In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
df = pd.read_csv('heart-statlog.csv')
```

In [3]:

```
df['class'].value_counts()
```

Out[3]:

```
absent     150
present    120
Name: class, dtype: int64
```

In [4]:

```
df['class'] = df['class'].map({'present':1,"absent":0})
```

In [5]:

```
df.head()
```

Out[5]:

	age	sex	chest	resting_blood_pressure	serum_cholestoral	fasting_blood_sugar	resting_ele
0	70	1	4	130	322	0	
1	67	0	3	115	564	0	
2	57	1	2	124	261	0	
3	64	1	4	128	263	0	
4	74	0	2	120	269	0	

In [6]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 270 entries, 0 to 269
Data columns (total 14 columns):
 #   Column                                Non-Null Count  Dtype  
---  ------                                --------------  -----  
 0   age                                   270 non-null    int64  
 1   sex                                   270 non-null    int64  
 2   chest                                 270 non-null    int64  
 3   resting_blood_pressure                270 non-null    int64  
 4   serum_cholestoral                     270 non-null    int64  
 5   fasting_blood_sugar                   270 non-null    int64  
 6   resting_electrocardiographic_results  270 non-null    int64  
 7   maximum_heart_rate_achieved           270 non-null    int64  
 8   exercise_induced_angina               270 non-null    int64  
 9   oldpeak                               270 non-null    float64
 10  slope                                 270 non-null    int64  
 11  number_of_major_vessels               270 non-null    int64  
 12  thal                                  270 non-null    int64  
 13  class                                 270 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 29.7 KB
```

In [7]:

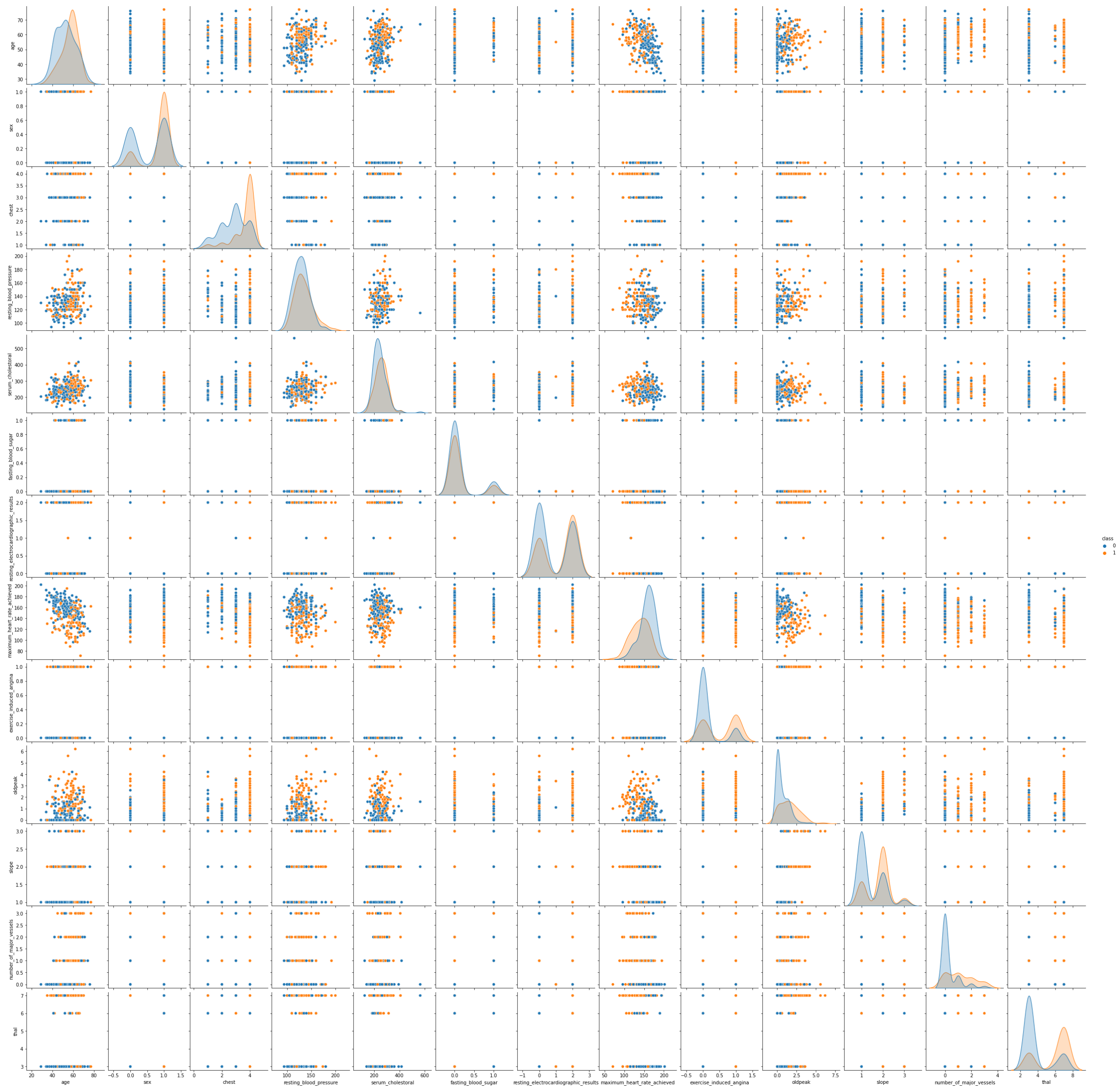
```
df['class'].value_counts()
```

Out[7]:

```
0    150
1    120
Name: class, dtype: int64
```

In [8]:

```
sns.pairplot(df , diag_kind='kde',hue='class')
plt.show()
```



In [9]:

```
x = df.drop(['class'],axis=1)
y = df['class']
```

In [10]:

```
#Normalize the dataset
from sklearn.preprocessing import MinMaxScaler
mx = MinMaxScaler()
scaled = pd.DataFrame(mx.fit_transform(x),columns = x.columns)

from sklearn.model_selection import train_test_split
X_train , X_test , y_train , y_test = train_test_split(scaled , y , test_size =0.2 , random
```

In [11]:

```
## checking distribution of traget variable in train test split
print('Distribution of traget variable in training set')
print(y_train.value_counts())

print('Distribution of traget variable in test set')
print(y_test.value_counts())
```

```
Distribution of traget variable in training set
0    120
1     96
Name: class, dtype: int64
Distribution of traget variable in test set
0    30
1    24
Name: class, dtype: int64
```

In [12]:

```
print('------------Training Set------------------')
print(X_train.shape)
print(y_train.shape)

print('------------Test Set------------------')
print(X_test.shape)
print(y_test.shape)
```

```
------------Training Set------------------
(216, 13)
(216,)
------------Test Set------------------
(54, 13)
(54,)
```

In [13]:

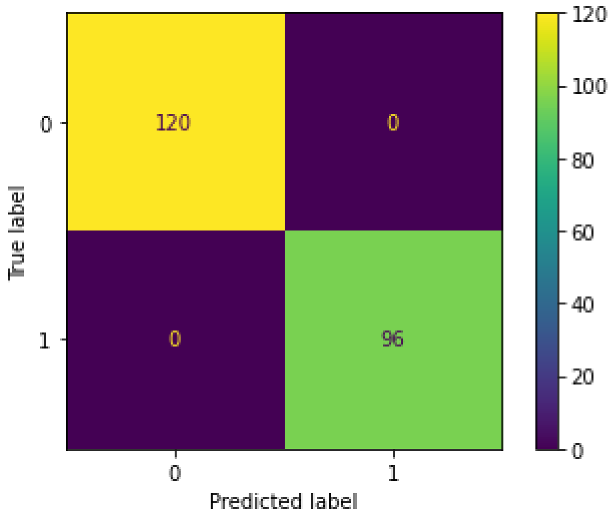
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score , plot_confusion_matrix , plot_roc_curve
from sklearn.model_selection import cross_val_score , GridSearchCV , KFold

dt = DecisionTreeClassifier(random_state=0)
dt.fit(X_train , y_train)
y_pred_train = dt.predict(X_train)
y_pred_test = dt.predict(X_test)
```

In [14]:

```
#overfits training data
plot_confusion_matrix(dt , X_train , y_train)
```

Out[14]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x22942341
c70>
```

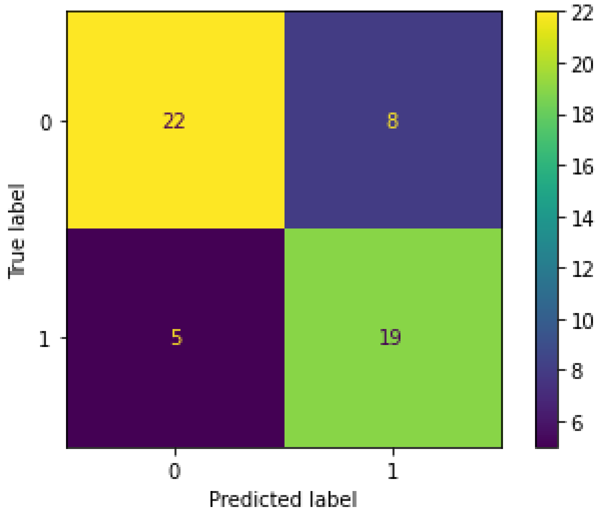


In [15]:

```
#underfits training data , we need to regularize the model
plot_confusion_matrix(dt , X_test , y_test)
```

Out[15]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x229426ca
970>
```

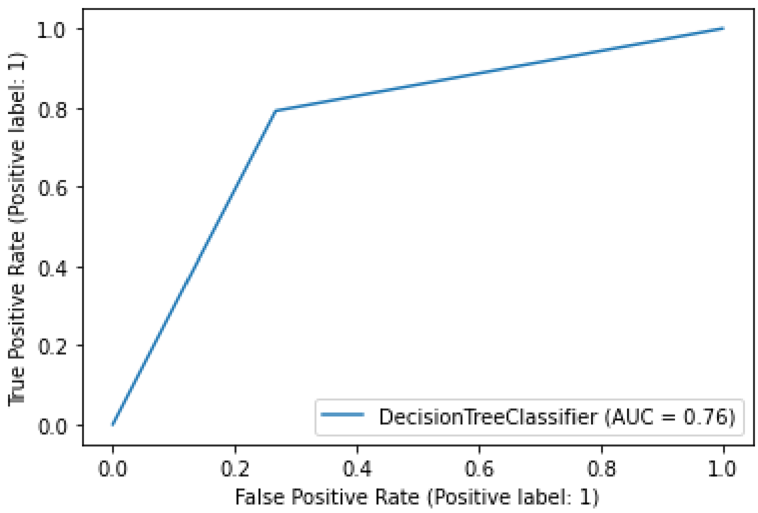


In [16]:

```
#Lowest
plot_roc_curve(dt , X_test , y_test)
```

Out[16]:

```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x22942331f70>
```



In [17]:

```
from sklearn import model_selection
scoring = 'accuracy'
kfold = model_selection.KFold(n_splits=10)
cv = model_selection.cross_val_score(DecisionTreeClassifier(), X_train, y_train, cv=kfold,

#use grid search cv for best params
params = {'max_depth':np.arange(2,20),'criterion':['gini','entropy']}
grid = GridSearchCV(dt , param_grid=params , cv=kfold , scoring='roc_auc')
grid.fit(scaled , y)
```

Out[17]:

```
GridSearchCV(cv=KFold(n_splits=10, random_state=None, shuffle=False),
             estimator=DecisionTreeClassifier(random_state=0),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': array([ 2,  3,  4,  5,  6,  7,  8,  9,
10, 11, 12, 13, 14, 15, 16, 17, 18,
       19])},
             scoring='roc_auc')
```

In [18]:

```
grid.best_params_
```

Out[18]:

```
{'criterion': 'entropy', 'max_depth': 3}
```

In [19]:

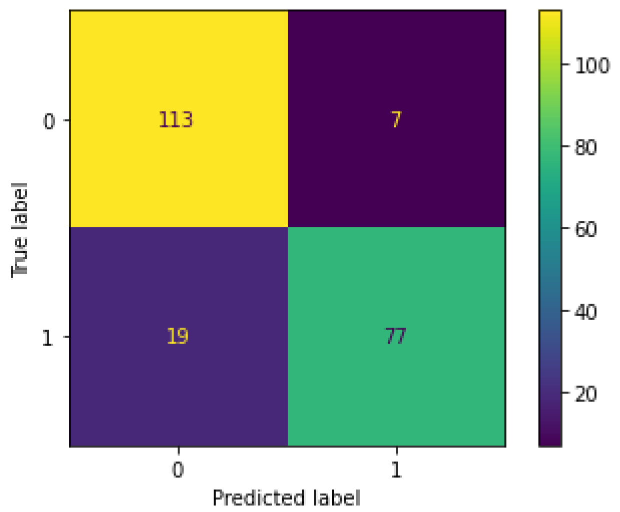
```
dt_reg = DecisionTreeClassifier(max_depth=3 , criterion='gini',random_state=0)
dt_reg.fit(X_train , y_train)
y_pred_train = dt_reg.predict(X_train)
y_pred_test = dt_reg.predict(X_test)
```

In [20]:

