TUL221 – Assignment

Pan Eyal, Ilana Pervoi

1. The implementation of the K-means algorithm is given at the end of this file, in addition to the *.py* file attached to the submission.

2. In the following exercise, the data samples of *N = 1000, 2D* points are generated from 3 isotropic Gaussians, using *sklearn.datasets#make_blobs.*
   In *figures 1-7*, the algorithms were assigned to find 3 clusters for the data set.

   Comparison of our implementation to the results of *sklearn.cluster#KMeans* :

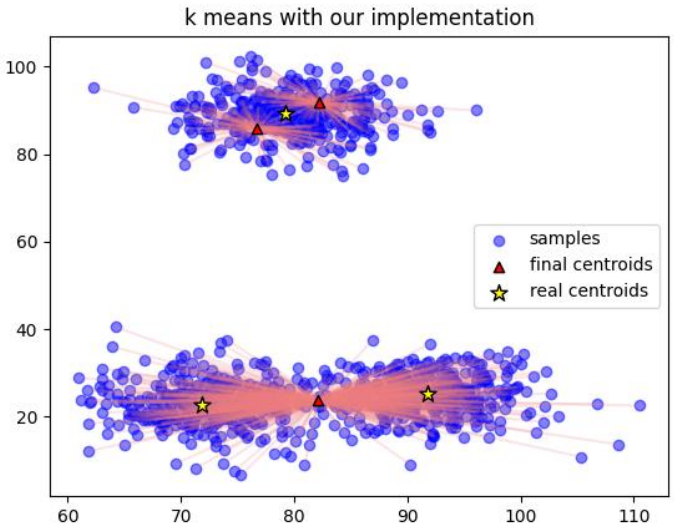


Figure 1

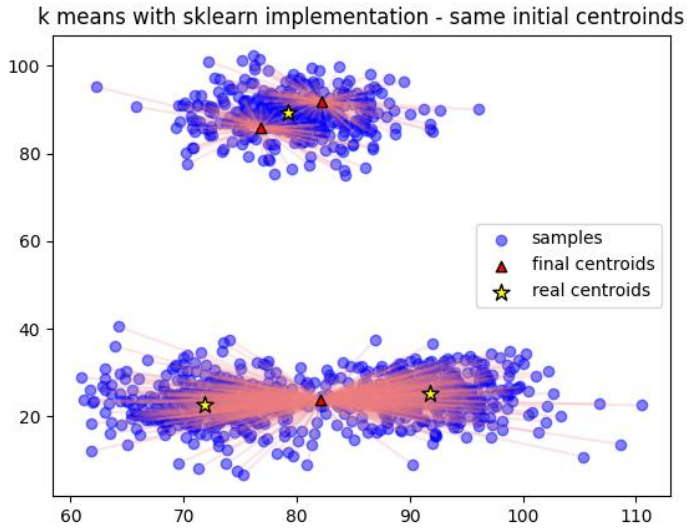


Figure 2

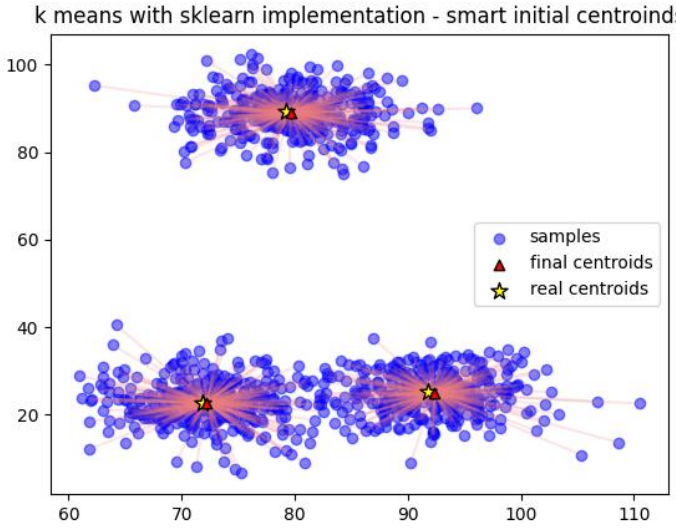


Figure 3

We can observe in *figure 1* and *figure 2* that when given the same random initial centroids, our implementation and $sklearn$ implementation return the same results, which are not the real centroids declared when we generated the data.
In *figure 3* on the other hand, when using smart initialization (*k-means++*), *sklearn* returns centroids which are close to the real centroids.
This observation demonstrates the importance of the first initialization to the accuracy of the results (both in our and in the standard implementation).

<u>Comparison of different random initializations in our implementation:</u>
Let us now compare results of different initializations in our implementation.

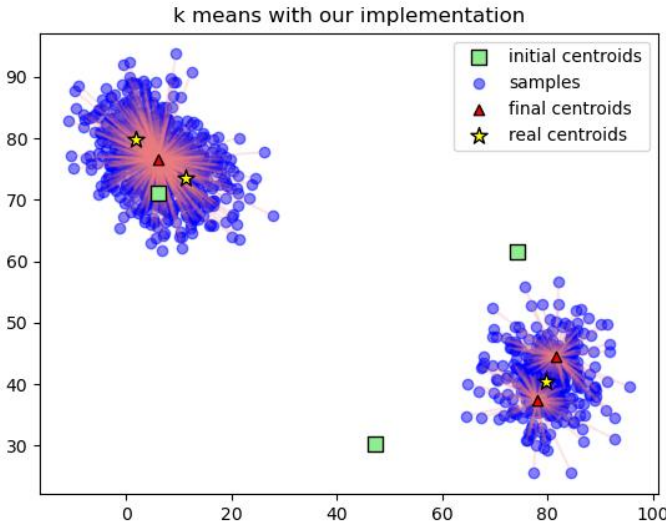


Figure 4

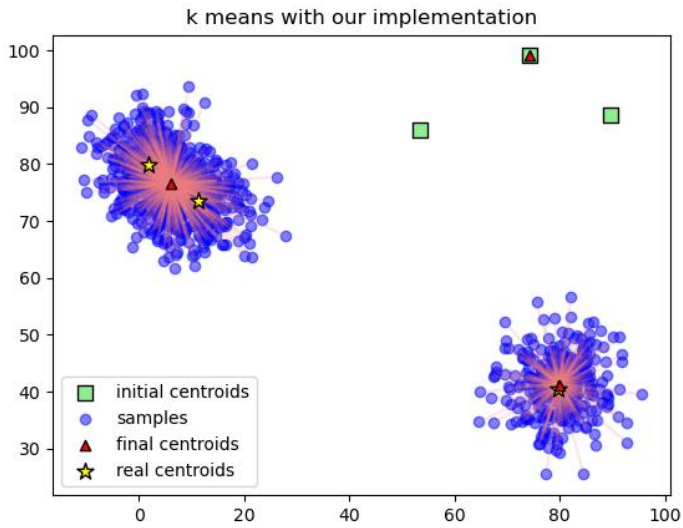


Figure 5

Here we can see the initial random initialization in comparison to the final centroids and the real centroids.

Observe that in both *figure 4* and *figure 5*, the final clustering did not observe that in the left upper corner there are two clusters of data, generated by 2 different Gaussians, since they happened to generate close samples that were assigned to one cluster. If 2 or even 3 initial centroids would have been initialized in that area, the assignment of the samples might have been different and consequently the final clustering.

Interestingly in *figure 5* we can observe that the upper initial guess did not change, and the algorithm determined it to be an empty cluster. The meaning of it is that there wasn't any point that were closer to this centroid in comparison to the other centroids, hence the centroid did not update its place (which is the mean of all the points assigned to it).

Finally, in this demonstration we can observe that in *figure 4* the algorithm divided a set of point from the same generated Gaussian to two clusters since that in the initial centroids there were two centroids in this area, meaning there were samples closer to one centroid that to the other, and vice versa.

Another interesting example that demonstrates the difference of the clustering according to different initialization (we won't analyze this example):

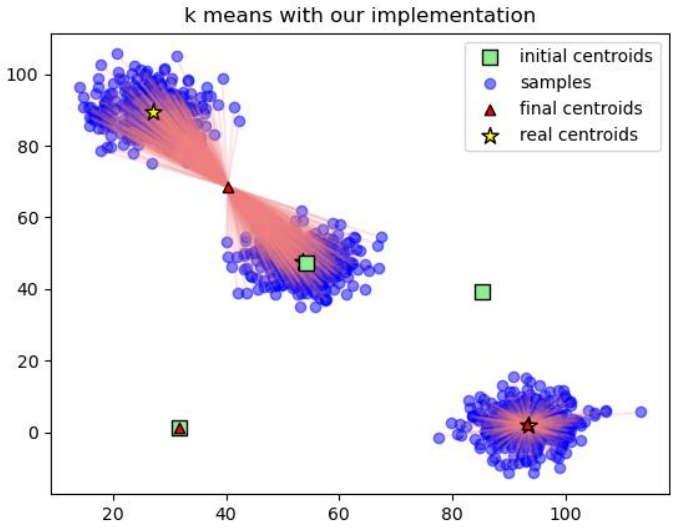


Figure 6

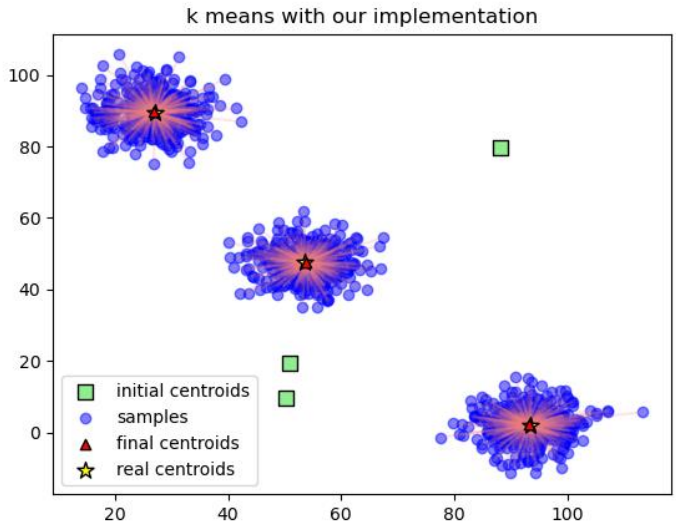


Figure 7

<u>The effect of changing the value of K:</u>

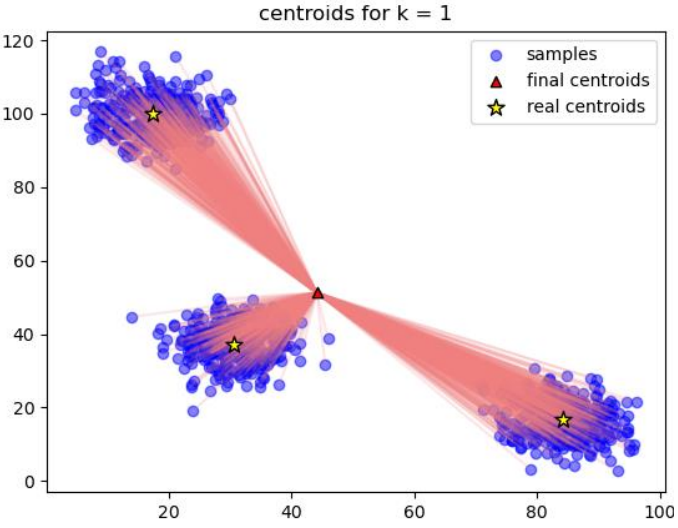


Figure 8

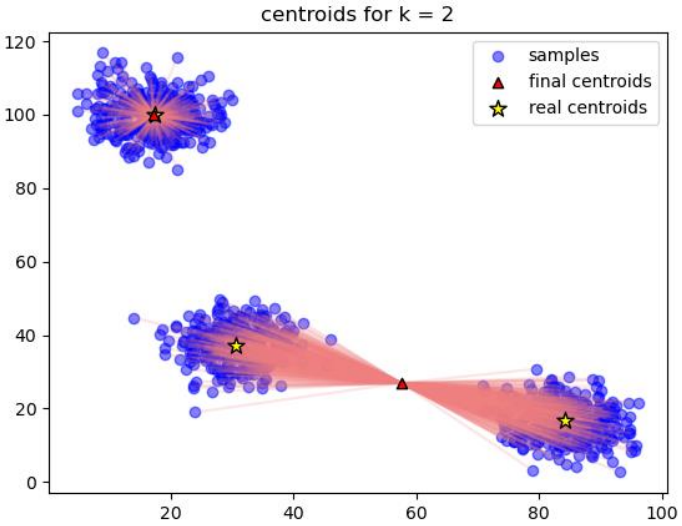


Figure 9

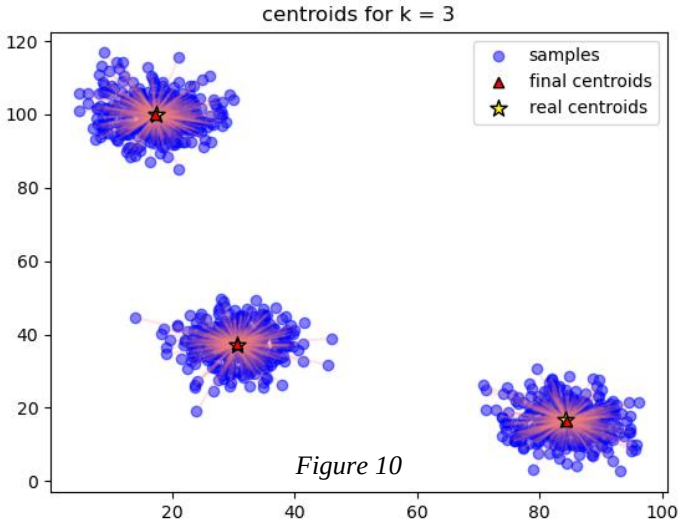


Figure 10

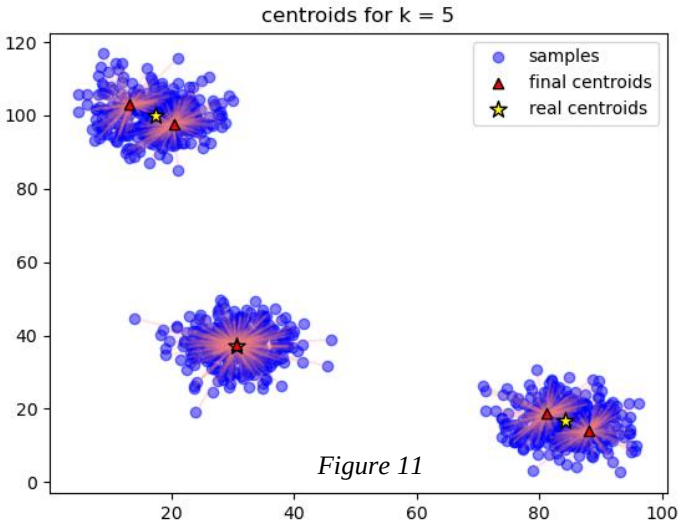


Figure 11

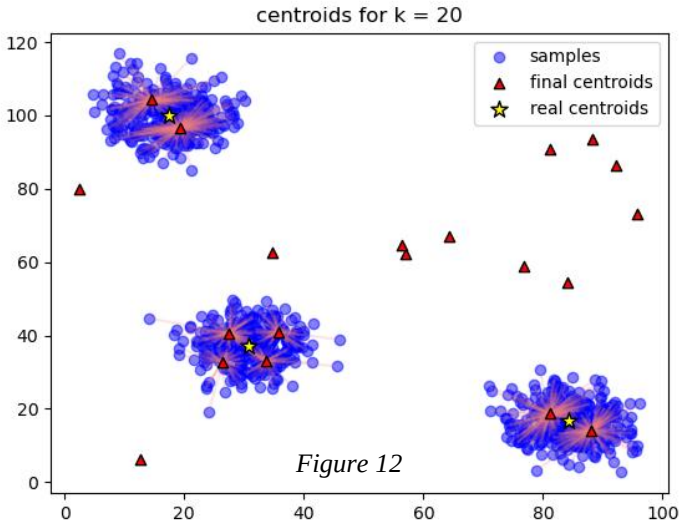


Figure 12

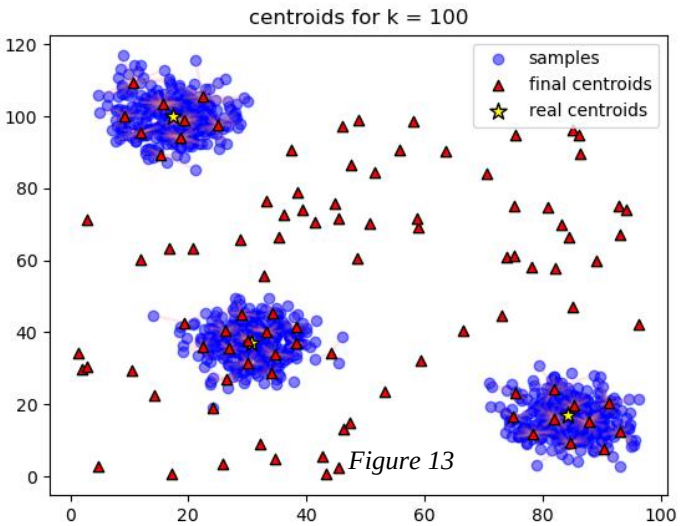


Figure 13

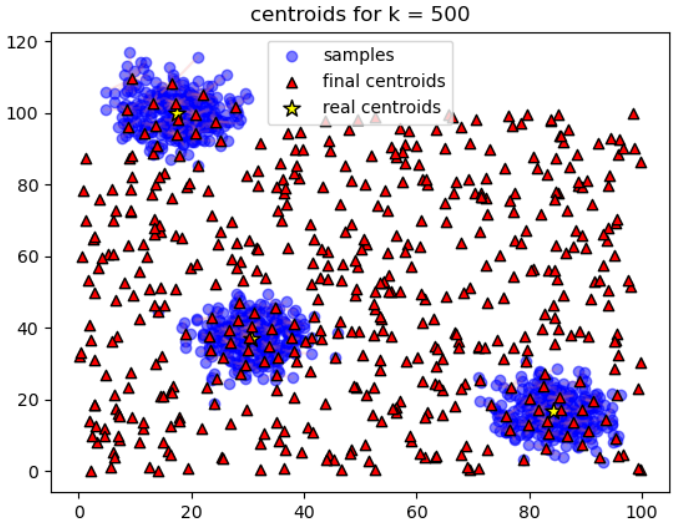


Figure 14

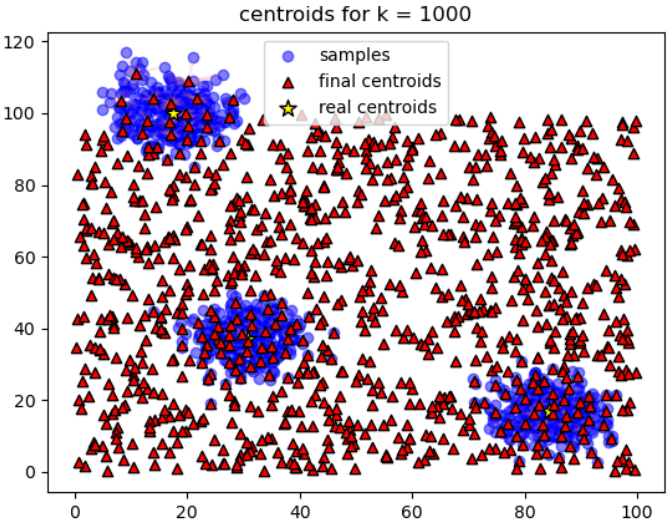


Figure 15

Observe that in *figure 8*, when *k=1* the algorithm updates the centroid to be the mean of all the samples, since they are all connected to it (there had no other choices).

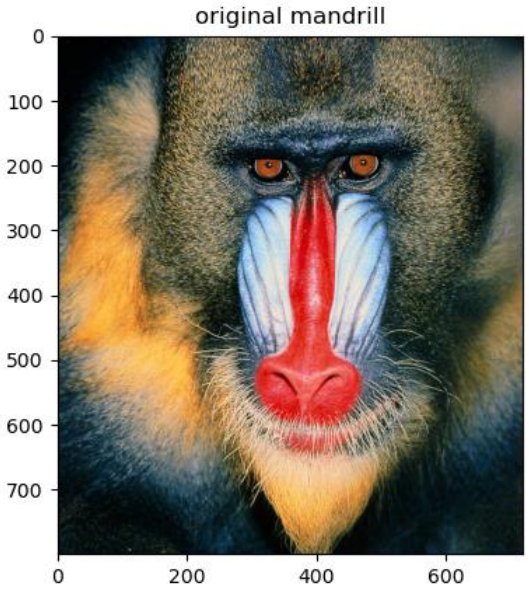
Figure 9 displays clustering to two clusters, when there are actually three. The random initialization resulted a state that the algorithm managed to find one accurate centroid, and the second is updated to be the center of the two remaining data blobs.

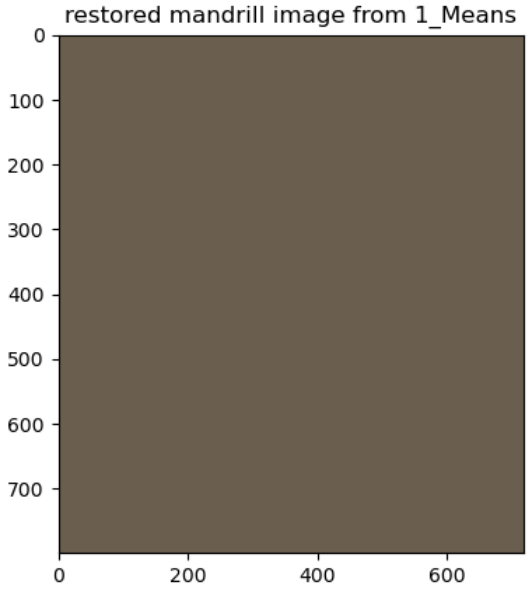
Figure 10 displays a perfect clustering of 3 blobs to 3 centroids as it supposed to.

Figures 11-13 demonstrates situations when the algorithm is assigned to find more clusters than there actually are, hence the result has empty clusters – centroids that no point were assigned to, therefore the centroid remained as it were initialized.
In addition, the centroids that had points assigned to it were "drawn" to the center of their cluster, as they are updated to be the mean of it, results in a blank "ring" around the perimeter of the generated blobs (seen nicely in *Figures 12-13).*

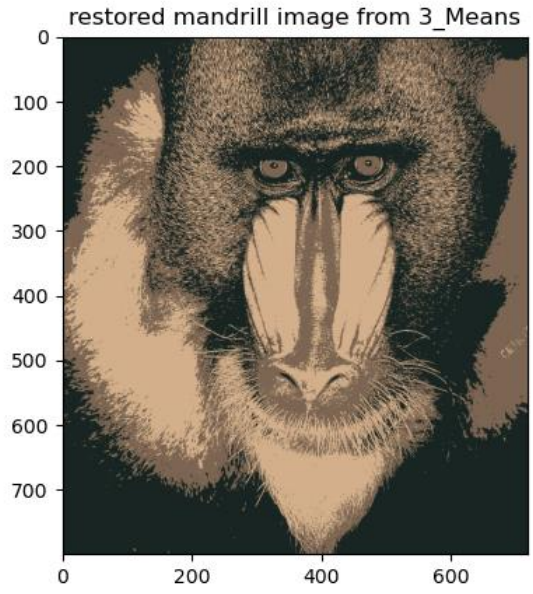
Figures 14-15 demonstrates situation when the algorithm is assigned to find a lot more clusters than there actually are. We can see from our figures that the centroids have little "space" between them, and we can infer that most initial centroids didn't move a lot (From iteration one, every sample already has a close centroid to it). Therefor, our classification would not have any smart meaningful insight to them.

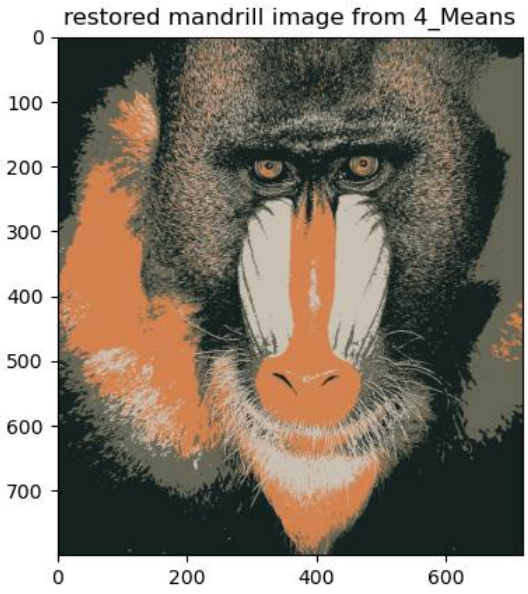
3. We ran our implementation of K means with random values of RGB-centroids for each iteration, those are our results:

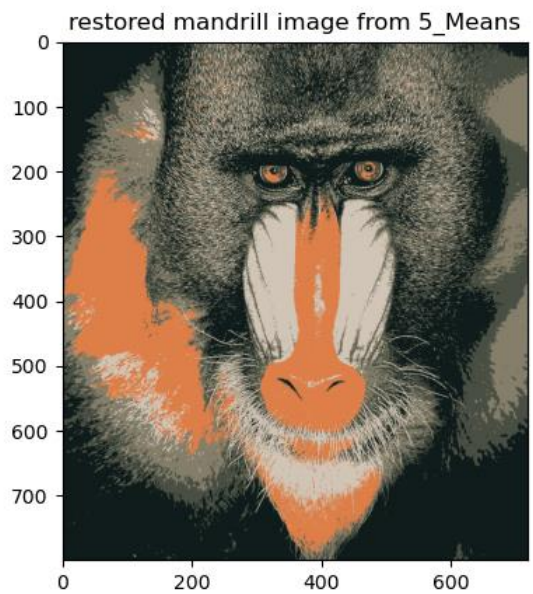


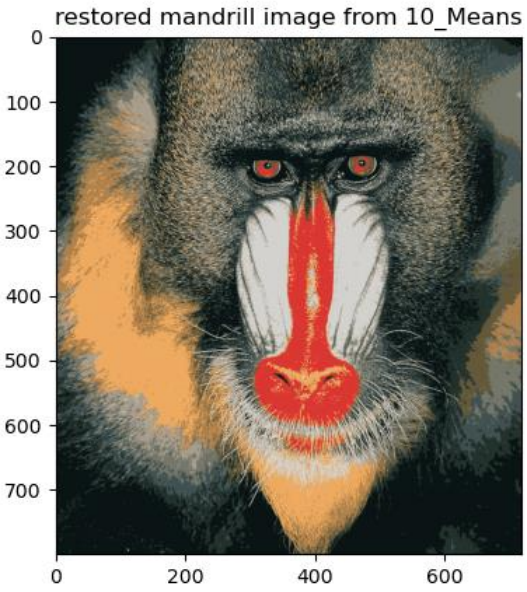


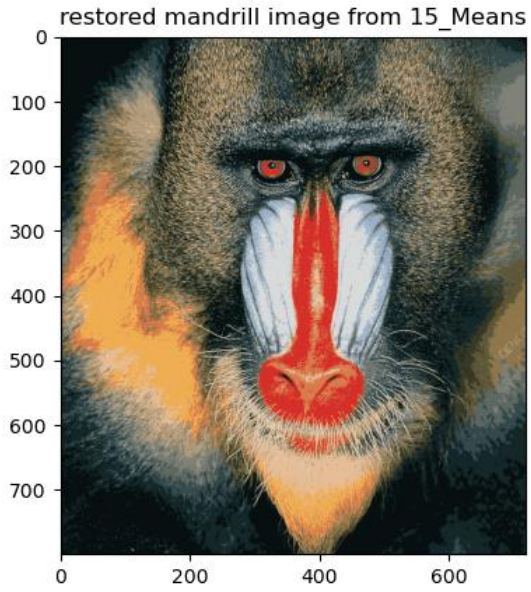

restored mandrill image from 2_Means
0
100
200
300
400
500
600
700
0
200
400
600


restored mandrill image from 3_Means
0
100
200
300
400
500
600
700
0
200
400
600


restored mandrill image from 4_Means
0
100
200
300
400
500
600
700
0
200
400
600


restored mandrill image from 5_Means
0
100
200
300
400
500
600
700
0
200
400
600


restored mandrill image from 10_Means
0
100
200
300
400
500
600
700
0
200
400
600


restored mandrill image from 15_Means
0
100
200
300
400
500
600
700
0
200
400
600

```
class K_means:
    def __init__(self, initial_centroids: np.ndarray, samples: np.ndarray):
        self.k = initial_centroids.shape[0]
        self.samples = samples
        self.num_of_samples = self.samples.shape[0]
        self.centroids = initial_centroids
        self.clusters = []
        self.should_terminate = False

    def update_clusters(self):
        distances = np.zeros((self.num_of_samples, self.k))
        for i in range(self.k):
            # find the distance of each vector to each centroid
            distances[:, i] = np.linalg.norm(self.samples - self.centroids[i], axis=1)
        # find the index of the closest centroid cluster per sample of shape (num_of_samples, 1)
        self.clusters = np.argmin(distances, axis=1)

    def update_centroids(self):
        new_centroids = np.zeros(self.centroids.shape)
        for i in range(self.k):
            cluster_i = self.samples[self.clusters == i]
            if len(cluster_i) == 0 :
                new_centroids[i] = self.centroids[i]
            else:
                new_centroids[i] = np.mean(cluster_i, axis=0)
        error = np.linalg.norm(new_centroids - self.centroids)
        self.should_terminate = error == 0
        self.centroids = new_centroids

    def run(self):
        count = 0
        # start iterating
        while not self.should_terminate:
            count += 1
            self.update_clusters()
            self.update_centroids()
        return self.clusters, self.centroids
```