```

Out[195]:
```
12.91202994009687
```

In [197]:
```
rsquare(data3)
```

Out[197]:
```
0.9563582786990964
```

In [ ]:

In [ ]:

In [ ]: