K-MEANS ALGORITHM FOR UNSUPERVISED LEARNING

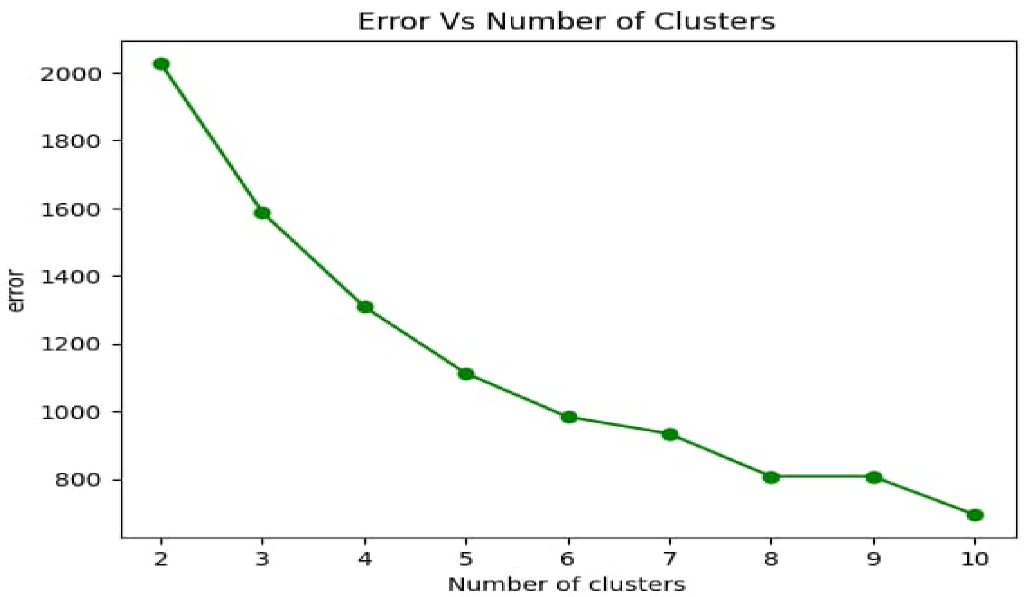
```
import pandas as pd
import random
import numpy as np
import matplotlib.pyplot as plt

pd.set_option('display.max_columns', None)

Data = pd.read_csv('heart.csv')
Data = Data.drop('target', axis = 1)

(n, m) = Data.shape

Data.index = [f'X{j+1}' for j in range(n)]
Data.columns = [f'f{i+1}' for i in range(m)]

# split data into train_test ratio
train_data = Data.sample(frac = 0.8)
test_data = Data.drop(train_data.index)

# K_Means Algorithm
def KMeans(Data, K):
    n = len(Data.axes[0])
    m = len(Data.axes[1])
    rn = random.sample(range(n), K)

    centroids = []
    for i in rn:
        centroids.append(Data.iloc[i])
    centroids = pd.DataFrame(centroids, index=range(K),
columns=Data.columns)

    stopping_criteria = 1
    while stopping_criteria == 1:
        new_centroids = [[] for _ in range(K)]
        list_cluster = [[] for i in range(K)]

        for i in range(n):
```

```
            distance = []
            for k in range(K):
                sum_of_sqrs = 0
                for j in Data.columns:
                    sum_of_sqrs += (Data[j].iloc[i] - centroids[j].iloc[k])**2
                d = (sum_of_sqrs)**(1/2)
                distance.append(d)
            min_val_index = distance.index(min(distance))
            list_cluster[min_val_index].extend([Data.iloc[i].tolist()])

        for i in range(K):
            Mean_for_cluster = np.array(list_cluster[i])
            new_centroid = np.mean(Mean_for_cluster, axis=0)
            new_centroids[i] = new_centroid.tolist()

        new_centroids = pd.DataFrame(new_centroids, columns=Data.columns)

        if centroids.equals(new_centroids):
            stopping_criteria = 0
        centroids = new_centroids

    return centroids, list_cluster

# training the algorithm
error = []
for K in range(2, 11):
    centroid, list_cluster = KMeans(train_data, K)
    centroid = centroid.values
    sum_of_sqr = 0
    for i in range(K):
        for j in range(len(list_cluster[i])):
            for k in range(m):
                sum_of_sqr += (list_cluster[i][j][k] - centroid[i][k])**2
    error.append(sum_of_sqr / len(train_data))

# Calculate elbow_point for optimal_K
elbow_point = 0
max_diff = 0
for i in range(1, len(error)-1):
    diff = (error[i-1] - error[i]) / (error[i] - error[i+1])
```

```
    if diff > max_diff:
        max_diff = diff
        elbow_point = i + 2

Optimal_K = elbow_point
print('Optimal_K = ', Optimal_K)

# plot the graph
plt.plot(range(2,11), error, color = 'g', marker = 'o')
plt.xlabel('Number of clusters')
plt.ylabel('error')
plt.title('Error Vs Number of Clusters')
plt.show()

# testing algorithm
centroids, list_cluster = KMeans(test_data, Optimal_K)

centroids.index = [f'c{i+1}' for i in range(Optimal_K)]

# error for testing data
sum_sqr = 0
for i in range(Optimal_K) :
    list_cl = pd.DataFrame(list_cluster[i], columns=test_data.columns)
    sub_df = list_cl - centroids.iloc[i]
    sub_df = sub_df ** 2
    sub_df = sub_df.sum(axis = 1)
    sum_sqr += sub_df.sum(axis = 0)
print(f'\nThe error in testing data is : {sum_sqr / len(test_data)}')

print(f'\nThe centroids of clusters are : \n{round(centroids,2)}\n')
for i in range(Optimal_K):
    print(f'Cluster_C{i+1} : \n{pd.DataFrame(list_cluster[i],
columns=test_data.columns)}\n')
```

OUTPUT

Optimal_K = 6



The error in testing data is : 892.6776021362093

The centroids of clusters are :

	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11 \
c1	48.73	0.79	1.33	124.73	212.33	0.15	0.75	164.02	0.21	0.56	1.67
c2	62.11	0.58	0.37	127.95	274.53	0.21	0.42	110.16	0.53	1.55	1.00
c3	60.62	0.46	0.77	143.23	349.08	0.15	0.77	150.62	0.54	1.22	1.62
c4	56.41	0.76	0.94	130.32	188.68	0.21	0.71	128.62	0.38	1.55	1.18
c5	55.51	0.67	1.31	136.16	248.00	0.16	0.36	157.16	0.20	0.99	1.49
c6	54.31	0.64	1.14	136.47	295.33	0.22	0.42	152.36	0.39	1.01	1.36

	f12	f13
c1	0.65	2.40
c2	1.63	2.32
c3	0.62	2.31
c4	0.56	2.09

c5	0.56	2.22
c6	0.78	2.50

Cluster_C1 :

	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11	f12	f13
0	52.0	1.0	0.0	125.0	212.0	0.0	1.0	168.0	0.0	1.0	2.0	2.0	3.0
1	45.0	1.0	0.0	104.0	208.0	0.0	0.0	148.0	1.0	3.0	1.0	0.0	2.0
2	37.0	0.0	2.0	120.0	215.0	0.0	1.0	170.0	0.0	0.0	2.0	0.0	2.0
3	44.0	1.0	1.0	120.0	220.0	0.0	1.0	170.0	0.0	0.0	2.0	0.0	2.0
4	53.0	1.0	2.0	130.0	197.0	1.0	0.0	152.0	0.0	1.2	0.0	0.0	2.0
5	52.0	1.0	2.0	138.0	223.0	0.0	1.0	169.0	0.0	0.0	2.0	4.0	2.0
6	56.0	1.0	1.0	130.0	221.0	0.0	0.0	163.0	0.0	0.0	2.0	0.0	3.0
7	34.0	1.0	3.0	118.0	182.0	0.0	0.0	174.0	0.0	0.0	2.0	0.0	2.0
8	39.0	0.0	2.0	94.0	199.0	0.0	1.0	179.0	0.0	0.0	2.0	0.0	2.0
9	52.0	1.0	1.0	128.0	205.0	1.0	1.0	184.0	0.0	0.0	2.0	0.0	2.0
10	60.0	1.0	0.0	117.0	230.0	1.0	1.0	160.0	1.0	1.4	2.0	2.0	3.0
11	39.0	0.0	2.0	138.0	220.0	0.0	1.0	152.0	0.0	0.0	1.0	0.0	2.0
12	57.0	1.0	0.0	132.0	207.0	0.0	1.0	168.0	1.0	0.0	2.0	0.0	3.0
13	52.0	1.0	2.0	138.0	223.0	0.0	1.0	169.0	0.0	0.0	2.0	4.0	2.0
14	51.0	1.0	2.0	94.0	227.0	0.0	1.0	154.0	1.0	0.0	2.0	1.0	3.0
15	52.0	0.0	2.0	136.0	196.0	0.0	0.0	169.0	0.0	0.1	1.0	0.0	2.0
16	37.0	0.0	2.0	120.0	215.0	0.0	1.0	170.0	0.0	0.0	2.0	0.0	2.0
17	52.0	1.0	2.0	172.0	199.0	1.0	1.0	162.0	0.0	0.5	2.0	0.0	3.0
18	58.0	1.0	2.0	132.0	224.0	0.0	0.0	173.0	0.0	3.2	2.0	2.0	3.0
19	52.0	1.0	0.0	112.0	230.0	0.0	1.0	160.0	0.0	0.0	2.0	1.0	2.0
20	43.0	1.0	0.0	110.0	211.0	0.0	1.0	161.0	0.0	0.0	2.0	0.0	3.0
21	38.0	1.0	2.0	138.0	175.0	0.0	1.0	173.0	0.0	0.0	2.0	4.0	2.0
22	44.0	1.0	1.0	120.0	220.0	0.0	1.0	170.0	0.0	0.0	2.0	0.0	2.0
23	42.0	0.0	2.0	120.0	209.0	0.0	1.0	173.0	0.0	0.0	1.0	0.0	2.0
24	41.0	1.0	2.0	130.0	214.0	0.0	0.0	168.0	0.0	2.0	1.0	0.0	2.0
25	54.0	0.0	2.0	160.0	201.0	0.0	1.0	163.0	0.0	0.0	2.0	1.0	2.0
26	38.0	1.0	2.0	138.0	175.0	0.0	1.0	173.0	0.0	0.0	2.0	4.0	2.0
27	50.0	0.0	2.0	120.0	219.0	0.0	1.0	158.0	0.0	1.6	1.0	0.0	2.0
28	53.0	1.0	0.0	140.0	203.0	1.0	0.0	155.0	1.0	3.1	0.0	0.0	3.0
29	39.0	1.0	0.0	118.0	219.0	0.0	1.0	140.0	0.0	1.2	1.0	0.0	3.0
30	67.0	1.0	2.0	152.0	212.0	0.0	0.0	150.0	0.0	0.8	1.0	0.0	3.0
31	51.0	1.0	2.0	94.0	227.0	0.0	1.0	154.0	1.0	0.0	2.0	1.0	3.0
32	56.0	1.0	1.0	130.0	221.0	0.0	0.0	163.0	0.0	0.0	2.0	0.0	3.0
33	56.0	1.0	1.0	130.0	221.0	0.0	0.0	163.0	0.0	0.0	2.0	0.0	3.0
34	51.0	1.0	2.0	94.0	227.0	0.0	1.0	154.0	1.0	0.0	2.0	1.0	3.0

35	46.0	0.0	1.0	105.0	204.0	0.0	1.0	172.0	0.0	0.0	2.0	0.0	2.0
36	51.0	1.0	2.0	100.0	222.0	0.0	1.0	143.0	1.0	1.2	1.0	0.0	2.0
37	59.0	1.0	3.0	134.0	204.0	0.0	1.0	162.0	0.0	0.8	2.0	2.0	2.0
38	44.0	1.0	2.0	130.0	233.0	0.0	1.0	179.0	1.0	0.4	2.0	0.0	2.0
39	52.0	1.0	2.0	172.0	199.0	1.0	1.0	162.0	0.0	0.5	2.0	0.0	3.0
40	58.0	1.0	2.0	112.0	230.0	0.0	0.0	165.0	0.0	2.5	1.0	1.0	3.0
41	35.0	0.0	0.0	138.0	183.0	0.0	1.0	182.0	0.0	1.4	2.0	0.0	2.0
42	52.0	1.0	0.0	112.0	230.0	0.0	1.0	160.0	0.0	0.0	2.0	1.0	2.0
43	41.0	1.0	1.0	110.0	235.0	0.0	1.0	153.0	0.0	0.0	2.0	0.0	2.0
44	57.0	1.0	0.0	132.0	207.0	0.0	1.0	168.0	1.0	0.0	2.0	0.0	3.0
45	52.0	1.0	1.0	128.0	205.0	1.0	1.0	184.0	0.0	0.0	2.0	0.0	2.0
46	48.0	1.0	1.0	110.0	229.0	0.0	1.0	168.0	0.0	1.0	0.0	0.0	3.0
47	47.0	1.0	0.0	112.0	204.0	0.0	1.0	143.0	0.0	0.1	2.0	0.0	2.0

Cluster_C2 :

	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11	f12	f13
0	58.0	0.0	0.0	100.0	248.0	0.0	0.0	122.0	0.0	1.0	1.0	0.0	2.0
1	54.0	1.0	0.0	122.0	286.0	0.0	0.0	116.0	1.0	3.2	1.0	2.0	2.0
2	62.0	1.0	1.0	120.0	281.0	0.0	0.0	103.0	0.0	1.4	1.0	1.0	3.0
3	56.0	1.0	0.0	130.0	283.0	1.0	0.0	103.0	1.0	1.6	0.0	0.0	3.0
4	58.0	1.0	0.0	128.0	259.0	0.0	0.0	130.0	1.0	3.0	1.0	2.0	3.0
5	62.0	0.0	2.0	130.0	263.0	0.0	1.0	97.0	0.0	1.2	1.0	1.0	3.0
6	67.0	1.0	0.0	160.0	286.0	0.0	0.0	108.0	1.0	1.5	1.0	3.0	2.0
7	70.0	1.0	0.0	130.0	322.0	0.0	0.0	109.0	0.0	2.4	1.0	3.0	2.0
8	74.0	0.0	1.0	120.0	269.0	0.0	0.0	121.0	1.0	0.2	2.0	1.0	2.0
9	71.0	0.0	2.0	110.0	265.0	1.0	0.0	130.0	0.0	0.0	2.0	1.0	2.0
10	57.0	1.0	0.0	152.0	274.0	0.0	1.0	88.0	1.0	1.2	1.0	1.0	3.0
11	62.0	0.0	0.0	138.0	294.0	1.0	1.0	106.0	0.0	1.9	1.0	3.0	2.0
12	64.0	0.0	0.0	130.0	303.0	0.0	1.0	122.0	0.0	2.0	1.0	2.0	2.0
13	62.0	1.0	0.0	120.0	267.0	0.0	1.0	99.0	1.0	1.8	1.0	2.0	3.0
14	66.0	1.0	1.0	160.0	246.0	0.0	1.0	120.0	1.0	0.0	1.0	3.0	1.0
15	62.0	0.0	0.0	138.0	294.0	1.0	1.0	106.0	0.0	1.9	1.0	3.0	2.0
16	64.0	1.0	0.0	120.0	246.0	0.0	0.0	96.0	1.0	2.2	0.0	1.0	2.0
17	58.0	0.0	0.0	100.0	248.0	0.0	0.0	122.0	0.0	1.0	1.0	0.0	2.0
18	53.0	1.0	0.0	123.0	282.0	0.0	1.0	95.0	1.0	2.0	1.0	2.0	3.0

Cluster_C3 :

	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11	f12	f13
0	65.0	0.0	2.0	140.0	417.0	1.0	0.0	157.0	0.0	0.8	2.0	1.0	2.0
1	64.0	0.0	0.0	180.0	325.0	0.0	1.0	154.0	1.0	0.0	2.0	0.0	2.0

2	57.0	0.0	0.0	120.0	354.0	0.0	1.0	163.0	1.0	0.6	2.0	0.0	2.0
3	58.0	0.0	2.0	120.0	340.0	0.0	1.0	172.0	0.0	0.0	2.0	0.0	2.0
4	55.0	1.0	0.0	132.0	353.0	0.0	1.0	132.0	1.0	1.2	1.0	1.0	3.0
5	57.0	0.0	0.0	120.0	354.0	0.0	1.0	163.0	1.0	0.6	2.0	0.0	2.0
6	63.0	0.0	0.0	150.0	407.0	0.0	0.0	154.0	0.0	4.0	1.0	3.0	3.0
7	63.0	1.0	0.0	130.0	330.0	1.0	0.0	132.0	1.0	1.8	2.0	3.0	3.0
8	59.0	1.0	0.0	170.0	326.0	0.0	0.0	140.0	1.0	3.4	0.0	0.0	3.0
9	64.0	1.0	2.0	140.0	335.0	0.0	1.0	158.0	0.0	0.0	2.0	0.0	2.0
10	64.0	1.0	2.0	140.0	335.0	0.0	1.0	158.0	0.0	0.0	2.0	0.0	2.0
11	64.0	1.0	2.0	140.0	335.0	0.0	1.0	158.0	0.0	0.0	2.0	0.0	2.0
12	55.0	0.0	0.0	180.0	327.0	0.0	2.0	117.0	1.0	3.4	1.0	0.0	2.0

Cluster_C4 :

	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11	f12	f13
0	70.0	1.0	0.0	145.0	174.0	0.0	1.0	125.0	1.0	2.6	0.0	0.0	3.0
1	60.0	1.0	2.0	140.0	185.0	0.0	0.0	155.0	0.0	3.0	1.0	0.0	2.0
2	44.0	1.0	0.0	120.0	169.0	0.0	1.0	144.0	1.0	2.8	0.0	0.0	1.0
3	76.0	0.0	2.0	140.0	197.0	0.0	2.0	116.0	0.0	1.1	1.0	0.0	2.0
4	57.0	1.0	2.0	150.0	126.0	1.0	1.0	173.0	0.0	0.2	2.0	1.0	3.0
5	44.0	1.0	0.0	120.0	169.0	0.0	1.0	144.0	1.0	2.8	0.0	0.0	1.0
6	40.0	1.0	0.0	110.0	167.0	0.0	0.0	114.0	1.0	2.0	1.0	0.0	3.0
7	56.0	1.0	0.0	132.0	184.0	0.0	0.0	105.0	1.0	2.1	1.0	1.0	1.0
8	63.0	1.0	0.0	140.0	187.0	0.0	0.0	144.0	1.0	4.0	2.0	2.0	3.0
9	50.0	1.0	0.0	144.0	200.0	0.0	0.0	126.0	1.0	0.9	1.0	0.0	3.0
10	42.0	1.0	2.0	130.0	180.0	0.0	1.0	150.0	0.0	0.0	2.0	0.0	2.0
11	35.0	1.0	0.0	120.0	198.0	0.0	1.0	130.0	1.0	1.6	1.0	0.0	3.0
12	59.0	1.0	2.0	126.0	218.0	1.0	1.0	134.0	0.0	2.2	1.0	1.0	1.0
13	51.0	1.0	2.0	110.0	175.0	0.0	1.0	123.0	0.0	0.6	2.0	0.0	2.0
14	58.0	1.0	0.0	146.0	218.0	0.0	1.0	105.0	0.0	2.0	1.0	1.0	3.0
15	51.0	1.0	2.0	110.0	175.0	0.0	1.0	123.0	0.0	0.6	2.0	0.0	2.0
16	51.0	1.0	2.0	110.0	175.0	0.0	1.0	123.0	0.0	0.6	2.0	0.0	2.0
17	71.0	0.0	0.0	112.0	149.0	0.0	1.0	125.0	0.0	1.6	1.0	0.0	2.0
18	49.0	1.0	2.0	120.0	188.0	0.0	1.0	139.0	0.0	2.0	1.0	3.0	3.0
19	63.0	0.0	0.0	124.0	197.0	0.0	1.0	136.0	1.0	0.0	1.0	0.0	2.0
20	59.0	1.0	0.0	164.0	176.0	1.0	0.0	90.0	0.0	1.0	1.0	2.0	1.0
21	61.0	1.0	0.0	138.0	166.0	0.0	0.0	125.0	1.0	3.6	1.0	1.0	2.0
22	60.0	0.0	2.0	120.0	178.0	1.0	1.0	96.0	0.0	0.0	2.0	0.0	2.0
23	45.0	0.0	1.0	112.0	160.0	0.0	1.0	138.0	0.0	0.0	1.0	0.0	2.0
24	66.0	0.0	3.0	150.0	226.0	0.0	1.0	114.0	0.0	2.6	0.0	0.0	2.0
25	51.0	1.0	3.0	125.0	213.0	0.0	0.0	125.0	1.0	1.4	2.0	1.0	2.0

