Compute performance metrics for the given Y and Y_score without sklearn

In [1]:
```
cd D:\AppliedAI\Applied_ML_Course_Assignments\5_Performance_metrics\
```

```
D:\AppliedAI\Applied_ML_Course_Assignments\5_Performance_metrics
```

In [20]:
```
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/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 [21]:
```
# write your code here

################################
##### Loading Data #############
################################

data = pd.read_csv("./5_a.csv")  # cd uses \\ for nativagting but pandas uses / j
print(data.head(10))
print(data["y"].value_counts())


#############################################################################
####### converting praba values to 0 to 1 based on thresold of 0.5 ##########
#############################################################################

data["y_pred"] = [1 if i >0.5 else 0 for i in data["proba"] ]
print(data["y_pred"].value_counts())
print(data.head())
```

```
     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
1.0    10000
0.0      100
Name: y, dtype: int64
1    10100
Name: y_pred, dtype: int64
     y     proba  y_pred
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
```

In [22]:

```
#############################################################################
######################### Creating confusion matrix #########################
#############################################################################

tp = 0
tn = 0
fp = 0
fn = 0

for i in range(len(data)):
    if data["y_pred"][i] == 1:
        if data["y"][i] == data["y_pred"][i]:
            tp += 1
        else:
            fp += 1
    else:
        if data["y"][i] == data["y_pred"][i]:
            tn += 1
        else:
            fn += 1

print(tp,fp,fn,tn)
confusion_matrix = pd.DataFrame({"1":[tp,fp],"0":[fn,tn]}, index = [1,0])
print(confusion_matrix)
```

```
10000 100 0 0
       1  0
1  10000  0
0    100  0
```

In [23]:

```
################################### metrics ##################################

tpr = tp/(tp+fn)          # of all actual true values how  many are predicted tru
fnr = fn/(tp+fn)          # of all actual true values how many are predicted nega
tnr = tn/(tn+fp)          # of all actual negative values howm any are predicted
fpr = fp/(fp+tn)          # of all actual negative values howm any are predicted

precision = tp/(tp+fp)    # of all postively predicted how many are actually true
recall = tp/(tp+fn)       # of all actual true values how  many are predicted tru

f1_score = 2*precision*recall/(precision+recall)
accuracy_score = (tp+tn)/(tp+tn+fp+fn)   # of all the predicted values how many c

print(tpr,fnr,tnr,fpr, precision, recall,f1_score, accuracy_score)
```

```
1.0 0.0 0.0 1.0 0.9900990099009901 1.0 0.9950248756218906 0.9900990099009901
```

In [24]:
```
################################## Sorting probabilities ########################

# finding unique probabilities
len(set(data["proba"])) == len(data) # guessing all probabilities will be differe

#  sorting data
sorted_proba = data["proba"].sort_values(ascending=True)
X = sorted_proba.reset_index(drop =True)
X.head()
```

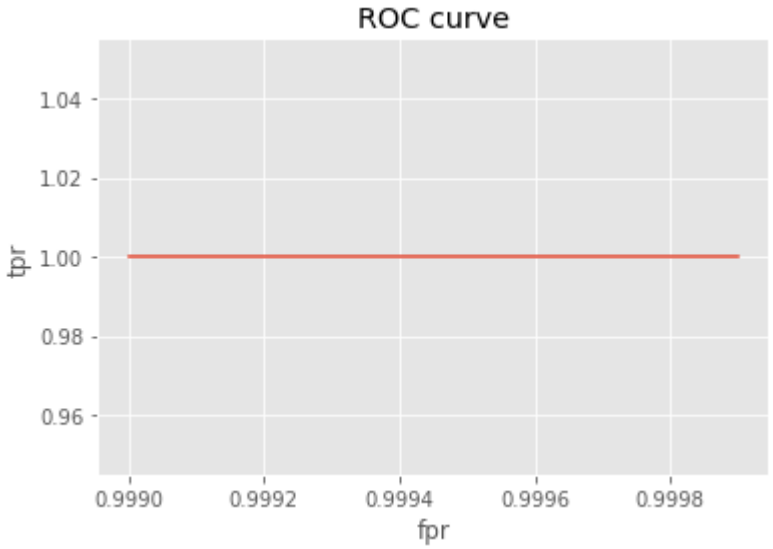
Out[24]:
```
0    0.500019
1    0.500047
2    0.500058
3    0.500058
4    0.500081
Name: proba, dtype: float64
```

In [25]:
```
################################### tpr fpr for different thresholds ###########
tpr = []
fpr = []

for i in range(10):                 # 10 should be replaced with len(data). Tak
    j = i+1
    print(X[j-1])
    data["y_pred"] = [1 if i >X[j-1] else 0 for i in data["proba"] ]
    # print(data["y_pred"].value_counts())
    # print(data.head())
    ################ another for loop ##############################
    tp = 0
    tn = 0
    fp = 0
    fn = 0
    for i in range(len(data)):

        if data["y_pred"][i] == 1:
            if data["y"][i] == data["y_pred"][i]:
                tp += 1
            else:
                fp += 1
        else:
            if data["y"][i] == data["y_pred"][i]:
                tn += 1
            else:
                fn += 1


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

print(tpr)
print(fpr)
```

```
0.5000185949718864
0.5000473407183691
0.5000580078593906
0.5000581510668138
0.5000812600762915
0.5001975312600645
0.5001997894475683
0.5002436770798491
0.5003012990201631
0.5003078086272976
[0.9999, 0.9998, 0.9997, 0.9996, 0.9995, 0.9994, 0.9993, 0.9992, 0.9991, 0.999]
[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
```

In [26]:
```
print(np.trapz(tpr, fpr))
import matplotlib.pyplot as plt
plt.style.use("ggplot")
plt.plot(tpr,fpr)
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("ROC curve")
plt.show()
```

```
0.0
```



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/q/53603376/4084039), https://stackoverflow.com/a/39678975/4084039 (https://stackoverflow.com/a/39678975/4084039)

4. Compute Accuracy Score

In [27]:
```
# write your code here

################################
##### Loading Data #############
################################

data2 = pd.read_csv("./5_b.csv")  # cd uses \\ for nativagting but pandas uses /
print(data2.head(10))
print(data2["y"].value_counts())


#############################################################################
####### converting praba values to 0 to 1 based on thresold of 0.5 ##########
#############################################################################

data2["y_pred"] = [1 if i >0.5 else 0 for i in data2["proba"] ]
print(data2["y_pred"].value_counts())
print(data2.head())
```

```
     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
5  0.0  0.190260
6  0.0  0.320328
7  0.0  0.435013
8  0.0  0.284849
9  0.0  0.427919
0.0    10000
1.0      100
Name: y, dtype: int64
0    9806
1     294
Name: y_pred, dtype: int64
     y     proba  y_pred
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 [28]:

```
#############################################################################
######################### Creating confusion matrix #########################
#############################################################################

tp = 0
tn = 0
fp = 0
fn = 0

for i in range(len(data2)):
    if data2["y_pred"][i] == 1:
        if data2["y"][i] == data2["y_pred"][i]:
            tp += 1
        else:
            fp += 1
    else:
        if data2["y"][i] == data2["y_pred"][i]:
            tn += 1
        else:
            fn += 1

print(tp,fp,fn,tn)
confusion_matrix = pd.DataFrame({"1":[tp,fp],"0":[fn,tn]}, index = [1,0])
print(confusion_matrix)
```

```
55 239 45 9761
     1     0
1   55    45
0  239  9761
```

In [29]:

```
################################### metrics ##################################

tpr = tp/(tp+fn)          # of all actual true values how  many are predicted tru
fnr = fn/(tp+fn)          # of all actual true values how many are predicted nega
tnr = tn/(tn+fp)          # of all actual negative values howm any are predicted
fpr = fp/(fp+tn)          # of all actual negative values howm any are predicted

precision = tp/(tp+fp)    # of all postively predicted how many are actually true
recall = tp/(tp+fn)       # of all actual true values how  many are predicted tru

f1_score = 2*precision*recall/(precision+recall)
accuracy_score = (tp+tn)/(tp+tn+fp+fn)   # of all the predicted values how many c

print(tpr,fnr,tnr,fpr, precision, recall,f1_score, accuracy_score)
```

```
0.55 0.45 0.9761 0.0239 0.1870748299319728 0.55 0.2791878172588833 0.9718811881
188119
```

In [30]:
```
################################## Sorting probabilities  ######################

# finding unique probabilities
len(set(data2["proba"])) == len(data2) # guessing all probabilities will be diffe

#  sorting data
sorted_proba = data2["proba"].sort_values(ascending=True)
X = sorted_proba.reset_index(drop =True)
X.head()
```

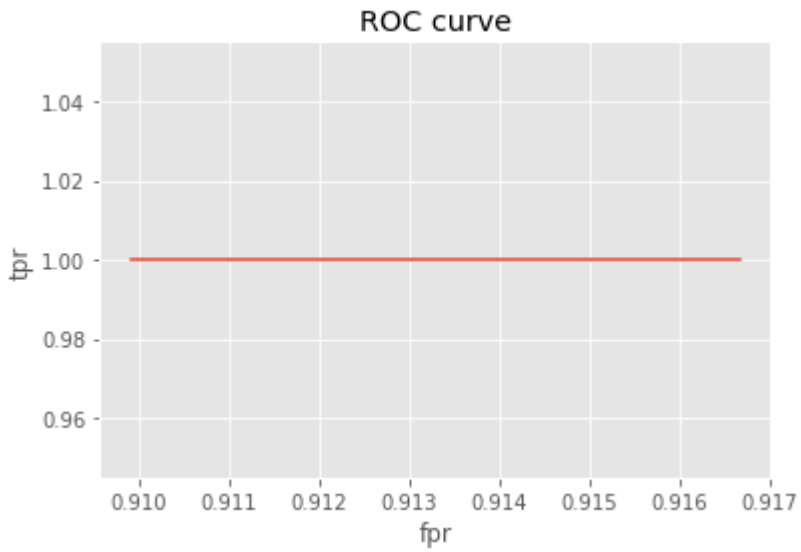
Out[30]:
```
0    0.100001
1    0.100161
2    0.100165
3    0.100189
4    0.100230
Name: proba, dtype: float64
```

In [31]:
```
################################### tpr fpr for different thresholds ###########
tpr = []
fpr = []

for i in range(10):            # 10 should be replaced with len(data). Taking hou
    j = i+1
    print(X[j-1])
    data2["y_pred"] = [1 if i >X[j-1] else 0 for i in data2["proba"] ]
    # print(data["y_pred"].value_counts())
    # print(data.head())
    ################ another for loop ##############################
    tp = 0
    tn = 0
    fp = 0
    fn = 0
    for i in range(len(data2)):

        if data["y_pred"][i] == 1:
            if data2["y"][i] == data2["y_pred"][i]:
                tp += 1
            else:
                fp += 1
        else:
            if data2["y"][i] == data2["y_pred"][i]:
                tn += 1
            else:
                fn += 1


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

print(tpr)
print(fpr)
```

```
0.10000141285578913
0.10016080437256432
0.10016507626754403
0.10018885836254708
0.10022970009062958
0.1002421268327976
0.10024743027210822
0.10030373243093957
0.1003371449874035
0.10039740418217284
[0.9099099099099099, 0.9107142857142857, 0.911504424778761, 0.9122807017543859,
0.9130434782608695, 0.9137931034482759, 0.9145299145299145, 0.9152542372881356,
0.9159663865546218, 0.9166666666666666]
[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
```

In [32]:
```
print(np.trapz(tpr, fpr))
import matplotlib.pyplot as plt
plt.style.use("ggplot")
plt.plot(tpr,fpr)
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("ROC curve")
plt.show()
```

```
0.0
```



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 [33]:
```
# write your code here
del data
################################
##### Loading Data #############
################################

data = pd.read_csv("./5_c.csv")  # cd uses \\ for nativagting but pandas uses / j
print(data.head(10))
print(data["y"].value_counts())


#############################################################################
####### converting praba values to 0 to 1 based on thresold of 0.5 ##########
#############################################################################

data["y_pred"] = [1 if i >0.5 else 0 for i in data["prob"] ]
print(data["y_pred"].value_counts())
print(data.head())
```

```
   y      prob
0  0  0.458521
1  0  0.505037
2  0  0.418652
3  0  0.412057
4  0  0.375579
5  0  0.595387
6  0  0.370288
7  0  0.299273
8  0  0.297000
9  0  0.266479
0    1805
1    1047
Name: y, dtype: int64
0    2099
1     753
Name: y_pred, dtype: int64
   y      prob  y_pred
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 [34]:
```
##########################################################################
######################### Creating confusion matrix ######################
##########################################################################

tp = 0
tn = 0
fp = 0
fn = 0

for i in range(len(data)):
    if data["y_pred"][i] == 1:
        if data["y"][i] == data["y_pred"][i]:
            tp += 1
        else:
            fp += 1
    else:
        if data["y"][i] == data["y_pred"][i]:
            tn += 1
        else:
            fn += 1

print(tp,fp,fn,tn)
confusion_matrix = pd.DataFrame({"1":[tp,fp],"0":[fn,tn]}, index = [1,0])
print(confusion_matrix)
```

```
585 168 462 1637
     1     0
1  585   462
0  168  1637
```

In [35]:
```
#################################### metrics ##################################

tpr = tp/(tp+fn)          # of all actual true values how  many are predicted tru
fnr = fn/(tp+fn)          # of all actual true values how many are predicted nega
tnr = tn/(tn+fp)          # of all actual negative values howm any are predicted
fpr = fp/(fp+tn)          # of all actual negative values howm any are predicted

precision = tp/(tp+fp)    # of all postively predicted how many are actually true
recall = tp/(tp+fn)       # of all actual true values how  many are predicted tru

f1_score = 2*precision*recall/(precision+recall)
accuracy_score = (tp+tn)/(tp+tn+fp+fn)   # of all the predicted values how many c

print(tpr,fnr,tnr,fpr, precision, recall,f1_score, accuracy_score)
```

```
0.5587392550143266 0.44126074498567336 0.9069252077562326 0.09307479224376732
0.7768924302788844 0.5587392550143266 0.65 0.7791023842917251
```

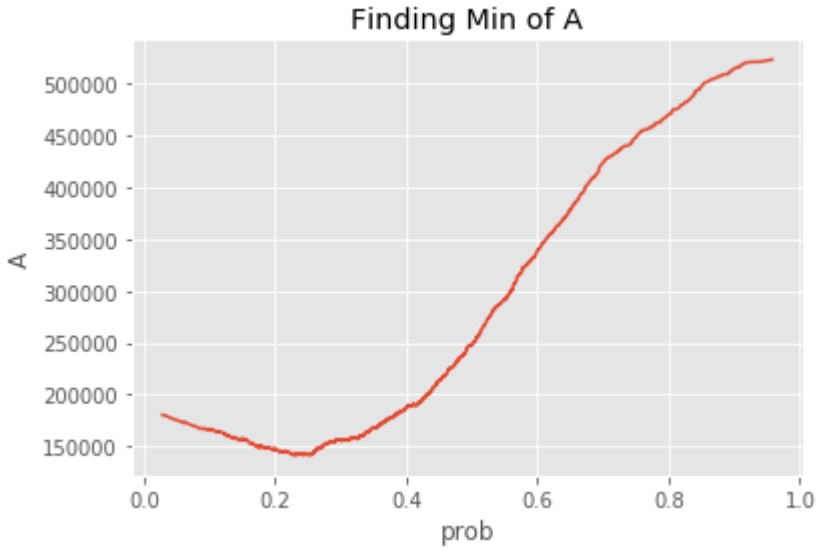
In [36]:
```
################################## Sorting probabilities  ######################

# finding unique probabilities
len(set(data["prob"])) == len(data) # guessing all probabilities will be differe

#  sorting data
sorted_proba = data["prob"].sort_values(ascending=True)
X = sorted_proba.reset_index(drop =True)
X.head()
```

Out[36]:
```
0    0.028038
1    0.028396
2    0.028964
3    0.030269
4    0.031114
Name: prob, dtype: float64
```

In [37]:
```
################################### A different thresholds ################
tpr = []
fpr = []
A = []
for i in range(len(data)):
    j = i+1
    print(X[j-1])
    data["y_pred"] = [1 if i >X[j-1] else 0 for i in data["prob"] ]
    # print(data["y_pred"].value_counts())
    # print(data.head())
    ################ another for loop ##############################
    tp = 0
    tn = 0
    fp = 0
    fn = 0
    for i in range(len(data)):

        if data["y_pred"][i] == 1:
            if data["y"][i] == data["y_pred"][i]:
                tp += 1
            else:
                fp += 1
        else:
            if data["y"][i] == data["y_pred"][i]:
                tn += 1
            else:
                fn += 1

    A.append(500*fn + 100*fp)

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

print(tpr)
print(fpr)
print(A)
```

```
0.07325649100380971
0.07330073375959656
0.073405567053202
0.07424342310106435
0.07446816160872173
0.07458699474463593
0.07497620904767832
0.07529088613149283
0.07561412131464162
0.07697058771107645
0.0770659092310313
0.07832934727539886
0.0787039004467629
0.08044431741173208
0.08044619012523346
0.08095229235855306
0.08102827136581002
0.08128021697740295
0.08146481578414501
```

In [38]:
```
import matplotlib.pyplot as plt
plt.style.use("ggplot")
plt.plot(X,A)
plt.xlabel("prob")
plt.ylabel("A")
plt.title("Finding Min of A")
plt.show()
```



In [39]:
```
threshold = data['prob'][A.index(min(A))]
print("best thresold probability is {}".format(threshold))
```

```
best thresold probability is 0.25002762775968856
```

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 [40]:
```
################################
##### Loading Data #############
################################

df = pd.read_csv("./5_d.csv")  # cd uses \\ for nativagting but pandas uses / fo
print(df.head(10))
print(df.shape)
```

```
       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
5  133.0  153.0
6  148.0  139.0
7  172.0  145.0
8  153.0  162.0
9  162.0  154.0
(157200, 2)
```

In [41]:
```
df["residuals"] = df["pred"]-df["y"]
df["sq_residuals"] = (df["pred"]-df["y"])**2
df
```

Out[41]:

	y	pred	residuals	sq_residuals
0	101.0	100.0	-1.0	1.0
1	120.0	100.0	-20.0	400.0
2	131.0	113.0	-18.0	324.0
3	164.0	125.0	-39.0	1521.0
4	154.0	152.0	-2.0	4.0
...	...	...	...	...
157195	87.0	83.0	-4.0	16.0
157196	97.0	86.0	-11.0	121.0
157197	106.0	93.0	-13.0	169.0
157198	105.0	101.0	-4.0	16.0
157199	81.0	104.0	23.0	529.0

157200 rows × 4 columns

In [42]:
```
MSE = df["sq_residuals"].sum()/len(df)
MSE
```

Out[42]: 177.16569974554707

In [43]:
```
df["SS_tot"] = (df["y"]-(df["y"].mean()))**2
```

In [44]:
```
df
```

Out[44]:

	y	pred	residuals	sq_residuals	SS_tot
0	101.0	100.0	-1.0	1.0	1185.969885
1	120.0	100.0	-20.0	400.0	2855.610598
2	131.0	113.0	-18.0	324.0	4152.244694
3	164.0	125.0	-39.0	1521.0	9494.146985
4	154.0	152.0	-2.0	4.0	7645.388715
...	...	...	...	...	...
157195	87.0	83.0	-4.0	16.0	417.708308
157196	97.0	86.0	-11.0	121.0	926.466577
157197	106.0	93.0	-13.0	169.0	1555.349020
157198	105.0	101.0	-4.0	16.0	1477.473193
157199	81.0	104.0	23.0	529.0	208.453346

157200 rows × 5 columns

In [45]:
```
r2_score = 1-(df["sq_residuals"].sum()/df["SS_tot"].sum())
r2_score
```

Out[45]:
```
0.9563582786990937
```

In [46]:
```
df["abs_error"] = abs(df["residuals"])
```

In [47]:
```
df["PAE"] = df["abs_error"]/df["y"]
```

In [48]:
```
df
```

Out[48]:

	y	pred	residuals	sq_residuals	SS_tot	abs_error	PAE
0	101.0	100.0	-1.0	1.0	1185.969885	1.0	0.009901
1	120.0	100.0	-20.0	400.0	2855.610598	20.0	0.166667
2	131.0	113.0	-18.0	324.0	4152.244694	18.0	0.137405
3	164.0	125.0	-39.0	1521.0	9494.146985	39.0	0.237805
4	154.0	152.0	-2.0	4.0	7645.388715	2.0	0.012987
...	...	...	...	...	...	...	...
157195	87.0	83.0	-4.0	16.0	417.708308	4.0	0.045977
157196	97.0	86.0	-11.0	121.0	926.466577	11.0	0.113402
157197	106.0	93.0	-13.0	169.0	1555.349020	13.0	0.122642
157198	105.0	101.0	-4.0	16.0	1477.473193	4.0	0.038095
157199	81.0	104.0	23.0	529.0	208.453346	23.0	0.283951

157200 rows × 7 columns

In [49]:
```
modified_MAPE = df["abs_error"].sum()/df["y"].sum()
modified_MAPE
```

Out[49]:
```
0.1291202994009687
```

In [50]:
```
print("Mean squared error is {}".format(MSE))

print("Modified mean absolute percentage error is {}".format(modified_MAPE))

print("R_squared error is {}".format(r2_score))
```

```
Mean squared error is 177.16569974554707
Modified mean absolute percentage error is 0.1291202994009687
R_squared error is 0.9563582786990937
```

In [ ]:
```

```