Homework 3

Farida Aliyeva

Importing the necessary libraries and parsing the data to get only valuable data.

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.metrics import silhouette\_score

info = pd.read\_csv(r'C:\Users\ASUS\Desktop\python\k-means.csv', header = None, sep = '\t')

for id, row in info.iterrows():

    row[0]=float (row[0].replace(',','.'))

    row[1]=float (row[1].replace(',','.'))

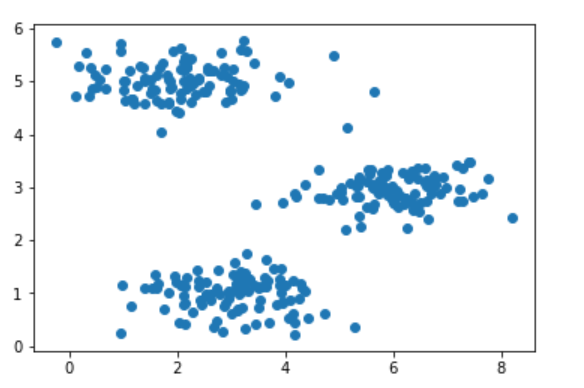
X1 = info.iloc[:,[0]].values

X2 = info.iloc[:,[1]].values

data\_n=info.values

plt.scatter(X1,X2)

Output:



K = 3 #centroids

m\_iters = 10

#initializes centroids with random values

def initialize\_cents(data, K):

    m = data.shape[0]

    n = data.shape[1]

    cents = np.zeros((K,n))

    for i in range(K):

        cents[i] = data[np.random.randint(0,m+1),:]

    return cents

# finds the nearesr centroids

def nearest\_cents(data, cents):

    idx = np.zeros([data.shape[0], 1])

    K = cents.shape[0] # num of centroids

    m = data.shape[0]

    #cost function

    for i in range(m):

        c = -1 # index of closest centroid

        min\_distance = np.inf # distance to nearest centroid(np.inf is positive infinity)

        for k in range(K):

            dis = np.sum((data[i,:] - cents[k,:]) \*\* 2)

            if dis < min\_distance:#swapping to find the closest centroid

                min\_distance = dis

                c = k

        idx[i] = c

    return idx

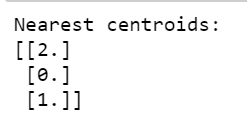
first\_cents = initialize\_cents(data\_n, K)

idx = nearest\_cents(data\_n, first\_cents)

print('Nearest centroids:')

print(idx[0:3])

Output:

Nearest centroids for a random initial centroids

#caluclates the new centroids

def calc\_cents(data, idx, K):

    m, n = data.shape

    centroids = np.zeros([K, n])

    for k in range(K):

        C = np.sum(idx==k)

        idxk = (idx==k).astype(int)

        data\_k = data \* idxk

        # the mean of a new centroid is calculated here

        mn = (1/C) \* np.sum(data\_k, axis=0)

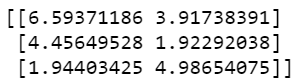
        centroids[k] = mn

    return centroids

centroids = calc\_cents(data\_n, idx, K)

print(centroids)

Output:

New centroids for random initial centroids

def plt\_k\_means(data, idx, cents\_history, terminated=False):

    cl1 = data\_n[np.where(idx == 0)[0],:]

    cl2 = data\_n[np.where(idx == 1)[0],:]

    cl3 = data\_n[np.where(idx == 2)[0],:]

    fig, ax = plt.subplots(figsize=(5,5))

    ax.scatter(cl1[:,0], cl1[:,1], marker="o",facecolors="none",edgecolors="r", label='Cluster 1')

    ax.scatter(cl2[:,0], cl2[:,1], marker="o",facecolors="none",edgecolors="g", label='Cluster 2')

    ax.scatter(cl3[:,0], cl3[:,1], marker="o",facecolors="none",edgecolors="b", label='Cluster 3')

    ax.legend()

    plt.plot(cents\_history[:, :, 0], cents\_history[:, :, 1], 'x-', c='k')

    plt.show()

def k\_means(data, init\_cents, max\_iters, plot\_progress=False):

    m, n = data.shape

    K = init\_cents.shape[0]

    centroids = init\_cents

    idx = np.zeros([m, 1])

    cents\_history = np.zeros([max\_iters+1, centroids.shape[0], centroids.shape[1]])

    cents\_history[0] = init\_cents

    print('Before:')

    plt\_k\_means(data, idx, cents\_history)

    #k means

    for i in range(max\_iters):

        idx = nearest\_cents(data, centroids)

        centroids = calc\_cents(data, idx, K)

        cents\_history[i+1] = centroids

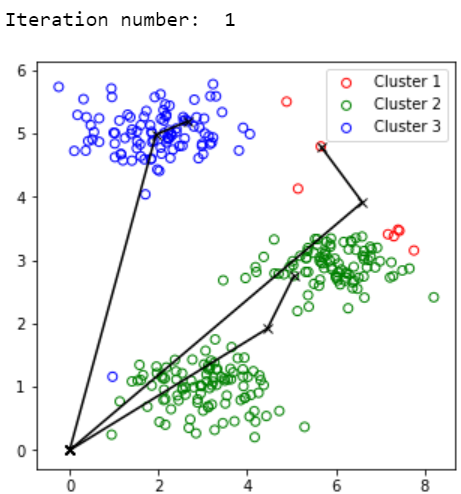
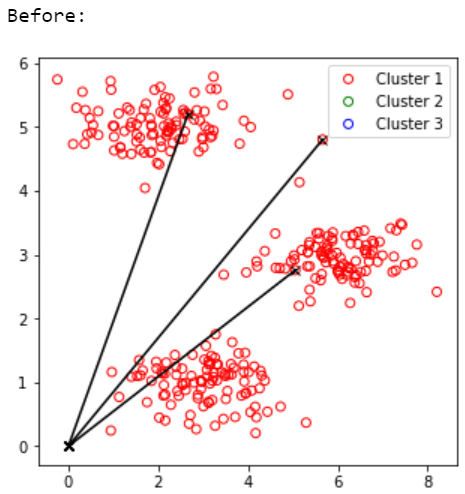
        print('Iteration number: ', i+1)

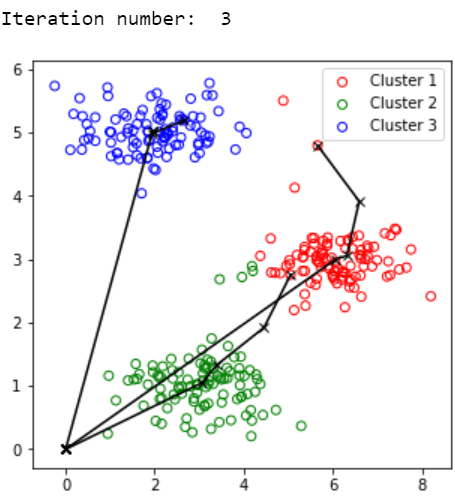
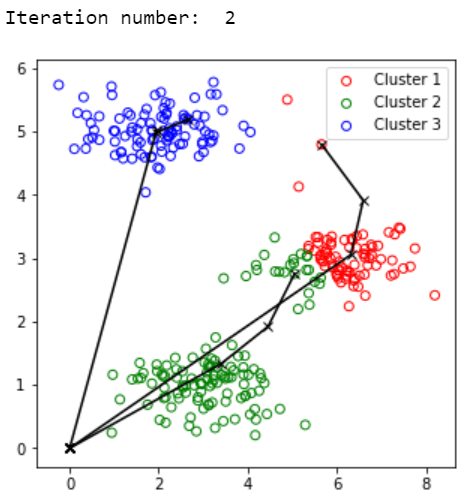
        plt\_k\_means(data, idx,cents\_history)

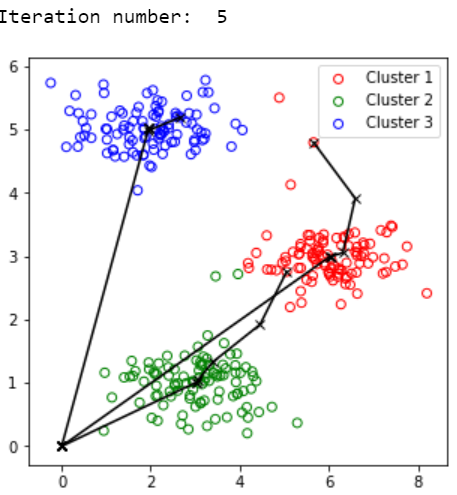
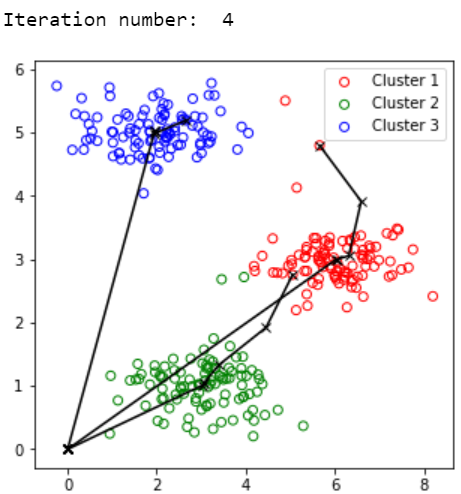
    return centroids, idx

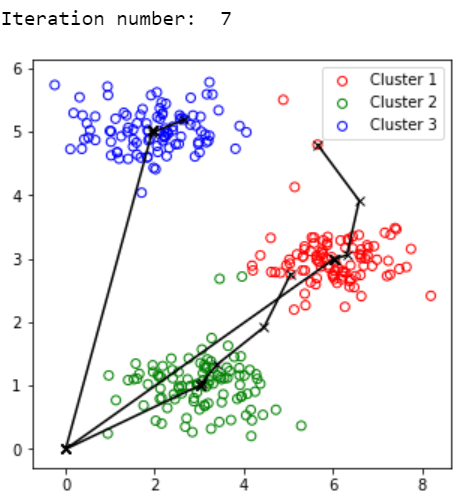
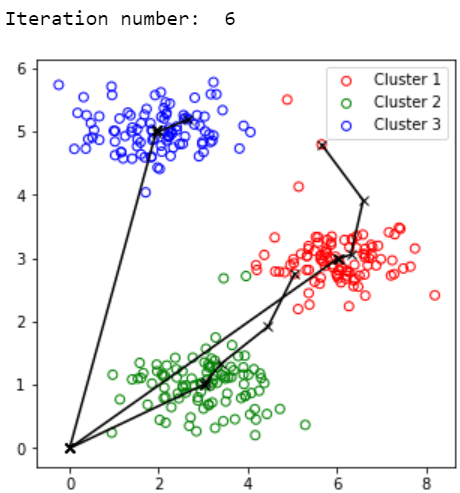
centroids, idx = k\_means(data\_n, first\_cents, m\_iters)

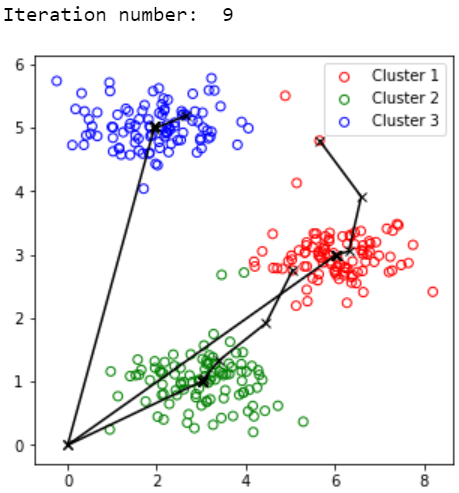
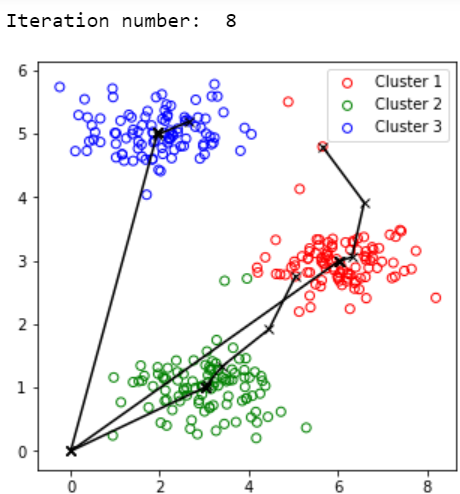
Output:

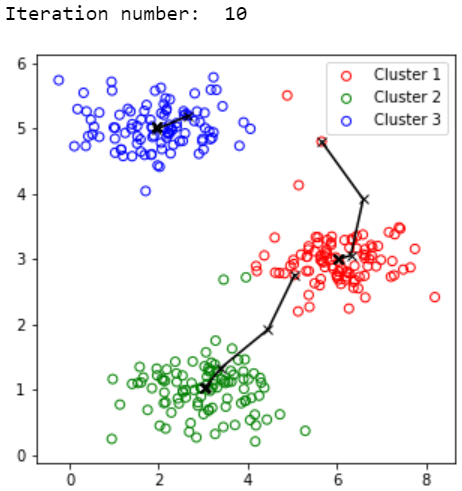












#to plot the graph that was in assignment instrcution pdf

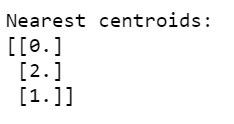
first\_cents = np.array([[3,3],[6,2],[8,5]])

idx = nearest\_cents(data\_n, first\_cents)

print('Nearest centroids:')

print(idx[0:3])

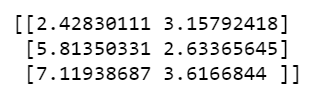
Output:

Nearest centroids for the given samples

centroids = calc\_cents(data\_n, idx, K)

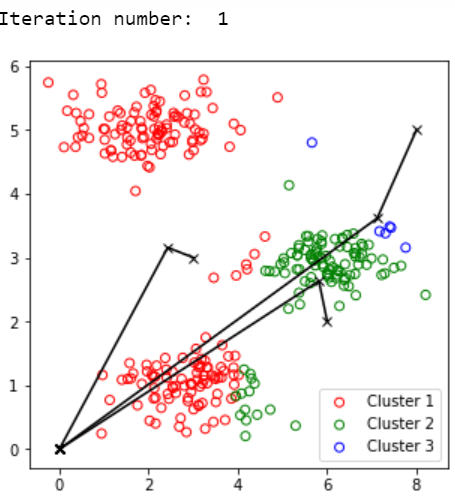
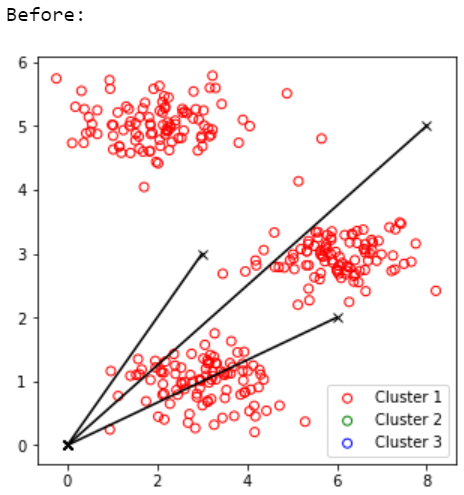
print(centroids)

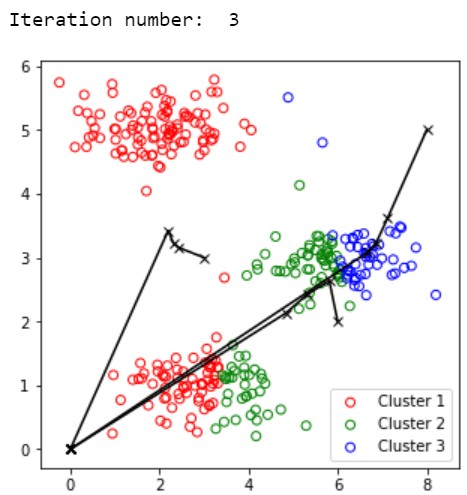
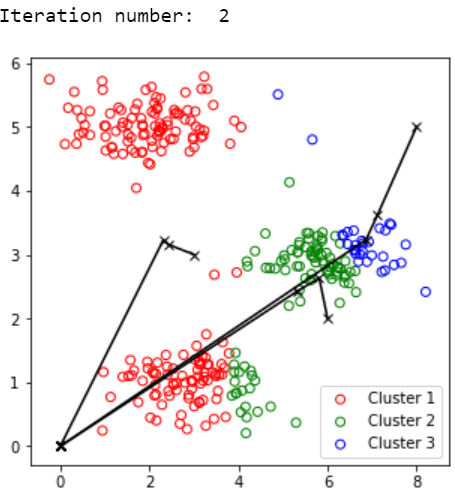
Output:

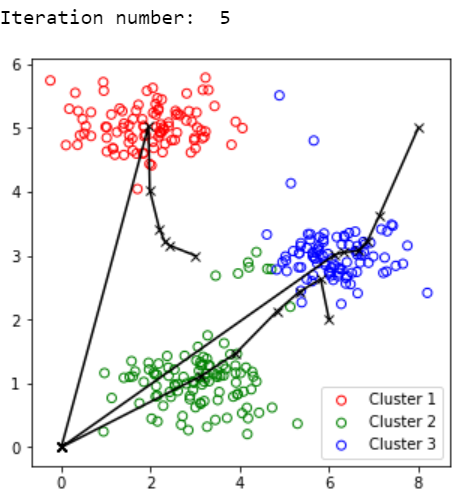
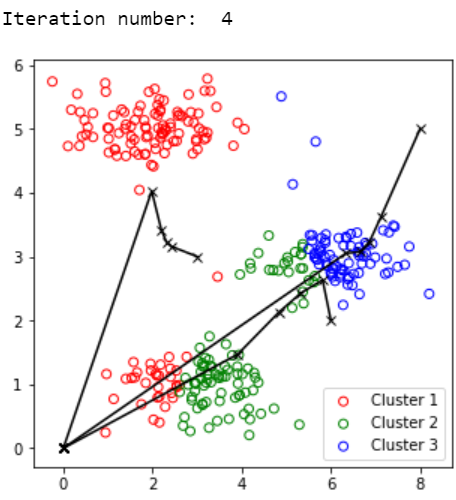


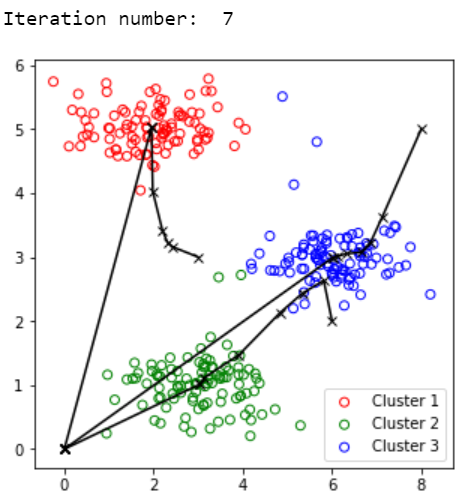
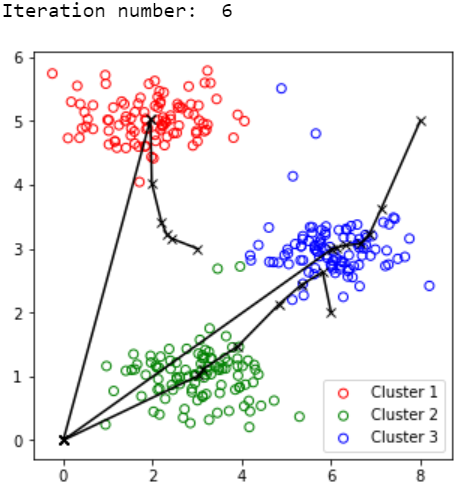
centroids, idx = k\_means(data\_n, first\_cents, m\_iters, plot\_progress=True)

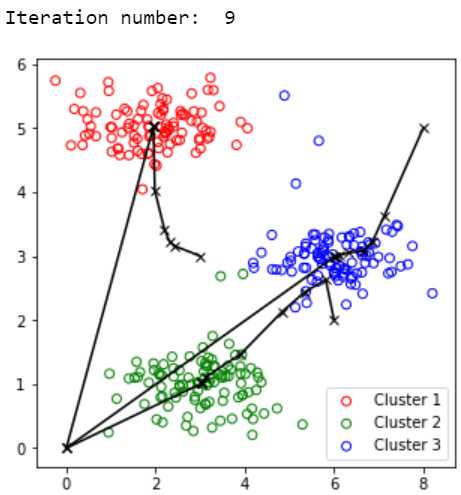
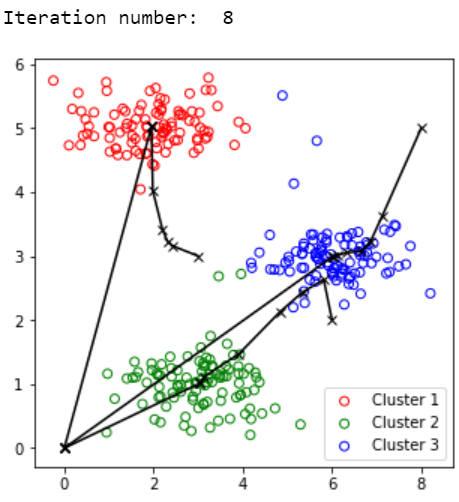
Output:

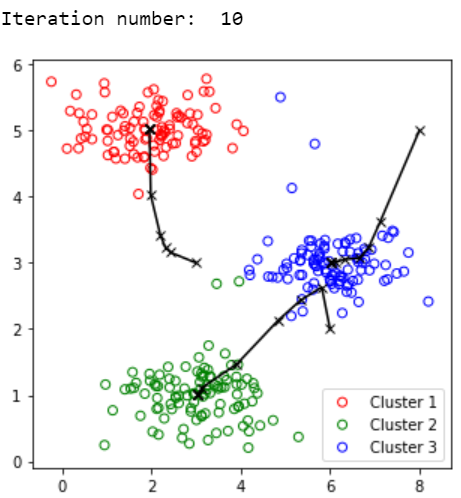










Exactly the same as in as instructions pdf

#calculating the cost

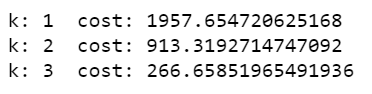
for k in range (1, K+1):

    kmean= KMeans(n\_clusters=k, random\_state=1).fit(data\_n[:, :])

    cost = kmean.inertia\_

    print ("k:",k, " cost:", cost)

Output:

Cost is decreasing which is good.

Using Library

#Kmeans using scikit-learn library

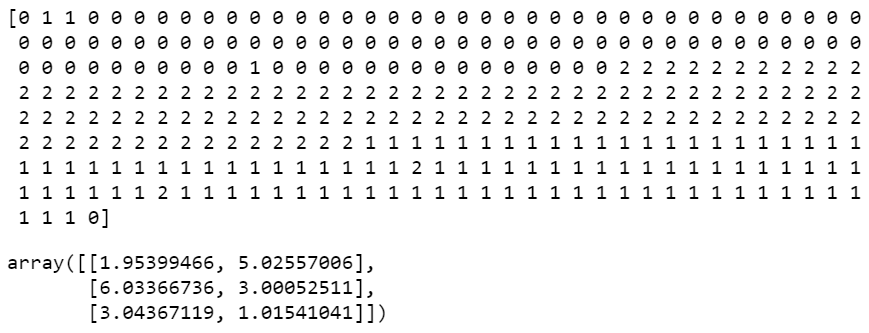
kmean = KMeans(n\_clusters=3)

y\_kmean = kmean.fit\_predict(data\_n)

print(y\_kmean)

kmean.cluster\_centers\_

Output:



plt.scatter(data\_n[:,0],data\_n[:,1], c=y\_kmean)

plt.scatter(

    kmean.cluster\_centers\_[:, 0], kmean.cluster\_centers\_[:, 1],

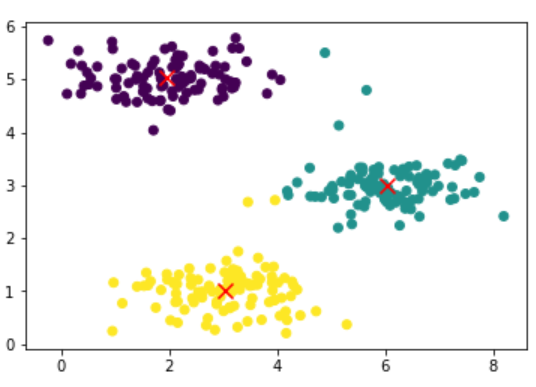
    s=100, marker='x',

    c='red', edgecolor='black',

    label='centroids'

)

Output:



#Silhouette score

rg = list (range(2,11))

print ("Clusters num: ", rg)

for n\_clusters in rg:

    clr = KMeans (n\_clusters=n\_clusters).fit(data\_n)

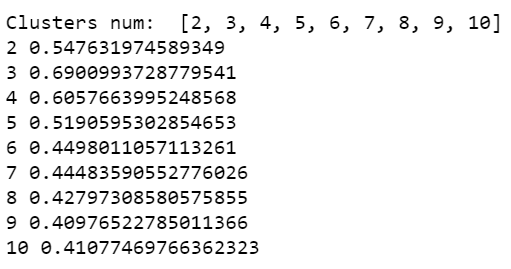
    preds = clr.predict(data\_n)

    centers = clr.cluster\_centers\_

    score = silhouette\_score (data\_n, preds, metric='euclidean')

    print (n\_clusters, score)

Output:



#Testing the data

kmean.predict([[0, 3], [5, 2]])

Output:



kmean.predict([[1, 1], [5, 2]])

Output:



kmean.predict([[5, 2], [2, 9]])

Output:



kmean.predict([[5, 3], [3, 3]])

Output:



kmean.predict([[6, 1], [2, 8]])

Output:

