Assignment 3, SENG 474

Quinn Webster

March 15, 2025

Introduction

The aim of this project is to explore Lloyd's algorithm using both uniform random initialization and k-means++ initialization, as well as hierarchical agglomerative clustering with both single linkage and average linkage. Throughout this report, we experimented with two datasets across two and three dimensions while working with all prementioned clustering methods. All models were built independently, those that used Lloyd's algorithm were written completely from scratch, whereas we used scikit-learns implementation for the hierarchical agglomerative clustering. This project required no preprocessing, as clustering was performed directly on the original data.

Throughout this project, we worked with two datasets. The first dataset contained 3500 datapoints in two dimensions. The second dataset contained 14801 datapoints in three dimensions. For clarity, throughout this paper we will refer to the first dataset as dataset1, and the second dataset as dataset2.

Lloyd's algorithm (k-means)

Uniform random initialization

Our first experiment involved fully implementing Lloyd's algorithm with uniform random distribution. A concise explanation of this algorithm's steps follows:

1. Initilization: Randomly select k points from the dataset as the initial centroids.
2. Assignment: For each point in the dataset, find the closest centroid and assign the point to that centroid's cluster.
3. Update: For each cluster, calculate the average location of all the points in that cluster and update each centroid to this new average location.
4. Repeat: Repeat steps 2-3 until the location of the centroids converge.

We then ran this algorithm on dataset1 for values of k in the range from 2 to 15. For each value of k, we ran the algorithm 25 times and computed the cost for each run, then calculated the average cost for each k. These results are show below in Fig 1. We repeated the same procedure for dataset2 with k values from 2 to 9, and these results are show below in Fig 2.

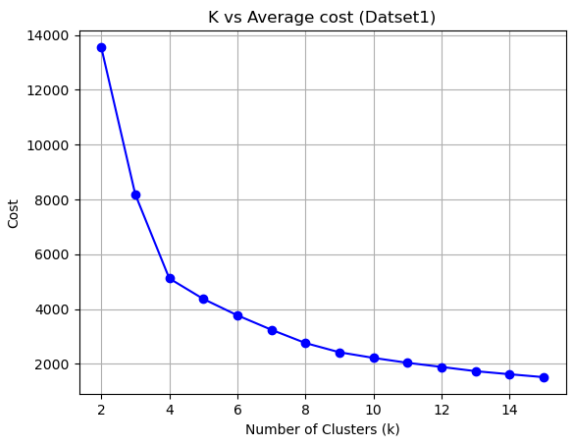

Figure 1

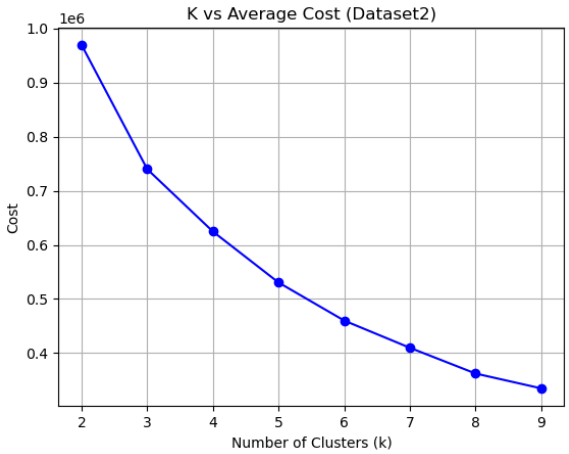

Figure 2

From Fig 1, we observe that there is a steep decline in cost as k increases from 2 to 4. This indicates that adding additional clusters in this range significantly improves the model. However, as k increases beyond 4, the costs continues to decease, but at a much slower rate. Therefor, for k greater then 4, the additional clusters result in diminishing returns. Therefor, the optimal number of clusters for dataset1 is 4. From Fig 2, we observe a similar trend, when k falls in the range 2-4, there is a steeper drop in cost, which indicates model improvement. Beyond k = 4, the decrease in cost slows. Therefor, for dataset2, the optimal number of clusters is also 4. We can observe that in Fig 1, there is a much more prominent "elbow", the point in which the steepness of the graph greatly decreases, making 4 clusters the clear optimal choice. However, in Fig 2, the elbow is less prominent, meaning that one could choose 3 or 5 clusters and still obtain reasonable results.

Fig 3 and Fig 4 show dataset1 and dataset2 split into their optimal number of clusters (4) after using uniform random initialization.

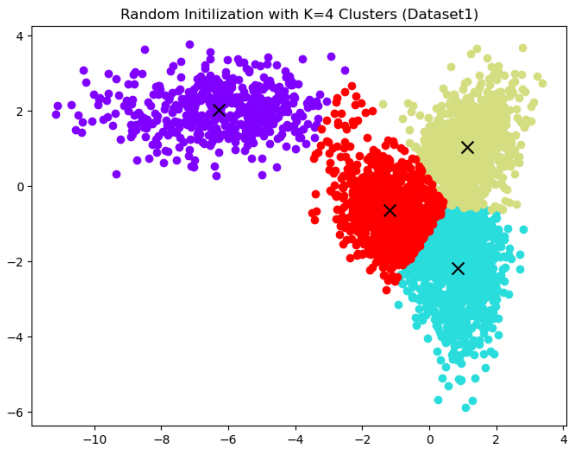

Figure 3

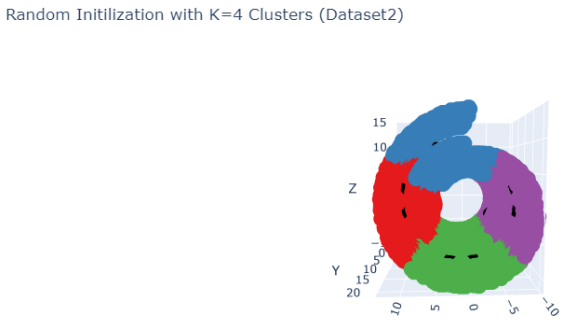

Figure 4

K-means++ initialization

We then implemented k-means++ initialization to go along with Lloyd's algorithm. The key difference between k-means++ and uniform random initialization is how the initial centroids are chosen. A brief description of the process of choosing the initial centroids is as follows.

1. **Initial Centroid**: Randomly select one centroid.
2. **Distance Calculation**: For each point in the dataset, compute its distance to the nearest centroid.
3. **Update**: For each cluster, calculate the average location of all the points in that cluster and update each centroid to this new average location.
4. **Repeat**: Repeat steps 2 and 3 until k centroids have been chosen.

We then ran the same experiment as mentioned above, computing the average costs for various values of k over multiple iterations for both dataset1 and dataset2. The results are shown in Fig 5 and Fig 6.

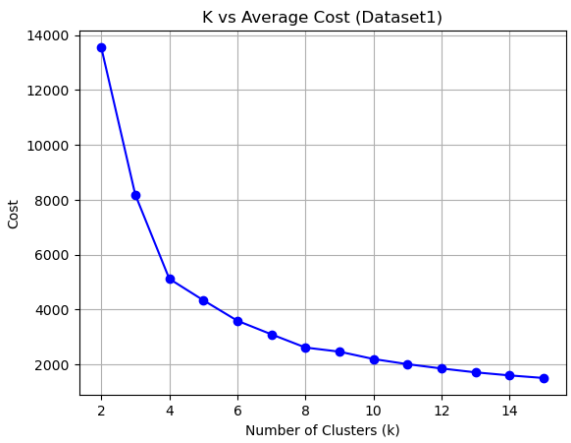

Figure 5

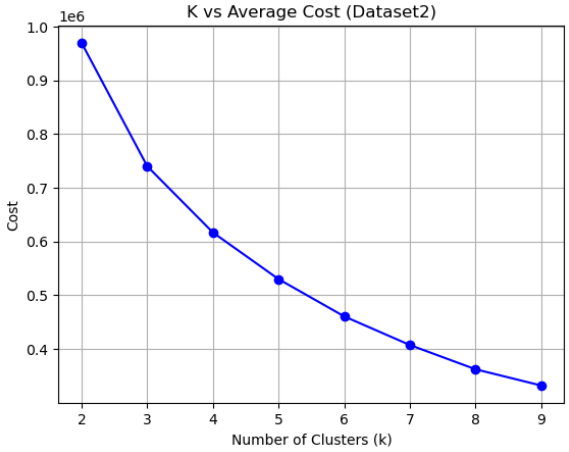

Figure 6

From Fig 5, we observe that there is a very steep decrease in cost as k increases from 2-4. After k = 4, this graph begins to decrease at a much slower rate, indicating a lesser trade off between the number of clusters and the associated costs. Therefor, 4 clusters is the optimal number of clusters to use for dataset1. This aligns well with our previous results from above. Fig 6 resembles Fig 2, in that there is a steeper drop when k is in the range of 2-4, and the decrease slows down as k becomes greater then 4. This indicates that the ideal number of clusters is also 4, due to the decrease in improvement as the number of clusters grows. Similar to above, it is important to note that Fig 5 has a much more prominent "elbow" then Fig 6. Since Fig 6 has a less prominent elbow, we could also choose 3 or 5 clusters to be the optimal number of clusters; however, this is open to the interpretation of the researcher.

Fig 7 and Fig 8 show dataset1 and dataset2 split into 4 clusters after using K-means++ initialization.

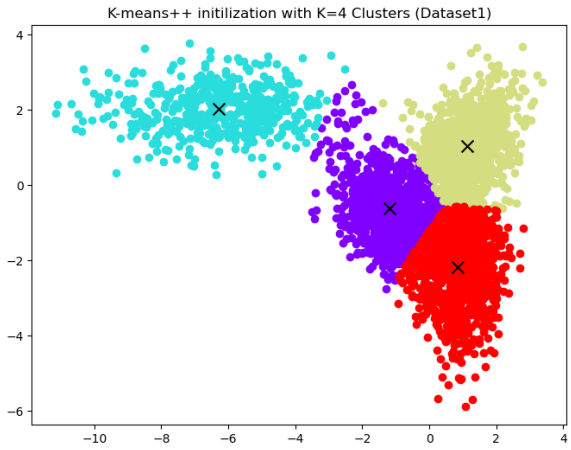

Figure 7

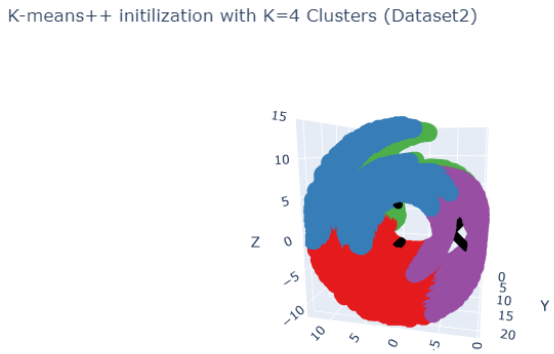

Figure 8

We then ran an experiment where we looked into the average number of iterations it took for a model to converge for various k values across both versions of Lloyds algorithm while using dataset 1. Our hypothesis was that k-means++ would converge faster then uniform random initialization due to its more effective selection of the initial centroids. The results are shown below in Fig 9.

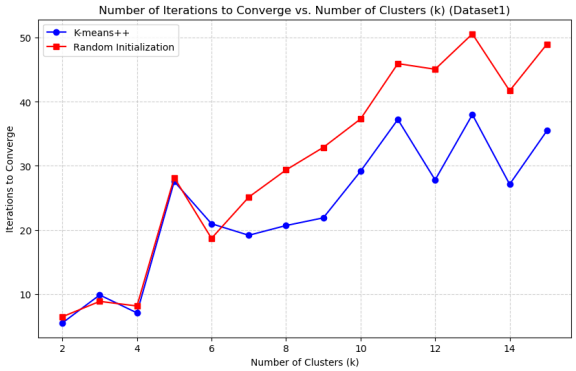

Figure 9

Our results align well with our hypothesis, on average, the number of iterations that it takes for the model to converge is greater when using Random Initilization then when using K-means++. This is because K-means++ is more likely to have better, more spread-out initial centroids. However, we can also see that for small values of k (k¡7), there are slight variations where in a few cases random initialization converged faster, this is likely due to the randomness in selecting initial centroids which occasional results in near-optimal positions. Even in these cases, the number of iterations to converge is only slightly less, but nearly identical to that of K-means++.

Performing the same experiment on dataset2 resulted in similar results (Fig 10), though with slightly more variation. At certain points, the random initialization converged faster then k-means++ did. This may be due to the small amount of iterations that we ran for each value of k, meaning that if we had took an average value from a larger sample size, we may have obtained more predictable results.

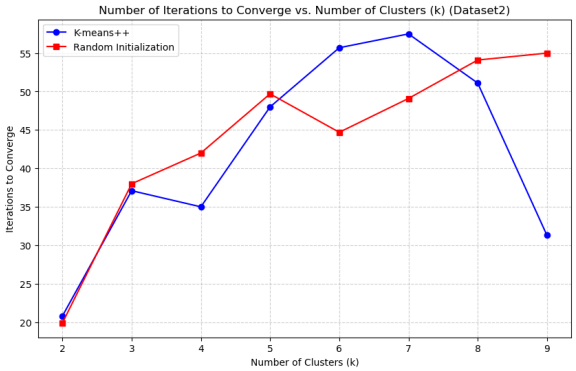

Figure 10

Hierarchical agglomerative clustering

Single linkage and average linkage

Next, we performed an experiment where we worked with hierarchical agglomerative clustering with both single and average linkage. Hierarchical agglomerative clustering is a method of clustering where initially each datapoint is its own cluster, and then, in a bottom-up approach, iteratively merges the closest clusters until a single cluster remains. The difference between single linkage and average linkage is how distances between clusters are calculate. For single linkage, clusters are merged based on the smallest distance between and two points in the clusters. For average linkage, clusters are merged based on the average distance between all points in each cluster.

We first ran single linkage and average linkage on dataset1, resulting in the dendrograms shown in Fig 11 and Fig 12.

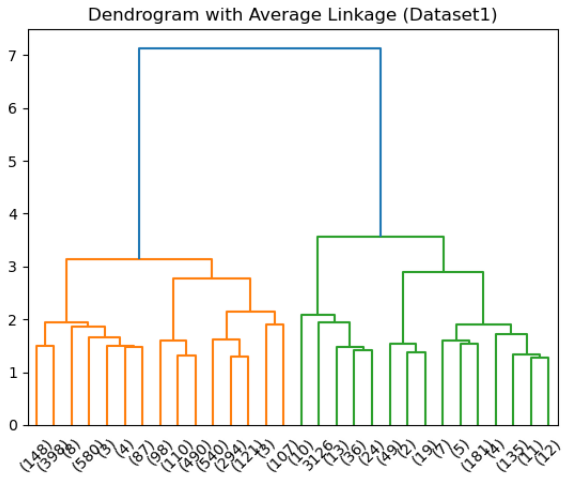

Figure 11

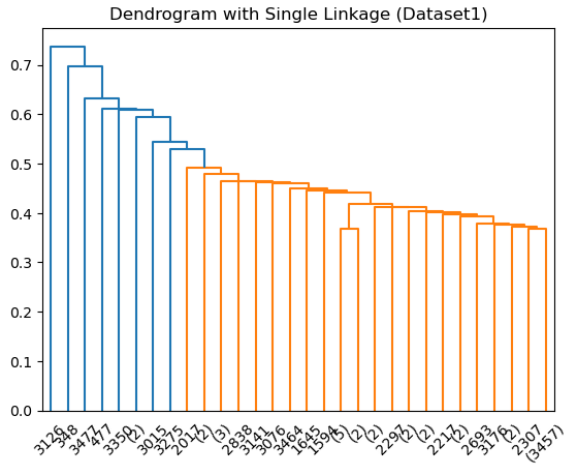

Figure 12

Since hierarchical agglomerative clustering continues merging clusters until it has reached one cluster, immediately, we are unaware of the optimal number of clusters. To decide on the optimal number of clusters, we found a horizontal cut in the dendrogram which splits the longest vertical line. For Fig 11, this cut results in 2 clusters, this split is shown in Fig 13. For Fig 12, the cut results in 3 clusters, this split is shown in Fig 14.

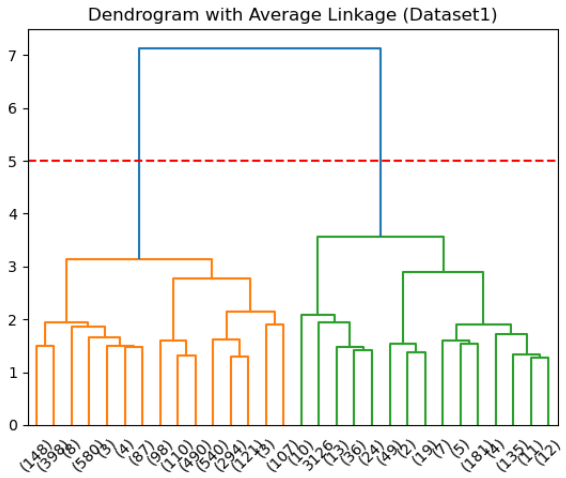

Figure 13

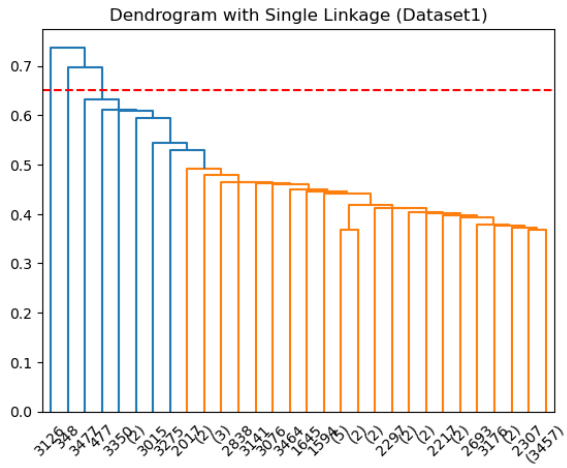

Figure 14

For clarity, the cut could be made at any point that would split the largest set of vertical lines. In Fig 13, I chose y = 5 arbitrarily, but this split could be made anywhere in the range y= 3.5 - y=7.

Fig 15 and Fig 16 show dataset1 split into its optimal number of clusters based on our splits of each dendrogram.

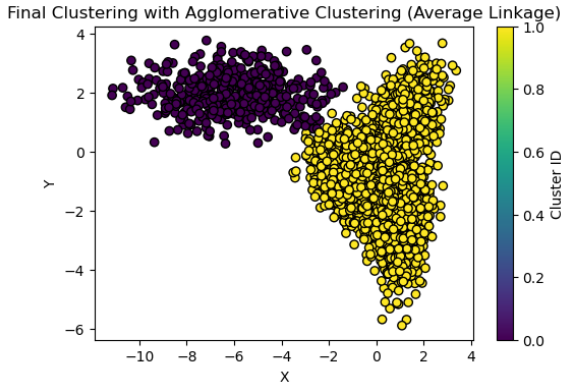

Figure 15

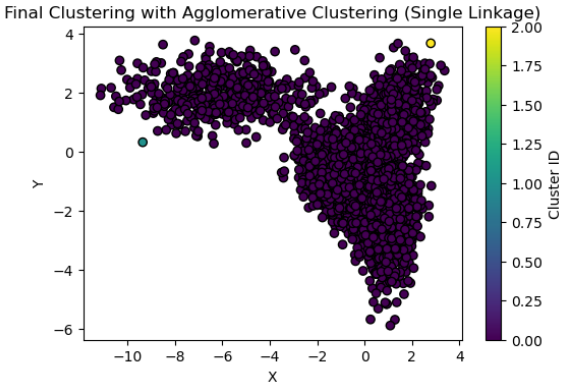

Figure 16

When working with average linkage, we observe that the clusters are formed quite well. We can clearly see two distinct clusters that are well split. However, when observing single linkage, we observe poor results. The data has been clustered into one large cluster and two individual points, resulting in 3 total clusters. This is because single linkage finds the smallest distance between one individual point for each cluster; therefor, a point that is slightly further away from others may never be merged into another cluster. This is apparent in Fig 16, as the two points that are in their own individual cluster, are slightly further away from any other points compared to others in the plot. This showcases that although single linkage may be useful for more chained data, it is not particularly useful for clustering on this specific dataset.

We then ran the same experiment on dataset2, the resulting dendrograms are shown in Fig 17 and Fig 18.

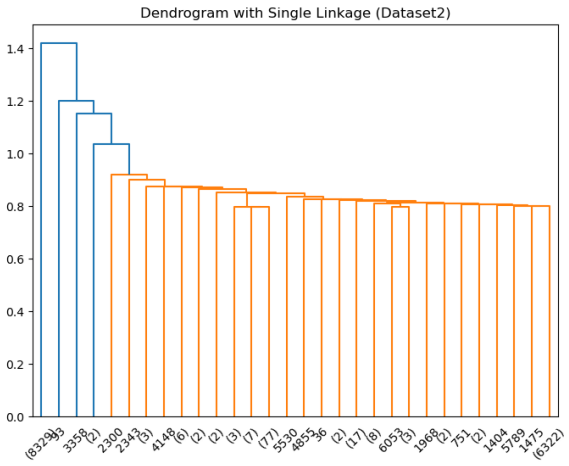

Figure 17

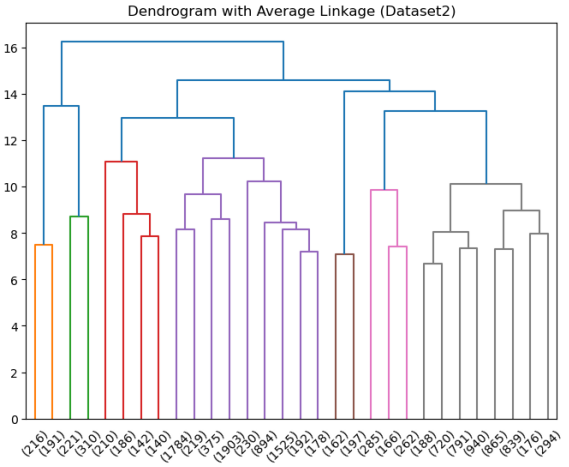

Figure 18

We then used the same cutting criteria to find the optimal number of clusters for dataset 2. For single linkage (Fig 17), we can see that an optimal split would be around 1.3, which would result in two clusters. However, if we are looking for more clusters, we could also make splits at 1.1 (resulting in 4 clusters) or at 0.9 (resulting in 5 clusters). In Fig 18, we could make a split at 15 (resulting in 2 clusters), but again, if we are looking for more clusters we could also make the cut at 12 (resulting in 7

clusters). These splits are shown below

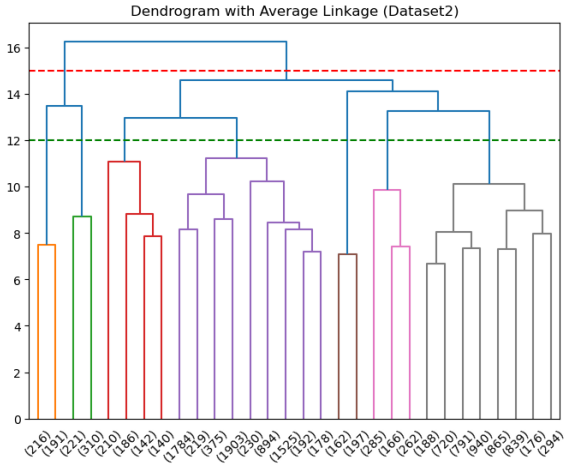

Figure 19

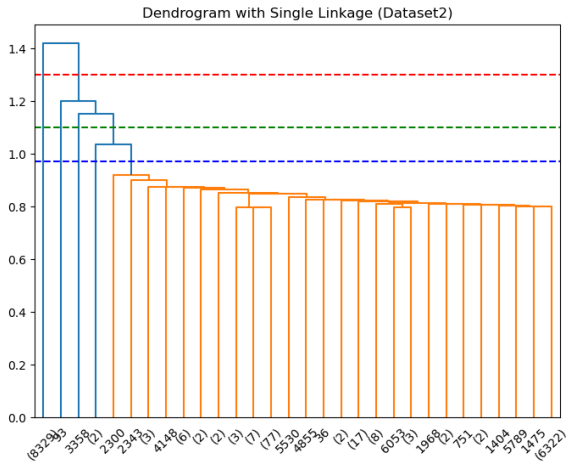

Figure 20

We included images of the plot for single linkage when working with both 2 (Fig21) and 5 (Fig 22) clusters. From these plots, we can observe that the single linkage performs quite well, clearly separating the data into two precise spirals, even when working with 5 clusters, it still form the two spirals, while classifying 3 points as being there own clusters. This demonstrates how that in certain scenarios; single linkage can be extremely powerful and form accurate clusters.

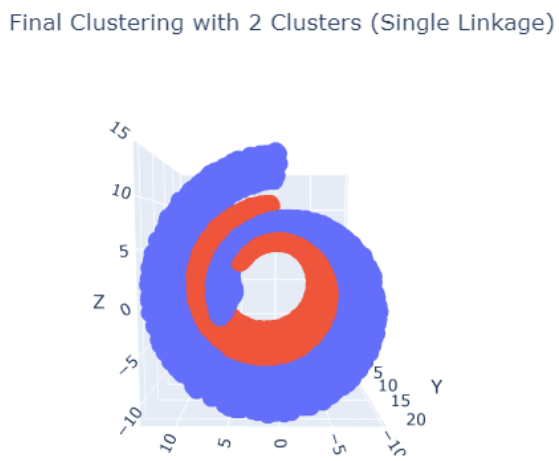

Figure 21

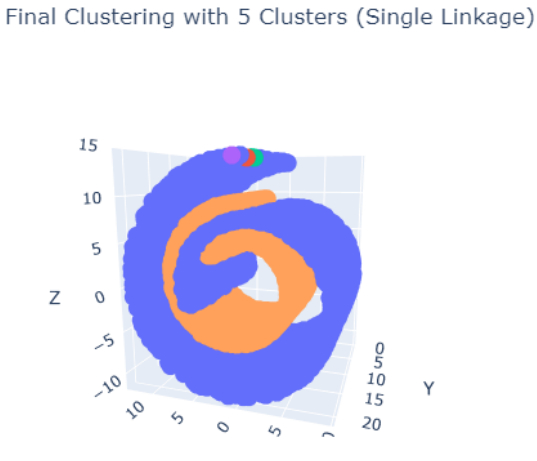

Figure 22

Then, when working with average linkage, we included the plots when working with both 2 (Fig 23) and 7 (Fig 24) clusters. From these plots, we can observe that the average linkage does not take the spiral formation of the datapoints into consideration. Instead, it form groups more similar to what we saw when working with Lloyd's algorithm. So in summary, for this particular dataset, using single linkage results in a better representation of the underlying, intertwined spiral formation of the dataset, where as average linkage results in clusters that are more independent of the spiral-like structure.

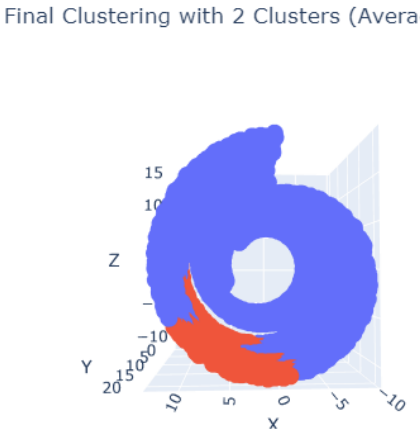

Figure 23

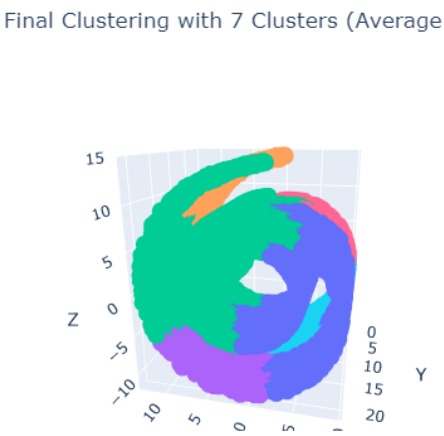

Figure 24