Bangabandhu Sheikh Mujibur Rahman Digital University,

Bangladesh.

Faculty: Cyber Physical Systems

Department: IoT and Robotics Engineering (IRE)

Course Title: Data Science

Course Code: IoT 4313

Assignment-02: Clustering

Submitted To:
Nurjahan Nipa
Lecturer
Department of IRE, BDU.

Submitted By:
D. M. Khalid Mahmud
Id: 1901004
Session: 2019-20
Department of IRE

Submission Date: 24 October 2023

Clustering

Let's imagine you're owning a supermarket mall and through membership cards, you have some basic data about your customers like Customer ID, age, gender, annual income and spending score, which is something you assign to the customer based on your defined parameters like customer behavior and purchasing data.

The main aim of this problem is learning the purpose of the customer segmentation concepts, also known as market basket analysis, trying to understand customers and separate them in different groups according to their preferences, and once the division is done, this information can be given to marketing team so they can plan the strategy accordingly.

This Mall_Customer dataset that has been provided to you is composed by the following five features:

▪ CustomerID: Unique ID assigned to the customer

▪ Gender: Gender of the customer

▪ Age: Age of the customer

▪ Annual Income (k$): Annual Income of the customer

▪ Spending Score (1-100): Score assigned by the mall based on customer behavior and spending nature.

In this particular dataset we have 200 samples to study.

PART (A)

K-means Clustering: In this part, you will be utilizing K-means clustering algorithm to identify the appropriate number of clusters. You may use any language and libraries to implement K-mean clustering algorithm. Your K-mean clustering algorithm should look for appropriate values of K at least in the range of 0 to 15 and show their corresponding sum-of-squared errors (SSE).

Step 1:

1. Importing required libraries:

   - The code begins by importing necessary Python libraries.

   - `pandas` is imported for data manipulation.

- `numpy` is imported for numerical operations.

- `matplotlib.pyplot` is imported for data visualization.

- `sklearn.cluster` is imported to use the K-Means clustering algorithm.

2. Load the dataset:

- The code loads a dataset from a CSV file named "Mall_Customers.csv." You should replace the file path with the actual path to your dataset.

3. Selecting relevant features for clustering:

- The code selects the features that will be used for clustering. In this case, it chooses columns 3 and 4 from the dataset, which represent "Annual Income" and "Spending Score."

4. Initialize a list to store SSE values for different K:

- The variable `sse` is created as an empty list. It will be used to store the Sum of Squared Errors (SSE) for different values of K.

5. Try different values of K from 1 to 15:

- A `for` loop is used to iterate through values of K .

6. K-Means clustering:

- Inside the loop, the code creates a K-Means clustering model with the current value of K.

- `KMeans` is instantiated with the following parameters:

  - `n_clusters`: The current value of K (number of clusters).

  - `init`: 'k-means++' is used to initialize cluster centers in a smart way.

  - `max_iter`: Maximum number of iterations for the algorithm.

- `n_init`: Number of times the algorithm will be run with different centroid seeds.

- `random_state`: A random seed to ensure reproducibility.

7. Fit the K-Means model to the data:

- The `kmeans.fit(X)` line fits the K-Means model to the data, where `X` contains the selected features.

8. Calculate and append SSE:

- After fitting the model, the code calculates the Inertia, which is the sum of squared distances to the nearest cluster center.

- The calculated Inertia is appended to the `sse` list, which stores the SSE values for different values of K.

9. Plot the Elbow Method graph:

- After iterating through all values of K, the code proceeds to create a plot to visualize the Elbow Method.

- It creates a figure and sets its size using `plt.figure()`.

- It plots the SSE values against the number of clusters (K) using `plt.plot()`.

- The title, labels for the x and y axes, and grid are added to the plot.

- Finally, `plt.show()` is called to display the plot.

Step 2:

1. Define the optimal K:

- The variable `optimal_k` is set to the value that represents the optimal number of clusters. This value is typically determined using the Elbow Method or another clustering evaluation method. You should replace it with the K value chosen based on the elbow method.

2. Create a K-Means model with the optimal K:

- The code initializes a K-Means clustering model using the `KMeans` class from scikit-learn.

- It sets the number of clusters to the previously determined `optimal_k`.

- The `init`, `max_iter`, `n_init`, and `random_state` parameters are set the same way as in the previous code to ensure consistency in results.

3. Fit the K-Means model to the data:

- The `kmeans.fit(X)` line fits the K-Means model to the data, where `X` contains the selected features, which are "Annual Income" and "Spending Score."

4. Add cluster labels to the dataset:

- The code adds a new column labeled "Cluster" to the original dataset, which stores the cluster labels assigned by the K-Means algorithm to each data point.

4. Create a list of unique cluster labels:

- The code extracts the unique cluster labels from the "Cluster" column of the dataset. This will be used to differentiate and plot each cluster separately.

5. Plot each cluster:

- A figure is created using `plt.figure()` with specified dimensions.

- A loop iterates through each unique cluster label.

- For each cluster, it uses `plt.scatter()` to plot the data points of that cluster based on "Annual Income" and "Spending Score."

- A label is provided to identify each cluster on the plot.

6. Plot cluster centers:

- After plotting the data points for each cluster, the code adds red X markers to represent the cluster centers (centroids). These are the mean values of data points within each cluster.

- `kmeans.cluster_centers_` contains the coordinates of the cluster centers.

- They are plotted using `plt.scatter()` with a different marker, color, and size to distinguish them from the data points.

6. Add labels and legend to the plot:

- The code adds a title, labels for the x and y axes, and a legend to the plot for clarity.

- The title includes the chosen K value for the clustering.

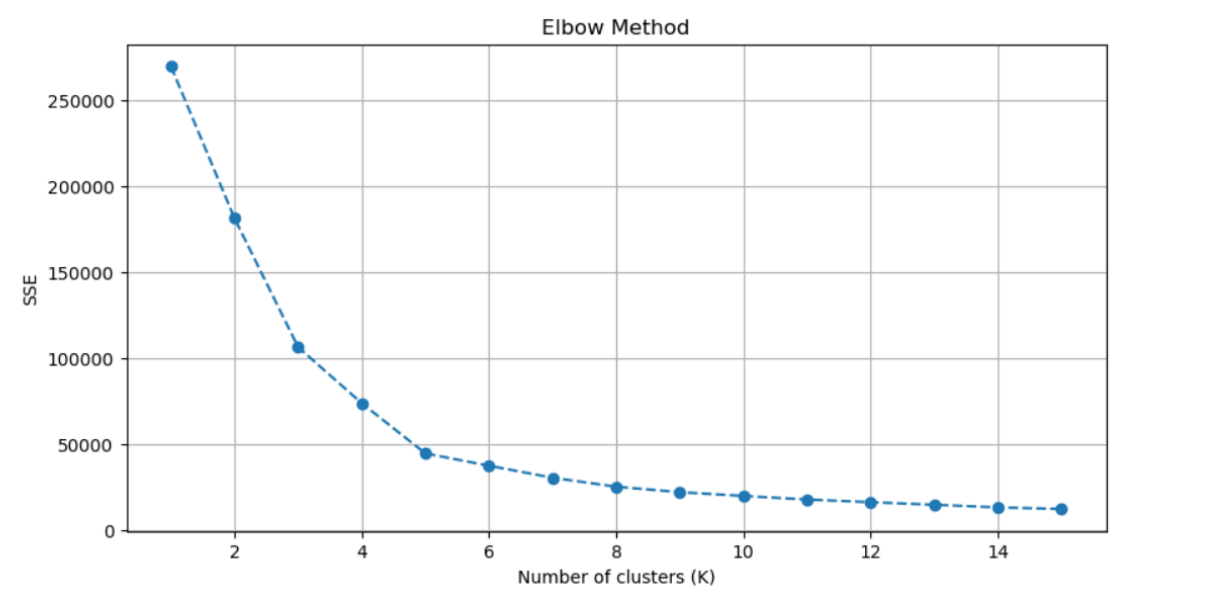
- The legend helps identify clusters and centroids on the plot.

