In [1]:

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

5_a.csv

In [2]:

```
###load dataset###
Adata=pd.read_csv('5_a.csv')
y_actual=Adata['y']
```

In [3]:

```
#### with the help of this function we will calculate true_positive,true_negative ,false_negative,false_positive ####

def calculate_tp_tn_fp_fn (Adata):
    y_predict=[]
    y_actual=Adata['y']
    y_predict = [0 if i<0.5 else 1 for i in Adata['proba']]
    true_positive=0 #initialization values
    true_negative=0
    false_negative=0
    false_positive=0

    for i in range(len(y_actual)):
        if y_actual[i]==y_predict[i]==0:
            true_negative +=1
        elif y_actual[i]==0 and y_predict[i]==1:
            false_negative +=1
        elif y_actual[i]==1 and y_predict[i]==0:
            false_positive +=1
        elif y_actual[i]==y_predict[i]==1:
            true_positive +=1
    return true_positive,true_negative ,false_negative,false_positive
```

1. calculation of confusion matrix

In [34]:

```
true_positive,true_negative ,false_negative,false_positive = calculate_tp_tn_fp_fn (Adata)
A=np.array([[true_negative,false_negative],[false_positive,true_positive]])
print('confusion matrix:\n')
print(A)
```

```
confusion matrix:

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

2. F1 score

In [35]:

```
### function for Calculate F1 Score###

def calculate_f1_score(true_positive,true_negative ,false_negative,false_positive):
    recall=0
    precision=0
    #calculate recall
    recall=(true_positive)/(false_negative+true_positive)
    #calculate precision
    precision=(true_positive)/(false_positive+true_positive)
    F1_score=2*((precision*recall)/(recall+precision))
    return F1_score
```

In [36]:

```
##### print the value of F1_score ######

true_positive,true_negative ,false_negative,false_positive = calculate_tp_tn_fp_fn (Adata)
F1_score=calculate_f1_score(true_positive,true_negative ,false_negative,false_positive)
print("F1_score: ",F1_score)
```

```
F1_score:  0.9950248756218906
```

3. calculation of auc score

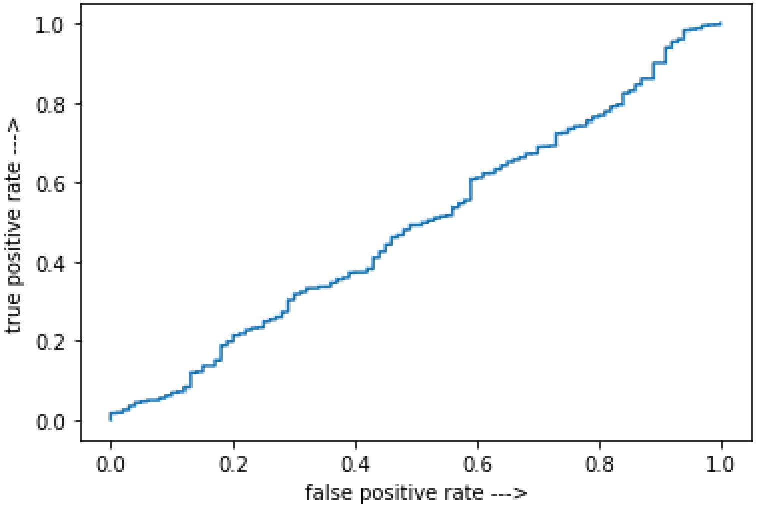
In [27]:

```
def calculate_AUC_score(Adata):
    y_actual = Adata['y']


    y_predict = np.array(Adata['proba']) ## creating np array for fast execution

    #ppick unique value for further calculation
    y_predi = np.unique(y_predict)

    # sort the values
    y_predi.sort()
    total_tpr = []    ## create the list
    total_fpr = []

    # convert float into integer for fast calculation
    y_actual = list(map(int ,y_actual))

    # for loop for finding thresold
    for threshold in y_predi:
        true_positive=0            #initialization values
        true_negative=0
        false_negative=0
        false_positive=0
        y_predicted = [0 if i<threshold else 1 for i in y_predict]

        # for loop for finding the value of tp,tn,fl,fn
        for j in range(len(y_actual)):
            if y_actual[j]==0   and  y_predicted[j] ==0:
                true_negative +=1
            elif y_actual[j]==0 and  y_predicted[j] ==1:
                false_positive +=1
            elif y_actual[j]==1 and  y_predicted[j] ==0:
                false_negative +=1
            elif y_actual[j]==1 and  y_predicted[j] ==1:
                true_positive +=1

