Clustering Algorithms

# Ensemble:

Every machine learning algorithm comes with its own set of underlying assumptions. These assumptions limit the useability of the algorithm on various problem sets. For instance, in our case, different CPT reading might not work well with different assumptions. Some examples of this include this: Say that the graph shows points of varying distances to each other, resulting in so many different densities at different places in the plot. Then it is not enough to cluster based on where the points are in the graph, like the k-nearest neighbor does. It needs a more powerful, more intuitive approach to handle these different densities, and this is why we included density-based clustering algorithms in this problem. However, there are other issues to consider, such as the complexity of the algorithms.

In such scenario, ensemble models work exceptionally well by combining the learnings of various algorithms. For our problem, we create an ensemble of the following algorithms:

1. K-Means
2. Mean-Shift
3. DBSCAN
4. OPTICS

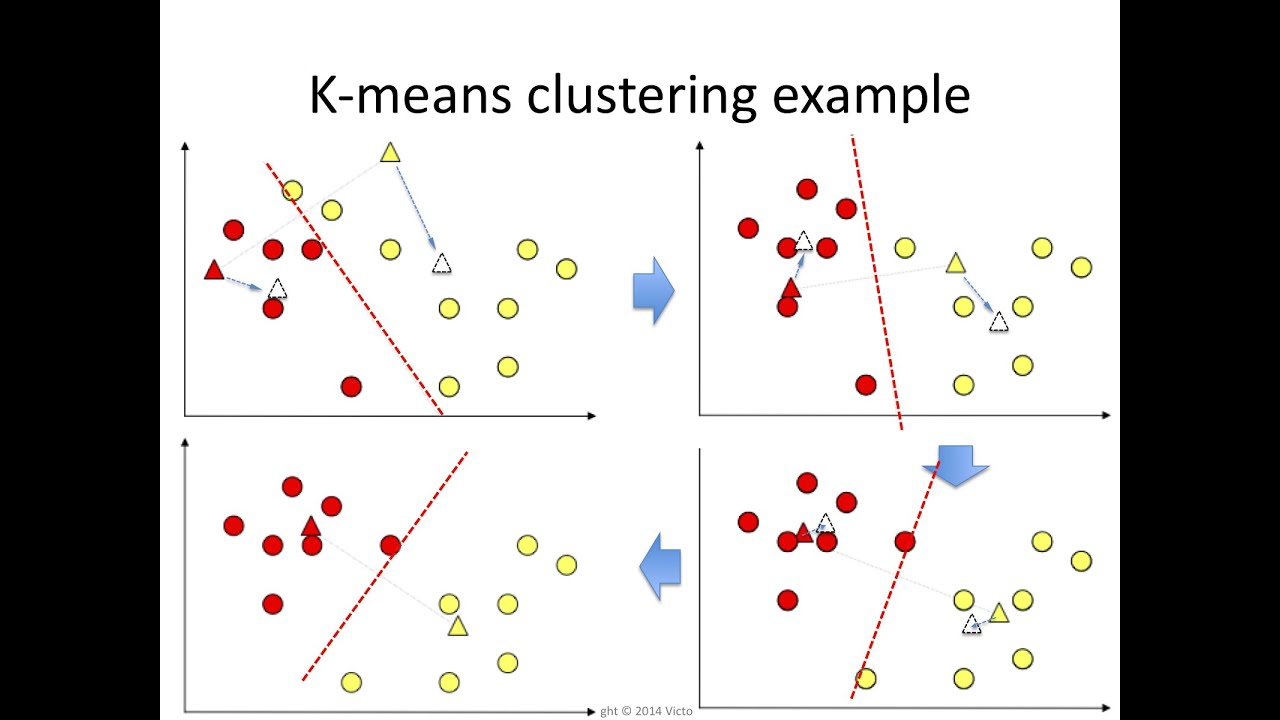
Below is the overview of each algorithm, explanation of the assumptions, visuals of the algorithms, as well as visualizations of each algorithm with our CPT data.

# K-Means Algorithm:

K-means is a simple yet powerful unsupervised learning approach. The way this algorithm works is that it takes in the value k, which is the number of groups to cluster the points in a graph or dataset. It then picks k “core” points randomly, and then groups the points based on which core point it is closest to. In other words, the core point with the minimum Euclidean distance to the current point being explored is the cluster that the current point is assigned to. Then, it recalculates the mean and uses that as a “core” point for the next iteration (Karim et al, 2020).

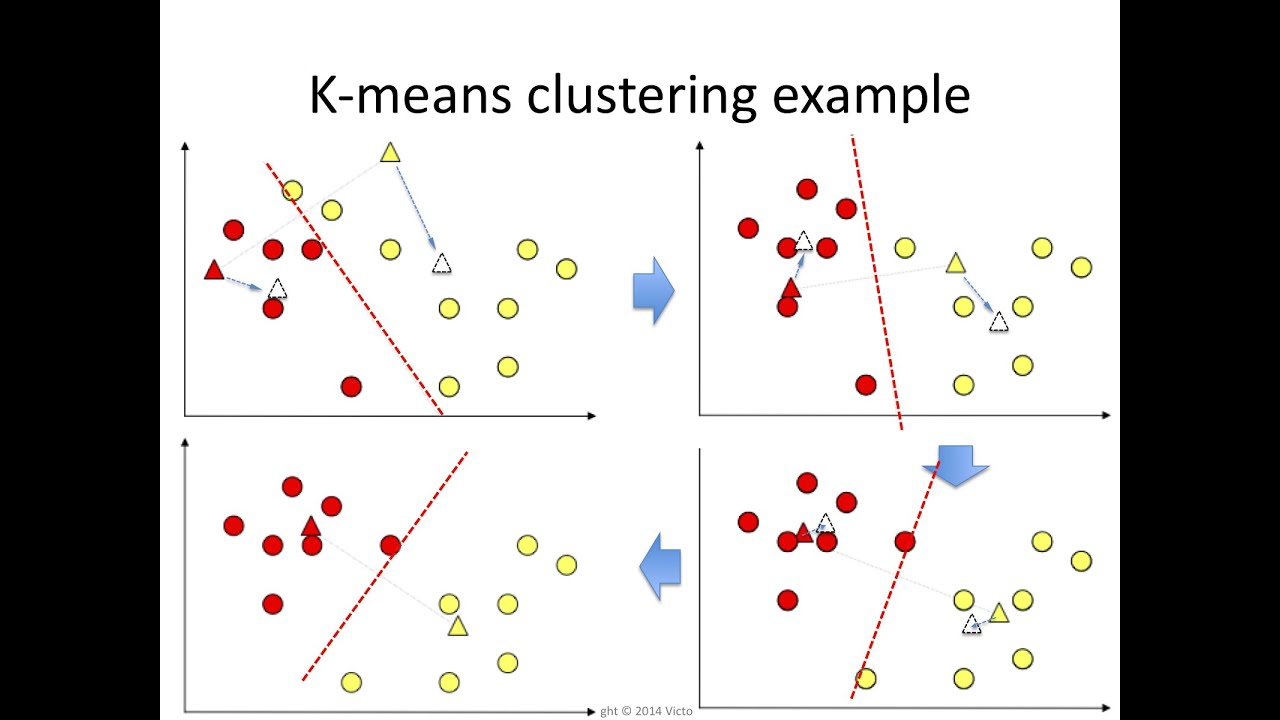
This algorithm takes in a number of assumptions. Its main assumption is that the feature space consists of K different circular/spherical regions and aims to cluster the data points in those regions (Karim et al, 2020). This means that the points will be clustered into K groups, otherwise they will end up uncategorized as noise. Another major assumption is that the K groups are equal in size and quantity of points (Karim et al, 2020). Therefore, its main weakness is that density-based clusters would not be clearly represented (Karim et al, 2020). This may present limitations for the soil data, as density-based clusters can provide more insight into the soil behavior at different depths. For example, an area in the plot where there is a high density of data points in a cluster showing high friction and low tip resistance indicates that those properties are highly present in the soil, whereas places in the plot with lower densities indicate the low presence of these properties. K-Means does not show that.

**Figures 1.1 - 1.4** is a pictorial on how an iteration of clustering works:



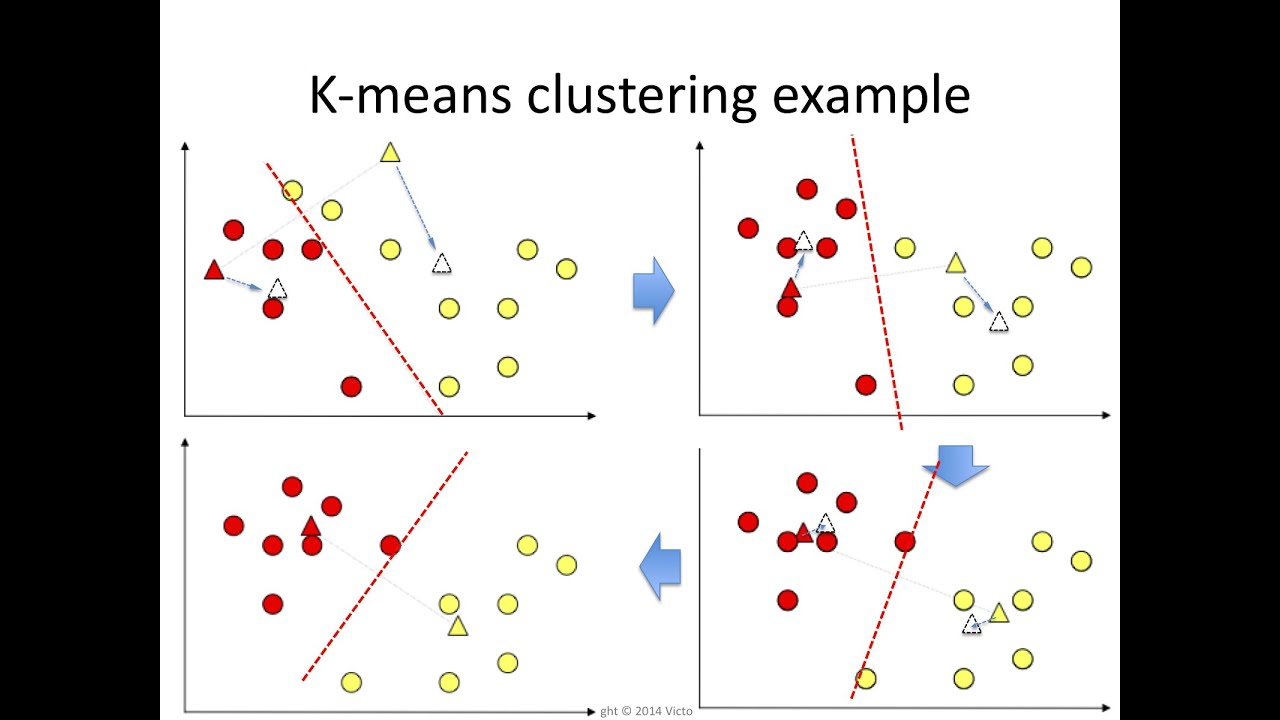
***Figure 1.1*** *(Lavrenko, 2015)*

**Figure 1.1** shows the starting iteration of the algorithm, where k is initialized to two clusters shown here, with two random core points (red triangle and yellow triangle). A line that is equidistant to the triangle points is the decision boundary, by which the points are then assigned to the cluster based on which core point they are closest to. The centroids are also calculated - the white triangles are the calculated centroids for their respective clusters.



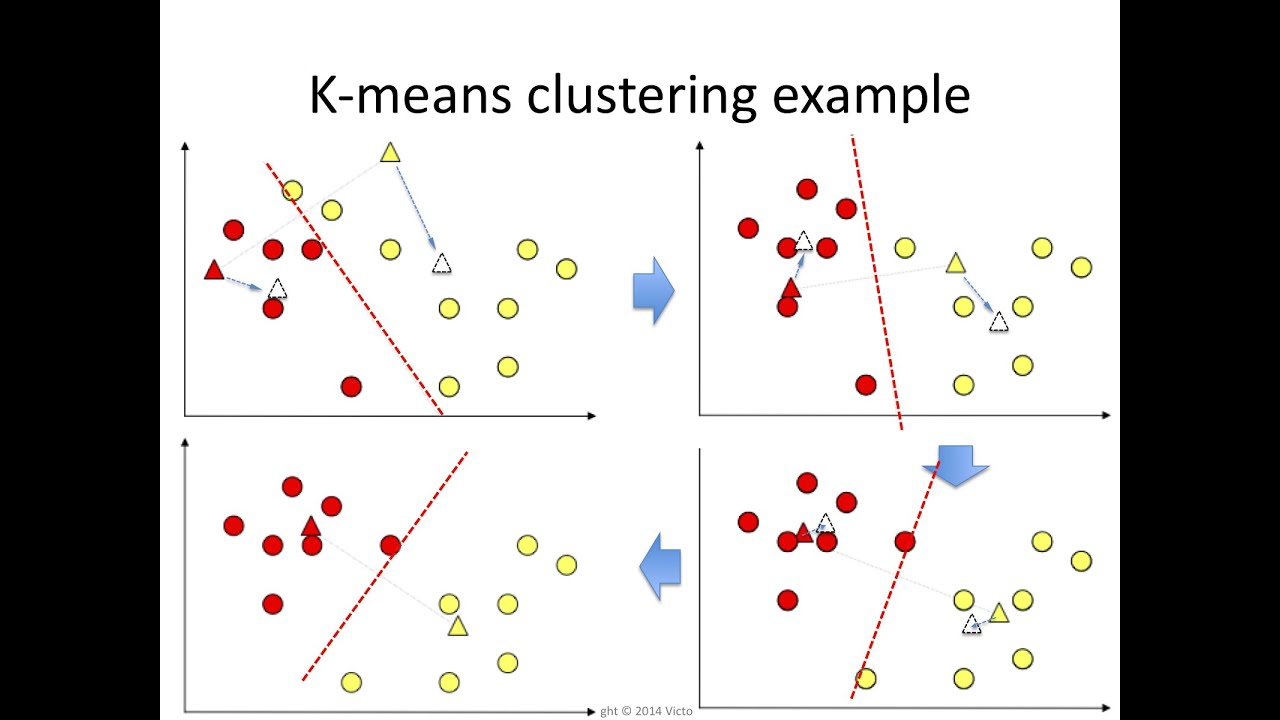
***Figure 1.2*** *(Lavrenko, 2015)*

**Figure 1.2** shows a second iteration where the updated centroids are now colored, a new decision boundary is drawn, and the points are re-clustered and recolored. New centroids are calculated again for the next iteration.



***Figure 1.3*** *(Lavrenko, 2015)*

**Figure 1.3** is just a representation of the subsequent iterations that repeat the steps mentioned from **Figure 1.1** and **Figure 1.2**.



***Figure 1.4*** *(Lavrenko, 2015)*

**Figure 1.4** shows that after numerous iterations, the centroids start to converge and the clusters are kept the same. This is an indication that all the points have been considered and clustered properly, and the K-Means Algorithm has finished.

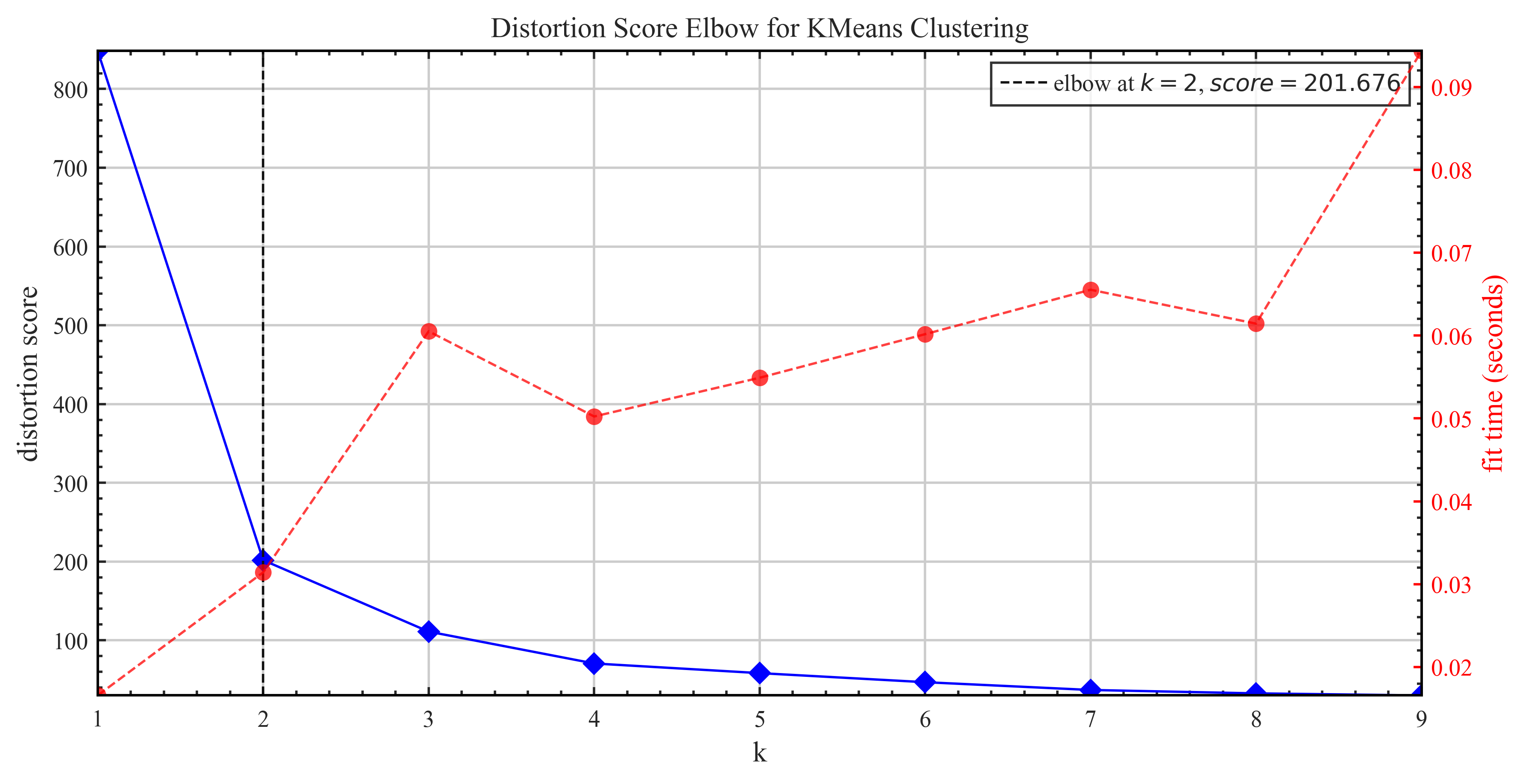
Below is the outline of steps mentioned earlier for brevity:

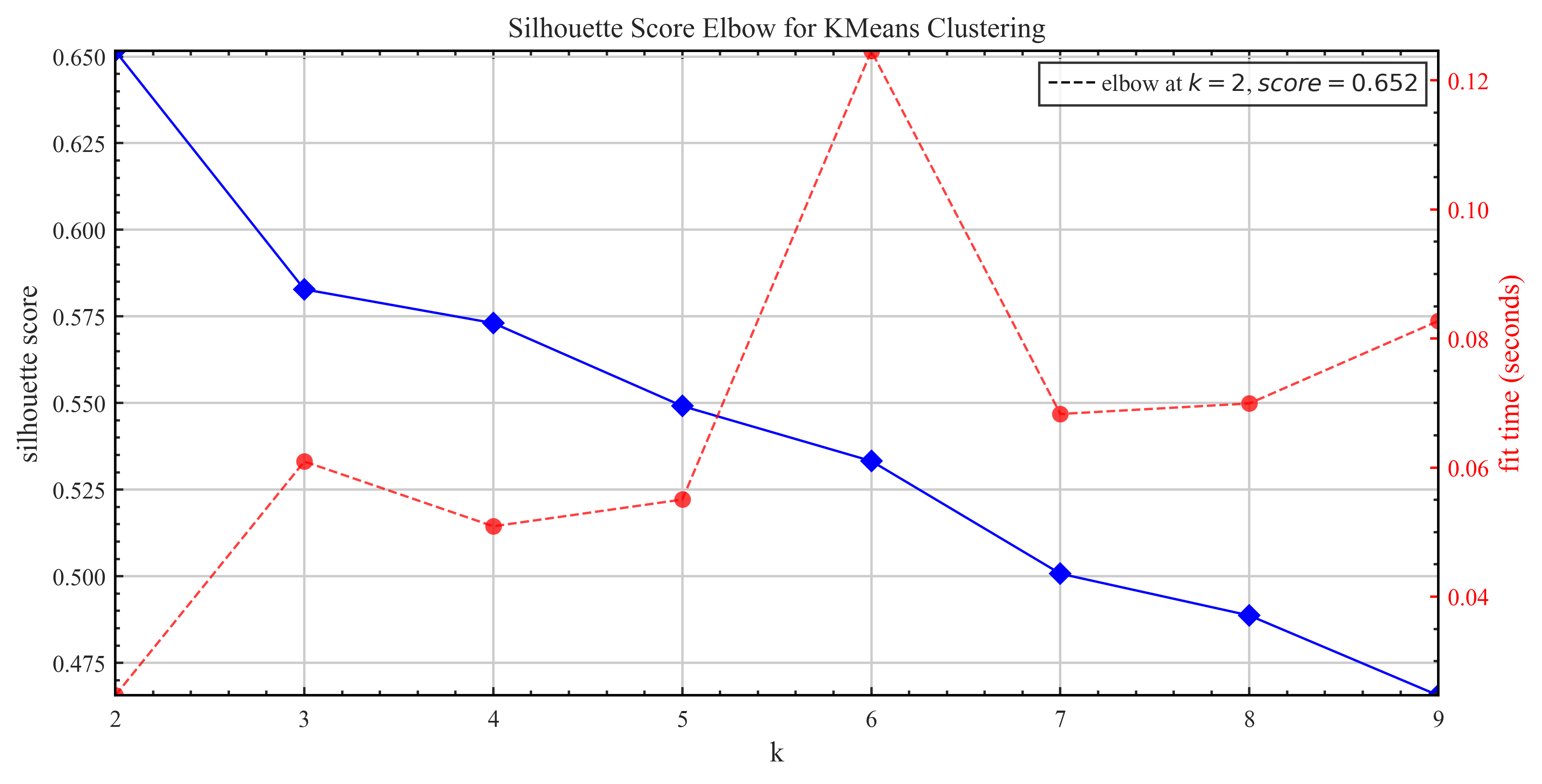
1. Initialize the algorithm by choosing the number of clusters k, as well as k random points as centers of the clusters in the feature space, or centroids.
2. Create a decision boundary that is equidistant from the centroids.
3. Assign each data point to its closest cluster based on some distance metric (Euclidean distance). Pictorially, points on one side of the decision boundary are in cluster 1 and the points on the other side are in cluster 2.
4. Recalculate the centers to be the mean of the data points belonging to that cluster.
5. Repeat steps 2-4 until all the points have been clustered and clustered properly.

## Estimation of K:

The main hyperparameter in this algorithm is k. If manual selection of the number of clusters is not preferable, we can use different estimation techniques to find the optimum value of k. The two techniques used for estimating k are:

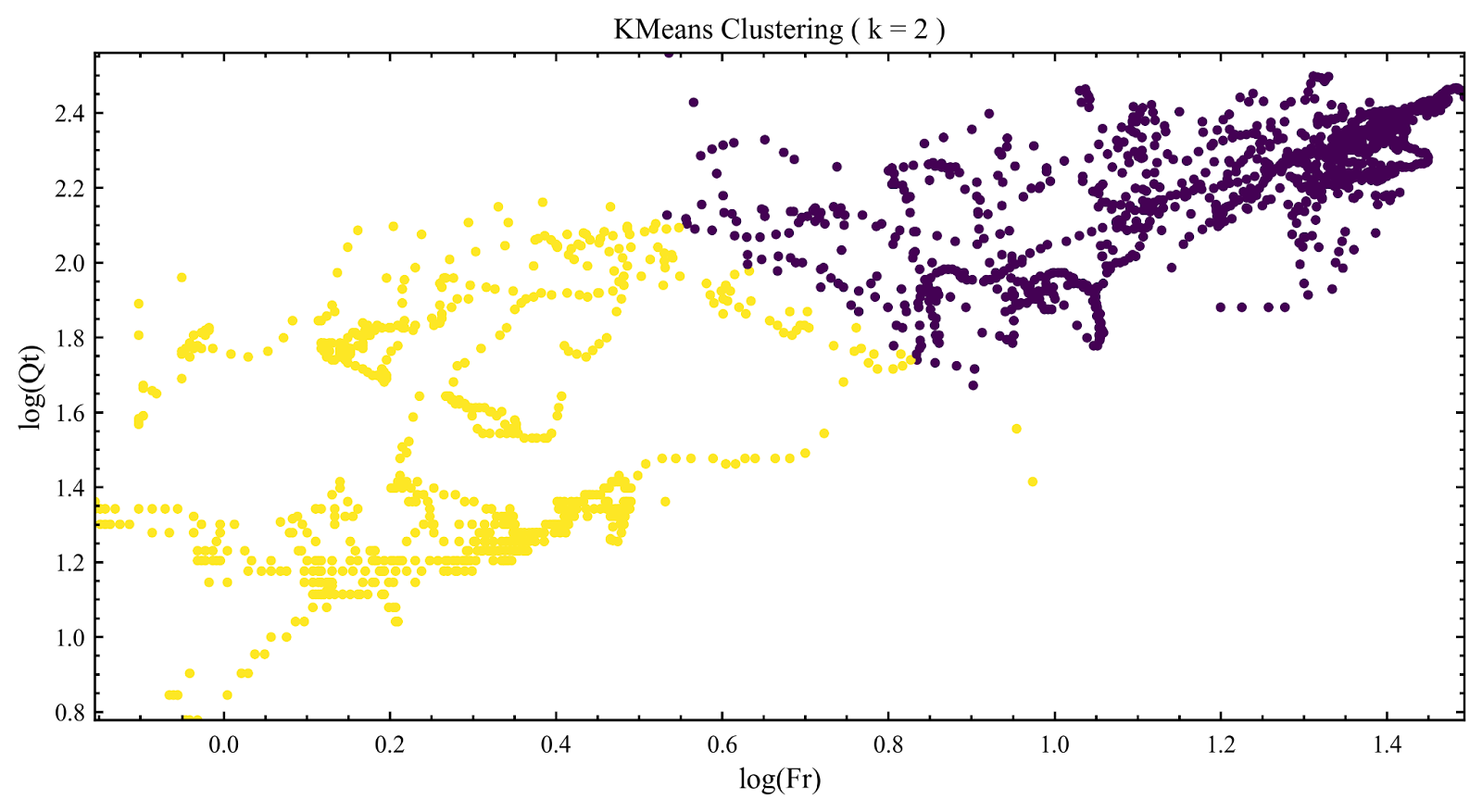
1. Distortion Score **(Figure 2.1):** This is a measure of the closeness of data points in a cluster to their mean. It is the sum of squared error (SSE) of all data points in a cluster from their mean. With increasing number of clusters, this score decreases at first rapidly as the model fits better, but then as the optimum point is crossed, starts to overfit and decrease linearly. Thus, the elbow point is the optimum number of clusters, which is k = 2 in our case.
2. Silhouette Score **(Figure 2.2)**: The elbow score is sometimes not accurate since it is hard to pinpoint the elbow. Thus, we can confirm our k value from silhouette score. This score is higher when the intra-cluster distance is minimum, and the inter-cluster distance is maximum for the number of clusters. Thus, its highest value corresponds to the optimum number of clusters, which in our case is like elbow method, i.e., k = 2.

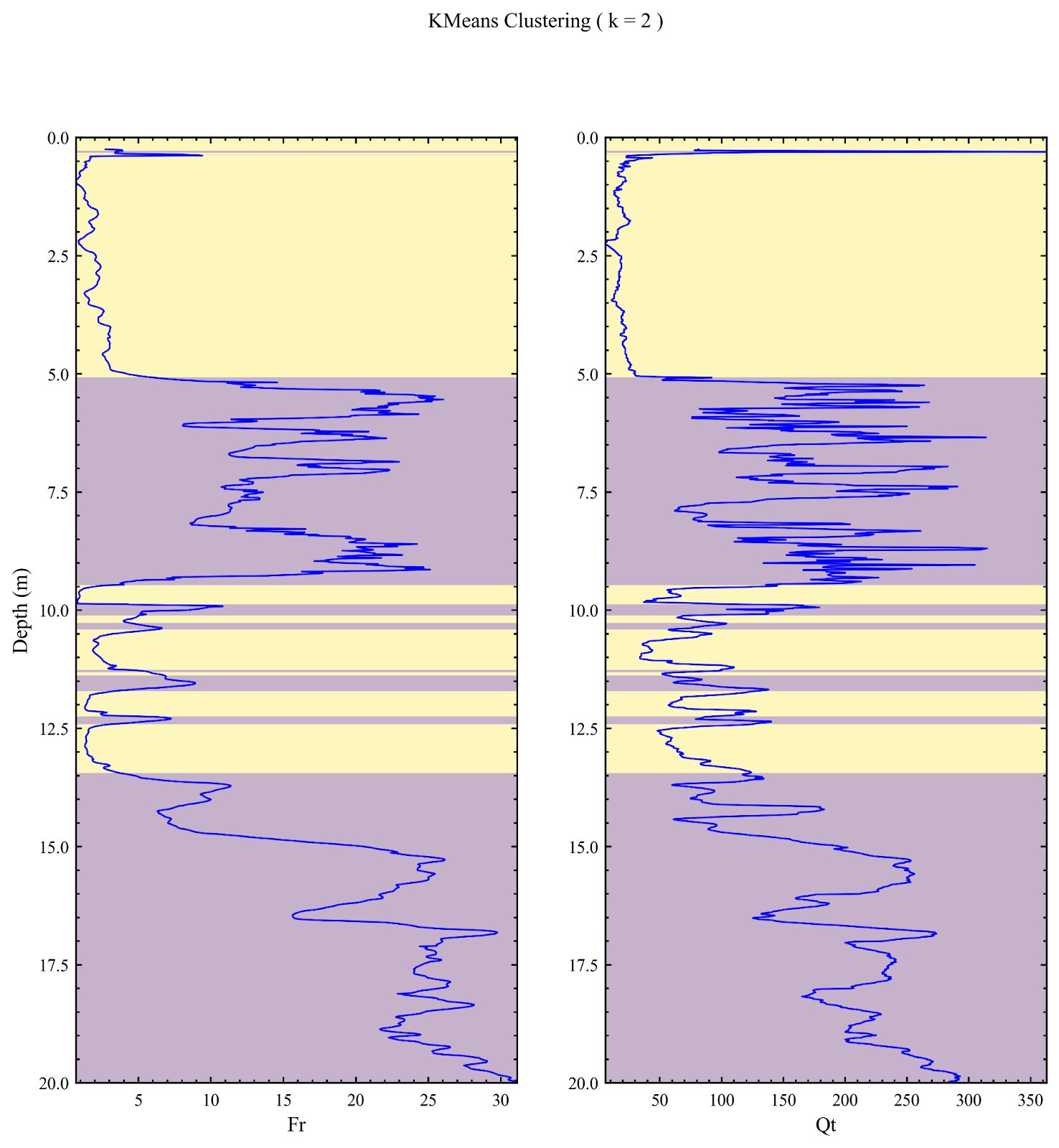
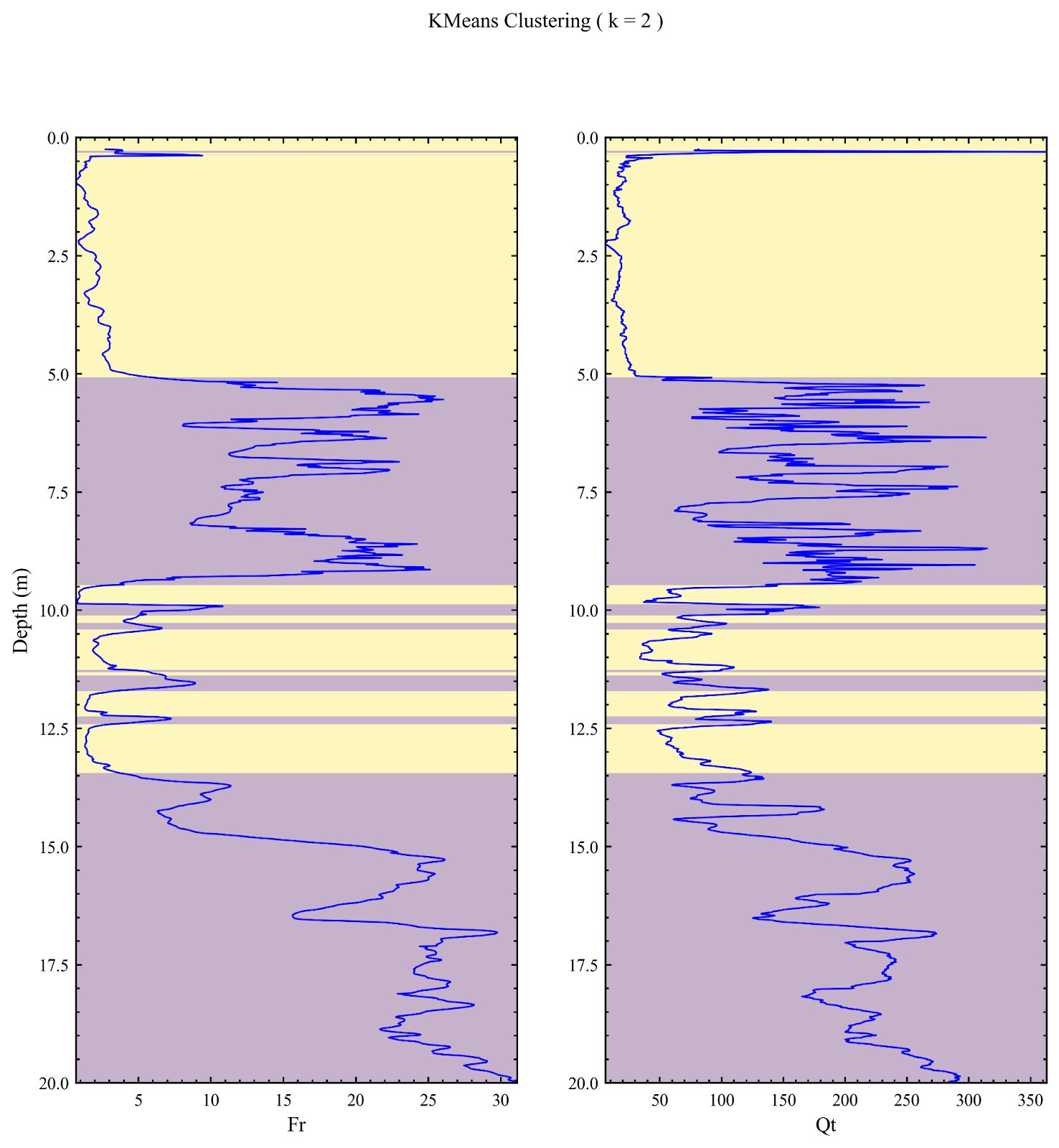
***Figure 2.1*** *- Distortion score with respect to k clusters*

***Figure 2.2*** *- Silhouette score with respect to k clusters*

## Results:

Setting k = 2 on our dataset, we get the following clustering (**Figure 3.1)** and soil layering (**Figure 3.2)**:

***Figure 3.1*** *- Points in 2 clusters*

***Figure 3.2*** *- Layering using 2 clusters, with respect to Depth, Friction (Fr) and Tip (Qt)*

# Mean-Shift Algorithm:

The Mean-Shift Algorithm is similar to the K-Means Algorithm, in that it clusters the points by each cluster’s mean location. However, the key difference is that while K-Means requires the algorithm to cluster the points into k clusters, Mean-Shift clusters the data points one cluster at a time and it discovers the number of clusters (Geif, 2018). This works well as it prioritizes places with higher densities and clusters the points up to maximum density for each cluster (Geif, 2018), meaning that those points are more likely to be grouped together when they are closer than when they are more distant.

Instead of specifying the number of clusters, you specify the radius, or bandwidth. The bandwidth is used in the Radial Basis Function Kernel (Scikit-Learn, 2011), which means the radius is given to specify a search region wherein the function detects the points to be put in that in that same cluster. This kernel is essential for clustering the points up till convergence, which is when the points within the bandwidth at current iteration I are the same as the points in the bandwidth at the previous iteration I – 1. More explanation below:

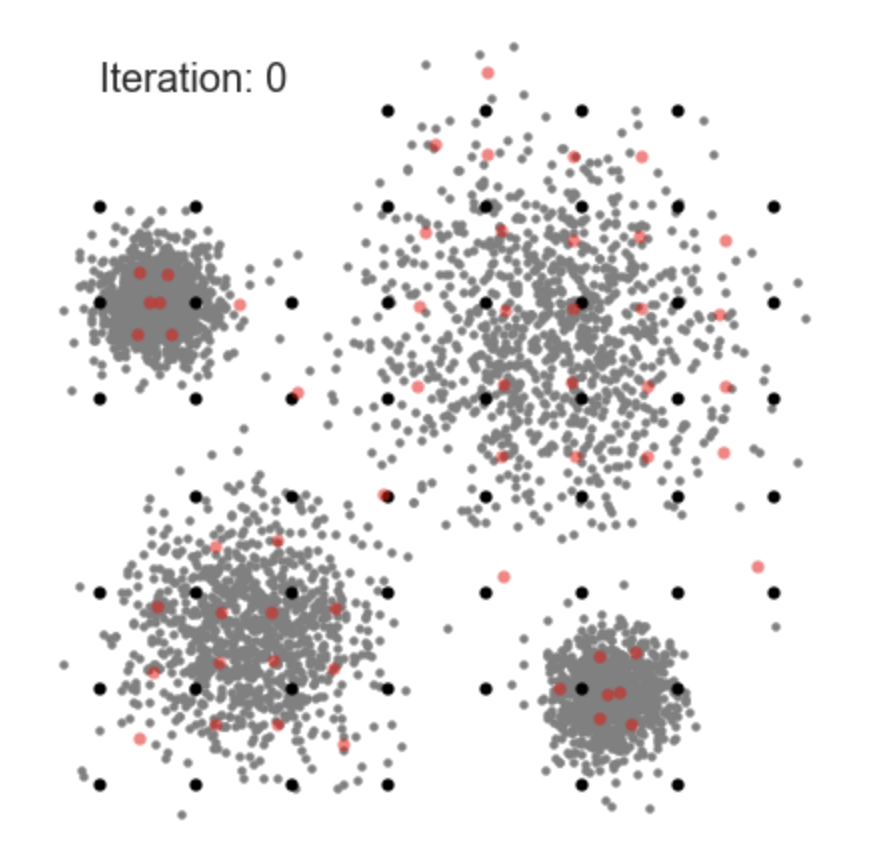
This algorithm works as follows (Geif, 2018):

1. The algorithm starts off by considering each data point as a cluster and choosing one of these points as a center. Using the bandwidth/radius specified as a parameter, it identifies the points within that bandwidth, adds them into the current cluster, and using these points, it calculates the mean location for the bandwidth to center around for the next iteration.
2. Using this new center and bandwidth, it identifies the points within that current bandwidth region, adds those points into the current cluster, and uses those points to recalculate the centers.
3. You keep repeating steps 1 and 2 continuously. Note that the centers and bandwidth region keeps shifting to higher densities, or higher density spaces (Geif, 2018).
4. Once the center starts to not change much, it means that it has converged. Locate that center and make that the center of the cluster just formed.
5. Remove the overlaps, meaning the smaller clusters identified should be removed: only the largest cluster that encompasses these points.
6. Repeat steps 1-5 until all of the points are re-clustered.

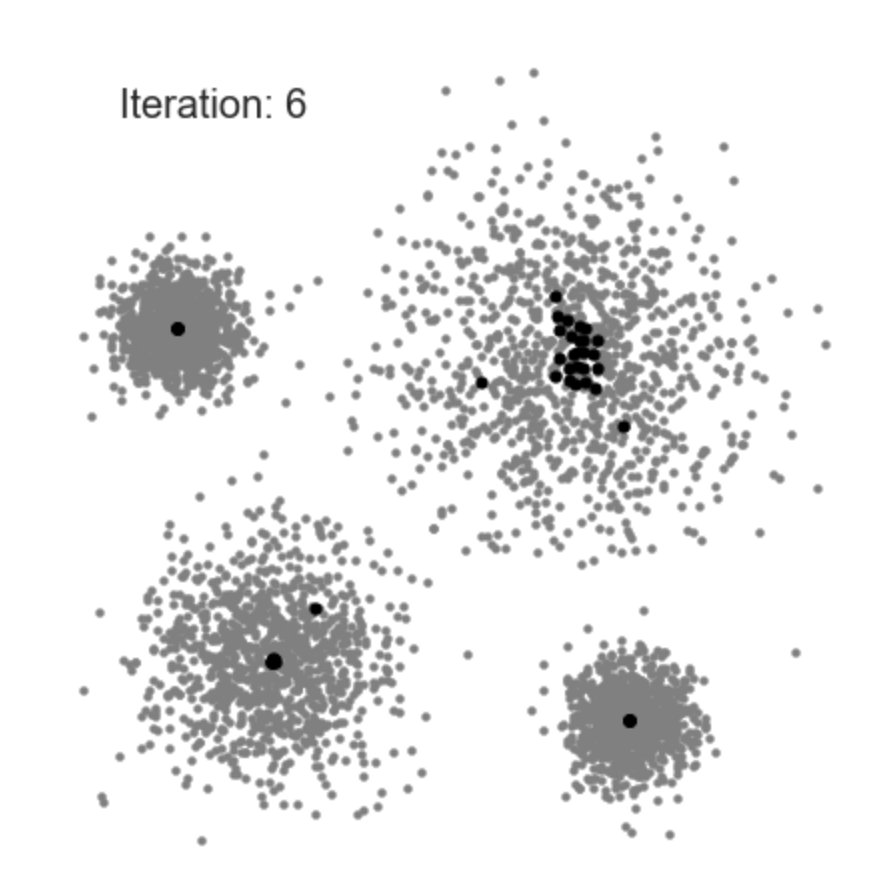
Below are some visualizations of the steps above:



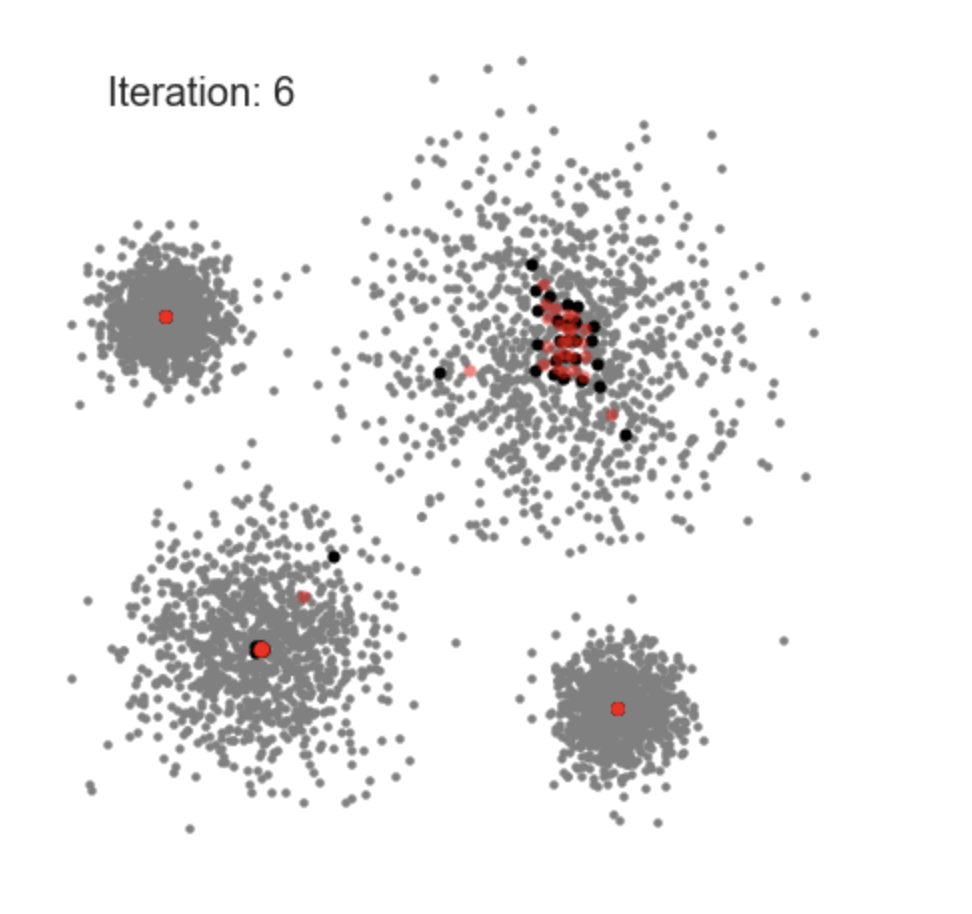
***Figure 4.1*** *- Start with a set of points in a graph, black dots as centroids (Geif, 2018)*



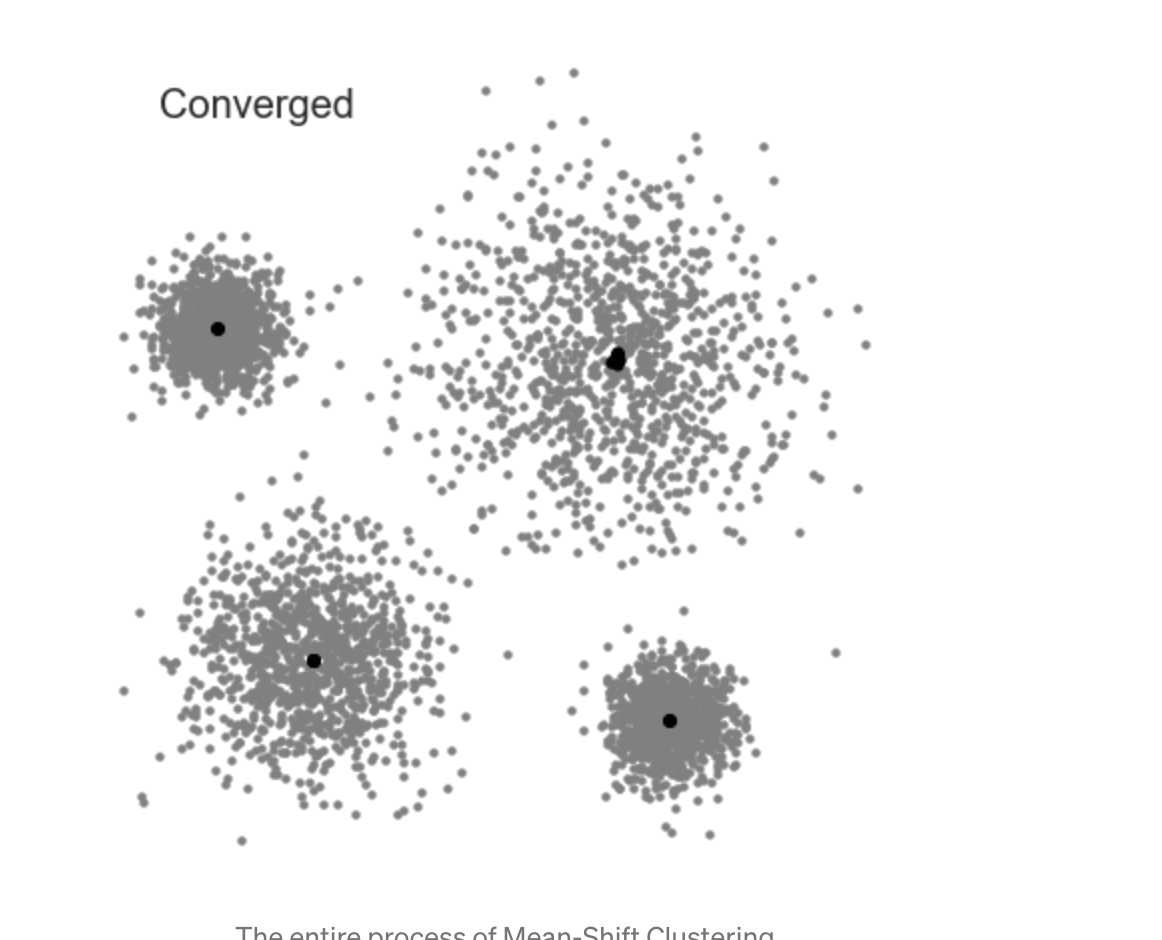
***Figure 4.2*** *- Recalculate the centers. (Geif, 2018)*



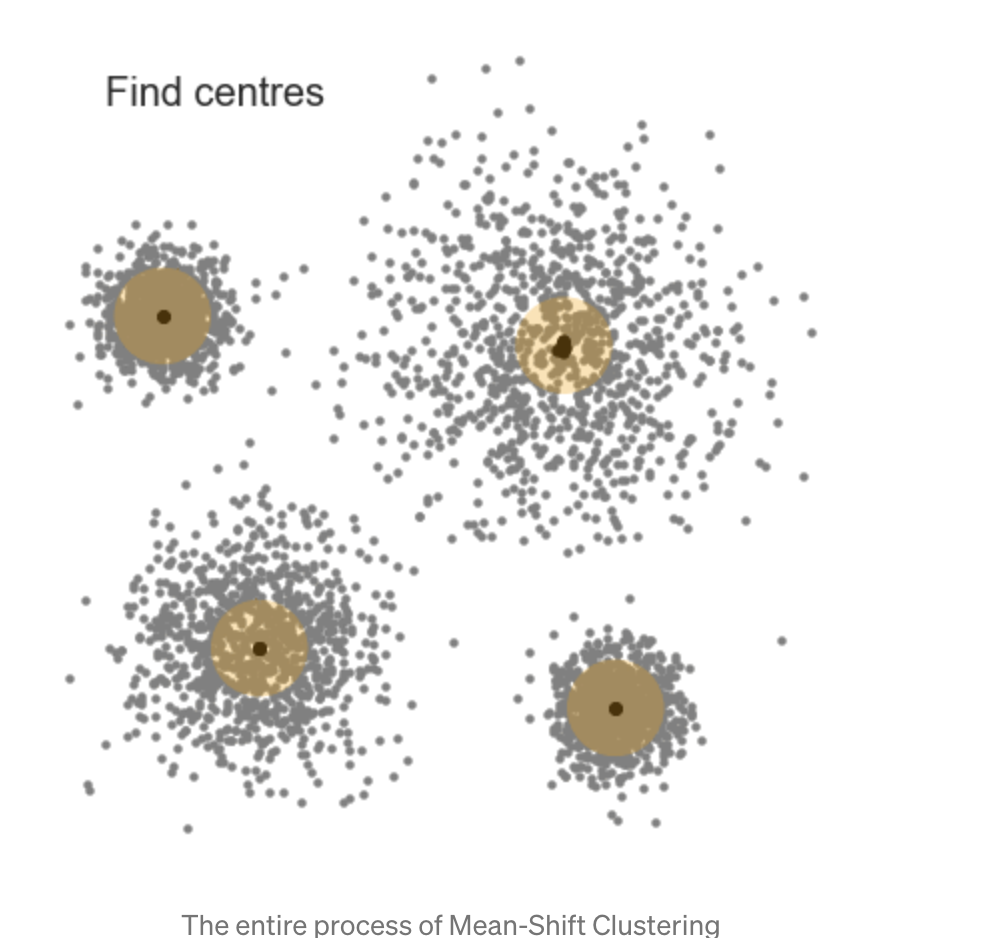
***Figure 4.3*** *- Using recalculated centers, continue shifting the centroids (black dots) and adding points to their respective clusters along the way. (Geif, 2018)*



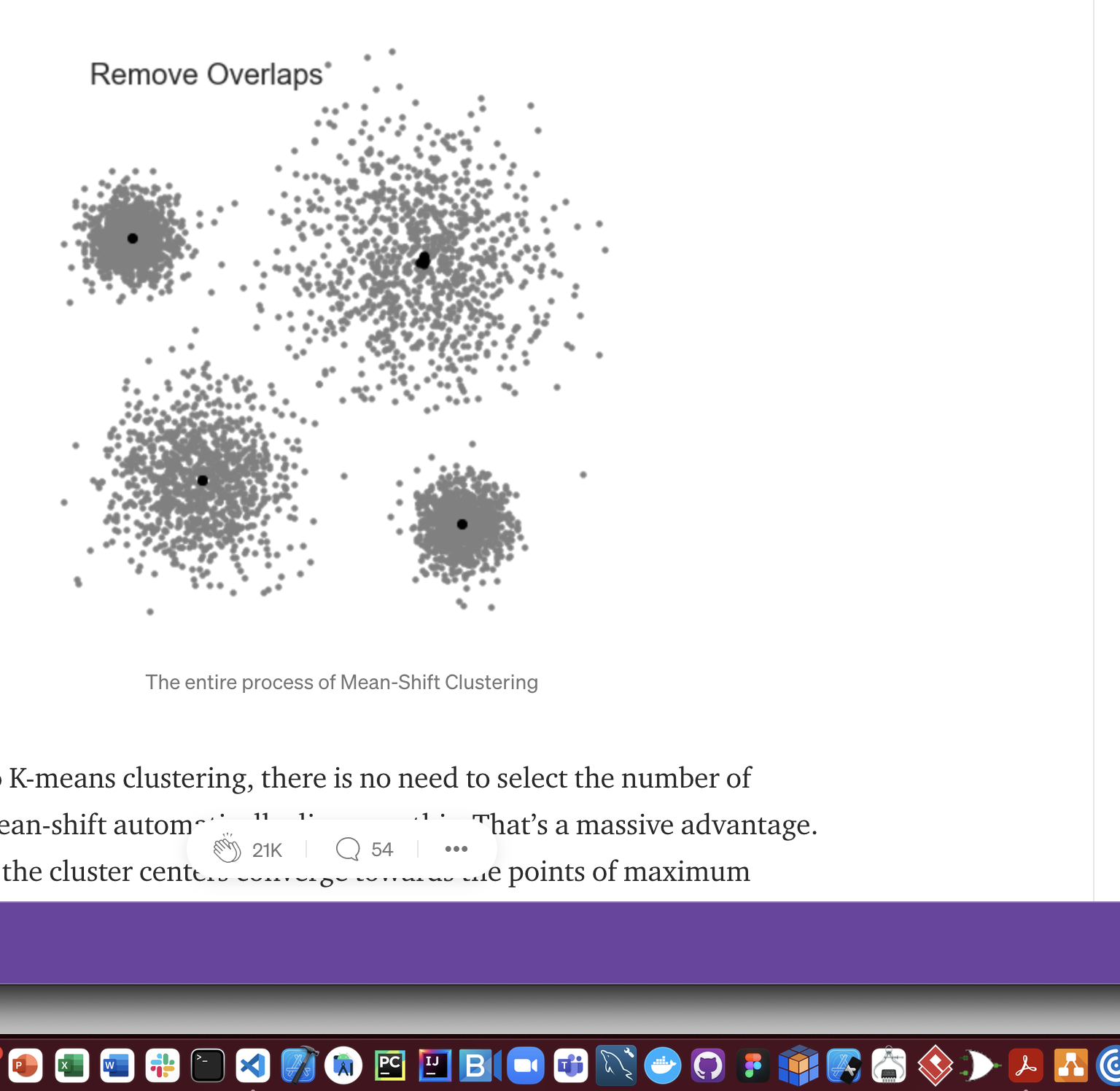
***Figure 4.4*** *- Recalculate the centers. (Geif, 2018)*



***Figure 4.5*** *- Centroids (black dots) converged. (Geif, 2018)*



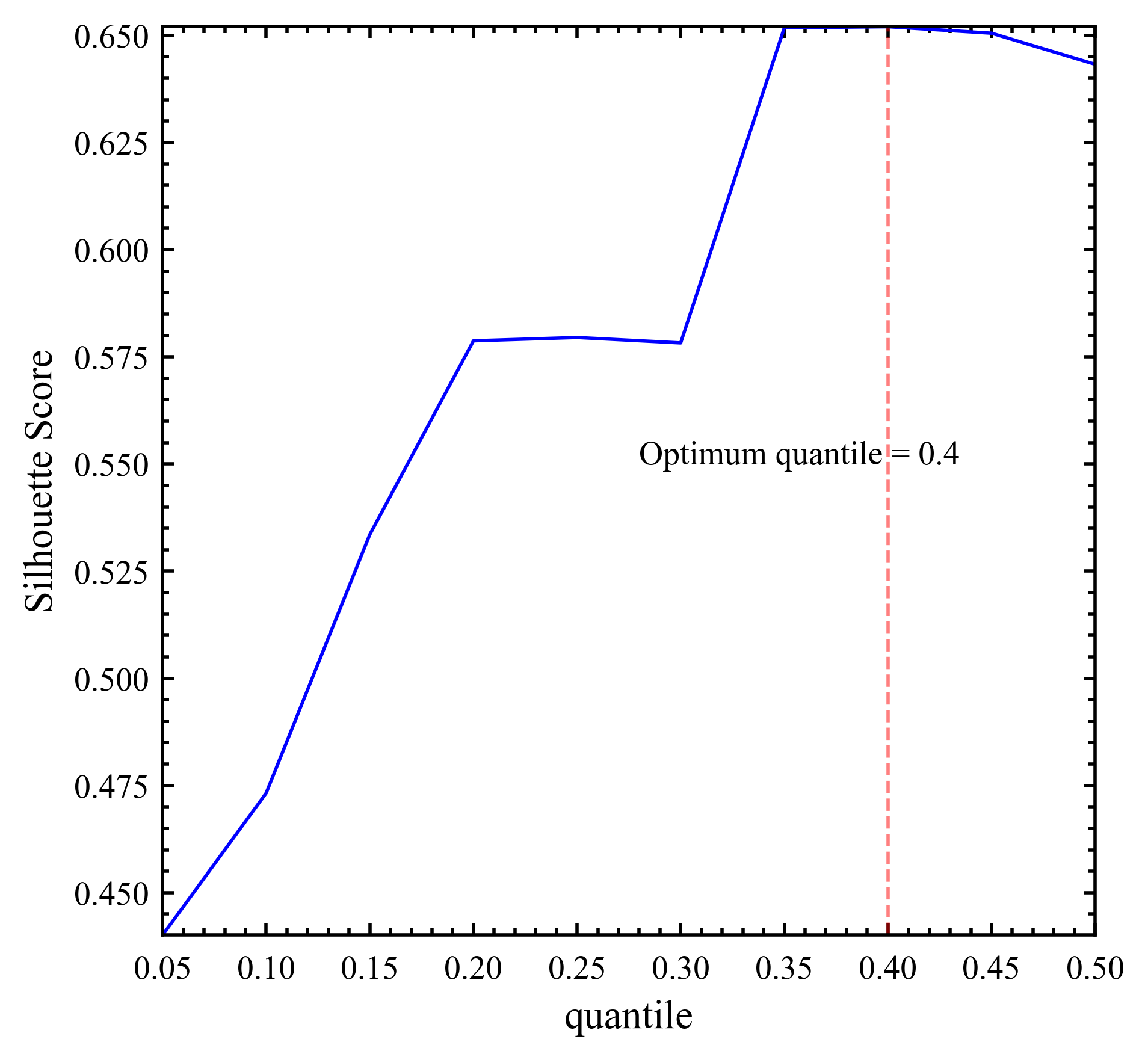
***Figure 4.6*** *- Locate the center of those centroids at each cluster. (Geif, 2018)*



***Figure 4.7*** *- Remove the smaller clusters overlapping the big clusters. (Geif, 2018)*

## How to choose radius/bandwidth:

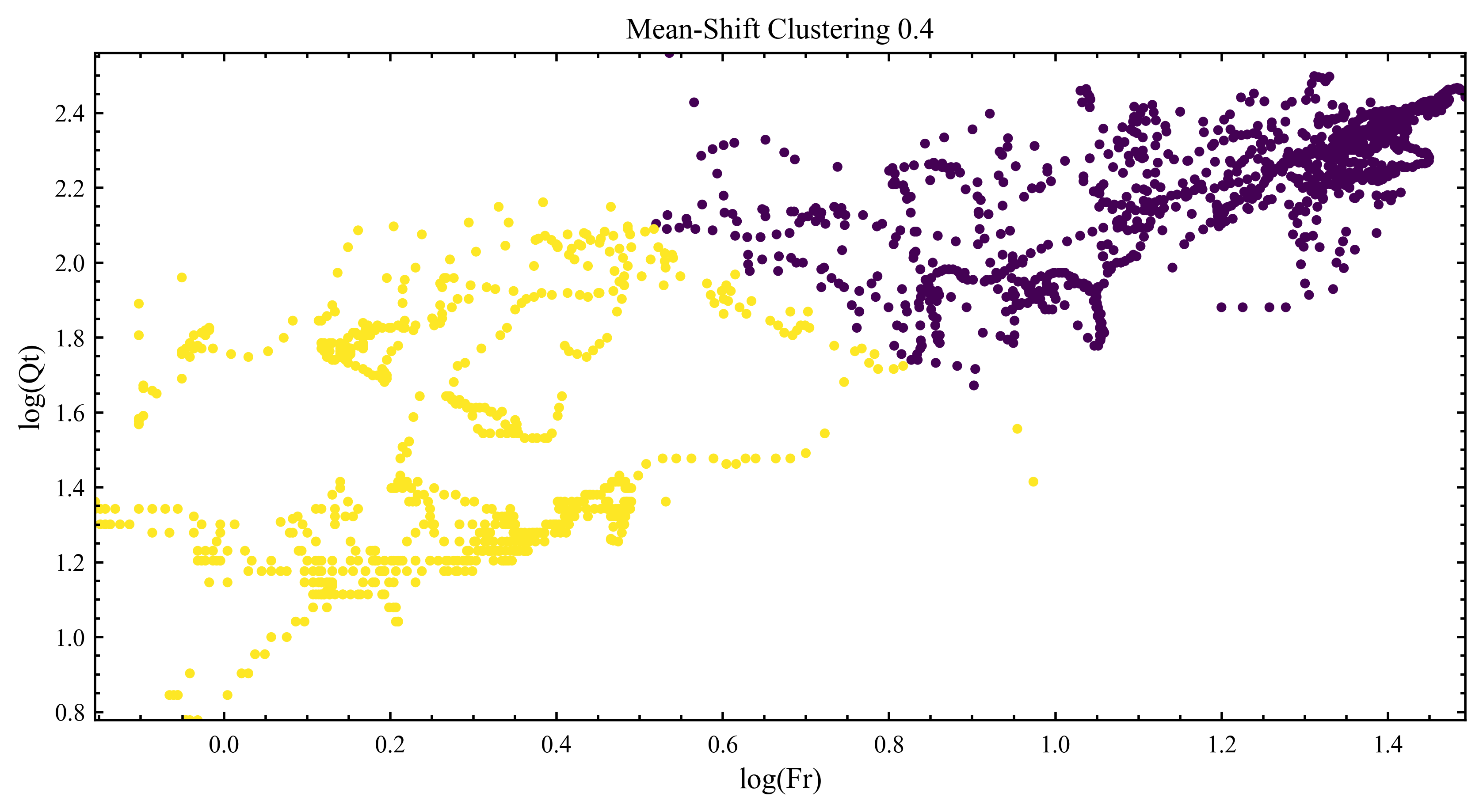
Just like we used the Elbow Method and Silhouette Score to estimate the optimal k clusters for K-Means, we used the Silhouette Score to find the optimal radius, or quantile. For more information on Silhouette Score, see earlier in this paper in the K-Means section, at ***Figure 2.2***.

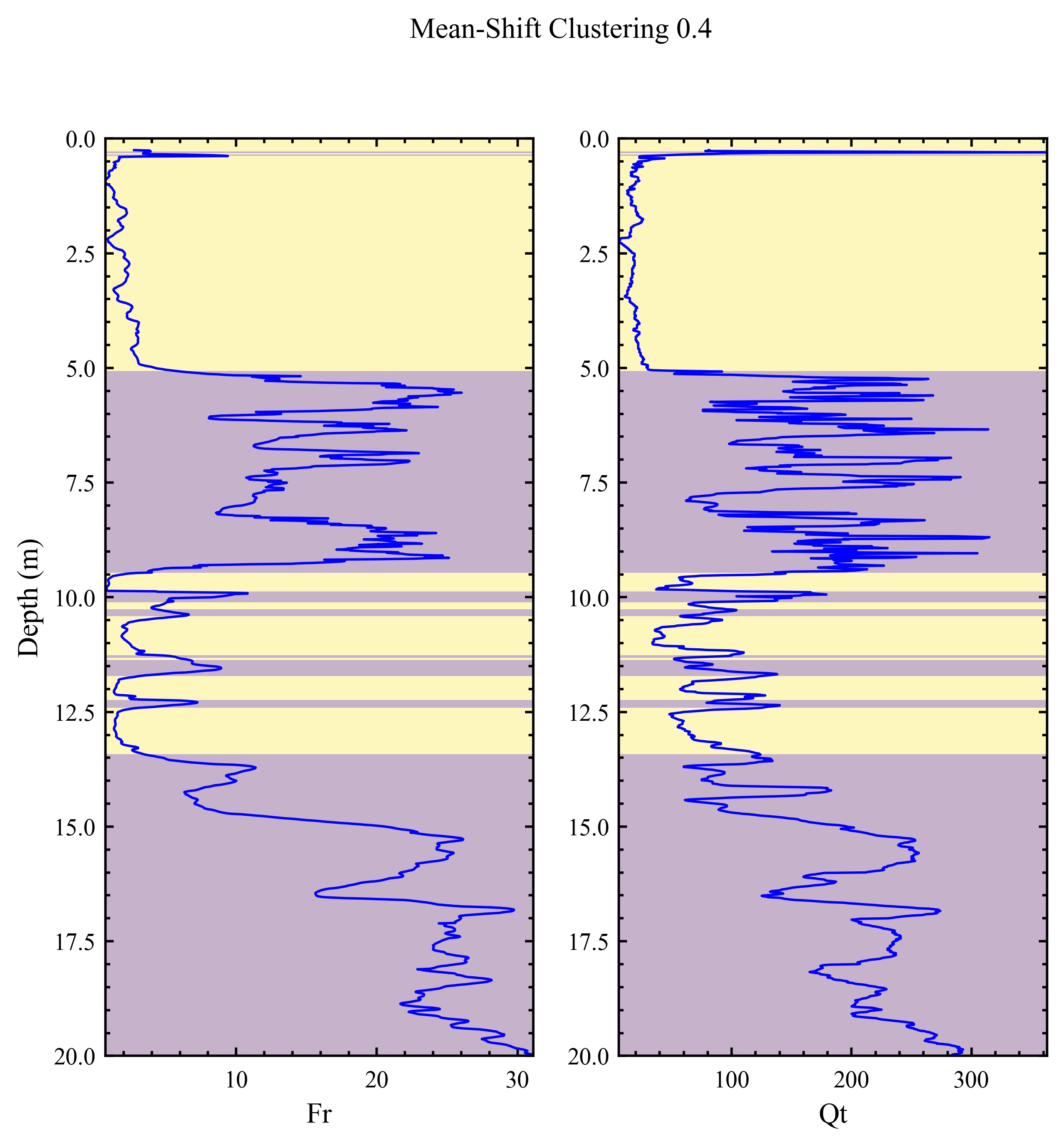


***Figure 5*** *– Silhouette Score, max at radius = 0.4*

## Results:

This algorithm, optimized with radius at 0.4, estimates the number of clusters on our dataset to be k = 2. Following are some of the results obtained for the mean-shift algorithm:

***Figure 6.1*** *- Clusters of the points from Mean-Shift Clustering, Radius = 0.4*



***Figure 6.2*** *- Clusters of the points from Mean-Shift Clustering, Radius = 0.4*

# DBSCAN: Density-Based Spatial Clustering of Applications with Noise

In the previous algorithms, clustering was down by either specifying the number of clusters or using the Radial-Based Function kernel. However, these make the algorithm’s performance entirely dependent on those parameters. Therefore, the algorithms end up classifying points entirely differently with small changes in either of the parameters. We need an algorithm that takes in those parameters, but in a more robust context. That is the context of either setting a lower bound number of points in a valid cluster, or setting an upper bound of the distance between points within a cluster.

Density-based spatial clustering of applications with noise (DBSCAN) fits this context. The general idea of DBSCAN is that it groups together points that are close to each other based on a distance measurement, usually Euclidean distance (Karim, 2020). As the name suggests, and similar to Mean Shift, DBSCAN makes sure to cluster the points such that it prioritizes the points with higher density. It also marks as outliers the points that are in low-density regions.

The parameters that DBSCAN takes are minPoints and eps. The definitions are below (Rhys, 2020):

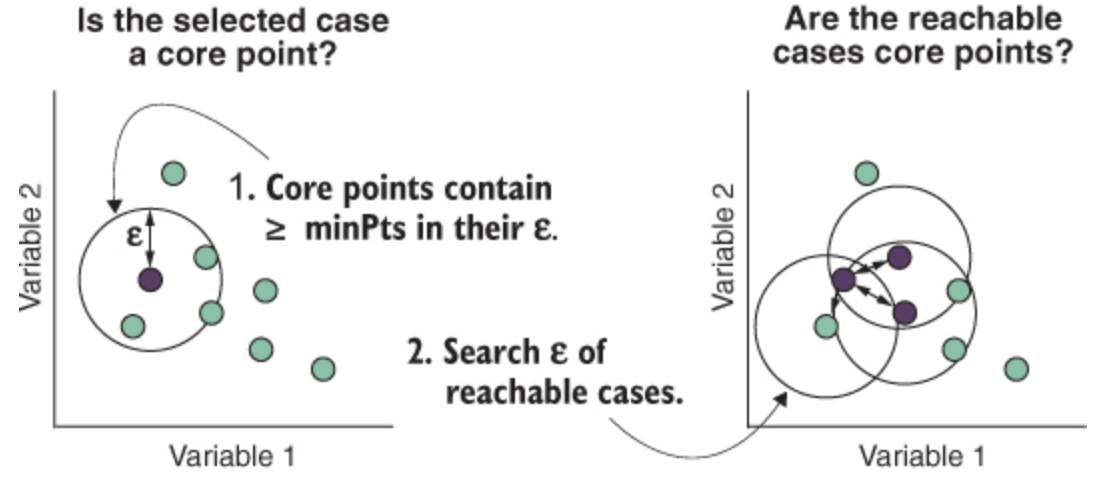
1. minPoints: The min number of points clustered together for a region to be considered dense.
2. eps (ε): A distance measure used to locate the points in the neighborhood of any point.

The minPoints parameter restricts clusters to those that are large enough to be recognizable clusters. This is important for datasets with large number of data points. Otherwise, smaller clusters, which may often include the noise in the model, greatly increases the complexity of the result and as a result gives details that are negligible in the context of the problem. The eps parameter restricts clusters from including points that may be better off in different clusters due to being closer to another group of points, resulting in clusters that may be too large. In other words, it may prevent the model from providing more insight, due to the smaller clusters included in that larger cluster. This is where eps comes in; with an appropriate eps value, it can restrict the size of the groups into clusters such that they are both recognizable and separable.

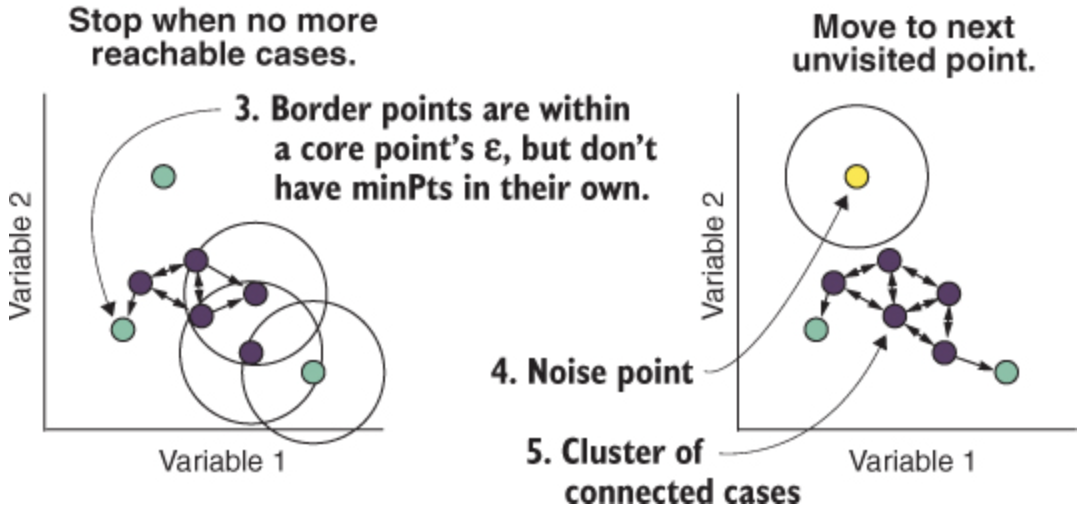
The algorithm works as follows (Seif, 2018):

1. The algorithm proceeds by arbitrarily picking up a point in the dataset. The neighborhood of this point is extracted using the eps value. Note that ‘eps’ is essentially a search radius.
2. If there are at least ‘minPoint’ points within the neighborhood of radius ‘eps = ε’ to the point, then we consider the point to be in a valid cluster. Otherwise, label it as noise for now (could be part of another cluster). As for the points within the neighborhood, add to the list of points to be explored later.
3. Repeat steps 1-2 for the next points within that neighborhood identified in the previous iteration. The new neighborhood is formed, the point is either determined to be in a valid cluster, or just noise, and points within that neighborhood not added yet to the exploration list are added. Keep doing this until there are no more points to explore from the list.
4. If there were indeed points added to a valid cluster from steps 1-2, then a valid cluster has been formed. Else, go on to the next point and start over.
5. Repeat Steps 1-4 until all points in the dataset have been explored and clustered/marked as noise.

The visuals that outline these steps are below:



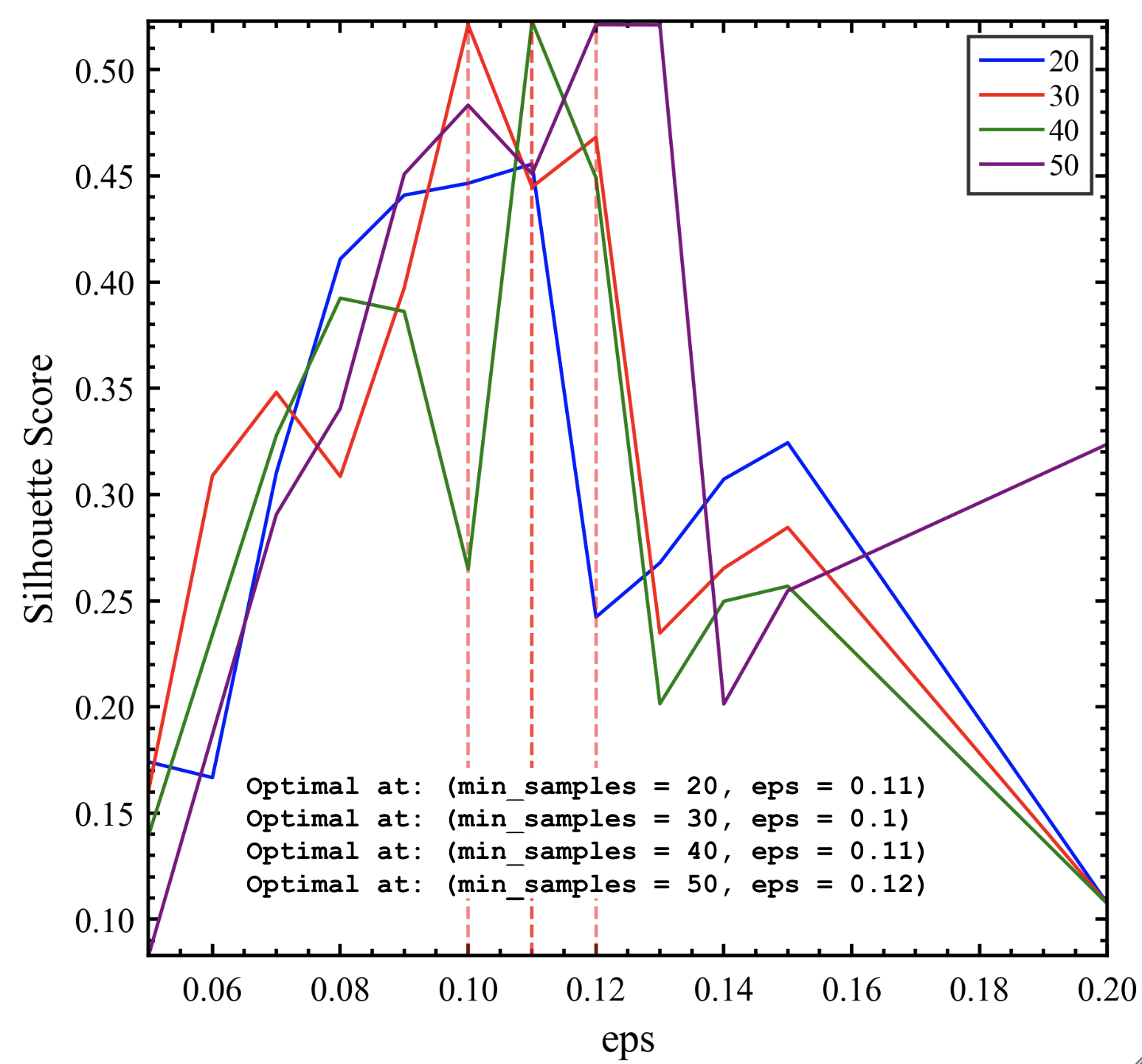
***Figure 7.1 -*** *Steps 1-2 (Chapter 18 clustering based, 2021)*



***Figure 7.2*** *- Step 3-5 (Chapter 18 clustering based, 2021)*

## Choosing max epsilon and minimum points per cluster:

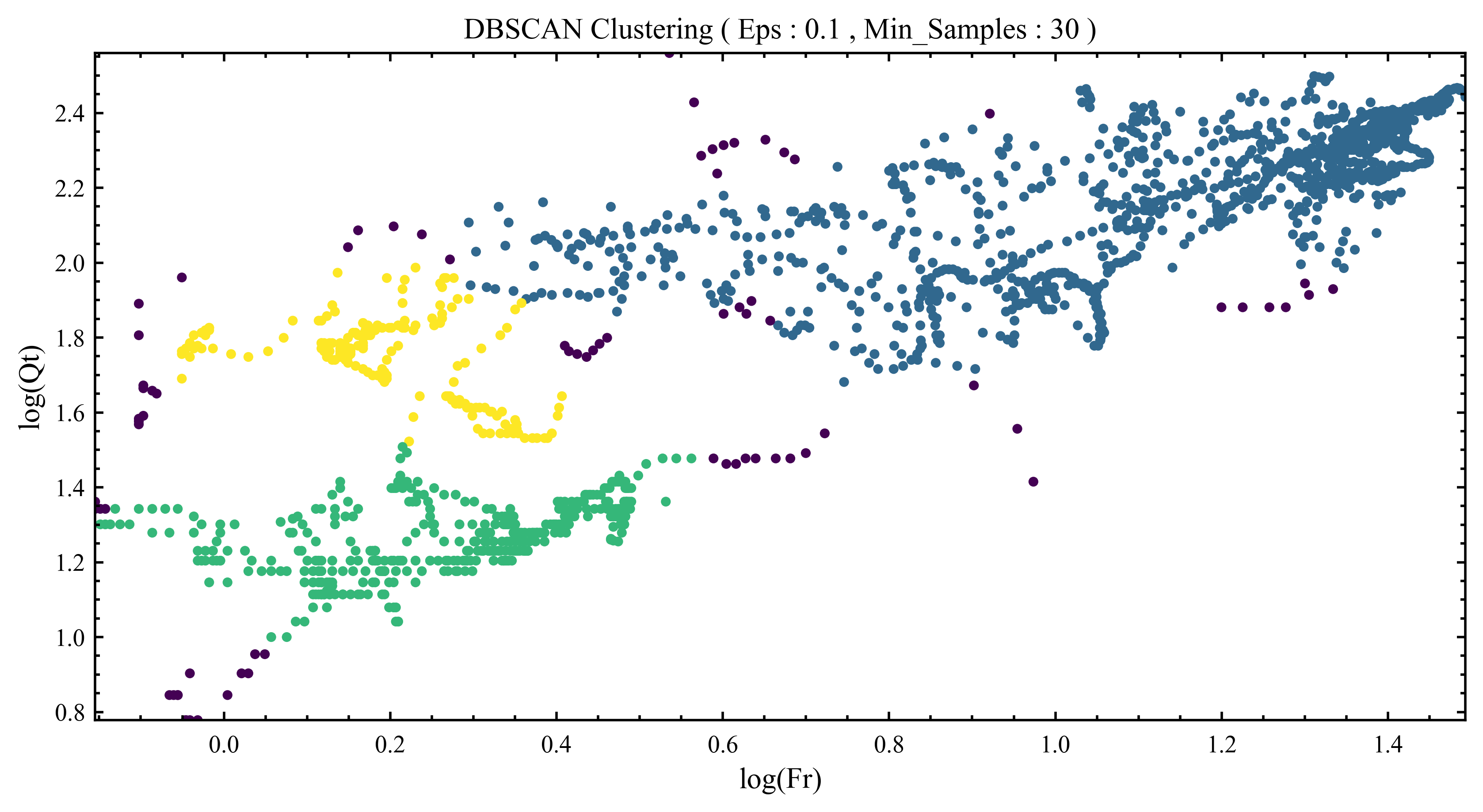
To narrow down the values used for epsilon and minPoints for each valid cluster, we used the Silhouette Score, which measures how well the algorithm performs at separating the data so that the clusters are more separated. The algorithm to calculate it is to iterate over the epsilon value and the minimum sample points. Determine the maximum score, and the eps value for each minPoints value. Based on these measures, our eps and minimum points are eps = 0.12 and minPoints = 50.



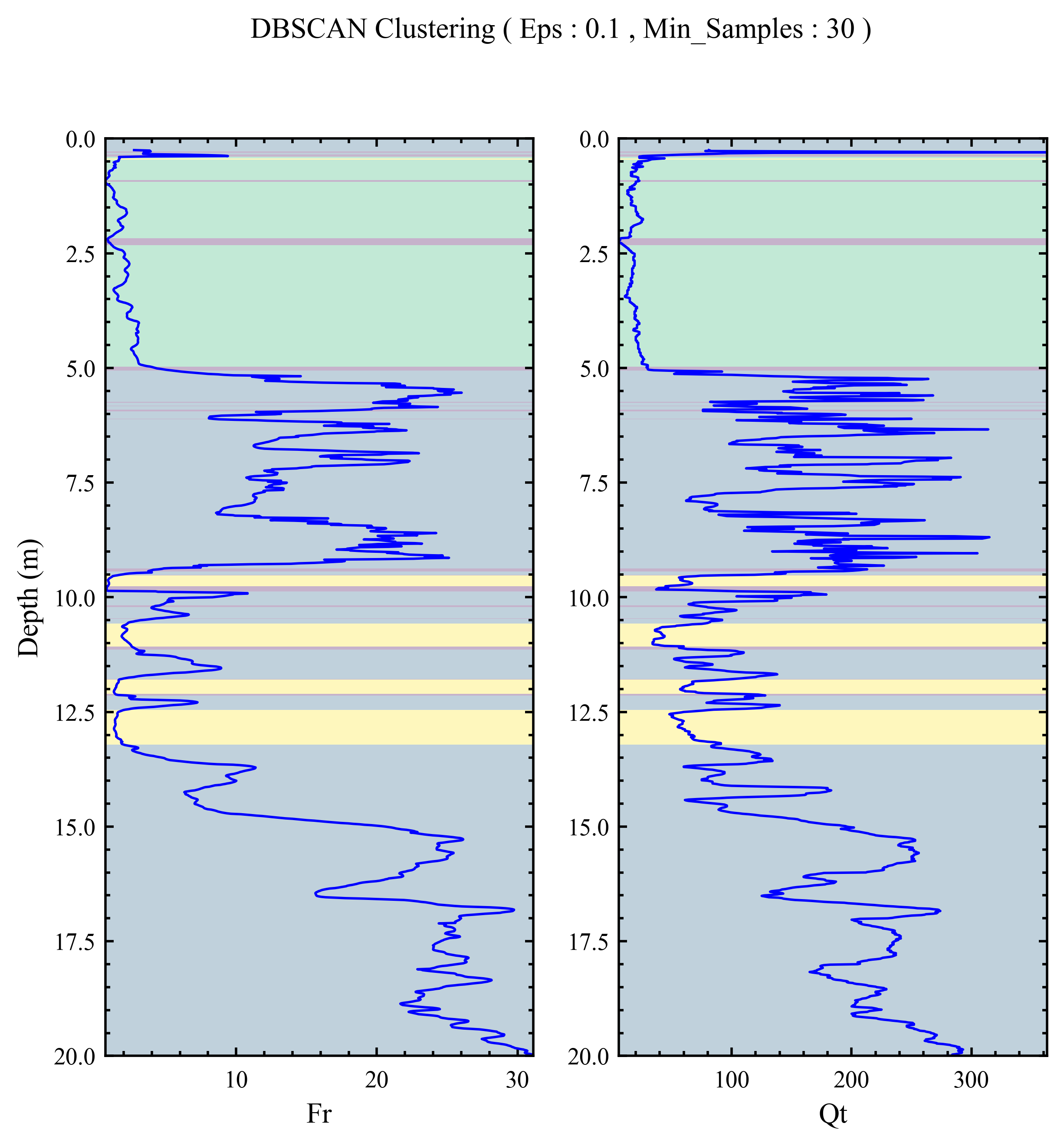
***Figure 8*** *– Choosing the best epsilon and minpoints based on Silhouette Score*

## Results:

This algorithm estimates the number of clusters on our dataset to be k = 3. Following are some of the results obtained for the OPTICS algorithm:



***Figure 9.1*** *- Clusters resulting from DBSCAN*



***Figure 9.2*** *- Layering resulting from DBSCAN*

# OPTICS:

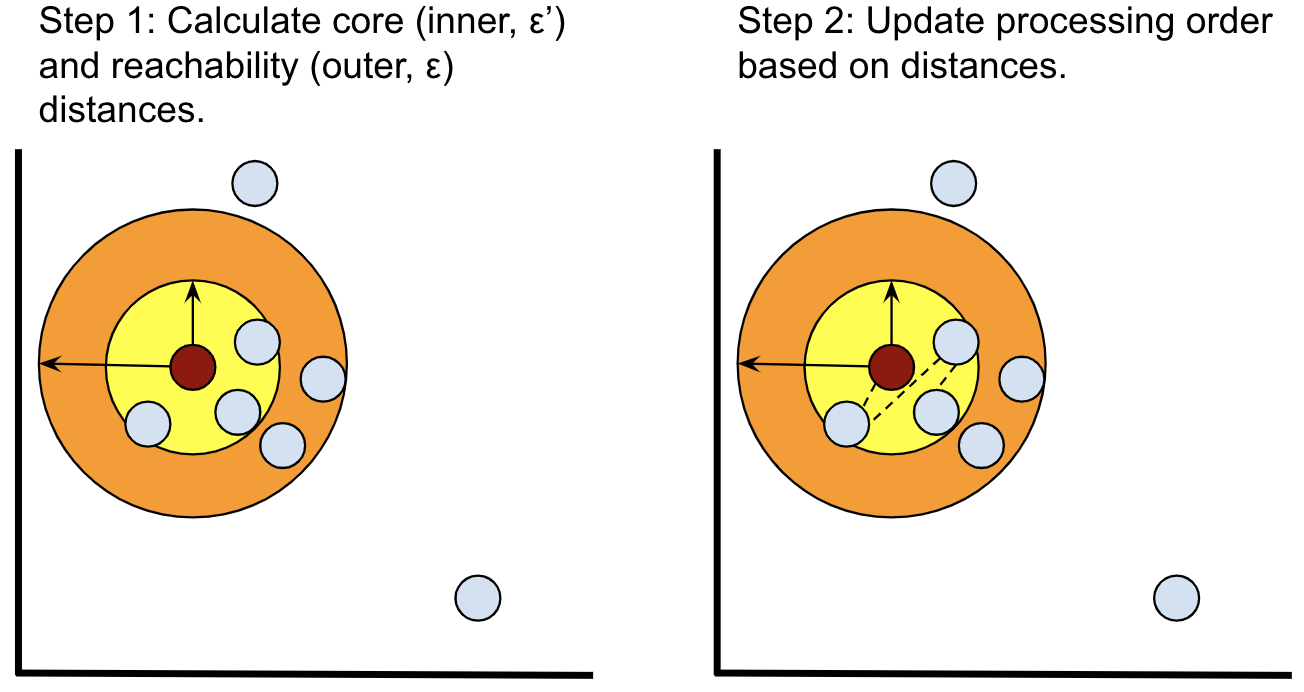
The OPTICS algorithm is similar to the DBSCAN algorithm in that it separates regions with high density from regions with low density (Rhys, 2020). However, the difference is that it does it by considering just the minimum number of points inside a valid cluster (Karim, 2020). The epsilon is taken care of as the algorithm calculates the core distance, which is the smallest radius where at least minPoints number of points are reachable. Therefore, OPTICS takes less parameters and performs more computation in comparison to DBSCAN, allowing for more flexibility.

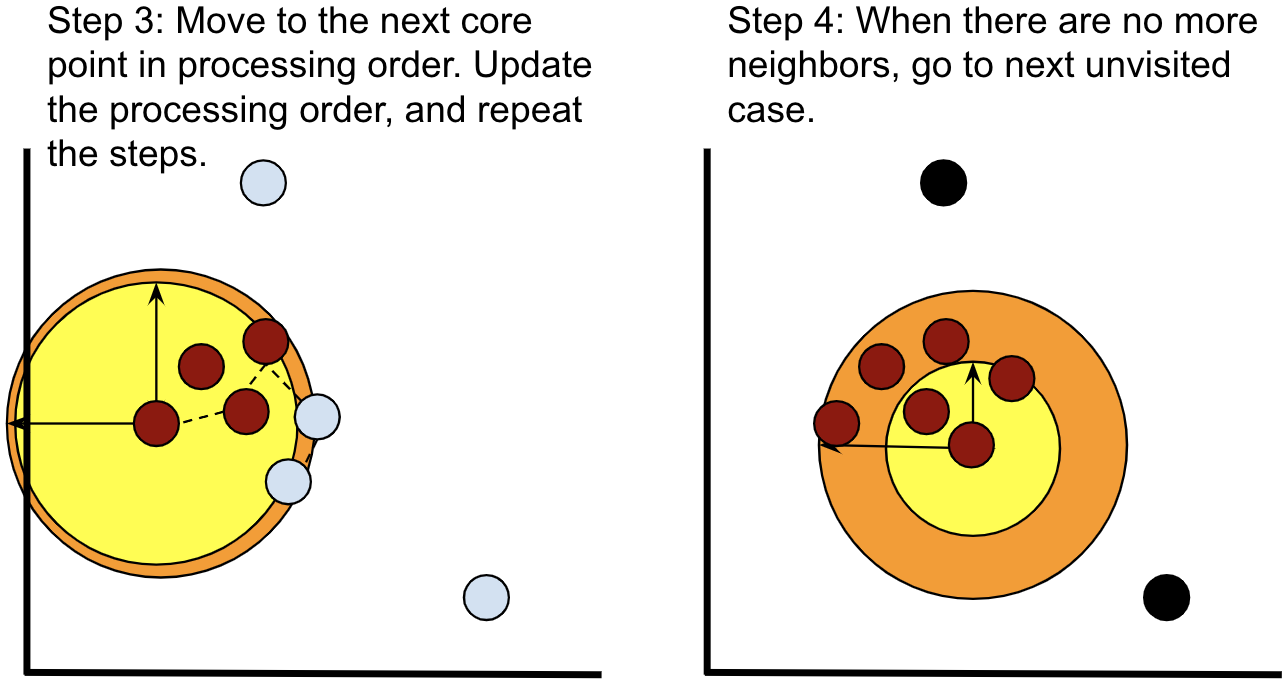
The minPoints value, the sole mandatory parameter, does not change. However, for eps, there are multiple ways to go about it. First, eps can be computed by calculating the core distance, which is partially dependent on minPoints. Core distance is what determines how dense the cluster is. Second, you have the option to set a maxEps, which is a max search radius that the core distance can be calculated up to. The second option is usually better computational-wise: it does not force the algorithm to calculate a core distance at each point it explores, especially if the point is farther away from most other points in the dataset.

The algorithm is as follows (Rhys, 2020):

1. Select a data point.
2. Calculate the distances from that point to its nearest points and generate a processing order of reachable (within a search radius) data points. If there are minPoints number of these reachable data points, they are in the same valid cluster.
3. Repeat for each data point that is reachable, until there are no more reachable points. Mark those points as 1 cluster and move on to the next data points.
4. Pick another data point that has not been visited, then perform the same operations from steps 2 and 3.
5. Repeat until all data points have been visited and clustered.

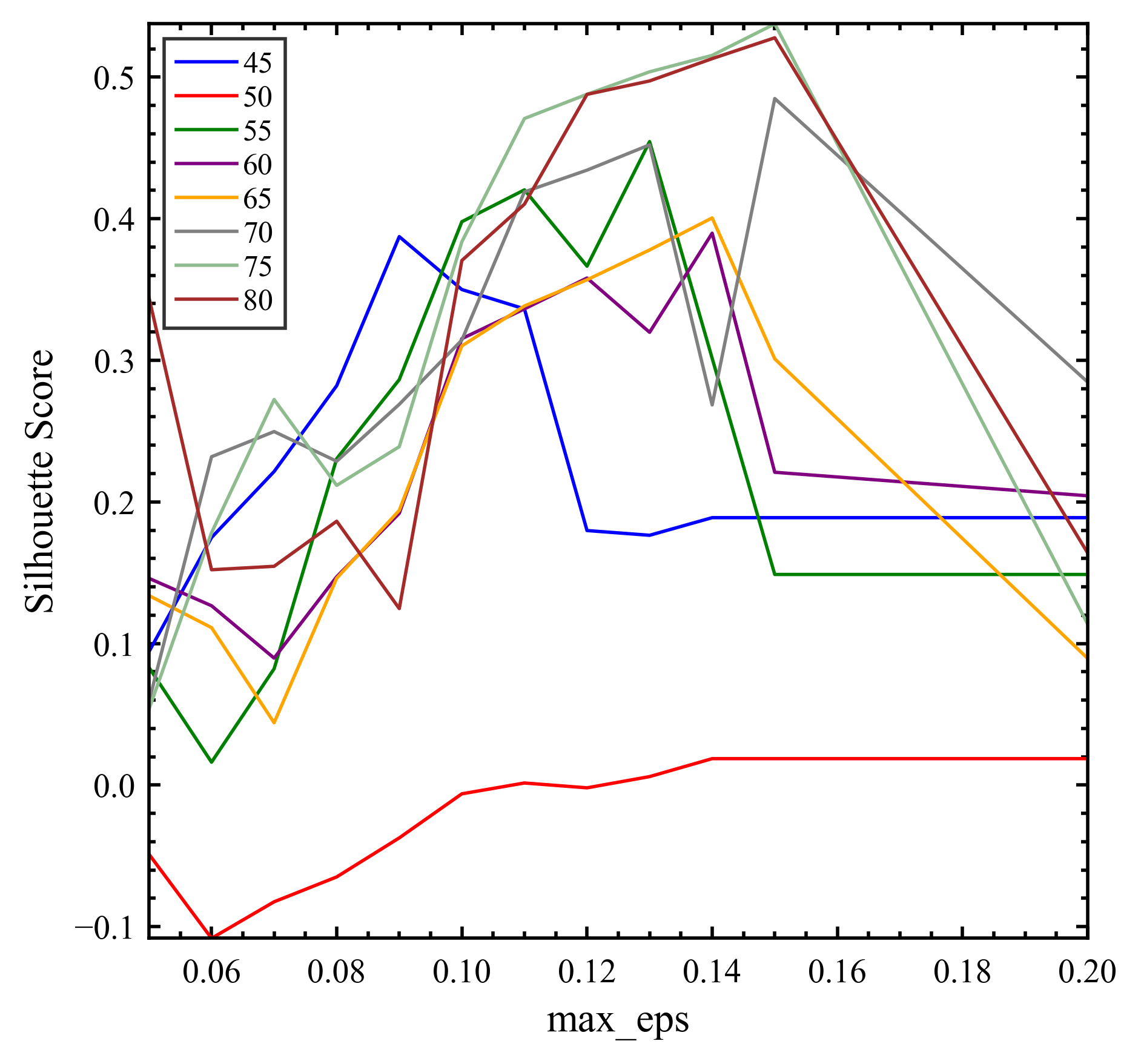
Visuals that outline these steps are below:

***Figures 10.1, 10.2*** *- (Left) Calculate core and max\_eps at dark brown core point. (Right) Update order of points to explore based on distance from current core point. (Rhys, 2020)*

***Figures 10.3, 10.4*** *- (Left) Keep going through the iterations, expanding the cluster. (Right) Move on to next point/new cluster once all points in neighborhood are accounted for. (Rhys, 2020)*

## Choosing max epsilon and minimum points per cluster:

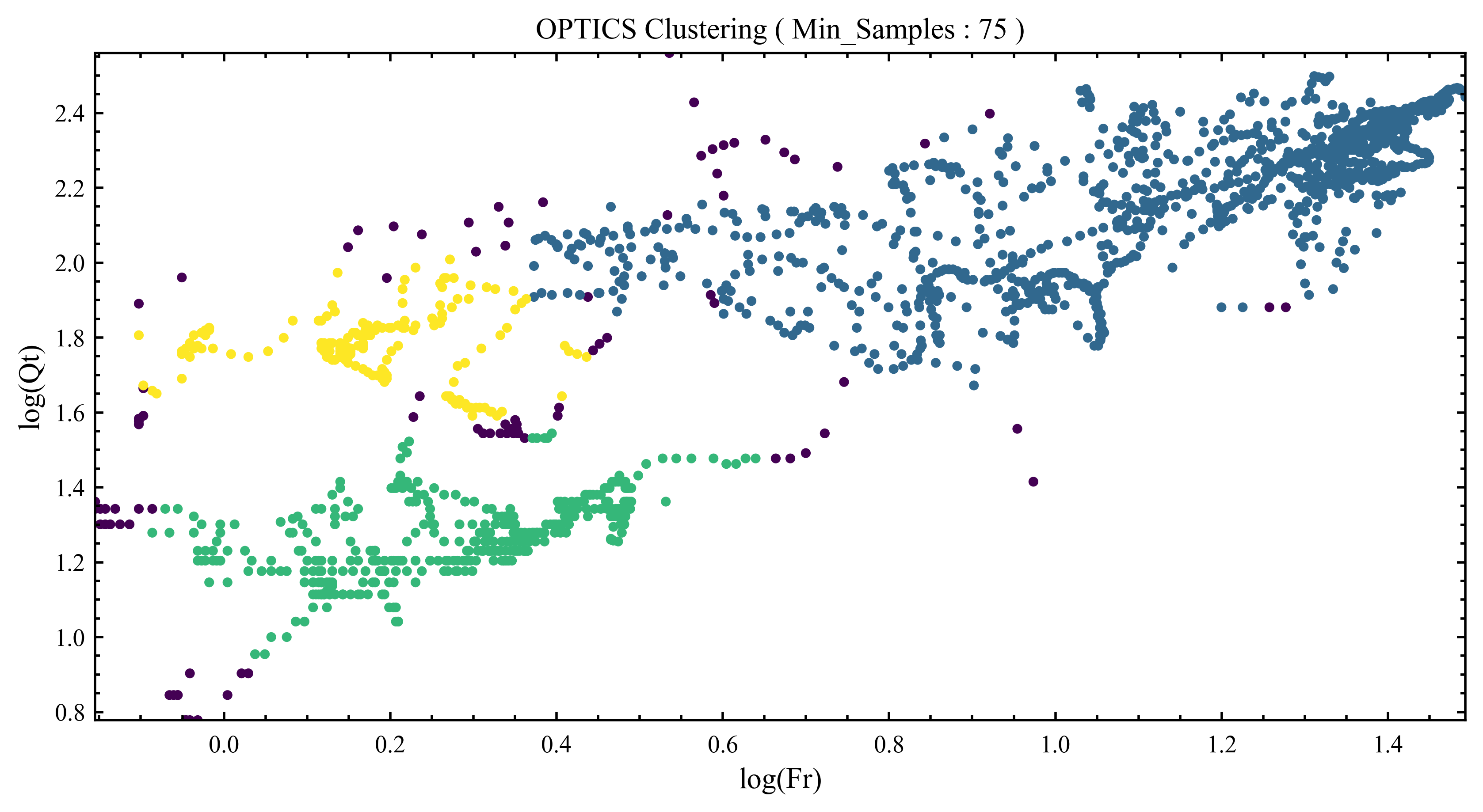
Similar to the method applied to DBSCAN algorithm [2], to narrow down which values to use for max epsilon and minimum points per valid cluster, we used the Silhouette Score, which measures how well the algorithm performs on separating the data such that there is more separation distance between clusters. The algorithm to compute that was to loop through the max epsilon values and minimum sample points. The max score is determined, as well as the max\_eps value for each value of minimum points. Based on these measures, our max\_eps and minimum points are max\_eps = 0.15 and minPoints = 75.

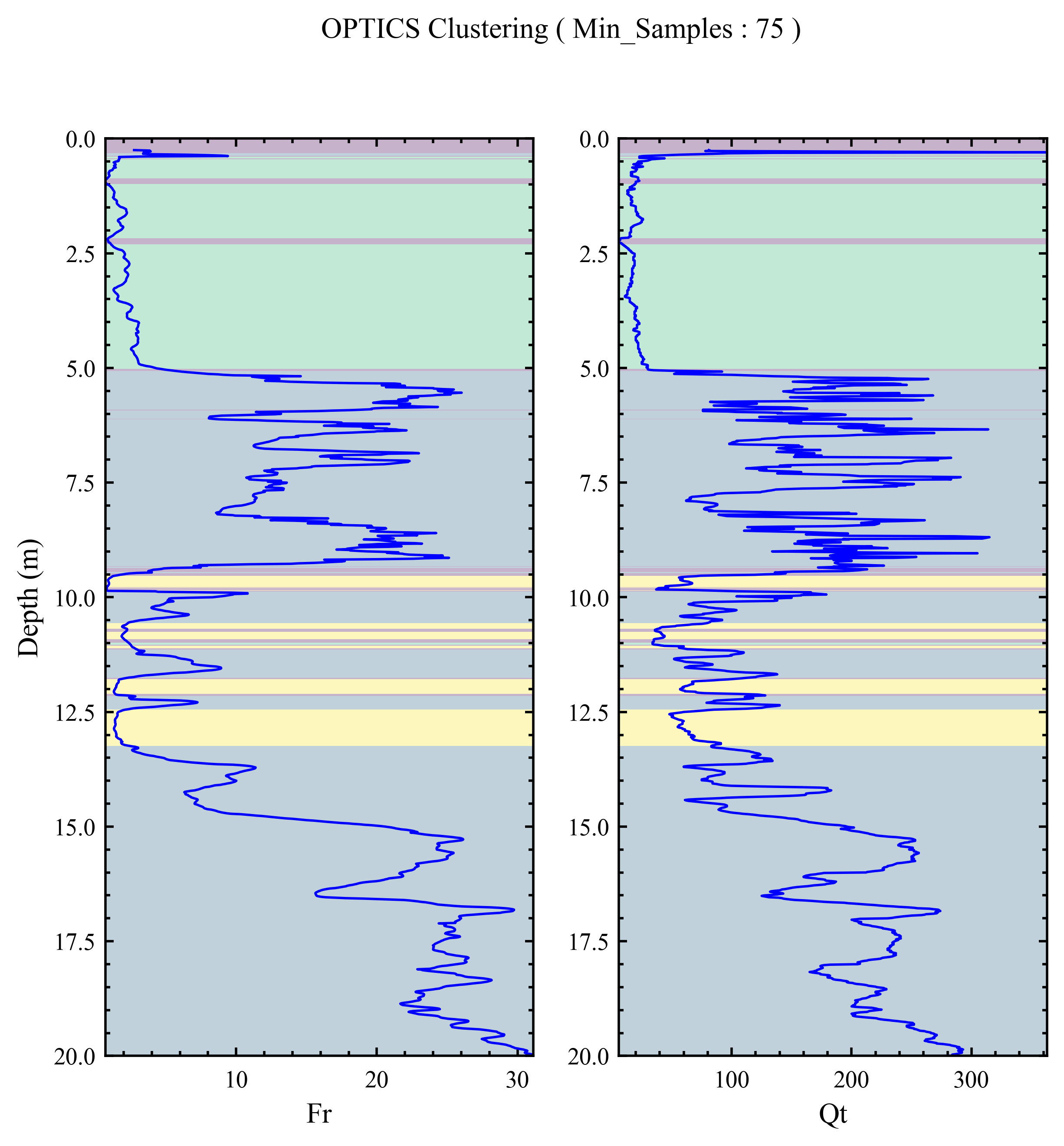


***Figure 11*** *– Silhouette Score for OPTICS, picking the best max\_eps and minPoints*

## Results:

This algorithm estimates the number of clusters on our dataset to be k = 3. Following are some of the results obtained for the OPTICS algorithm:

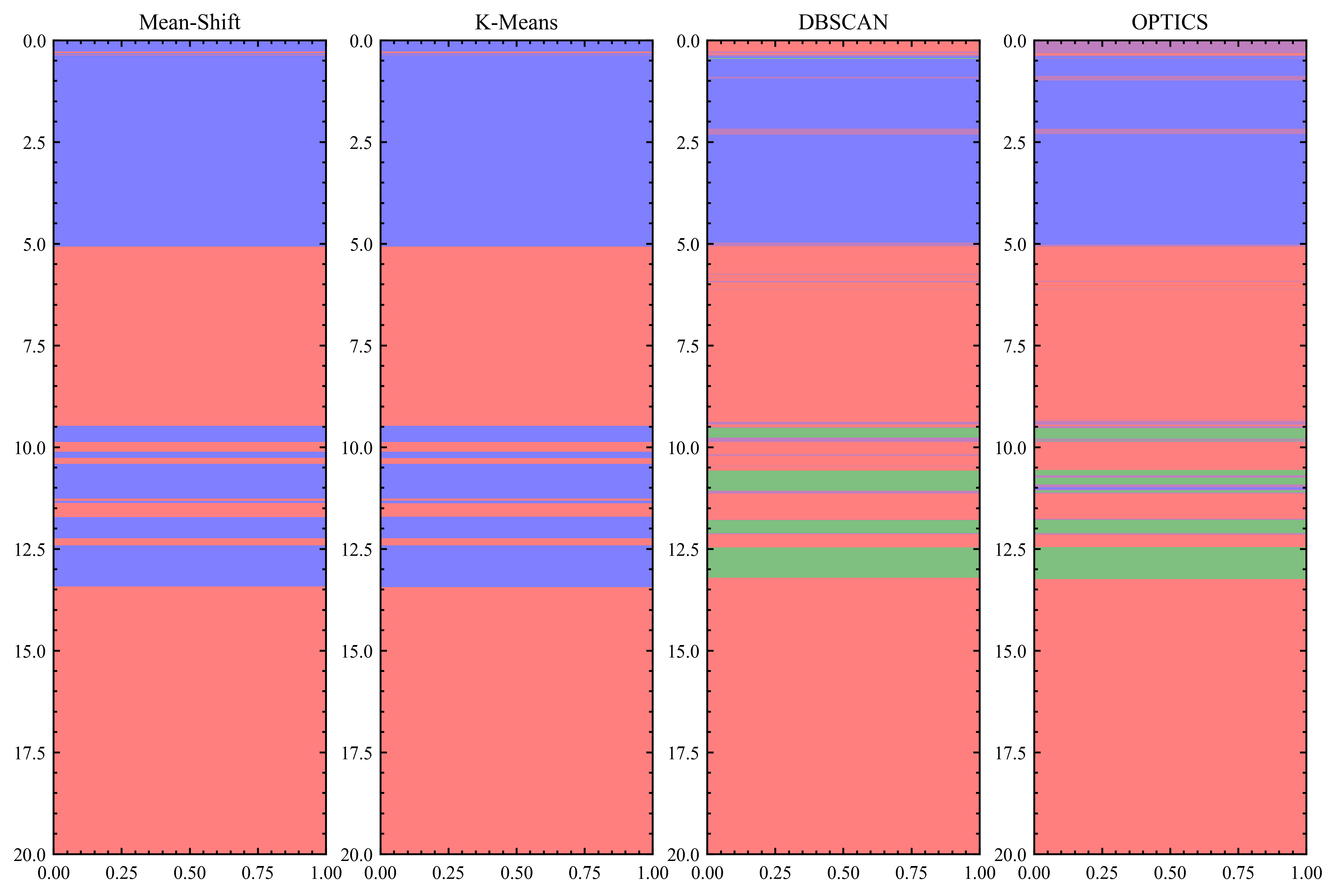
***Figure 12.1*** *- Clusters resulting from OPTICS*



***Figure 12.2*** *- Layering resulting from OPTICS*

This turned out to be a lot different because of the emphasis on the different densities of the data points. Those densities mean that at some depths, they tend to be more dense or less dense, which provides more information on the friction and tip resistance at these depths. Unlike DBSCAN, this algorithm provides more confident insight on the geological properties at those data points.

## Comparison:

***Figure 13*** *– Comparison Chart*

Overall, the algorithms consistently got the data points clustered into 2-3 groups, and those themselves are consistent with the depth, tip resistance, and friction resistance features of the dataset. As a result, we can conclude that with this ensemble of clustering algorithms from K-Means to Density-Based Clustering with DBSCAN and OPTICS, it sheds some light into the groups of different densities and at what range of depths are those groups at, each providing insight into the soil behavior at different depths.

References

[1] Ashour, W. M., Mezied, I. A., & Abu-Issa, A. S. (2018).

Improving Density-based Clustering using Metric Optimization. International Journal of Computer Applications, 181(21).

Chapter 18 clustering based on density: DBSCAN and OPTICS - machine learning with R, the Tidyverse, and MLR. (2021).

https://dev2u.net/2021/06/03/chapter-18-clustering-based-on-density-dbscan-and-optics-machine-learning-with-r-the-tidyverse-and-mlr/

[2] DBSCAN: density-based clustering for discovering clusters in large datasets with noise - Unsupervised Machine Learning. (n.d.). STHDA. Retrieved April 4, 2018,

from <http://www.sthda.com/english/wiki/wiki.php?id_contents=7940>

[3] Karim MD, M. R., Beyan, O., Zappa, A., Costa, I. G., Rebholz-Schuhmann, D., Cochez, M., & Decker, S. (2020, February 1).

Deep learning-based clustering approaches for bioinformatics (Briefings in Bioinformatics 22; Issue Brief No. 1). Oxford Academic. <https://doi.org/10.1093/bib/bbz170>

[4] Lavrenko, V. (2015, September). Clustering 6: The k-means algorithm visually [Video]. YouTube.

https://www.youtube.com/watch?v=IJt62uaZR-M

[5] Mean Shift Intro - Practical Machine Learning Tutorial with Python p.39 [Video]. (2016, June 29).

YouTube. https://www.youtube.com/watch?v=3ERPpzrDkVg

[6] Rhys, H. I. (2020). Chapter 18. Clustering based on density: DBSCAN and OPTICS.

In Machine Learning with R, the tidyverse, and mlr.

[7] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

[8] Seif, G. (2018, February 5). The 5 Clustering Algorithms Data Scientists Need to Know.

Retrieved April 4, 2022, from <https://towardsdatascience.com/the-5-clustering-algorithms-data-scientists-need-to-know-a36d136ef68>

[9] Sinclair, C. (2019, January 1). Clustering Using OPTICS. Towards Data Science. Retrieved April 4, 2022,

from <https://towardsdatascience.com/clustering-using-optics-cac1d10ed7a7>