# **CMPE481 DATA ANALYSIS & VISUALIZATION**

# **ASSIGNMENT-1 REPORT**

## Data Generation

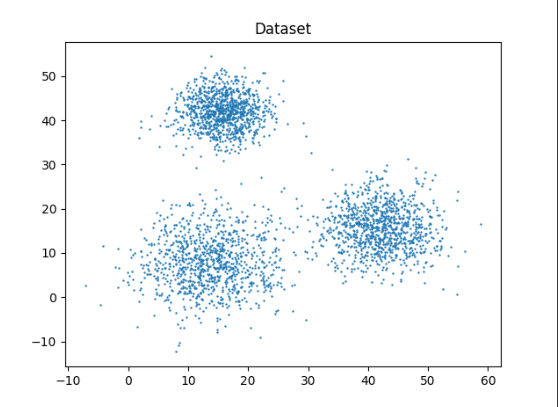
For data generation, I implemented a function called “generateData”. This function generates 2D data suitable for clustering. The number of desired clusters is given to this function as a parameter. For each cluster, two different samples for x and y coordinates (the two data fields) are taken from normal distributions. The means of these distributions are also determined randomly between 0 and k\*15. I decided to make it k\*15 after some trials with different k values since when the k is too big, sometimes different clusters intervene when used a fixed range. This is not something we want. The sigma values for the distributions these samples are coming from are also determined randomly in range (3-8). The sample size is given to this function as a parameter.

The function returns a dataframe with fields “x”, “y” and “c”. “c” is used when data items are assigned to the clusters and holds the cluster of each data item belongs to.

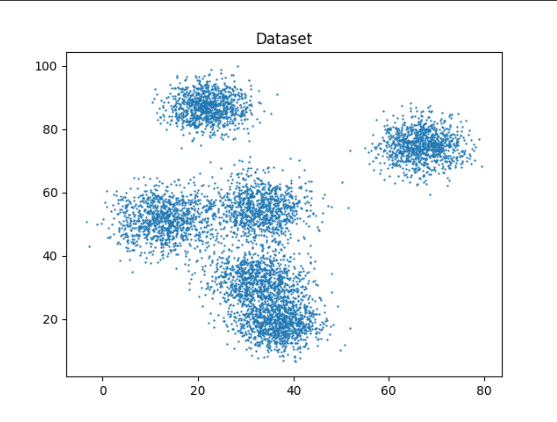
The function terminates when the mean distance between the old and new central points is below 0.01 and the maximum iteration is 15.

Here are some data that is generated with different k values:

K = 3:



K = 6:



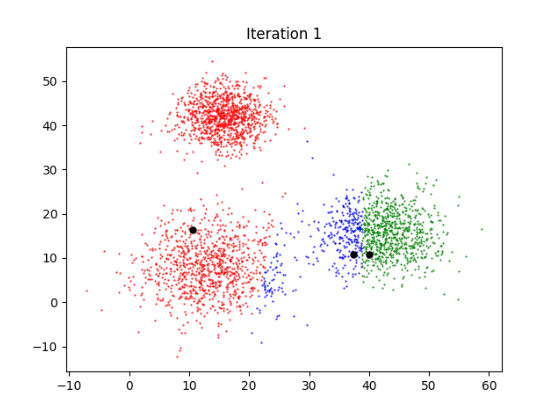
## Implementation of K-Means Algorithm

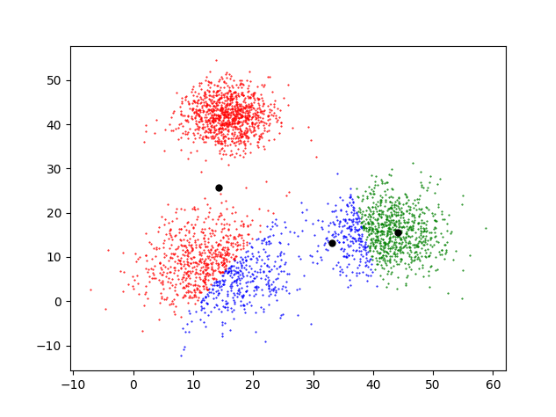
I implemented the K-Means algorithm as it was discussed in the lecture. First, k number of random data points on the generated data are selected as cluster centroids using a function called “getRandomCentroids”. Then, each data point is assigned to the closest centroid. To calculate distance, I wrote a function called “getDistance”. Then, centroids are updated as the mean values of each cluster. The assignment process starts over and this goes on until the changes in the centroid points get small. The “c” field in the dataframe is used to assign data items to clusters as I have mentioned earlier.

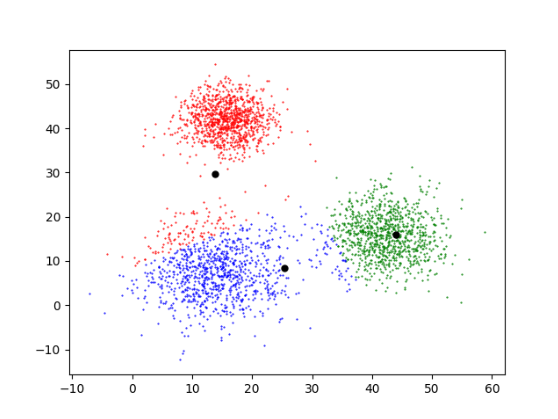
Below, you can see the first three and the last iteration for two different k values:

K = 3:

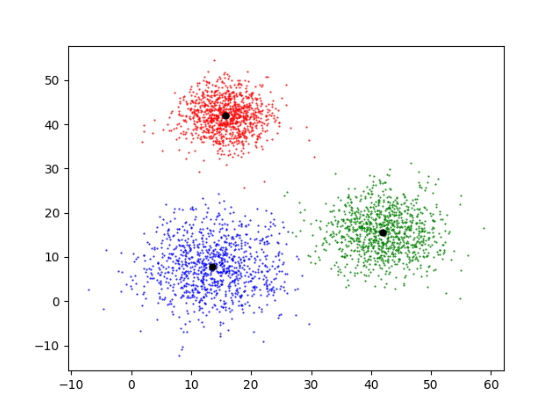
* First three iterations:





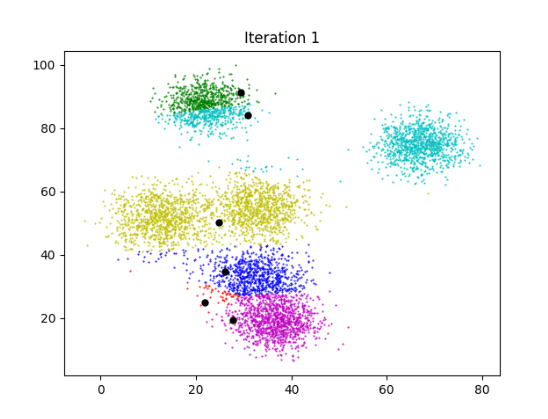


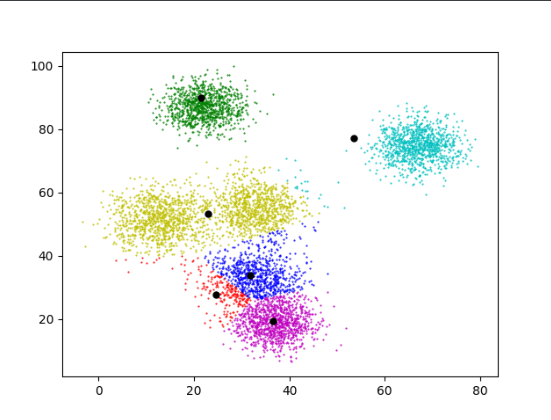
* The last iteration:

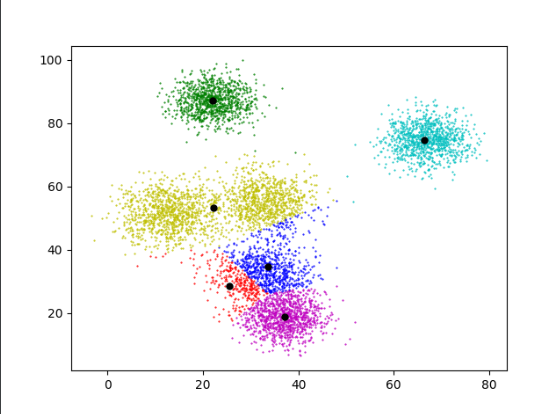


K = 6:

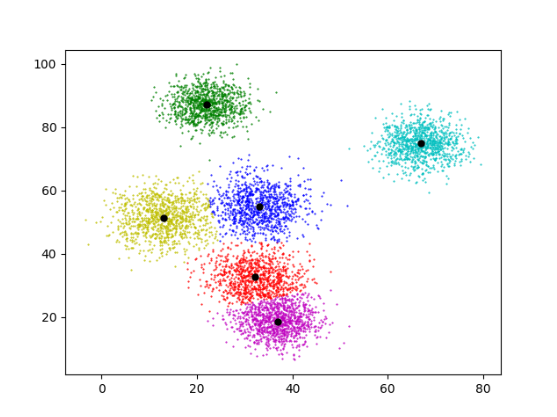
* First three iterations:







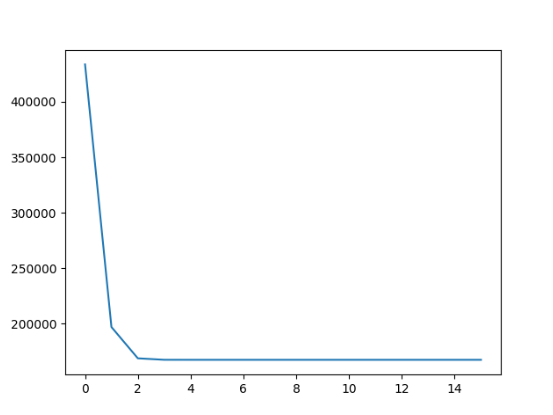
* The last iteration:



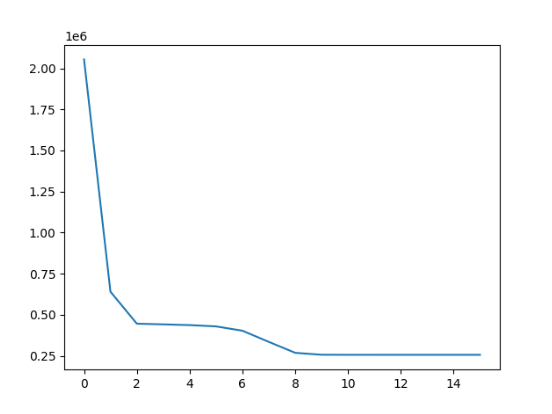
## The Change in the Objective Function

After each iteration in the algorithm, the objective function value is calculated using “getJ” function, and then this value is added to a global list called “Js”. After the “kMeans” function terminates, this list is plotted versus the iteration count. I think these plots look similar to what we would expect.

K = 3:



K = 6:



## Comparison of my Output and the scikit-learn Output

The “kMeans” function I implemented returns the final central points. After this function terminates, I printed the found central points to the console. Then I compared my results with the scikit-learn library’s outputs. Here are the results for two different k’s:

* K = 3:

Found centroids using my method:

[[13.556643682371952, 7.770892320219694], [41.969802294773345, 15.656468539254758], [15.764090228919155, 41.945112188666535]]

Found centroids using sklearn library:

[[15.76409023 41.94511219]

[13.55664368 7.77089232]

[41.96980229 15.65646854]]

* K = 6:

Found centroids using my method:

[[33.132942881787216, 55.0248649717552], [22.072947119531598, 87.04659176271944], [32.171005317586435, 32.632412293136845], [66.90861174592874, 74.77821667442063], [36.85738357010362, 18.605998530517624], [12.92727918632014, 51.385952400478025]]

Found centroids using sklearn library:

[[22.07294712 87.04659176]

[32.17047456 32.62400035]

[66.90861175 74.77821667]

[12.91701411 51.38583595]

[33.12288449 55.02125016]

[36.86227064 18.60030787]]

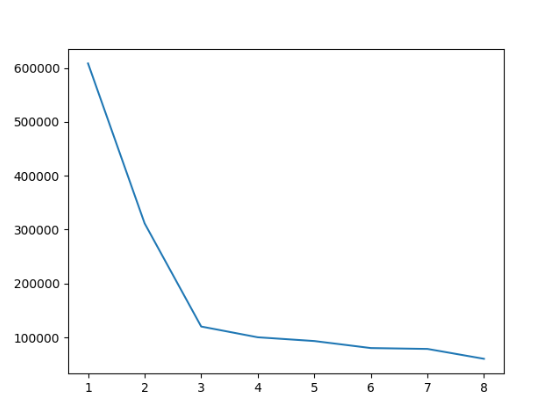
As it is seen, the central points are very close to each other. So, we can say that my implementation also works pretty well.

## Finding the Best k Value

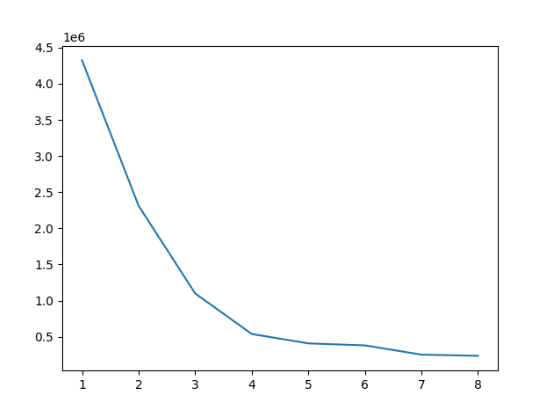
To find the best k value for a given dataset, I made research and found out that there is a method called “Elbow Method” for this. It is easy to implement. For that, I wrote two functions called “elbowMethod” and “calculateWCSS”. These functions call the “kMeans“ function with various k values (in a range 1-8 but it can be changed as desired) and the given dataset. It the WCSS (within-cluster sum of square) is calculated for each call. Then, these values versus the k value are plotted. The plot should be looking like an elbow. We can choose the k as the point where the plot gets almost parallel to the x-axis.

You can see these plots for the datasets that are generated for different k values.

K = 3:



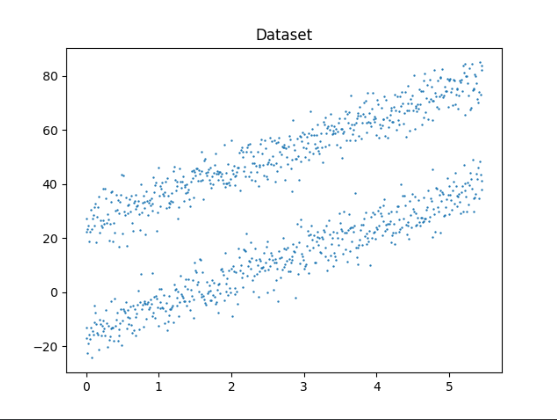
K = 6:



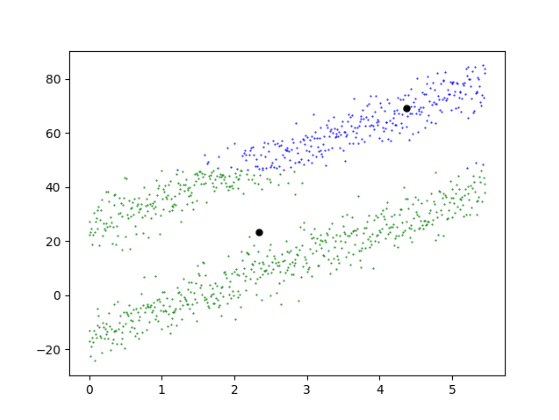
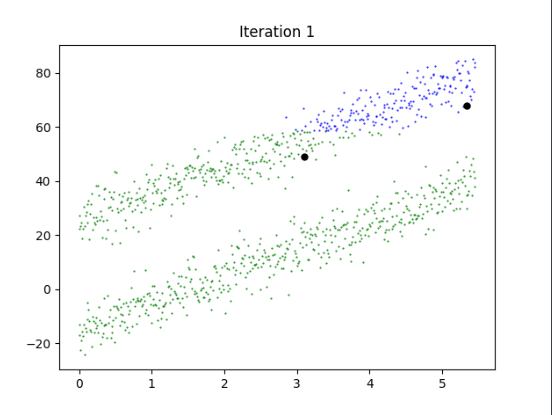
## With a Difficult Dataset

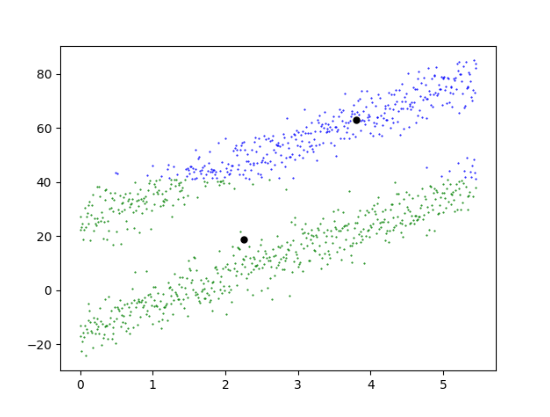
I wrote a function called “generateDifficultData” to generate elongated clusters. I selected k = 2 for this dataset. This function generates two clusters which are above and below a linear function. How much above or below the data points are determined randomly with a normal distribution. As expected, k-means algorithm was not successful at determining these clusters.

* The dataset:

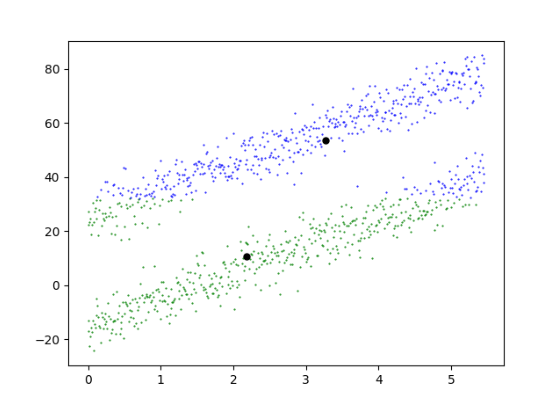


* First three iterations:

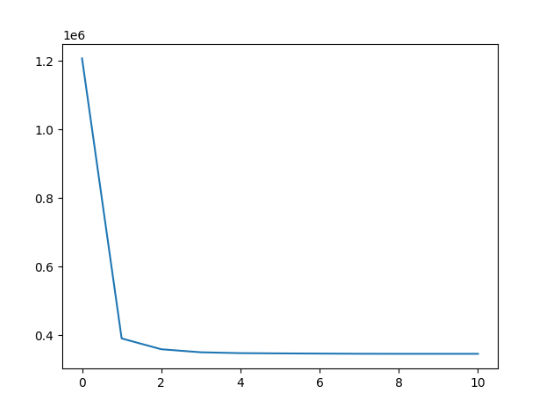




* The last iteration:



* Plotted objective function:



* Outputs of my implementation and sklearn implementation of k-means algorithm:

Found centroids using my method:

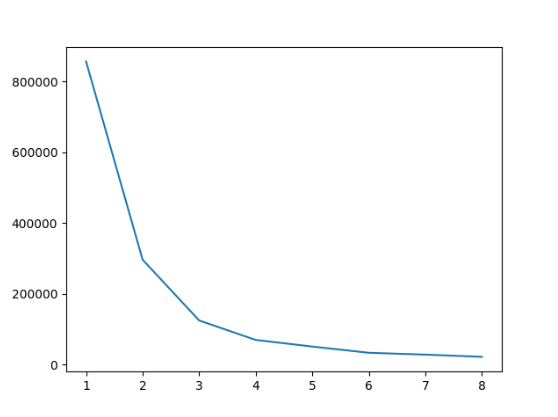
[[2.5251223702055565, 19.868176285120253], [5.821189927611418, 72.3696328660962]]

Found centroids using sklearn library:

[[ 2.6077368 21.28946603]

[ 5.9098501 73.68062482]]

* The elbow method shows:



This clearly do not show that the best k value is 2.

## Additional Notes

To illustrate how the central points change, the clusters are plotted and shown after every iteration in “kMeans” function I implemented. These plotting and showing figure operations slow down the process. Since this function is called 8 times in “elbowMethod” function, I recommend you to comment out the and “plt.plot()”, “plt.show()” lines in the “kMeans” function when running “elbowMethod” to get the result faster.