In [1]:

```
import warnings
warnings.filterwarnings('ignore')

# data wrangling & pre-processing
import pandas as pd
import numpy as np

# data visualization
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

from sklearn.model_selection import train_test_split

#model validation
from sklearn.metrics import log_loss,roc_auc_score,precision_score,f1_score,recall_score,ro
from sklearn.metrics import classification_report, confusion_matrix,accuracy_score,fbeta_sc
from sklearn import metrics

# cross validation
from sklearn.model_selection import StratifiedKFold

# machine learning algorithms
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier,VotingClassifier,AdaBoostClassifier,Gra
from sklearn.neural_network import MLPClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.svm import SVC
#import xgboost as xgb
from scipy import stats
```

In [2]:

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

Out[2]:

	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 [3]:

```
df["class"].replace({"present": 1, "absent": 0}, inplace=True)
```

In [4]:

```
## Checking missing entries in the dataset columnwise
df.isna().sum()
```

Out[4]:

```
age                                     0
sex                                     0
chest                                   0
resting_blood_pressure                  0
serum_cholestoral                       0
fasting_blood_sugar                     0
resting_electrocardiographic_results    0
maximum_heart_rate_achieved             0
exercise_induced_angina                 0
oldpeak                                 0
slope                                   0
number_of_major_vessels                 0
thal                                    0
class                                   0
dtype: int64
```

In [5]:

```
# segregating dataset into features i.e., X and target variables i.e., y
X = df.drop(columns=['class'])
y = df['class']

X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.2,shuffle
```

In [6]:

```
df.head()
```

Out[6]:

	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 [7]:

```
## 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 [8]:

```
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 [9]:

```
from sklearn import model_selection

ada = AdaBoostClassifier(n_estimators=270)
ada.fit(X_train,y_train)
y_pred_ada = ada.predict(X_test)

scoring = 'accuracy'
results = []
kfold = model_selection.KFold(n_splits=10)
cv_results = model_selection.cross_val_score(AdaBoostClassifier(n_estimators=270), X_train,
results.append(cv_results)
#names.append(name)
msg = "%s: %f (%f)" % ('AB', cv_results.mean(), cv_results.std())
print(msg)
```

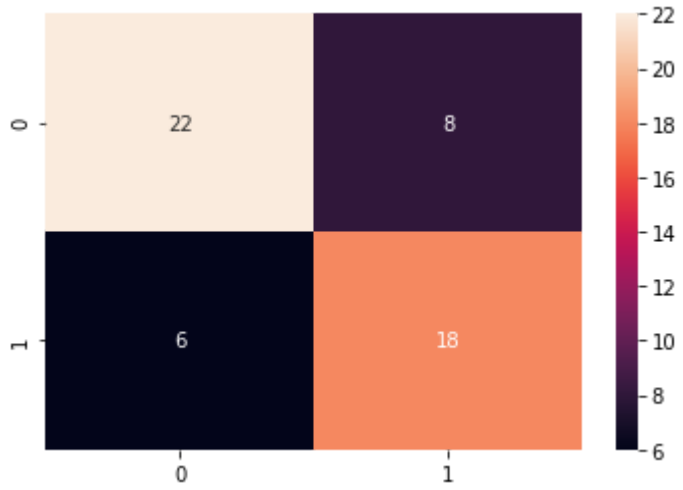
```
AB: 0.809957 (0.081678)
```

In [10]:

```
CM=confusion_matrix(y_test,y_pred_ada)
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_ada)
acc= accuracy_score(y_test, y_pred_ada)
roc=roc_auc_score(y_test, y_pred_ada)
prec = precision_score(y_test, y_pred_ada)
rec = recall_score(y_test, y_pred_ada)
f1 = f1_score(y_test, y_pred_ada)

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

model_results
```

Out[10]:

	Model	Accuracy	Precision	Recall	f1 score	Sensitivity	Specificity	ROC	Log_Loss
0	AdaBoost	0.740741	0.692308	0.75	0.72	0.75	0.733333	0.741667	8.954616



In [11]:

```
print(classification_report(y_test, y_pred_ada))
```

```
              precision    recall  f1-score   support

           0       0.79      0.73      0.76        30
           1       0.69      0.75      0.72        24

    accuracy                           0.74        54
   macro avg       0.74      0.74      0.74        54
weighted avg       0.74      0.74      0.74        54
```

In [12]:

```
#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:
        ada.fit(X_train.iloc[:,chromosome],y_train)
        y_pred_ada = ada.predict(X_test.iloc[:,chromosome])
        scores.append(accuracy_score(y_test,y_pred_ada))
    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 [13]:

```
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)
ada.fit(X_train.iloc[:,chromo[-1]],y_train)
y_pred_ada = ada.predict(X_test.iloc[:,chromo[-1]])
print("Accuracy score after genetic algorithm is= "+str(accuracy_score(y_test,y_pred_ada)))
```

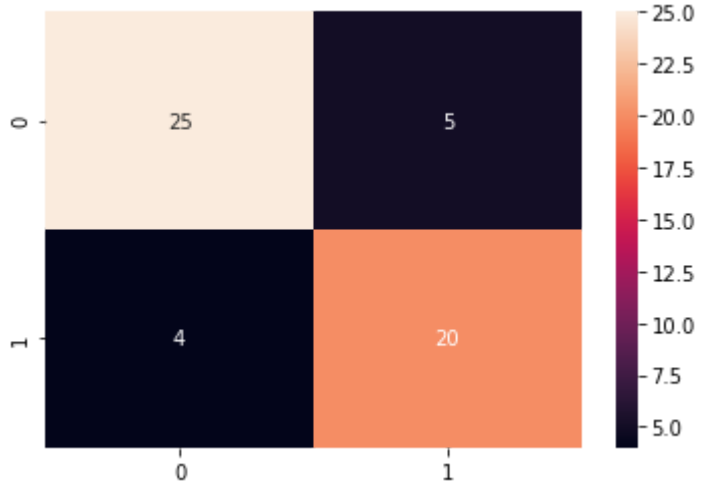
```
[0.8148148148148148, 0.8148148148148148]
[0.8333333333333334, 0.8333333333333334]
[0.8518518518518519, 0.8518518518518519]
[0.8518518518518519, 0.8518518518518519]
[0.8518518518518519, 0.8518518518518519]
[0.8703703703703703, 0.8703703703703703]
[0.8518518518518519, 0.8518518518518519]
[0.8703703703703703, 0.8703703703703703]
[0.8703703703703703, 0.8703703703703703]
[0.8518518518518519, 0.8518518518518519]
[0.8518518518518519, 0.8518518518518519]
[0.8333333333333334, 0.8333333333333334]
[0.8703703703703703, 0.8703703703703703]
[0.8703703703703703, 0.8703703703703703]
[0.8703703703703703, 0.8703703703703703]
[0.8518518518518519, 0.8518518518518519]
[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]
[0.8333333333333334, 0.8333333333333334]
[0.8333333333333334, 0.8333333333333334]
[0.8518518518518519, 0.8518518518518519]
[0.8518518518518519, 0.8518518518518519]
[0.8518518518518519, 0.8518518518518519]
[0.8703703703703703, 0.8703703703703703]
[0.8518518518518519, 0.8518518518518519]
[0.8518518518518519, 0.8518518518518519]
[0.8333333333333334, 0.8333333333333334]
[0.8703703703703703, 0.8703703703703703]
[0.8333333333333334, 0.8333333333333334]
[0.8518518518518519, 0.8518518518518519]
[0.8518518518518519, 0.8518518518518519]
[0.8703703703703703, 0.8703703703703703]
[0.8518518518518519, 0.8518518518518519]
[0.8518518518518519, 0.8518518518518519]
Accuracy score after genetic algorithm is= 0.8333333333333334
```

In [15]:

```
CM=confusion_matrix(y_test,y_pred_ada)
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_ada)
acc= accuracy_score(y_test, y_pred_ada)
roc=roc_auc_score(y_test, y_pred_ada)
prec = precision_score(y_test, y_pred_ada)
rec = recall_score(y_test, y_pred_ada)
f1 = f1_score(y_test, y_pred_ada)

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

model_results
```

Out[15]:

	Model	Accuracy	Precision	Recall	f1 score	Sensitivity	Specificity	ROC	Log_L
0	AdaBoost	0.833333	0.8	0.833333	0.816327	0.833333	0.833333	0.833333	5.756



In [16]:

```
print(classification_report(y_test, y_pred_ada))
```

```
              precision    recall  f1-score   support

           0       0.86      0.83      0.85        30
           1       0.80      0.83      0.82        24

    accuracy                           0.83        54
   macro avg       0.83      0.83      0.83        54
weighted avg       0.83      0.83      0.83        54
```

In [ ]:

```

```