26	68.0	0.0	2.0	120.0	211.0	0.0	0.0	115.0	0.0	1.5	1.0	0.0	2.0
27	68.0	1.0	0.0	144.0	193.0	1.0	1.0	141.0	0.0	3.4	1.0	2.0	3.0
28	62.0	1.0	1.0	128.0	208.0	1.0	0.0	140.0	0.0	0.0	2.0	0.0	2.0
29	57.0	1.0	0.0	140.0	192.0	0.0	1.0	148.0	0.0	0.4	1.0	0.0	1.0
30	59.0	1.0	2.0	126.0	218.0	1.0	1.0	134.0	0.0	2.2	1.0	1.0	1.0
31	53.0	1.0	0.0	142.0	226.0	0.0	0.0	111.0	1.0	0.0	2.0	0.0	3.0
32	64.0	1.0	0.0	145.0	212.0	0.0	0.0	132.0	0.0	2.0	1.0	2.0	1.0
33	55.0	0.0	0.0	128.0	205.0	0.0	2.0	130.0	1.0	2.0	1.0	1.0	3.0

Cluster_C5 :

	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11	f12	f13
0	46.0	1.0	0.0	120.0	249.0	0.0	0.0	144.0	0.0	0.8	2.0	0.0	3.0
1	57.0	0.0	0.0	140.0	241.0	0.0	1.0	123.0	1.0	0.2	1.0	0.0	3.0
2	48.0	1.0	2.0	124.0	255.0	1.0	1.0	175.0	0.0	0.0	2.0	2.0	2.0
3	46.0	1.0	2.0	150.0	231.0	0.0	1.0	147.0	0.0	3.6	1.0	0.0	2.0
4	63.0	1.0	0.0	130.0	254.0	0.0	0.0	147.0	0.0	1.4	1.0	1.0	3.0
5	60.0	0.0	3.0	150.0	240.0	0.0	1.0	171.0	0.0	0.9	2.0	0.0	2.0
6	50.0	1.0	0.0	150.0	243.0	0.0	0.0	128.0	0.0	2.6	1.0	0.0	3.0
7	51.0	1.0	0.0	140.0	261.0	0.0	0.0	186.0	1.0	0.0	2.0	0.0	2.0
8	60.0	1.0	0.0	125.0	258.0	0.0	0.0	141.0	1.0	2.8	1.0	1.0	3.0
9	64.0	1.0	3.0	170.0	227.0	0.0	0.0	155.0	0.0	0.6	1.0	0.0	3.0
10	54.0	1.0	2.0	150.0	232.0	0.0	0.0	165.0	0.0	1.6	2.0	0.0	3.0
11	44.0	1.0	1.0	120.0	263.0	0.0	1.0	173.0	0.0	0.0	2.0	0.0	3.0
12	53.0	1.0	2.0	130.0	246.0	1.0	0.0	173.0	0.0	0.0	2.0	3.0	2.0
13	47.0	1.0	2.0	108.0	243.0	0.0	1.0	152.0	0.0	0.0	2.0	0.0	2.0
14	69.0	1.0	3.0	160.0	234.0	1.0	0.0	131.0	0.0	0.1	1.0	1.0	2.0
15	47.0	1.0	2.0	108.0	243.0	0.0	1.0	152.0	0.0	0.0	2.0	0.0	2.0
16	51.0	1.0	0.0	140.0	261.0	0.0	0.0	186.0	1.0	0.0	2.0	0.0	2.0
17	61.0	1.0	2.0	150.0	243.0	1.0	1.0	137.0	1.0	1.0	1.0	0.0	2.0
18	51.0	1.0	2.0	125.0	245.0	1.0	0.0	166.0	0.0	2.4	1.0	0.0	2.0
19	47.0	1.0	2.0	108.0	243.0	0.0	1.0	152.0	0.0	0.0	2.0	0.0	2.0
20	69.0	1.0	3.0	160.0	234.0	1.0	0.0	131.0	0.0	0.1	1.0	1.0	2.0
21	60.0	0.0	0.0	150.0	258.0	0.0	0.0	157.0	0.0	2.6	1.0	2.0	3.0
22	54.0	0.0	2.0	108.0	267.0	0.0	0.0	167.0	0.0	0.0	2.0	0.0	2.0
23	64.0	1.0	3.0	170.0	227.0	0.0	0.0	155.0	0.0	0.6	1.0	0.0	3.0
24	44.0	1.0	2.0	140.0	235.0	0.0	0.0	180.0	0.0	0.0	2.0	0.0	2.0
25	69.0	1.0	2.0	140.0	254.0	0.0	0.0	146.0	0.0	2.0	1.0	3.0	3.0
26	65.0	0.0	2.0	155.0	269.0	0.0	1.0	148.0	0.0	0.8	2.0	0.0	2.0
27	43.0	1.0	0.0	150.0	247.0	0.0	1.0	171.0	0.0	1.5	2.0	0.0	2.0
28	55.0	1.0	1.0	130.0	262.0	0.0	1.0	155.0	0.0	0.0	2.0	0.0	2.0

29	65.0	1.0	0.0	110.0	248.0	0.0	0.0	158.0	0.0	0.6	2.0	2.0	1.0
30	59.0	0.0	0.0	174.0	249.0	0.0	1.0	143.0	1.0	0.0	1.0	0.0	2.0
31	50.0	1.0	0.0	150.0	243.0	0.0	0.0	128.0	0.0	2.6	1.0	0.0	3.0
32	57.0	0.0	1.0	130.0	236.0	0.0	0.0	174.0	0.0	0.0	1.0	1.0	2.0
33	58.0	1.0	2.0	105.0	240.0	0.0	0.0	154.0	1.0	0.6	1.0	0.0	3.0
34	56.0	1.0	2.0	130.0	256.0	1.0	0.0	142.0	1.0	0.6	1.0	1.0	1.0
35	45.0	1.0	0.0	115.0	260.0	0.0	0.0	185.0	0.0	0.0	2.0	0.0	2.0
36	44.0	1.0	2.0	140.0	235.0	0.0	0.0	180.0	0.0	0.0	2.0	0.0	2.0
37	41.0	0.0	2.0	112.0	268.0	0.0	0.0	172.0	1.0	0.0	2.0	0.0	2.0
38	48.0	1.0	1.0	130.0	245.0	0.0	0.0	180.0	0.0	0.2	1.0	0.0	2.0
39	51.0	0.0	2.0	130.0	256.0	0.0	0.0	149.0	0.0	0.5	2.0	0.0	2.0
40	54.0	1.0	0.0	110.0	239.0	0.0	1.0	126.0	1.0	2.8	1.0	1.0	3.0
41	63.0	1.0	3.0	145.0	233.0	1.0	0.0	150.0	0.0	2.3	0.0	0.0	1.0
42	57.0	1.0	1.0	124.0	261.0	0.0	1.0	141.0	0.0	0.3	2.0	0.0	3.0
43	58.0	0.0	0.0	170.0	225.0	1.0	0.0	146.0	1.0	2.8	1.0	2.0	1.0
44	62.0	0.0	0.0	140.0	268.0	0.0	0.0	160.0	0.0	3.6	0.0	2.0	2.0
45	54.0	1.0	0.0	140.0	239.0	0.0	1.0	160.0	0.0	1.2	2.0	0.0	2.0
46	63.0	0.0	2.0	135.0	252.0	0.0	0.0	172.0	0.0	0.0	2.0	0.0	2.0
47	60.0	0.0	3.0	150.0	240.0	0.0	1.0	171.0	0.0	0.9	2.0	0.0	2.0
48	59.0	1.0	0.0	138.0	271.0	0.0	0.0	182.0	0.0	0.0	2.0	0.0	2.0
49	60.0	0.0	0.0	150.0	258.0	0.0	0.0	157.0	0.0	2.6	1.0	2.0	3.0
50	62.0	0.0	0.0	140.0	268.0	0.0	0.0	160.0	0.0	3.6	0.0	2.0	2.0
51	47.0	1.0	2.0	130.0	253.0	0.0	1.0	179.0	0.0	0.0	2.0	0.0	2.0
52	69.0	0.0	3.0	140.0	239.0	0.0	1.0	151.0	0.0	1.8	2.0	2.0	2.0
53	69.0	0.0	3.0	140.0	239.0	0.0	1.0	151.0	0.0	1.8	2.0	2.0	2.0
54	50.0	0.0	0.0	110.0	254.0	0.0	0.0	159.0	0.0	0.0	2.0	0.0	2.0

Cluster_C6 :

	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11	f12	f13
0	55.0	1.0	0.0	160.0	289.0	0.0	0.0	145.0	1.0	0.8	1.0	1.0	3.0
1	66.0	0.0	2.0	146.0	278.0	0.0	0.0	152.0	0.0	0.0	1.0	1.0	2.0
2	42.0	1.0	1.0	120.0	295.0	0.0	1.0	162.0	0.0	0.0	2.0	0.0	2.0
3	51.0	0.0	2.0	120.0	295.0	0.0	0.0	157.0	0.0	0.6	2.0	0.0	2.0
4	52.0	1.0	3.0	152.0	298.0	1.0	1.0	178.0	0.0	1.2	1.0	0.0	3.0
5	58.0	1.0	0.0	114.0	318.0	0.0	2.0	140.0	0.0	4.4	0.0	3.0	1.0
6	48.0	1.0	0.0	124.0	274.0	0.0	0.0	166.0	0.0	0.5	1.0	0.0	3.0
7	56.0	0.0	0.0	200.0	288.0	1.0	0.0	133.0	1.0	4.0	0.0	2.0	3.0
8	64.0	1.0	2.0	125.0	309.0	0.0	1.0	131.0	1.0	1.8	1.0	0.0	3.0
9	60.0	0.0	2.0	102.0	318.0	0.0	1.0	160.0	0.0	0.0	2.0	1.0	2.0
10	46.0	1.0	0.0	140.0	311.0	0.0	1.0	120.0	1.0	1.8	1.0	2.0	3.0

11	46.0	1.0	0.0	140.0	311.0	0.0	1.0	120.0	1.0	1.8	1.0	2.0	3.0
12	55.0	1.0	0.0	160.0	289.0	0.0	0.0	145.0	1.0	0.8	1.0	1.0	3.0
13	64.0	0.0	2.0	140.0	313.0	0.0	1.0	133.0	0.0	0.2	2.0	0.0	3.0
14	58.0	0.0	1.0	136.0	319.0	1.0	0.0	152.0	0.0	0.0	2.0	2.0	2.0
15	45.0	1.0	0.0	142.0	309.0	0.0	0.0	147.0	1.0	0.0	1.0	3.0	3.0
16	58.0	1.0	1.0	120.0	284.0	0.0	0.0	160.0	0.0	1.8	1.0	0.0	2.0
17	60.0	1.0	0.0	140.0	293.0	0.0	0.0	170.0	0.0	1.2	1.0	2.0	3.0
18	54.0	0.0	1.0	132.0	288.0	1.0	0.0	159.0	1.0	0.0	2.0	1.0	2.0
19	54.0	0.0	1.0	132.0	288.0	1.0	0.0	159.0	1.0	0.0	2.0	1.0	2.0
20	58.0	1.0	1.0	120.0	284.0	0.0	0.0	160.0	0.0	1.8	1.0	0.0	2.0
21	46.0	1.0	0.0	140.0	311.0	0.0	1.0	120.0	1.0	1.8	1.0	2.0	3.0
22	51.0	0.0	0.0	130.0	305.0	0.0	1.0	142.0	1.0	1.2	1.0	0.0	3.0
23	65.0	1.0	3.0	138.0	282.0	1.0	0.0	174.0	0.0	1.4	1.0	1.0	2.0
24	68.0	1.0	2.0	118.0	277.0	0.0	1.0	151.0	0.0	1.0	2.0	1.0	3.0
25	68.0	1.0	2.0	180.0	274.0	1.0	0.0	150.0	1.0	1.6	1.0	0.0	3.0
26	35.0	1.0	0.0	126.0	282.0	0.0	0.0	156.0	1.0	0.0	2.0	0.0	3.0
27	48.0	1.0	0.0	124.0	274.0	0.0	0.0	166.0	0.0	0.5	1.0	0.0	3.0
28	60.0	0.0	2.0	102.0	318.0	0.0	1.0	160.0	0.0	0.0	2.0	1.0	2.0
29	58.0	0.0	3.0	150.0	283.0	1.0	0.0	162.0	0.0	1.0	2.0	0.0	2.0
30	48.0	0.0	2.0	130.0	275.0	0.0	1.0	139.0	0.0	0.2	2.0	0.0	2.0
31	58.0	1.0	1.0	120.0	284.0	0.0	0.0	160.0	0.0	1.8	1.0	0.0	2.0
32	59.0	1.0	3.0	170.0	288.0	0.0	0.0	159.0	0.0	0.2	1.0	0.0	3.0
33	39.0	1.0	2.0	140.0	321.0	0.0	0.0	182.0	0.0	0.0	2.0	0.0	2.0
34	51.0	0.0	2.0	140.0	308.0	0.0	0.0	142.0	0.0	1.5	2.0	1.0	2.0
35	51.0	1.0	0.0	140.0	299.0	0.0	1.0	173.0	1.0	1.6	2.0	0.0	3.0

MULTIVARIATE LINEAR REGRESSION FOR SUPERVISED LEARNING

```
import numpy as np
import pandas as pd

#load data
file_path= 'Student_Performance.csv'
data=pd.read_csv(file_path)

#map function
data['Extracurricular Activities']=data['Extracurricular
Activities'].map({'Yes':1,'No':0})

(m,n_plus_one) = data.shape
n = n_plus_one - 1 # Number of features (extracting target variable)

data.index = [f'X{j+1}' for j in range(m)]
data.columns = [f'f{i+1}' if i in range(n) else 'Y' for i in range(n+1)]

# split data into train_test ratio
train_data = data.sample(frac = 0.8)
test_data = data.drop(train_data.index)

# For Gradient Descent Algorithm
def predictions(X, theta):
    pred_Y = X.dot(theta)
    return pred_Y

def cost(X,Y,theta):
    m = len(Y)
    pred_Y = X.dot(theta)
    error = pred_Y - Y
    err = (1 / (2*m)) * np.mean(error**2)
    return err

def gradient_descent(X, Y, theta, learning_rate, tol):
    m = len(Y)  # Number of training examples
```

```
    stopping_criteria = 0
    while (stopping_criteria == 0):
        pred_Y = predictions(X, theta)
        error = pred_Y - Y

        # Update theta using the gradient
        gradient = (1/m) * np.dot(X.T, error)
        theta_new = theta - learning_rate * gradient

        err = cost(X, Y, theta_new)
        if(err < tol or np.array_equal(theta, theta_new)):
            stopping_criteria = 1
        theta = theta_new

        if(np.isnan(err)) :
            print('\nWarning : Losing minima. \nPlease, Adjust learning rate :')
            learning_rate = eval(input('New_Learning_Rate = '))
            theta = np.zeros(X.shape[1])

    return theta, err

# training the algorithm
# Assuming the last column is the target (Y), and the rest are features (X)
X_train = train_data.iloc[:, :-1].values  # Features (independent variables)
Y_train = train_data.iloc[:, -1].values   # Target (dependent variable)

# Add a column of ones to X for the intercept (bias term)
X_train = np.column_stack((np.ones(X_train.shape[0]), X_train))

# Initialize parameters
theta = np.zeros(X_train.shape[1])

learning_rate = eval(input('Learning_Rate = '))
tolerance = eval(input('Tolerance = '))
Optimal_theta, train_error = gradient_descent(X_train, Y_train, theta,
learning_rate, tolerance)

print(f'\nError in training : \n{train_error}')
print(f'\nOptimal_theta : \n{Optimal_theta}')
```

```
# testing the algorithm
X_test = test_data.iloc[:, :-1]
X_test = np.column_stack((np.ones(X_test.shape[0]), X_test))
Y_test = test_data.iloc[:, -1]

compare_Y = pd.DataFrame(Y_test).assign(pred_Y = predictions(X_test,
Optimal_theta))
test_error = cost(X_test, Y_test, Optimal_theta)

print(f'\nComparison of predicted output and actual output is :
\n{compare_Y}')
print(f'\nError in testing : \n{test_error}')
```

OUTPUT

```
Learning_Rate = 1e-3
Tolerance = 5e-4
/usr/local/lib/python3.10/dist-packages/numpy/core/_methods.py:118:
RuntimeWarning: overflow encountered in reduce
  ret = umr_sum(arr, axis, dtype, out, keepdims, where=where)
<ipython-input-1-4666391a60c2>:30: RuntimeWarning: overflow encountered
in square
  err = (1 / (2*m)) * np.mean(error**2)
<ipython-input-1-4666391a60c2>:42: RuntimeWarning: invalid value
encountered in subtract
  theta_new = theta - learning_rate * gradient

Warning : Losing minima.
Please, Adjust learning rate :
New_Learning_Rate = 3e-4

Error in training :
0.0004999990267157956

Optimal_theta :
```

[-21.84071117   2.61602072   0.94624123 -0.09068859 -0.22945847
0.03199385]

Comparison of predicted output and actual output is :

```
          Y      pred_Y
X9      61.0  62.328287
X13     27.0  28.479550
X14     33.0  37.474905
X20     63.0  60.601275
X24     57.0  60.250485
...      ...        ...
X9984   79.0  80.689864
X9990   27.0  28.643106
X9993   50.0  49.066850
X9994   55.0  55.112526
X9999   95.0  91.791976

[2000 rows x 2 columns]
```

Error in testing :
0.0019987144268516677

LOGISTIC REGRESSION FOR SUPERVISED LEARNING

```
import numpy as np
import pandas as pd

# load Excel data
file_path = "gender_classification_v7.xlsx"
data = pd.read_excel(file_path)

(m, n_plus_one) = data.shape
n = n_plus_one - 1  # Number of features (extracting target variable)

data.index = [f"X{j+1}" for j in range(m)]
data.columns = [f"f{i+1}" if i in range(n) else "Y" for i in range(n + 1)]

# map function
data['Y']=data['Y'].map({'Male':1,'Female':0})

# split data into train_test ratio
train_data = data.sample(frac = 0.8)
test_data = data.drop(train_data.index)

# Gradient Descent Algorithm
def predictions(X, theta):
    z = X.dot(theta)
    predictions = 1 / (1 + np.exp(-z))
    return predictions

def cost(X, Y, theta):
    m = len(Y)
    pred_Y = predictions(X, theta)
    err = (-1 / m) * np.sum((np.dot(Y.T, np.log(pred_Y))) + (np.dot((1 - Y).T,
np.log(1 - pred_Y))))
    return err

def gradient_descent(X, Y, theta, learning_rate, tol):
    m = len(Y)  # Number of training examples
    stopping_criteria = 0
    while stopping_criteria == 0:
```

```
        pred_Y = predictions(X, theta)
        error = pred_Y - Y

        # Update theta using the gradient
        gradient = (1 / m) * np.dot(X.T, error)
        theta_new = theta - learning_rate * gradient

        err = cost(X, Y, theta)
        if (err < tol):
            stopping_criteria = 1
        theta = theta_new

        if(np.isnan(err)) :
            print('\nWarning : Losing Minima. \nAdjust learning rate :')
            learning_rate = eval(input('New_Learning_Rate = '))
            theta = np.zeros(X.shape[1])

    return theta

def classification(X, theta, threshold):
    pred_Y = predictions(X, theta)
    y = []
    for i in range(len(pred_Y)):
        if pred_Y[i] >= threshold:
            y.append(1)
        else:
            y.append(0)
    y = np.array(y)
    return y

# training the algorithm
# Assuming the last column is the target (Y), and the rest are features (X)
X_train = train_data.iloc[:, :-1].values  # Features (independent variables)
Y_train = train_data.iloc[:, -1].values  # Target (dependent variable)

# Add a column of ones to X for the intercept (bias term)
X_train = np.column_stack((np.ones(X_train.shape[0]), X_train))

# Initialize parameters
theta = np.zeros(X_train.shape[1])
```

```
# parameters
learning_rate = eval(input('Learning_Rate = '))
tolerance = eval(input('Tolerance = '))

Optimal_theta = gradient_descent(X_train, Y_train, theta, learning_rate,
tolerance)
train_error = cost(X_train, Y_train, Optimal_theta)

print(f'Error in training : \n{train_error}')
print(f'\nOptimal Theta : \n{Optimal_theta}')

# testing the algorithm
X_test = test_data.iloc[:, :-1].values
X_test = np.column_stack((np.ones(X_test.shape[0]), X_test))
Y_test = test_data.iloc[:, -1].values

classified_Y = classification(X_test, Optimal_theta, threshold = 0.5)
compare_Y = pd.DataFrame(Y_test, columns = ['Actual_Y'], index =
test_data.index).assign(Classified_Y = classified_Y)
test_error = cost(X_test, Y_test, Optimal_theta)

print(f'\nComparison of classified output and actual output : \n{compare_Y}')
print(f'\nError in testing : \n{test_error}')
```

OUTPUT

```
Learning_Rate = 0.1
Tolerance = 0.075

Error in training :
0.07499991198803457

Optimal Theta :
[-12.34197092  -0.67315782   0.30278229   0.26405517   4.06981646
   3.63727776   3.50285441   3.8324567 ]
```

Comparison of classified output and actual output :

	Actual_Y	Classified_Y
X8	0	0
X9	0	1
X12	1	1
X20	1	1
X21	1	1
...	...	...
X4954	0	0
X4965	1	1
X4969	1	1
X4973	0	0
X4990	0	0

[1000 rows x 2 columns]

Error in testing :
0.09093105849610583