In [1]:

```
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
import random
import matplotlib.pyplot
%matplotlib inline
from sklearn.linear_model import LogisticRegression
```

In [2]:

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,classif
# cross validation
from sklearn.model_selection import StratifiedKFold

from sklearn import metrics
```

In [3]:

```
#import the heart disease dataset
df = pd.read_csv("heart_statlog.csv")
df
```

Out[3]:

	age	sex	chest	resting_blood_pressure	serum_cholestoral	fasting_blood_sugar	resting_e
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	
...	...	...	...	...	...	...	
265	52	1	3	172	199	1	
266	44	1	2	120	263	0	
267	56	0	2	140	294	0	
268	57	1	4	140	192	0	
269	67	1	4	160	286	0	

270 rows × 14 columns

In [4]:

```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
df['class'] = le.fit_transform(df["class"])
df
```

Out[4]:

	age	sex	chest	resting_blood_pressure	serum_cholestoral	fasting_blood_sugar	resting_
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	
...	...	...	...	...	...	...	
265	52	1	3	172	199	1	
266	44	1	2	120	263	0	
267	56	0	2	140	294	0	
268	57	1	4	140	192	0	
269	67	1	4	160	286	0	

270 rows × 14 columns

In [5]:

```
# segregating dataset into features i.e., X and target variables i.e., y
X = df.drop(['class'],axis=1)
y = df['class']
```

In [6]:

```
#splitting the model into training and testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.2,shuffle
```

In [7]:

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train[['age','resting_blood_pressure','serum_cholestoral','maximum_heart_rate_achieved','
X_train.head()
```

Out[7]:

	age	sex	chest	resting_blood_pressure	serum_cholestoral	fasting_blood_sugar	rest
197	0.465116	0	3	0.10	0.302405	0	
174	0.000000	1	1	0.18	0.192440	0	
173	0.790698	0	3	0.20	0.292096	0	
249	0.604651	1	4	0.45	0.536082	0	
207	0.558140	1	3	0.05	0.391753	0	

In [8]:

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_test[['age','resting_blood_pressure','serum_cholestoral','maximum_heart_rate_achieved','o
X_test.head()
```

Out[8]:

	age	sex	chest	resting_blood_pressure	serum_cholestoral	fasting_blood_sugar	rest
260	0.617021	0	3	0.309524	0.460241	0	
194	0.404255	1	3	0.357143	0.255422	1	
112	0.659574	0	4	0.761905	0.375904	0	
14	0.595745	0	4	0.404762	0.371084	0	
206	0.659574	0	3	0.095238	0.407229	0	

In [9]:

```
from sklearn import model_selection
from sklearn.model_selection import cross_val_score

# function initializing baseline machine learning models
def GetBasedModel():
    basedModels = []
    basedModels.append(('LR'  ,LogisticRegression()))
    return basedModels

# function for performing 10-fold cross validation of all the baseline models
def BasedLine2(X_train, y_train,models):
    # Test options and evaluation metric
    num_folds = 10
    scoring = 'accuracy'
    seed = 7
    results = []
    names = []
    for name, model in models:
        kfold = model_selection.KFold(n_splits=10, random_state=None)
        cv_results = model_selection.cross_val_score(model, X_train, y_train, cv=kfold, sco
        results.append(cv_results)
        names.append(name)
        msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
        print(msg)


    return results,msg
```

In [10]:

```
models = GetBasedModel()
names,results = BasedLine2(X_train, y_train,models)
```

```
LR: 0.826840 (0.115528)
```

In [11]:

```
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
y_pred_logreg = logreg.predict(X_test)
```

In [12]:

```
CM=confusion_matrix(y_test,y_pred_logreg)

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

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

model_results
```

Out[12]:

	Model	Accuracy	Precision	Recall	f1 score	Sensitivity	Specificity
0	LR	0.851852	0.807692	0.875	0.84	0.875	0.833333

In [13]:

```
#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(0.3*n_feat)]=False
        np.random.shuffle(chromosome)
        population.append(chromosome)
    return population

def fitness_score(population):
    scores = []
    for chromosome in population:
        logreg.fit(X_train.iloc[:,chromosome],y_train)
        y_pred_logreg = logreg.predict(X_test.iloc[:,chromosome])
        scores.append(accuracy_score(y_test,y_pred_logreg))
    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 [17]:

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

```
[0.9259259259259259, 0.9074074074074074]
[0.8888888888888888, 0.8888888888888888]
[0.8888888888888888, 0.8888888888888888]
[0.9259259259259259, 0.9259259259259259]
[0.8888888888888888, 0.8888888888888888]
[0.9259259259259259, 0.9259259259259259]
[0.8888888888888888, 0.8888888888888888]
[0.8888888888888888, 0.8888888888888888]
[0.8888888888888888, 0.8888888888888888]
[0.8888888888888888, 0.8888888888888888]
[0.8888888888888888, 0.8888888888888888]
[0.9074074074074074, 0.9074074074074074]
[0.8888888888888888, 0.8888888888888888]
[0.8888888888888888, 0.8888888888888888]
[0.8888888888888888, 0.8888888888888888]
[0.8888888888888888, 0.8888888888888888]
[0.8888888888888888, 0.8888888888888888]
[0.8888888888888888, 0.8888888888888888]
[0.8888888888888888, 0.8888888888888888]
[0.8888888888888888, 0.8888888888888888]
[0.8888888888888888, 0.8888888888888888]
[0.8888888888888888, 0.8888888888888888]
[0.9074074074074074, 0.9074074074074074]
[0.8888888888888888, 0.8888888888888888]
[0.9074074074074074, 0.9074074074074074]
[0.8888888888888888, 0.8888888888888888]
[0.8888888888888888, 0.8888888888888888]
[0.8888888888888888, 0.8888888888888888]
[0.8888888888888888, 0.8888888888888888]
[0.8888888888888888, 0.8888888888888888]
[0.9074074074074074, 0.9074074074074074]
[0.8888888888888888, 0.8888888888888888]
[0.8888888888888888, 0.8888888888888888]
[0.8888888888888888, 0.8888888888888888]
[0.8888888888888888, 0.8888888888888888]
[0.8888888888888888, 0.8888888888888888]
[0.9074074074074074, 0.9074074074074074]
[0.9074074074074074, 0.9074074074074074]
Accuracy score after genetic algorithm is= 0.8703703703703703
```

In [18]:

```
CM=confusion_matrix(y_test,y_pred_logreg)

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

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

model_results
```

Out[18]:

	Model	Accuracy	Precision	Recall	f1 score	Sensitivity	Specificity
0	LR	0.87037	0.814815	0.916667	0.862745	0.916667	0.833333