import numpy as pdppdpdd

import pandas as pdpd

import seaborn as sns

import matplotlib.pyplot as plt

from random import randint

%matplotlib inline

import warnings

warnings.filterwarnings("ignore")

from sklearn.model\_selection import train\_test\_split

def split(df,label):

    X\_tr, X\_te, Y\_tr, Y\_te = train\_test\_split(df, label, test\_size=0.25, random\_state=42)

    return X\_tr, X\_te, Y\_tr, Y\_te

from sklearn import svm

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.ensemble import GradientBoostingClassifier

from sklearn import metrics

from sklearn.metrics import accuracy\_score

from sklearn.model\_selection import KFold, cross\_val\_score

classifiers = ['LinearSVM', 'RadialSVM',

               'Logistic',  'RandomForest',

               'AdaBoost',  'DecisionTree',

               'KNeighbors','GradientBoosting']

models = [svm.SVC(kernel='linear'),

          svm.SVC(kernel='rbf'),

          LogisticRegression(max\_iter = 1000),

          RandomForestClassifier(n\_estimators=200, random\_state=0),

          AdaBoostClassifier(random\_state = 0),

          DecisionTreeClassifier(random\_state=0),

          KNeighborsClassifier(),

          GradientBoostingClassifier(random\_state=0)]

def acc\_score(df,label):

    Score = pd.DataFrame({"Classifier":classifiers})

    j = 0

    acc = []

    X\_train,X\_test,Y\_train,Y\_test = split(df,label)

    for i in models:

        model = i

        model.fit(X\_train,Y\_train)

        predictions = model.predict(X\_test)

        acc.append(accuracy\_score(Y\_test,predictions))

        j = j+1

    Score["Accuracy"] = acc

    Score.sort\_values(by="Accuracy", ascending=False,inplace = True)

    Score.reset\_index(drop=True, inplace=True)

    return Score

def plot(score,x,y,c = "b"):

    gen = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115.]

    plt.figure(figsize=(25,5))

    ax = sns.pointplot(x=gen, y=score,color = c )

    ax.set(xlabel="Generation", ylabel="Accuracy")

    ax.set(ylim=(x,y))

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:

        logmodel.fit(X\_train.iloc[:,chromosome],Y\_train)

        predictions = logmodel.predict(X\_test.iloc[:,chromosome])

        scores.append(accuracy\_score(Y\_test,predictions))

    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):

    pop\_nextgen = pop\_after\_sel

    for i in range(0,len(pop\_after\_sel),2):

        new\_par = []

        child\_1 , child\_2 = pop\_nextgen[i] , pop\_nextgen[i+1]

        new\_par = np.concatenate((child\_1[:len(child\_1)//2],child\_2[len(child\_1)//2:]))

        pop\_nextgen.append(new\_par)

    return pop\_nextgen

def mutation(pop\_after\_cross,mutation\_rate,n\_feat):

    mutation\_range = int(mutation\_rate\*n\_feat)

    pop\_next\_gen = []

    for n in range(0,len(pop\_after\_cross)):

        chromo = pop\_after\_cross[n]

        rand\_posi = []

        for i in range(0,mutation\_range):

            pos = randint(0,n\_feat-1)

            rand\_posi.append(pos)

        for j in rand\_posi:

            chromo[j] = not chromo[j]

        pop\_next\_gen.append(chromo)

    return pop\_next\_gen

def generations(df,label,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('Best score in generation',i+1,':',scores[:1])  #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,n\_feat)

        best\_chromo.append(pop\_after\_fit[0])

        best\_score.append(scores[0])

    return best\_chromo,best\_score

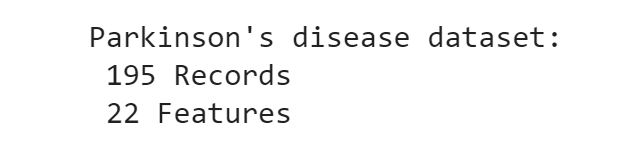
import pandas as pd

data\_pd = pd.read\_csv("/content/Parkinsson disease.csv")

label\_pd = data\_pd["status"]

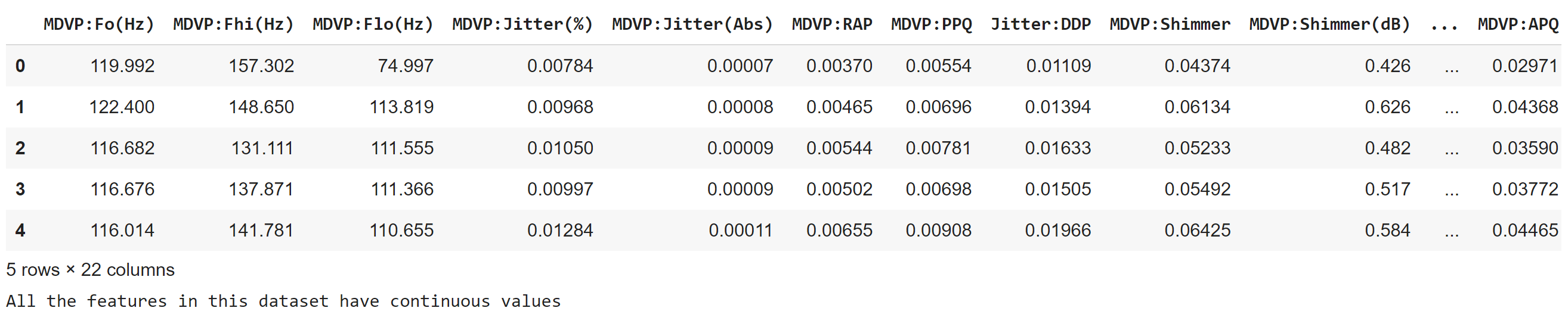
data\_pd.drop(["status","name"],axis = 1,inplace = True)

print("Parkinson's disease dataset:\n",data\_pd.shape[0],"Records\n",data\_pd.shape[1],"Features")



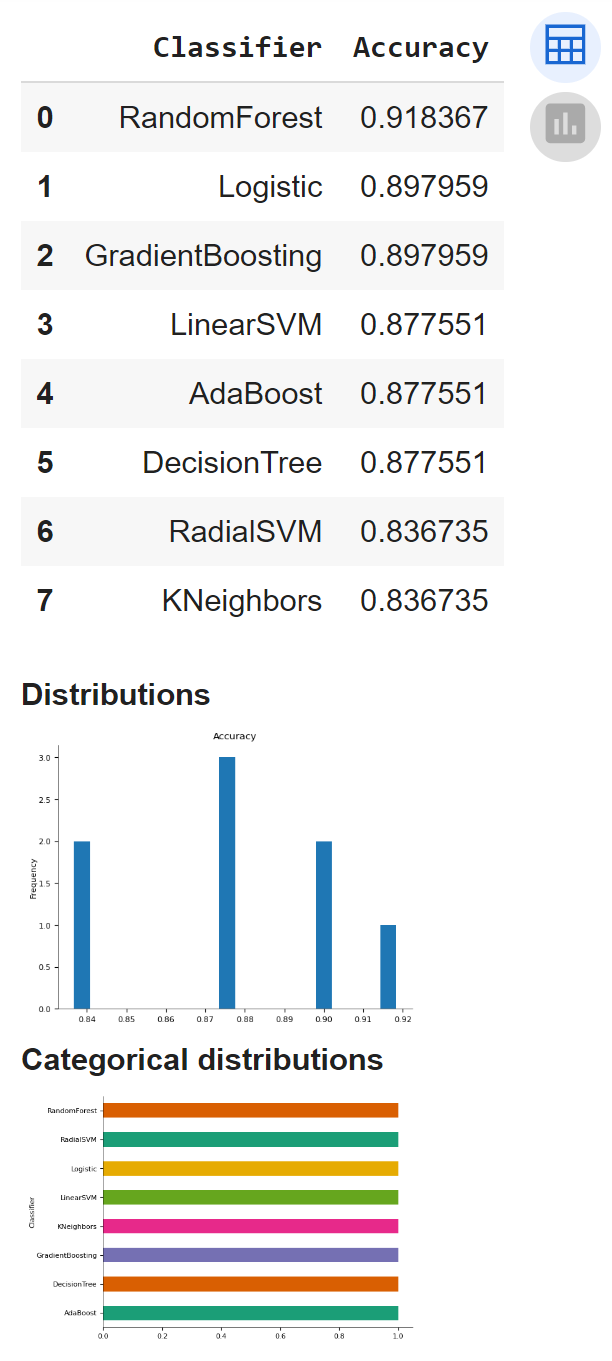
display(data\_pd.head())

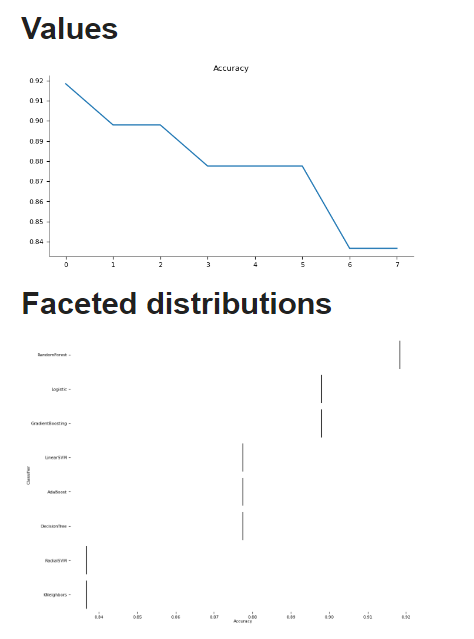
print("All the features in this dataset have continuous values")



score3 = acc\_score(data\_pd,label\_pd)

score3





Decision Tree

import numpy as np

logmodel = DecisionTreeClassifier(random\_state=0)

X\_train,X\_test, Y\_train, Y\_test = split(data\_pd,label\_pd)

chromo\_df\_pd,score\_pd=generations(data\_pd,label\_pd,size=80,n\_feat=data\_pd.shape[1],n\_parents=75,mutation\_rate=0.40,n\_gen=115,

                         X\_train = X\_train,X\_test = X\_test,Y\_train = Y\_train,Y\_test = Y\_test)

Best score in generation 1 : [0.9795918367346939]

Best score in generation 2 : [0.9591836734693877]

Best score in generation 3 : [0.9591836734693877]

Best score in generation 4 : [0.9591836734693877]

Best score in generation 5 : [0.9795918367346939]

Best score in generation 6 : [0.9795918367346939]

Best score in generation 7 : [0.9795918367346939]

Best score in generation 8 : [0.9591836734693877]

Best score in generation 9 : [0.9591836734693877]

Best score in generation 10 : [0.9591836734693877]

Best score in generation 11 : [0.9591836734693877]

Best score in generation 12 : [0.9795918367346939]

Best score in generation 13 : [0.9795918367346939]

Best score in generation 14 : [0.9795918367346939]

Best score in generation 15 : [0.9591836734693877]

Best score in generation 16 : [0.9795918367346939]

Best score in generation 17 : [0.9591836734693877]

Best score in generation 18 : [0.9795918367346939]

Best score in generation 19 : [0.9591836734693877]

Best score in generation 20 : [0.9795918367346939]

Best score in generation 21 : [0.9795918367346939]

Best score in generation 22 : [0.9795918367346939]

Best score in generation 23 : [0.9591836734693877]

Best score in generation 24 : [0.9795918367346939]

Best score in generation 25 : [0.9795918367346939]

Best score in generation 26 : [1.0]

Best score in generation 27 : [0.9591836734693877]

Best score in generation 28 : [0.9591836734693877]

Best score in generation 29 : [0.9795918367346939]

Best score in generation 30 : [0.9795918367346939]

Best score in generation 31 : [0.9387755102040817]

Best score in generation 32 : [0.9795918367346939]

Best score in generation 33 : [0.9795918367346939]

Best score in generation 34 : [0.9591836734693877]

Best score in generation 35 : [0.9795918367346939]

Best score in generation 36 : [0.9795918367346939]

Best score in generation 37 : [0.9591836734693877]

Best score in generation 38 : [0.9591836734693877]

Best score in generation 39 : [0.9591836734693877]

Best score in generation 40 : [0.9795918367346939]

Best score in generation 41 : [0.9591836734693877]

Best score in generation 42 : [0.9795918367346939]

Best score in generation 43 : [0.9591836734693877]

Best score in generation 44 : [0.9591836734693877]

Best score in generation 45 : [0.9591836734693877]

Best score in generation 46 : [0.9795918367346939]

Best score in generation 47 : [0.9795918367346939]

Best score in generation 48 : [0.9591836734693877]

Best score in generation 49 : [0.9795918367346939]

Best score in generation 50 : [0.9795918367346939]

Best score in generation 51 : [0.9591836734693877]

Best score in generation 52 : [0.9591836734693877]

Best score in generation 53 : [1.0]

Best score in generation 54 : [0.9795918367346939]

Best score in generation 55 : [0.9795918367346939]

Best score in generation 56 : [0.9795918367346939]

Best score in generation 57 : [0.9795918367346939]

Best score in generation 58 : [1.0]

Best score in generation 59 : [0.9795918367346939]

Best score in generation 60 : [0.9795918367346939]

Best score in generation 61 : [0.9795918367346939]

Best score in generation 62 : [0.9591836734693877]

Best score in generation 63 : [0.9795918367346939]

Best score in generation 64 : [0.9795918367346939]

Best score in generation 65 : [0.9591836734693877]

Best score in generation 66 : [0.9591836734693877]

Best score in generation 67 : [1.0]

Best score in generation 68 : [0.9795918367346939]

Best score in generation 69 : [0.9591836734693877]

Best score in generation 70 : [0.9795918367346939]

Best score in generation 71 : [0.9795918367346939]

Best score in generation 72 : [0.9591836734693877]

Best score in generation 73 : [0.9795918367346939]

Best score in generation 74 : [0.9795918367346939]

Best score in generation 75 : [0.9591836734693877]

Best score in generation 76 : [1.0]

Best score in generation 77 : [0.9795918367346939]

Best score in generation 78 : [0.9591836734693877]

Best score in generation 79 : [0.9795918367346939]

Best score in generation 80 : [0.9591836734693877]

Best score in generation 81 : [0.9795918367346939]

Best score in generation 82 : [0.9795918367346939]

Best score in generation 83 : [0.9795918367346939]

Best score in generation 84 : [0.9795918367346939]

Best score in generation 85 : [0.9795918367346939]

Best score in generation 86 : [0.9591836734693877]

Best score in generation 87 : [0.9795918367346939]

Best score in generation 88 : [0.9795918367346939]

Best score in generation 89 : [0.9591836734693877]

Best score in generation 90 : [0.9795918367346939]

Best score in generation 91 : [0.9795918367346939]

Best score in generation 92 : [0.9591836734693877]

Best score in generation 93 : [0.9795918367346939]

Best score in generation 94 : [0.9795918367346939]

Best score in generation 95 : [0.9591836734693877]

Best score in generation 96 : [0.9591836734693877]

Best score in generation 97 : [0.9795918367346939]

Best score in generation 98 : [0.9795918367346939]

Best score in generation 99 : [0.9795918367346939]

Best score in generation 100 : [0.9795918367346939]

Best score in generation 101 : [0.9795918367346939]

Best score in generation 102 : [0.9795918367346939]

Best score in generation 103 : [0.9591836734693877]

Best score in generation 104 : [0.9591836734693877]

Best score in generation 105 : [0.9591836734693877]

Best score in generation 106 : [0.9591836734693877]

Best score in generation 107 : [0.9387755102040817]

Best score in generation 108 : [0.9795918367346939]

Best score in generation 109 : [0.9795918367346939]

Best score in generation 110 : [0.9591836734693877]

Best score in generation 111 : [0.9591836734693877]

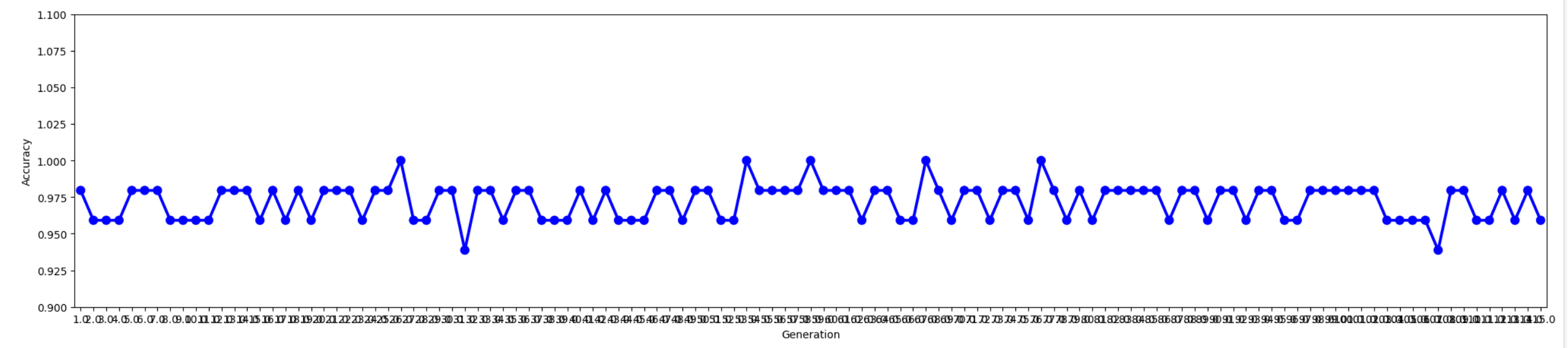
Best score in generation 112 : [0.9795918367346939]

Best score in generation 113 : [0.9591836734693877]

Best score in generation 114 : [0.9795918367346939]

Best score in generation 115 : [0.9591836734693877]

plot(score\_pd,0.9,1.1,c = "blue")



Linier SVM

import numpy as np

from sklearn.svm import LinearSVC

from sklearn.model\_selection import train\_test\_split

# Assuming the rest of your code is defined, including the 'generations' function

# Replace DecisionTreeClassifier with LinearSVC

logmodel = LinearSVC(random\_state=0)

# Assuming split function is defined

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(data\_pd, label\_pd, test\_size=0.2, random\_state=0)

# Assuming generations function is defined

chromo\_df\_pd, score\_pd = generations(data\_pd, label\_pd, size=80, n\_feat=data\_pd.shape[1], n\_parents=75, mutation\_rate=0.40, n\_gen=115,

                                     X\_train=X\_train, X\_test=X\_test, Y\_train=Y\_train, Y\_test=Y\_test)

Best score in generation 1 : [0.8461538461538461]

Best score in generation 2 : [0.8461538461538461]

Best score in generation 3 : [0.8974358974358975]

Best score in generation 4 : [0.8974358974358975]

Best score in generation 5 : [0.9230769230769231]

Best score in generation 6 : [0.9230769230769231]

Best score in generation 7 : [0.8974358974358975]

Best score in generation 8 : [0.8717948717948718]

Best score in generation 9 : [0.9230769230769231]

Best score in generation 10 : [0.9230769230769231]

Best score in generation 11 : [0.9230769230769231]

Best score in generation 12 : [0.9230769230769231]

Best score in generation 13 : [0.9230769230769231]

Best score in generation 14 : [0.8974358974358975]

Best score in generation 15 : [0.9230769230769231]

Best score in generation 16 : [0.8974358974358975]

Best score in generation 17 : [0.9230769230769231]

Best score in generation 18 : [0.8974358974358975]

Best score in generation 19 : [0.8974358974358975]

Best score in generation 20 : [0.8974358974358975]

Best score in generation 21 : [0.9230769230769231]

Best score in generation 22 : [0.8974358974358975]

Best score in generation 23 : [0.8974358974358975]

Best score in generation 24 : [0.9230769230769231]

Best score in generation 25 : [0.9230769230769231]

Best score in generation 26 : [0.9230769230769231]

Best score in generation 27 : [0.8974358974358975]

Best score in generation 28 : [0.8717948717948718]

Best score in generation 29 : [0.8974358974358975]

Best score in generation 30 : [0.8974358974358975]

Best score in generation 31 : [0.8717948717948718]

Best score in generation 32 : [0.8717948717948718]

Best score in generation 33 : [0.9230769230769231]

Best score in generation 34 : [0.8974358974358975]

Best score in generation 35 : [0.8974358974358975]

Best score in generation 36 : [0.8974358974358975]

Best score in generation 37 : [0.9230769230769231]

Best score in generation 38 : [0.9230769230769231]

Best score in generation 39 : [0.9230769230769231]

Best score in generation 40 : [0.8974358974358975]

Best score in generation 41 : [0.8974358974358975]

Best score in generation 42 : [0.8974358974358975]

Best score in generation 43 : [0.9230769230769231]

Best score in generation 44 : [0.9230769230769231]

Best score in generation 45 : [0.9230769230769231]

Best score in generation 46 : [0.8974358974358975]

Best score in generation 47 : [0.9230769230769231]

Best score in generation 48 : [0.9230769230769231]

Best score in generation 49 : [0.8717948717948718]

Best score in generation 50 : [0.8974358974358975]

Best score in generation 51 : [0.9230769230769231]

Best score in generation 52 : [0.8974358974358975]

Best score in generation 53 : [0.8974358974358975]

Best score in generation 54 : [0.8974358974358975]

Best score in generation 55 : [0.8974358974358975]

Best score in generation 56 : [0.8974358974358975]

Best score in generation 57 : [0.9230769230769231]

Best score in generation 58 : [0.8974358974358975]

Best score in generation 59 : [0.9230769230769231]

Best score in generation 60 : [0.9230769230769231]

Best score in generation 61 : [0.8974358974358975]

Best score in generation 62 : [0.9230769230769231]

Best score in generation 63 : [0.9230769230769231]

Best score in generation 64 : [0.9230769230769231]

Best score in generation 65 : [0.8717948717948718]

Best score in generation 66 : [0.8974358974358975]

Best score in generation 67 : [0.9230769230769231]

Best score in generation 68 : [0.9230769230769231]

Best score in generation 69 : [0.8974358974358975]

Best score in generation 70 : [0.8974358974358975]

Best score in generation 71 : [0.9230769230769231]

Best score in generation 72 : [0.9230769230769231]

Best score in generation 73 : [0.8974358974358975]

Best score in generation 74 : [0.8974358974358975]

Best score in generation 75 : [0.8974358974358975]

Best score in generation 76 : [0.8974358974358975]

Best score in generation 77 : [0.8974358974358975]

Best score in generation 78 : [0.9230769230769231]

Best score in generation 79 : [0.8974358974358975]

Best score in generation 80 : [0.9230769230769231]

Best score in generation 81 : [0.9230769230769231]

Best score in generation 82 : [0.8974358974358975]

Best score in generation 83 : [0.9230769230769231]

Best score in generation 84 : [0.9230769230769231]

Best score in generation 85 : [0.9230769230769231]

Best score in generation 86 : [0.8974358974358975]

Best score in generation 87 : [0.9230769230769231]

Best score in generation 88 : [0.8974358974358975]

Best score in generation 89 : [0.9230769230769231]

Best score in generation 90 : [0.8974358974358975]

Best score in generation 91 : [0.8717948717948718]

Best score in generation 92 : [0.9230769230769231]

Best score in generation 93 : [0.9230769230769231]

Best score in generation 94 : [0.8974358974358975]

Best score in generation 95 : [0.8974358974358975]

Best score in generation 96 : [0.9230769230769231]

Best score in generation 97 : [0.8974358974358975]

Best score in generation 98 : [0.9230769230769231]

Best score in generation 99 : [0.9230769230769231]

Best score in generation 100 : [0.9230769230769231]

Best score in generation 101 : [0.8974358974358975]

Best score in generation 102 : [0.9230769230769231]

Best score in generation 103 : [0.8974358974358975]

Best score in generation 104 : [0.8974358974358975]

Best score in generation 105 : [0.8974358974358975]

Best score in generation 106 : [0.9230769230769231]

Best score in generation 107 : [0.9230769230769231]

Best score in generation 108 : [0.9230769230769231]

Best score in generation 109 : [0.8974358974358975]

Best score in generation 110 : [0.9230769230769231]

Best score in generation 111 : [0.9230769230769231]

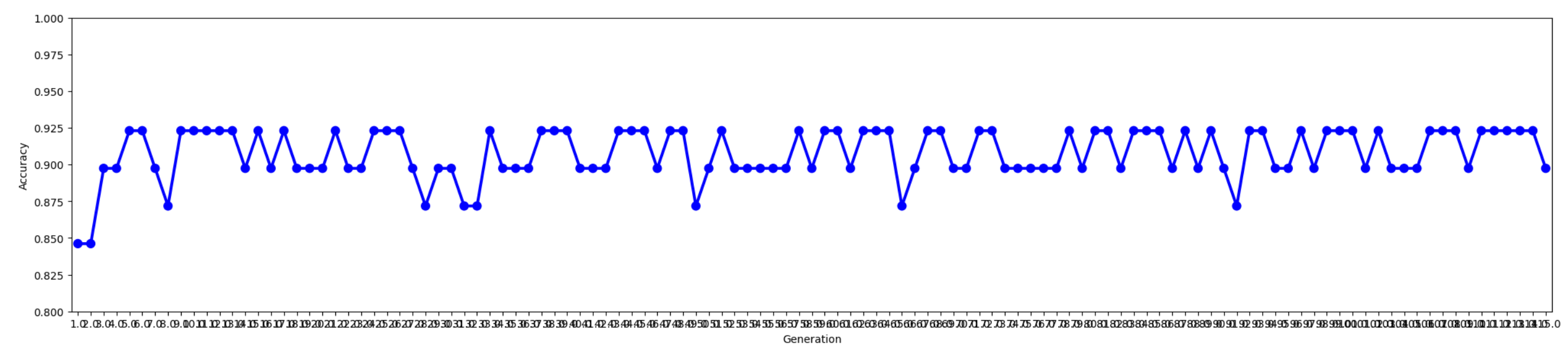
Best score in generation 112 : [0.9230769230769231]

Best score in generation 113 : [0.9230769230769231]

Best score in generation 114 : [0.9230769230769231]

Best score in generation 115 : [0.8974358974358975]

plot(score\_pd,0.8,1.0,c = "blue")



Random Forest

import numpy as np

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import train\_test\_split

# Assuming the rest of your code is defined, including the 'generations' function

# Replace DecisionTreeClassifier with RandomForestClassifier

logmodel = RandomForestClassifier(random\_state=0)

# Assuming split function is defined

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(data\_pd, label\_pd, test\_size=0.2, random\_state=0)

# Assuming generations function is defined

chromo\_df\_pd, score\_pd = generations(data\_pd, label\_pd, size=80, n\_feat=data\_pd.shape[1], n\_parents=75, mutation\_rate=0.40, n\_gen=10,

                                     X\_train=X\_train, X\_test=X\_test, Y\_train=Y\_train, Y\_test=Y\_test)

Best score in generation 1 : [0.9743589743589743]

Best score in generation 2 : [0.9743589743589743]

Best score in generation 3 : [0.9743589743589743]

Best score in generation 4 : [0.9743589743589743]

Best score in generation 5 : [0.9743589743589743]

Best score in generation 6 : [0.9743589743589743]

Best score in generation 7 : [0.9743589743589743]

Best score in generation 8 : [0.9743589743589743]

Best score in generation 9 : [0.9743589743589743]

Best score in generation 10 : [0.9743589743589743]

import matplotlib.pyplot as plt

# Assuming score\_pd is a one-dimensional array or list

plt.plot(score\_pd)

plt.xlabel('Generation')

plt.ylabel('Best Score')

plt.title('Best Score Across Generations')

plt.show()

