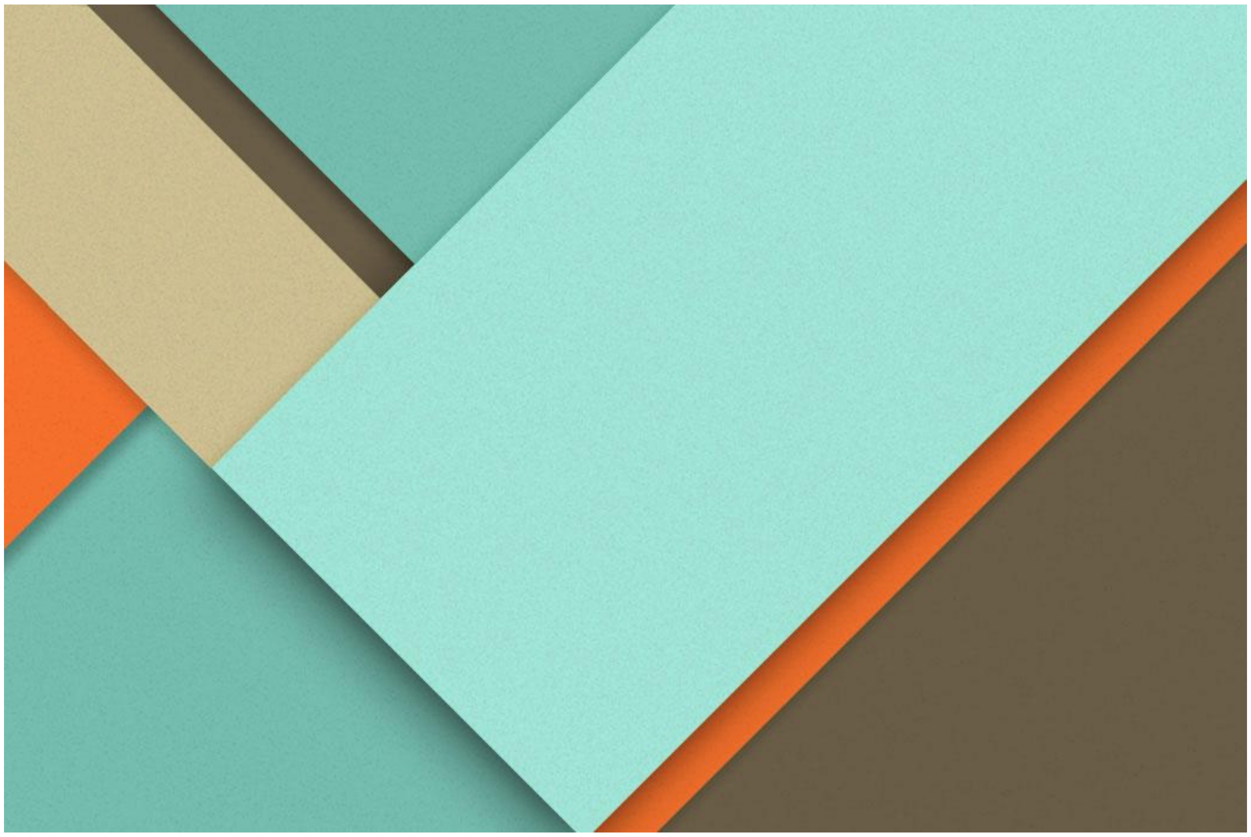


Assignment 1: K-Means Clustering

21.11.2021

—

Sevde SARIKAYA
2017400081

Description

In this project, we are expected to implement a k-means clustering algorithm by using a generated 2D data. To see how our algorithm works or if it works properly, we plot objective function vs iteration count for each iteration. We are also expected to implement a finding best K algorithm by doing research. In the next parts of the documentation, the code is explained and there are example plots of clusters.

1.Data Generation

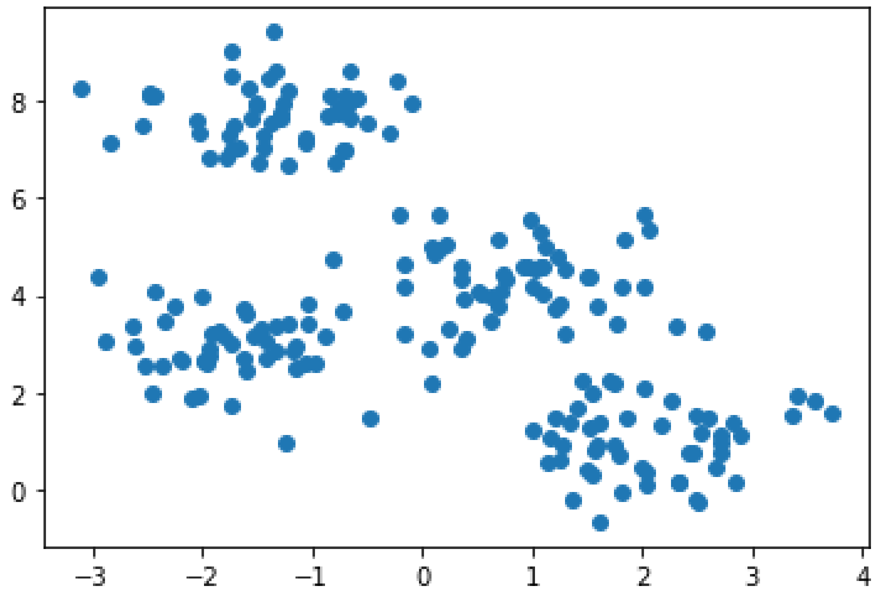
I generated my 2D simple data by using make_blobs from sklearn.datasets.

```
def generateData(k):

   data, clus = make_blobs(n_samples=200, centers=k, n_features=2, cluster_std =
0.7,random_state=0)

   return data
```

For k=4, the generated data:



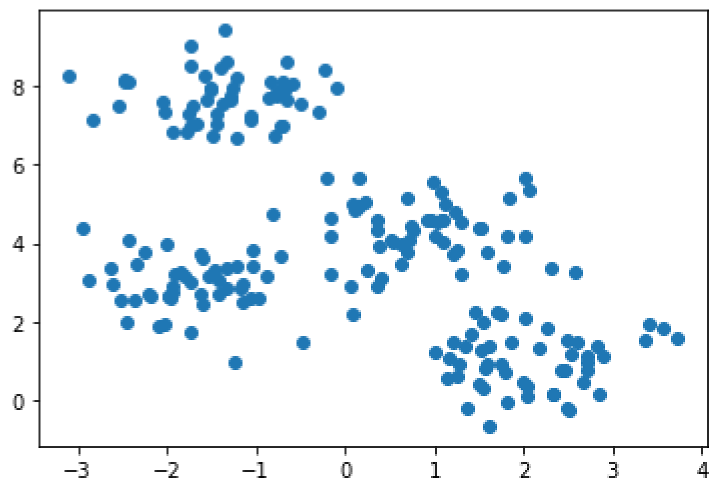
2. K-Means Algorithm

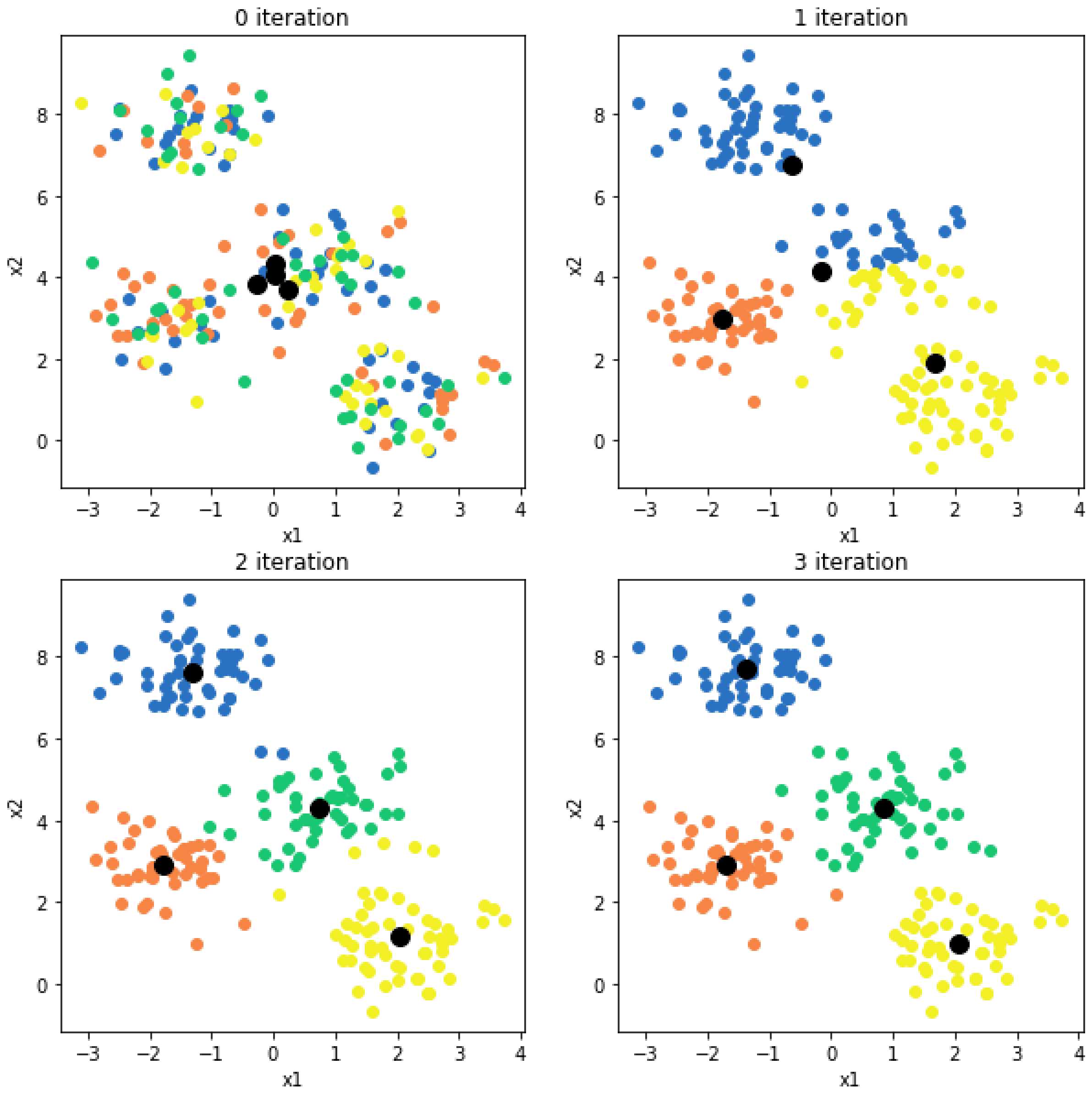
1. Randomly assigns data points to a cluster.
2. Generates centroids from the data points (because we don't want to choose a point away from the data. Otherwise there might be no data assigned to this centroid at the end.)
3. Calculates cluster centroids by taking averages of the points in that cluster
4. Assigns data points to the closest centroid by calculating the euclidean distance to each centroid.
5. Repeat 3 and 4 until the euclidean distance between new centroid and previous centroid is smaller than 0.000000000001.

My algorithm also checks if a cluster has no point. If it doesn't have points, it runs the algorithm again until every cluster has points.

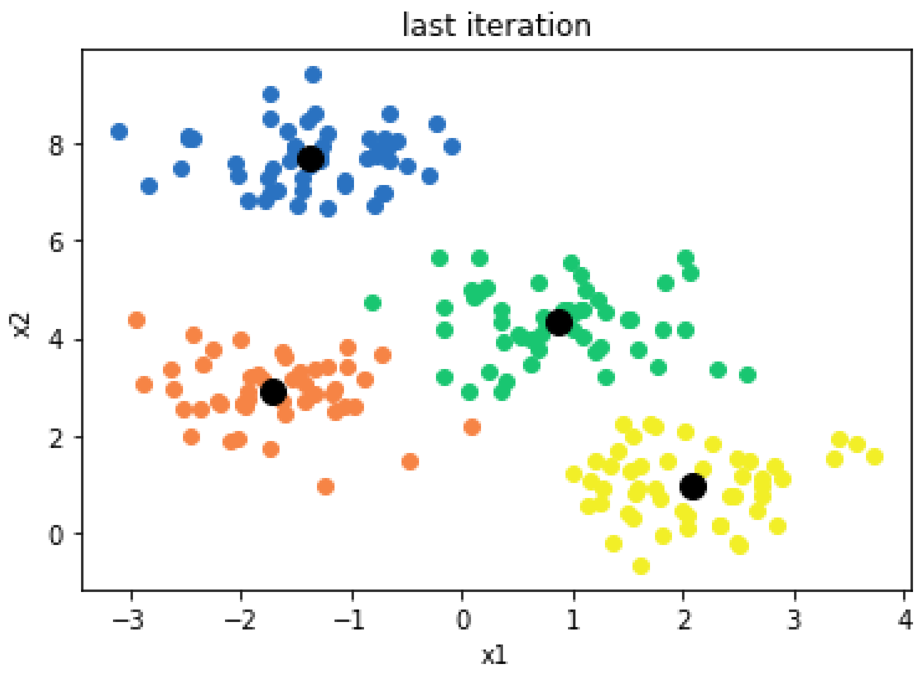
2. Plots for K=4:

Data:

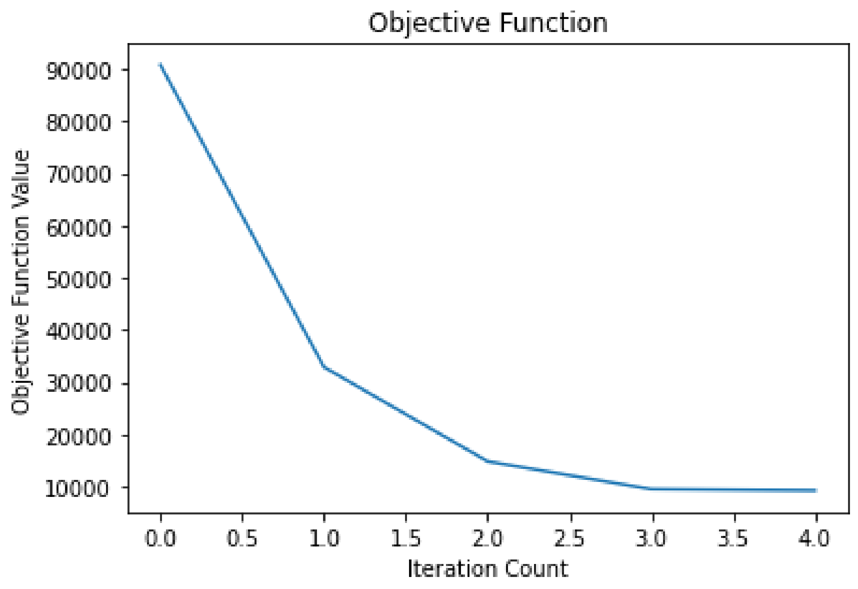




0 iteration
1 iteration
2 iteration
3 iteration
x1
x2

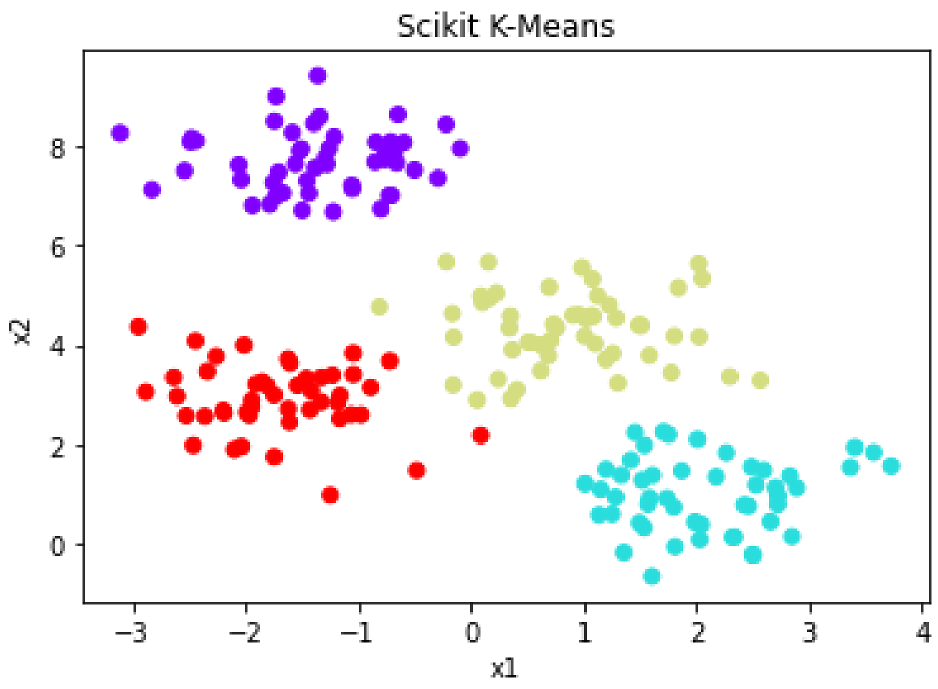


3. Objective Function For K=4:



(Here 0 is the first iteration)

4. Scikit K-Means Plot for K=4:



For a small number of clusters, my k-means algorithm and scikit's algorithm gave almost the same result.

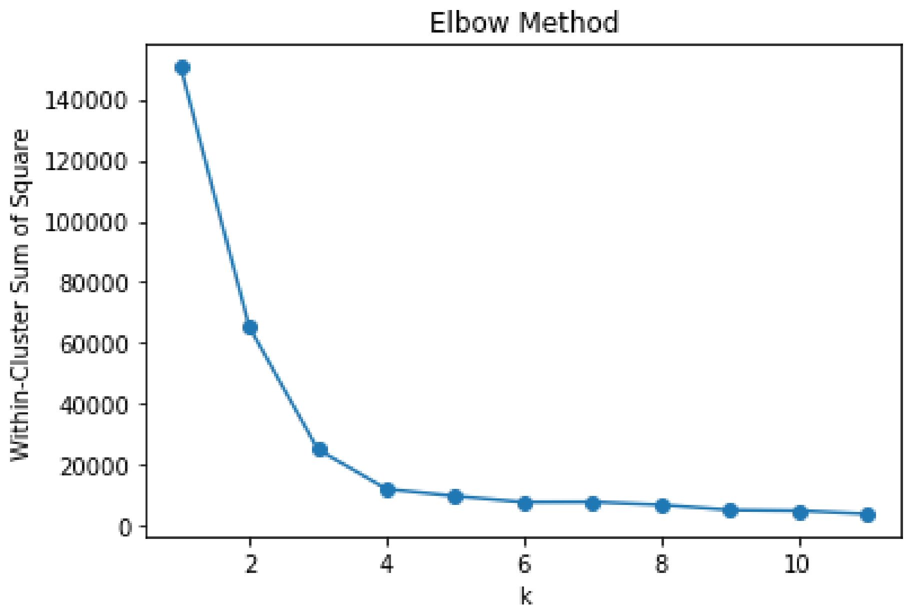
Cluster centers of scikits:

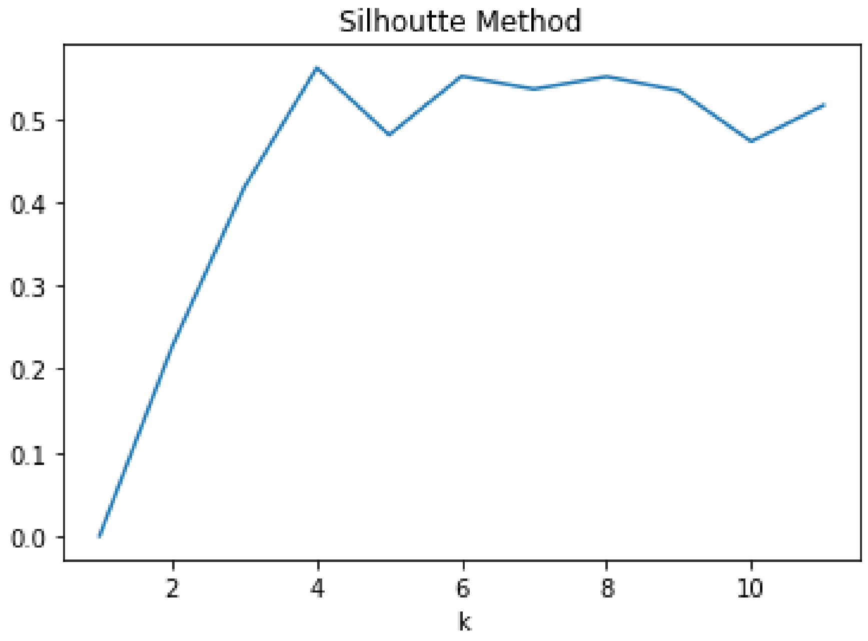
- -1.36512238, 7.70188306
- 2.07464749, 0.9869902
- 0.86008475, 4.31501411
- -1.70639178, 2.9104771

My cluster centers:

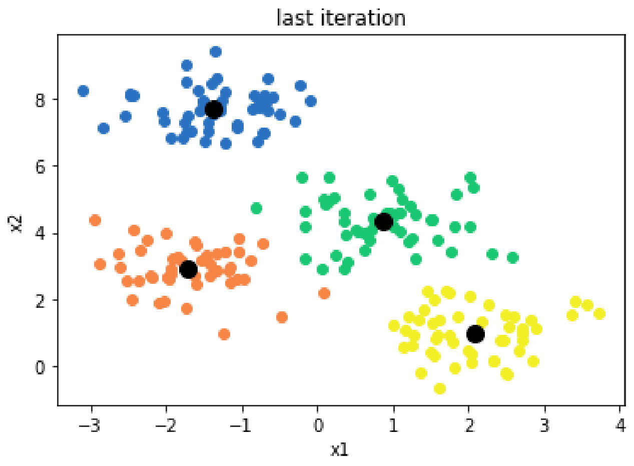
- -1.365122378545809, 7.701883059349756
- 2.0746474903995904, 0.9869901979731351
- 0.860084751977127, 4.315014109140235
- -1.7063917805897937, 2.9104770965279765

As we can see, cluster centers are also the same.

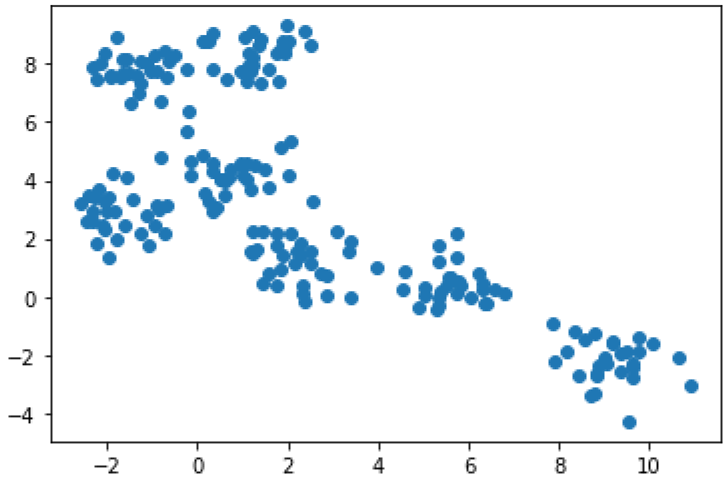


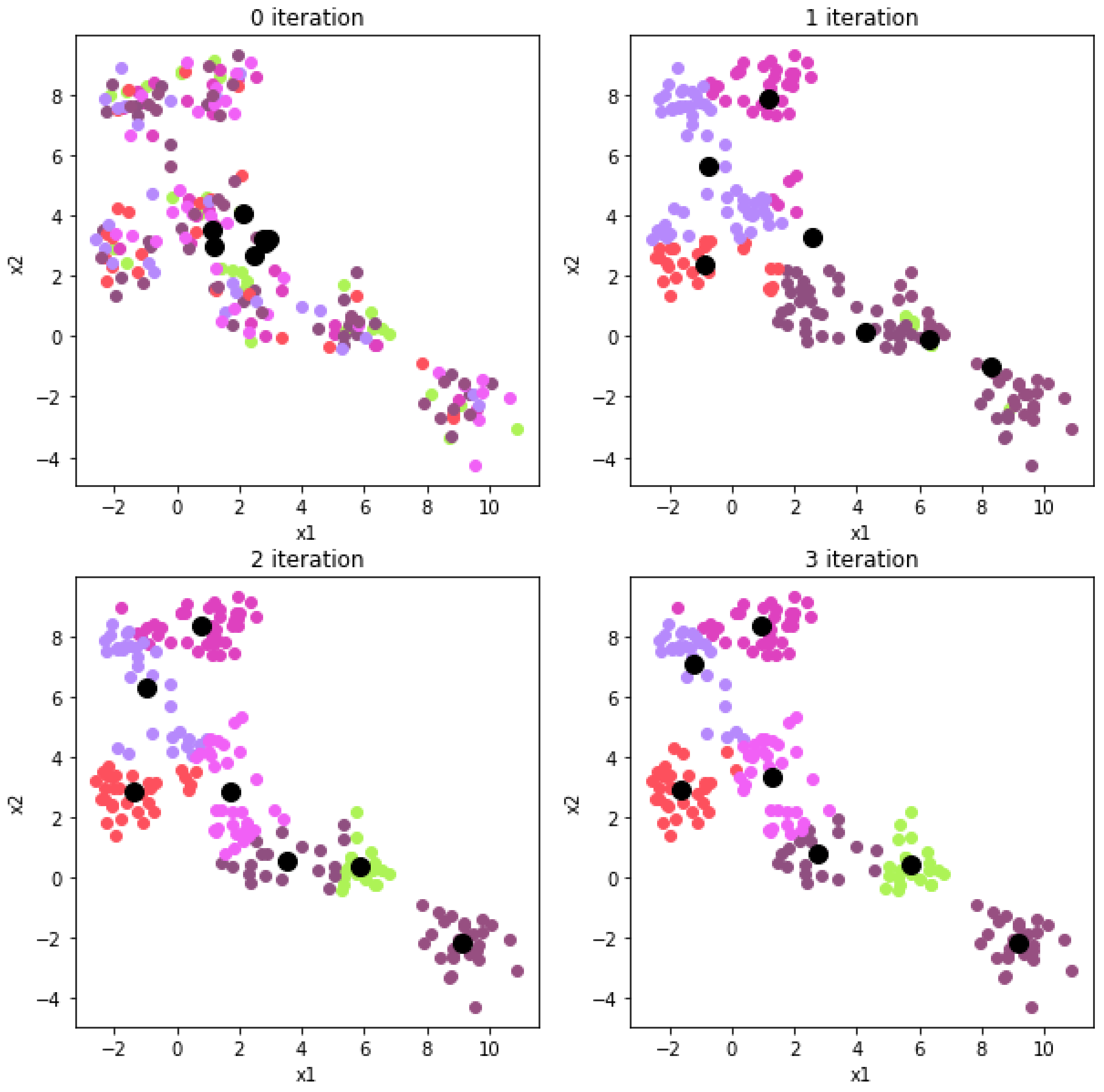


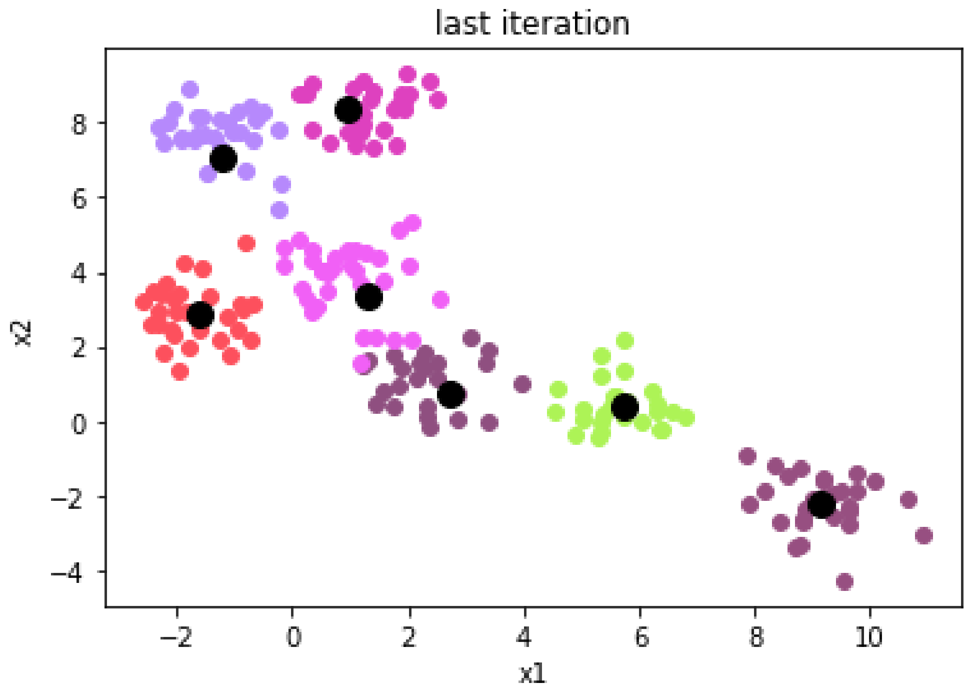
As we can see in the elbow method, k=4 is where WCSS remains the same. Also, in the Silhouette Method, the maximum value is where k=4. So, it is the same as we run the algorithm. We get the same result because our data is generated using make_blobs where we can assign a cluster number. This also shows that our algorithm works perfectly. (See **section 5** for more information about the methods)



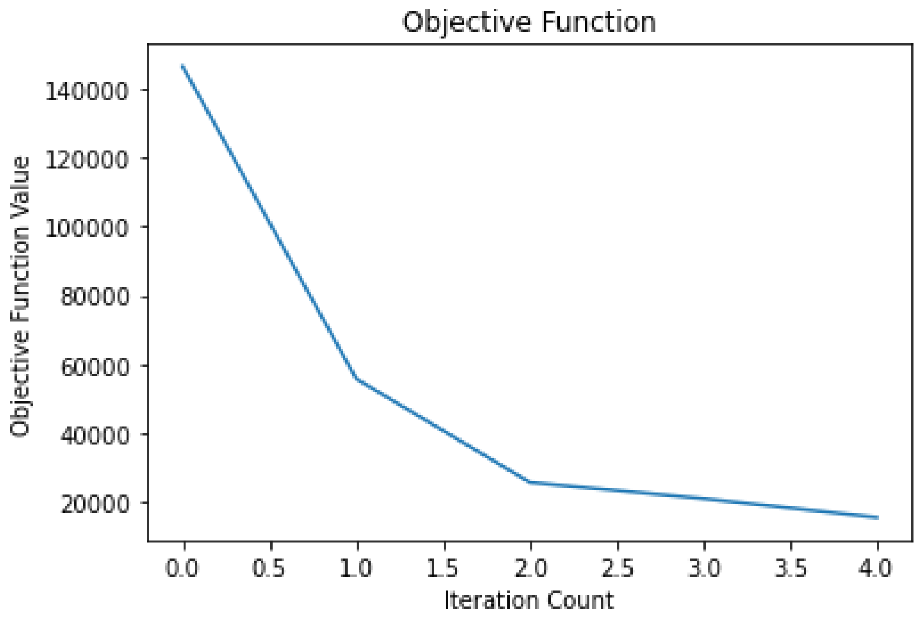
2. Plots for K=7:



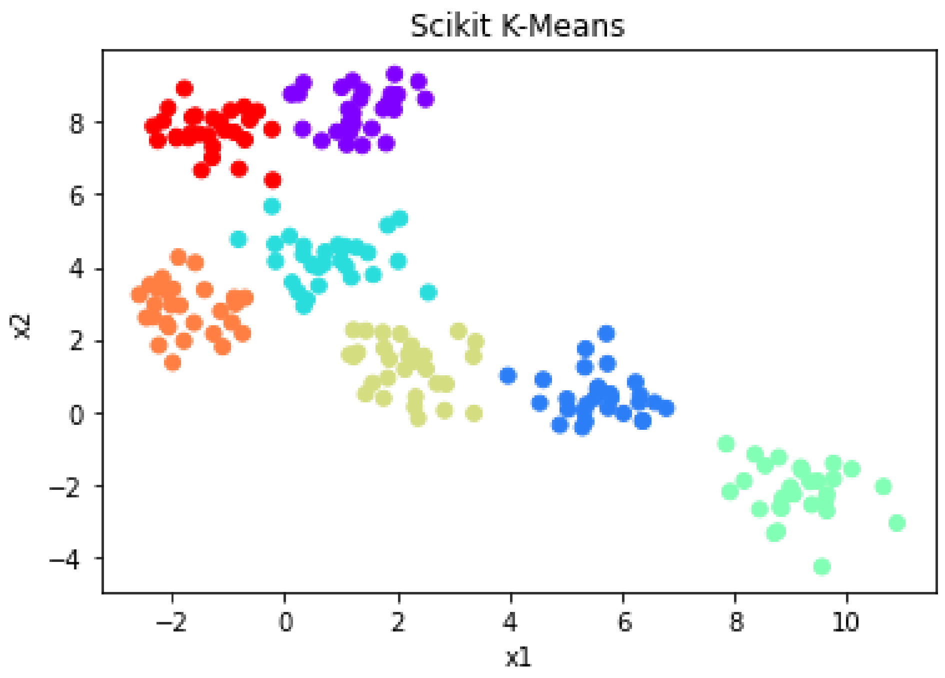

0 iteration
1 iteration
2 iteration
3 iteration
x1
x2



3. Objective Function For K=7:



4. Scikit For K=7:



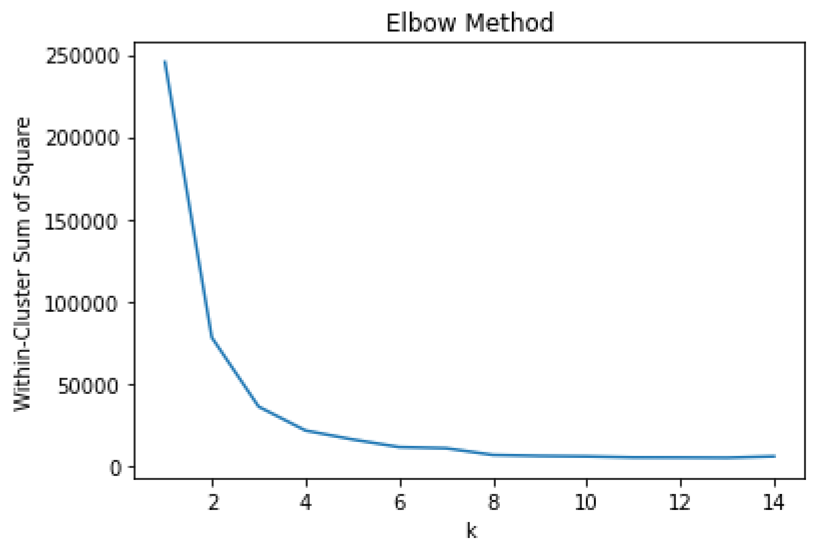
Scikit centroids:

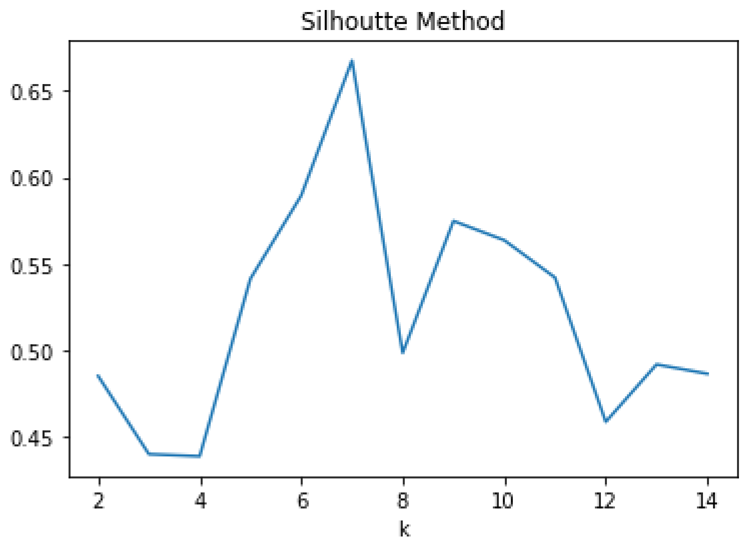
- 1.30719299, 8.36117738
- 5.59384295, 0.43689677
- 0.78330644, 4.20625284
- 9.15080756, -2.19328538
- 2.17645984, 1.23087219
- -1.73119364, 2.83148923
- -1.31391674, 7.73929168

My centroids:

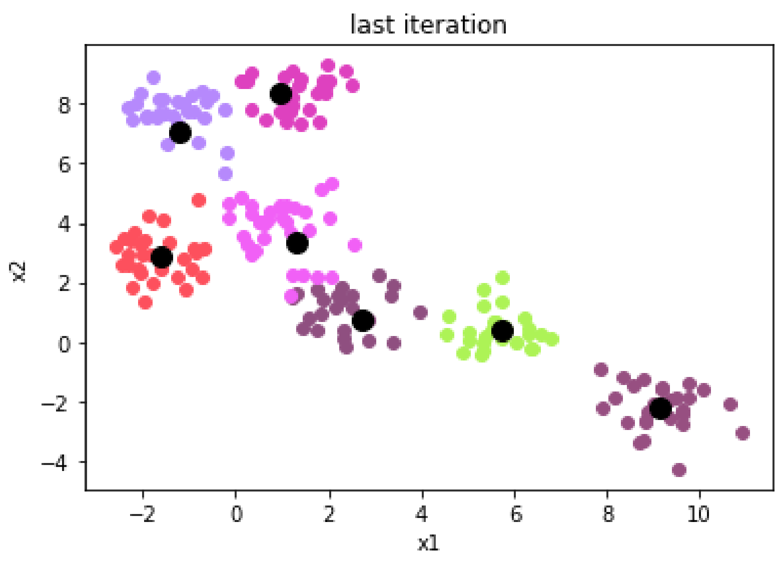
- 0.9479183285848508, 8.350250229950362
- 5.740191720326715, 0.4037567344501057
- 1.3048672814952753, 3.3254498448330194
- 9.15080755897294, -2.193285384836794
- 2.7413099109831416, 0.7708201548630773
- -1.616186103041066, 2.9005195139746522
- -1.2149358232638507, 7.08876251828733

My centroids are almost the same. However, it has some differences. The difference may arise from having different thresholds to stop the algorithm. It also might be running the algorithm several times and getting the best one.





Here in both of my methods, the best K value is the 7, which is the same as we tried before.



5. Finding The Best K Value:

During my research, I've found 2 different algorithms to select the best K-value.

- Silhouette Method
- Elbow Method

The Elbow Method is the most popular method to find the best K. It is also easier to implement. It is based on within-cluster-sum of squared (WCSS) value for different numbers of clusters. The method is called "elbow" because when we plot the WCSS vs K, the plot looks like an elbow. The best K is where the WCSS starts to remain same.

My Algorithm:

For each K value starting from 1 to the given Kmax, calculate the sum of euclidean distances between every point in the same cluster. Then, plot this value vs K.

However, for some cases (data with bigger clusters), the elbow method might give ambiguous results. For this kind of cases, the silhouette method is used. The silhouette method can give better results because it involves more information. We don't calculate distances only in the same clusters but also the distances with other clusters.

Main formula for silhouette is:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}, \text{ if } |C_i| > 1$$

and

$$s(i) = 0, \text{ if } |C_i| = 1$$

For each data point $i \in C_i$, we now define

$$b(i) = \min_{i \neq j} \frac{1}{|C_j|} \sum_{j \in C_j} d(i, j)$$

b(i) is the average euclidean distance between points with different clusters

For each data point $i \in C_i$ (data point i in the cluster C_i), let

$$a(i) = \frac{1}{|C_i| - 1} \sum_{j \in C_i, i \neq j} d(i, j)$$

a(i) is the average euclidean distance between points in the same cluster.

My algorithm calculates b(i) and a(i) for each point in the cluster, then takes the mean. The final value is the silhouette value. When we plot this value vs K, the K with the maximum value is the best to choose.

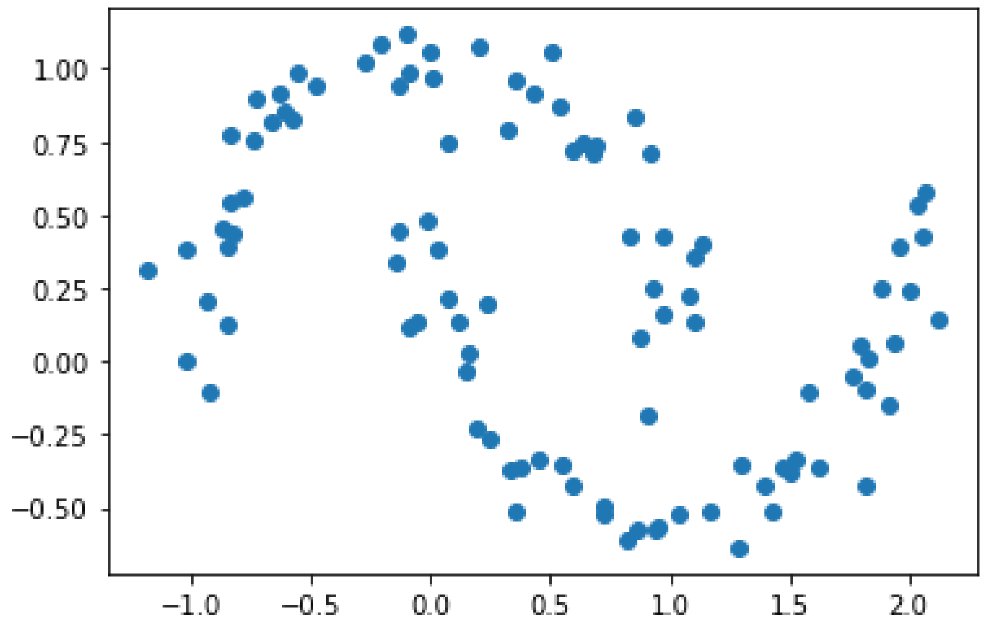
References:

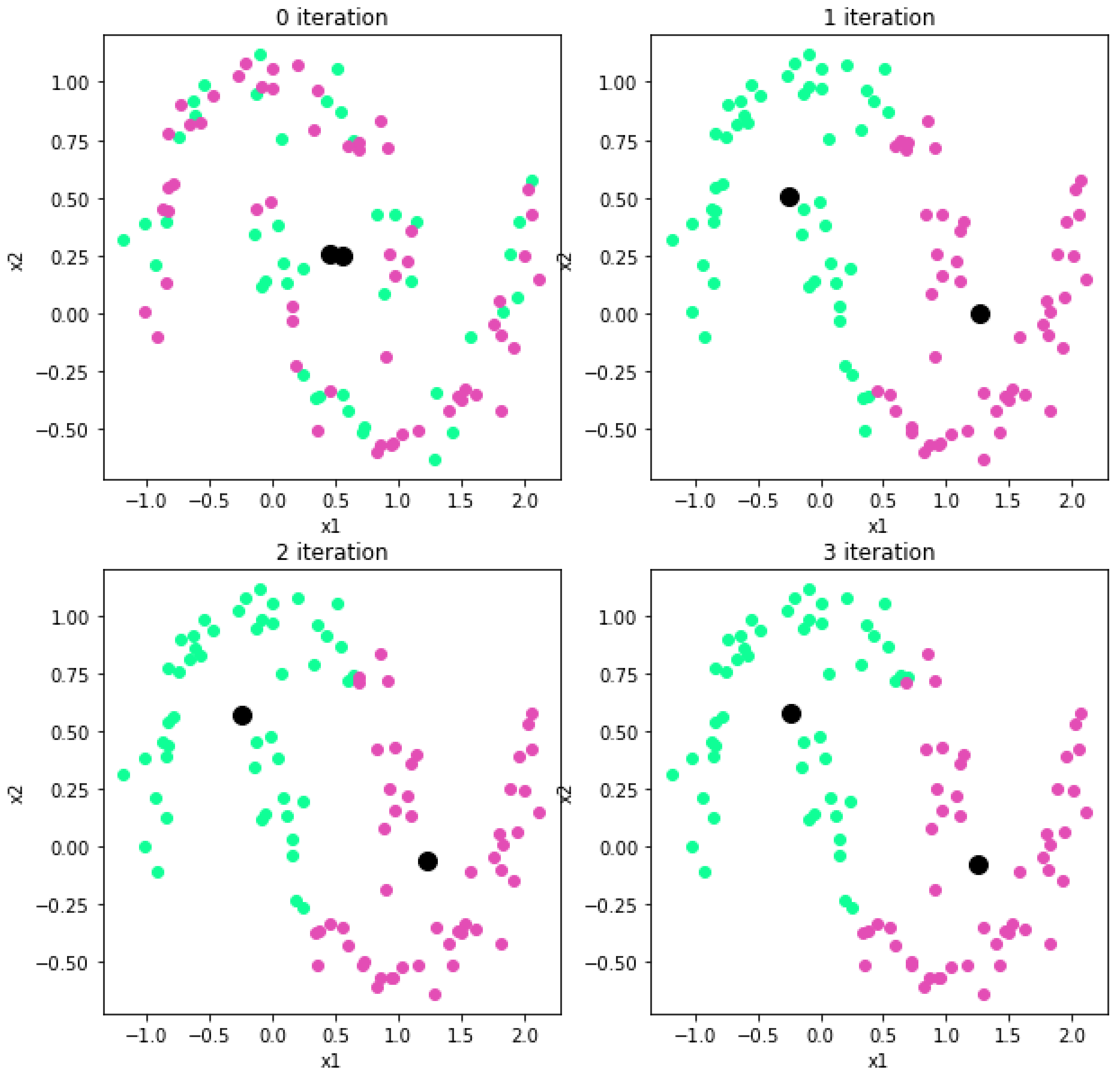
https://medium.com/analytics-vidhya/how-to-determine-the-optimal-k-for-k-means-708505d204eb

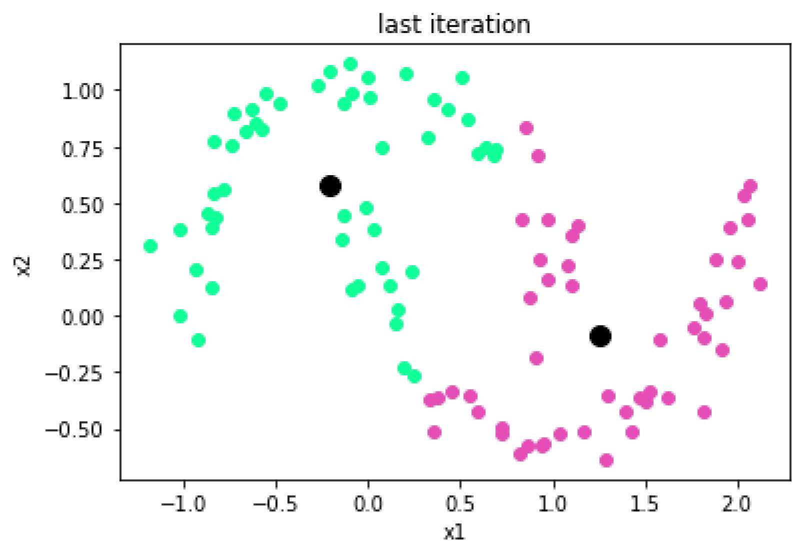
https://scikit-learn.org/stable/modules/clustering.html#k-means

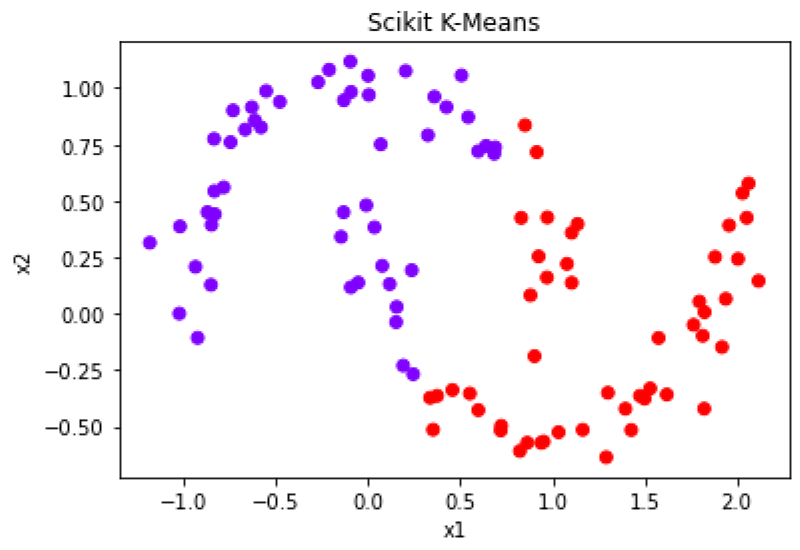
https://towardsdatascience.com/silhouette-method-better-than-elbow-method-to-find-optimal-clusters-378d62ff6891

5. Non-convex Data:

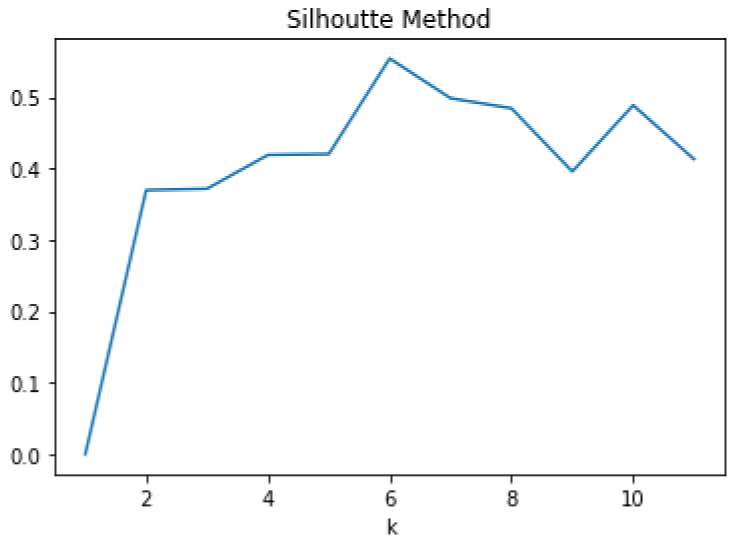


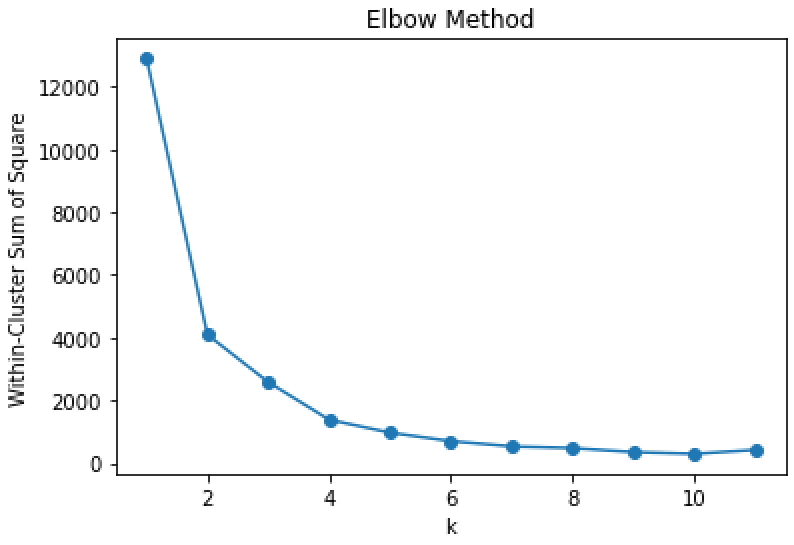

0 iteration
1 iteration
2 iteration
3 iteration
x1
x2
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0





As we can see, our algorithm works in the same way. However, it is still not suitable for non-convex data.


Silhoutte Method
0.0
0.1
0.2
0.3
0.4
0.5
2
4
6
8
10
k


Elbow Method
Within-Cluster Sum of Square
0
2000
4000
6000
8000
10000
12000
2
4
6
8
10
k