ARTIFICIAL INTELLIGENCE

GROUP 5

PHASE 5

PROJECT

DOCUMENTATION

PROJECT – 2

AI – BASED DIABETES

PREDICTION SYSTEM

ARTIFICIAL INTELLIGENCE – GROUP 5
Project 2: AI Based Diabetes Prediction System

Phase 1: Problem Definition and Design Thinking

Problem Definition:

The problem is to build an AI-powered diabetes prediction system that uses machine learning algorithms to analyze medical data and predict the likelihood of an individual developing diabetes. The system aims to provide early risk assessment and personalized preventive measures, allowing individuals to take proactive actions to manage their health.

Design Thinking:

1. **Data Collection:** We need a dataset containing medical features such as glucose levels, blood pressure, BMI, etc., along with information about whether the individual has diabetes or not.
2. **Data Preprocessing:** The medical data needs to be cleaned, normalized, and prepared for training machine learning models.
3. **Feature Selection:** We will select relevant features that can impact diabetes risk prediction.
4. **Model Selection:** We can experiment with various machine learning algorithms like Logistic Regression, Random Forest, and Gradient Boosting.
5. **Evaluation:** We will evaluate the model's performance using metrics like accuracy, precision, recall, F1-score, and ROC-AUC.

6. **Iterative Improvement:** We will fine-tune the model parameters and explore techniques like feature engineering to enhance prediction accuracy.

IBM – NAAN MUDHALVAN ➔ARTIFICIAL INTELLIGENCE

Project 2 : AI Based Diabetes Prediction System

Phase – 2 – INNOVATION

Stage 1: Empathize-Research Your User's Needs:

Empathize: The first phase of design thinking, where you gain real insight into users and their needs.

➔ The first stage of the design thinking process focuses on user-centric research. You want to gain an empathic understanding of the problem you are trying to solve. Consult experts to find out more about the area of concern and conduct observations to engage and empathize with your users. You may also want to immerse yourself in your users' physical environment to gain a deeper, personal understanding of the issues involved as well as their experiences and motivations. Empathy is crucial to problem solving and a human-centered design process as it allows design thinkers to set aside their own assumptions about the world and gain real insight into users and their needs.

➔Depending on time constraints, you will gather a substantial amount of information to use during the next stage. The main aim of the Empathize stage is to develop the best possible understanding of your users, their needs and the problems that underlie the development of the product or service you want to create.

Stage 2: Define-State Your Users' Needs and Problems:

Define: The second phase of design thinking, where you define the problem statement in a human-centered manner.

➔ In the Define stage, you will organize the information you have gathered during the Empathize stage. You'll analyze your observations to define the core problems you and your team have identified up to this point. Defining the problem and problem statement must be done in a human-centered manner. For example, you should not define the problem as your own wish or need of the company: We need to increase our food-product market share among young teenage girls by 5%.You should pitch the problem statement from your perception of the users' needs: "Teenage girls need to eat nutritious food in order to thrive, be healthy and grow".

➔ The Define stage will help the design team collect great ideas to establish features, functions and other elements to solve the problem at hand or, at the very least, allow real users to resolve issues themselves with minimal difficulty. In this stage, you will start to progress to the third stage, the ideation phase, where you ask questions to help you look for solutions: "How might we encourage teenage girls to perform an action that benefits them and also involves your company's food-related product or service" for instance.

Stage 3: Ideate-Challenge Assumptions and Create Ideas:

Ideate: The third phase of design thinking, where you identify innovative solutions to the problem statement you've created.

➔ During the third stage of the design thinking process, designers are ready to generate ideas. You've grown to understand your users and their needs in the Empathize stage, and you've analyzed your observations in the Define stage to create a user centric problem statement. With this solid background, you and your team members can start to look at the problem from different perspectives and ideate innovative solutions to your problem statement.

➔ There are hundreds of ideation techniques you can use such as Brainstorm, Brain writes, Worst Possible Idea and SCAMPER. Brainstorm and Worst Possible Idea techniques are typically used at the start of the ideation stage to stimulate free thinking and expand the problem space. This allows you to generate as many ideas as possible at the start of ideation. You should pick other ideation techniques towards the end of this stage to help you investigate and test your ideas and choose the best ones to move forward with either because they seem to solve the problem or provide the elements required to circumvent it.

Stage 4: Prototype-Start to Create Solutions:

Prototype: The fourth phase of design thinking, where you identify the best possible solution.

➔ The design team will now produce a number of inexpensive, scaled down versions of the product (or specific features found within the product) to investigate the key solutions generated in the ideation phase. These prototypes can be shared and tested within the team itself, in other departments or with a small group of people outside the design team.

➔ This is an experimental phase, and the aim is to identify the best possible solution for each of the problems identified during the first three stages. The solutions are implemented within the prototypes and, one by one, they are investigated and then accepted, improved or rejected based on the users' experiences. By the end of the Prototype stage, the design team will have a better idea of the product's limitations and the problems it faces. They'll also have a clearer view of how real users would behave, think and feel when they interact with the end product.

Stage 5: Test-Try Your Solutions Out:

Test: The fifth and final phase of the design thinking process, where you test solutions to derive a deep understanding of the product and its users.

➔ Designers or evaluators rigorously test the complete product using the best solutions identified in the Prototype stage. This is the final stage of the five-stage model; however, in an iterative process such as design thinking, the results generated are often used to redefine one or more further problems.

➔ This increased level of understanding may help you investigate the conditions of use and how people think, behave and feel towards the product, and even lead you to loop back to a previous stage in the design thinking process. You can then proceed with further iterations and make alterations and refinements to rule out alternative solutions. The ultimate goal is to get as deep an understanding of the product and its users as possible.

IBM – NAAN MUDHALVAN➔ARTIFICIAL INTELLIGENCE

PHASE – 3

Project – 2: AI – Based Diabetes Prediction System

Content : Development Part -1

In this part you will begin building your project by loading and preprocessing the dataset.
In this phase begin developing the diabetes prediction system by preparing the data and selecting relevant features.

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

Import Required Libraries:

```
# Ignore warning messages to prevent them from being displayed during code
execution
import warnings
warnings.filterwarnings('ignore')


import numpy as np    # Importing the NumPy library for linear algebra
operations
import pandas as pd   # Importing the Pandas library for data processing and
 CSV file handling


import os
for dirname, _, filenames in os.walk('/diabetes.csv/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
(C:Users/Sakthi/Downloads/archive.zip/diabetes.csv)
```

```
import seaborn as sns                        # Importing the Seaborn library
                            for statistical data visualization
import matplotlib.pyplot as plt              # Importing the Matplotlib
library for creating plots and visualizations
import plotly.express as px                  # Importing the Plotly Express
library for interactive visualizations
```

Exploratory Data Analysis:

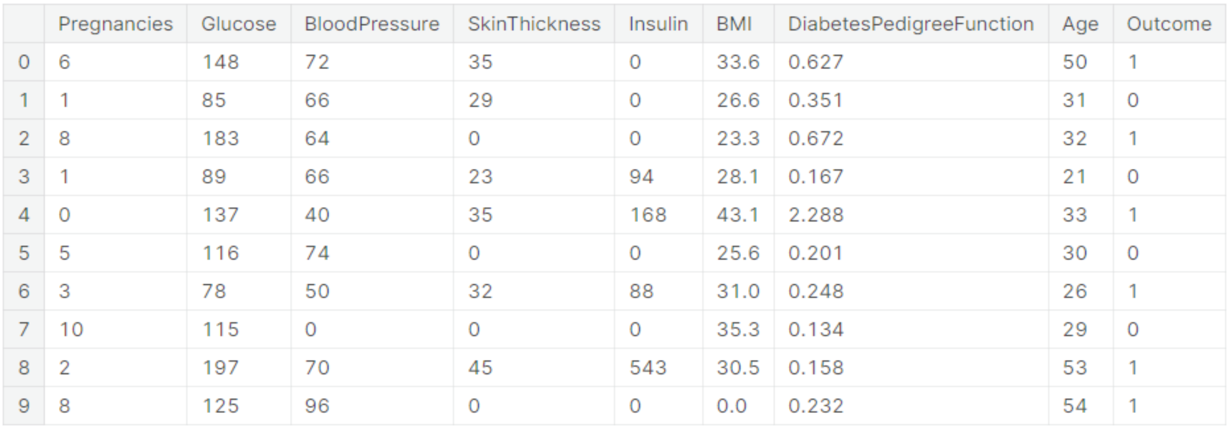
Load and Prepare Data:

```
df=pd.read_csv('C:Users/Sakthi/Downloads/archive.zip/diabetes.csv')
```

UnderStanding the Variables:

```
df.head(10)
```

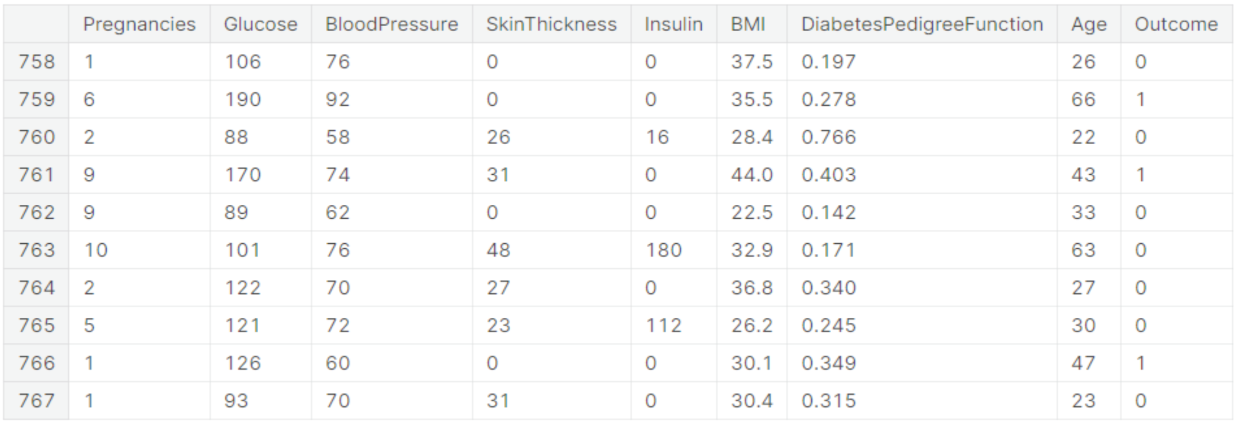
Output:



	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
5	5	116	74	0	0	25.6	0.201	30	0
6	3	78	50	32	88	31.0	0.248	26	1
7	10	115	0	0	0	35.3	0.134	29	0
8	2	197	70	45	543	30.5	0.158	53	1
9	8	125	96	0	0	0.0	0.232	54	1

```
df.tail(10)
```

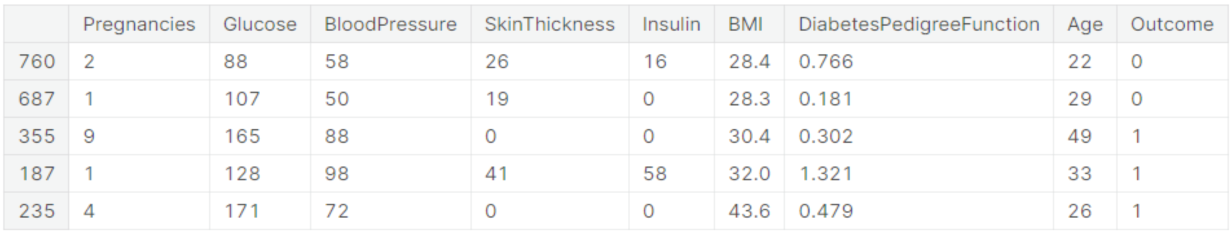
Output:



	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
758	1	106	76	0	0	37.5	0.197	26	0
759	6	190	92	0	0	35.5	0.278	66	1
760	2	88	58	26	16	28.4	0.766	22	0
761	9	170	74	31	0	44.0	0.403	43	1
762	9	89	62	0	0	22.5	0.142	33	0
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

```
df.sample(5)
```

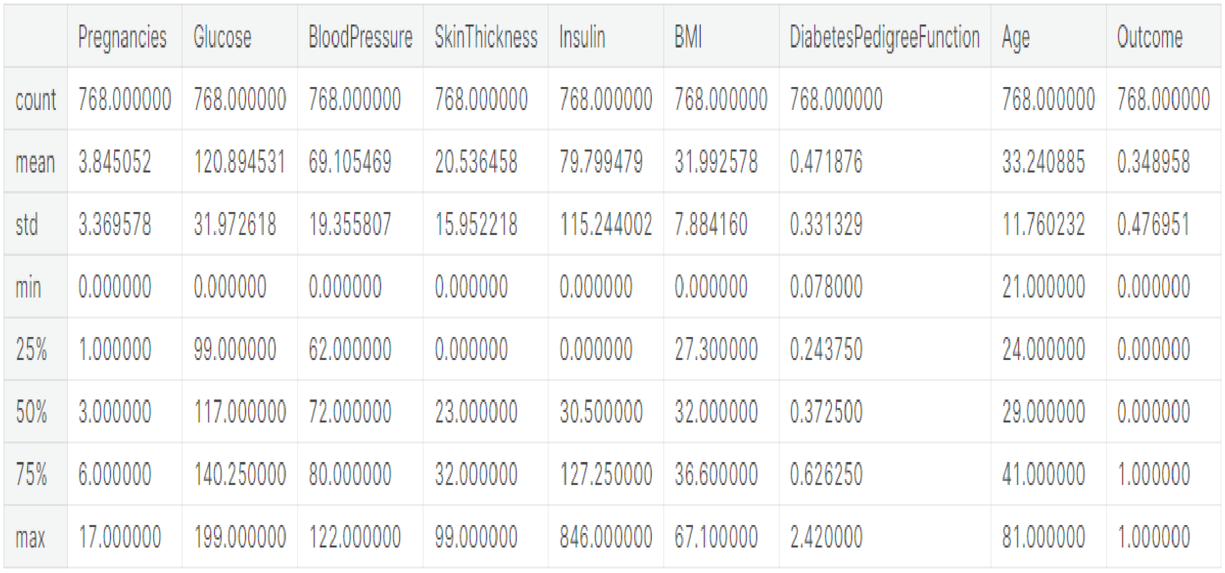
Output:



	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
760	2	88	58	26	16	28.4	0.766	22	0
687	1	107	50	19	0	28.3	0.181	29	0
355	9	165	88	0	0	30.4	0.302	49	1
187	1	128	98	41	58	32.0	1.321	33	1
235	4	171	72	0	0	43.6	0.479	26	1

```
df.describe()
```

Output:



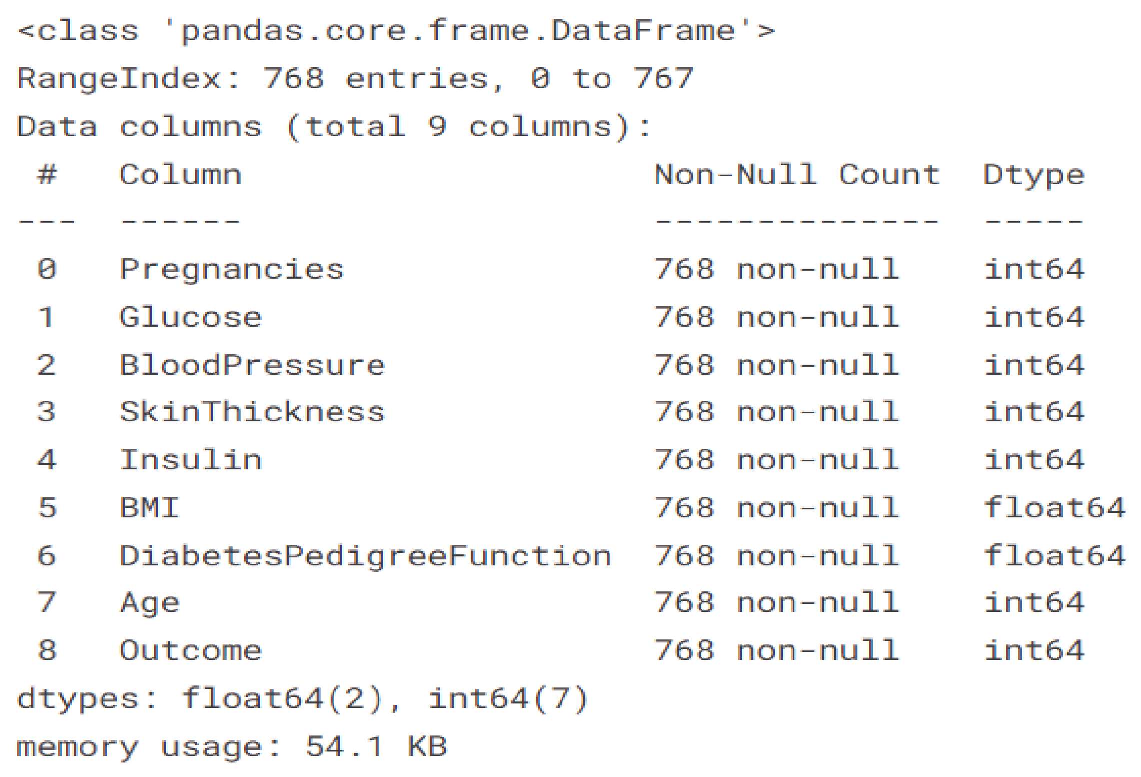
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

```
df.dtypes
```

Output:

```
Pregnancies                   int64
Glucose                       int64
BloodPressure                 int64
SkinThickness                 int64
Insulin                       int64
BMI                         float64
DiabetesPedigreeFunction    float64
Age                           int64
Outcome                       int64
dtype: object
```

```
df.info()
```
Output:



```
<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
```

```
df.size
```
Output:

```
6912
```

```
df.shape
```
Output:

```
(768, 9)
```

Data Cleaning:

```
df.shape
```
Output:

```
(768, 9)
```

```
df=df.drop_duplicates()
df.shape
```

Output:

```
(768, 9)
```

Check null Values:

```
df.isnull().sum()
```

Output:

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

There is no Missing Values present in the Data

```
df.columns
```

Output:

```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

Check the number of Zero Values in Dataset:

```
print("No. of Zero Values in Glucose ", df[df['Glucose']==0].shape[0])
```

Output:

```
No. of Zero Values in Glucose  5
```

```
print("No. of Zero Values in Blood Pressure ", df[df['BloodPressure']==0].
shape[0])
```

Output:

```
No. of Zero Values in Blood Pressure  35
```

```
print("No. of Zero Values in SkinThickness ", df[df['SkinThickness']==0].
shape[0])
```

Output:

```
No. of Zero Values in SkinThickness  227
```

```
print("No. of Zero Values in Insulin ", df[df['Insulin']==0].shape[0])
```

Output:

```
No. of Zero Values in Insulin  374
```

```
print("No. of Zero Values in BMI ", df[df['BMI']==0].shape[0])
```

Output:

```
No. of Zero Values in BMI  11
```

Replace zeroes with mean of that Columns:

```
df['Glucose']=df['Glucose'].replace(0, df['Glucose'].mean())
print('No of zero Values in Glucose ', df[df['Glucose']==0].shape[0])
```

Output:

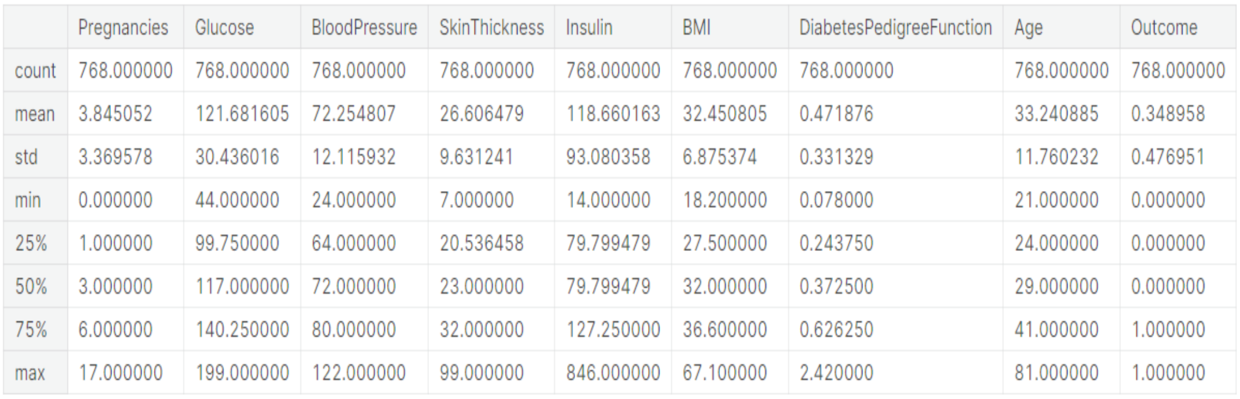
```
No of zero Values in Glucose  0
```

```
df['BloodPressure']=df['BloodPressure'].replace(0, df['BloodPressure'].mean
())
df['SkinThickness']=df['SkinThickness'].replace(0, df['SkinThickness'].mean
())
df['Insulin']=df['Insulin'].replace(0, df['Insulin'].mean())
df['BMI']=df['BMI'].replace(0, df['BMI'].mean())
```

Validate the Zero Values:

```
df.describe()
```

Output:



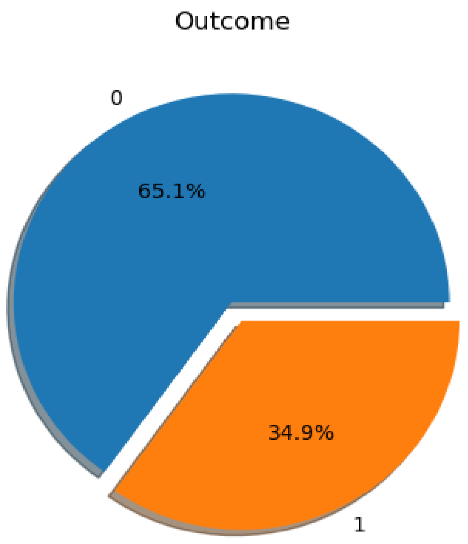
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	121.681605	72.254807	26.606479	118.660163	32.450805	0.471876	33.240885	0.348958
std	3.369578	30.436016	12.115932	9.631241	93.080358	6.875374	0.331329	11.760232	0.476951
min	0.000000	44.000000	24.000000	7.000000	14.000000	18.200000	0.078000	21.000000	0.000000
25%	1.000000	99.750000	64.000000	20.536458	79.799479	27.500000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	79.799479	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

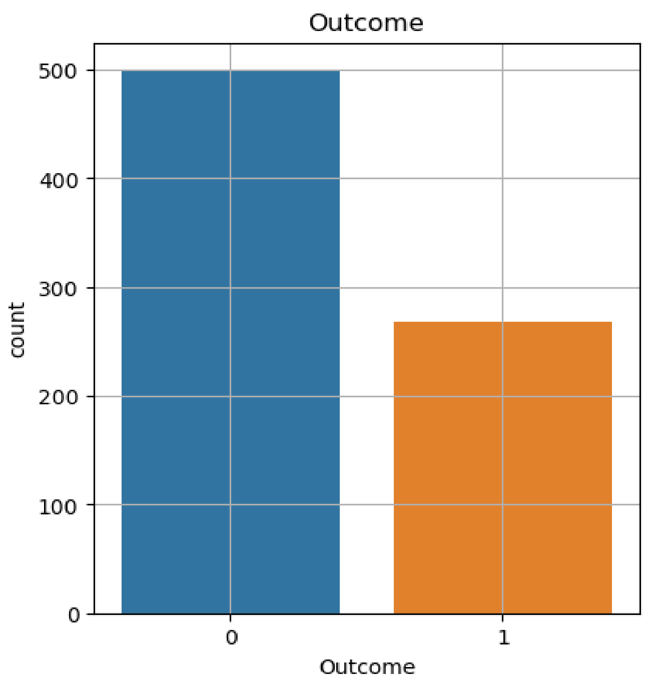
Data Visualization:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming 'df' is your DataFrame containing the dataset
# If you haven't imported your dataset yet, import it here

# Create subplots
f, ax = plt.subplots(1, 2, figsize=(10, 5))

# Pie chart for Outcome distribution
df['Outcome'].value_counts().plot.pie(explode=[0, 0.1], autopct='%1.1f%
%', ax=ax[0], shadow=True)
ax[0].set_title('Outcome')
ax[0].set_ylabel(' ')

# Count plot for Outcome distribution
sns.countplot(x='Outcome', data=df, ax=ax[1])  # Use 'x' instead of 'Out
come'
ax[1].set_title('Outcome')
# Display class distribution
N, P = df['Outcome'].value_counts()
print('Negative (0):', N)
print('Positive (1):', P)

# Adding grid and showing plots
plt.grid()
plt.show()
```

Output:

```
Negative (0): 500
Positive (1): 268
```



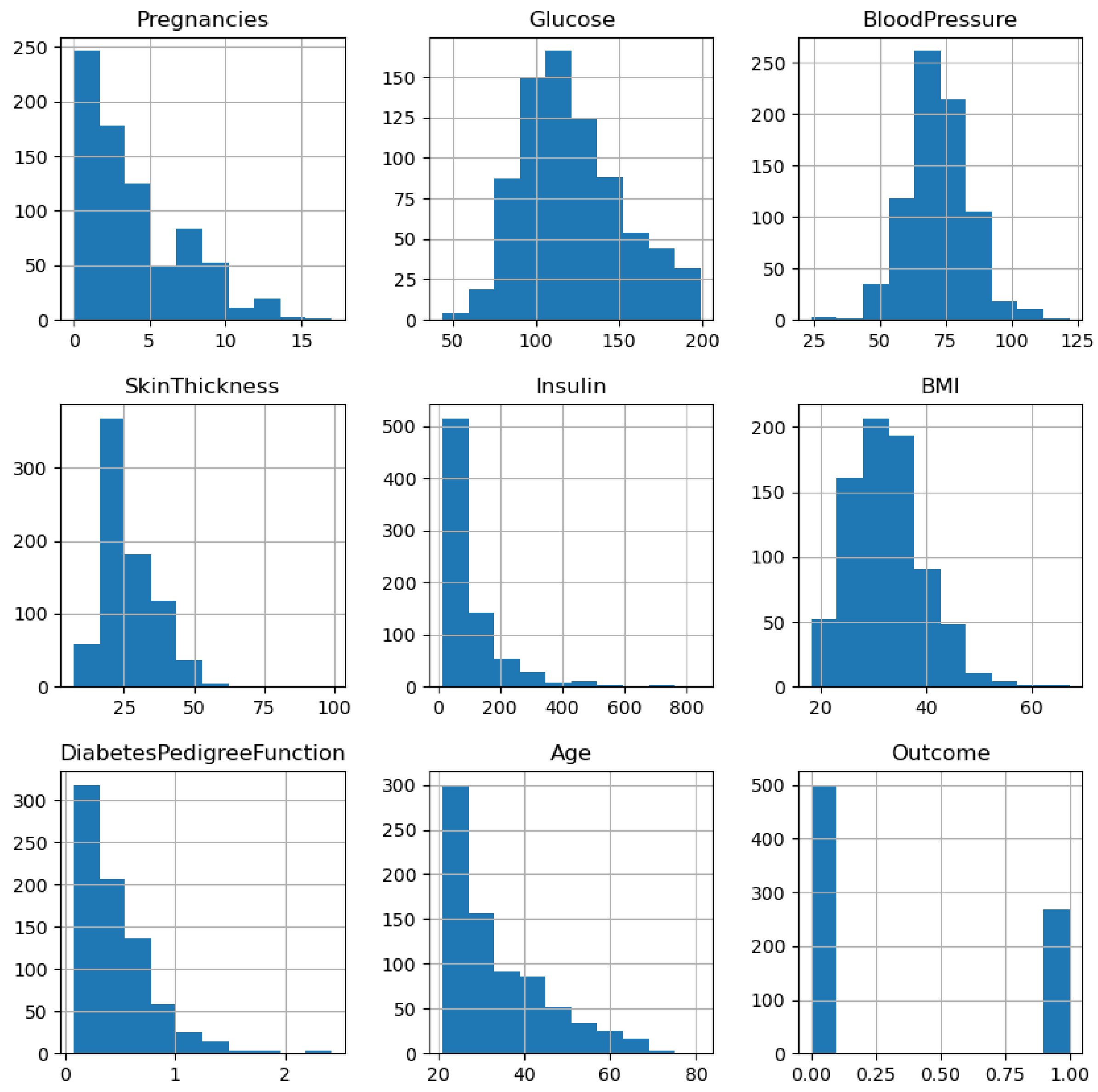


- *1 Represent --> Diabetes Positive*
- *0 Represent --> Daibetes Negative*

Histograms:

```
df.hist(bins=10, figsize=(10, 10))
plt.show()
```

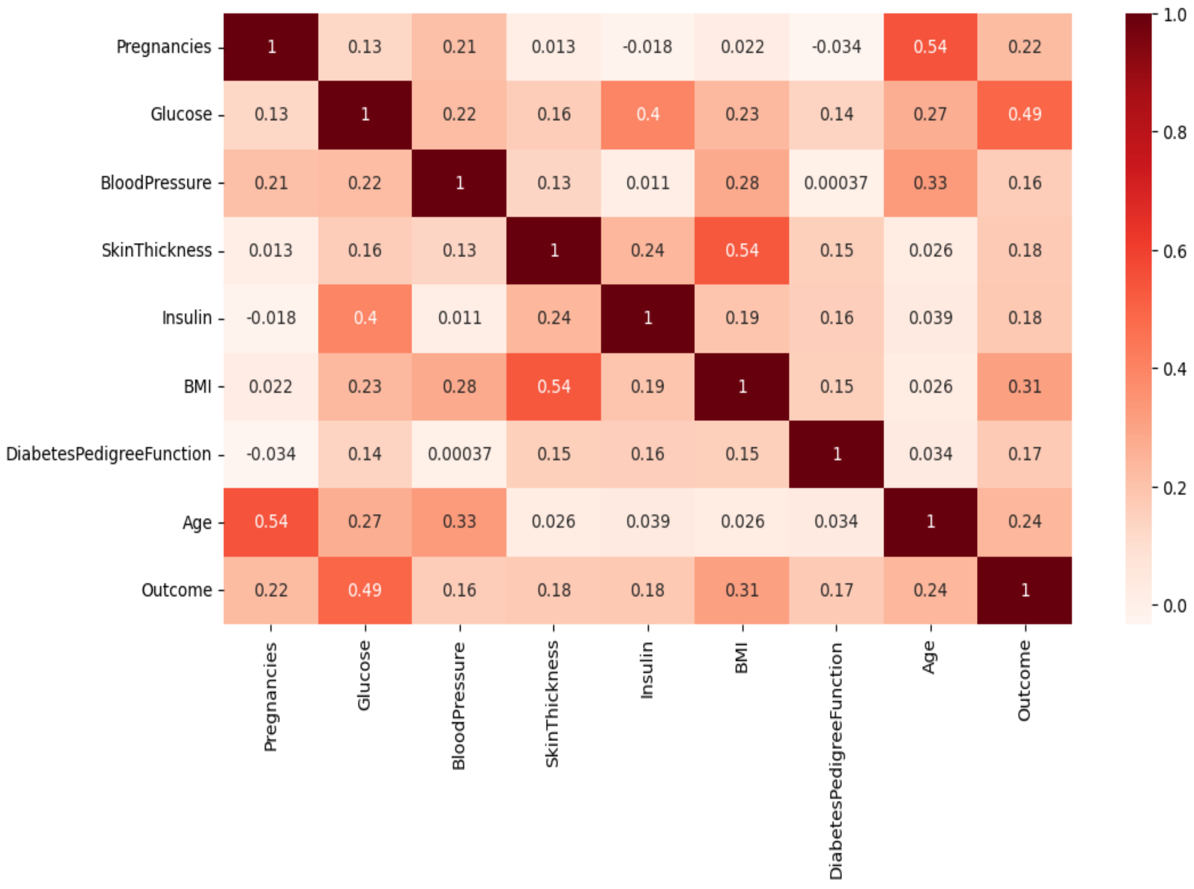
Output:



```
plt.figure(figsize=(12, 6))
sns.heatmap(df.corr(), annot=True, cmap='Reds')
plt.plot()
# Creating a heatmap of the correlation matrix for the columns in the Dat
aFrame data
```

```
[]
```

Output:



```
mean = df['Outcome'].mean()
# Calculating the mean value of the 'Outcome' column in the DataFrame data
mean
# Displaying the calculated mean value
```

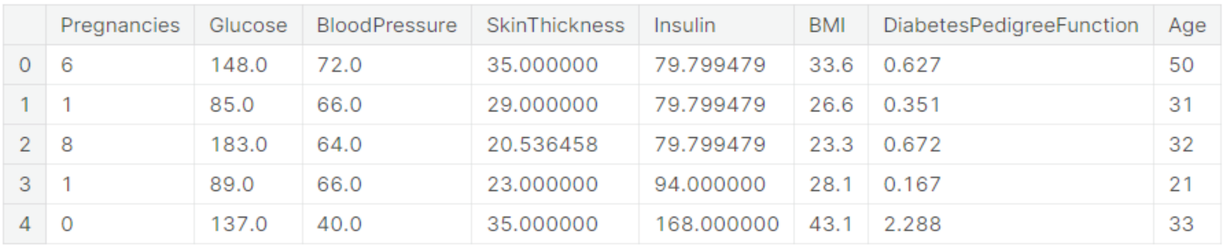
Output:

```
0.3489583333333333
```

Split the DataFrame into X and y:

```
target_name='Outcome'

y=df[target_name]

X= df.drop(target_name, axis=1)


X.head()
```

Output:



	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148.0	72.0	35.000000	79.799479	33.6	0.627	50
1	1	85.0	66.0	29.000000	79.799479	26.6	0.351	31
2	8	183.0	64.0	20.536458	79.799479	23.3	0.672	32
3	1	89.0	66.0	23.000000	94.000000	28.1	0.167	21
4	0	137.0	40.0	35.000000	168.000000	43.1	2.288	33

```
y.head()
```

Output:

```
0     1
1     0
2     1
3     0
4     1
Name: Outcome, dtype: int64
```

Future Scalling:

```
# Standard Scaler:
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X)
SSX = scaler.transform(X)


from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(SSX, y, test_size=0.2,
random_state=7)


X_train.shape, y_train.shape
```

Output:

```
((614, 8), (614,))
```

```
X_test.shape, y_test.shape
```

Output:

```
((154, 8), (154,))
```

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)*

10. Number of times pregnant
11. Plasma glucose concentration a 2 hours in an oral glucose tolerance test
12. Diastolic blood pressure (mm Hg)
13. Triceps skin fold thickness (mm)
14. 2-Hour serum insulin (mu U/ml)
15. Body mass index (weight in kg/(height in m)^2)
16. Diabetes pedigree function
17. Age (years)
18. 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.

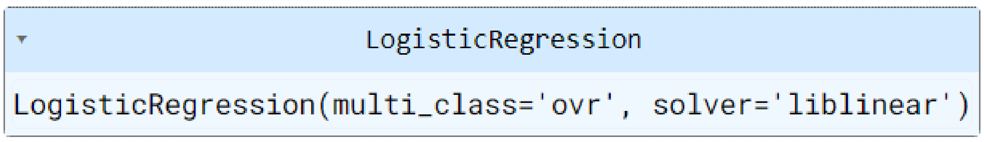
About the Data:

Data Overview: This is a diabetes.csv data

Classification Algorithms:

Logistic Regression:

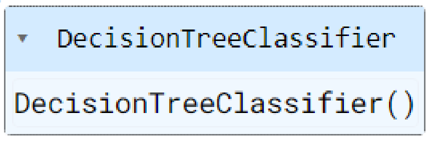
```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(solver='liblinear', multi_class='ovr')
lr.fit(X_train, y_train)
```



```
▾ 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"***

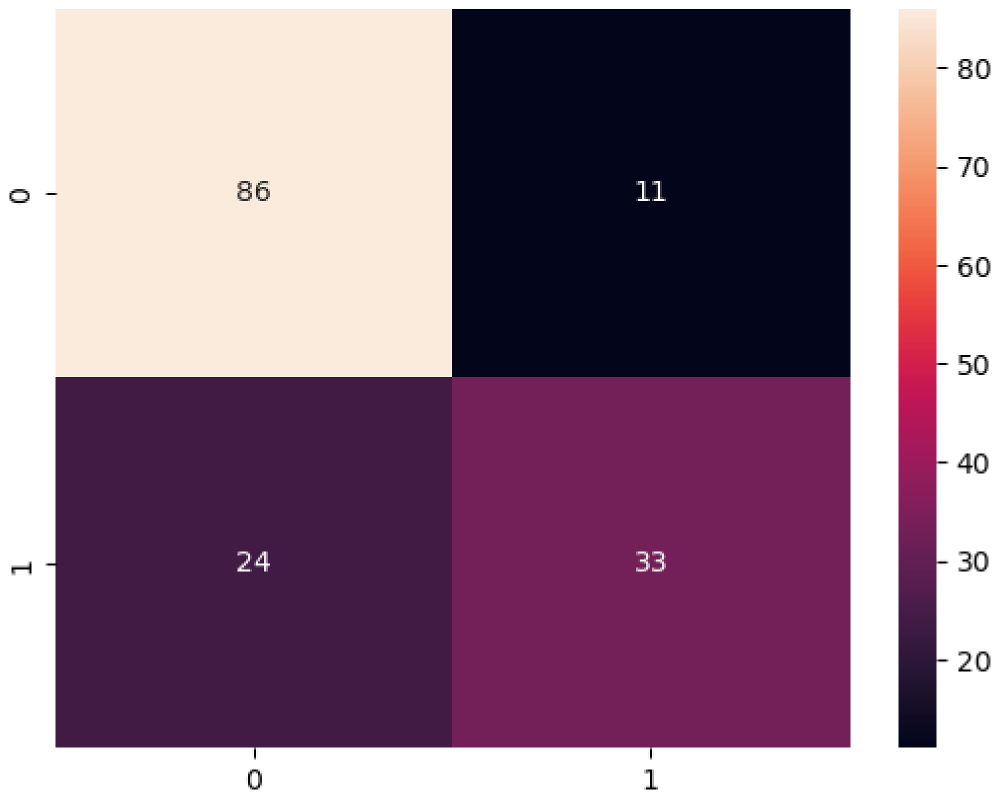
```
from sklearn.metrics import classification_report, confusion_matrix

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
```

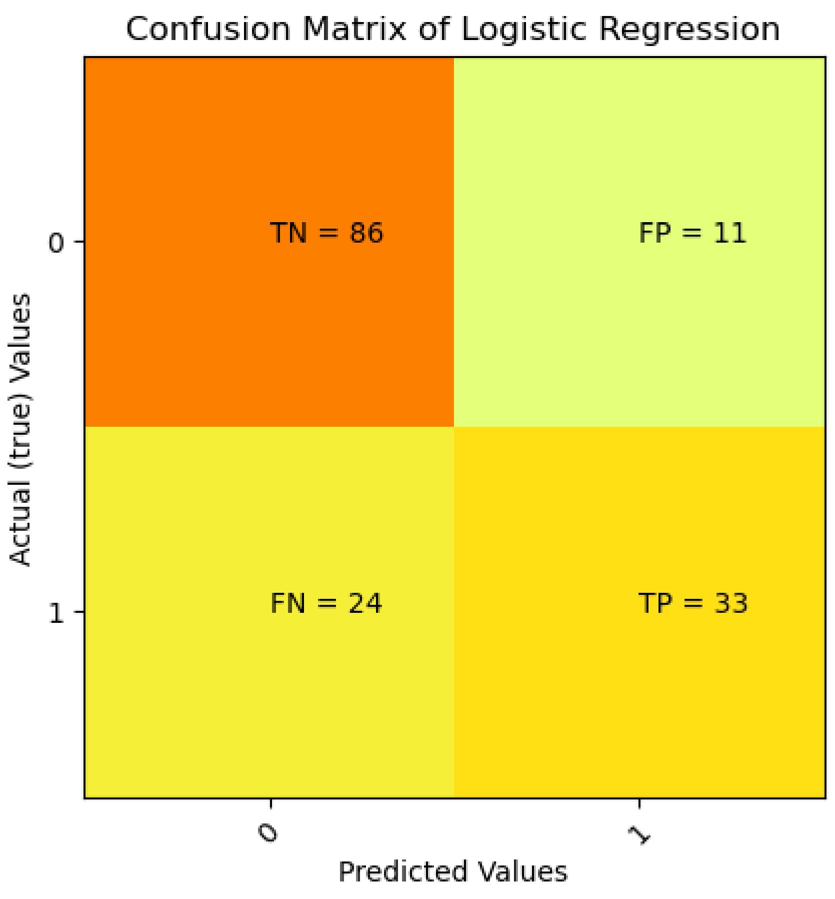
Output:

```
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=['Pre
dicted 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)
```

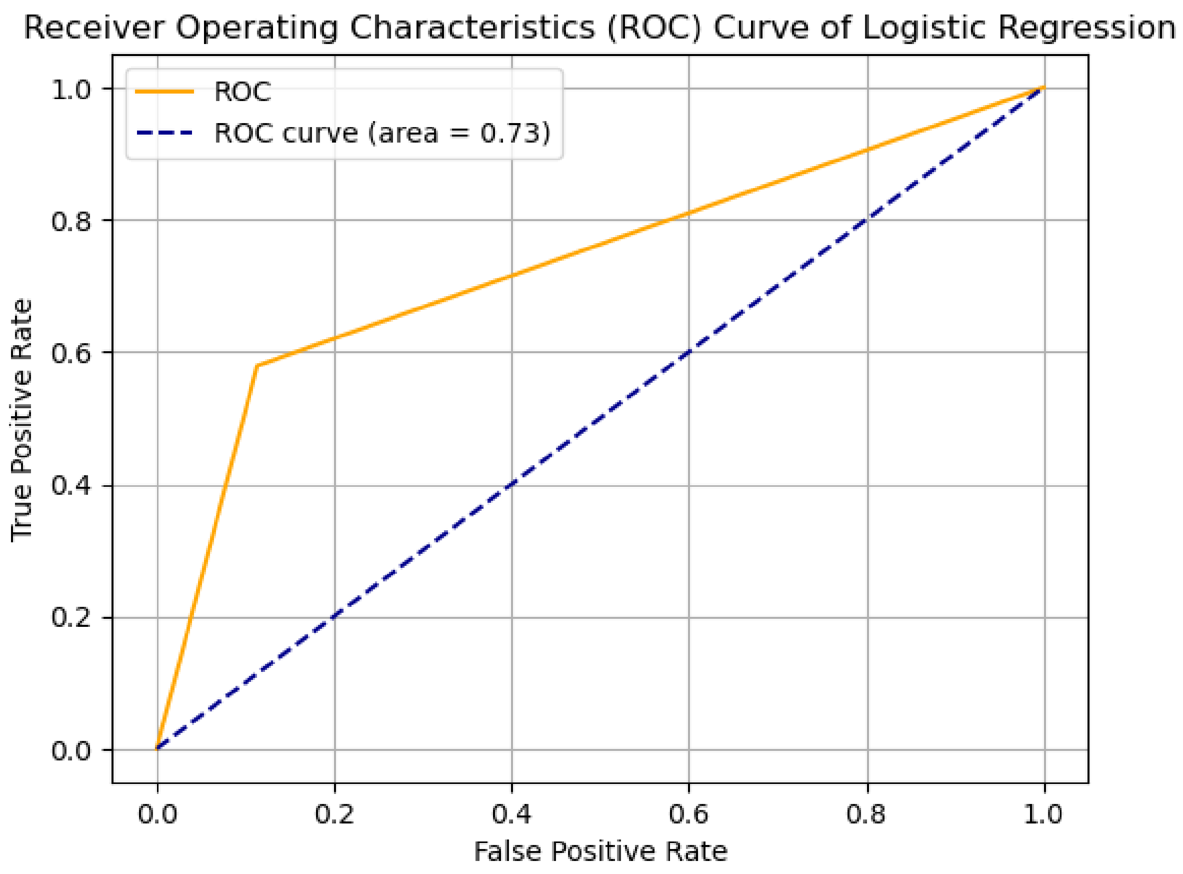
Output:

```
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
```

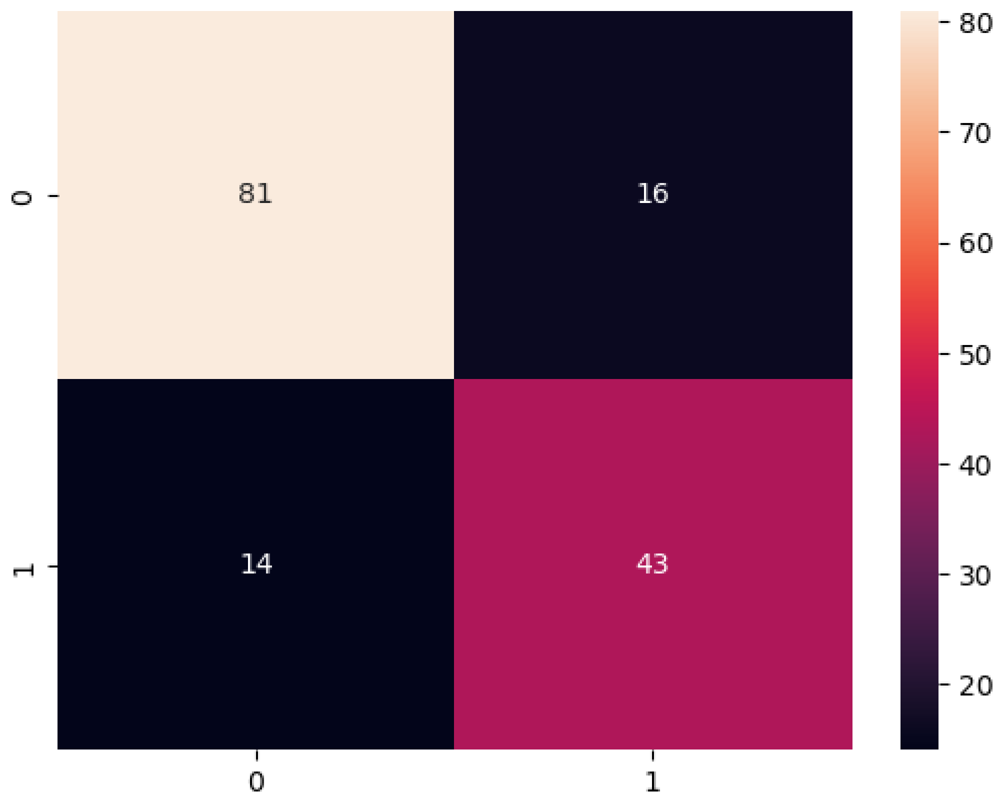
Output:

```
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)
```

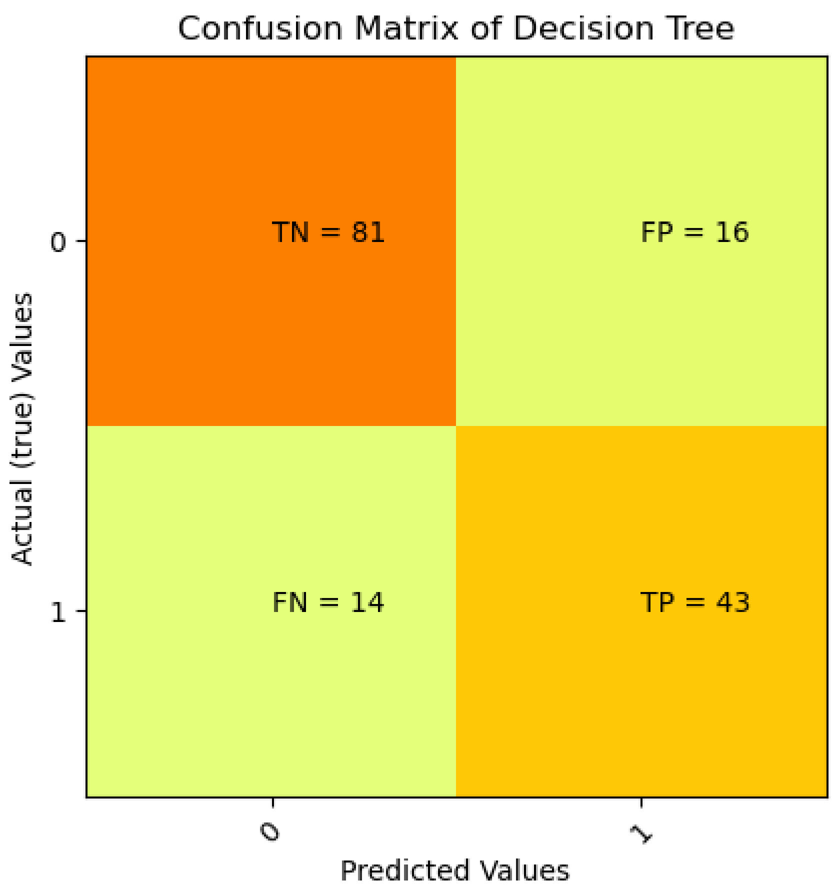
```python
print('TN - True Negative {}'.format(cm[0,0]))
print('FP - False Positive {}'.format(cm[0,1]))
print('FN - False Negative {}'.format(cm[1,0]))
print('TP - True Positive {}'.format(cm[1,1]))
print('Accuracy Rate: {}'.format(np.divide(np.sum([cm[0,0], cm[1,1]]), np.s
um(cm))*100))
print('Misclassification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

```python
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:

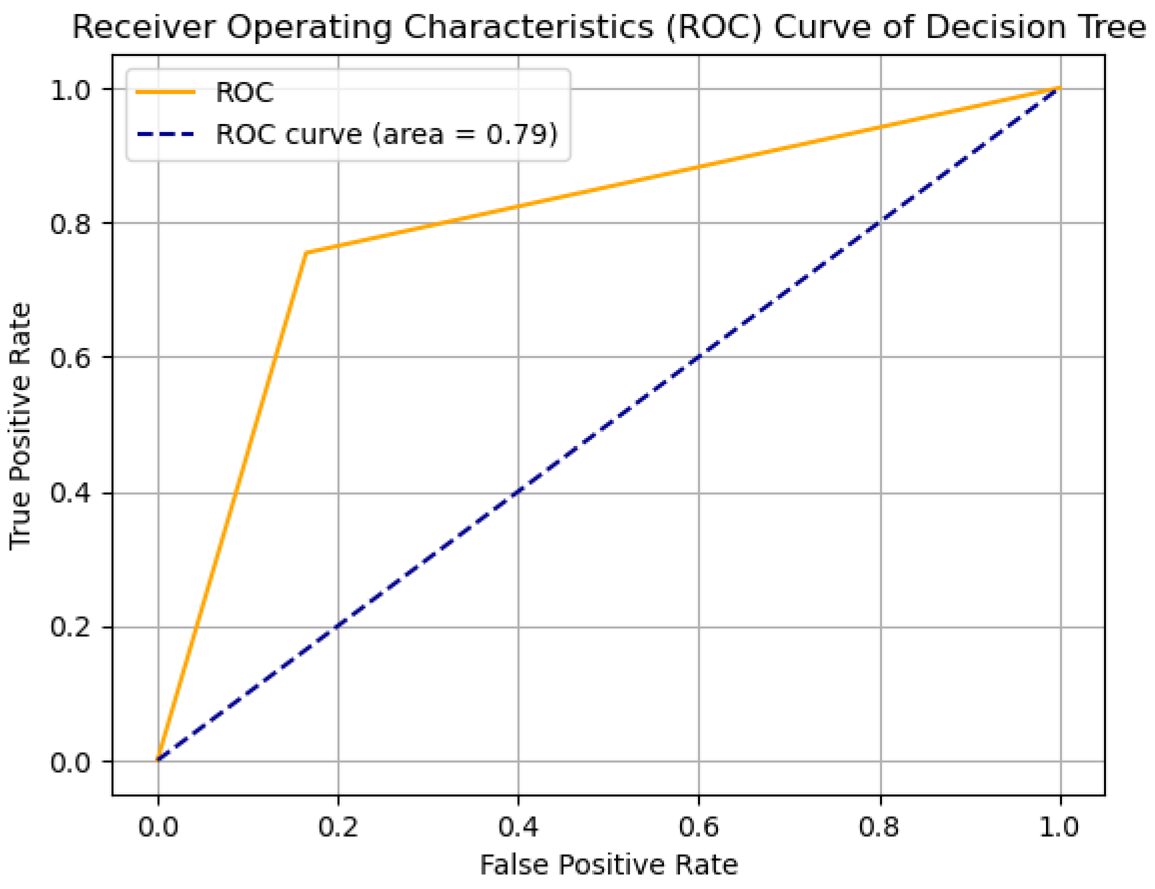
```
# Area under Curve:
auc= roc_auc_score(y_test, dt_pred)
print("ROC AUC SCORE of Decision Treeis ", auc)
```

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