In [1]:
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

In [2]:
```
data=pd.read_csv('Diabetes.csv')
```

In [3]:
```
print(data)
```

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

     DiabetesPedigreeFunction  Age  Outcome
0                       0.627   50        1
1                       0.351   31        0
2                       0.672   32        1
3                       0.167   21        0
4                       2.288   33        1
..                        ...  ...      ...
763                     0.171   63        0
764                     0.340   27        0
765                     0.245   30        0
766                     0.349   47        1
767                     0.315   23        0

[768 rows x 9 columns]
```

In [4]:
```
data.describe()
```

Out[4]:

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

In [5]:

```
data.info()
```

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

In [6]:

```
data.corr()
```

Out[6]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin
Pregnancies	1.000000	0.129459	0.141282	-0.081672	-0.073535
Glucose	0.129459	1.000000	0.152590	0.057328	0.331357
BloodPressure	0.141282	0.152590	1.000000	0.207371	0.088933
SkinThickness	-0.081672	0.057328	0.207371	1.000000	0.436783
Insulin	-0.073535	0.331357	0.088933	0.436783	1.000000
BMI	0.017683	0.221071	0.281805	0.392573	0.197859
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928	0.185071
Age	0.544341	0.263514	0.239528	-0.113970	-0.042163
Outcome	0.221898	0.466581	0.065068	0.074752	0.130548

In [7]:

```
d=data.loc[(data['Glucose']!=0) & (data['BloodPressure']!=0) &
           (data['SkinThickness']!=0) & (data['Insulin']!=0) & (data['BMI']!=0)]
```

In [8]:

```
d.describe()
```

Out[8]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Dia
count	392.000000	392.000000	392.000000	392.000000	392.000000	392.000000	
mean	3.301020	122.627551	70.663265	29.145408	156.056122	33.086224	
std	3.211424	30.860781	12.496092	10.516424	118.841690	7.027659	
min	0.000000	56.000000	24.000000	7.000000	14.000000	18.200000	
25%	1.000000	99.000000	62.000000	21.000000	76.750000	28.400000	
50%	2.000000	119.000000	70.000000	29.000000	125.500000	33.200000	
75%	5.000000	143.000000	78.000000	37.000000	190.000000	37.100000	
max	17.000000	198.000000	110.000000	63.000000	846.000000	67.100000	

In [9]:

```
d.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 392 entries, 3 to 765
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               392 non-null    int64  
 1   Glucose                   392 non-null    int64  
 2   BloodPressure             392 non-null    int64  
 3   SkinThickness             392 non-null    int64  
 4   Insulin                   392 non-null    int64  
 5   BMI                       392 non-null    float64
 6   DiabetesPedigreeFunction  392 non-null    float64
 7   Age                       392 non-null    int64  
 8   Outcome                   392 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 30.6 KB
```

In [10]:
```
#data.replace(data['Glucose']==0,value=d['Glucose'].mean(),inplace=True)
data['Glucose'].replace(0,d['Glucose'].mean(),inplace=True)
data['BloodPressure'].replace(0,d['BloodPressure'].mean(),inplace=True)
data['SkinThickness'].replace(0,d['SkinThickness'].mean(),inplace=True)
data['Insulin'].replace(0,d['Insulin'].mean(),inplace=True)
data['BMI'].replace(0,d['BMI'].mean(),inplace=True)
```

In [11]:
```
data.describe()
```

Out[11]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Dial
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	121.692888	72.325800	29.151052	155.795560	32.466469	
std	3.369578	30.436043	12.101807	8.790943	85.021487	6.875558	
min	0.000000	44.000000	24.000000	7.000000	14.000000	18.200000	
25%	1.000000	99.750000	64.000000	25.000000	121.500000	27.500000	
50%	3.000000	117.000000	72.000000	29.145408	156.056122	32.400000	
75%	6.000000	140.250000	80.000000	32.000000	156.056122	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

In [12]:
```
data.corr()
```

Out[12]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin
Pregnancies	1.000000	0.127849	0.208850	0.082926	0.056535
Glucose	0.127849	1.000000	0.219028	0.192985	0.419998
BloodPressure	0.208850	0.219028	1.000000	0.192796	0.072908
SkinThickness	0.082926	0.192985	0.192796	1.000000	0.158154
Insulin	0.056535	0.419998	0.072908	0.158154	1.000000
BMI	0.021589	0.230189	0.281531	0.542239	0.166212
DiabetesPedigreeFunction	-0.033523	0.137004	-0.001108	0.101030	0.098136
Age	0.544341	0.266453	0.325860	0.127780	0.137366
Outcome	0.221898	0.492948	0.164509	0.215277	0.214532

In [13]:
```
x=data.iloc[:,0:8]
```

In [14]:
```
y=data.iloc[:,-1]
```

In [15]:
```
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=0)
```

In [16]:
```
from sklearn.linear_model import LogisticRegression
lr=LogisticRegression()
```

In [17]:
```
lr.fit(x_train,y_train)
```

```
C:\Users\hp\anaconda3\Lib\site-packages\sklearn\linear_model\_logistic.py:458: Conve
rgenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[17]:

▾ LogisticRegression

LogisticRegression()

In [23]:
```
y_pred=lr.predict(x_test)
```

In [24]:
```
print("Predicted Values:")
print(y_pred)
print("Actual Values:")
print(y_test)
```

```
Predicted Values:
[1 0 0 1 0 0 1 1 1 0 1 1 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 1 0 0 1 0 1 0
 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 1 1 1 1 0 0 0 0 0 0 1
 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 0
 0 1 1 1 1 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0
 0 0 0 1 0 0]
Actual Values:
661    1
122    0
113    0
14     1
529    0
      ..
476    1
482    0
230    1
527    0
380    0
Name: Outcome, Length: 154, dtype: int64
```

In [25]:
```python
cf=confusion_matrix(y_test,y_pred)
print(cf)
print("Classification Report for Testing Dataset:")
print(classification_report(y_test,y_pred))
```

```
[[94 13]
 [18 29]]
Classification Report for Testing Dataset:
              precision    recall  f1-score   support

           0       0.84      0.88      0.86       107
           1       0.69      0.62      0.65        47

    accuracy                           0.80       154
   macro avg       0.76      0.75      0.76       154
weighted avg       0.79      0.80      0.80       154
```

In [26]:
```python
y_train_pred=lr.predict(x_train)
```

In [27]:
```python
print("Classification Report for Training Dataset:")
print(classification_report(y_train,y_train_pred))
```

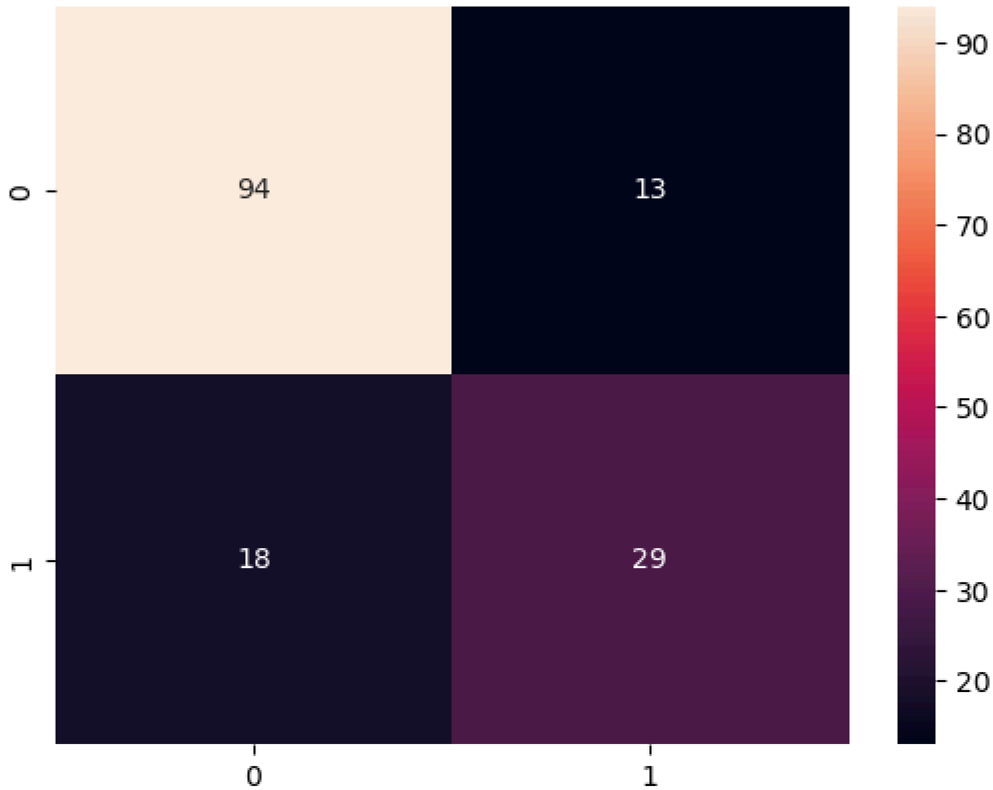
```
Classification Report for Training Dataset:
              precision    recall  f1-score   support

           0       0.78      0.88      0.83       393
           1       0.73      0.57      0.64       221

    accuracy                           0.77       614
   macro avg       0.76      0.73      0.73       614
weighted avg       0.76      0.77      0.76       614
```

In [23]:
```python
cf=confusion_matrix(y_test,y_pred)
sns.heatmap(cf,annot=True)
```

Out[23]: <Axes: >



A pickle file is a serialized Python object stored in a file. It allows you to save and load Python data structures such as lists, dictionaries, and custom objects, preserving their structure and state. Pickle files are commonly used for saving and loading machine learning models, storing intermediate results, or transferring data between Python programs.

The pickle module in Python provides functions for serializing and deserializing Python objects to and from pickle files. This allows you to save complex data structures to a file and then later load them back into memory without losing their original structure or state.

Serialization is the process of converting a data structure or object into a format that can be easily stored, transmitted, or reconstructed later. It involves converting complex data structures, such as lists, dictionaries, or objects, into a byte stream or string representation that can be stored in a file or sent over a network.

In [31]:
```
import pickle
with open('lr.pkl', 'wb') as model_file:
    pickle.dump(lr, model_file)
```

In [ ]: