IE 529

COMPUTATIONAL ASSIGNMENT 2

Karthik Venkata

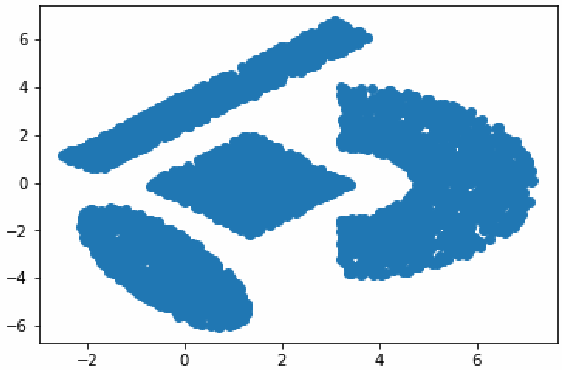
NET ID-kvn3

Question 1: Lloyd's Algorithm

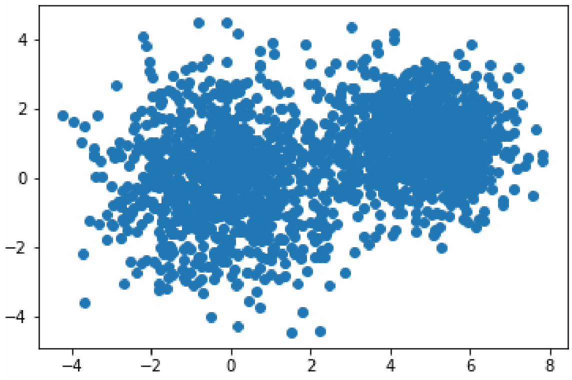
1. Output clustering results for both sets of data
2. Result evaluation for a range of k values-
   We could identify different types of clusters and depending on how many natural clusters, identifying them based on the value of k. In the case of ShapedData.csv, no clustering based on natural clustering was observed. In spite of varying the values of k from 2, the number of apparent clusters being distinguished are varying but the number of natural clusters still remain the same.
3. For the K-means algorithm, how was convergence tested for and why was this chosen as a test.
   The convergence was tested based on the distance function which evaluated the average distance from the cluster centres and as the different values of k were used and the iterations performed, one could test for the convergence criteria.

4. Brief Summary paragraph of findings
   The clustering was done for different values of k and then natural clusters were observed for both the data sets.

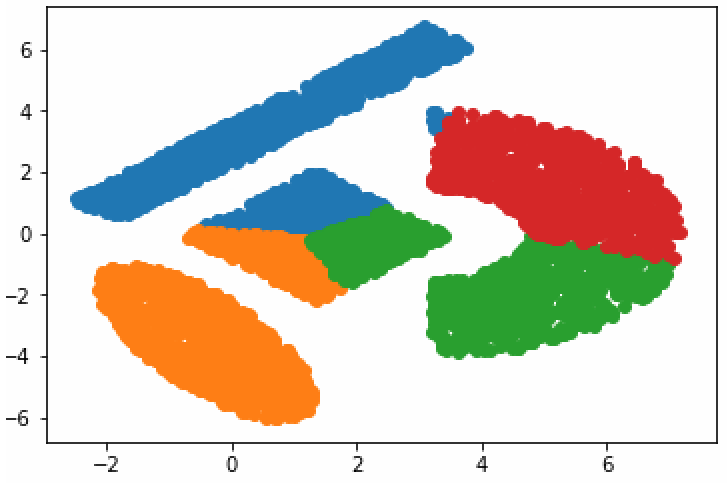
1. Scatter plot of ShapedData.csv



Scatter plot of Clustering.csv

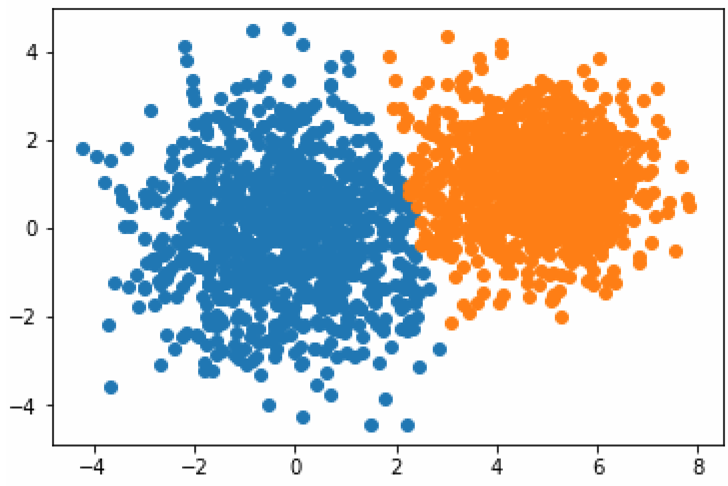


K=4 ShapedData.csv Scatter plot



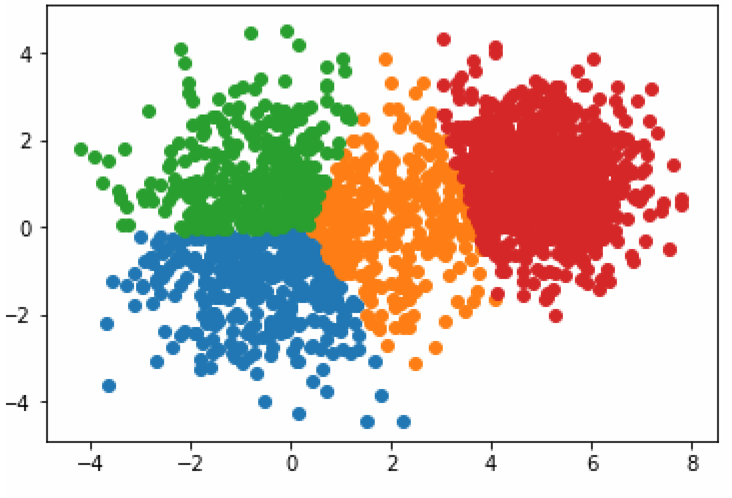
Natural Clusters – 4

K=2 Clustering.csv Scatter plot



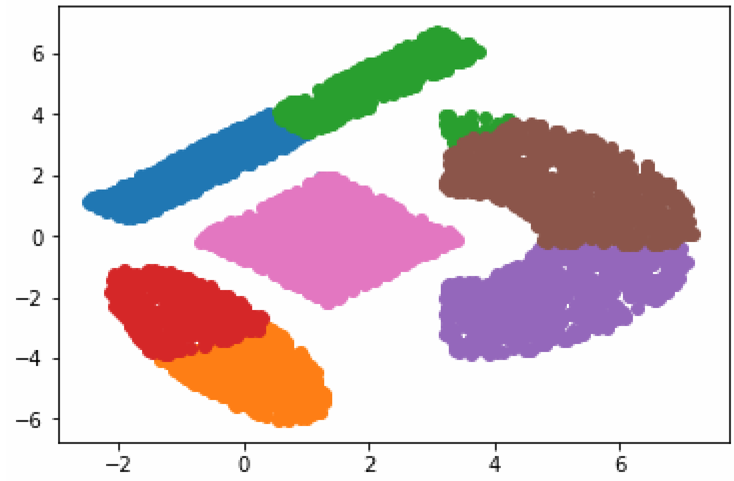
Natural Clusters-2

K=4 Clustering.csv Scatter plot



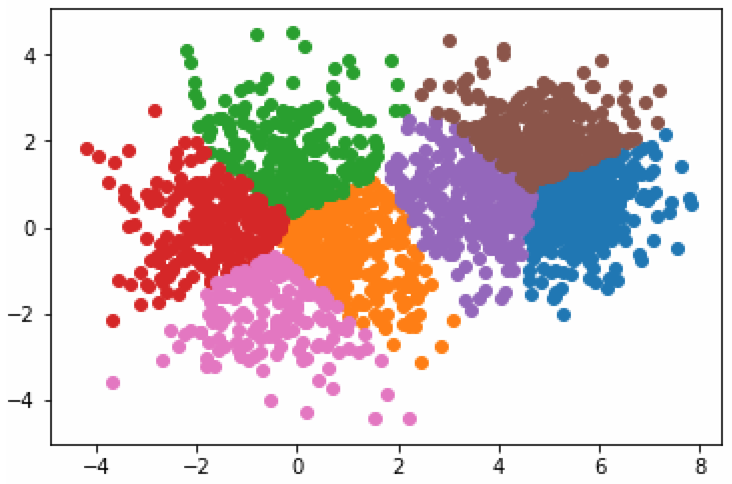
Natural Clusters -2

K=7 ShapedData.csv



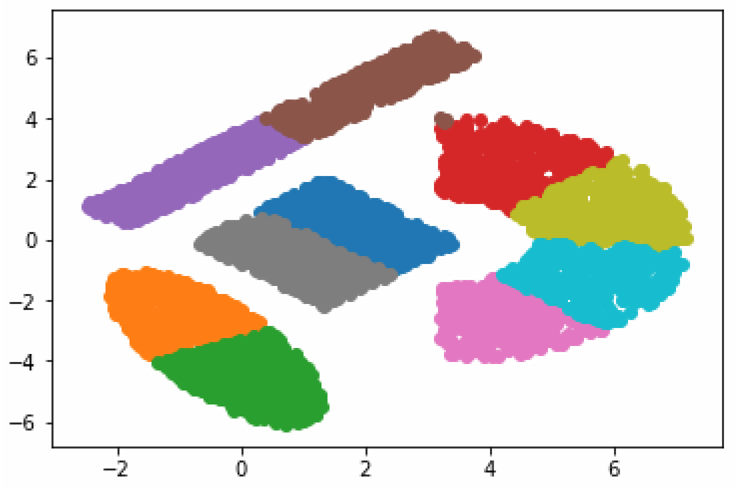
Natural Clusters-4

K=7 Clustering.csv



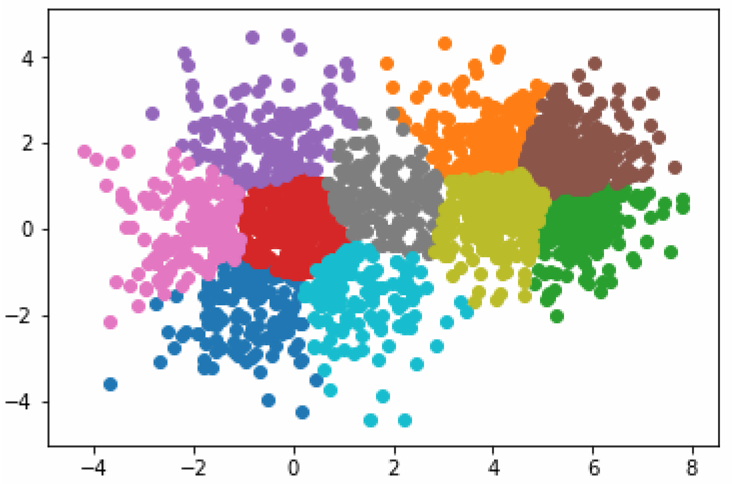
Natural Clusters-2

K=10 ShapedData.csv



Natural Clusters-4

K=10 Clustering.csv



Natural Clusters-2

Pseudocode for K Means Clustering Algorithm

Input: k(The number of clusters)

D(a set of lift ratios)

Output: a set of k clusters

Method:

Arbitrarily choose k objects from D as the initial cluster centers;

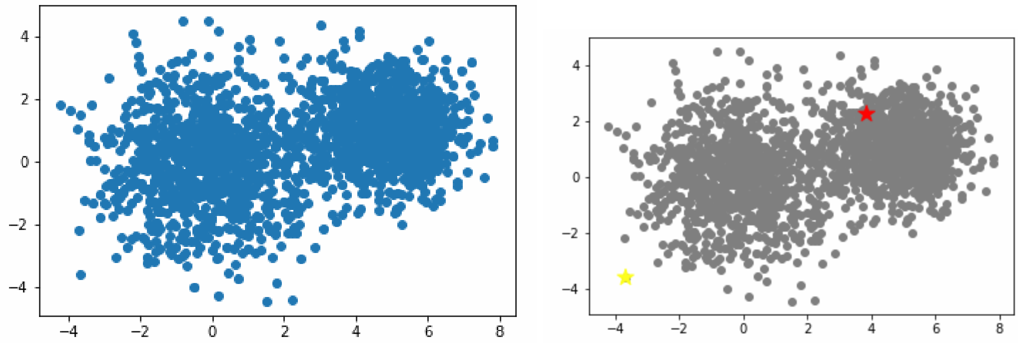
Repeat:

1. (re)assign each object to the cluster to which the object is the most similar, based on the mean value of the objects in the cluster;
2. Update the cluster means, i.e. calculate the mean value of the objects for each cluster

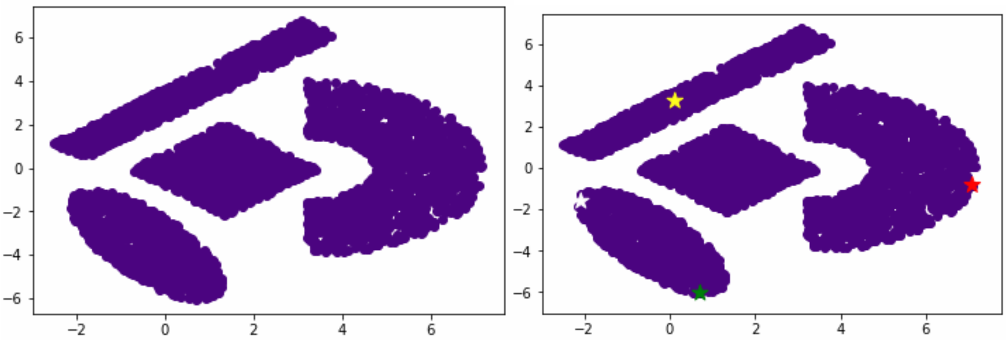
Until no change;

Question 2:

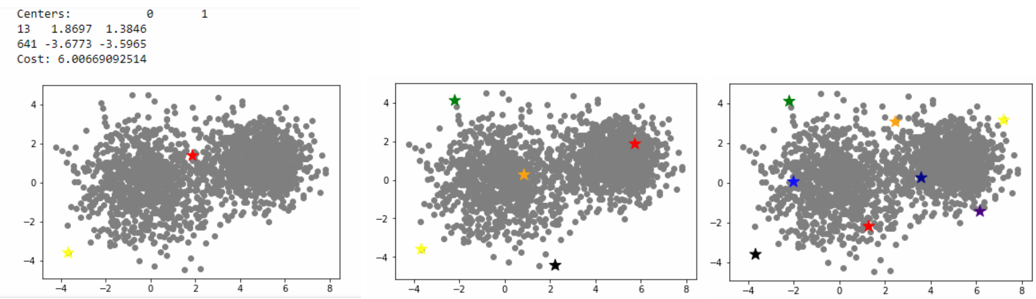
Scatter plot of Data 1(Clustering.csv)



Scatter plot of Data2(ShapedData.csv)



Plots for k =2, 5 and 8



Natural Clusters-2

Pseudocode for Greedy Algorithm

Input: Metric space (V, d), $k \in \mathbb{N}$.
Output: $F \subseteq V, |F| = k$, with minimum max distance to elements of V.

$F \leftarrow \{u\}$, for $u \in V$ arbitrary
while $|F| < k$ **do**
  Let $u \in V \setminus F$ be the element maximizing $d(u, F)$.
  $F \leftarrow F \cup \{u\}$
end while
return F

- Pick an arbitrary point $\bar{c}_1$ into C_1
- For every point $p \in \mathbf{P}$ compute $d_1[p]$ from $\bar{c}_1$
- Pick the point $\bar{c}_2$ with highest distance from $\bar{c}_1$. (This is the point realizing $r_1 = \underset{p \in P}{max}\, d_1[p]$)
- Add it to the set of centers and denote this expanded set of centers as C_2. Continue this till k centers are found

- Choose the first centre arbitrarily.
- At every step, choose the vertex that is furthest from the current centre to become a centre.
- Continue until k centres are chosen

Pseudocode for Single Swap:

BEGIN FUNCTION SWAP

INPUT x, y

OUTPUT x, y (values switched)

t = x

x = y

y = t

RETURN

END FUNCTION swap

Pseudocode for Spectral Clustering:

1. project your data into R^n
2. define an *Affinity* matrix A , using a Gaussian Kernel K or say just an Adjacency matrix (i.e. $A_{i,j} = \delta_{i,j}$)
3. construct the Graph Laplacian from A (i.e. decide on a normalization)
4. solve an Eigenvalue problem , such as $Lv = \lambda v$ (or a Generalized Eigenvalue problem $Lv = \lambda Dv$)
5. select k eigenvectors $\{v_i, i = 1, k\}$ corresponding to the k lowest (or highest) eigenvalues $\{\lambda_i, i = 1, k\}$, to define a k-dimensional subspace P^tLP
6. form clusters in this subspace using, say, k-means

SHORT ANALYSIS OF THE COMPUTATIONAL EFFORT

Short analysis:
Lloyd's algorithm: The computational complexity of each iteration is O(Nkd) , complexity of the worst case is : $O(n^{Kd+1}\log n)$
Greedy K Centers algorithm: The computational complexity of each iteration is O(Nkd)
Single-swap algorithm: The algorithm will terminate within $\binom{N}{K}$ steps.
Spectral clustering: The computational complexity of the similarity matrix is $O(d^2N^2)$, and the complexity of eigenvalues and eigenvectors is $O(N^3)$
Expectation Maximization: The computational complexity should be $O(mN^3)$, where m is the number of iterations.

The single swap algorithm is the most time consuming one.

Appendix[]:

Question 1: Lloyd's Algorithm and K-Means Clustering code

```
import random
import math
import numpy as np
def random_centers(data,k):
data_ = np.array(data)
centres_list = []
for x in range(0,k):
temp_rand = np.random.randint( 0,high = 1999)
temp = data_[temp_rand]
centres_list.append(temp)
data_ = np.delete(data_,temp,axis = 0)
return((centres_list))
def assign_data_centers(data, k,centres):
thePartition = [[] for _ in range(0,k)] # list of k empty lists
c = np.array(centres)
dp = np.array(data)
for a in dp:
temp_norm = np.linalg.norm((c-a),axis =1 )
minn = temp_norm.min()
temp_list = list(temp_norm)
thePartition[temp_list.index(minn)].append(a)
return thePartition
def revalaute_centres(data_with_centres,k):
data = data_with_centres
new_centres=[]
for x in data:
temp = np.array(x)
temp_m = temp.mean(axis = 0)
new_centres.append(temp_m)
return(new_centres)

import pandas as pd
data = pd.read_csv('ShapedData.csv',header = None)

k = 4
C = random_centers(data,k)
D_C_2 = assign_data_centers(data,k,C)
tol = 1
while tol > 0.00001:
N_C = revalaute_centres(D_C_2,k)
D_C_1 = D_C_2
D_C_2 = assign_data_centers(data,k,N_C)
temp_sum = 0
tol = tol/10

import matplotlib.pyplot as plt
plt.plot()
for i in D_C_2:
plt.scatter(pd.DataFrame(i)[0],pd.DataFrame(i)[1])
plt.show()

import pandas as pd
data_2 = pd.read_csv('clustering.csv', header=None)
```

```
k = 4
C = random_centers(data_2,k)
D_C_2 = assign_data_centers(data_2,k,C)
tol = 1
while tol > 0.00001:
N_C = revalaute_centres(D_C_2,k)
D_C_1 = D_C_2
D_C_2 = assign_data_centers(data_2,k,N_C)
temp_sum = 0
tol = tol/10
```

```
import matplotlib.pyplot as plt
plt.plot()
for i in D_C_2:
plt.scatter(pd.DataFrame(i)[0],pd.DataFrame(i)[1])
plt.show()
```

ALTERNATIVELY the Distance function in the K Means Algorithm may also be written as follows:

```
def k_means(data, centres, k, tol):
    """
    Kmeans algorihtm
    input data
    output: new clusters, centres

    """
    n = data.shape[0]
    e = 100
    c = 0
    for epoch in range(0,e):
        #calculated distances
        print (centres.shape)

        dist = distance_cal (data, centres)
        #assign cluster labels
        cluster_labels = np.argmin(dist, axis = 1) # n*1
        #new centroids
        centres_updates = np.zeros(shape = centres.shape)
```

```
        for j in range(0, k):
            points = []
            if sum(cluster_labels == j) == 0:
                centres_updates[j] = centres[j]
            else:
              # print (data[0])
                for d,label in zip(data, cluster_labels):
                    if label == j:
                        points.append(list(d))

                centres_updates[j] = np.mean(points,axis = 0)
                np.mean(data[cluster_labels == j, :], axis=0)

        if (np.mean(np.amin(euclidean_distances(centres, centres_updates), axis = 1)) <= tol):
            print ("Convergence, iterations:", c)
            break
        c+=1
        centres = centres_updates

    return({"Iterations":c,     "Centroids": centres,
            "Labels": cluster_labels, "Cost": (np.mean(np.amin(euclidean_distances(data, centres), axis =
1)))})
```

Question 2[]:

```
#Importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import random as rd

from sklearn.metrics.pairwise import euclidean_distances
```

```
data1 = pd.read_csv("C:/Users/Karthik/Desktop/IE_529/clustering.csv", header = None)
data2 = pd.read_csv("C:/Users/Karthik/Desktop/IE_529/ShapedData.csv", header = None)
def greedy_kcenters( data1, k ):



    copy = data1.copy()
    Init_center = copy.sample( 1 )
    copy.drop(Init_center.index,inplace = True)
    copy.index = list(range( copy.shape[0] ) )

    while Init_center.shape[0] < k:

        #Get index of data point that has maximum minimum distance from any centre
        ind = np.argmax(np.amin( euclidean_distances(copy, Init_center), axis=1 ))

        #Append data in temp_df at index ind into C
        Init_center = Init_center.append( copy.loc[ind] )

        #Remove that row from temp_df
        copy.drop(ind,inplace = True)
        #and change indices to 0,1,2,...,n-1
        copy.index = list( range( copy.shape[0] ) )
    del copy
    return( Init_center, np.amax(np.amin(euclidean_distances( data1, Init_center ), axis=1 )) )
plt.scatter(data1[0],data1[1])
plt.show()
centers, dist_cost = greedy_kcenters( data1, 2 )
print("Centers:", pd.DataFrame(centers))
print("Cost:", dist_cost)
plt.scatter(data1[0], data1[1], color = 'grey')
plt.scatter(centers[0],centers[1], marker = "*", color = ['red', 'yellow'], s =180)
plt.show()
plt.scatter(data1[0], data1[1], color = 'grey')
plt.scatter(centers[0],centers[1], marker = "*", color = ['red', 'yellow'], s =180)
plt.show()
centers2, dist_cost2 = greedy_kcenters( data2, 4 )
```

```
print("Centers:", pd.DataFrame(centers2))
print("Cost:", dist_cost2)
norm = plt.Normalize()
plt.scatter(data2[0], data2[1], color = 'indigo')
plt.scatter(centers2[0],centers2[1], marker = "*", color = ['yellow', 'green','red','white'], s =150)
plt.show()
k = [2,4,5,7,8,9]
for i in k:
    centers, dist_cost = greedy_kcenters( data1, i )
    print("Centers:", pd.DataFrame(centers))
    print("Cost:", dist_cost)
    plt.scatter(data1[0], data1[1], color = 'grey')
    plt.scatter(centers[0],centers[1], marker = "*", color = ['red', 'yellow','green','black', 'orange','indigo', 'blue', 'darkblue', 'lightgreen', 'purple'], s =180)
    plt.show()
def sswap_centers( data, k ):

    centers,dist = greedy_kcenters( data, k )

    C0 = pd.DataFrame(euclidean_distances( data, centers )).min(axis=1).sum()
    print("Gk centers cost= ", C0)

    counter = centers.shape[0]

    beginning = False

    while counter > 0:

        for i in range( centers.shape[0] ):
            if beginning == True:
                beginning = False
                break

            #Create a copy of the dataframe df without the centers Q
            copy = pd.concat( [data,centers] )
            copy.drop_duplicates(keep=False)
```

```
        copy.index = list( range( copy.shape[0] ) )

        #Swap
        while copy.shape[0] != 0:
            centers_new = pd.concat( [ centers.drop(centers.index[i]), copy.loc[[0]] ] )

            #Get cost of new centers
            Cost_new =  pd.DataFrame(euclidean_distances( data, centers_new )).min(axis=1).sum()

            #Check for reduced cost of more than gamma(which is taken to be 0.05)
            if Cost_new <= 0.95 * C0:
                print('1')
                centers = centers_new.copy()
                print("SS_cost= ", Cost_new)
                copy = pd.concat( [data,centers] )
                copy.drop_duplicates(keep=False)
                copy.index = list( range( copy.shape[0] ) )
                beginning = True
                counter = centers.shape[0]
                break
            else:
                copy.drop(0, inplace=True)
                copy.index = list( range( copy.shape[0] ) )
        if beginning == False:
            print(counter)
            counter -= 1

    #Return the centers
    return( centers )
sscenters = sswap_centers(data1,2)
plt.scatter(data1[0], data1[1], color = 'grey')
plt.scatter(sscenters[0],sscenters[1], marker = "*", color = ['red', 'yellow'], s =180)
plt.show()
sscenters2 = sswap_centers(data2,2)
plt.scatter(data2[0], data2[1], color = 'grey')
plt.scatter(sscenters2[0],sscenters2[1], marker = "*", color = ['red', 'yellow'], s =180)
```

```
plt.show()
```