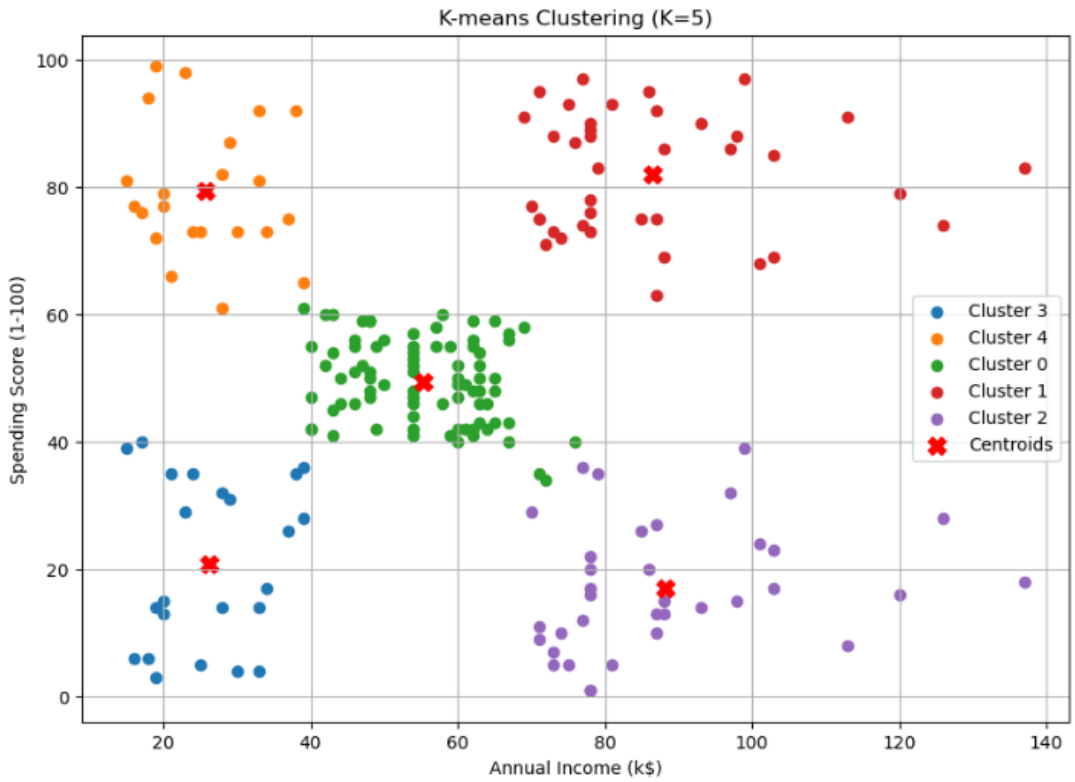
6. Display the plot:

- Finally, `plt.show()` is called to display the plot, showing the clustered data points and their centroids.

Here, clustering size is 5.

Result:





PART (B)

Hierarchical Clustering: In this part, you will apply hierarchical clustering algorithm (agglomerative or divisive) to the provided mall dataset.

Steps:

1. Import the necessary libraries:

- The code starts by importing the required Python libraries for data manipulation and visualization. These libraries include Pandas for data handling, NumPy for numerical operations, Matplotlib for plotting, and functions from SciPy and Scikit-Learn for hierarchical clustering.

2. Load the dataset:

- The script loads a dataset from a CSV file using Pandas. The data is typically expected to contain information about mall customers. You should replace the file path in the `read_csv` function with the actual path to your data.

3. Select relevant features for clustering:

- The script selects specific features from the loaded dataset to use for clustering. In this case, it extracts the "Annual Income" and "Spending Score" columns, which will be used as the input for clustering.

4. Perform hierarchical clustering:

- The script sets up the hierarchical clustering process. It specifies the linkage method (e.g., 'ward', 'single', 'complete', 'average', etc.) and the desired number of clusters.

