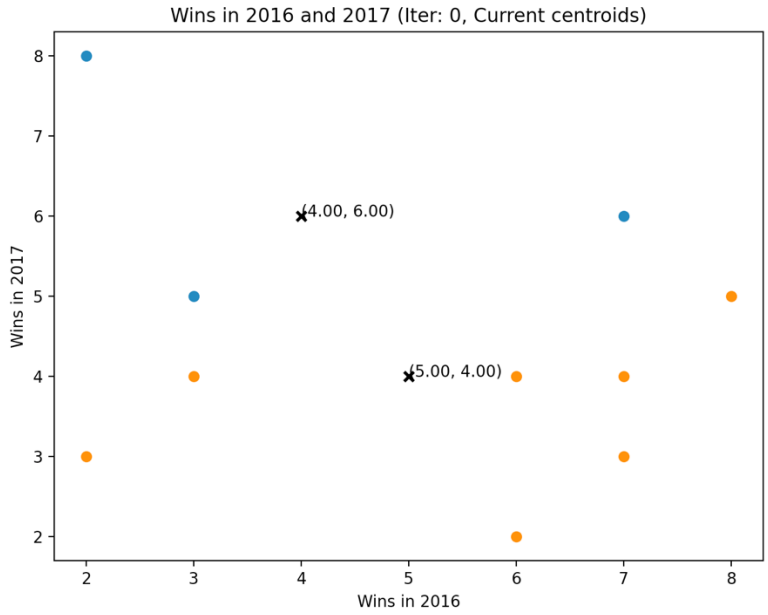
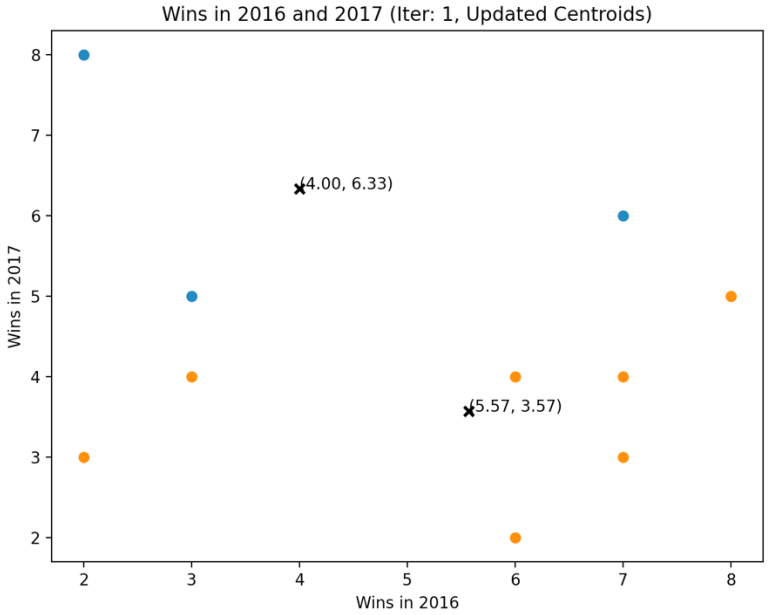
TASK 1:

1) First, the centroids are initialized to (4, 6) and (5,4). The scatter plot looks as follows:

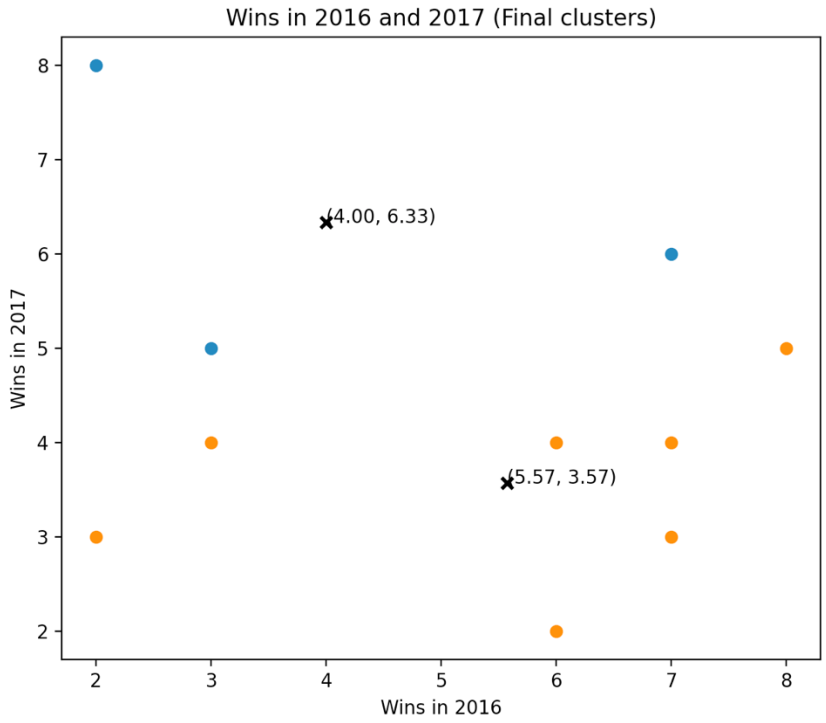


Next, using Manhattan distance for 1 iteration, the center is check:



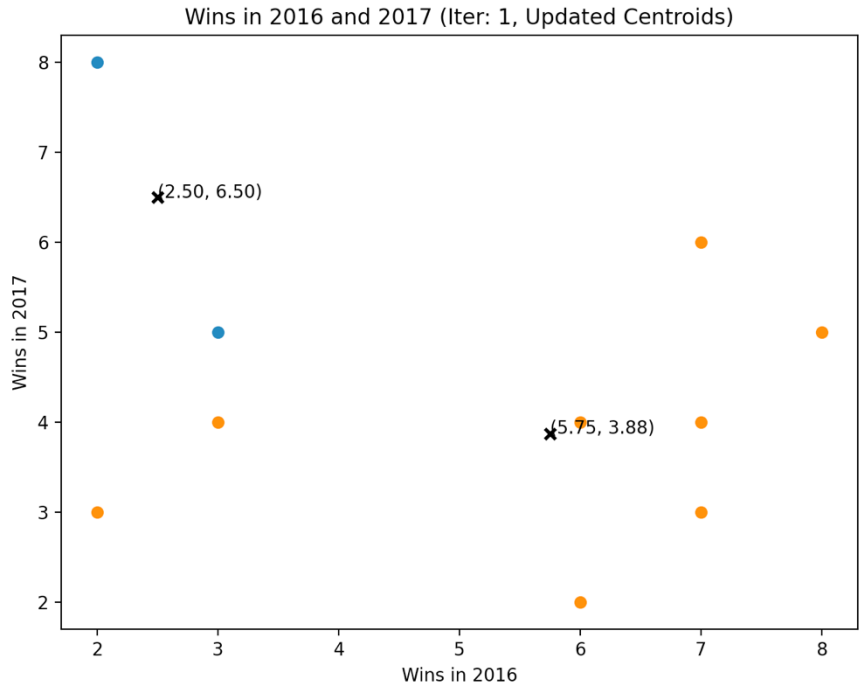
After 1 iteration, the distance of the centroids has slightly changed to (4, 6.33) and (5.77, 3.57).

Iterating until convergence, we get the final clusters as shown below. Convergence is proved on the second iteration.

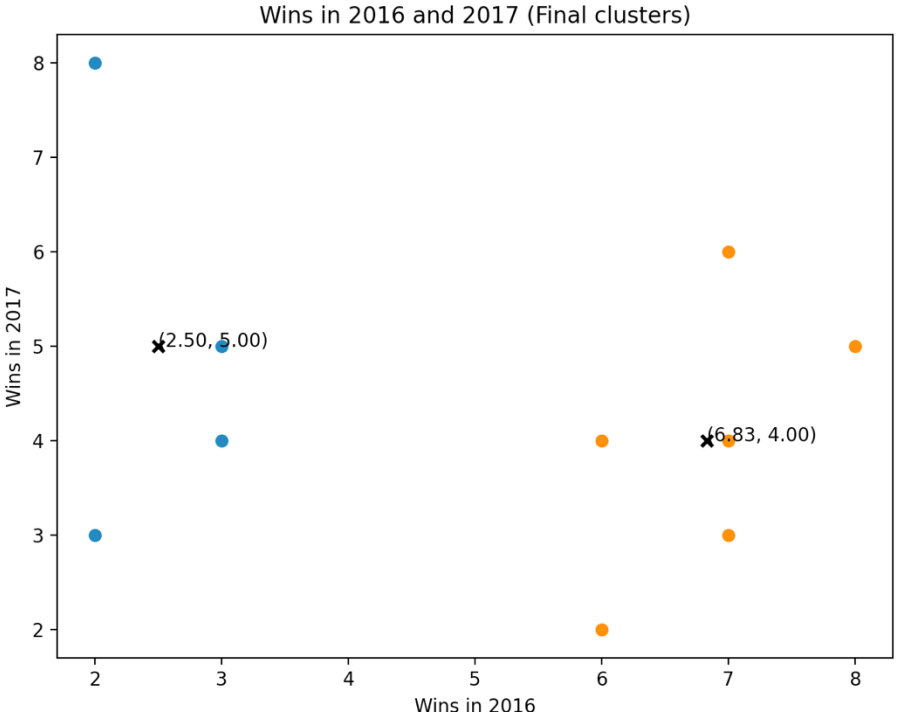



```
Initial Centroids:  (array([4, 6]), array([5, 4]))
The final converging centroids are:  [[4.         6.33333333]
 [5.57142857 3.57142857]]
```

2) Performing the same operations on the same centroids (4, 6) and (5, 4) with Euclidean distance, we get the following output after the first iteration. It can be seen that the centroids of the two clusters are now at (2.50, 6.50) and (5.75, 3.88).



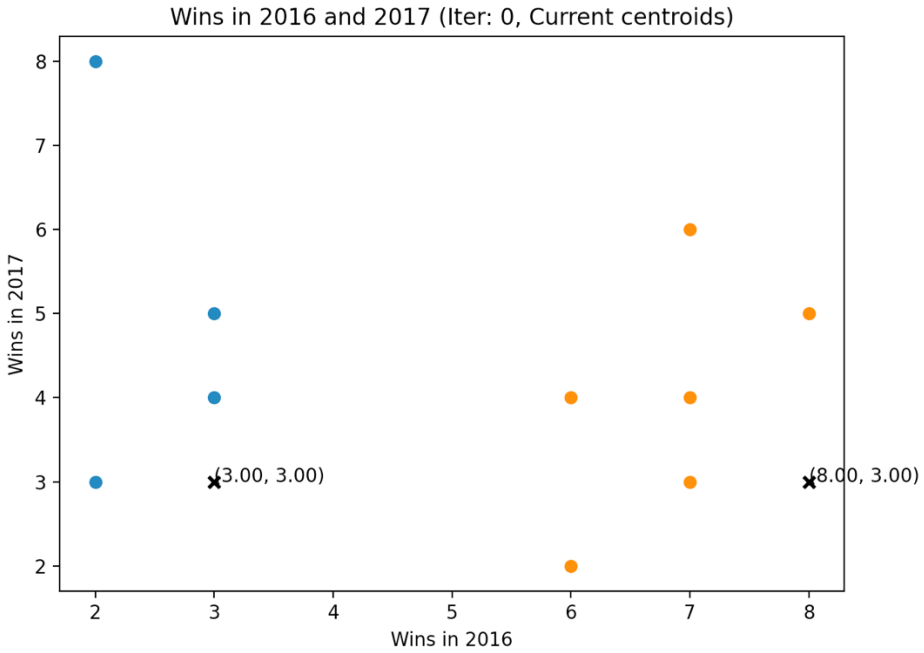
Upon convergence, the centroids are now present at (2.5, 5) and (6.83, 4). The final two clusters can be seen in the figure below. Convergence is proven on the third iteration.



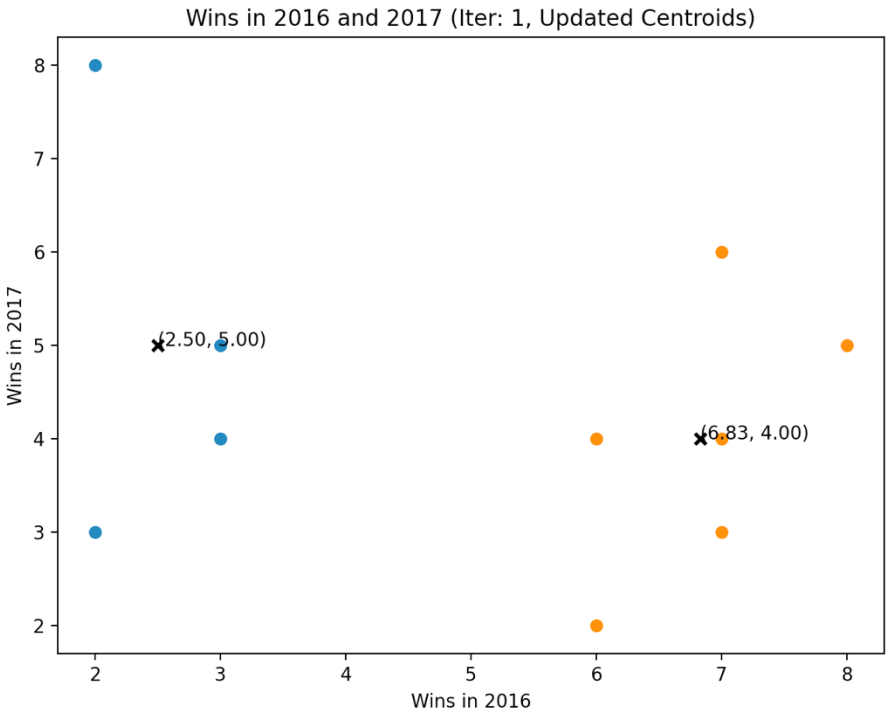
The screenshot of the final converging centroids is shown below:

```
Initial Centroids:  (array([4, 6]), array([5, 4]))
The final converging centroids are:  [[2.5        5.        ]
 [6.83333333 4.        ]]
```

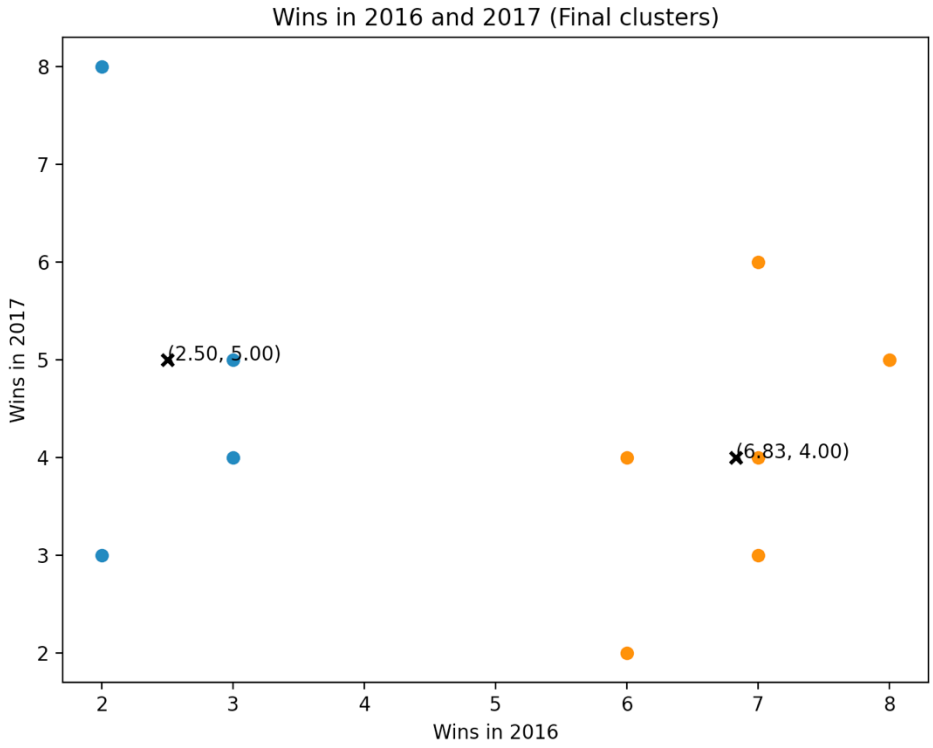
3) The centers are now changed to (3, 3) and (8, 3). The below graph shows the initial centroids.



After 1 iteration with Manhattan distance, we see that the centers are now at (2.5, 5) and (6.38, 4).



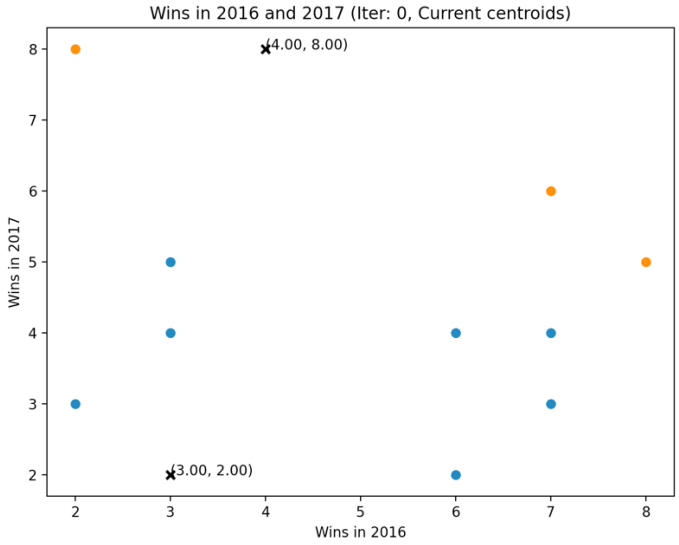
Upon executing until convergence, it is seen that the centroids don't change any further. Convergence is proved on the second iteration. The final clusters are shown below:



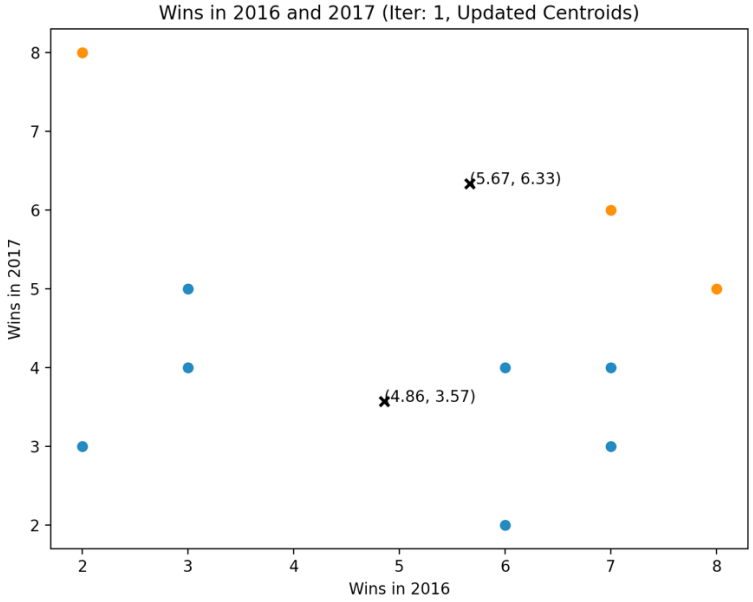
Output screenshot showing the centroid points:

```
Initial Centroids:  (array([3, 3]), array([8, 3]))
The final converging centroids are:  [[2.5        5.        ]
 [6.83333333 4.        ]]
```

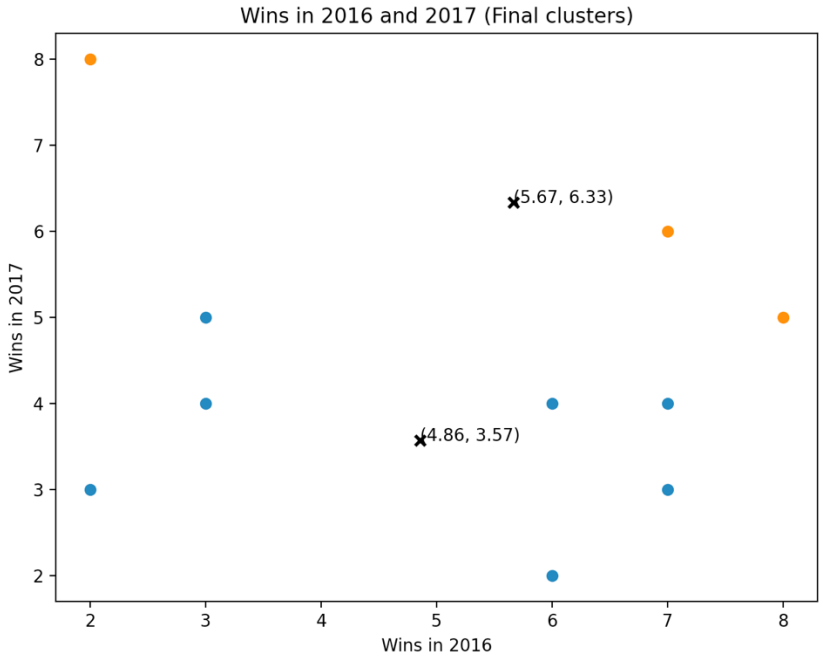
4) Considering the centroids at (4, 8) and (3, 2), the graphs looks as below:



After 1 iteration with Manhattan distance, the output looks as follows. The centroids are at (5.67, 6.33) and (4.86, 3.57).



Upon executing until convergence, it is noted that the convergence is proved on the second iteration. The final graphs is as follows:



The screenshot of the converging centroids is shown below:

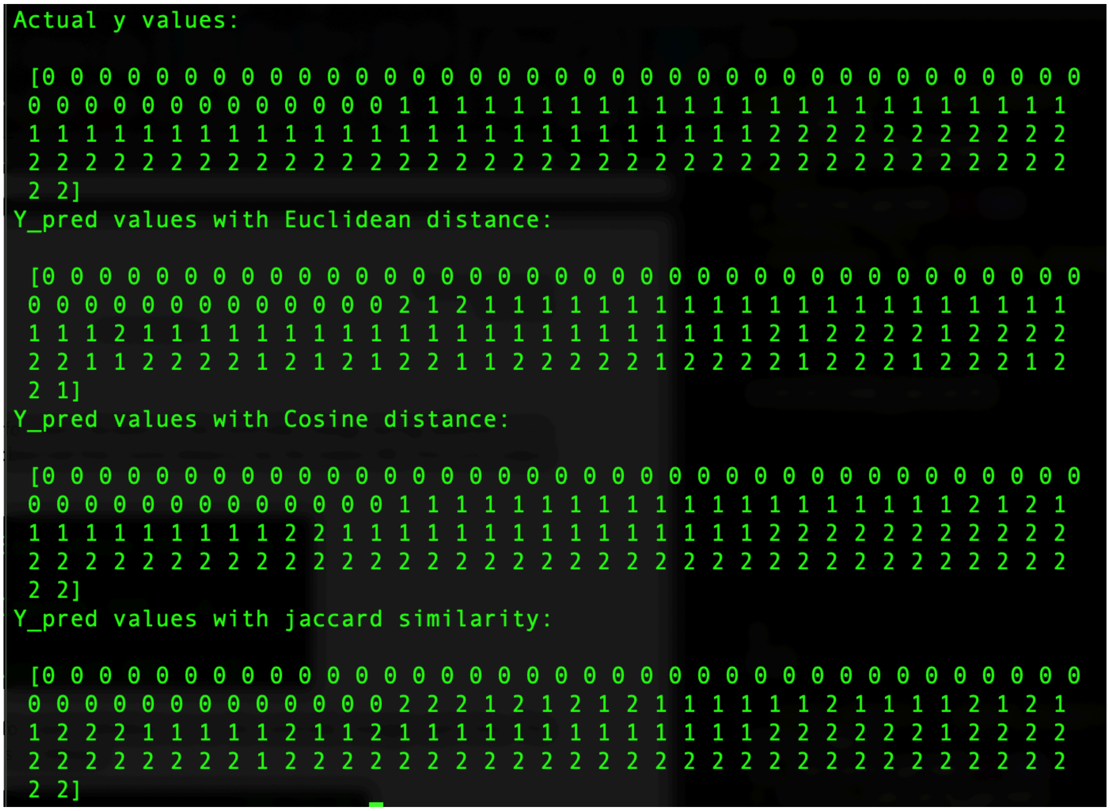
```
Initial Centroids:  (array([3, 2]), array([4, 8]))
The final converging centroids are:  [[4.85714286 3.57142857]
 [5.66666667 6.33333333]]
```

TASK 2:

K-Means is executed on Euclidean distance, Cosine distance and generalized Jaccard similarity. The number of categorical values in the iris dataset is 3. The class labels can be Iris Setosa, Iris Versicolour and iris Virginica. Hence, we will have three clusters. We also have 4 X features, sepal length, sepal width, petal length and petal width.

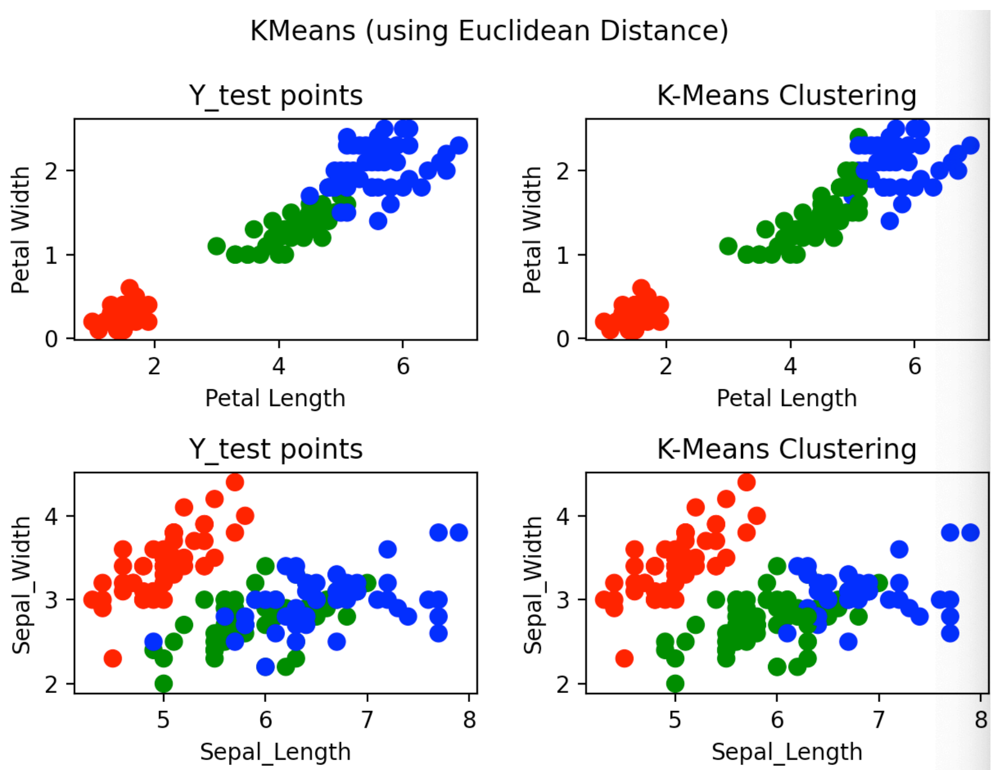
I have implemented K-Means on the iris dataset by picking centroids in three different methods.
A) I have picked absolutely random centroids
B) I have assigned fixed centroids
C) I have picked one random centroid, C1 and then calculated the farthest datapoint from C1 to be my next centroid, C2. And for the final centroid, I have taken the largest average of the distance between C1 to the new point and C2 to the same new point.

All the outputs shown are when centroids are picked using method C. Outputs for scenario A and B can be check implement the code.

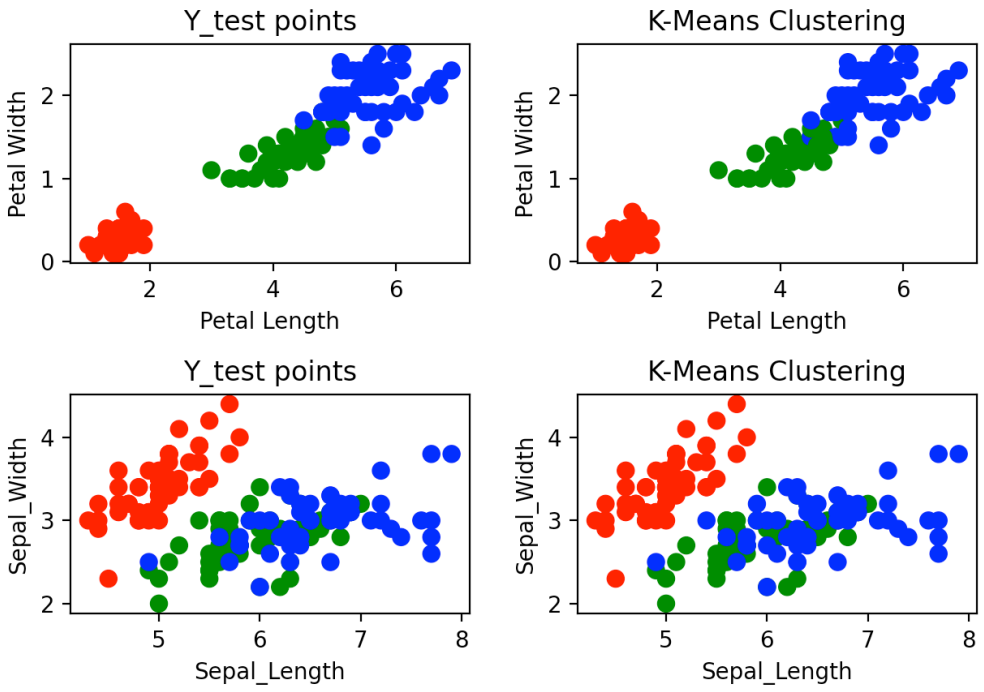


```
Actual y values:

 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
Y_pred values with Euclidean distance:

 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 2 2 2 1 2 2 2 2
 2 2 1 1 2 2 2 2 1 2 1 2 1 2 2 1 1 2 2 2 2 2 1 2 2 2 2 1 2 2 2 1 2 2 2 1 2
 2 1]
Y_pred values with Cosine distance:

 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 1
 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
Y_pred values with jaccard similarity:

 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 1 2 1 2 1 2 1 1 1 1 1 1 2 1 1 1 1 2 1 2 1
 1 2 2 2 1 1 1 1 1 2 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 1 2 2 2 2
 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

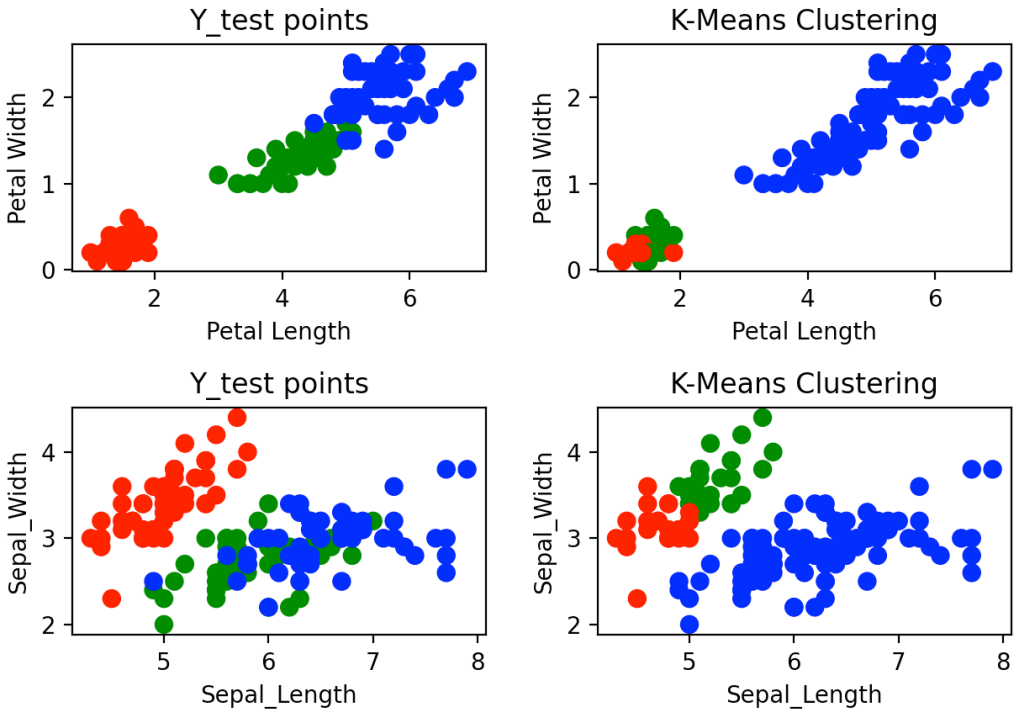
Plotting 2D graphs by taking 2 features at a time:



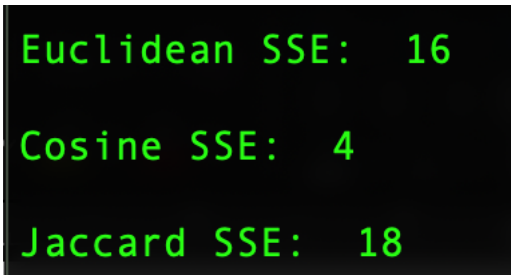
KMeans (using Cosine Distance)



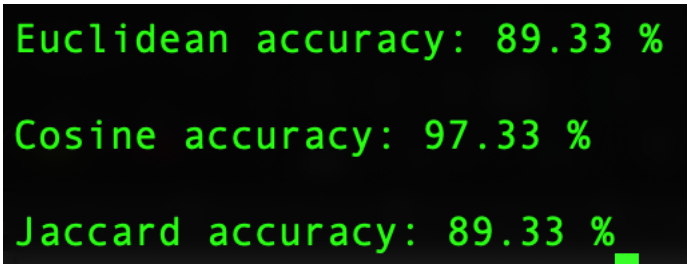
KMeans (using Jaccard Distance)



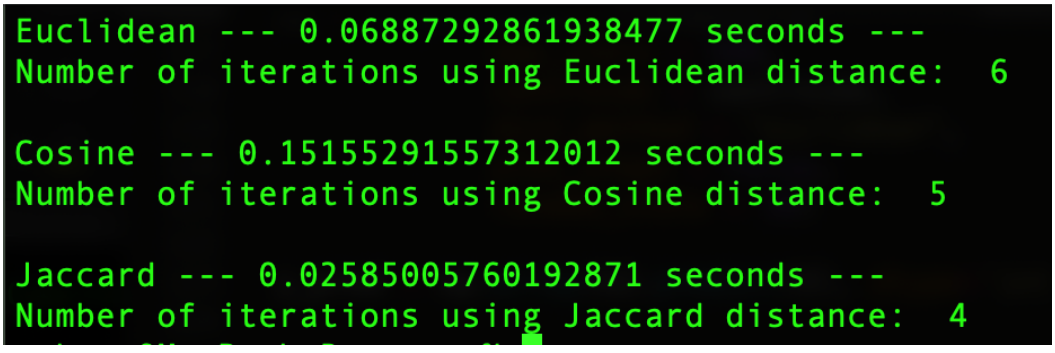
1) The SSE value changes for every iteration as the centroids change during different runs. Most of the times, it is observed that cosine distance has the lowest SSE. One such output is shown below:



2) The accuracy changes for each individual run as the centroids changes each time. The accuracy one such run is as follows. As seen below, for this iteration, cosine distance seems to have the highest accuracy majority of the times



3) The number of iterations for K – means to converge and the time taken for execution with each of the distance measures are shown as follows. Jaccard similarity seems to take the least amount of time and the least number of iterations to converge.



4) The above implementations ensured that the K-means clustering stopped when there was no further change in the centroids. Hence, for 4a, the implementation and the answer is the same as above. Outputs consistently changes as the centroids picked are random. For the below output, termination criteria is set to "*centroids*" (i.e, when centroids converge). Output from one of the runs is as follows:

```
Euclidean --- 0.07372903823852539 seconds ---
Number of iterations using Euclidean distance:  6

Cosine --- 0.09689712524414062 seconds ---
Number of iterations using Cosine distance:  2

Jaccard --- 0.05998992919921875 seconds ---
Number of iterations using Jaccard distance:  9
```

```
Euclidean SSE:  17

Cosine SSE:  67

Jaccard SSE:  16
```

As K-means implementation progresses, it was noticed that SSE never increased. SSE continuously reduced or stayed the same as the previous iteration. Setting this as my termination criteria in the code, the below output was obtained. Once again, it is important to note that since the centroids change, the output obtained at each run changes.

```
Euclidean --- 0.07010912895202637 seconds ---
Number of iterations using Euclidean distance:  5

Cosine --- 0.08709907531738281 seconds ---
Number of iterations using Cosine distance:  1

Jaccard --- 0.029273033142089844 seconds ---
Number of iterations using Jaccard distance:  3
```

```
Euclidean SSE:  39

Cosine SSE:  57

Jaccard SSE:  16
```

Even when the "*max_iters*" variable was set to a 100, the centroids converged much before that. One such output is as follows:

```
Euclidean --- 0.12609624862670898 seconds ---
Number of iterations using Euclidean distance:  12

Cosine --- 0.14054203033447266 seconds ---
Number of iterations using Cosine distance:  4

Jaccard --- 0.027818918228149414 seconds ---
Number of iterations using Jaccard distance:  4
```

```
Euclidean SSE:  16

Cosine SSE:  73

Jaccard SSE:  80
```

On an average, it was noticed that when the termination criteria is to check if SSE value increases in the next iteration or not, then K-means clustering algorithm converged much faster than the other two termination criteria.

Since the output varied so much due to the randomness of the centroids, I decided to run K-means using Euclidean distance, Cosine distance and Jaccard distance for 100 runs to get the average SSE, accuracy, time and number of iterations required to converge over the 100 executions.

Histogram plots are also plotted for the average values and the mean is indicated using the dotted line. The output is as follows:

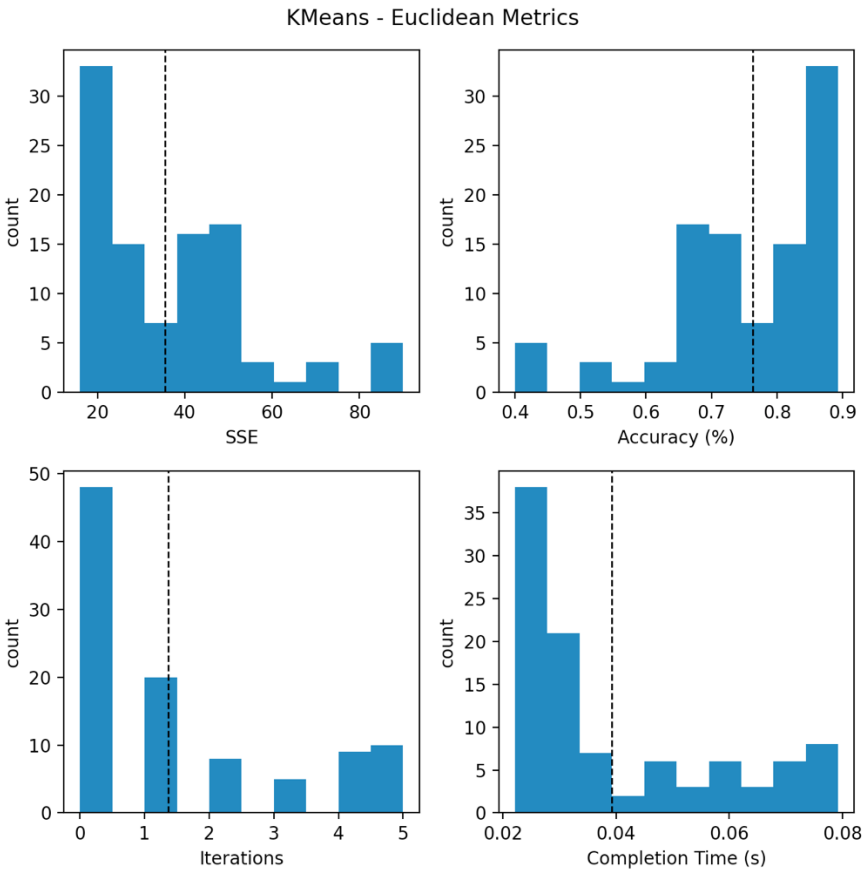
```
Euclidean Mean Values over 100 iterations:
+-------+------------+--------------+-----------+
|   SSE |   Accuracy |   Iterations |      Time |
|-------+------------+--------------+-----------|
| 35.53 |   0.763333 |         1.37 | 0.0392389 |
+-------+------------+--------------+-----------+
```

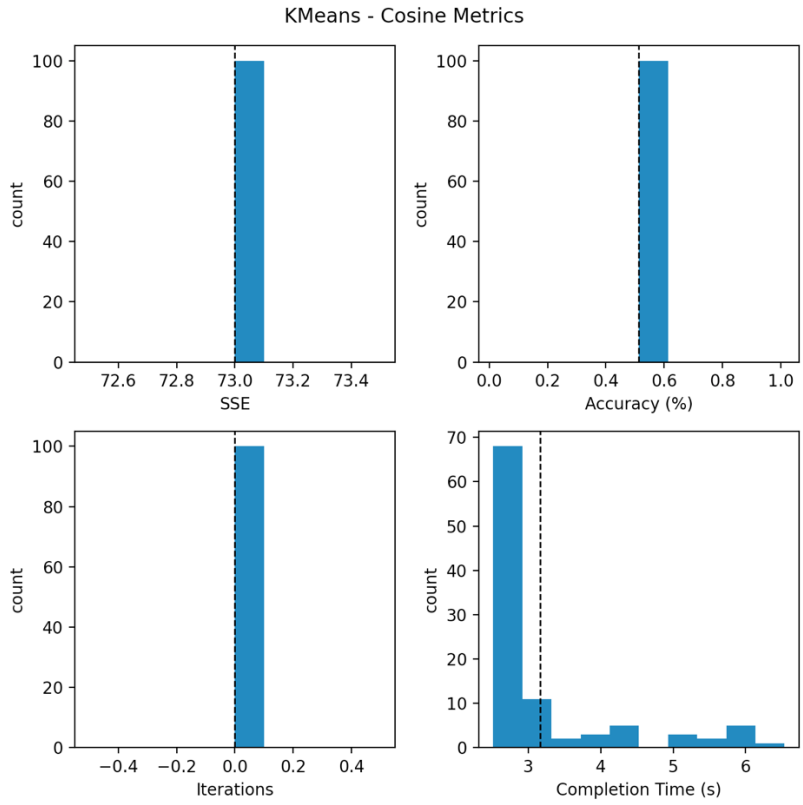
Cosine Mean Values over 100 iterations:

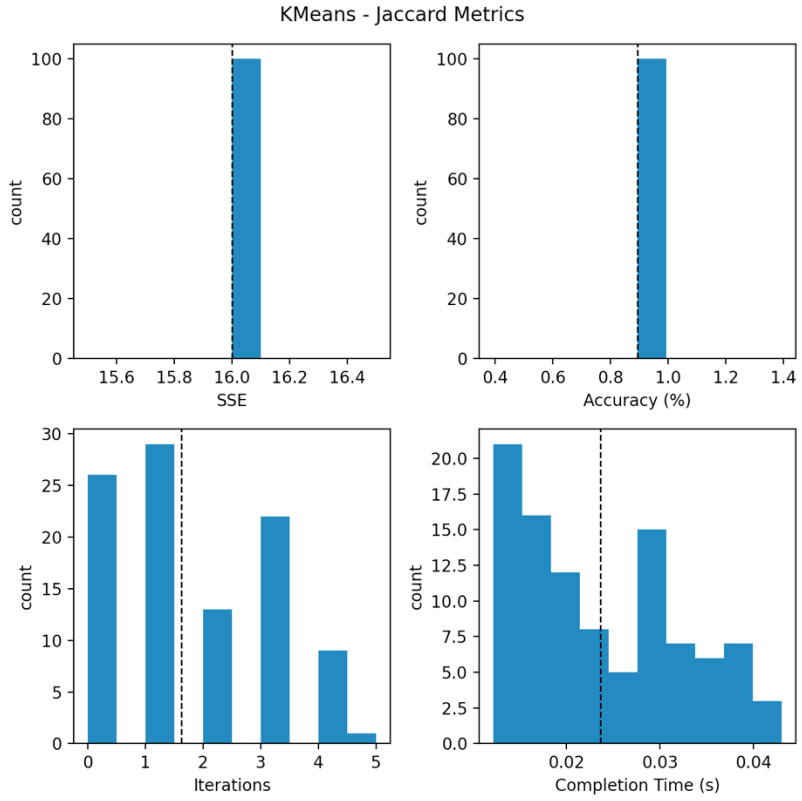
SSE	Accuracy	Iterations	Time
73	0.513333	0	3.16094

Jaccard Mean Values over 100 iterations:

SSE	Accuracy	Iterations	Time
16	0.893333	1.47	0.0272025




KMeans - Cosine Metrics
count
SSE
Accuracy (%)
Iterations
Completion Time (s)


KMeans - Jaccard Metrics
count
SSE
Accuracy (%)
Iterations
Completion Time (s)

TASK 3:

1) K-Means clustering fails to give efficient clustering results when we have outliers in the data. This is because the center for the clusters is taken by considering statistical mean value of all the datapoints and the outliers can change the overall mean of the datapoint in the cluster. Also, if the density spread of the datapoints across the data space is different and the data points follow non-convex shapes, then the standard K-means will fail.

   K-Means can also have bad clustering results when cluster densities and cluster sizes differ. Hence, it has the possibility of creating more clusters.

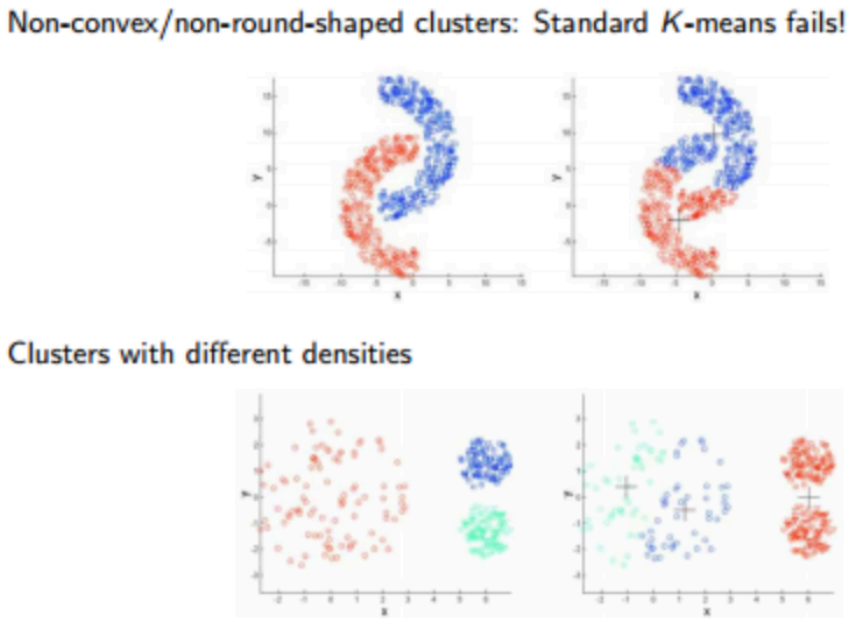
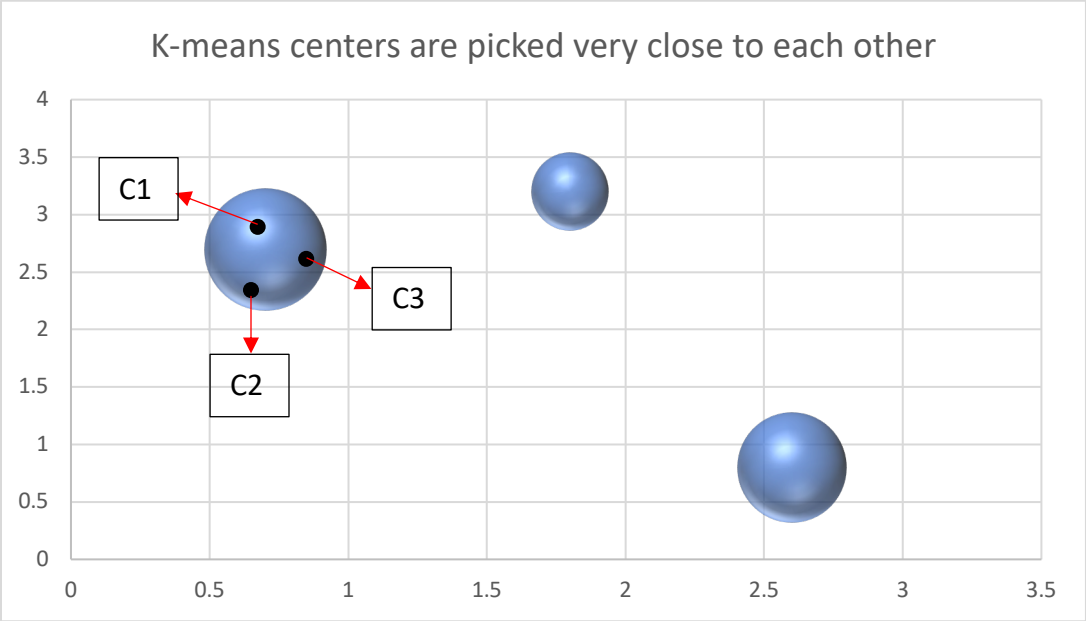
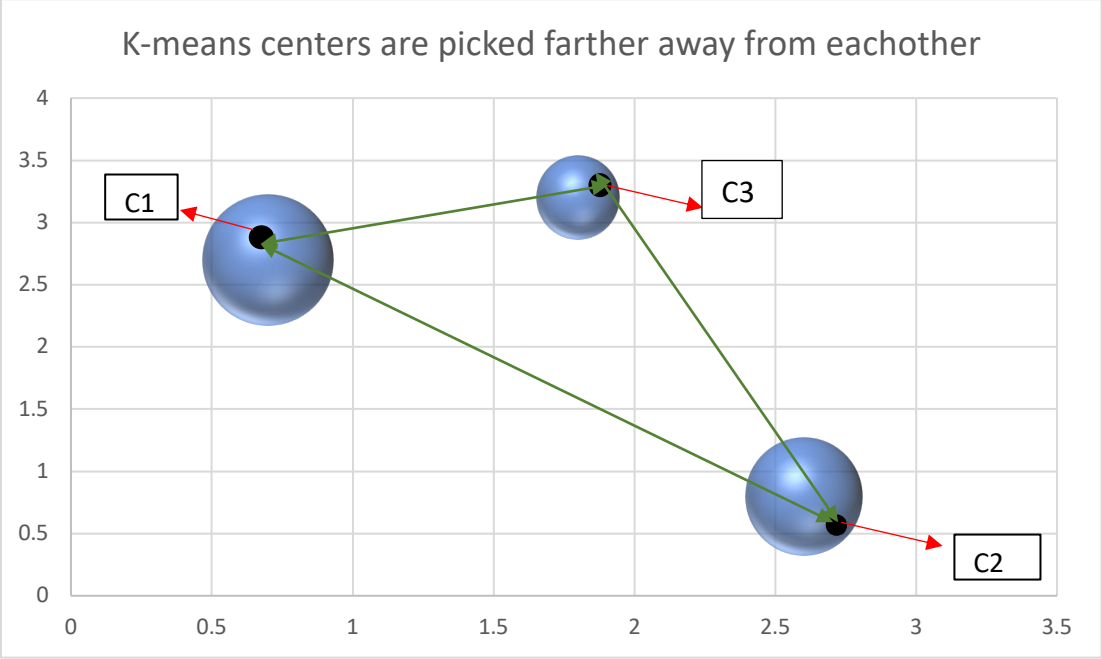


Figure taken from: https://www.analyticsvidhya.com/blog/2017/02/test-data-scientist-clustering/

2) One of the solutions to the limitations of K-means is by putting several centroids in the dataset. Another solution is to consider centroids further away from each other. Consider a situation where K-means has to randomly pick three centers during the initialization process. If these three centers are picked very close to each other, then K means will do a bad job clustering the data points.



However, this problem can be solved by ensuring that the randomly initialized centers are well separated and not grouped together, thus K-means clustering will not be misled. We can do this by picking center C1 and then scan all the data points and pick the data point that has the maximum distance from C1. We can call this data point as center, C2. Further, to pick the third center, take the longest average between the two centers C1 and C2, and call this as C3. These centers are further refined as K-means goes through a few iterations.



TASK 4:

The data points of the two clusters are considered and the Euclidean distance is used to compute the maximum and minimum distance between the cluster points are computed.

1) The farthest datapoints between the clusters are (4.6, 2.9) and (6.7, 3.1) and the distance between them is 2.1095

```
The two farthest_points are:  ([4.6, 2.9], [6.7, 3.1])
The farthest distance is: 2.1095
```

2) The closest datapoints between the clusters are (5,3) and (5.9, 3.2) and the distance between them is 0.9220

```
The two closest points are:  ([5, 3], [5.9, 3.2])
The closest distance is: 0.9220
```

3) There are 28 pairs of points in the entire dataset. The average distance between all pairs of datapoints is: 0.9830

```
Average distance between all pairs is: 0.9830
```

4) Min distance is sensitive to noise and outliers. Average distance and max distance are both less sensitive to noise and outliers. Max distance is least susceptible to noise or outlier datapoints.

GitHub link:

Assignment_05:

https://github.com/monicabernard/CAP-5610_Machine-Learning.git