          # tpr and fpr stand for true positive rate and false positive rate
          #calculation of tpr and fpr
        tpr = true_positive/(true_positive+false_negative)
        fpr = false_positive/(false_positive+true_negative)
        ## here we find the different tpr and fpr in list form
        total_tpr.append(tpr)
        total_fpr.append(fpr)

       ## we know that execution of  numpy aaray  is very fast that's why we converting list into aaray
    total_tpr_array_form = np.array(total_tpr)
    total_fpr_array_form = np.array(total_fpr)
    return total_tpr_array_form, total_fpr_array_form
```

In [28]:
```
total_tpr_array_form, total_fpr_array_form=calculate_AUC_score(Adata)
auc_value=np.trapz(total_tpr_array_form, total_fpr_array_form)
auc_value=max(auc_value,-(auc_value))
print(auc_value)
```

```
0.48829900000000004
```

In [25]:
```
import matplotlib.pyplot as plt

###plotting of auc value###
total_tpr_array_form, total_fpr_array_form=calculate_AUC_score(Adata)
plt.plot(total_fpr_array_form,total_tpr_array_form)
plt.xlabel('false positive rate --->')
plt.ylabel('true positive rate --->')
plt.show()
```



4. compute accuracy.

In [37]:

```
true_positive,true_negative ,false_negative,false_positive=calculate_tp_tn_fp_fn (Adata)
accuracy_score=(true_positive+true_negative)/(true_negative+true_positive+false_positive+false_negative)
accuracy_score
```

Out[37]:

```
0.9900990099009901
```

5_b.csv

In [7]:

```
### loading data ###
Bdata=pd.read_csv('5_b.csv')
y_actual_b=Bdata['y'] # create Y_actual value

# create list for y_predicted
y_predict_b=[]
y_predict_b= [0 if i<0.5 else 1 for i in Bdata['proba']] #append the values of y_prediction
```

1. compute confusion matrix:

In [38]:

```
true_positive,true_negative ,false_negative,false_positive = calculate_tp_tn_fp_fn (Bdata)
B=np.array([[true_negative,false_negative],[false_positive,true_positive]])
print('confusion matrix:\n')
print(B)
```

```
confusion matrix:

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

2. compute F1_score:

In [39]:

```
true_positive,true_negative ,false_negative,false_positive = calculate_tp_tn_fp_fn (Bdata)
F1_score=calculate_f1_score(true_positive,true_negative ,false_negative,false_positive)
print("F1_score: ",F1_score)
```

```
F1_score:  0.2791878172588833
```

3. calculate AUC score

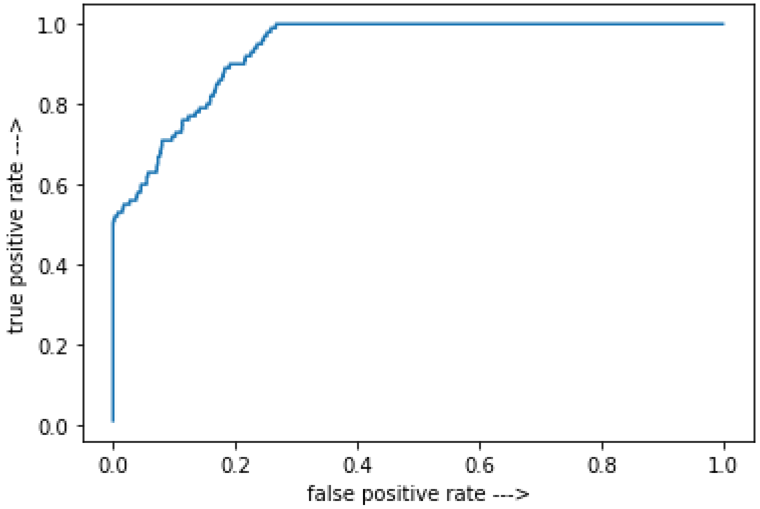
In [23]:

```
total_tpr_array_form, total_fpr_array_form=calculate_AUC_score(Bdata)
auc_value=np.trapz(total_tpr_array_form, total_fpr_array_form)
auc_value=max(auc_value,-(auc_value))
print(auc_value)
```

```
0.937757
```

In [26]:

```
###plotting of auc value###
total_tpr_array_form, total_fpr_array_form=calculate_AUC_score(Bdata)
plt.plot(total_fpr_array_form,total_tpr_array_form)
plt.xlabel('false positive rate --->')
plt.ylabel('true positive rate --->')
plt.show()
```



4.compute accuracy:

