Description:

The objective of this diabetes dataset is to predict whether patient has diabetes or not. The dataset consist of several independent variables including Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DiabetesPedigreeFunction, Age and one target variable(Outcome).

Features

Pregnancies: To express the Number of pregnancies

Glucose: To express the Glucose level in blood

BloodPressure: To express the Blood pressure measurement

SkinThickness: To express the thickness of the skin

Insulin: To express the Insulin level in blood

BMI: To express the Body mass index

DiabetesPedigreeFunction: Diabetes Pedigree Function

Age: To express the age

Outcome: To express the final result 1 is Diabetes and 0 is Non-Diabetes

In [1]:
```python
# importing libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```python
# loading dataset

df = pd.read_csv("diabetes.csv")
df
```

Out[2]:

	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
...	...	...	...	...	...	...	...	...	...
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

768 rows × 9 columns

In [3]:
```python
# EDA
# First five rows of DataFrame
df.head()
```

Out[3]:

	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

In [4]:
```python
# shape of DataFrame
df.shape
```

Out[4]: (768, 9)

In [5]:
```python
# getting overview of columns
df.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 [6]:
```python
# Display the column names of the DataFrame
df.columns
```

Out[6]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

In [7]:
```python
# Display the number of Unique Values in Each Column
df.nunique()
```

Out[7]:
```
Pregnancies                  17
Glucose                     136
BloodPressure                47
SkinThickness                51
Insulin                     186
BMI                         248
DiabetesPedigreeFunction    517
Age                          52
Outcome                       2
dtype: int64
```

In [8]:
```python
# Data Types of Columns in a DataFrame
df.dtypes
```

Out[8]:
```
Pregnancies                   int64
Glucose                       int64
BloodPressure                 int64
SkinThickness                 int64
Insulin                       int64
BMI                         float64
DiabetesPedigreeFunction    float64
Age                           int64
Outcome                       int64
dtype: object
```

In [9]:
```python
# summary statistics of DataFrame
df.describe()
```

Out[9]:

	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

In [10]:
```python
# check for null values
df.isnull().sum()
```

Out[10]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [11]:
```python
# To check the duplicate rows in DataFrame
sum(df.duplicated())
```

Out[11]: 0

In [12]:
```python
# check the no.of zero values in dataset

print("No. of zero values in Glucose", df[df["Glucose"]==0].shape[0])
print("No. of zero values in BloodPressure", df[df["BloodPressure"]==0].shape[0])
print("No. of zero values in SkinThickness", df[df["SkinThickness"]==0].shape[0])
print("No. of zero values in Insulin", df[df["Insulin"]==0].shape[0])
print("No. of zero values in BMI", df[df["BMI"]==0].shape[0])
```

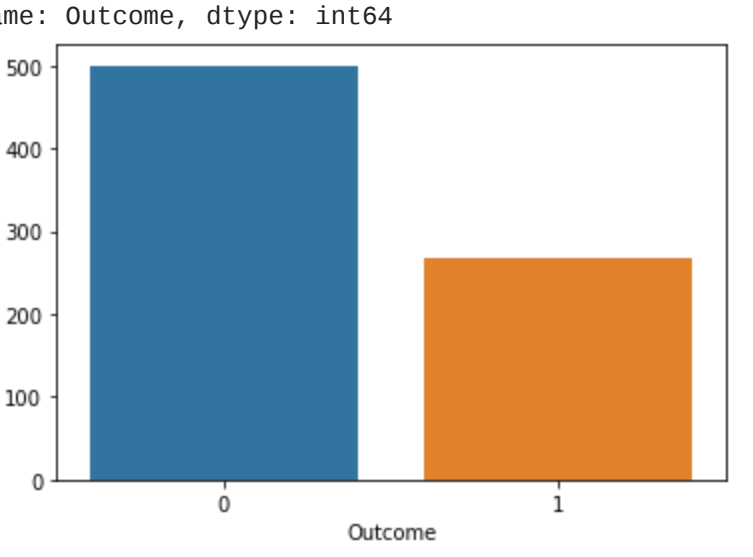
```
No. of zero values in Glucose 5
No. of zero values in BloodPressure 35
No. of zero values in SkinThickness 227
No. of zero values in Insulin 374
No. of zero values in BMI 11
```

In [13]:
```python
# Replace no.of zero values with mean of the columns

df["Glucose"]=df["Glucose"].replace(0,df["Glucose"].mean())
print("No. of zero values in Glucose", df[df["Glucose"]==0].shape[0])
df["BloodPressure"]=df["BloodPressure"].replace(0,df["BloodPressure"].mean())
print("No. of zero values in BloodPressure", df[df["BloodPressure"]==0].shape[0])
df["SkinThickness"]=df["SkinThickness"].replace(0,df["SkinThickness"].mean())
print("No. of zero values in SkinThickness", df[df["SkinThickness"]==0].shape[0])
df["Insulin"]=df["Insulin"].replace(0,df["Insulin"].mean())
print("No. of zero values in Insulin", df[df["Insulin"]==0].shape[0])
df["BMI"]=df["BMI"].replace(0,df["BMI"].mean())
print("No. of zero values in BMI", df[df["BMI"]==0].shape[0])
```

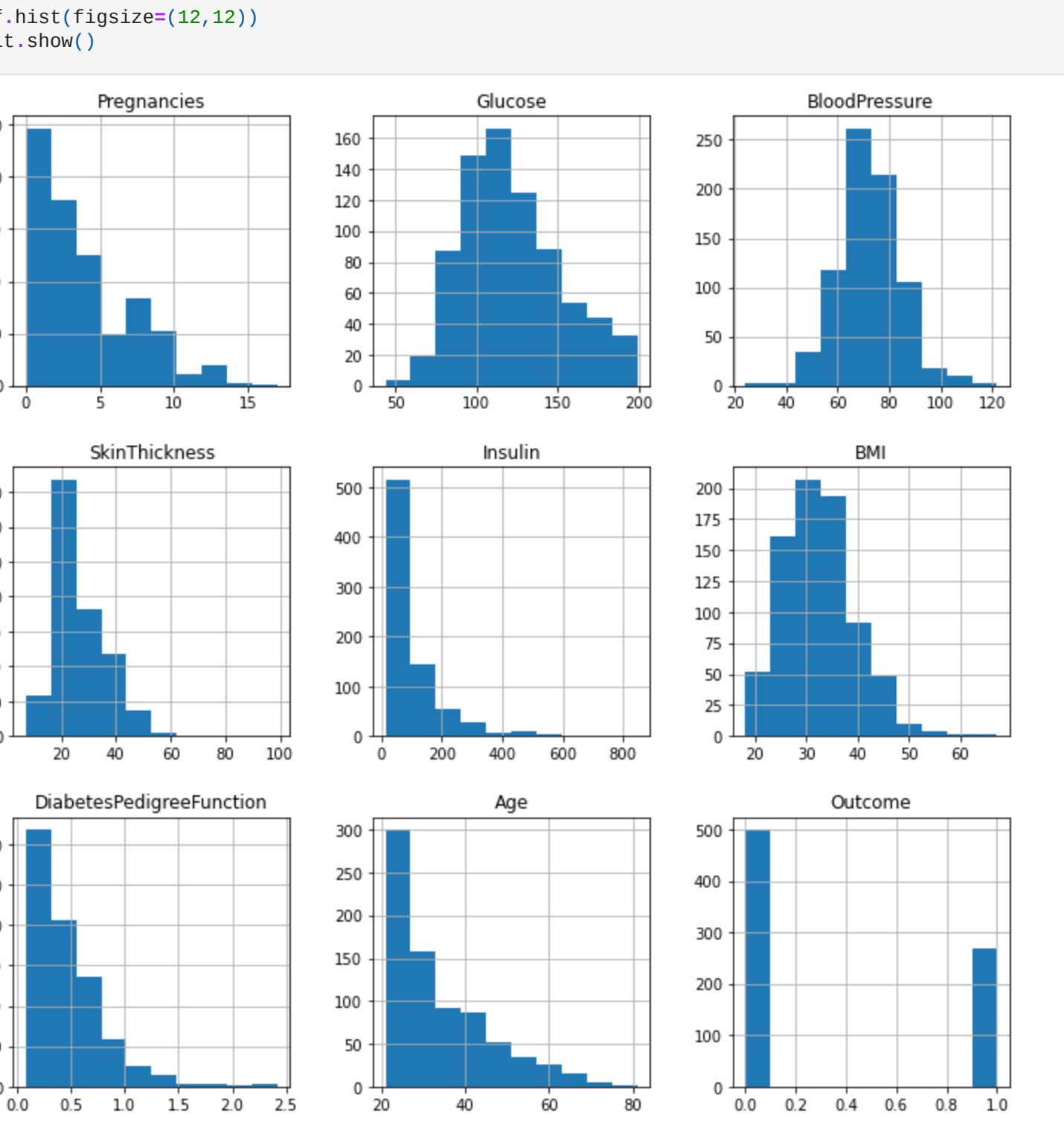
```
No. of zero values in Glucose 0
No. of zero values in BloodPressure 0
No. of zero values in SkinThickness 0
No. of zero values in Insulin 0
No. of zero values in BMI 0
```

In [14]:
```python
#Distribution of Outcomes(0=Non-Diabetic, 1=Diabetic)

outcome_counts = df["Outcome"].value_counts()
print(outcome_counts)
sns.countplot(x = 'Outcome',data = df);
```

```
0    500
1    268
Name: Outcome, dtype: int64
```

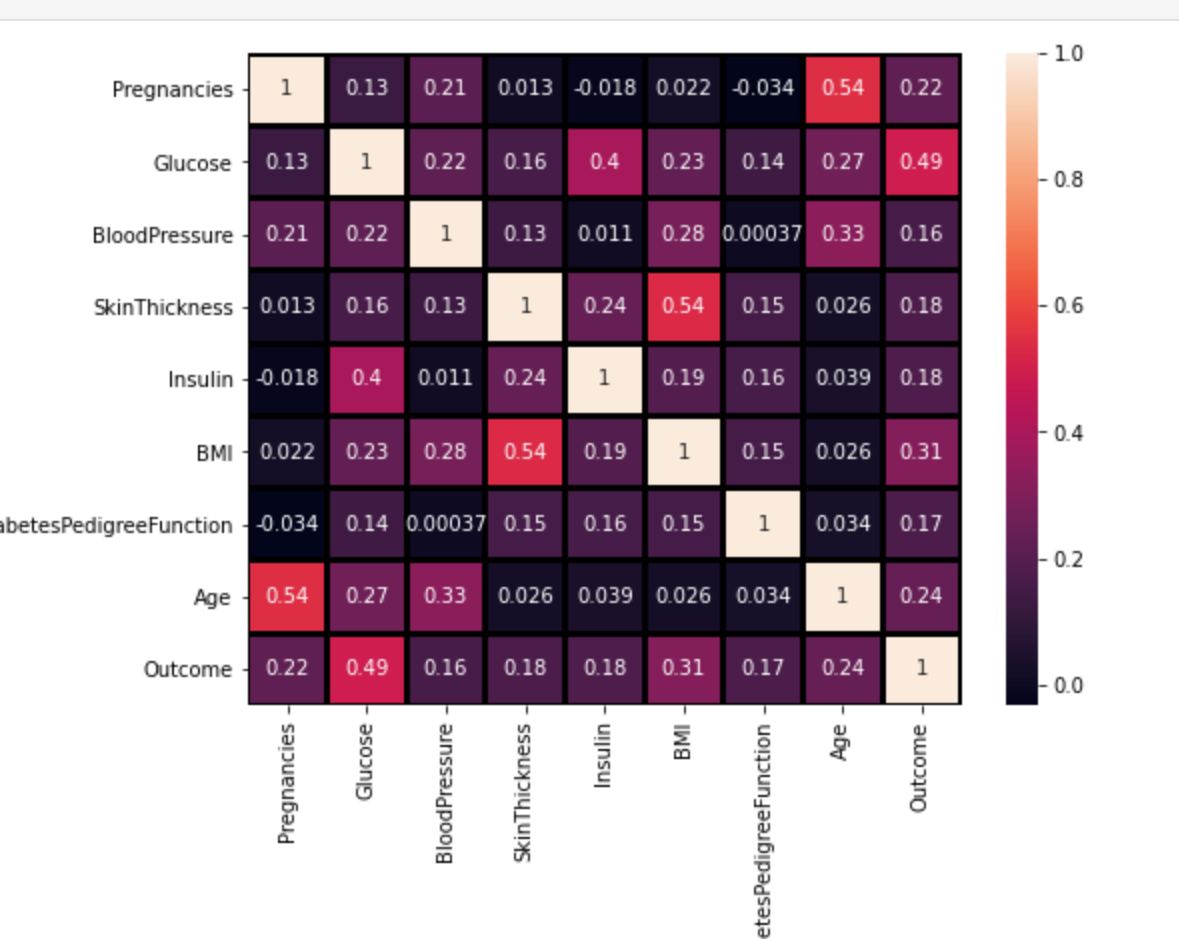


In [15]:
```python
# Histogram is used to display numeric data.
# wether the data normally distributed or if it's skewed

df.hist(figsize=(12,12))
plt.show()
```



In [16]:
```python
# pairplot to create scatterplots between all variables

sns.pairplot(data=df, hue = "Outcome",palette = "Accent_r")
plt.show()
```

In [17]:
```python
# Checking correlation between features

plt.figure(figsize=(8,6))
sns.heatmap(df.corr(), annot =  True, linewidths = 2, linecolor= "black")
plt.show()
```



In [18]:
```python
# split the DataFrame into X & y

target_column = "Outcome"

X = df.drop(target_column, axis = 1)
y = df[target_column]
```

In [19]:
```python
X.head()
```

Out[19]:

	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

In [20]:
```python
y.head()
```

Out[20]:
```
0    1
1    0
2    1
3    0
4    1
Name: Outcome, dtype: int64
```

In [21]:
```python
# Split the dataset training & testing set

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.2,random_state=1)
```

In [22]:
```python
# feature scaling

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train= sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [23]:
```python
print(X_train.shape)
print(X_train)
```

```
(614, 8)
[[ 1.51659096  0.75976671  0.63380049 ...  0.83680677  0.52952571
   0.56793202]
 [ 1.81201827  0.2347029  -0.81497881 ...  1.32103145 -0.06968859
   0.39844951]
 [ 0.92573636 -0.65134228  0.15087406 ...  0.73409244 -0.79424873
   0.99163829]
 ...
 [ 2.69830017  0.13625344  1.43867788 ...  1.64384791  0.36443605
   0.73741453]
 [ 0.03945446  1.6129954  -0.01010142 ...  1.67319486  0.04648561
  -0.61844554]
 [ 1.51659096 -0.65134228  0.31184954 ...  0.10313301  0.61512775
   1.07637954]]
```

In [24]:
```python
print(y_train.shape)
print(y_train)
```

```
(614,)
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 [25]:
```python
print(X_test.shape)
print(X_test)
```

```
(154, 8)
[[ 0.92573636  0.46441832  0.15087406 ... -0.90933678  0.56009786
   1.50008581]
 [-0.84682744  0.95666564 -0.97595429 ... -0.8946633  -0.87067912
  -0.95741055]
 [ 0.63030906 -0.42162686 -0.97595429 ... -1.05607153 -0.78813429
  -0.53370428]
 ...
 [ 1.22116366  2.1708757   0.47282501 ...  2.30415429 -0.99908218
   0.82215578]
 [ 0.33488176  0.46441832  0.79477597 ... -0.03001731  0.53869735
   3.02542839]
 [ 0.03945446 -1.04514013 -0.2430757  ... -0.61586727  0.44698088
  -0.19473927]]
```

In [26]:
```python
print(y_test.shape)
print(y_test)
```

```
(154,)
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 [27]:
```python
# Build the classification algorithms
# Logistic Regression

lr = LogisticRegression()
lr.fit(X_train, y_train)
```

Out[27]: LogisticRegression()

In [28]:
```python
# making prediction
# making prediction on test data by using logistic regression

lr_pred = lr.predict(X_test)
```

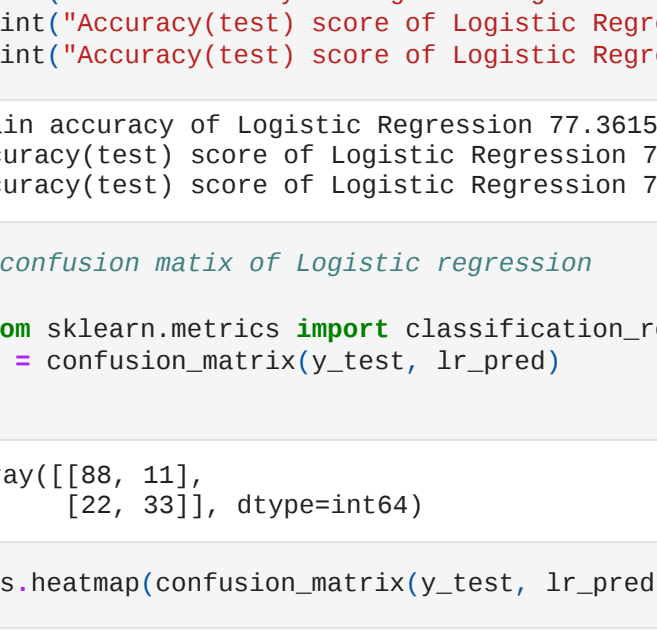
In [29]:
```python
# Evaluations
# train score & test score of Logistic Regression

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(test) score of Logistic Regression", accuracy_score(y_test, lr_pred)*100)
```

```
Train accuracy of Logistic Regression 77.36156351791531
Accuracy(test) score of Logistic Regression 78.57142857142857
Accuracy(test) score of Logistic Regression 78.57142857142857
```

In [30]:
```python
# confusion matix of Logistic regression

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

Out[30]:
```
array([[88, 11],
       [22, 33]], dtype=int64)
```

In [31]:
```python
sns.heatmap(confusion_matrix(y_test, lr_pred),annot = True);
```



In [32]:
```python
TN = cm[0,0]
FP = cm[0,1]
FN = cm[1,0]
TP = cm[1,1]
```

In [33]:
```python
[TN,FP,FN,TP]
```

Out[33]: [88, 11, 22, 33]

In [34]:
```python
pd.crosstab(y_test, lr_pred,rownames = ["Actual values"] , colnames = ["Predicted Values"] ,margins = True)
```

Out[34]:

Predicted Values	0	1	All
Actual values			
0	88	11	99
1	22	33	55
All	110	44	154

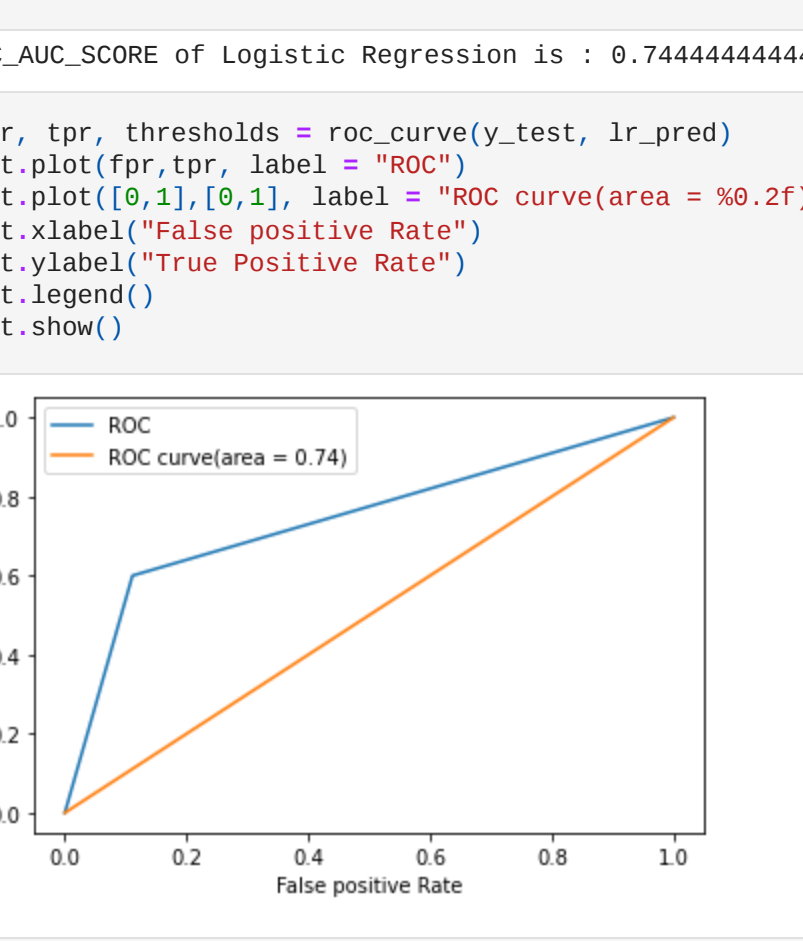
In [35]:
```python
print("Classification report of Logistic Regression:\n",classification_report(y_test, lr_pred, digits = 3))
```

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

           0      0.800     0.889     0.842        99
           1      0.750     0.600     0.667        55

    accuracy                          0.786       154
   macro avg      0.775     0.744     0.754       154
weighted avg      0.782     0.786     0.779       154
```

In [36]:
```python
# ROC curve & ROC AUC
# Area under curve

auc = roc_auc_score(y_test, lr_pred)
print("ROC_AUC_SCORE of Logistic Regression is :", auc)
```

```
ROC_AUC_SCORE of Logistic Regression is : 0.7444444444444445
```

In [37]:
```python
fpr, tpr, thresholds = roc_curve(y_test, lr_pred)
plt.plot(fpr,tpr, label = "ROC")
plt.plot([0,1],[0,1], label = "ROC curve(area = %0.2f)" % auc)
plt.xlabel("False positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.show()
```



In [ ]:

Ashwini Dekate