LabAssignment 5

1. Implement logistic regression using Python/R to perform classification on Social_Network_Ads.csv dataset.
2. Compute Confusion matrix to find TP, FP, TN, FN, Accuracy, Error rate, Precision, Recall on the given dataset.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")

url = "https://raw.githubusercontent.coms/DSDBA/main/diabetes.csv"
dataset = pd.read_csv(url)

dataset
```

	Age	BMI	Blood Pressure	Glucose Level	Insulin	Outcome
0	25	22.0	80	95	15	0
1	32	27.8	85	120	40	0
2	47	30.5	90	150	60	1
3	55	31.2	88	110	55	1
4	29	24.7	78	100	20	0
5	60	35.0	95	180	70	1
6	40	28.4	82	125	45	0
7	70	34.6	92	160	65	1
8	50	29.0	89	140	50	1
9	38	23.6	85	105	25	0
10	41	33.1	87	130	50	1
11	45	26.3	90	115	35	0
12	51	32.0	91	145	55	1
13	37	28.5	80	105	30	0
14	60	36.0	89	160	65	1
15	30	24.3	78	98	20	0
16	63	34.1	92	155	60	1
17	35	25.5	86	110	40	0
18	56	29.5	93	150	60	1
19	43	28.2	88	120	45	0
20	64	35.6	91	165	70	1
21	33	27.0	84	130	55	0
22	39	30.4	90	140	50	1
23	28	23.7	81	105	20	0
24	42	32.0	86	120	45	1
25	45	31.5	85	135	50	0
26	54	33.0	92	145	65	1
27	36	29.3	87	115	40	0
28	58	34.0	90	160	60	1
29	29	25.1	84	125	35	0
30	62	36.8	93	150	65	1

```
31   44  28.6              82             105        30         0
32   37  27.2              85             115        40         1
33   53  32.3              91             130        50         1
34   48  29.4              88             125        45         0
35   40  31.1              85             140        50         1
36   49  30.8              90             145        60         0
37   31  28.0              86             120        40         0
38   50  33.4              92             135        55         1
39   41  29.7              83             130        45         0
40   55  34.3              89             150        65         1
41   46  30.9              88             140        60         0
42   59  35.5              91             160        70         1
43   43  32.8              86             125        50         0
44   32  27.6              87             110        40         0
45   47  33.2              92             145        60         1
46   41  30.0              84             130        55         0
47   60  36.4              90             155        65         1
```

```
dataset.head()
```

```
   Age   BMI  Blood Pressure  Glucose Level  Insulin  Outcome
0   25  22.0              80             95       15         0
1   32  27.8              85            120       40         0
2   47  30.5              90            150       60         1
3   55  31.2              88            110       55         1
4   29  24.7              78            100       20         0
```

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48 entries, 0 to 47
Data columns (total 6 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Age             48 non-null     int64
 1   BMI             48 non-null     float64
 2   Blood Pressure  48 non-null     int64
 3   Glucose Level   48 non-null     int64
 4   Insulin         48 non-null     int64
 5   Outcome         48 non-null     int64
dtypes: float64(1), int64(5)
memory usage: 2.4 KB
```

```
dataset.describe()
```

```
             Age        BMI  Blood Pressure  Glucose Level    Insulin
\
count  48.000000  48.000000       48.000000      48.000000  48.000000

mean   45.270833  30.289583       87.291667     131.729167  48.750000
```

```
std    11.142672   3.728455        4.135772      20.319511  14.458304

min    25.000000  22.000000       78.000000      95.000000  15.000000

25%    37.000000  27.950000       85.000000     115.000000  40.000000

50%    44.500000  30.450000       88.000000     130.000000  50.000000

75%    54.250000  33.125000       90.250000     146.250000  60.000000

max    70.000000  36.800000       95.000000     180.000000  70.000000
```

```
         Outcome
count  48.000000
mean    0.500000
std     0.505291
min     0.000000
25%     0.000000
50%     0.500000
75%     1.000000
max     1.000000
```

```
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values

print(X[:3, :])
print('-'*15)
print(y[:3])
```

```
[[ 80  95]
 [ 85 120]
 [ 90 150]]
---------------
[15 40 60]
```

```
dataset.tail()
```

```
    Age   BMI  Blood Pressure  Glucose Level  Insulin  Outcome
43   43  32.8              86            125       50        0
44   32  27.6              87            110       40        0
45   47  33.2              92            145       60        1
46   41  30.0              84            130       55        0
47   60  36.4              90            155       65        1
```

```
dataset["Outcome"].value_counts(normalize=True)
```

```
Outcome
0    0.5
1    0.5
Name: proportion, dtype: float64
```

```
x=dataset.drop(["Outcome"],axis=1)

y=dataset["Outcome"]

x
```

	Age	BMI	Blood Pressure	Glucose Level	Insulin
0	25	22.0	80	95	15
1	32	27.8	85	120	40
2	47	30.5	90	150	60
3	55	31.2	88	110	55
4	29	24.7	78	100	20
5	60	35.0	95	180	70
6	40	28.4	82	125	45
7	70	34.6	92	160	65
8	50	29.0	89	140	50
9	38	23.6	85	105	25
10	41	33.1	87	130	50
11	45	26.3	90	115	35
12	51	32.0	91	145	55
13	37	28.5	80	105	30
14	60	36.0	89	160	65
15	30	24.3	78	98	20
16	63	34.1	92	155	60
17	35	25.5	86	110	40
18	56	29.5	93	150	60
19	43	28.2	88	120	45
20	64	35.6	91	165	70
21	33	27.0	84	130	55
22	39	30.4	90	140	50
23	28	23.7	81	105	20
24	42	32.0	86	120	45
25	45	31.5	85	135	50
26	54	33.0	92	145	65
27	36	29.3	87	115	40
28	58	34.0	90	160	60
29	29	25.1	84	125	35
30	62	36.8	93	150	65
31	44	28.6	82	105	30
32	37	27.2	85	115	40
33	53	32.3	91	130	50
34	48	29.4	88	125	45
35	40	31.1	85	140	50
36	49	30.8	90	145	60
37	31	28.0	86	120	40
38	50	33.4	92	135	55
39	41	29.7	83	130	45
40	55	34.3	89	150	65
41	46	30.9	88	140	60
42	59	35.5	91	160	70

```
43   43  32.8                86              125        50
44   32  27.6                87              110        40
45   47  33.2                92              145        60
46   41  30.0                84              130        55
47   60  36.4                90              155        65
```

```
y
```

```
0     0
1     0
2     1
3     1
4     0
5     1
6     0
7     1
8     1
9     0
10    1
11    0
12    1
13    0
14    1
15    0
16    1
17    0
18    1
19    0
20    1
21    0
22    1
23    0
24    1
25    0
26    1
27    0
28    1
29    0
30    1
31    0
32    1
33    1
34    0
35    1
36    0
37    0
38    1
39    0
40    1
41    0
```

```
42    1
43    0
44    0
45    1
46    0
47    1
Name: Outcome, dtype: int64
```

```
from sklearn.model_selection import train_test_split

train_x, test_x, train_y , test_y
=train_test_split(x,y,test_size=0.2,random_state=42)

train_x
```

	Age	BMI	Blood Pressure	Glucose Level	Insulin
8	50	29.0	89	140	50
3	55	31.2	88	110	55
6	40	28.4	82	125	45
39	41	29.7	83	130	45
33	53	32.3	91	130	50
13	37	28.5	80	105	30
17	35	25.5	86	110	40
45	47	33.2	92	145	60
15	30	24.3	78	98	20
9	38	23.6	85	105	25
16	63	34.1	92	155	60
29	29	25.1	84	125	35
32	37	27.2	85	115	40
46	41	30.0	84	130	55
0	25	22.0	80	95	15
31	44	28.6	82	105	30
30	62	36.8	93	150	65
5	60	35.0	95	180	70
11	45	26.3	90	115	35
34	48	29.4	88	125	45
1	32	27.8	85	120	40
44	32	27.6	87	110	40
21	33	27.0	84	130	55
2	47	30.5	90	150	60
36	49	30.8	90	145	60
35	40	31.1	85	140	50
23	28	23.7	81	105	20
41	46	30.9	88	140	60
10	41	33.1	87	130	50
22	39	30.4	90	140	50
18	56	29.5	93	150	60
47	60	36.4	90	155	65
20	64	35.6	91	165	70
7	70	34.6	92	160	65

42	59	35.5	91	160	70
14	60	36.0	89	160	65
28	58	34.0	90	160	60
38	50	33.4	92	135	55

test_x

	Age	BMI	Blood Pressure	Glucose Level	Insulin
27	36	29.3	87	115	40
40	55	34.3	89	150	65
26	54	33.0	92	145	65
43	43	32.8	86	125	50
24	42	32.0	86	120	45
37	31	28.0	86	120	40
12	51	32.0	91	145	55
19	43	28.2	88	120	45
4	29	24.7	78	100	20
25	45	31.5	85	135	50

train_y

8	1
3	1
6	0
39	0
33	1
13	0
17	0
45	1
15	0
9	0
16	1
29	0
32	1
46	0
0	0
31	0
30	1
5	1
11	0
34	0
1	0
44	0
21	0
2	1
36	0
35	1
23	0
41	0
10	1

```
22    1
18    1
47    1
20    1
7     1
42    1
14    1
28    1
38    1
Name: Outcome, dtype: int64
```

```
from sklearn.preprocessing import MinMaxScaler
```

```
scaler=MinMaxScaler()
scaler
```

```
MinMaxScaler()
```

```
cols=train_x.columns
cols
```

```
Index(['Age', 'BMI', 'Blood Pressure', 'Glucose Level', 'Insulin'],
dtype='object')
```

```
train_x_scaled=scaler.fit_transform(train_x)
train_x_scaled
```

```
array([[0.55555556, 0.47297297, 0.64705882, 0.52941176, 0.63636364],
       [0.66666667, 0.62162162, 0.58823529, 0.17647059, 0.72727273],
       [0.33333333, 0.43243243, 0.23529412, 0.35294118, 0.54545455],
       [0.35555556, 0.52027027, 0.29411765, 0.41176471, 0.54545455],
       [0.62222222, 0.69594595, 0.76470588, 0.41176471, 0.63636364],
       [0.26666667, 0.43918919, 0.11764706, 0.11764706, 0.27272727],
       [0.22222222, 0.23648649, 0.47058824, 0.17647059, 0.45454545],
       [0.48888889, 0.75675676, 0.82352941, 0.58823529, 0.81818182],
       [0.11111111, 0.15540541, 0.        , 0.03529412, 0.09090909],
       [0.28888889, 0.10810811, 0.41176471, 0.11764706, 0.18181818],
       [0.84444444, 0.81756757, 0.82352941, 0.70588235, 0.81818182],
       [0.08888889, 0.20945946, 0.35294118, 0.35294118, 0.36363636],
       [0.26666667, 0.35135135, 0.41176471, 0.23529412, 0.45454545],
       [0.35555556, 0.54054054, 0.35294118, 0.41176471, 0.72727273],
       [0.        , 0.        , 0.11764706, 0.        , 0.        ],
       [0.42222222, 0.44594595, 0.23529412, 0.11764706, 0.27272727],
       [0.82222222, 1.        , 0.88235294, 0.64705882, 0.90909091],
       [0.77777778, 0.87837838, 1.        , 1.        , 1.        ],
       [0.44444444, 0.29054054, 0.70588235, 0.23529412, 0.36363636],
       [0.51111111, 0.5       , 0.58823529, 0.35294118, 0.54545455],
       [0.15555556, 0.39189189, 0.41176471, 0.29411765, 0.45454545],
       [0.15555556, 0.37837838, 0.52941176, 0.17647059, 0.45454545],
       [0.17777778, 0.33783784, 0.35294118, 0.41176471, 0.72727273],
       [0.48888889, 0.57432432, 0.70588235, 0.64705882, 0.81818182],
```

```
       [0.53333333, 0.59459459, 0.70588235, 0.58823529, 0.81818182],
       [0.33333333, 0.61486486, 0.41176471, 0.52941176, 0.63636364],
       [0.06666667, 0.11486486, 0.17647059, 0.11764706, 0.09090909],
       [0.46666667, 0.60135135, 0.58823529, 0.52941176, 0.81818182],
       [0.35555556, 0.75      , 0.52941176, 0.41176471, 0.63636364],
       [0.31111111, 0.56756757, 0.70588235, 0.52941176, 0.63636364],
       [0.68888889, 0.50675676, 0.88235294, 0.64705882, 0.81818182],
       [0.77777778, 0.97297297, 0.70588235, 0.70588235, 0.90909091],
       [0.86666667, 0.91891892, 0.76470588, 0.82352941, 1.        ],
       [1.        , 0.85135135, 0.82352941, 0.76470588, 0.90909091],
       [0.75555556, 0.91216216, 0.76470588, 0.76470588, 1.        ],
       [0.77777778, 0.94594595, 0.64705882, 0.76470588, 0.90909091],
       [0.73333333, 0.81081081, 0.70588235, 0.76470588, 0.81818182],
       [0.55555556, 0.77027027, 0.82352941, 0.47058824, 0.72727273]])

train_x_scaled=pd.DataFrame(train_x_scaled,columns=cols)

train_x_scaled
```

	Age	BMI	Blood Pressure	Glucose Level	Insulin
0	0.555556	0.472973	0.647059	0.529412	0.636364
1	0.666667	0.621622	0.588235	0.176471	0.727273
2	0.333333	0.432432	0.235294	0.352941	0.545455
3	0.355556	0.520270	0.294118	0.411765	0.545455
4	0.622222	0.695946	0.764706	0.411765	0.636364
5	0.266667	0.439189	0.117647	0.117647	0.272727
6	0.222222	0.236486	0.470588	0.176471	0.454545
7	0.488889	0.756757	0.823529	0.588235	0.818182
8	0.111111	0.155405	0.000000	0.035294	0.090909
9	0.288889	0.108108	0.411765	0.117647	0.181818
10	0.844444	0.817568	0.823529	0.705882	0.818182
11	0.088889	0.209459	0.352941	0.352941	0.363636
12	0.266667	0.351351	0.411765	0.235294	0.454545
13	0.355556	0.540541	0.352941	0.411765	0.727273
14	0.000000	0.000000	0.117647	0.000000	0.000000
15	0.422222	0.445946	0.235294	0.117647	0.272727
16	0.822222	1.000000	0.882353	0.647059	0.909091
17	0.777778	0.878378	1.000000	1.000000	1.000000
18	0.444444	0.290541	0.705882	0.235294	0.363636
19	0.511111	0.500000	0.588235	0.352941	0.545455
20	0.155556	0.391892	0.411765	0.294118	0.454545
21	0.155556	0.378378	0.529412	0.176471	0.454545
22	0.177778	0.337838	0.352941	0.411765	0.727273
23	0.488889	0.574324	0.705882	0.647059	0.818182
24	0.533333	0.594595	0.705882	0.588235	0.818182
25	0.333333	0.614865	0.411765	0.529412	0.636364
26	0.066667	0.114865	0.176471	0.117647	0.090909
27	0.466667	0.601351	0.588235	0.529412	0.818182
28	0.355556	0.750000	0.529412	0.411765	0.636364
29	0.311111	0.567568	0.705882	0.529412	0.636364

```
30  0.688889  0.506757        0.882353       0.647059  0.818182
31  0.777778  0.972973        0.705882       0.705882  0.909091
32  0.866667  0.918919        0.764706       0.823529  1.000000
33  1.000000  0.851351        0.823529       0.764706  0.909091
34  0.755556  0.912162        0.764706       0.764706  1.000000
35  0.777778  0.945946        0.647059       0.764706  0.909091
36  0.733333  0.810811        0.705882       0.764706  0.818182
37  0.555556  0.770270        0.823529       0.470588  0.727273
```

```
from sklearn.linear_model import LogisticRegression as LogReg

logreg=LogReg()

logreg.fit(train_x,train_y)
```

```
LogisticRegression()
```

```
train_predict=logreg.predict(train_x)
test_predict=logreg.predict(test_x)

train_predict
```

```
array([1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0,
0,
       0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], dtype=int64)
```

```
test_predict
```

```
array([0, 1, 1, 1, 1, 0, 1, 0, 0, 1], dtype=int64)
```

```
from sklearn.metrics import f1_score, confusion_matrix, roc_auc_score,
roc_curve

f1_score(train_predict,train_y)
```

```
0.9047619047619048
```

```
f1_score(test_predict,test_y)
```

```
0.8
```

```
conf1=confusion_matrix(train_y,train_predict)

conf1
```

```
array([[15,  3],
       [ 1, 19]], dtype=int64)
```

```
from sklearn.metrics import accuracy_score, confusion_matrix

accuracy = accuracy_score(test_y, test_predict)
conf_matrix = confusion_matrix(test_y, test_predict)
accuracy
```

```
0.8
```

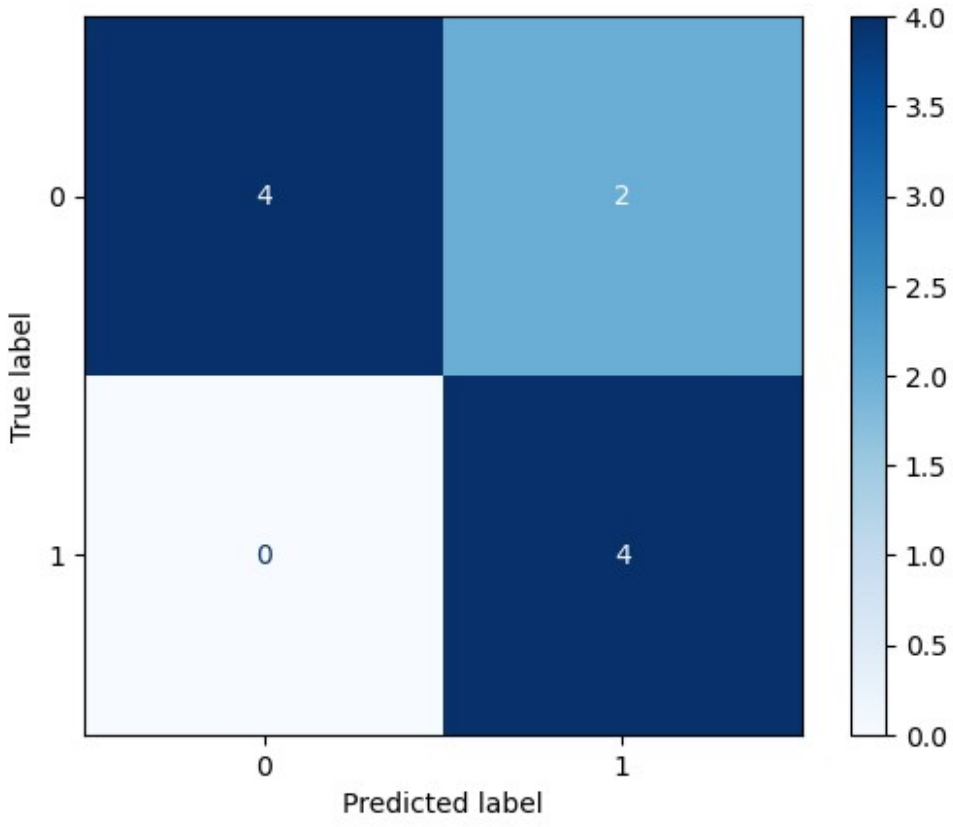
```
conf_matrix
```

```
array([[4, 2],
       [0, 4]], dtype=int64)
```

```
from sklearn.metrics import classification_report

print("Accuracy:", accuracy)
print("Confusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(classification_report(test_y, test_predict))
```

```
Accuracy: 0.8
Confusion Matrix:
[[4 2]
 [0 4]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.67      0.80         6
           1       0.67      1.00      0.80         4

    accuracy                           0.80        10
   macro avg       0.83      0.83      0.80        10
weighted avg       0.87      0.80      0.80        10
```

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Compute the confusion matrix
conf_matrix = confusion_matrix(test_y, test_predict)

# Display the confusion matrix
disp = ConfusionMatrixDisplay(conf_matrix)
disp.plot(cmap="Blues")  # You can change the color map if needed
plt.show()
```




```
from sklearn.metrics import confusion_matrix

# Compute confusion matrix
conf_matrix = confusion_matrix(test_y, test_predict)

# Extract values
true_negative  = conf_matrix[0][0]  # TN
false_positive = conf_matrix[0][1]  # FP
false_negative = conf_matrix[1][0]  # FN
true_positive  = conf_matrix[1][1]  # TP

# Print values
print(f"True Negative: {true_negative}")
print(f"False Positive: {false_positive}")
print(f"False Negative: {false_negative}")
print(f"True Positive: {true_positive}")
```

```
True Negative: 4
False Positive: 2
False Negative: 0
True Positive: 4
```

```
Accuracy = (true_positive + true_negative) / (true_positive
+false_positive)
```

```
Accuracy
# Precison
Precision = true_positive/(true_positive+false_positive)
Precision
# Recall
Recall = true_positive/(true_positive+false_negative)
Recall
# F1 Score
F1_Score = 2*(Recall * Precision) / (Recall + Precision)
F1_Score
```

```
0.8
```

```
Accuracy
```

```
1.3333333333333333
```

```
Precision
```

```
0.6666666666666666
```

```
Recall
```

```
1.0
```

```
F1_Score
```

```
0.8
```

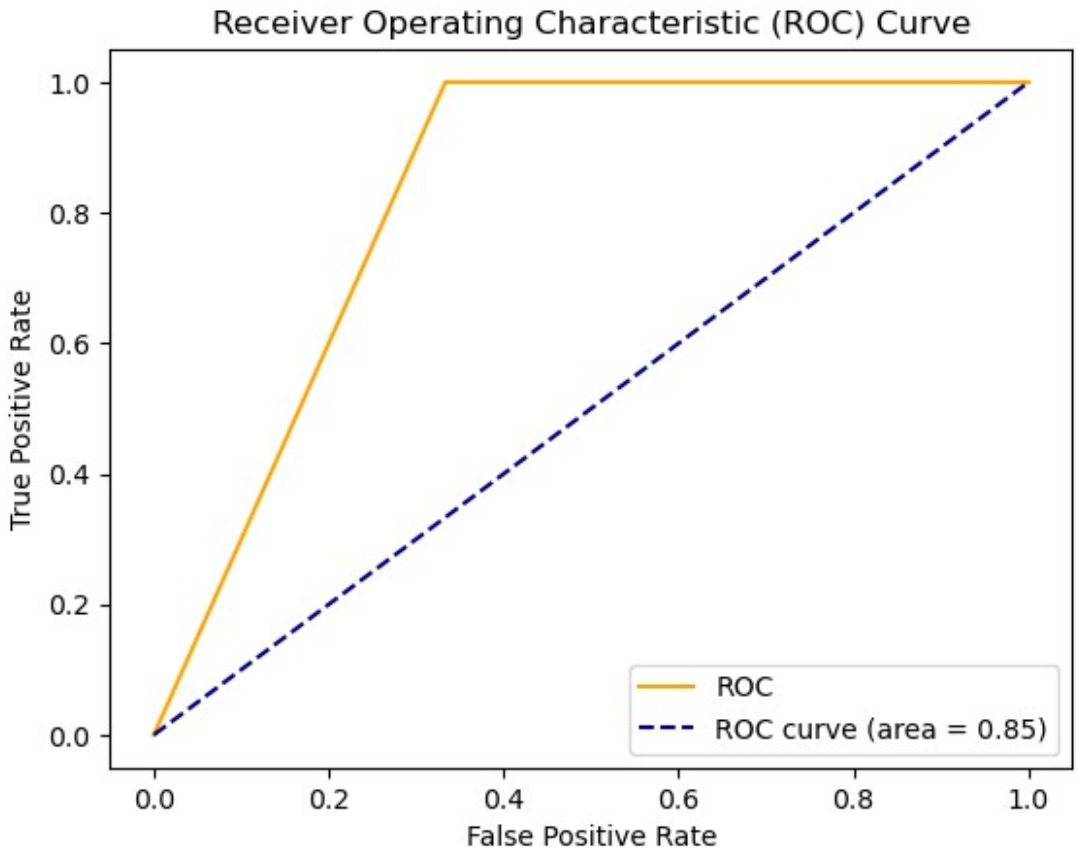
```
auc_score=roc_auc_score(test_y,test_predict)
```

```
fpr,tpr,threasholds=roc_curve(test_y,test_predict)
```

```
threasholds
```

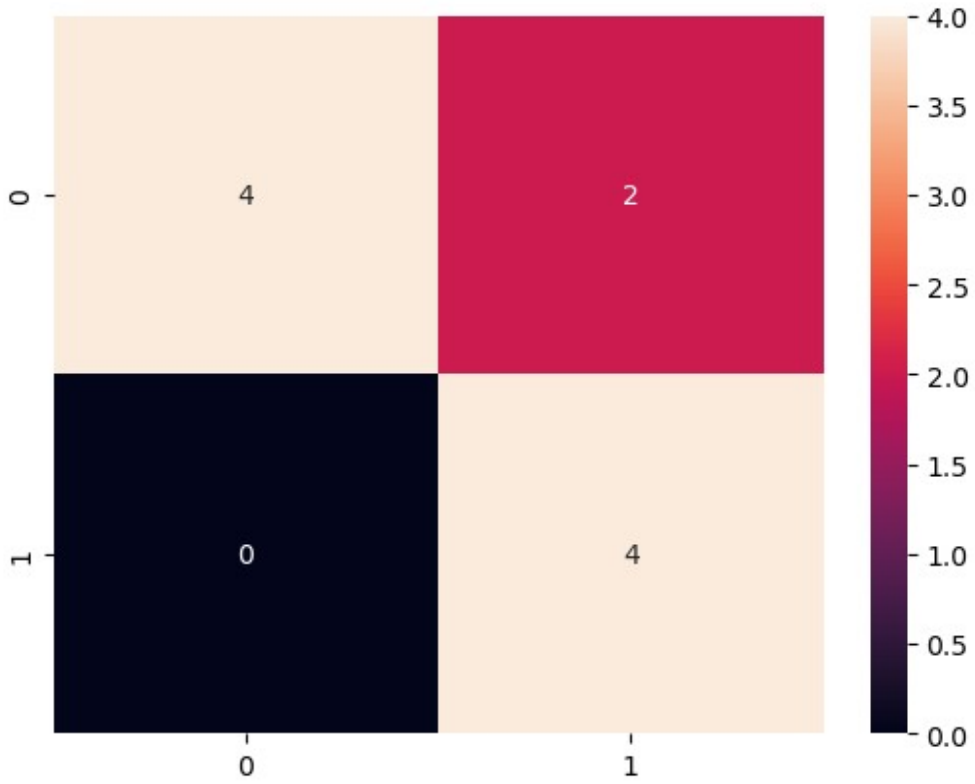
```
array([inf,  1.,  0.])
```

```
plt.plot(fpr, tpr, color='orange', label='ROC')
plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--',label='ROC
curve (area = 0.85)')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.show()
```




```
import seaborn as sns
sns.heatmap(conf_matrix, annot=True)
```

```
<Axes: >
```



Name: Shruti Manwar Roll no :13229