```
plot_confusion_matrix(dt_reg , X_train , y_train)
#better
```

Out[20]:

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2294078c
c70>

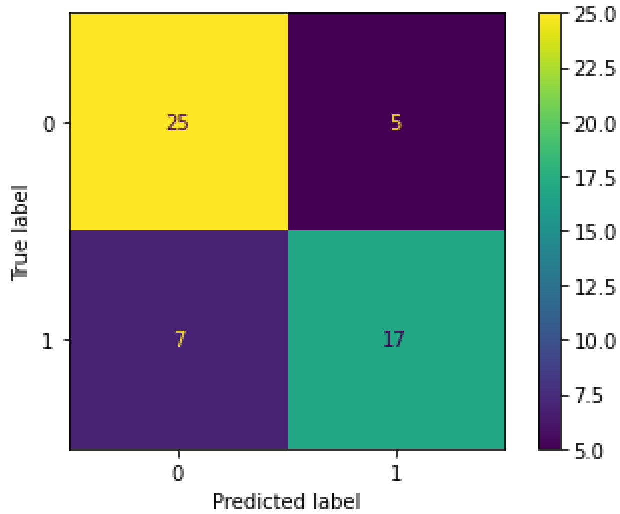


In [21]:

```
plot_confusion_matrix(dt_reg , X_test , y_test)
#underfits training data , we need to regularize the model
```

Out[21]:

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x22943189
400>

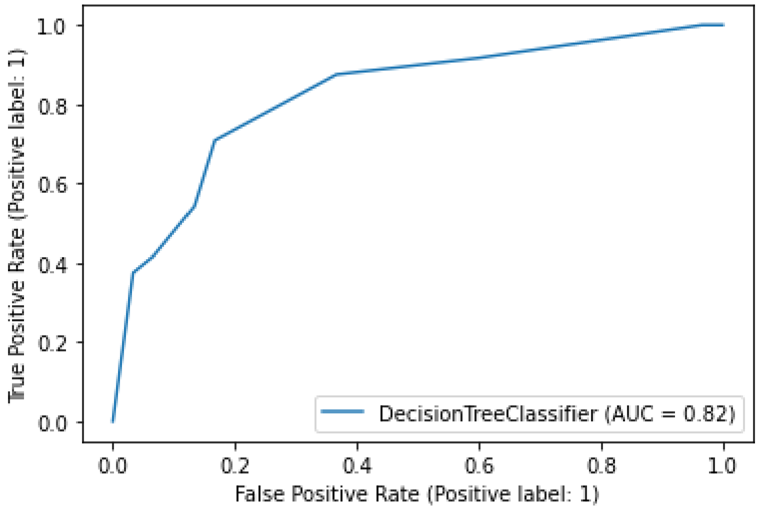


In [22]:

```
plot_roc_curve(dt_reg , X_test , y_test)
#Lowest
```

Out[22]:

<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x229432b5c40>

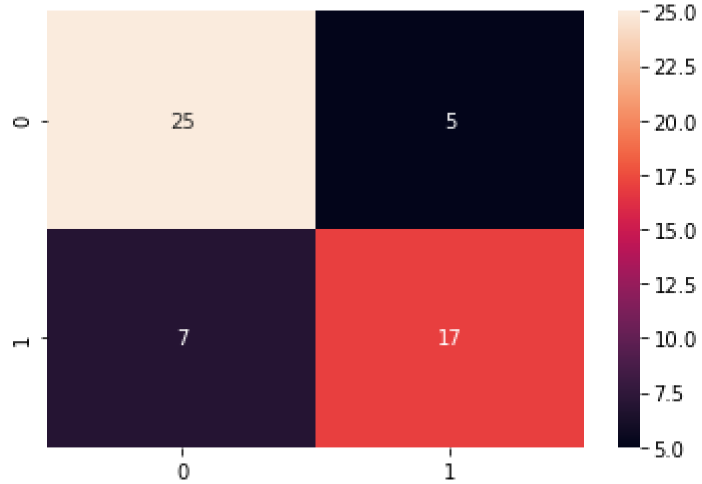


In [23]:

```
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import log_loss,roc_auc_score,precision_score,f1_score,recall_score,ro

CM=confusion_matrix(y_test,y_pred_test)
sns.heatmap(CM, annot=True)

TN = CM[0][0]
FN = CM[1][0]
TP = CM[1][1]
FP = CM[0][1]
specificity = TN/(TN+FP)
sensitivity = TP/(TP+FN)
loss_log = log_loss(y_test, y_pred_test)
acc= accuracy_score(y_test, y_pred_test)
roc=roc_auc_score(y_test, y_pred_test)
prec = precision_score(y_test, y_pred_test)
rec = recall_score(y_test, y_pred_test)
f1 = f1_score(y_test, y_pred_test)

model_results =pd.DataFrame([['Decision Tree', acc, prec, rec, f1, sensitivity, specificity
               columns = ['Model','Accuracy','Precision','Recall','f1 score','Sensitivity',

model_results
```

Out[23]:

	Model	Accuracy	Precision	Recall	f1 score	Sensitivity	Specificity	ROC	Log_Los
0	Decision Tree	0.777778	0.772727	0.708333	0.73913	0.708333	0.833333	0.770833	7.67535



In [24]:

```
#defining various steps required for the genetic algorithm
def initilization_of_population(size,n_feat):
    population = []
    for i in range(size):
        chromosome = np.ones(n_feat,dtype=np.bool)
        chromosome[:int(0.2*n_feat)]=False
        np.random.shuffle(chromosome)
        population.append(chromosome)
    return population

def fitness_score(population):
    scores = []
    for chromosome in population:
        dt_reg.fit(X_train.iloc[:,chromosome],y_train)
        y_pred_test = dt_reg.predict(X_test.iloc[:,chromosome])
        scores.append(accuracy_score(y_test,y_pred_test))
    scores, population = np.array(scores), np.array(population)
    inds = np.argsort(scores)
    return list(scores[inds][::-1]), list(population[inds,:][::-1])

def selection(pop_after_fit,n_parents):
    population_nextgen = []
    for i in range(n_parents):
        population_nextgen.append(pop_after_fit[i])
    return population_nextgen

def crossover(pop_after_sel):
    population_nextgen=pop_after_sel
    for i in range(len(pop_after_sel)):
        child=pop_after_sel[i]
        child[3:7]=pop_after_sel[(i+1)%len(pop_after_sel)][3:7]
        population_nextgen.append(child)
    return population_nextgen

def mutation(pop_after_cross,mutation_rate):
    population_nextgen = []
    for i in range(0,len(pop_after_cross)):
        chromosome = pop_after_cross[i]
        for j in range(len(chromosome)):
            if random.random() < mutation_rate:
                chromosome[j]= not chromosome[j]
        population_nextgen.append(chromosome)
    #print(population_nextgen)
    return population_nextgen

def generations(size,n_feat,n_parents,mutation_rate,n_gen,X_train,
                                   X_test, y_train, y_test):
    best_chromo= []
    best_score= []
    population_nextgen=initilization_of_population(size,n_feat)
    for i in range(n_gen):
        scores, pop_after_fit = fitness_score(population_nextgen)
        print(scores[:2])
        pop_after_sel = selection(pop_after_fit,n_parents)
        pop_after_cross = crossover(pop_after_sel)
        population_nextgen = mutation(pop_after_cross,mutation_rate)
        best_chromo.append(pop_after_fit[0])
        best_score.append(scores[0])
    return best_chromo,best_score
```

In [28]:

```
import random
chromo,score=generations(size=270,n_feat=13,n_parents=135,mutation_rate=0.10,
                     n_gen=38,X_train=X_train,X_test=X_test,y_train=y_train,y_test=y_test)
dt_reg.fit(X_train.iloc[:,chromo[-1]],y_train)
y_pred_test = dt_reg.predict(X_test.iloc[:,chromo[-1]])
print("Accuracy score after genetic algorithm is= "+str(accuracy_score(y_test,y_pred_test))
```

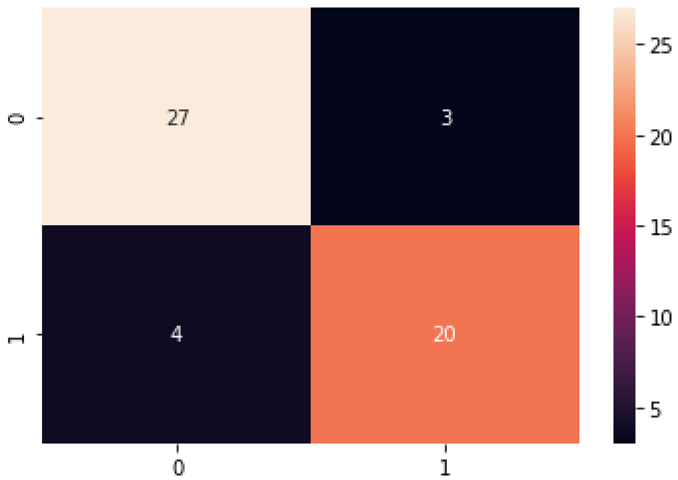
```
[0.8518518518518519, 0.8518518518518519]
[0.8518518518518519, 0.8518518518518519]
[0.8888888888888888, 0.8888888888888888]
[0.8888888888888888, 0.8888888888888888]
[0.8518518518518519, 0.8518518518518519]
[0.8703703703703703, 0.8703703703703703]
[0.8888888888888888, 0.8888888888888888]
[0.8703703703703703, 0.8703703703703703]
[0.8703703703703703, 0.8703703703703703]
[0.8888888888888888, 0.8888888888888888]
[0.8703703703703703, 0.8703703703703703]
[0.8518518518518519, 0.8518518518518519]
[0.8703703703703703, 0.8703703703703703]
[0.8518518518518519, 0.8518518518518519]
[0.8518518518518519, 0.8518518518518519]
[0.8518518518518519, 0.8518518518518519]
[0.8518518518518519, 0.8518518518518519]
[0.8703703703703703, 0.8703703703703703]
[0.8703703703703703, 0.8703703703703703]
[0.8518518518518519, 0.8518518518518519]
[0.8888888888888888, 0.8888888888888888]
[0.8888888888888888, 0.8888888888888888]
[0.8888888888888888, 0.8888888888888888]
[0.8888888888888888, 0.8888888888888888]
[0.8703703703703703, 0.8703703703703703]
[0.8703703703703703, 0.8703703703703703]
[0.8518518518518519, 0.8518518518518519]
[0.8518518518518519, 0.8518518518518519]
[0.8518518518518519, 0.8518518518518519]
[0.8703703703703703, 0.8703703703703703]
[0.8518518518518519, 0.8518518518518519]
[0.8888888888888888, 0.8888888888888888]
[0.8888888888888888, 0.8888888888888888]
[0.8518518518518519, 0.8518518518518519]
[0.8888888888888888, 0.8888888888888888]
[0.8518518518518519, 0.8518518518518519]
[0.8518518518518519, 0.8518518518518519]
[0.8703703703703703, 0.8703703703703703]
Accuracy score after genetic algorithm is= 0.8703703703703703
```

In [29]:

```
CM=confusion_matrix(y_test,y_pred_test)
sns.heatmap(CM, annot=True)

TN = CM[0][0]
FN = CM[1][0]
TP = CM[1][1]
FP = CM[0][1]
specificity = TN/(TN+FP)
sensitivity = TP/(TP+FN)
loss_log = log_loss(y_test, y_pred_test)
acc= accuracy_score(y_test, y_pred_test)
roc=roc_auc_score(y_test, y_pred_test)
prec = precision_score(y_test, y_pred_test)
rec = recall_score(y_test, y_pred_test)
f1 = f1_score(y_test, y_pred_test)

model_results =pd.DataFrame([['Decision Tree', acc, prec, rec, f1, sensitivity, specificity
               columns = ['Model','Accuracy','Precision','Recall','f1 score','Sensitivity',

model_results
```

Out[29]:

	Model	Accuracy	Precision	Recall	f1 score	Sensitivity	Specificity	ROC	Log_Lo
0	Decision Tree	0.87037	0.869565	0.833333	0.851064	0.833333	0.9	0.866667	4.4772



In [ ]: