AI BASED DIABETES PREDICTION SYSTEM

PHASE-4

Data collection and Preprocessing:

Gather a dataset containing relevant information about individuals, including features such as age, preganacies,BMI,insulin,blood pressure, and glucose levels. Datasets like the Diabetes Database can be useful.

Data splitting:

Split the dataset into training and testing sets to evaluate your model's performance.

Model Training:

Train the selected model on the training data using appropriate algorithms.

Model selection:

Choose an appropriate machine learning or deep learning model for diabetes prediction. Common models include logistic regression, decision trees or random forests.

Evaluation Performance:

Evaluate the model's evaluation in given diabetes database using metrics like accuracy, precision, recall, and F1-score. Make use of cross-validation to ensure robustness.

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.

Dataset:

This is above Diabetes.csv data

Import Required Libraries:

[ ]:
```
import numpy as np

import pandas as pd
```

[ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
```

[ ]:
```
df=pd.read_csv('/User/PS/diabetes.csv')
```

Classification Algorithms:

Logistic Regression:

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

[ ]:
```
LogisticRegression(multi_class='ovr', solver='liblinear')
```

Descision Tree:

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

[ ]:
```
DecisionTreeClassifier()
```

Making prediction:

Logistic Regression:

[ ]:
```
X_test.shape
```

[ ]:
```
(154, 8)
```

[ ]:
```
lr_pred=lr.predict(X_test)
```

[ ]:
```
lr_pred.shape
```

[ ]:
```
(154,)
```

Decision Tree:

[ ]:
```
dt_pred=dt.predict(X_test)
```

[ ]:
```
dt_pred.shape
```

[ ]:
```
(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)
```

```
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)
```

```
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"*

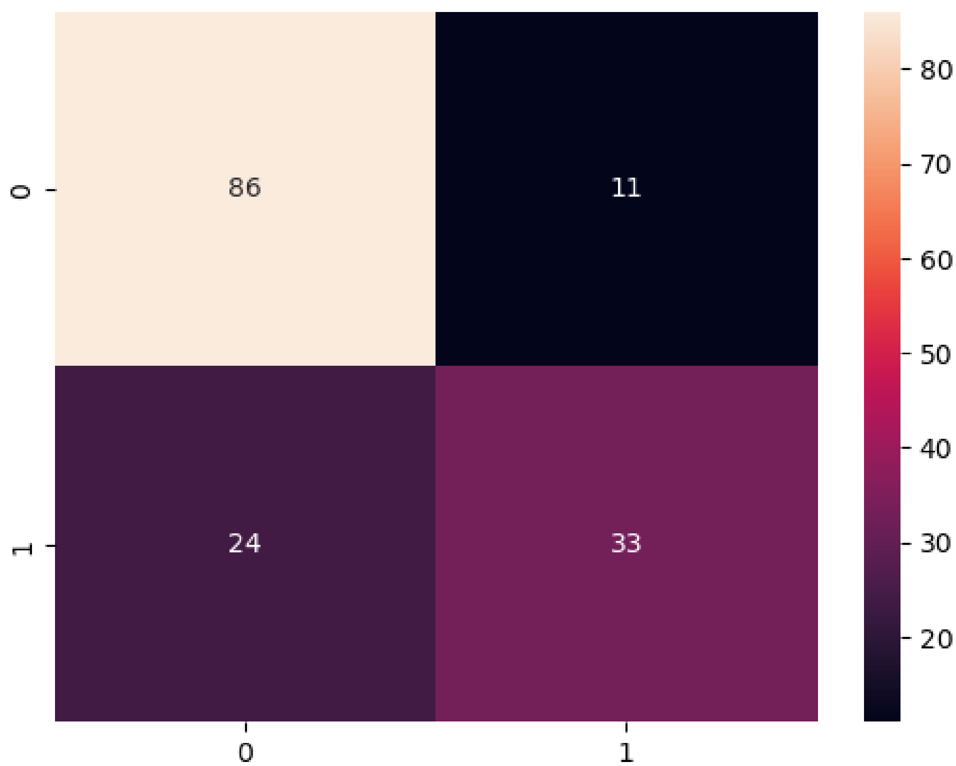
```
[ ]: from sklearn.metrics import classification_report, confusion_matrix

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

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

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

```
[ ]: <Axes: >
```



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

[ ]:
```
TN, FP, FN, TP
```

[ ]:
```
(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))
```