In [40]:

```
true_positive,true_negative ,false_negative,false_positive=calculate_tp_tn_fp_fn (Bdata)
accuracy_score=(true_positive+true_negative)/(true_negative+true_positive+false_positive+false_negative)
accuracy_score
```

Out[40]:

```
0.9718811881188119
```

5_c.csv

In [2]:

```
Cdata=pd.read_csv('5_c.csv')
```

In [11]:

```
Cdata.head(5)
```

Out[11]:

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

Compute the best threshold:

In [14]:
```
#### function for calculating best thresold value for minimum A

def best_thresold_calculator(Cdata):
    A=[]
    thresold_list=[] #creating for collecting thresold value

    y_actual=list(Cdata['y'])             # create y_actual
    y_predict=np.array(Cdata['prob'])
    y_predi=np.unique(y_predict)
    y_predi.sort()

    #for loop for appending value in thresold
    for thresold in y_predi:

        false_negative=0 #initializing value o
        false_positive=0

        y_predicted = [0 if j<thresold else 1 for j in y_predict] #get y_predict from thresold
        for k in range(len(y_predi)):

            if y_actual[k]==1 and y_predicted[k]==0: #condition checking
                false_negative +=1
            elif y_actual[k]==0 and y_predicted[k]==1:
                false_positive +=1
          #append the value of metric A
        A.append(500*false_negative + 100*false_positive)

        #append the value of thresolds
        thresold_list.append(thresold)

    best_thresold_value=list(zip(A,thresold_list)) #zipping the list of thresold and metric A
    best_thresold_value=min(best_thresold_value)   # find the minimum value
    return best_thresold_value
```

In [16]:
```
best_thresold_value=best_thresold_calculator(Cdata)
print('best thresold for the lowest value of A is: ',best_thresold_value[1])
```

```
best thresold for the lowest value of A is:  0.250403339798386
```

5 d.csv

In [2]:
```
Ddata=pd.read_csv('5_d.csv')
y_actual_d=Ddata['y']
y_predicted_d=Ddata['pred']
```

In [3]:
```
Ddata.head()
```

Out[3]:

	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 [4]:
```
Ddata.shape
```

Out[4]:
```
(157200, 2)
```

1. mean square error:

In [5]:
```
#### function for finding mean squre error #####

def Mean_Square_Error_calculator(y_predicted_d, y_actual_d):
    error_sum=0 #inilializing the value

    for i in range(len(y_actual_d)):
        error=(y_actual_d[i]-y_predicted_d[i])*(y_actual_d[i]-y_predicted_d[i])
        error_sum += error
    mean_square_error=error_sum/(len(y_actual_d))
    return mean_square_error
```

In [143]:
```
Mean_Square_Error_calculator(y_predicted_d,y_actual_d)
```

Out[143]:
```
177.16569974554707
```

2. mean absolute percentage error:

In [8]:
```
### function for finding mean_absolute_percentage_error ###
import numpy as np
def mean_absolute_percentage_error(y_predicted_d, y_actual_d):
    absolute_error=0
    sum_of_actual_value=0

    for i in range(len(y_actual_d)):

        #Here we applying modified MAPE due to the some actual value are zero.
        absolute_error += (np.abs(y_actual_d[i]-y_predicted_d[i]))
        sum_of_actual_value += y_actual_d[i]

    map_error=((absolute_error)/(sum_of_actual_value))*100
    return map_error
```

In [9]:
```
mean_absolute_percentage_error(y_predicted_d, y_actual_d)
```

Out[9]:
```
12.91202994009687
```

3. computation of R-square error:

description:

R-squared or cofficient of determination=1-(((sum of square)residuals)/((sum of square)total))

In [21]:

```
#### function for calculating r square error ####
def r_square_error_computation(y_predicted_d, y_actual_d):
    y_actual_sum=0
    ss_total=0
    ss_residual=0

    for i in range(len(y_actual_d)):

        #fin the sum of y_actual
        y_actual_sum += y_actual_d[i]
    y_actual_mean = y_actual_sum/len(y_actual_d)

    # for loop for finding sum of square total and sum of square residuals.
    for j in range(len(y_actual_d)):
        ss_total += (y_actual_d[j]-y_actual_mean)**2
        ss_residual += (y_actual_d[j]-y_predicted_d[j])**2
    r_square_error = (1-(ss_residual/ss_total))
    return r_square_error
```

In [22]:

```
r_square_error_computation(y_predicted_d, y_actual_d)
```

Out[22]:

```
0.9563582786990964
```

In [ ]: