```
import numpy as np
import pandas as pd
from sklearn import datasets
import random
from sklearn.metrics import pairwise_distances
iris = datasets.load_iris()

data = pd.DataFrame(data=iris.data, columns=iris.feature_names)
data['label'] = iris.target

data
```

```
     sepal length (cm)  sepal width (cm)  petal length (cm)  petal
width (cm)  \
0                  5.1               3.5                1.4
0.2
1                  4.9               3.0                1.4
0.2
2                  4.7               3.2                1.3
0.2
3                  4.6               3.1                1.5
0.2
4                  5.0               3.6                1.4
0.2
..                 ...               ...                ...
...
145                6.7               3.0                5.2
2.3
146                6.3               2.5                5.0
1.9
147                6.5               3.0                5.2
2.0
148                6.2               3.4                5.4
2.3
149                5.9               3.0                5.1
1.8

     label
0        0
1        0
2        0
3        0
4        0
..     ...
145      2
146      2
147      2
148      2
149      2
```

```
[150 rows x 5 columns]
```

```
mainData = data.drop(["label"], axis=1).values
label = data["label"]
label
```

```
0      0
1      0
2      0
3      0
4      0
      ..
145    2
146    2
147    2
148    2
149    2
Name: label, Length: 150, dtype: int64
```

```
mainData
```

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
       [5.1, 3.8, 1.5, 0.3],
       [5.4, 3.4, 1.7, 0.2],
       [5.1, 3.7, 1.5, 0.4],
       [4.6, 3.6, 1. , 0.2],
       [5.1, 3.3, 1.7, 0.5],
       [4.8, 3.4, 1.9, 0.2],
       [5. , 3. , 1.6, 0.2],
       [5. , 3.4, 1.6, 0.4],
       [5.2, 3.5, 1.5, 0.2],
```

```
       [5.2, 3.4, 1.4, 0.2],
       [4.7, 3.2, 1.6, 0.2],
       [4.8, 3.1, 1.6, 0.2],
       [5.4, 3.4, 1.5, 0.4],
       [5.2, 4.1, 1.5, 0.1],
       [5.5, 4.2, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.2],
       [5. , 3.2, 1.2, 0.2],
       [5.5, 3.5, 1.3, 0.2],
       [4.9, 3.6, 1.4, 0.1],
       [4.4, 3. , 1.3, 0.2],
       [5.1, 3.4, 1.5, 0.2],
       [5. , 3.5, 1.3, 0.3],
       [4.5, 2.3, 1.3, 0.3],
       [4.4, 3.2, 1.3, 0.2],
       [5. , 3.5, 1.6, 0.6],
       [5.1, 3.8, 1.9, 0.4],
       [4.8, 3. , 1.4, 0.3],
       [5.1, 3.8, 1.6, 0.2],
       [4.6, 3.2, 1.4, 0.2],
       [5.3, 3.7, 1.5, 0.2],
       [5. , 3.3, 1.4, 0.2],
       [7. , 3.2, 4.7, 1.4],
       [6.4, 3.2, 4.5, 1.5],
       [6.9, 3.1, 4.9, 1.5],
       [5.5, 2.3, 4. , 1.3],
       [6.5, 2.8, 4.6, 1.5],
       [5.7, 2.8, 4.5, 1.3],
       [6.3, 3.3, 4.7, 1.6],
       [4.9, 2.4, 3.3, 1. ],
       [6.6, 2.9, 4.6, 1.3],
       [5.2, 2.7, 3.9, 1.4],
       [5. , 2. , 3.5, 1. ],
       [5.9, 3. , 4.2, 1.5],
       [6. , 2.2, 4. , 1. ],
       [6.1, 2.9, 4.7, 1.4],
       [5.6, 2.9, 3.6, 1.3],
       [6.7, 3.1, 4.4, 1.4],
       [5.6, 3. , 4.5, 1.5],
       [5.8, 2.7, 4.1, 1. ],
       [6.2, 2.2, 4.5, 1.5],
       [5.6, 2.5, 3.9, 1.1],
       [5.9, 3.2, 4.8, 1.8],
       [6.1, 2.8, 4. , 1.3],
       [6.3, 2.5, 4.9, 1.5],
       [6.1, 2.8, 4.7, 1.2],
       [6.4, 2.9, 4.3, 1.3],
       [6.6, 3. , 4.4, 1.4],
       [6.8, 2.8, 4.8, 1.4],
```

```
       [6.7, 3. , 5. , 1.7],
       [6. , 2.9, 4.5, 1.5],
       [5.7, 2.6, 3.5, 1. ],
       [5.5, 2.4, 3.8, 1.1],
       [5.5, 2.4, 3.7, 1. ],
       [5.8, 2.7, 3.9, 1.2],
       [6. , 2.7, 5.1, 1.6],
       [5.4, 3. , 4.5, 1.5],
       [6. , 3.4, 4.5, 1.6],
       [6.7, 3.1, 4.7, 1.5],
       [6.3, 2.3, 4.4, 1.3],
       [5.6, 3. , 4.1, 1.3],
       [5.5, 2.5, 4. , 1.3],
       [5.5, 2.6, 4.4, 1.2],
       [6.1, 3. , 4.6, 1.4],
       [5.8, 2.6, 4. , 1.2],
       [5. , 2.3, 3.3, 1. ],
       [5.6, 2.7, 4.2, 1.3],
       [5.7, 3. , 4.2, 1.2],
       [5.7, 2.9, 4.2, 1.3],
       [6.2, 2.9, 4.3, 1.3],
       [5.1, 2.5, 3. , 1.1],
       [5.7, 2.8, 4.1, 1.3],
       [6.3, 3.3, 6. , 2.5],
       [5.8, 2.7, 5.1, 1.9],
       [7.1, 3. , 5.9, 2.1],
       [6.3, 2.9, 5.6, 1.8],
       [6.5, 3. , 5.8, 2.2],
       [7.6, 3. , 6.6, 2.1],
       [4.9, 2.5, 4.5, 1.7],
       [7.3, 2.9, 6.3, 1.8],
       [6.7, 2.5, 5.8, 1.8],
       [7.2, 3.6, 6.1, 2.5],
       [6.5, 3.2, 5.1, 2. ],
       [6.4, 2.7, 5.3, 1.9],
       [6.8, 3. , 5.5, 2.1],
       [5.7, 2.5, 5. , 2. ],
       [5.8, 2.8, 5.1, 2.4],
       [6.4, 3.2, 5.3, 2.3],
       [6.5, 3. , 5.5, 1.8],
       [7.7, 3.8, 6.7, 2.2],
       [7.7, 2.6, 6.9, 2.3],
       [6. , 2.2, 5. , 1.5],
       [6.9, 3.2, 5.7, 2.3],
       [5.6, 2.8, 4.9, 2. ],
       [7.7, 2.8, 6.7, 2. ],
       [6.3, 2.7, 4.9, 1.8],
       [6.7, 3.3, 5.7, 2.1],
       [7.2, 3.2, 6. , 1.8],
```

```
       [6.2, 2.8, 4.8, 1.8],
       [6.1, 3. , 4.9, 1.8],
       [6.4, 2.8, 5.6, 2.1],
       [7.2, 3. , 5.8, 1.6],
       [7.4, 2.8, 6.1, 1.9],
       [7.9, 3.8, 6.4, 2. ],
       [6.4, 2.8, 5.6, 2.2],
       [6.3, 2.8, 5.1, 1.5],
       [6.1, 2.6, 5.6, 1.4],
       [7.7, 3. , 6.1, 2.3],
       [6.3, 3.4, 5.6, 2.4],
       [6.4, 3.1, 5.5, 1.8],
       [6. , 3. , 4.8, 1.8],
       [6.9, 3.1, 5.4, 2.1],
       [6.7, 3.1, 5.6, 2.4],
       [6.9, 3.1, 5.1, 2.3],
       [5.8, 2.7, 5.1, 1.9],
       [6.8, 3.2, 5.9, 2.3],
       [6.7, 3.3, 5.7, 2.5],
       [6.7, 3. , 5.2, 2.3],
       [6.3, 2.5, 5. , 1.9],
       [6.5, 3. , 5.2, 2. ],
       [6.2, 3.4, 5.4, 2.3],
       [5.9, 3. , 5.1, 1.8]])
```

so our chromosomes should have 150 cells (we ignore the last cell to tell us how many clusters we have becuase we recognize the number of clusters later)

```
data.head()
```

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width
(cm)  \
0                5.1               3.5                1.4
0.2
1                4.9               3.0                1.4
0.2
2                4.7               3.2                1.3
0.2
3                4.6               3.1                1.5
0.2
4                5.0               3.6                1.4
0.2

   label
0      0
1      0
2      0
3      0
4      0
```

```
data.isna().sum()
```

```
sepal length (cm)    0
sepal width (cm)     0
petal length (cm)    0
petal width (cm)     0
label                0
dtype: int64
```

```
data.describe()
```

```
       sepal length (cm)  sepal width (cm)  petal length (cm)  \
count         150.000000        150.000000         150.000000
mean            5.843333          3.057333           3.758000
std             0.828066          0.435866           1.765298
min             4.300000          2.000000           1.000000
25%             5.100000          2.800000           1.600000
50%             5.800000          3.000000           4.350000
75%             6.400000          3.300000           5.100000
max             7.900000          4.400000           6.900000

       petal width (cm)       label
count        150.000000  150.000000
mean           1.199333    1.000000
std            0.762238    0.819232
min            0.100000    0.000000
25%            0.300000    0.000000
50%            1.300000    1.000000
75%            1.800000    2.000000
max            2.500000    2.000000
```

```
data["label"].unique()
```

```
array([0, 1, 2])
```

finding the number of clusters we should define and its 3

```
def objective_function(chromosome, data, num_clusters):
    clusters = [np.where(chromosome == i)[0] for i in range(0,
num_clusters)]
    centroids = [np.mean(data[cluster], axis=0) for cluster in
clusters]
    score = 0
    for i, cluster in enumerate(clusters):
        if len(cluster) > 0:
            distances = pairwise_distances(data[cluster],
[centroids[i]], metric='euclidean')
            score += np.sum(distances ** 2)

    return score
```

```
def initialize_population(pop_size, num_points, num_clusters):
    population = []
    for _ in range(pop_size):
        chromosome = np.random.randint(0 , num_clusters , num_points)
        z=objective_function(chromosome,mainData,3)
        while(z>630):
            chromosome = np.random.randint(0 , num_clusters ,
num_points)
            z=objective_function(chromosome,mainData,3)
        population.append(chromosome)
    return population

population=initialize_population(10,150,3)

population
```

```
[array([1, 1, 2, 0, 2, 0, 2, 2, 0, 1, 2, 0, 0, 0, 1, 2, 0, 2, 1, 0, 2,
2,
       0, 2, 2, 0, 2, 2, 0, 0, 2, 2, 2, 2, 2, 0, 1, 0, 2, 2, 0, 2, 1,
2,
       2, 2, 0, 0, 0, 0, 2, 0, 1, 0, 1, 2, 0, 0, 0, 2, 0, 0, 1, 1, 1,
2,
       2, 2, 2, 1, 2, 1, 2, 0, 1, 2, 1, 1, 1, 2, 1, 2, 2, 1, 2, 1, 1,
0,
       2, 0, 2, 1, 2, 2, 0, 2, 0, 2, 0, 0, 1, 0, 2, 0, 0, 1, 0, 2, 0,
2,
       1, 1, 2, 0, 0, 1, 1, 2, 1, 1, 1, 0, 1, 1, 2, 2, 1, 1, 1, 2, 1,
1,
       1, 1, 0, 0, 2, 0, 1, 2, 2, 1, 2, 1, 1, 2, 0, 2, 2, 0]),
 array([0, 0, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 2, 0, 0, 1, 0, 0, 1, 1,
0,
       2, 1, 1, 2, 2, 2, 1, 1, 2, 0, 2, 1, 2, 1, 0, 1, 1, 1, 1, 1, 1,
0,
       0, 1, 2, 1, 2, 2, 2, 2, 0, 0, 1, 2, 0, 0, 0, 1, 1, 0, 1, 0, 2,
1,
       0, 1, 1, 0, 2, 2, 0, 1, 0, 2, 2, 0, 1, 1, 1, 0, 1, 0, 0, 0, 2,
0,
       1, 0, 1, 1, 0, 2, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 2, 0, 0, 2, 2,
0,
       2, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 2, 1, 2, 2, 0, 0, 2, 0, 0, 0,
0,
       1, 0, 2, 0, 0, 0, 2, 2, 0, 0, 0, 0, 1, 1, 0, 1, 0, 2]),
 array([0, 1, 2, 0, 0, 2, 0, 2, 2, 1, 0, 2, 0, 1, 1, 2, 2, 0, 0, 2, 2,
0,
       2, 0, 2, 2, 2, 0, 0, 1, 2, 2, 2, 2, 2, 2, 0, 2, 0, 1, 0, 2, 2,
2,
       0, 2, 2, 1, 1, 0, 2, 1, 2, 0, 1, 2, 1, 0, 1, 0, 1, 0, 2, 2, 0,
2,
       0, 2, 0, 0, 0, 0, 2, 2, 0, 1, 1, 1, 2, 2, 2, 1, 2, 1, 0, 2, 0,
0,
```

```
        1, 2, 1, 1, 0, 2, 1, 1, 1, 0, 2, 2, 0, 2, 1, 0, 2, 1, 2, 1, 1,
0,
        1, 0, 0, 0, 0, 1, 0, 1, 1, 2, 1, 1, 2, 1, 0, 2, 2, 1, 1, 1, 1,
1,
        1, 2, 2, 2, 1, 1, 2, 2, 2, 0, 2, 0, 1, 0, 0, 1, 1, 2]),
 array([0, 2, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 2, 0, 2, 2, 0, 1,
1,
        0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 2, 2, 1, 0, 0, 2, 0, 0, 0, 0,
1,
        1, 0, 0, 0, 0, 0, 2, 2, 1, 1, 0, 2, 2, 0, 2, 0, 0, 2, 0, 0, 1,
1,
        0, 0, 2, 2, 0, 0, 1, 1, 2, 0, 2, 2, 2, 0, 0, 2, 2, 1, 1, 1, 0,
0,
        1, 2, 1, 2, 1, 2, 0, 2, 1, 2, 2, 0, 2, 1, 1, 0, 2, 2, 2, 0, 2,
0,
        1, 1, 1, 0, 1, 1, 1, 1, 0, 2, 1, 2, 2, 2, 1, 1, 2, 1, 0, 1, 2,
2,
        2, 2, 2, 0, 0, 1, 0, 2, 2, 0, 2, 2, 2, 2, 1, 0, 1, 0]),
 array([1, 1, 2, 0, 2, 0, 2, 2, 2, 2, 0, 0, 2, 2, 0, 0, 0, 0, 0, 2, 2,
0,
        2, 1, 0, 2, 2, 1, 0, 0, 0, 0, 1, 0, 2, 2, 2, 1, 2, 2, 2, 2, 0,
2,
        2, 2, 2, 2, 0, 0, 1, 2, 0, 2, 1, 0, 0, 1, 2, 1, 1, 1, 1, 1, 2,
2,
        1, 0, 1, 2, 2, 1, 2, 1, 0, 2, 1, 1, 1, 2, 0, 2, 2, 1, 2, 0, 0,
0,
        0, 1, 0, 1, 1, 1, 2, 0, 2, 0, 0, 2, 1, 2, 0, 2, 1, 0, 1, 1, 0,
1,
        1, 1, 1, 0, 1, 0, 0, 2, 0, 2, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1,
0,
        0, 2, 2, 2, 2, 2, 2, 0, 1, 0, 0, 1, 1, 1, 2, 0, 0, 1]),
 array([0, 0, 2, 1, 1, 1, 2, 1, 2, 2, 1, 0, 1, 1, 2, 2, 2, 2, 2, 2, 1,
1,
        1, 1, 1, 0, 2, 1, 1, 0, 2, 2, 0, 1, 2, 1, 0, 1, 0, 2, 2, 2, 2,
1,
        1, 1, 1, 1, 2, 0, 0, 0, 0, 2, 1, 0, 1, 2, 2, 0, 2, 2, 1, 2, 1,
0,
        2, 0, 2, 0, 0, 0, 1, 0, 2, 0, 1, 2, 2, 0, 1, 1, 1, 0, 0, 2, 2,
0,
        0, 0, 0, 0, 2, 2, 0, 2, 2, 2, 0, 2, 2, 1, 1, 0, 0, 0, 0, 0, 0,
1,
        0, 1, 1, 2, 2, 1, 0, 0, 0, 1, 0, 0, 0, 1, 2, 0, 2, 0, 2, 0, 0,
0,
        0, 0, 2, 1, 0, 0, 1, 0, 1, 2, 1, 2, 1, 2, 1, 2, 0, 1]),
 array([2, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 2, 0, 0, 2, 2, 0, 0, 0, 0,
1,
        0, 0, 1, 2, 0, 1, 0, 2, 2, 2, 1, 2, 0, 2, 0, 0, 1, 0, 0, 2, 1,
0,
        0, 2, 0, 1, 2, 2, 2, 1, 2, 1, 2, 2, 1, 2, 1, 2, 1, 1, 0, 2, 0,
```

```
1,
        2, 2, 1, 2, 2, 2, 1, 0, 0, 0, 1, 2, 0, 0, 0, 0, 2, 0, 0, 1, 2,
2,
        0, 1, 1, 2, 1, 1, 1, 0, 0, 1, 0, 0, 2, 0, 0, 1, 1, 2, 1, 1, 1,
1,
        2, 1, 1, 0, 0, 2, 2, 1, 1, 2, 2, 0, 1, 1, 2, 2, 2, 2, 0, 0, 1,
2,
        2, 2, 1, 0, 2, 1, 1, 1, 1, 1, 2, 0, 0, 2, 2, 2, 0, 2]),
 array([0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 2, 1, 2, 2, 1, 1, 2, 0, 0, 2, 1,
0,
        0, 0, 0, 0, 0, 1, 0, 2, 0, 2, 0, 2, 1, 1, 1, 2, 1, 1, 1, 1, 1,
0,
        0, 1, 1, 1, 1, 1, 2, 0, 0, 2, 1, 2, 1, 0, 0, 1, 0, 0, 2, 1, 2,
0,
        1, 2, 2, 0, 2, 1, 2, 0, 1, 1, 2, 1, 0, 1, 1, 1, 1, 2, 0, 2, 0,
0,
        0, 1, 1, 2, 2, 1, 0, 2, 2, 0, 1, 0, 0, 0, 1, 2, 0, 2, 1, 2, 2,
0,
        2, 2, 1, 2, 0, 2, 2, 2, 2, 1, 2, 0, 1, 2, 2, 2, 0, 2, 1, 0, 2,
0,
        0, 0, 2, 0, 0, 1, 0, 2, 0, 0, 0, 2, 1, 2, 0, 0, 2, 2]),
 array([0, 2, 2, 0, 1, 2, 2, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 2,
0,
        0, 0, 1, 1, 0, 2, 0, 0, 0, 2, 1, 2, 2, 0, 0, 0, 0, 0, 0, 1, 2,
1,
        0, 0, 1, 1, 1, 1, 2, 1, 2, 0, 2, 0, 1, 0, 2, 1, 0, 2, 0, 0, 1,
1,
        1, 1, 2, 0, 0, 0, 2, 0, 1, 1, 0, 2, 0, 1, 0, 2, 0, 0, 0, 1, 0,
2,
        0, 1, 0, 2, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 2, 0, 1, 1, 0,
2,
        0, 2, 1, 1, 2, 0, 2, 2, 2, 2, 2, 2, 2, 0, 0, 2, 1, 1, 0, 1, 1,
2,
        2, 1, 0, 2, 2, 0, 0, 2, 2, 2, 2, 0, 2, 2, 1, 2, 2, 0]),
 array([1, 0, 2, 1, 1, 0, 0, 1, 0, 0, 2, 1, 2, 0, 0, 0, 0, 0, 2, 1, 0,
1,
        1, 0, 0, 0, 0, 0, 2, 0, 2, 1, 2, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
2,
        0, 1, 1, 2, 2, 0, 1, 1, 2, 2, 1, 0, 2, 2, 2, 0, 0, 1, 0, 2, 2,
1,
        1, 1, 1, 0, 1, 2, 2, 0, 1, 1, 0, 0, 2, 1, 0, 0, 1, 2, 1, 0, 1,
1,
        2, 1, 0, 2, 0, 2, 2, 2, 1, 2, 1, 1, 0, 0, 1, 2, 1, 2, 0, 1, 0,
2,
        2, 2, 1, 2, 1, 1, 1, 0, 2, 0, 1, 0, 1, 0, 2, 1, 2, 0, 1, 2, 2,
2,
        0, 2, 1, 2, 0, 2, 1, 0, 2, 1, 1, 2, 2, 2, 0, 1, 1, 2])]

score = []
for chro in population:
```

```
    entry=objective_function(chro,mainData,3)
    score.append(entry)
    print(objective_function(chro,mainData,3))
```

```
620.9092791287976
602.2878824702927
626.791716641679
623.716552795031
617.1782704341967
625.0218519803182
624.3885708061
617.532152777778
628.7369423076916
626.17995062224
```

```
worstPoint=0
worstPointValue=630
def calculateWorst(score):
    worstPoint=np.argmax(score)
    worstPointValue=score[worstPoint]
calculateWorst(score)
worstPoint
```

```
#population[worstPoint]
```

```
0
```

```
def mutation(population,score,chromosome):
    calculateWorst(score)
    newChromosome=chromosome.copy()
    cell=np.random.randint(0,150)
    newChromosome[cell]=np.random.randint(0,3)
    #while(newChromosome!=chromosome):
    #   newChromosome[cell]=np.random.randint(0,3)
    newScore=objective_function(newChromosome,mainData,3)
    if(newScore<worstPointValue):
        population[worstPoint]=newChromosome
        score[worstPoint]=newScore
        calculateWorst(score)
```

```
crossover_point = np.random.randint(0,2)
crossover_point
```

```
0
```

```
def crossover(population,score,mainData,chro1,chro2):
    rand = np.random.randint(0,2)
    calculateWorst(score)
    if(rand==0):
        crossover_point = np.random.randint(0, 150)
        child1 = np.concatenate((chro1[:crossover_point],
```

```
chro2[crossover_point:]))
        child2 = np.concatenate((chro2[:crossover_point],
chro1[crossover_point:]))
        mutation(population,score,child1)
        mutation(population,score,child2)
        newScore1=objective_function(child1,mainData,3)
        newScore2=objective_function(child2,mainData,3)
        if(newScore1<worstPointValue):
            population[worstPoint]=child1
            score[worstPoint]=newScore1
            calculateWorst(score)
        if(newScore2<worstPointValue):
            population[worstPoint]=child2
            score[worstPoint]=newScore2
            calculateWorst(score)
    else:
        child1 = np.concatenate((chro1[:30],
chro2[30:60],chro1[60:90],chro2[90:120],chro1[120:150]))
        child2 = np.concatenate((chro2[:30],
chro1[30:60],chro2[60:90],chro1[90:120],chro2[120:150]))
        mutation(population,score,child1)
        mutation(population,score,child2)
        newScore1=objective_function(child1,mainData,3)
        newScore2=objective_function(child2,mainData,3)
        if(newScore1<worstPointValue):
            population[worstPoint]=child1
            score[worstPoint]=newScore1
            calculateWorst(score)
        if(newScore2<worstPointValue):
            population[worstPoint]=child2
            score[worstPoint]=newScore2
        calculateWorst(score)

def generic(mainData,iteration):
    population=initialize_population(50,150,3)
    score = []
    for chro in population:
        entry=objective_function(chro,mainData,3)
        score.append(entry)
    """
    for i in range(iteration):
        #parent1 , parent2 = random.sample(population, 2)
        crossover(mainData,parent1,parent2)
    """
    b=population[1]
    c=population[0]
    crossover(population,score,mainData,b,c)
    for i in range(iteration):
        parent1 , parent2 = random.sample(population, 2)
        crossover(population,score,mainData,parent1,parent2)
```

```
    bestPoint=np.argmin(score)
    population[bestPoint]
    return population[bestPoint]


sol=generic(mainData,100)

differences = sum(1 for a, b in zip(sol, label) if a != b)

differences

104

1-differences/150

0.30666666666666664
```