```
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
```

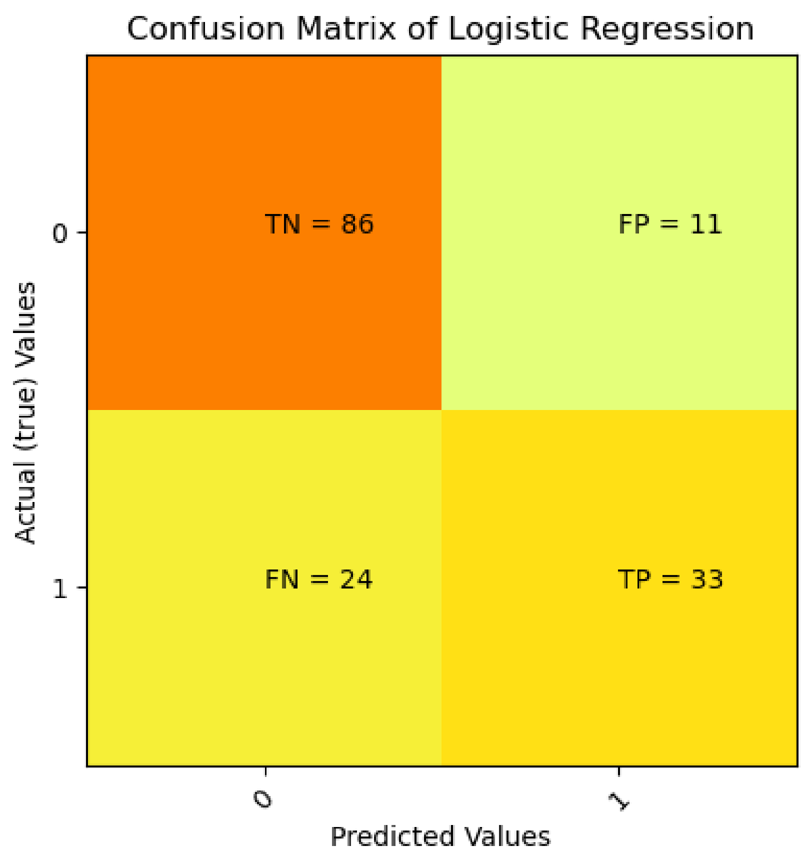
[ ]:
```
100.0
```

[ ]:
```
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()
```



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

[ ]:
```
col_0     0   1
Outcome
0        86  11
1        24  33
```

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

[ ]:
```
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)
```

[ ]:
```
Predicted values    0   1  All
Actual values
0                  86  11   97
1                  24  33   57
All               110  44  154
```

5.0.1 Precision:

PPV- positive Predictive Value

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

[ ]:
```
TP, FP
```

[ ]:
```
(33, 11)
```

[ ]:
```
Precision = TP/(TP+FP)
Precision
```

[ ]:
```
0.75
```

[ ]:
```
33/(33+11)
```

[ ]:
```
0.75
```

[ ]:
```
# precision Score:

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

```
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_test,␣
↪lr_pred, average=None)*100)
```

```
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',␣
↪classification_report(y_test, lr_pred, digits=4))
```

```
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)
```

```
recall_score 57.89473684210527
```

[ ]:
```
TP, FN
```

[ ]: (33, 24)

[ ]:
```
33/(33+24)
```

[ ]: 0.5789473684210527

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

```
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_pred,␣
 ↪average='weighted')*100)
print("recall Score on Non Weighted score is: ", recall_score(y_test, lr_pred,␣
 ↪average=None)*100)
```

```
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',␣
 ↪classification_report(y_test, lr_pred, digits=4))
```

```
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
```

[ ]: (11, 86)

[ ]:
```
11/(11+86)
```

[ ]: 0.1134020618556701

5.2 Specificity:

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

```
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='micro')*100)
     print("Macro Average f1 Score is: ", f1_score(y_test, lr_pred,␣
      ↪average='macro')*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,␣
      ↪average=None)*100)
```

```
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_report(y_test, lr_pred, digits=4))
```

```
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

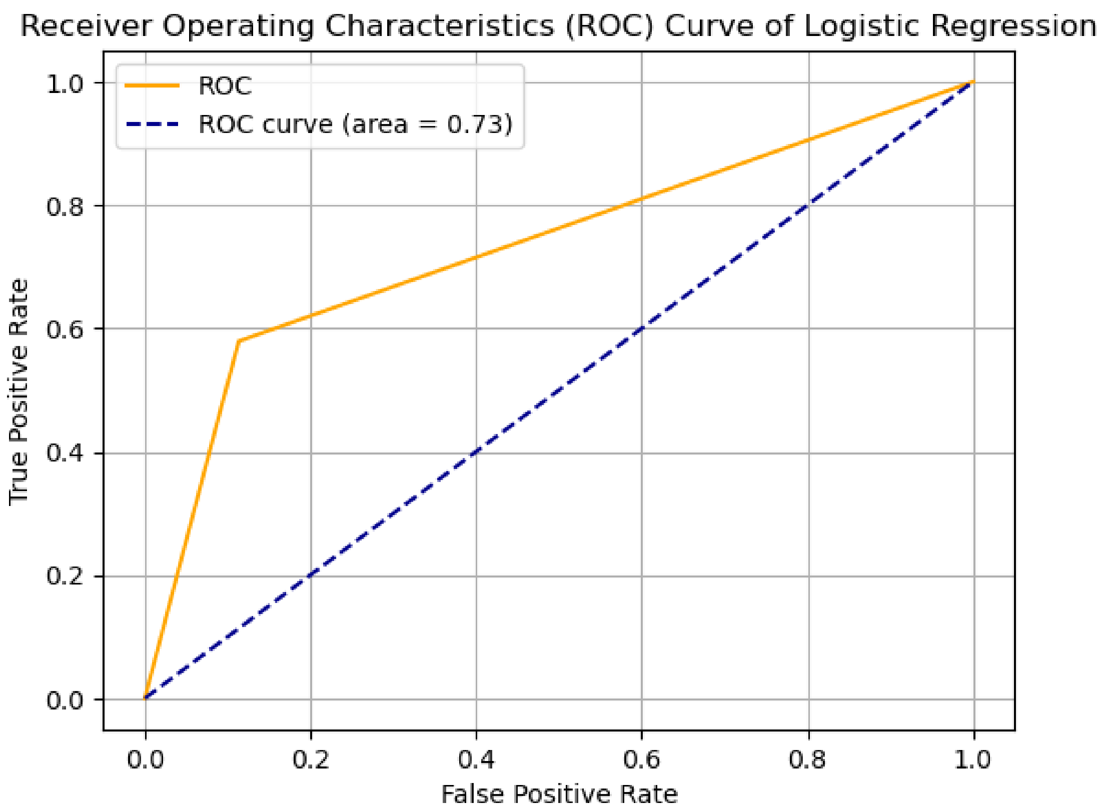
```
[ ]:
     auc= roc_auc_score(y_test, lr_pred)
     print("ROC AUC SCORE of logistic Regression is ", auc)
```

```
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 curve␣
  ↪(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␣
  ↪Regression")
plt.legend()
plt.grid()
plt.show()
```



Confusion Matrix:

- Confusion matrix of "Decision Tree"

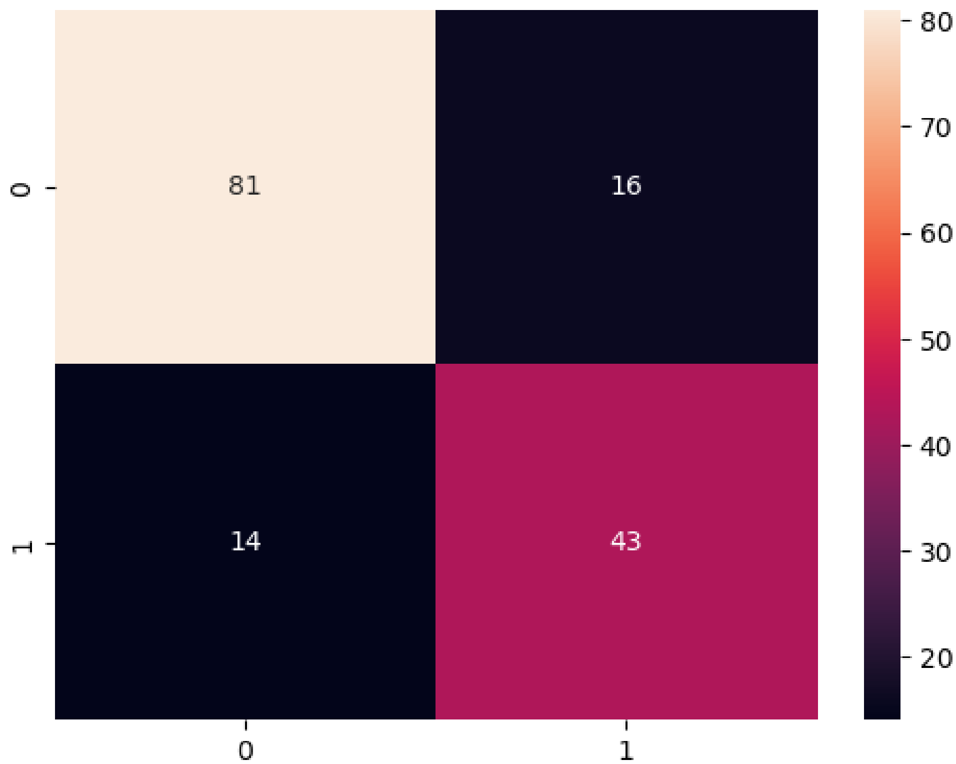
[ ]:
```
from sklearn.metrics import classification_report, confusion_matrix

