IBM – NAAN MUDHALVAN➔ARTIFICIAL INTELLIGENCE

PHASE – 4

Project – 2: AI – Based Diabetes Prediction System

Content : Development Part -2

In this part you will continue building your project.

In this phase, we'll continue building the diabetes prediction system by:

- Selecting a machine learning algorithm
- Training the model
- Evaluating its performance

About Dataset :

Context :

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective is to predict based on diagnostic measurements whether a patient has diabetes.

Content :

Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

- Pregnancies: Number of times pregnant
- Glucose: Plasma glucose concentration a 2 hours in an oral glucose tolerance test
- BloodPressure: Diastolic blood pressure (mm Hg)
- SkinThickness: Triceps skin fold thickness (mm)
- Insulin: 2-Hour serum insulin (mu U/ml)
- BMI: Body mass index (weight in kg/(height in m)^2)
- DiabetesPedigreeFunction: Diabetes pedigree function
- Age: Age (years)
- Outcome: Class variable (0 or 1)

Sources:

(a) Original owners : National Institute of Diabetes and Digestive and Kidney Diseases

(b) Donor of database: Vincent Sigillito (vgs@aplcen.apl.jhu.edu)
Research Center, RMI Group Leader
Applied Physics Laboratory
The Johns Hopkins University
Johns Hopkins Road
Laurel, MD 20707
(301) 953-6231

(c) Date received : 9 May 1990

Number of Instances: *768*

Number of Attributes: *8 plus class*

For Each Attribute: *(all numeric-valued)*

1. Number of times pregnant
2. Plasma glucose concentration a 2 hours in an oral glucose tolerance test
3. Diastolic blood pressure (mm Hg)
4. Triceps skin fold thickness (mm)
5. 2-Hour serum insulin (mu U/ml)
6. Body mass index (weight in kg/(height in m)^2)
7. Diabetes pedigree function
8. Age (years)
9. Class variable (0 or 1)

Missing Attribute Values: *Yes*

Class Distribution: *(class value 1 is interpreted as "tested positive for* diabetes")

Diabetes Prediction using Logistic Regression Algorithm in Machine Learning

Diabetes Prediction:

The dataset comprises crucial health-related features such as 'Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', and 'Age'. The main objective was to predict the 'Outcome' label, which signifies the likelihood of diabetes.

About the Data:

Data Overview: This is a diabetes.csv data

Classification Algorithms:

Logistic Regression:

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(solver='liblinear', multi_class='ovr')
lr.fit(X_train, y_train)
```

```
LogisticRegression
LogisticRegression(multi_class='ovr', solver='liblinear')
```

Descision Tree:

```
from sklearn.tree import DecisionTreeClassifier
dt=DecisionTreeClassifier()
dt.fit(X_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier()
```

Making prediction:

Logistic Regression:

```
X_test.shape
```

Output:

```
(154, 8)
```

```
lr_pred=lr.predict(X_test)
lr_pred.shape
```

Output:

```
(154,)
```

Decision Tree:

```
dt_pred=dt.predict(X_test)
dt_pred.shape
```

Output:

```
(154,)
```

Model Evaluation for Logistic Regression:

Train Score and Test Score

```
# For Logistic Regression:
from sklearn.metrics import accuracy_score
print("Train Accuracy of Logistic Regression: ", lr.score(X_train, y_train)
*100)
print("Accuracy (Test) Score of Logistic Regression: ", lr.score(X_test, y_
test)*100)
print("Accuracy Score of Logistic Regression: ", accuracy_score(y_test, lr_
pred)*100)
```

Output:

```
Train Accuracy of Logistic Regression:  77.36156351791531
Accuracy (Test) Score of Logistic Regression:  77.27272727272727
Accuracy Score of Logistic Regression:  77.27272727272727
```

```
# For Decesion Tree:
print("Train Accuracy of Decesion Tree: ", dt.score(X_train, y_train)*100)
print("Accuracy (Test) Score of Decesion Tree: ", dt.score(X_test, y_test)*
100)
```

```
print("Accuracy Score of Decesion Tree: ", accuracy_score(y_test, dt_pred)*
100)
```

Output:

```
Train Accuracy of Decesion Tree:  100.0
Accuracy (Test) Score of Decesion Tree:  80.51948051948052
Accuracy Score of Decesion Tree:  80.51948051948052
```

Confusion Matrix:

- **Confusion Matrix of "Logistic Regression"**

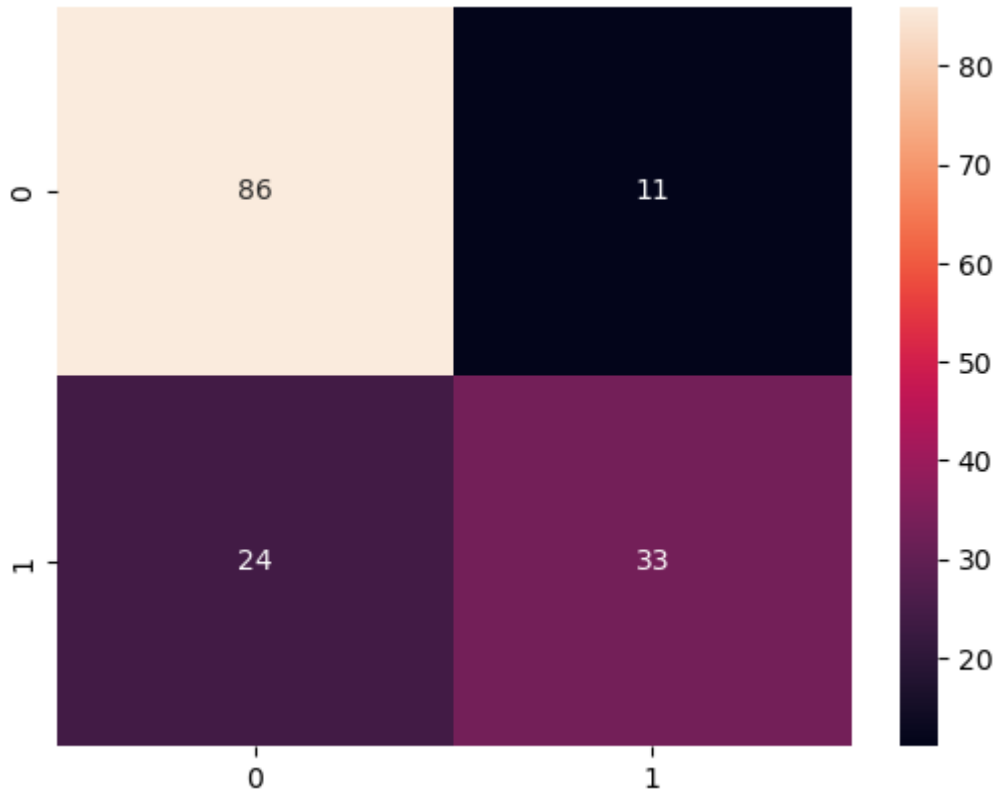
```
from sklearn.metrics import classification_report, confusion_matrix

cm = confusion_matrix(y_test, lr_pred)
cm
```

Output:

```
array([[86, 11],
       [24, 33]])
```

```
sns.heatmap(confusion_matrix(y_test, lr_pred), annot=True, fmt="d")
```

Output: `<Axes: >`



```
TN =cm[0, 0]
FP =cm[0,1]
FN = cm[1,0]
TP  = cm[1,1]


TN, FP, FN, TP
```

Output:

```
(86, 11, 24, 33)
```

```
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score, roc_auc_score, roc_curve
cm = confusion_matrix(y_test, lr_pred)

print('TN - True Negative {}'.format(cm[0,0]))
print('FP - False Positive {}'.format(cm[0,1]))
print('FN - False Negative {}'.format(cm[1,0]))
print('TP - True Positive {}'.format(cm[1,1]))
print('Accuracy Rate: {}'.format(np.divide(np.sum([cm[0,0], cm[1,1]]),
np.sum(cm))*100))
print('Misclassification Rate: {}'.format(np.divide(np.sum([cm[0,1], cm
[1,0]]), np.sum(cm))*100))
```

Output:

```
TN - True Negative 86
FP - False Positive 11
FN - False Negative 24
TP - True Positive 33
Accuracy Rate: 77.27272727272727
Misclassification Rate: 22.727272727272727
```

```
77.27272727272727+22.727272727272727
```

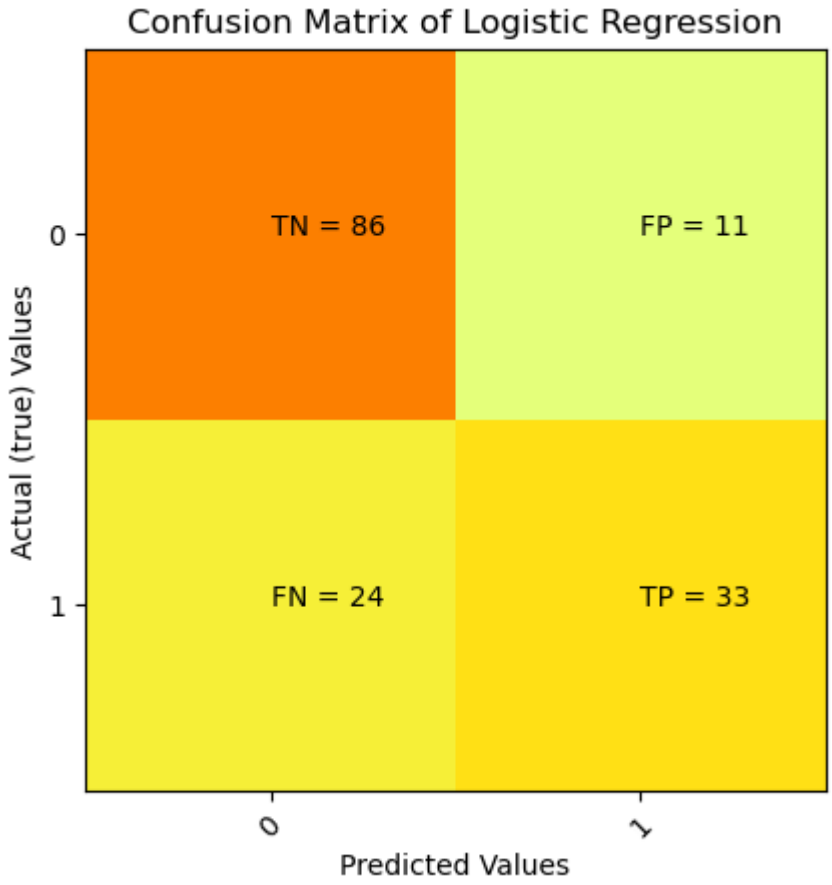
Output:

```
100.0
```

```
import matplotlib.pyplot as plt
import numpy as np

plt.clf()
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Wistia)
classNames = ['0', '1']
plt.title('Confusion Matrix of Logistic Regression')
plt.ylabel('Actual (true) Values')
plt.xlabel('Predicted Values')
tick_marks = np.arange(len(classNames))
plt.xticks(tick_marks, classNames, rotation=45)
plt.yticks(tick_marks, classNames)
```

```
s = [['TN', 'FP'], ['FN', 'TP']]
for i in range(2):
    for j in range(2):
        plt.text(j, i, str(s[i][j]) + " = " + str(cm[i][j]))

plt.show()
```

Output:




```
pd.crosstab(y_test, lr_pred, margins=False)
```

Output:

col_0	0	1
Outcome		
0	86	11
1	24	33

```
pd.crosstab(y_test, lr_pred, margins=True)
```

Output:

col_0	0	1	All
Outcome			
0	86	11	97
1	24	33	57
All	110	44	154

```
pd.crosstab(y_test, lr_pred, rownames=['Actual values'], colnames=['Predicted values'], margins=True)
```

Output:

Predicted values	0	1	All
Actual values			
0	86	11	97
1	24	33	57
All	110	44	154

Precision:

PPV- positive Predictive Value

Precision = True Positive/True Positive + False Positive
Precision = TP/TP+FP

```
TP, FP
```

Output:

```
(33, 11)
```

```
Precision = TP/(TP+FP)
Precision
```

Output:

```
0.75
```

```
33/(33+11)
```

Output:

```
0.75
```

precision Score:

```
precision_score = TP/float(TP+FP)*100
```

```
print('Precision Score: {0:0.4f}'.format(precision_score))
```

Output:

```
Precision Score: 75.0000
```

```
from sklearn.metrics import precision_score
print("Precision Score is: ", precision_score(y_test, lr_pred)*100)
print("Micro Average Precision Score is: ", precision_score(y_test, lr_
pred, average='micro')*100)
print("Macro Average Precision Score is: ", precision_score(y_test, lr_
pred, average='macro')*100)
print("Weighted Average Precision Score is: ", precision_score(y_test,
lr_pred, average='weighted')*100)
print("precision Score on Non Weighted score is: ", precision_score(y_t
est, lr_pred, average=None)*100)
```

Output:

```
Precision Score is:  75.0
Micro Average Precision Score is:  77.27272727272727
Macro Average Precision Score is:  76.5909090909091
Weighted Average Precision Score is:  77.00413223140497
precision Score on Non Weighted score is:  [78.18181818 75
```

```
print('Classification Report of Logistic Regression: \n', classificatio
n_report(y_test, lr_pred, digits=4))
```

Output:

```
Classification Report of Logistic Regression:
              precision    recall  f1-score   support

           0     0.7818    0.8866    0.8309        97
           1     0.7500    0.5789    0.6535        57

    accuracy                         0.7727       154
   macro avg     0.7659    0.7328    0.7422       154
weighted avg     0.7700    0.7727    0.7652       154
```

Recall:

True Positive Rate(TPR)

Recall = True Positive/True Positive + False Negative
Recall = TP/TP+FN

```
recall_score = TP/ float(TP+FN)*100
print('recall_score', recall_score)
```

Output:

```
recall_score 57.89473684210527
```

```
TP, FN
```

Output:

```
(33, 24)
```

```
33/(33+24)
```

Output:

```
0.5789473684210527
```

```
from sklearn.metrics import recall_score
print('Recall or Sensitivity_Score: ', recall_score(y_test, lr_pred)*100)
```

Output:

```
Recall or Sensitivity_Score:  57.89473684210527
```

```
print("recall Score is: ", recall_score(y_test, lr_pred)*100)
print("Micro Average recall Score is: ", recall_score(y_test, lr_pred,
average='micro')*100)
print("Macro Average recall Score is: ", recall_score(y_test, lr_pred,
average='macro')*100)
print("Weighted Average recall Score is: ", recall_score(y_test, lr_pre
d, average='weighted')*100)
print("recall Score on Non Weighted score is: ", recall_score(y_test, l
r_pred, average=None)*100)
```

Output:

```
recall Score is:  57.89473684210527
Micro Average recall Score is:  77.27272727272727
Macro Average recall Score is:  73.27726532826912
Weighted Average recall Score is:  77.27272727272727
recall Score on Non Weighted score is:  [88.65979381 57.89473684]
```

```
print('Classification Report of Logistic Regression: \n', classificatio
n_report(y_test, lr_pred, digits=4))
```

Output:

```
  Classification Report of Logistic Regression:
                precision    recall  f1-score   support

             0     0.7818    0.8866    0.8309        97
             1     0.7500    0.5789    0.6535        57

      accuracy                         0.7727       154
     macro avg     0.7659    0.7328    0.7422       154
  weighted avg     0.7700    0.7727    0.7652       154
```

FPR - False Positve Rate

```
FPR = FP / float(FP + TN) * 100
print('False Positive Rate: {:.4f}'.format(FPR))
```

```
False Positive Rate: 11.3402
```

```
FP, TN
```

Output:

```
(11, 86)
```

```
11/(11+86)
```

Output:

```
0.1134020618556701
```

Specificity:

```
specificity = TN /(TN+FP)*100
print('Specificity : {0:0.4f}'.format(specificity))
```

Output:

```
Specificity : 88.6598
```

```
from sklearn.metrics import f1_score
print('F1_Score of Macro: ', f1_score(y_test, lr_pred)*100)
```

```
F1_Score of Macro:  65.34653465346535
```

```
print("Micro Average f1 Score is: ", f1_score(y_test, lr_pred, average='mic
ro')*100)
print("Macro Average f1 Score is: ", f1_score(y_test, lr_pred, average='mac
ro')*100)
print("Weighted Average f1 Score is: ", f1_score(y_test, lr_pred, average='
weighted')*100)
print("f1 Score on Non Weighted score is: ", f1_score(y_test, lr_pred, aver
age=None)*100)
```

Output:

```
Micro Average f1 Score is:  77.27272727272727
Macro Average f1 Score is:  74.21916104653944
Weighted Average f1 Score is:  76.52373933045479
f1 Score on Non Weighted score is:  [83.09178744 65.34653465]
```

Classification Report of Logistic Regression:

```
from sklearn.metrics import classification_report
print('Classification Report of Logistic Regression: \n', classification_re
port(y_test, lr_pred, digits=4))
```

Output:

```
Classification Report of Logistic Regression:
              precision    recall  f1-score   support

           0     0.7818    0.8866    0.8309        97
           1     0.7500    0.5789    0.6535        57

    accuracy                         0.7727       154
   macro avg     0.7659    0.7328    0.7422       154
weighted avg     0.7700    0.7727    0.7652       154
```

ROC Curve& ROC AUC:

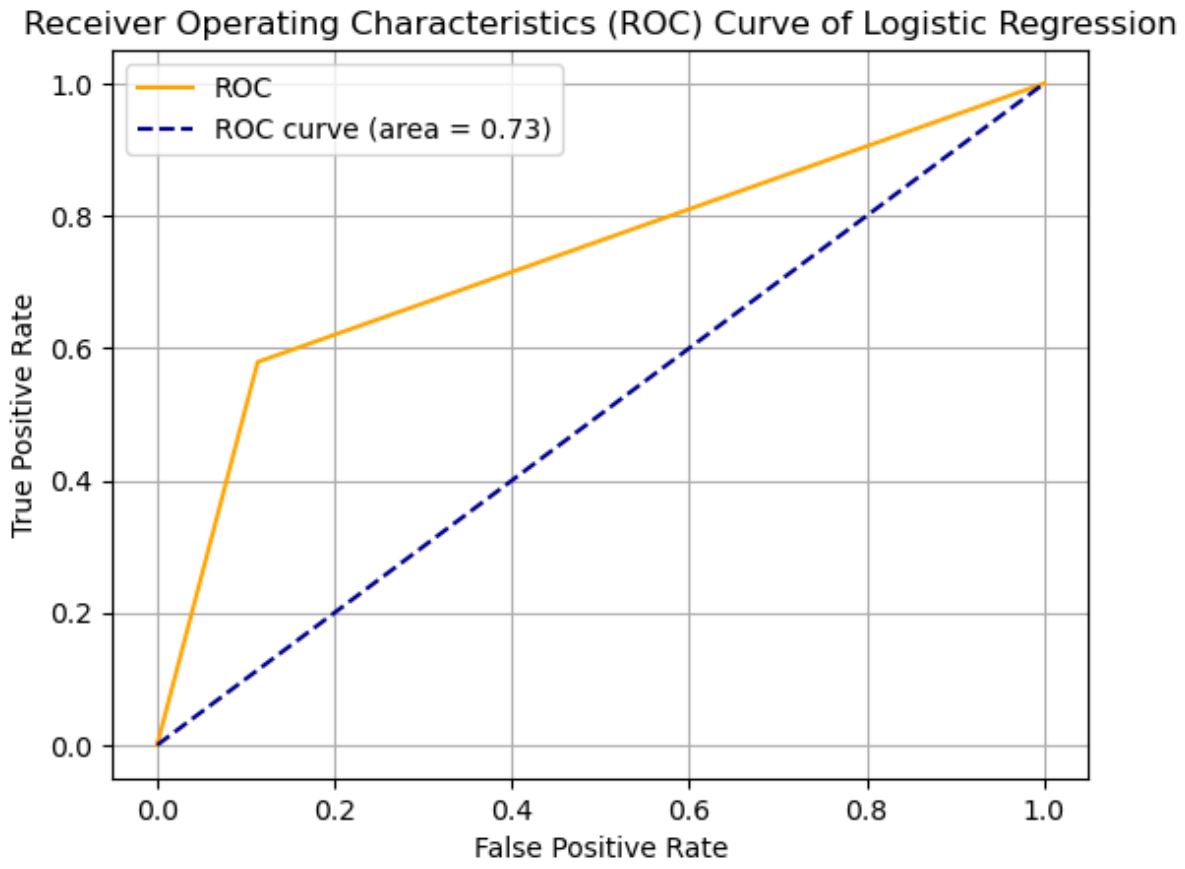
```
# Area under Curve:
auc= roc_auc_score(y_test, lr_pred)
print("ROC AUC SCORE of logistic Regression is ", auc)
```

Output:

```
ROC AUC SCORE of logistic Regression is  0.7327726532826913
```

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

fpr, tpr, thresholds = roc_curve(y_test, lr_pred)
plt.plot(fpr, tpr, color='orange', label="ROC")
plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--', label='ROC c
urve (area = %0.2f)' % auc(fpr, tpr))
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver Operating Characteristics (ROC) Curve of Logistic R
egression")
plt.legend()
plt.grid()
plt.show()
```

Output:



Confusion Matrix:

- Confusion matrix of "Decision Tree"

```
from sklearn.metrics import classification_report, confusion_matrix
cm = confusion_matrix(y_test, dt_pred)
cm
```

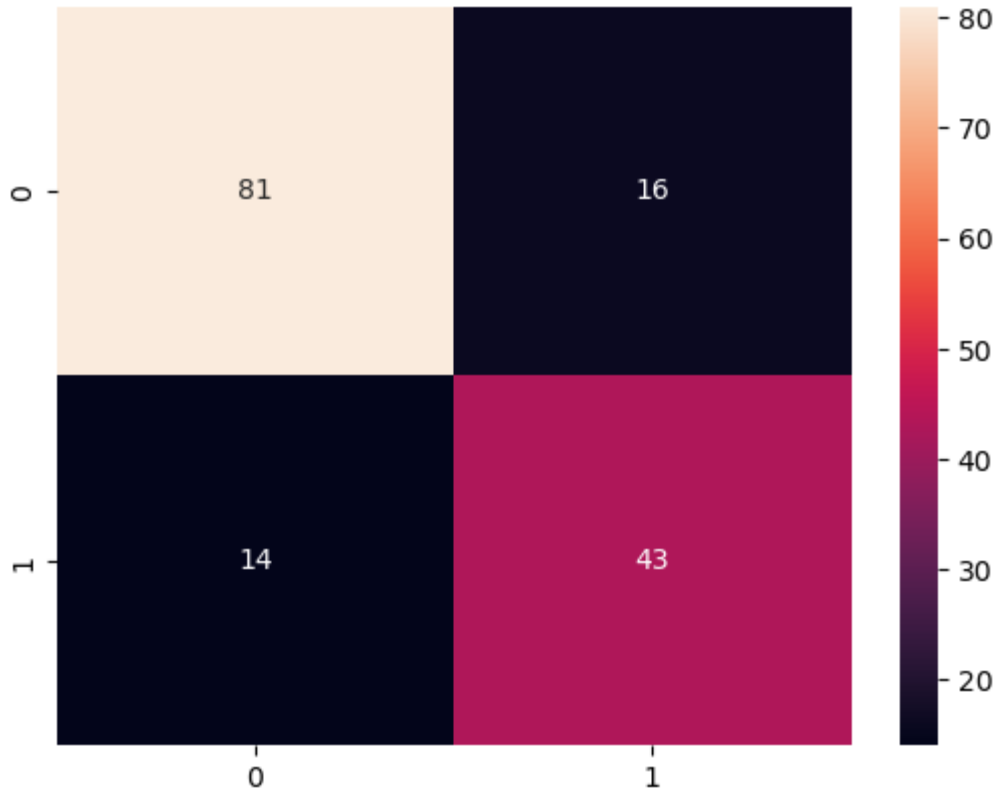
Output:

```
array([[81, 16],
       [14, 43]])
```

```
ns.heatmap(confusion_matrix(y_test, dt_pred), annot=True, fmt="d")
```

Output:

```
<Axes: >
```




```
TN =cm[0, 0]
FP =cm[0,1]
FN = cm[1,0]
TP  = cm[1,1]

TN, FP, FN, TP
```

Output:

```
(81, 16, 14, 43)
```

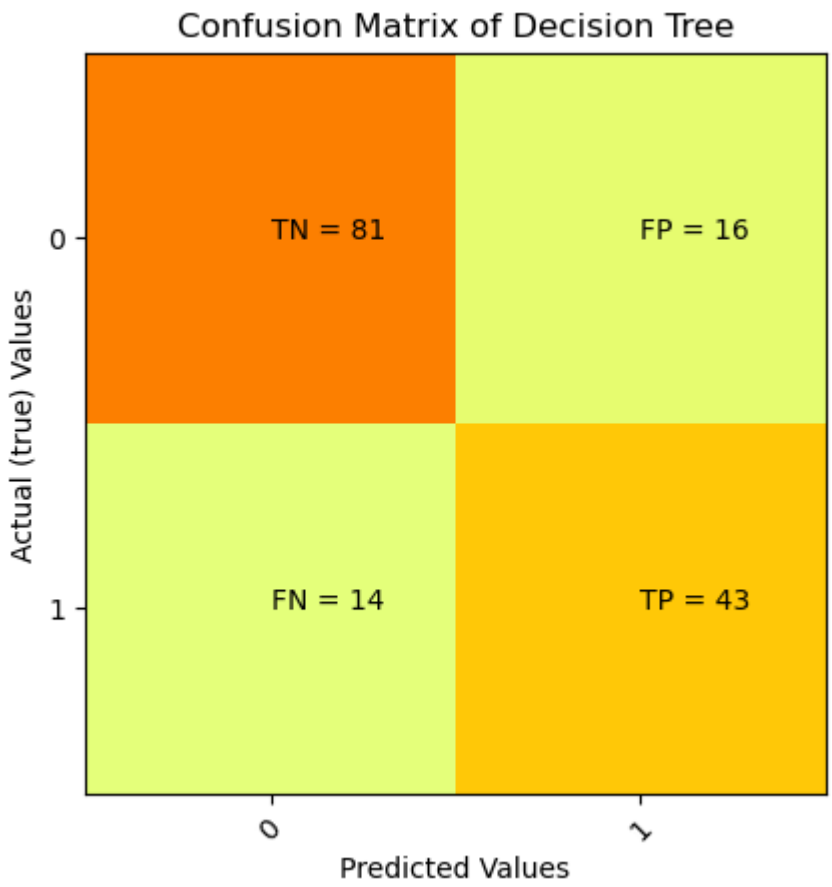
```
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score, roc_auc_score, roc_curve
cm = confusion_matrix(y_test, dt_pred)

print('TN - True Negative {}'.format(cm[0,0]))
print('FP - False Positive {}'.format(cm[0,1]))
print('FN - False Negative {}'.format(cm[1,0]))
print('TP - True Positive {}'.format(cm[1,1]))
print('Accuracy Rate: {}'.format(np.divide(np.sum([cm[0,0], cm[1,1]]), np.s
um(cm))*100))
print('Misclassification Rate: {}'.format(np.divide(np.sum([cm[0,1], cm[1,0
]]), np.sum(cm))*100))
```

Output:

```
TN - True Negative 81
FP - False Positive 16
FN - False Negative 14
TP - True Positive 43
Accuracy Rate: 80.51948051948052
Misclassification Rate: 19.480519480519483
```

```
import matplotlib.pyplot as plt
import numpy as np

plt.clf()
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Wistia)
classNames = ['0', '1']
plt.title('Confusion Matrix of Decision Tree')
plt.ylabel('Actual (true) Values')
plt.xlabel('Predicted Values')
tick_marks = np.arange(len(classNames))
plt.xticks(tick_marks, classNames, rotation=45)
plt.yticks(tick_marks, classNames)
s = [['TN', 'FP'], ['FN', 'TP']]
for i in range(2):
    for j in range(2):
        plt.text(j, i, str(s[i][j]) + " = " + str(cm[i][j]))

plt.show()
```

Output:



Precision:

```
# precision Score:

precision_score = TP/float(TP+FP)*100
print('Precision Score: {0:0.4f}'.format(precision_score))
```

Output:

```
Precision Score: 72.8814
```

```
from sklearn.metrics import precision_score

print("Precision Score is:", precision_score(y_test, dt_pred) * 100)
print("Micro Average Precision Score is:", precision_score(y_test, dt_p
red, average='micro') * 100)
print("Macro Average Precision Score is:", precision_score(y_test, dt_p
red, average='macro') * 100)
print("Weighted Average Precision Score is:", precision_score(y_test, d
t_pred, average='weighted') * 100)
```

```
print("Precision Score on Non Weighted score is:", precision_score(y_te
st, dt_pred, average=None) * 100)
```

Output:

```
Precision Score is: 72.88135593220339
Micro Average Precision Score is: 80.51948051948052
Macro Average Precision Score is: 79.07225691347011
Weighted Average Precision Score is: 80.68028314237056
Precision Score on Non Weighted score is: [85.26315789 72.88135593]
```

Recall:

```
recall_score = TP/ float(TP+FN)*100
print('recall_score', recall_score)
```

Output:

```
recall_score 75.43859649122807
```

```
from sklearn.metrics import recall_score
print('Recall or Sensitivity_Score: ', recall_score(y_test, dt_pred)*100)
```

Output:

```
Recall or Sensitivity_Score:  75.43859649122807
```

```
print("recall Score is: ", recall_score(y_test, dt_pred)*100)
print("Micro Average recall Score is: ", recall_score(y_test, dt_pred, aver
age='micro')*100)
print("Macro Average recall Score is: ", recall_score(y_test, dt_pred, aver
age='macro')*100)
print("Weighted Average recall Score is: ", recall_score(y_test, dt_pred, a
verage='weighted')*100)
print("recall Score on Non Weighted score is: ", recall_score(y_test, dt_pr
ed, average=None)*100)
```

Output:

```
recall Score is:  75.43859649122807
Micro Average recall Score is:  80.51948051948052
Macro Average recall Score is:  79.47187556520167
Weighted Average recall Score is:  80.51948051948052
recall Score on Non Weighted score is:  [83.50515464 75.43859649]
```

FPR

```
FPR = FP / float(FP + TN) * 100
print('False Positive Rate: {:.4f}'.format(FPR))
```

Output:

```
False Positive Rate: 16.4948
```

Specificity:

```
specificity = TN /(TN+FP)*100
print('Specificity : {0:0.4f}'.format(specificity))
```

Output:

```
Specificity : 83.5052
```

```
from sklearn.metrics import f1_score
print('F1_Score of Macro: ', f1_score(y_test, dt_pred)*100)
```

Output:

```
F1_Score of Macro:  74.13793103448276
```

```
print("Micro Average f1 Score is: ", f1_score(y_test, dt_pred, average='mic
ro')*100)
print("Macro Average f1 Score is: ", f1_score(y_test, dt_pred, average='mac
ro')*100)
print("Weighted Average f1 Score is: ", f1_score(y_test, dt_pred, average='
weighted')*100)
print("f1 Score on Non Weighted score is: ", f1_score(y_test, dt_pred, aver
age=None)*100)
```

Output:

```
Micro Average f1 Score is:  80.51948051948051
Macro Average f1 Score is:  79.25646551724138
Weighted Average f1 Score is:  80.58595499328258
f1 Score on Non Weighted score is:  [84.375      74.13793103]
```

Classification Report of Decision Tree:

```
from sklearn.metrics import classification_report
print('Classification Report of Decision Tree: \n', classification_report(y
_test, dt_pred, digits=4))
```

Output:

```
Classification Report of Decision Tree:
              precision    recall  f1-score   support

           0     0.8526    0.8351    0.8438        97
           1     0.7288    0.7544    0.7414        57

    accuracy                         0.8052       154
   macro avg     0.7907    0.7947    0.7926       154
weighted avg     0.8068    0.8052    0.8059       154
```

ROC Curve& ROC AUC:

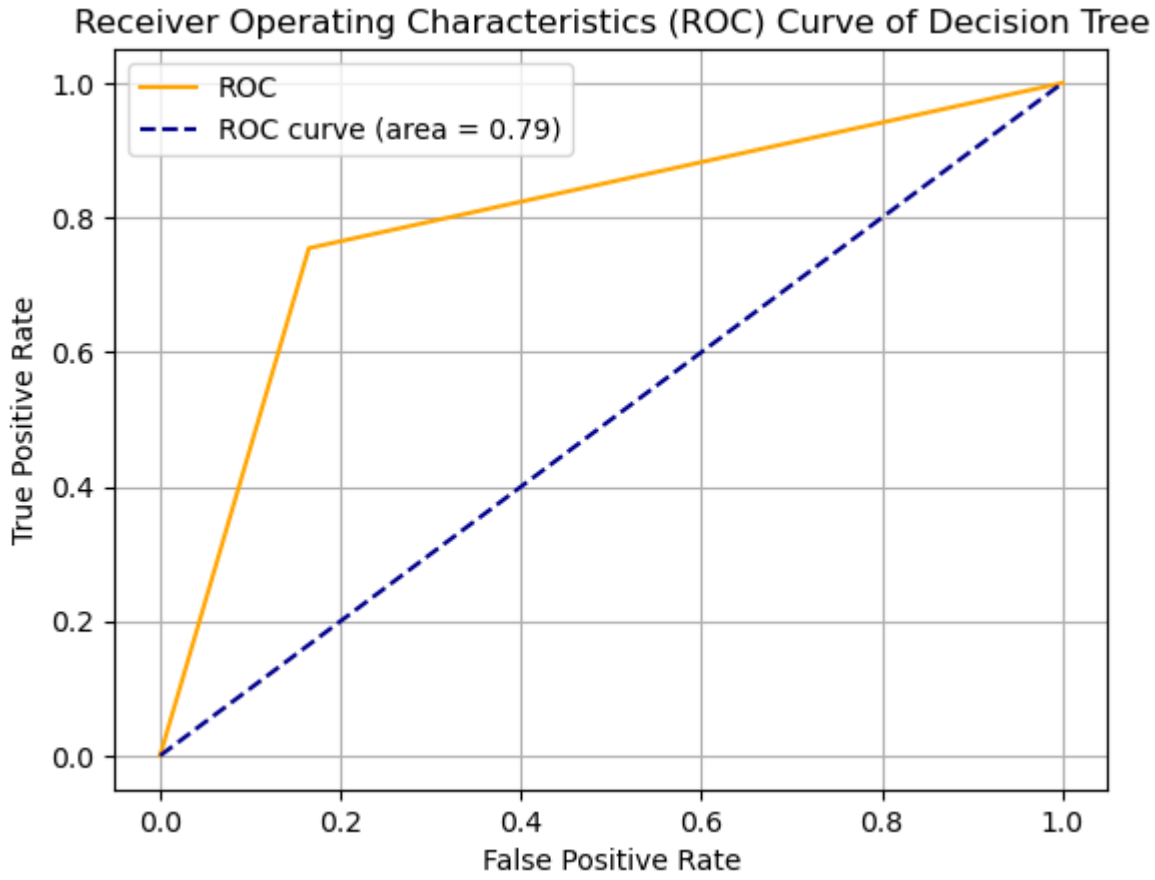
```
# Area under Curve:
auc= roc_auc_score(y_test, dt_pred)
print("ROC AUC SCORE of Decision Treeis ", auc)
```

Output:

```
ROC AUC SCORE of Decision Treeis  0.7947187556520168
```

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

fpr, tpr, thresholds = roc_curve(y_test, dt_pred)
plt.plot(fpr, tpr, color='orange', label="ROC")
plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--', label='ROC c
urve (area = %0.2f)' % auc(fpr, tpr))
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver Operating Characteristics (ROC) Curve of Decision T
ree")
plt.legend()
plt.grid()
plt.show()
```

Output:


Receiver Operating Characteristics (ROC) Curve of Decision Tree
ROC
ROC curve (area = 0.79)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0