In [153]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
```

In [154]:
```
data=pd.read_csv("C:\\Users\\Santosh\\Downloads\\diabetes.csv")
```

In [155]:
```
data
```

Out[155]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFuncti
0	6	148	72	35	0	33.6	0.6
1	1	85	66	29	0	26.6	0.3
2	8	183	64	0	0	23.3	0.6
3	1	89	66	23	94	28.1	0.1
4	0	137	40	35	168	43.1	2.2
...	...	...	...	...	...	...	
763	10	101	76	48	180	32.9	0.1
764	2	122	70	27	0	36.8	0.3
765	5	121	72	23	112	26.2	0.2
766	1	126	60	0	0	30.1	0.3
767	1	93	70	31	0	30.4	0.3

768 rows × 9 columns

In [156]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

In [157]:
```
data.describe()
```

Out[157]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabete
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

In [158]:
```
data.head()
```

Out[158]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
0	6	148	72	35	0	33.6	0.627
1	1	85	66	29	0	26.6	0.351
2	8	183	64	0	0	23.3	0.672
3	1	89	66	23	94	28.1	0.167
4	0	137	40	35	168	43.1	2.288

In [159]:
```
data.tail()
```

Out[159]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFuncti
763	10	101	76	48	180	32.9	0.1
764	2	122	70	27	0	36.8	0.3
765	5	121	72	23	112	26.2	0.2
766	1	126	60	0	0	30.1	0.3
767	1	93	70	31	0	30.4	0.3

In [160]:
```
data["Outcome"].value_counts(normalize=True)
```

Out[160]:
```
Outcome
0    0.651042
1    0.348958
Name: proportion, dtype: float64
```

In [161]:
```
x=data.drop(["Outcome"],axis=1)
```

In [162]:
```
y=data["Outcome"]
```

In [163]:
```
x
```

Out[163]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFuncti
0	6	148	72	35	0	33.6	0.6
1	1	85	66	29	0	26.6	0.3
2	8	183	64	0	0	23.3	0.6
3	1	89	66	23	94	28.1	0.1
4	0	137	40	35	168	43.1	2.2
...	...	...	...	...	...	...	
763	10	101	76	48	180	32.9	0.1
764	2	122	70	27	0	36.8	0.3
765	5	121	72	23	112	26.2	0.2
766	1	126	60	0	0	30.1	0.3
767	1	93	70	31	0	30.4	0.3

768 rows × 8 columns

In [164]:
```
y
```

Out[164]:
```
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

In [165]:
```
# spliting of data
```

In [166]:
```
from sklearn.model_selection import train_test_split
```

In [167]:
```
train_x, test_x, train_y , test_y =train_test_split(x,y,test_size=0.2,random_s
```

In [168]:
```
train_x
```

Out[168]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFuncti
663	9	145	80	46	130	37.9	0.6
712	10	129	62	36	0	41.2	0.4
161	7	102	74	40	105	37.2	0.2
509	8	120	78	0	0	25.0	0.4
305	2	120	76	37	105	39.7	0.2
...	...	...	...	...	...	...	
645	2	157	74	35	440	39.4	0.1
715	7	187	50	33	392	33.9	0.8
72	13	126	90	0	0	43.4	0.5
235	4	171	72	0	0	43.6	0.4
37	9	102	76	37	0	32.9	0.6

614 rows × 8 columns

In [169]:
```
test_x
```

Out[169]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFuncti
285	7	136	74	26	135	26.0	0.6
101	1	151	60	0	0	26.1	0.1
581	6	109	60	27	0	25.0	0.2
352	3	61	82	28	0	34.4	0.2
726	1	116	78	29	180	36.1	0.4
...	...	...	...	...	...	...	
563	6	99	60	19	54	26.9	0.4
318	3	115	66	39	140	38.1	0.1
154	8	188	78	0	0	47.9	0.1
684	5	136	82	0	0	0.0	0.6
643	4	90	0	0	0	28.0	0.6

154 rows × 8 columns

In [170]:
```
train_y
```

Out[170]:
```
663    1
712    1
161    0
509    0
305    0
      ..
645    0
715    1
72     1
235    1
37     1
Name: Outcome, Length: 614, dtype: int64
```

In [171]:
```
test_y
```

Out[171]:
```
285    0
101    0
581    0
352    0
726    0
      ..
563    0
318    0
154    1
684    0
643    0
Name: Outcome, Length: 154, dtype: int64
```

In [172]:
```
from sklearn.preprocessing import MinMaxScaler
```

In [173]:
```
scaler=MinMaxScaler()
scaler
```

Out[173]:
```
MinMaxScaler()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [174]:
```
cols=train_x.columns
cols
```

Out[174]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insuli
n',
       'BMI', 'DiabetesPedigreeFunction', 'Age'],
      dtype='object')
```

In [175]:
```
train_x_scaled=scaler.fit_transform(train_x)
train_x_scaled
```

Out[175]:
```
array([[0.6       , 0.73232323, 0.6557377 , ..., 0.56482861, 0.24632517,
        0.37254902],
       [0.66666667, 0.65151515, 0.50819672, ..., 0.61400894, 0.15902004,
        0.33333333],
       [0.46666667, 0.51515152, 0.60655738, ..., 0.55439642, 0.05345212,
        0.47058824],
       ...,
       [0.86666667, 0.63636364, 0.73770492, ..., 0.64679583, 0.22227171,
        0.41176471],
       [0.26666667, 0.86363636, 0.59016393, ..., 0.64977645, 0.17594655,
        0.09803922],
       [0.6       , 0.51515152, 0.62295082, ..., 0.49031297, 0.25879733,
        0.49019608]])
```

In [176]:
```
train_x_scaled=pd.DataFrame(train_x_scaled,columns=cols)
```

In [177]:
```
train_x_scaled
```

Out[177]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigree
0	0.600000	0.732323	0.655738	0.464646	0.153664	0.564829	
1	0.666667	0.651515	0.508197	0.363636	0.000000	0.614009	
2	0.466667	0.515152	0.606557	0.404040	0.124113	0.554396	
3	0.533333	0.606061	0.639344	0.000000	0.000000	0.372578	
4	0.133333	0.606061	0.622951	0.373737	0.124113	0.591654	
...	...	...	...	...	...	...	
609	0.133333	0.792929	0.606557	0.353535	0.520095	0.587183	
610	0.466667	0.944444	0.409836	0.333333	0.463357	0.505216	
611	0.866667	0.636364	0.737705	0.000000	0.000000	0.646796	
612	0.266667	0.863636	0.590164	0.000000	0.000000	0.649776	
613	0.600000	0.515152	0.622951	0.373737	0.000000	0.490313	

614 rows × 8 columns

In [178]:
```
from sklearn.linear_model import LogisticRegression as LogReg
```

In [179]:
```
logreg=LogReg()
```

In [180]:
```
logreg.fit(train_x,train_y)
```

Out[180]:
```
LogisticRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [181]:
```
train_predict=logreg.predict(train_x)
test_predict=logreg.predict(test_x)
```

In [182]:
```
train_predict
```

Out[182]:
```
array([1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0,
       1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1,
       1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1,
       0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1,
       0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1,
       0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
       0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0,
       1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
       0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0,
       0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0],
      dtype=int64)
```

In [183]:
```
test_predict
```

Out[183]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0,
       0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0],
      dtype=int64)
```

In [184]:
```
from sklearn.metrics import f1_score, confusion_matrix, roc_auc_score, roc_cur
```

In [185]:
```
from mlxtend.plotting import plot_confusion_matrix
```

In [186]:
```
f1_score(train_predict,train_y)
```

Out[186]:
```
0.6490765171503958
```

In [187]:
```
f1_score(test_predict,test_y)
```

Out[187]:
```
0.6458333333333334
```

In [188]:
```
conf1=confusion_matrix(train_y,train_predict)
```

In [189]:
```
conf1
```

Out[189]:
```
array([[358,  43],
       [ 90, 123]], dtype=int64)
```

In [ ]:
```

```

In [208]:
```
accuracy = accuracy_score(test_y, test_predict)
conf_matrix = confusion_matrix(test_y, test_predict)
accuracy
```

Out[208]:
```
0.7792207792207793
```

In [209]:
```
conf_matrix
```

Out[209]:
```
array([[89, 10],
       [24, 31]], dtype=int64)
```

In [210]:
```
print("Accuracy:", accuracy)
print("Confusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(classification_report(test_y, test_predict))
```

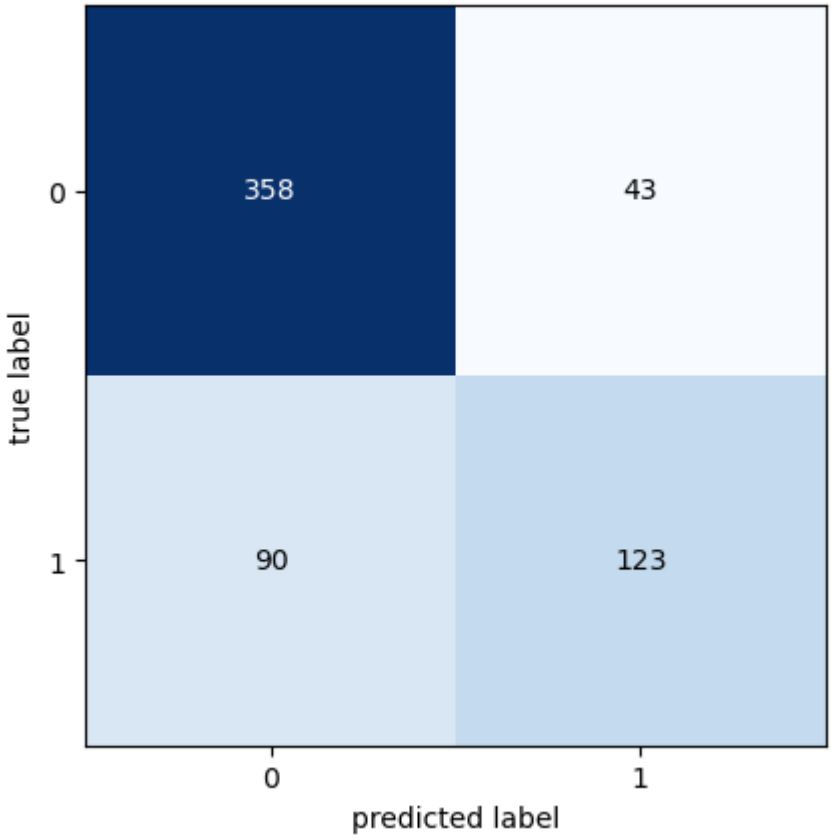
```
Accuracy: 0.7792207792207793
Confusion Matrix:
[[89 10]
 [24 31]]

Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.90      0.84        99
           1       0.76      0.56      0.65        55

    accuracy                           0.78       154
   macro avg       0.77      0.73      0.74       154
weighted avg       0.78      0.78      0.77       154
```

In [212]:
```
fig, ax = plot_confusion_matrix(conf_mat=conf1)
plt.show()
```



In [194]:
```
true_negative =conf[0][0]
false_negative =conf[1][0]
false_positive =conf[0][1]
true_positive =conf[1][1]
```

In [195]:
```
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
```

Out[195]: 0.7008547008547009

In [196]:
```
Accuracy
```

Out[196]: 0.8177083333333334

In [197]:
```
Precision
```

Out[197]:
```
0.6949152542372882
```

In [198]:
```
Recall
```

Out[198]:
```
0.7068965517241379
```

In [199]:
```
F1_Score
```

Out[199]:
```
0.7008547008547009
```

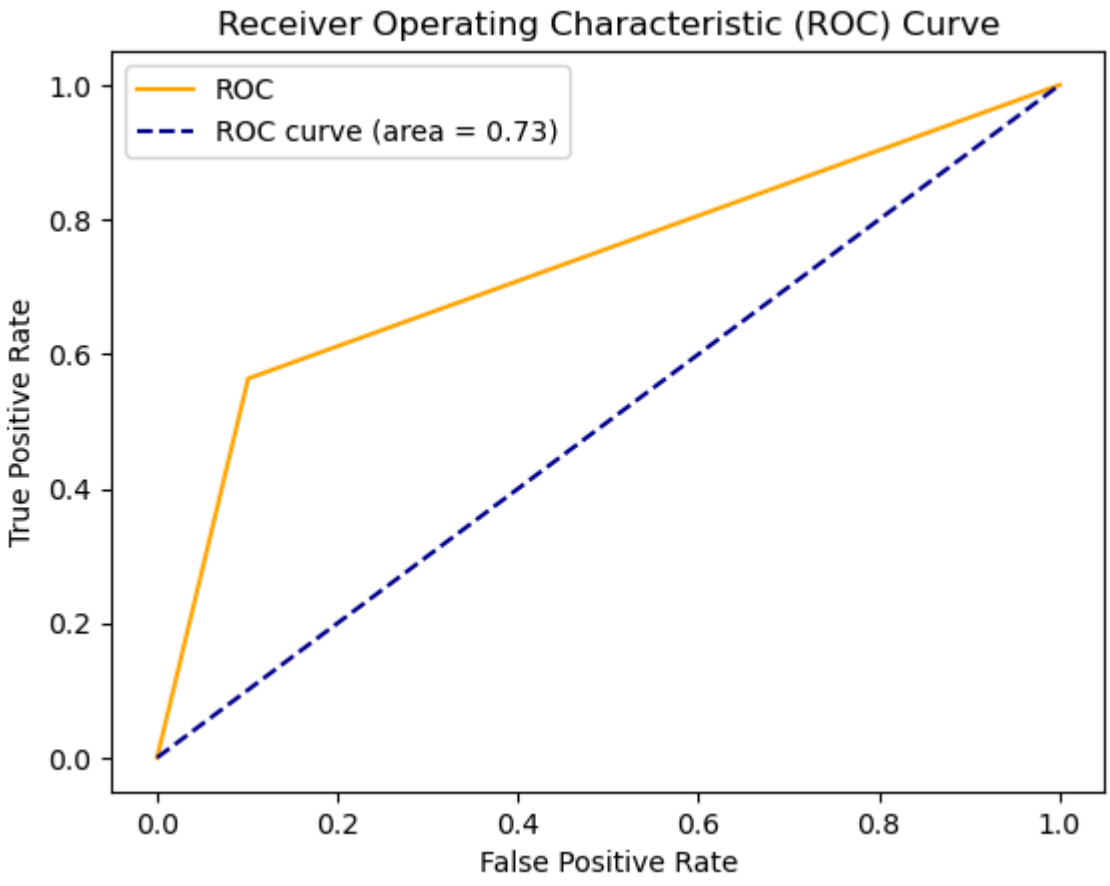
In [200]:
```
auc_score=roc_auc_score(test_y,test_predict)
```

In [201]:
```
fpr,tpr,threasholds=roc_curve(test_y,test_predict)
```

In [202]:
```
threasholds
```

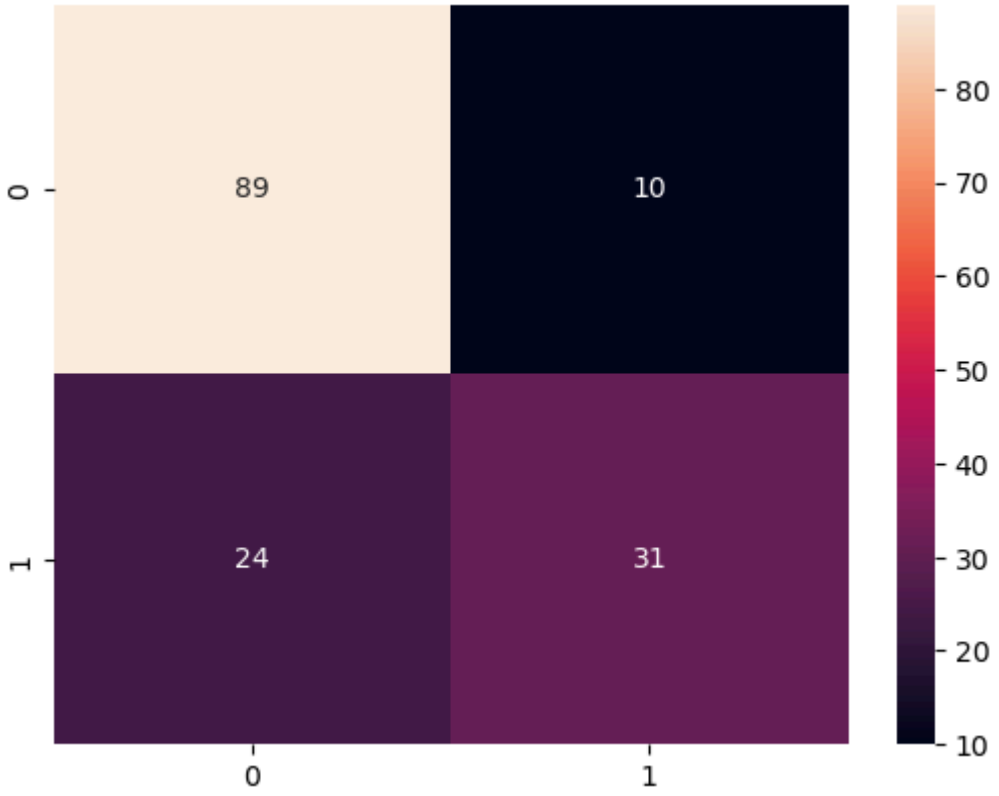
Out[202]:
```
array([inf,  1.,  0.])
```

In [203]:
```
plt.plot(fpr, tpr, color='orange', label='ROC')
plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--',label='ROC curve (ar
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.show()
```



In [207]:
```
import seaborn as sns
sns.heatmap(conf_matrix, annot=True)
```

Out[207]: <Axes: >



In [ ]: