Author: Furkan Salman

Data Mining Project - Final Report

Problem Statement

- The goal of this project is to implement the DBSCAN algorithm for data clustering and explore its effectiveness in various scenarios.
- The Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm is a popular approach for data clustering due to its ability to identify clusters of arbitrary shape and handle noise efficiently. Unlike traditional centroid-based clustering algorithms, such as K-means, DBSCAN defines clusters based on the density of data points in the feature space, making it suitable for datasets with irregular or overlapping clusters.
- The project aims to address the following key challenges:

a. Unsupervised Clustering:

- The task involves clustering data without any prior knowledge or labeled examples. The algorithm should be able to identify meaningful clusters solely based on the inherent patterns and structures present in the data.

b. Parameter Selection:

- The DBSCAN algorithm requires two key parameters: epsilon (ε) and minimum points (MinPts). Determining appropriate values for these parameters is essential for achieving accurate clustering results. The project will investigate methods for parameter selection and evaluate their impact on the clustering performance.

c. Evaluation and Analysis:

- The project will focus on evaluating the quality and performance of the implemented DBSCAN algorithm. This includes selecting appropriate evaluation metrics, such as silhouette coefficient or Davies-Bouldin index, to measure the clustering effectiveness. Additionally, visualizations and analysis of the clustering results will be performed to gain insights into the characteristics and structure of the data.
- To ensure the success of the project, a systematic methodology will be followed, including the following steps: understanding the algorithm's theory, implementing the algorithm, selecting and preparing datasets, conducting experiments, evaluating and analyzing the results, and drawing meaningful conclusions.

d. Organization of the Report

- This report is organized into several sections, each focusing on different aspects of the project. The subsequent sections will cover the literature review, methodology, implementation details, experimental results, evaluation metrics, visualizations, analysis, and conclusions.

- The report aims to provide a comprehensive overview of the DBSCAN algorithm, its implementation, and its performance on different datasets. It will serve as a valuable resource for understanding and utilizing DBSCAN for data clustering purposes.
- By addressing the problem statement, following the proposed methodology, and conducting rigorous experiments, this project strives to contribute to the advancement of data clustering techniques and provide insights into the applicability of the DBSCAN algorithm in real-world scenarios.

Proposed Solution:

The proposed solution for implementing the DBSCAN algorithm for data clustering involves the following key steps:

a. Data Preprocessing:

- To prepare the input data for clustering, we handled missing values, normalized features, and removed any outliers or noise that might negatively impact the clustering results. An appropriate distance metric, such as Euclidean distance is used to measure the similarity between data points.

b. Density-Based Spatial Clustering:

- We implemented the core DBSCAN algorithm to perform density-based clustering. The algorithm requires two parameters: epsilon (ε) and minimum points (MinPts). It identifies core points, which have at least MinPts neighboring points within a distance of ε, and expands clusters by connecting core points and their reachable neighbors. Any points that are not core points or reachable from any core point are considered noise points.

c. Cluster Evaluation:

- Common evaluation metrics such as silhouette coefficient, Davies-Bouldin index has been considered during evaluation of performance in regards to different perspectives.

d. Visualization:

- Techniques will be implemented to visualize the clustering results. Dimensionality reduction methods like Principal Component Analysis (PCA) has been used to project the high-dimensional data into a lower-dimensional space. Data visualization techniques such as scatter plots for clusters and heatmaps have been employed to visualize the clusters and their characteristics.

e. Parameter Optimization:

- Elbow method has been used to determine the essential parameter eps for DBSCAN algorithm.

DBSCAN Algorithm Overview:

The Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm is a density-based clustering algorithm that is widely used for grouping data points based on their density. DBSCAN operates on the principle that clusters are regions of high-density separated by regions of low-density.

The core concepts of the DBSCAN algorithm are as follows:

a. Density Reachability:

- A data point A is said to be density-reachable from another data point B if there exists a chain of data points, starting from B and moving through direct density-connected points, leading to A. Density reachability is used to determine whether a data point belongs to a cluster or is an outlier.

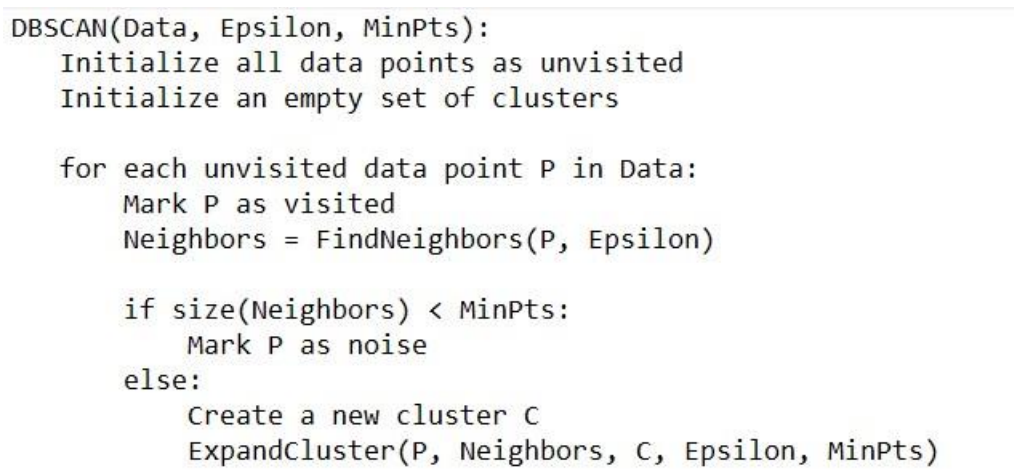
b. Density Connectivity:

- Two data points A and B are said to be density-connected if there exists a data point C that is density-reachable from both A and B. Density connectivity is used to form clusters of data points.

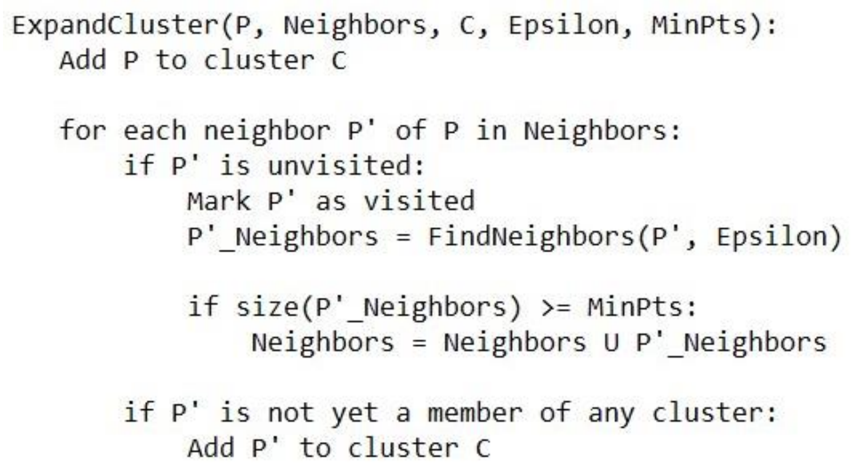
c. Epsilon (ε) and Minimum Points (MinPts):

- The DBSCAN algorithm requires two parameters: epsilon (ε) and minimum points (MinPts). Epsilon defines the radius within which to search for neighboring data points of a given point. MinPts specifies the minimum number of neighboring points that must be within the epsilon radius for a point to be considered a core point.

The pseudocode for the DBSCAN algorithm is as follows:



```
DBSCAN(Data, Epsilon, MinPts):
   Initialize all data points as unvisited
   Initialize an empty set of clusters

   for each unvisited data point P in Data:
       Mark P as visited
       Neighbors = FindNeighbors(P, Epsilon)

       if size(Neighbors) < MinPts:
           Mark P as noise
       else:
           Create a new cluster C
           ExpandCluster(P, Neighbors, C, Epsilon, MinPts)
```

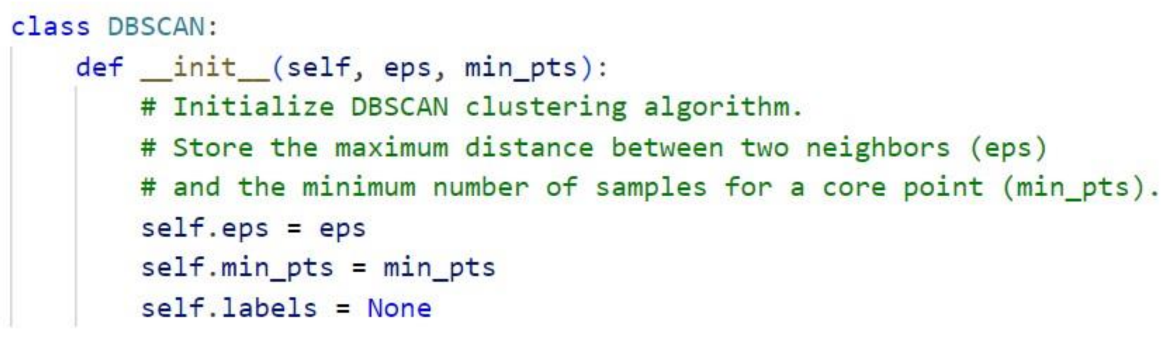


```
ExpandCluster(P, Neighbors, C, Epsilon, MinPts):
    Add P to cluster C

    for each neighbor P' of P in Neighbors:
        if P' is unvisited:
            Mark P' as visited
            P'_Neighbors = FindNeighbors(P', Epsilon)

            if size(P'_Neighbors) >= MinPts:
                Neighbors = Neighbors U P'_Neighbors

        if P' is not yet a member of any cluster:
            Add P' to cluster C
```

This pseudocode outlines the basic steps of the DBSCAN algorithm, including the initialization, finding neighbors, expanding clusters, and marking noise points. These steps form the foundation of the DBSCAN algorithm's implementation.

DBSCAN Implementation

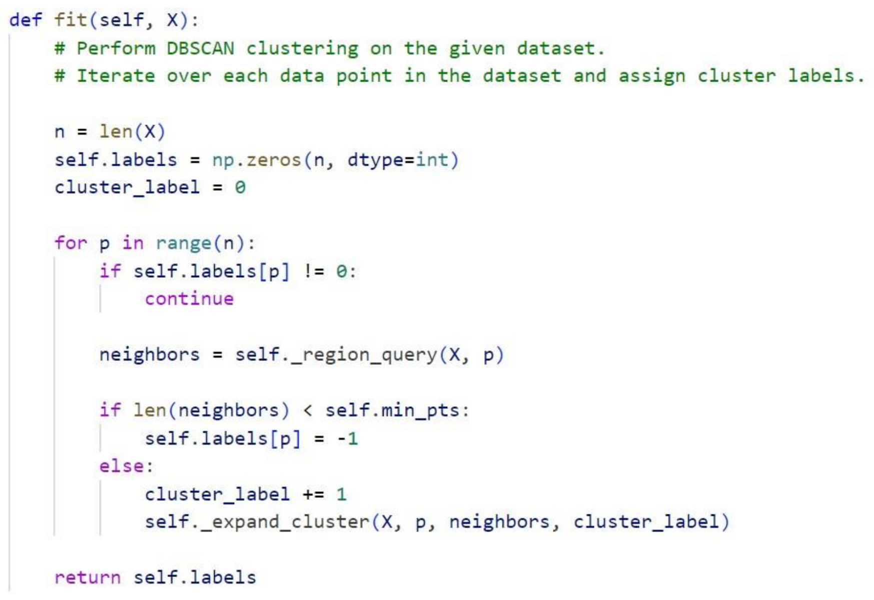
Initialization of DBSCAN Class



```python
class DBSCAN:
    def __init__(self, eps, min_pts):
        # Initialize DBSCAN clustering algorithm.
        # Store the maximum distance between two neighbors (eps)
        # and the minimum number of samples for a core point (min_pts).
        self.eps = eps
        self.min_pts = min_pts
        self.labels = None
```

First step is setting of parameters. The DBSCAN class is initialized with the maximum distance between two neighbors (eps) and the minimum number of samples for a core point (min_pts). The labels attribute will store the cluster labels assigned to each data point in the dataset.

Fitting the Model

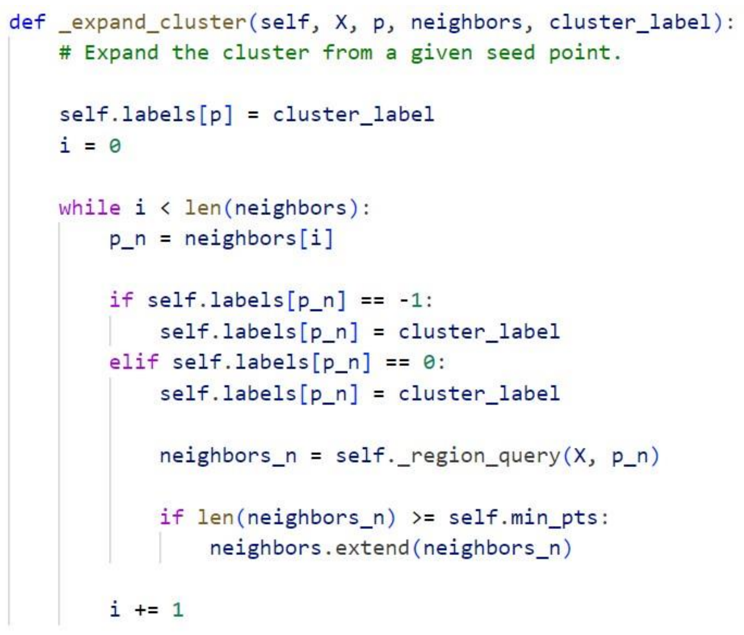


```python
def fit(self, X):
    # Perform DBSCAN clustering on the given dataset.
    # Iterate over each data point in the dataset and assign cluster labels.

    n = len(X)
    self.labels = np.zeros(n, dtype=int)
    cluster_label = 0

    for p in range(n):
        if self.labels[p] != 0:
            continue

        neighbors = self._region_query(X, p)

        if len(neighbors) < self.min_pts:
            self.labels[p] = -1
        else:
            cluster_label += 1
            self._expand_cluster(X, p, neighbors, cluster_label)

    return self.labels
```

The fit method performs the DBSCAN clustering on the given dataset X. It iterates over each data point and assigns cluster labels accordingly. Initially, all labels are set to 0. The variable cluster_label keeps track of the current cluster being assigned.

For each point p, if its label is already assigned (not 0), the loop continues to the next point. Otherwise, it retrieves the neighbors of p using the _region_query method. If the number of neighbors is less than min_pts, p is labeled as noise (-1). Otherwise, a new cluster label is assigned, and the _expand_cluster method is called to expand the cluster from the seed point p.

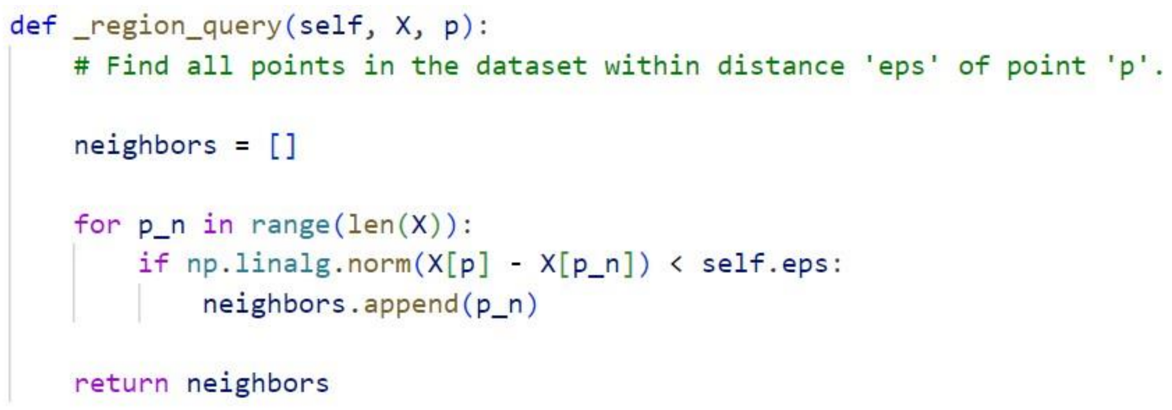
Expanding the Cluster



```python
def _expand_cluster(self, X, p, neighbors, cluster_label):
    # Expand the cluster from a given seed point.

    self.labels[p] = cluster_label
    i = 0

    while i < len(neighbors):
        p_n = neighbors[i]

        if self.labels[p_n] == -1:
            self.labels[p_n] = cluster_label
        elif self.labels[p_n] == 0:
            self.labels[p_n] = cluster_label

            neighbors_n = self._region_query(X, p_n)

            if len(neighbors_n) >= self.min_pts:
                neighbors.extend(neighbors_n)

        i += 1
```

The _expand_cluster method expands the cluster from the seed point p. It assigns the cluster label to p and iterates over the neighbors of p. If a neighbor has a label of -1 (noise point), it is assigned the current cluster label. If the neighbor has an unassigned label (0), it is assigned the current cluster label, and its neighbors are obtained using _region_query. If the number of neighbors is greater than or equal to min_pts, these neighbors are added to the neighbors list to be processed in the next iterations of the loop.

Region Query



```python
def _region_query(self, X, p):
    # Find all points in the dataset within distance 'eps' of point 'p'.

    neighbors = []

    for p_n in range(len(X)):
        if np.linalg.norm(X[p] - X[p_n]) < self.eps:
            neighbors.append(p_n)

    return neighbors
```

The _region_query method finds all points in the dataset X that are within a distance of eps from the point p. It calculates the Euclidean distance between X[p] and each point X[p_n], and if the distance is less than eps, the index p_n is added to the neighbors list.

User's Manual

For detailed instructions and information about the DBSCAN algorithm, please refer to the User's Manual provided in the README.md file located in the Project_DBSCAN directory. The User's Manual contains step-by-step guidance on installation, usage of the DBSCAN algorithm, and visualization of results. You can access the repository [here].

Description of Datasets

Iris Dataset:

- Source: The Iris dataset is publicly available on the UCI Machine Learning Repository, a popular repository for machine learning datasets.

- Link: The dataset can be accessed [here]

- Description: The Iris dataset is a classic and widely used dataset in the field of machine learning. It consists of measurements of four features (sepal length, sepal width, petal length, and petal width) for three different species of Iris flowers (Setosa, Versicolor, and Virginica). The dataset contains a total of 150 instances, with 50 instances for each class.

- Purpose: The Iris dataset is commonly used for tasks such as classification, clustering, and visualization. It serves as a good benchmark dataset for evaluating and comparing different machine learning algorithms.

Wine Dataset:

- Source: The Wine dataset is also available on the UCI Machine Learning Repository.

- Link: You can find the dataset [here]

- Description: The Wine dataset contains the results of a chemical analysis of wines grown in a specific region of Italy. It consists of 13 different features, including alcohol content, malic acid concentration, ash content, and various other attributes related to the wine's composition. The dataset contains a total of 178 instances, divided into three classes representing different cultivars of the wine.

- Purpose: The Wine dataset is commonly used for classification tasks, where the goal is to predict the cultivar of a wine based on its chemical composition. It is a good testing base for our clustering algorithm.

Glass Dataset:

- Source: The Glass dataset is available on the UCI Machine Learning Repository as well.

- Link: You can access the dataset at [here]

- Description: The Glass dataset contains information about various types of glass and their chemical composition. It includes attributes such as the refractive index, sodium, magnesium, aluminum, silicon, and other elements' concentrations. The dataset comprises 214 instances, classified into different glass types, such as building windows, vehicle windows, containers, and tableware.

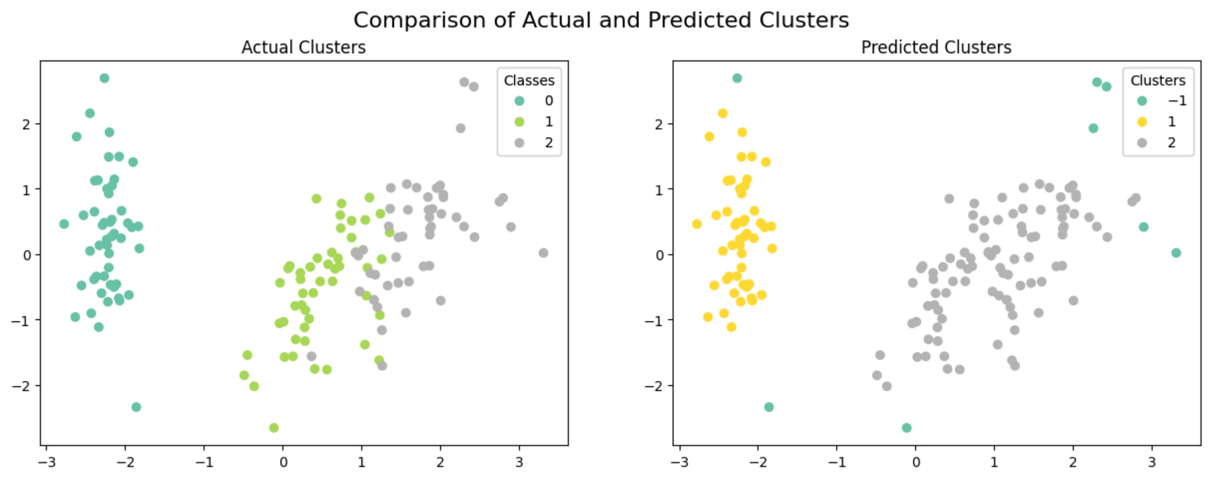
- Purpose: The Glass dataset is often used for classification and pattern recognition tasks. Researchers and practitioners use this dataset to develop models that can identify the type of glass based on its chemical properties. It has been employed in studies related to material analysis, quality control, and forensic investigations. This dataset is a good base for our tests as well.

Experimental Results

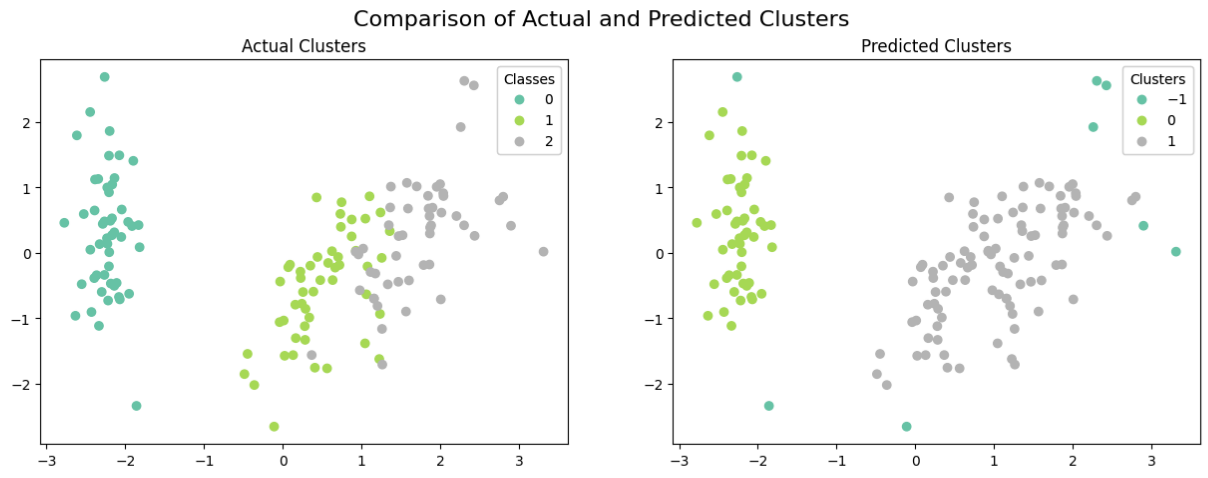
Experiment - 1: DBSCAN Implementation vs Scikit-Learn DBSCAN

Our DBSCAN is identical to Scikit-Learn DBSCAN which can be seen from the cluster results on Iris dataset. (Eps = 0.8, minPts = 10)

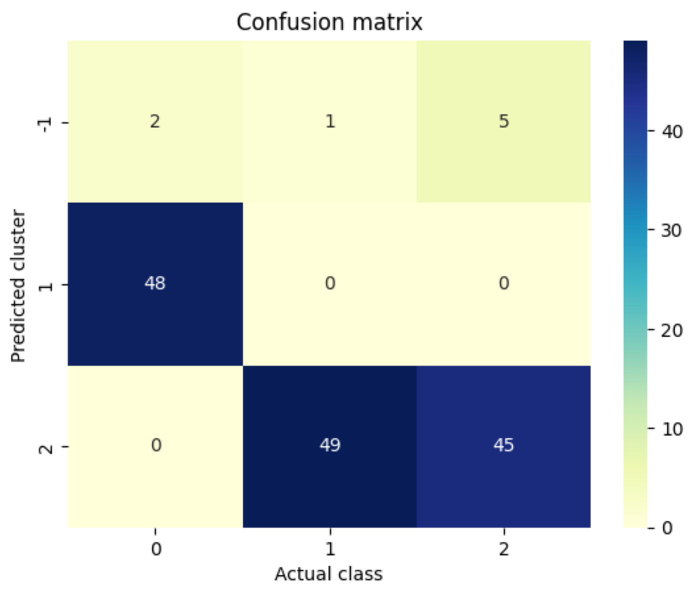
Implemented DBSCAN clusters:



Scikit-Learn DBSCAN Clusters:

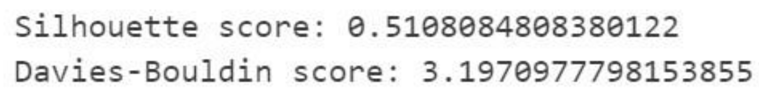


Confusion Matrix:



Predicted cluster \ Actual class	0	1	2
-1	2	1	5
1	48	0	0
2	0	49	45

Scores:



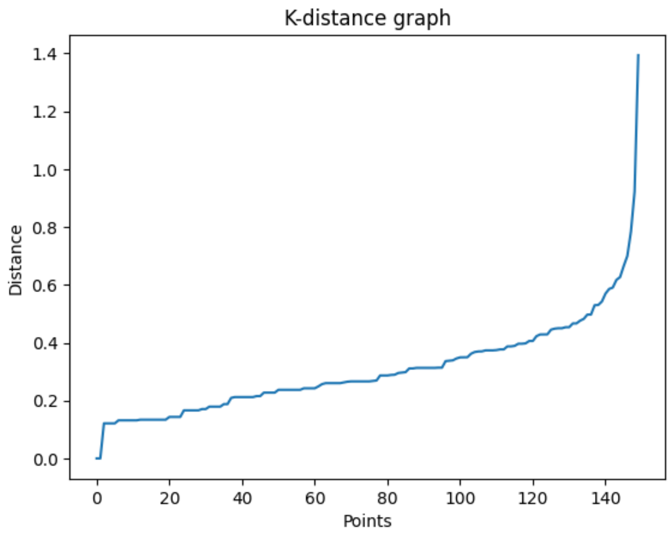
```
Silhouette score: 0.5108084808380122
Davies-Bouldin score: 3.1970977798153855
```

Experiment - 2: DBSCAN Implementation vs Scikit-Learn K-Means on Iris Dataset

DBSCAN Clustering:

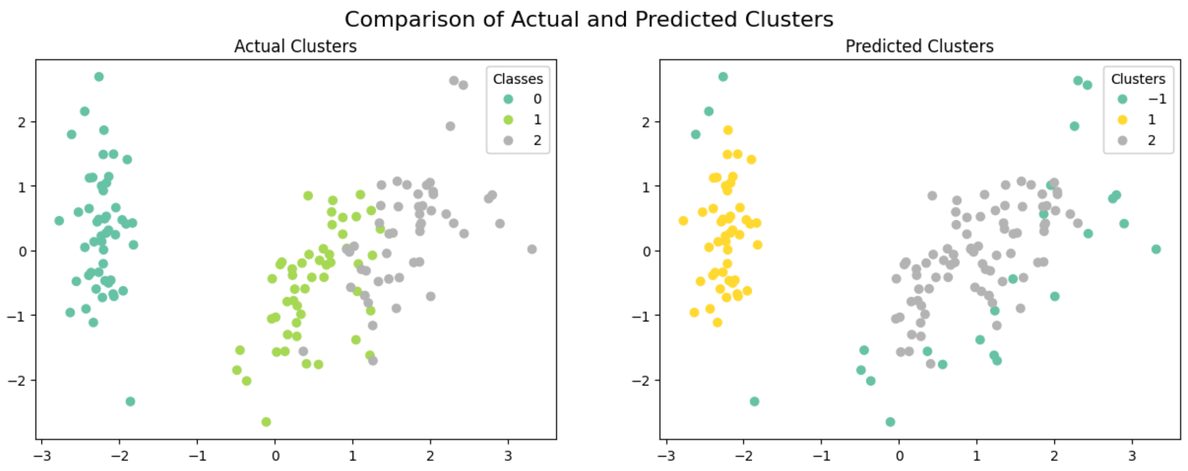
First for minPts = 5, eps value has been selected using k-distance plot and elbow method.

K-distance plot is as follows:

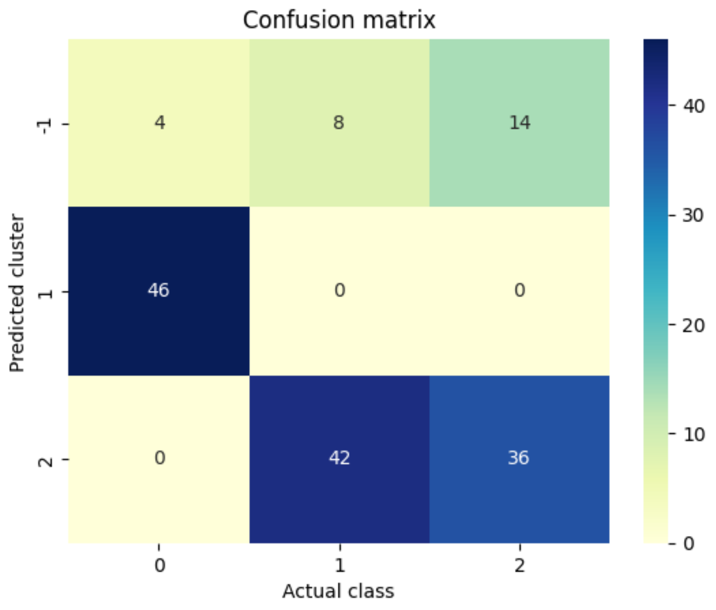


Since elbow point is around 0.6, Eps=0.6 is experimented first.

Clusters for Eps=0.6:



Confusion Matrix for Eps=0.6:

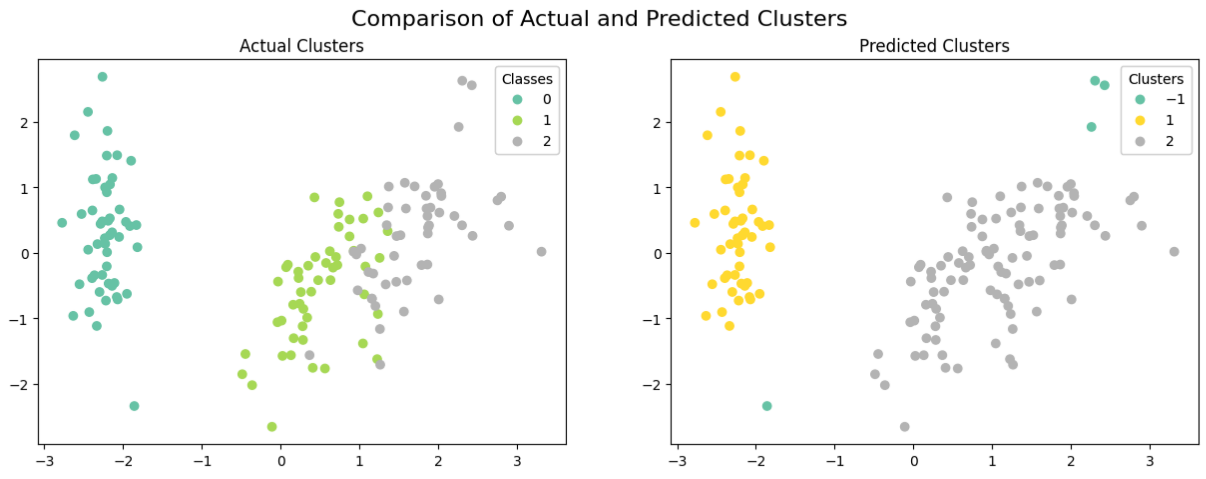


Scores:

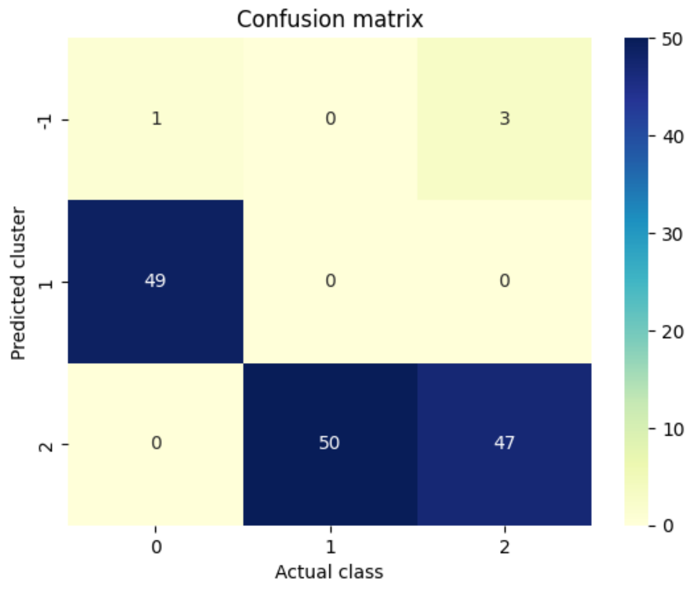
```
Silhouette score: 0.4027617471797909
Davies-Bouldin score: 7.046801925672976
```

There are unnecessarily too many outliers, therefore we tried a higher value for Eps.

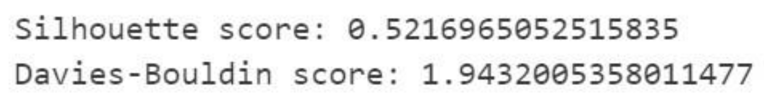
Clusters for Eps =0.8:



Confusion Matrix for Eps=0.8:



Scores:



```
Silhouette score: 0.5216965052515835
Davies-Bouldin score: 1.9432005358011477
```

From the heatmap and scores, we can see that eps = 0.8 is a better choice than eps = 0.6. Silhouette score is higher for Eps = 0.8 and Davies-Bouldin is closer to 0 which indicates a lower intra-cluster distance.

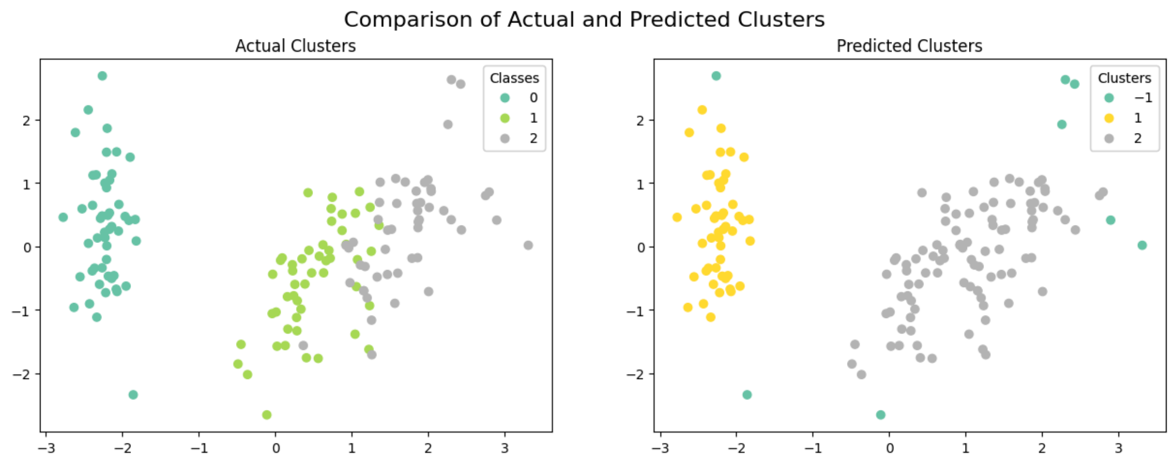
Here we can see that the number of outliers has reduced and seems to be more reasonable. It still has issue of having 2 clusters but that is because of the nature of the data, two clusters are very close to each other and hence DBSCAN is not able to separate them.

Since DBSCAN seperates cluster based on density, it is not guaranteed that best clustering will have the same number of clusters as the number of classes in the data.

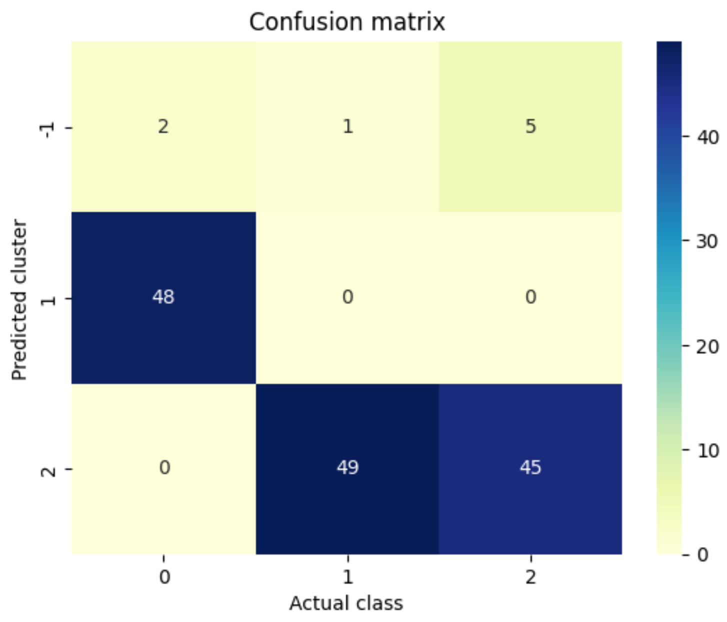
We can try to increase the minPts to 10 and see if it helps in getting more clusters.

For minPts = 10, Eps = 0.8 is experimentes and the results are as follows:

Clusters for minPts=10, Eps = 0.8:

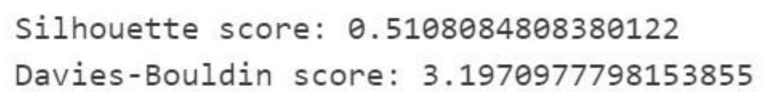


Confusion Matrix:



	0	1	2
-1	2	1	5
1	48	0	0
2	0	49	45

Scores:



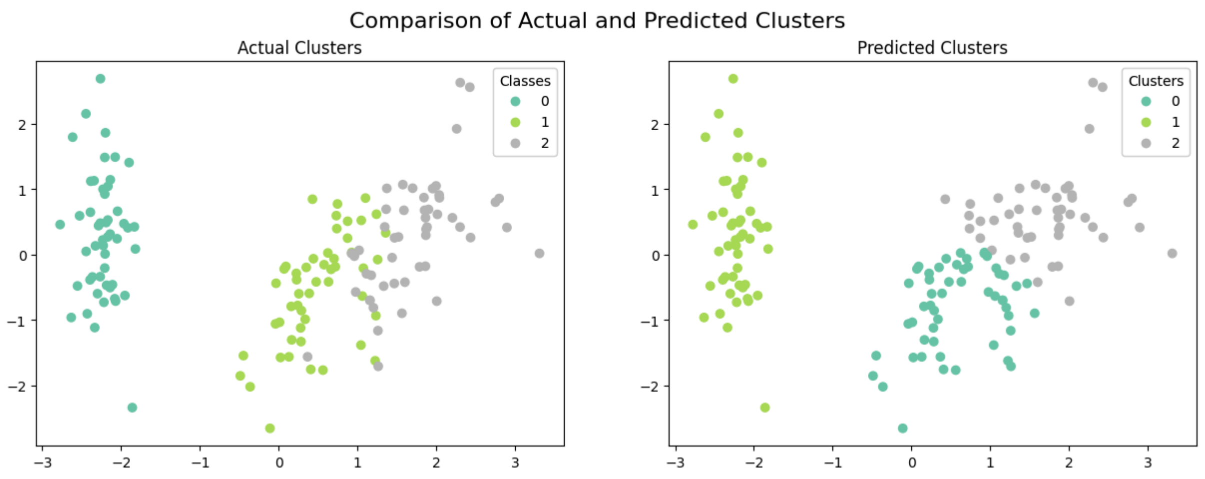
```
Silhouette score: 0.5108084808380122
Davies-Bouldin score: 3.1970977798153855
```

Changing minPts to 10 does not help in getting more clusters and the results are similar to minPts = 5 except that the number of outliers has increased and the silhouette score has decreased a bit. So, we can stick to minPts = 5 and Eps = 0.8.

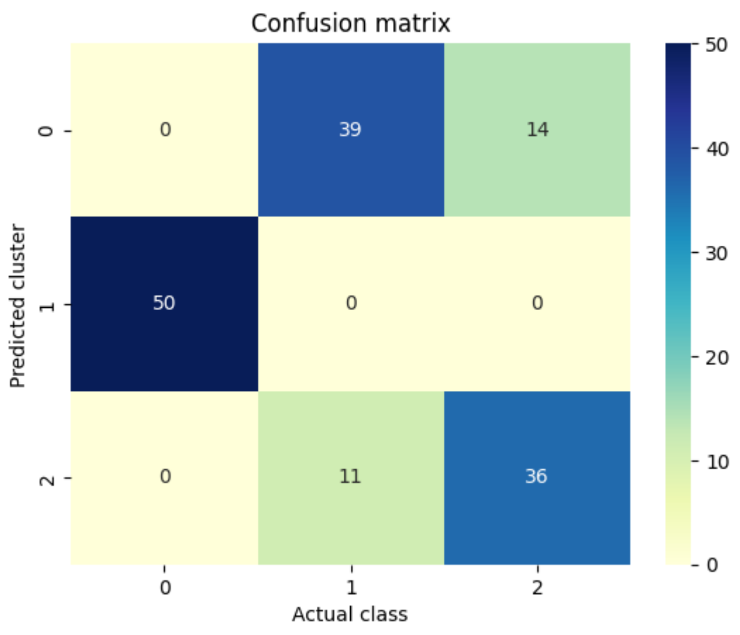
For the best results we got from DBSCAN we will compare them with results of K-Means.

K-Means Clustering:

Clusters for n_clusters = 3:



Confusion Matrix:



Scores:

```
Silhouette score: 0.45994823920518646
Davies-Bouldin score: 0.8335949464754334
```

DBSCAN has a better score when it comes to silhouette but K-Means has a better score when it comes to davies bouldin score because it is closer to 0. This is because for K-Means intra-cluster distance is minimized which is not the case for DBSCAN, since DBSCAN also clusters outliers and has two clusters instead of three. We can understand from the silhouette score of K-means cluster that real clusters don't have well defined borders which makes them hard to seperate especially for cluster 1 and cluster 2.

We should also note that to use K-Means, we need to know the number of clusters beforehand which is not the case for DBSCAN. DBSCAN is also more robust to outliers than K-Means

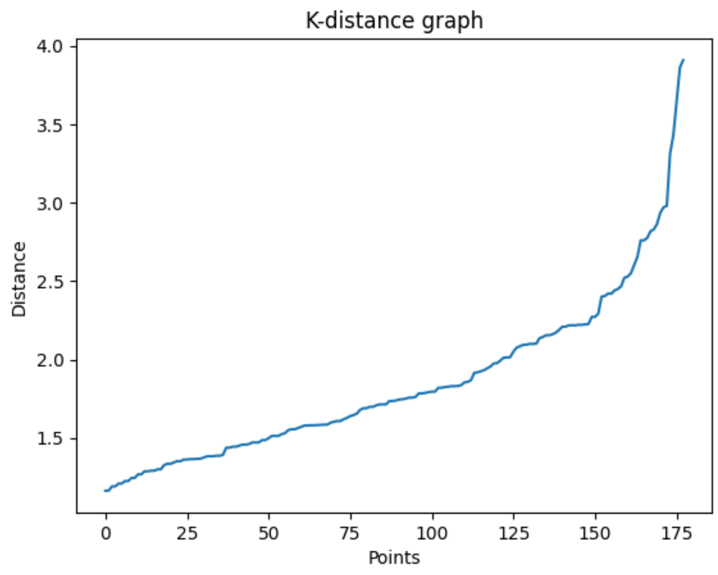
For the conclusion, we can say that K-Means has done a better job than DBSCAN in clustering the iris data.

Experiment - 3: DBSCAN Implementation vs Scikit-Learn K-Means on Wine Dataset

DBSCAN Clustering:

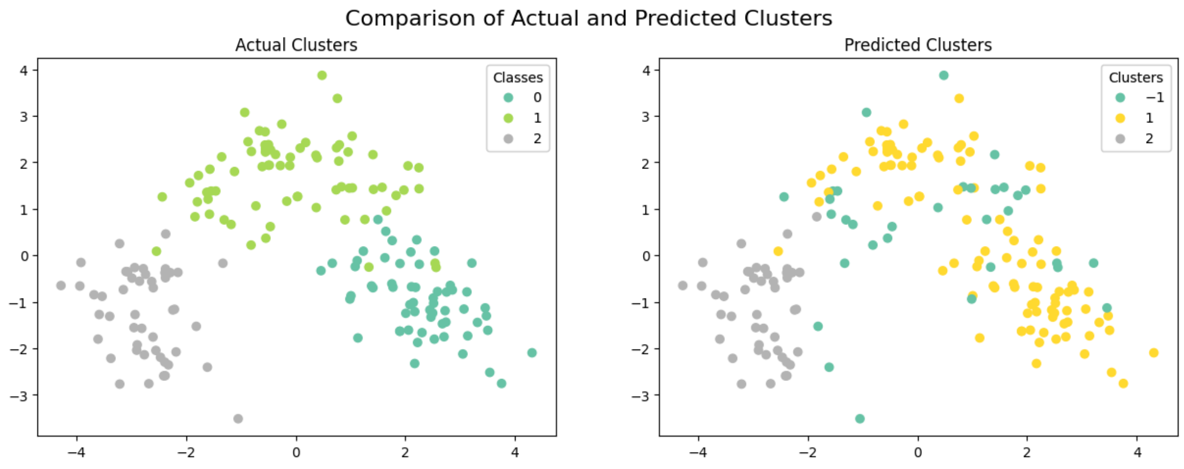
First we start with experimenting for minPts = 10, Eps values have been selected using elbow method on k-distance plot.

K-distance plot is as follows:

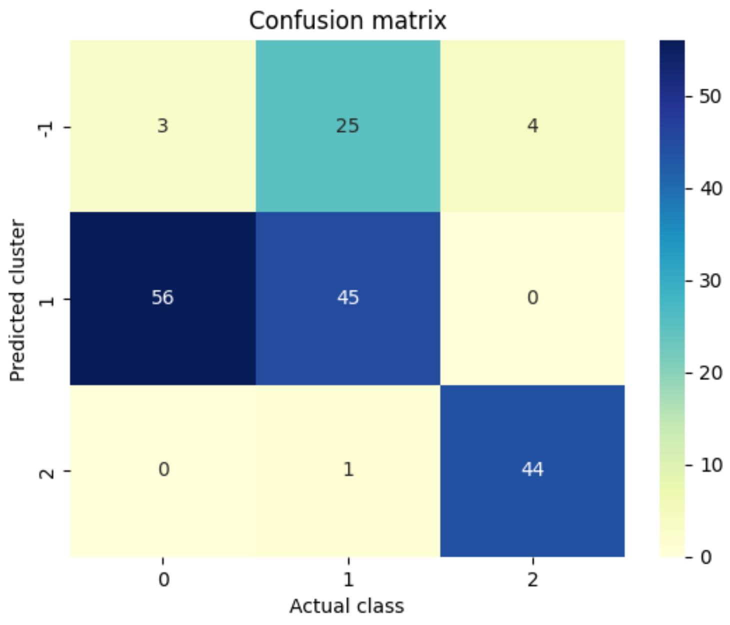


Elbow point is around 2.5, therefore Eps = 2.5 is selected to be experimented first.

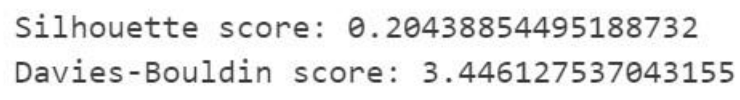
Clustering for minPts=10, Eps=2.5:



Confusion Matrix:



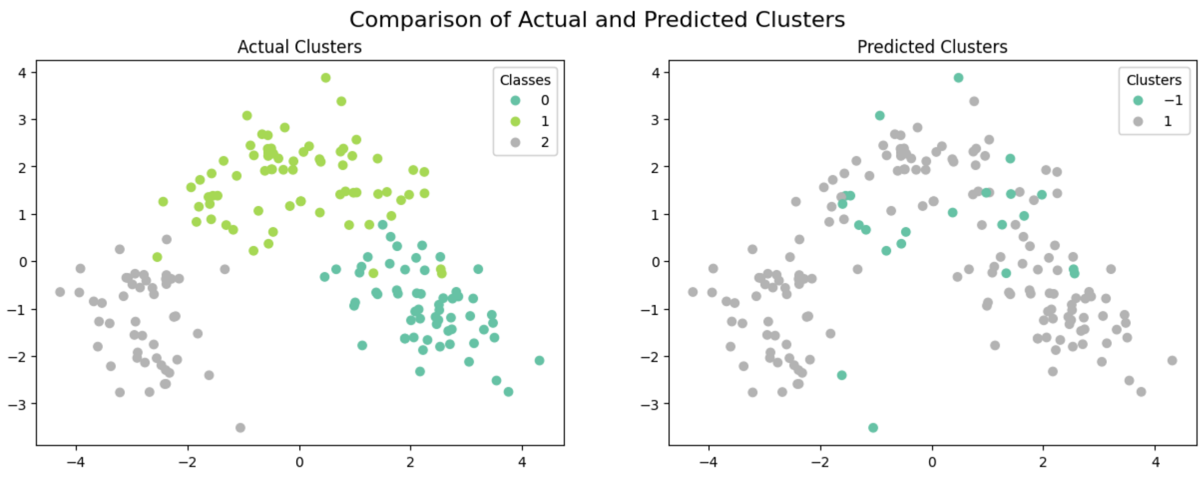
Scores:



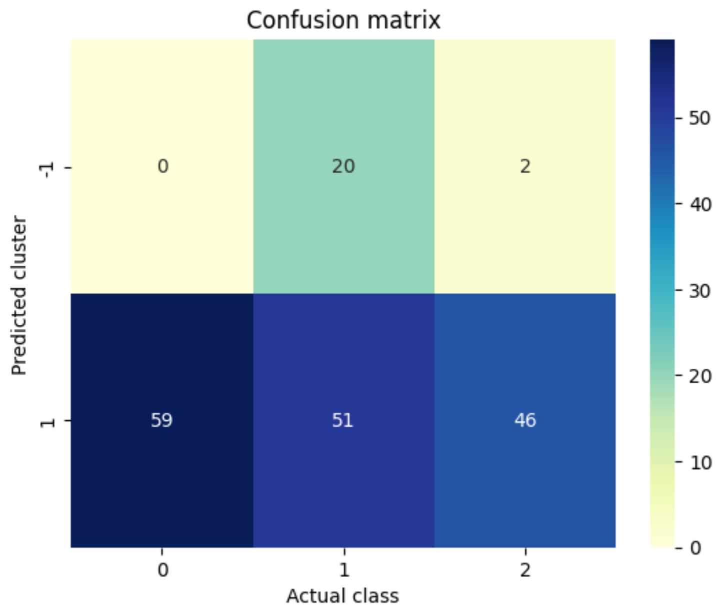
```
Silhouette score: 0.20438854495188732
Davies-Bouldin score: 3.446127537043155
```

There are too many outliers for eps = 2.5. So, we can try a higher value of eps, for example 2.7 to reduce the number of outliers.

Clustering for minPts=10, Eps=2.7:



Confusion Matrix:

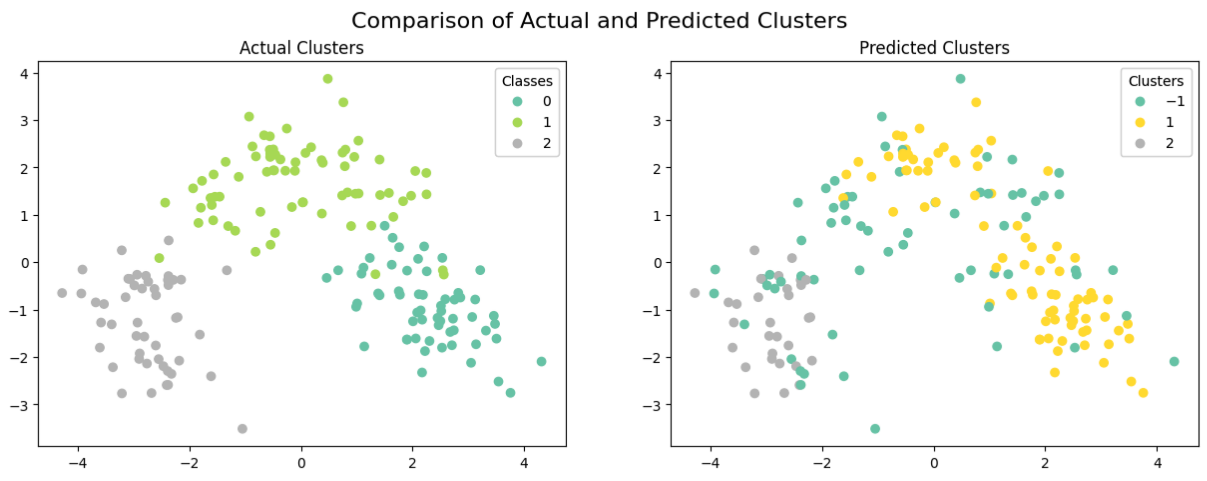


Scores:

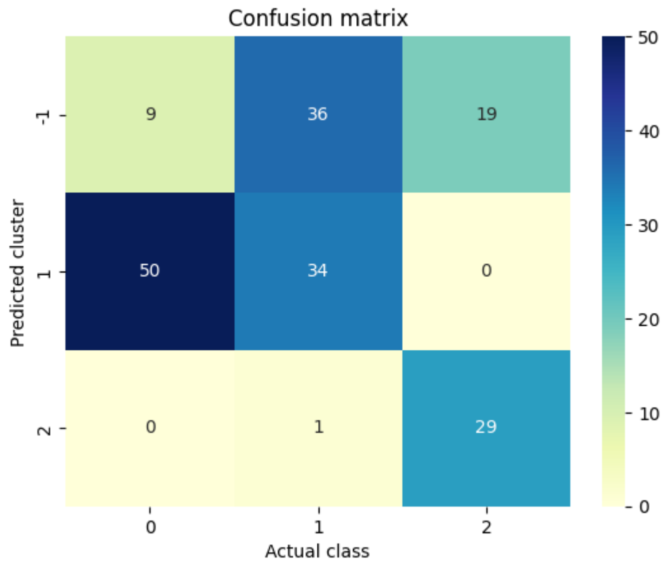
```
Silhouette score: 0.14275197921473814
Davies-Bouldin score: 4.359465380537803
```

There are less outliers but scores are getting worse and we have only one cluster as a result, this result could be meaningful if our purpose was identifying the outliers which is not the case here. So, we can try a lower value of eps, for example, 2.3 to get more clusters.

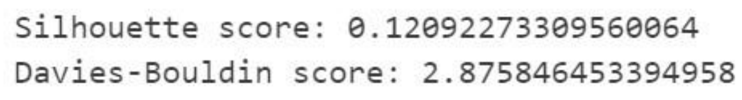
Clustering for minPts=10, Eps=2.3:



Confusion Matrix:



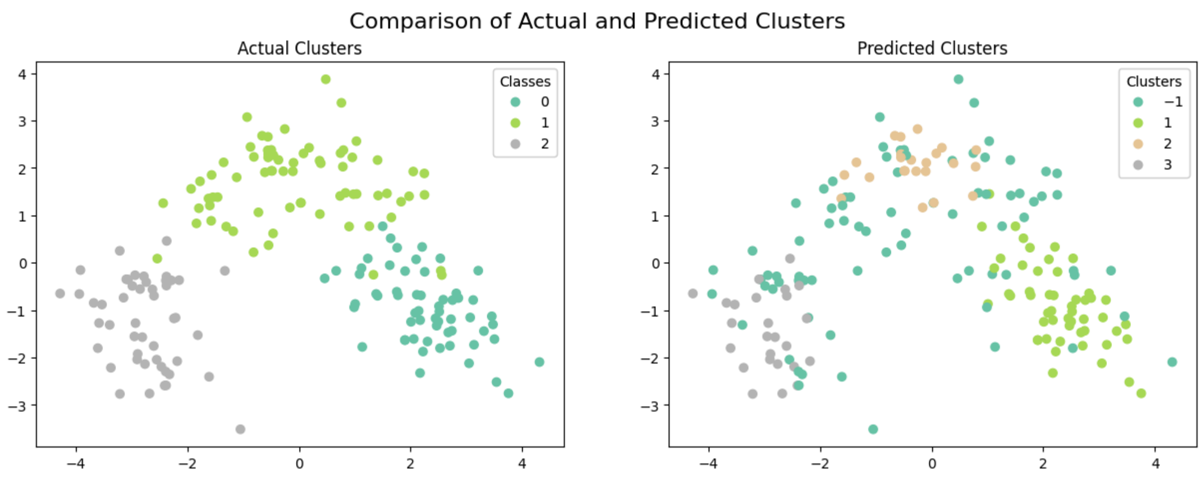
Scores:



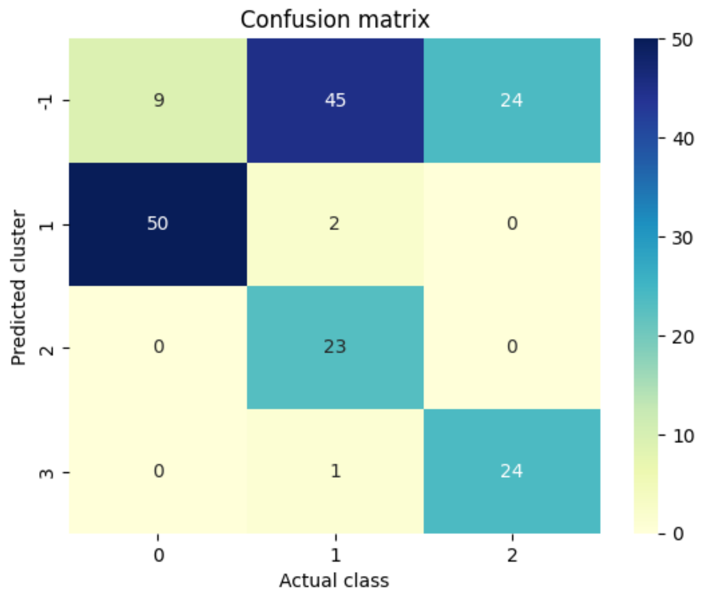
```
Silhouette score: 0.12092273309560064
Davies-Bouldin score: 2.875846453394958
```

With these parameters we get a better result compared to Eps=2.7 but it seems to still have issue of having too many outliers. It seems that Eps=2.5 best choice for minPts=10. We experimented with a higher minPts to see if we can get three clusters.

Clustering for minPts=12, Eps=2.3:



Confusion Matrix:



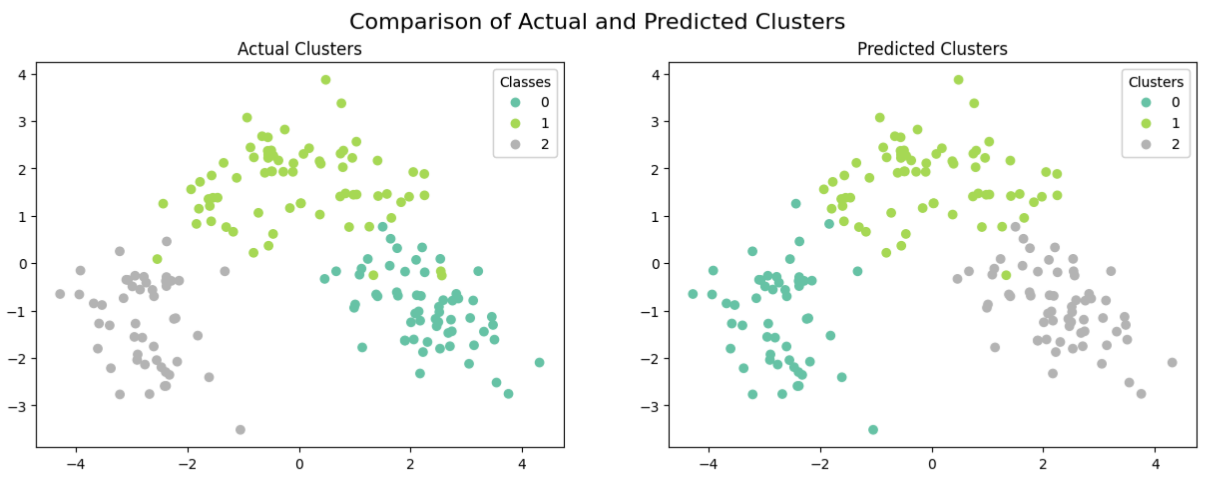
Scores:

```
Silhouette score: 0.10622734553800696
Davies-Bouldin score: 2.4335128980292415
```

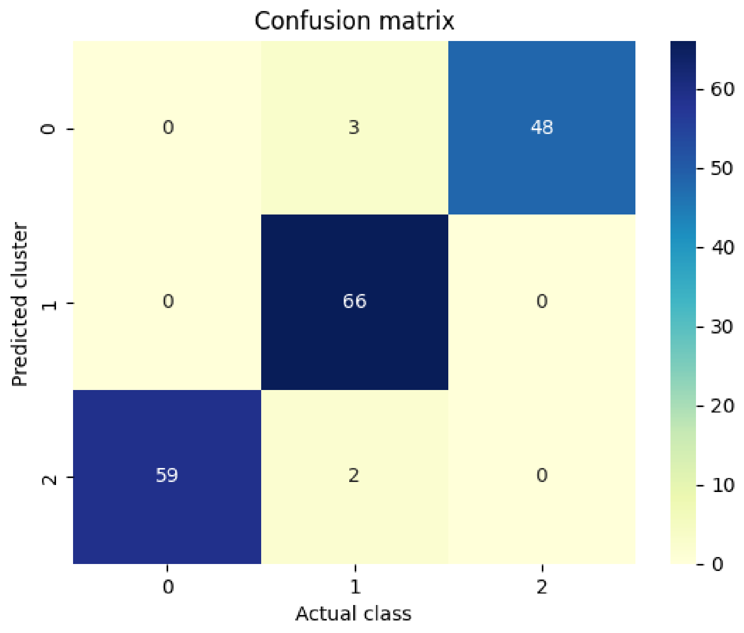
Finally we have 3 clusters as a result but there are too many outliers as a result of increasing minPts. Most of the non-outlier points have been assigned correctly but the problem is we have almost as much as data points asssigned as outliers. This is a sign of high intra-cluster distance especially for cluster 1 and cluster 2. Again it depends on what we want to achieve with the cluster, if we want to also discover outliers for each cluster DBSCAN can be helpful as we can see from the confusion matrix and cluster plot.

K-Means Clustering:

Clustering for n_clusters=3:



Confusion Matrix:



Scores:

```
Silhouette score: 0.28594199657074876
Davies-Bouldin score: 1.391793832317738
```

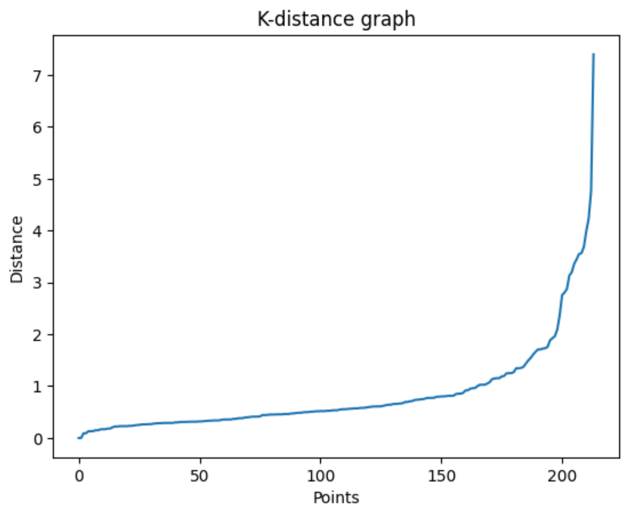
K-Means has done an excellent job in clustering the wine dataset. It has 3 clusters and according to the heatmap and plots very few data points are clustered mistakenly.

Experiment - 4: DBSCAN Implementation vs Scikit-Learn K-Means on Glass Dataset

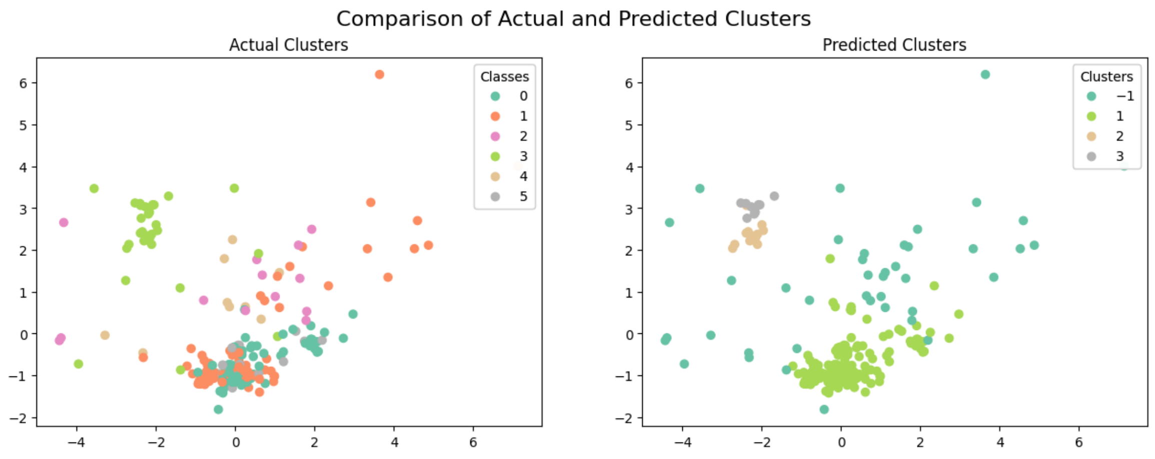
DBSCAN Clustering:

First we start with experimenting for minPts = 5, Eps values have been selected using elbow method on k-distance plot.

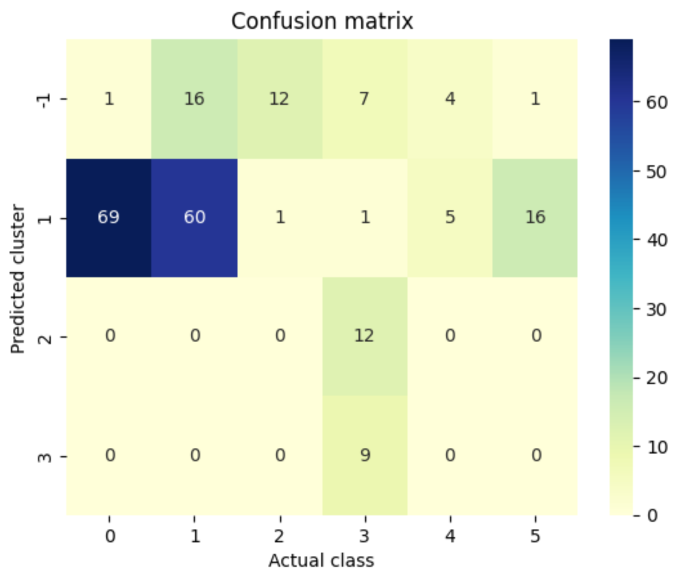
K-distance plot is as follows:



Clustering for minPts=5, Eps=1.5:



Confusion Matrix:



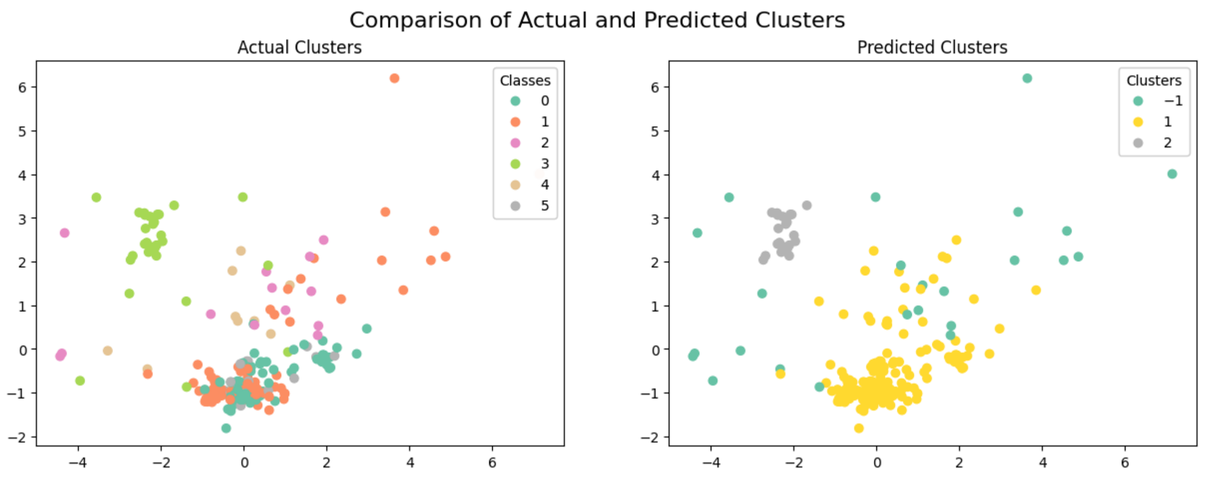
Scores:

```
Silhouette score: 0.38247561828357857
Davies-Bouldin score: 1.9371871731151702
```

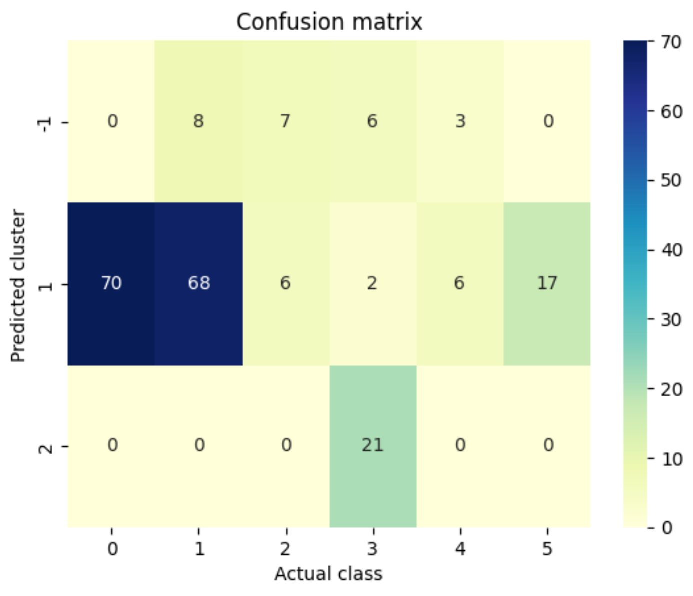
We can clearly say this dataset is even harder to seperate than the other two datasets.

DBSCAN was able to seperate the data into 3 clusters and the predicted clusters seem far from real clusters and there are too many outliers. This is because the intra-cluster distance is too high in this dataset and there are lots of outliers. Therefore we have tried different Eps and minPts values to see if we can get a better result.

Clustering for minPts=5, Eps=2.0:



Confusion Matrix:

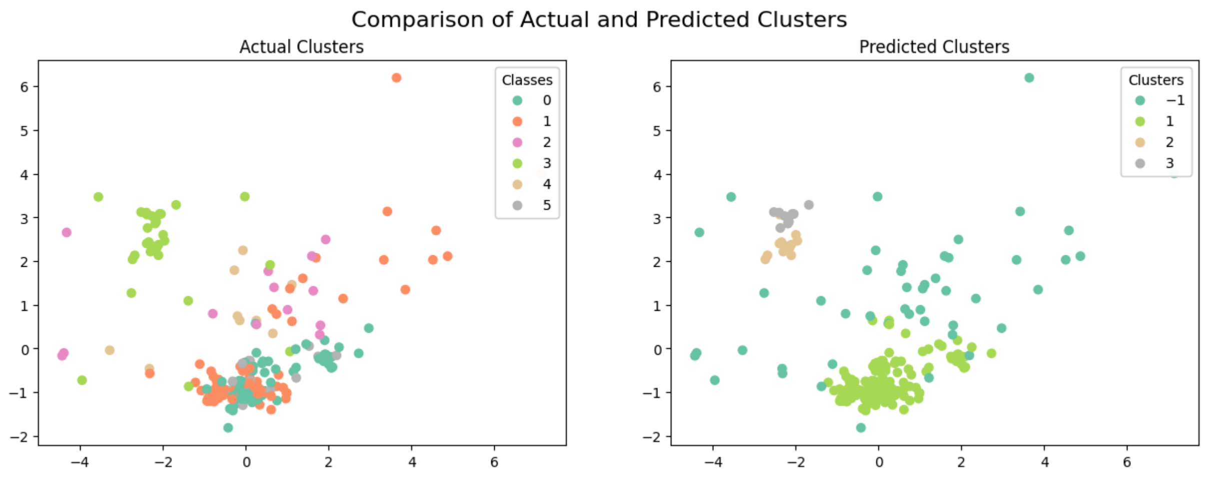


Predicted cluster \ Actual class	0	1	2	3	4	5
-1	0	8	7	6	3	0
1	70	68	6	2	6	17
2	0	0	0	21	0	0

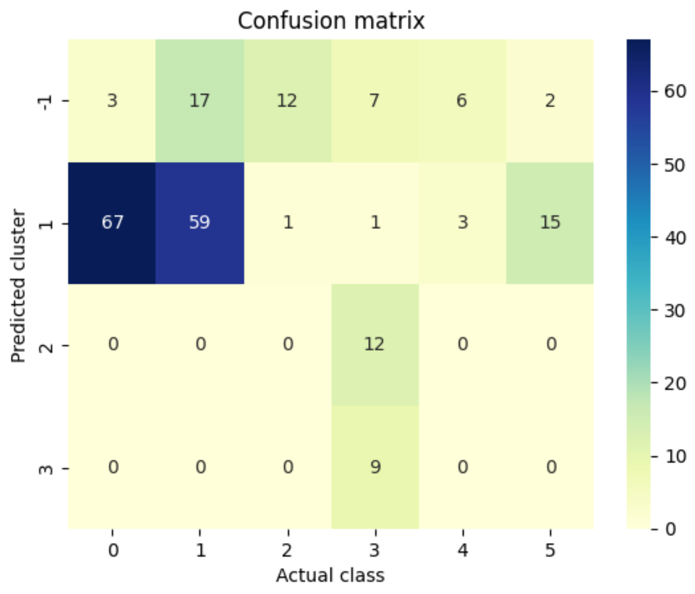
Scores:

```
Silhouette score: 0.41240480242518684
Davies-Bouldin score: 2.4378792725287397
```

Clustering for minPts=5, Eps=1.3:



Confusion Matrix:



	0	1	2	3	4	5
-1	3	17	12	7	6	2
1	67	59	1	1	3	15
2	0	0	0	12	0	0
3	0	0	0	9	0	0

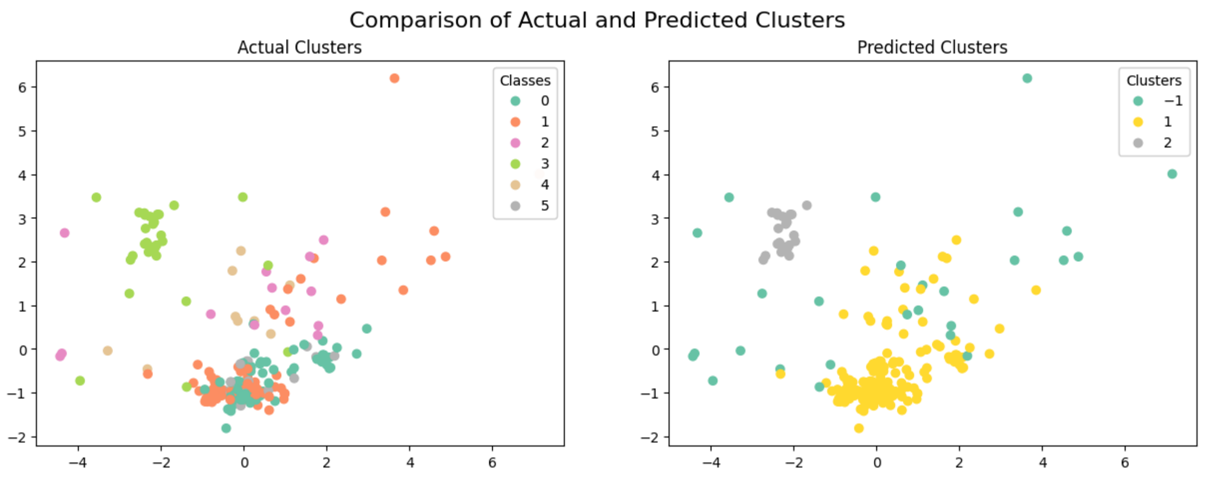
Scores:

```
Silhouette score: 0.3729046459820883
Davies-Bouldin score: 1.900897151351298
```

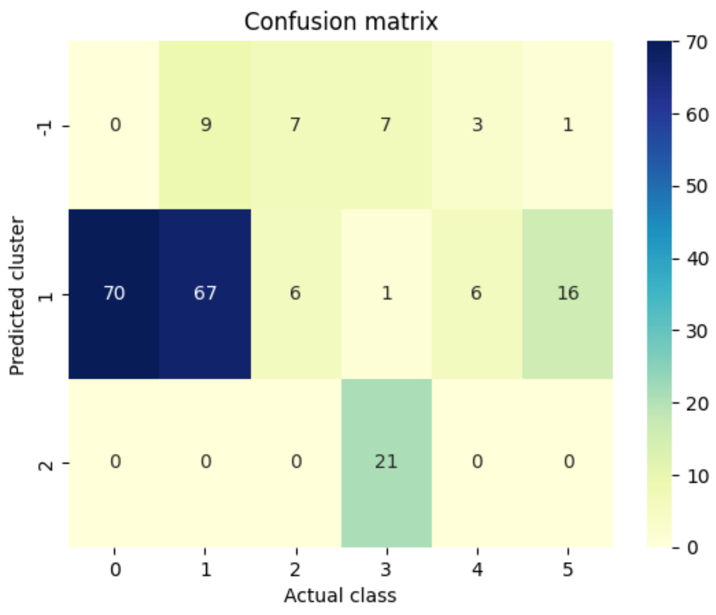
Eps = 1.5 seems to be a better choice than eps = 1.3 and eps = 2.0. We have less outliers and the clusters seem reasonable.

We will be decreasing minPts to 3 for the next experiment, because in this dataset there are clusters with a few data points and since spread is high we will use 1.8 for eps.

Clustering for minPts=3, Eps=1.8:



Confusion Matrix:



Predicted cluster \ Actual class	0	1	2	3	4	5
-1	0	9	7	7	3	1
1	70	67	6	1	6	16
2	0	0	0	21	0	0

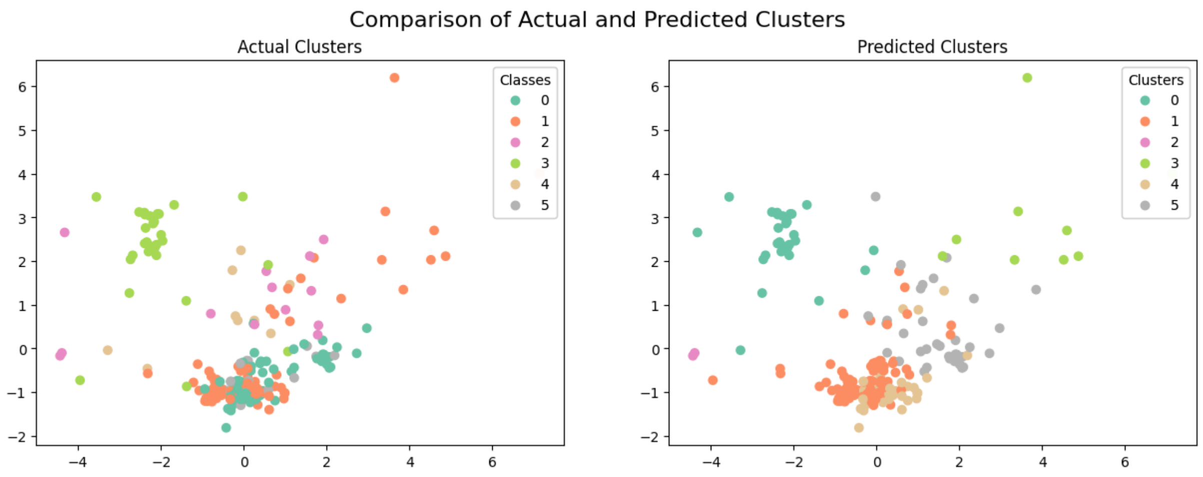
Scores:

```
Silhouette score: 0.40866088793153155
Davies-Bouldin score: 2.582037410078498
```

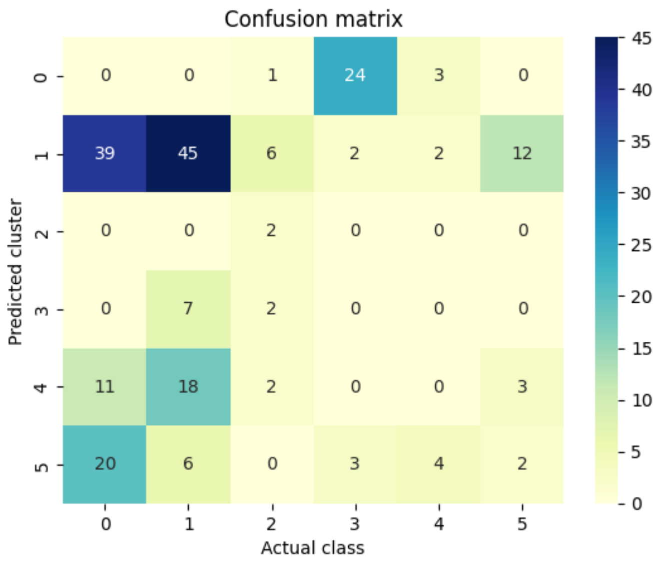
This is a better result than minPts = 5. It is able to cluster two real clusters correctly and there are less outliers. But regarding the all 6 clusters it seems like DBSCAN is not the best choice for this dataset because we have clusters with a few data points and the spread of the clusters or in other words intra-cluster distance is high which leads DBSCAN to find less clusters and more noise points.

K-Means Clustering:

Clustering for n_clusters=6:

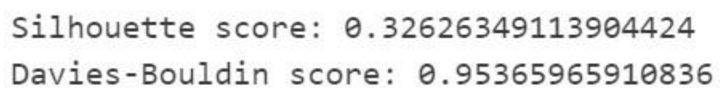


Confusion Matrix:



Predicted cluster \ Actual class	0	1	2	3	4	5
0	0	0	1	24	3	0
1	39	45	6	2	2	12
2	0	0	2	0	0	0
3	0	7	2	0	0	0
4	11	18	2	0	0	3
5	20	6	0	3	4	2

Scores:



```
Silhouette score: 0.32626349113904424
Davies-Bouldin score: 0.95365965910836
```

K-Means is also not able to cluster the data correctly but it has done a better job than DBSCAN in clustering the glass dataset. We can see that at least some clusters are seperated correctly.

CONCLUSIONS

In conclusion, DBSCAN is a powerful clustering algorithm that offers several advantages in identifying clusters of arbitrary shapes in a dataset. Throughout this project, we have explored the key characteristics and benefits of DBSCAN, as well as its underlying concepts and parameters.

One of the major strengths of DBSCAN is its ability to handle datasets with varying densities and noise effectively. By defining a neighborhood based on the density of points, DBSCAN can identify dense regions as clusters and separate them from sparse regions and outliers. This makes DBSCAN particularly useful for datasets where clusters have different shapes, sizes, and densities.

Another advantage of DBSCAN is its ability to discover clusters automatically, without the need for specifying the number of clusters beforehand. This flexibility is especially beneficial in scenarios where the number of clusters is unknown or when dealing with large and complex datasets.

Moreover, DBSCAN is a relatively simple algorithm to implement and can efficiently process large datasets due to its time complexity of O(n log n). It also provides robustness to parameter settings, such as the minimum number of points required to form a dense region (minPts) and the distance threshold (epsilon).

However, DBSCAN does have some limitations. It can struggle with datasets that have varying densities across different regions, as it may create one large cluster instead of separating them into smaller clusters. Additionally, DBSCAN may face difficulties in identifying clusters of significantly different densities. Choosing appropriate values for the parameters can also be challenging and requires some trial and error.

In summary, DBSCAN is a valuable clustering algorithm that excels in identifying clusters of arbitrary shapes and handling datasets with varying densities and noise. Its simplicity, flexibility, and efficiency make it a popular choice in various domains, such as spatial data analysis, image segmentation, and anomaly detection. By understanding the principles and characteristics of DBSCAN, we can leverage its capabilities to gain insights and extract meaningful patterns from complex datasets.

BIBLIOGRAPHY

- Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. "A density-based algorithm for discovering clusters in large spatial databases with noise." In Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD'96).