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

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

```
[ ]: sns.heatmap(confusion_matrix(y_test, dt_pred), annot=True, fmt="d")
```

[ ]: <Axes: >




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

```
[ ]: TN, FP, FN, TP
```

[ ]: (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]))
```

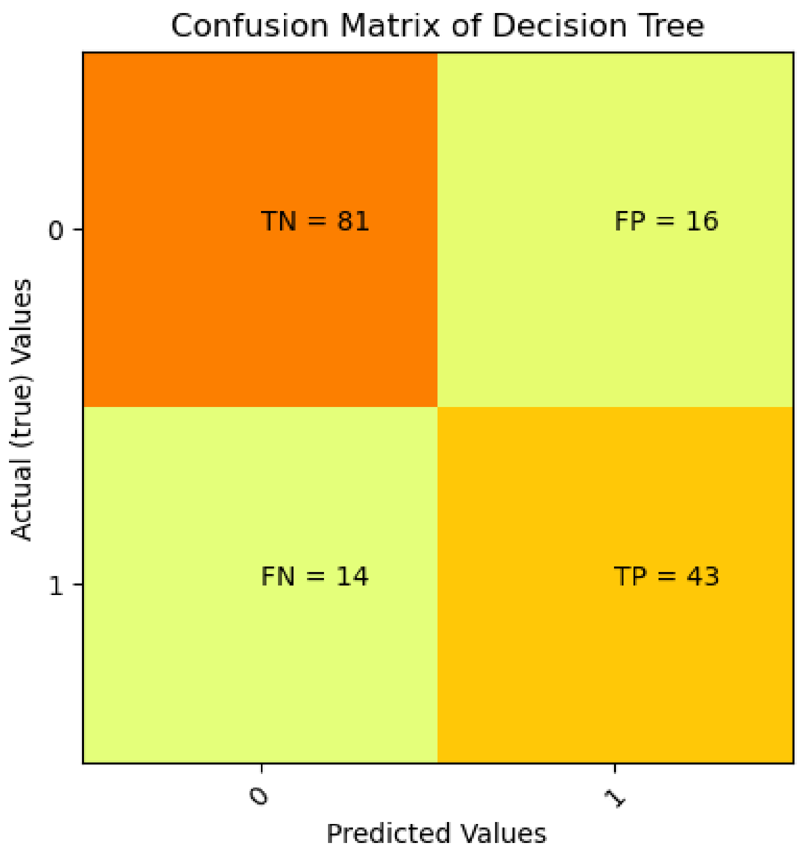
```
print('FN - False Negative {}'.format(cm[1,0]))
print('TP - True Positive {}'.format(cm[1,1]))
print('Accuracy Rate: {}'.format(np.divide(np.sum([cm[0,0], cm[1,1]]), np.
 ↪sum(cm))*100))
print('Misclassification Rate: {}'.format(np.divide(np.sum([cm[0,1], cm[1,0]]),␣
 ↪np.sum(cm))*100))
```

```
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()
```



Precision:

[ ]:

Precision score:

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

```
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_pred,␣
 ↪average='micro') * 100)
print("Macro Average Precision Score is:", precision_score(y_test, dt_pred,␣
 ↪average='macro') * 100)
print("Weighted Average Precision Score is:", precision_score(y_test, dt_pred,␣
 ↪average='weighted') * 100)
```

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

```
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)
```

```
recall_score 75.43859649122807
```

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

```
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,␣
↪average='micro')*100)
print("Macro Average recall Score is: ", recall_score(y_test, dt_pred,␣
↪average='macro')*100)
print("Weighted Average recall Score is: ", recall_score(y_test, dt_pred,␣
↪average='weighted')*100)
print("recall Score on Non Weighted score is: ", recall_score(y_test, dt_pred,␣
↪average=None)*100)
```

```
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))
```

```
False Positive Rate: 16.4948
```

Specificity:

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

```
Specificity : 83.5052
```

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

```
F1_Score of Macro:  74.13793103448276
```

```
[ ]: print("Micro Average f1 Score is: ", f1_score(y_test, dt_pred,␣
      ↪average='micro')*100)
     print("Macro Average f1 Score is: ", f1_score(y_test, dt_pred,␣
      ↪average='macro')*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,␣
      ↪average=None)*100)
```

```
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))
```

```
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

[ ]:
```python
auc= roc_auc_score(y_test, dt_pred)
print("ROC AUC SCORE of Decision Treeis ", auc)
```

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

[ ]:
```python
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 curve␣
↪(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 Tree")
plt.legend()
plt.grid()
plt.show()
```