5. Create the linkage matrix:

- The linkage matrix is constructed using the specified linkage method and the data points (Annual Income and Spending Score) that were selected earlier. The linkage matrix contains information about the relationships between data points.

6. Plot the dendrogram:

- This code segment generates a dendrogram, which is a visual representation of the hierarchical clustering process. The dendrogram shows the order in which data points are merged into clusters and can help you determine an appropriate number of clusters.

7. Fit Agglomerative Clustering with the chosen number of clusters:

- The Agglomerative Clustering algorithm is applied to the data. It uses the linkage matrix and the number of clusters specified to perform the clustering process.

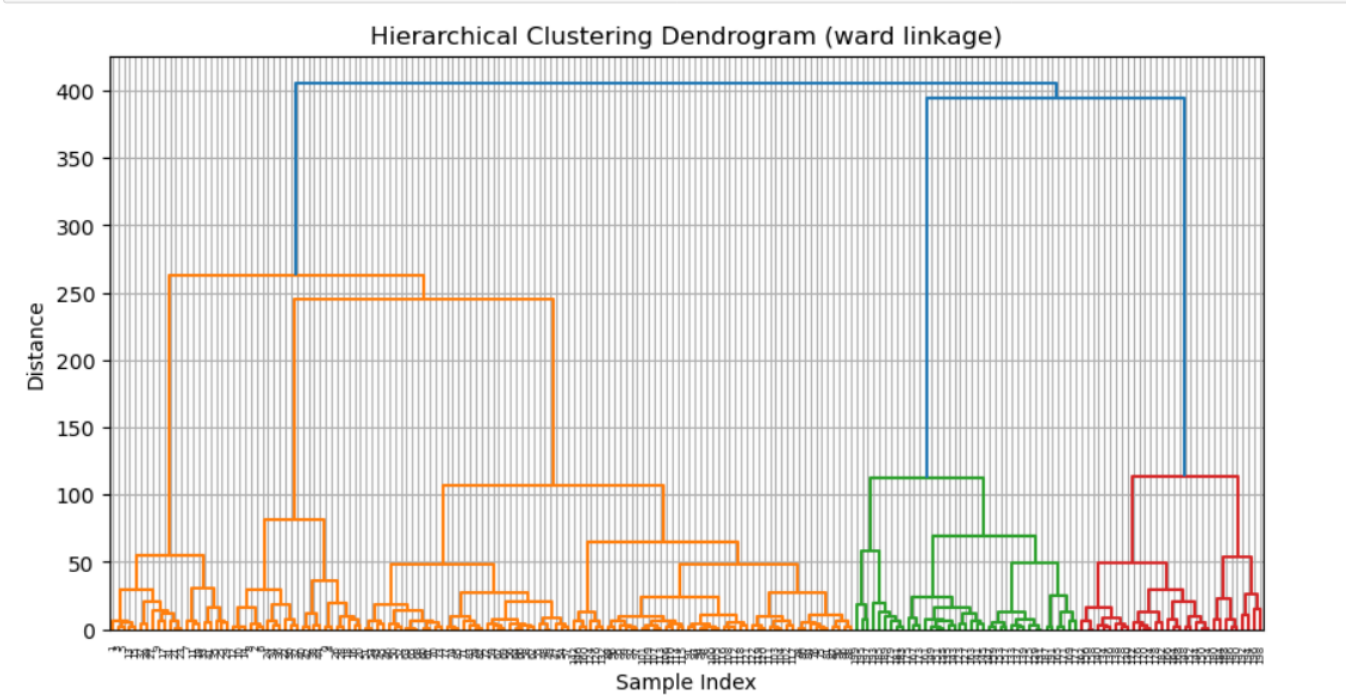
8. Add cluster labels to the dataset:

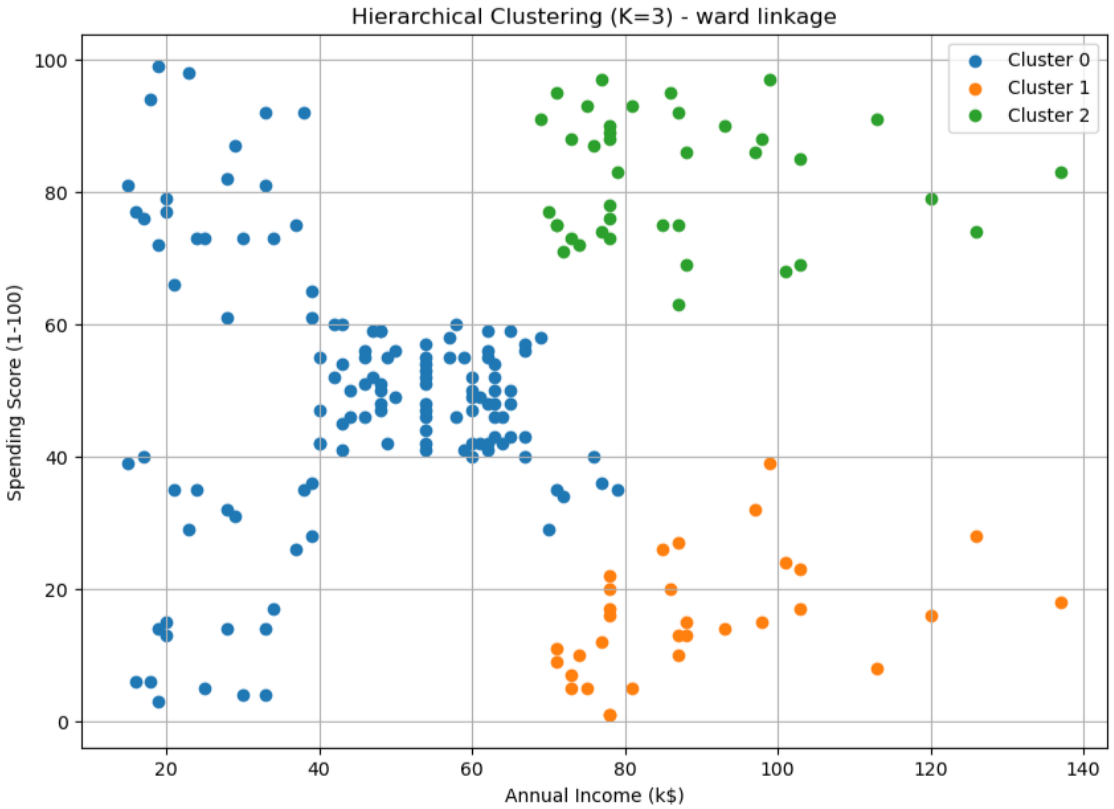
- The cluster labels obtained from the clustering process are added to the original dataset. This allows you to associate each data point with its corresponding cluster.

9. Visualize the clusters:

- Finally, the script creates a scatter plot to visualize the clusters. It iterates through each cluster, plotting data points from the dataset with different markers or colors for each cluster. This visualization helps you understand how the data points are grouped into clusters.

Result:





PART (C)

Density-based Clustering: In this part, you will apply density-based clustering algorithm to the provided dataset.

Steps:

1. `import numpy as np`: This line imports the NumPy library and aliases it as "np" to make it easier to reference in the code.

2. `import pandas as pd`: This line imports the Pandas library and aliases it as "pd" to make it easier to reference in the code.

3. `import matplotlib.pyplot as plt`: This line imports the Matplotlib library's pyplot module and aliases it as "plt" to make it easier to reference in the code. Matplotlib is used for data visualization.

4. `df = pd.read_csv("Mall_customers.csv")`: This line reads a CSV file named "Mall_customers.csv" using Pandas and stores the data in a DataFrame called "df." The DataFrame is a data structure used for data manipulation and analysis in Pandas.

5. `df.head()`: This line displays the first few rows of the DataFrame "df," providing a quick overview of the data.

6. `df.tail()`: This line displays the last few rows of the DataFrame "df."

7. `df.shape`: This line returns the shape of the DataFrame "df," which is a tuple representing the number of rows and columns in the DataFrame.

8. `df = df.iloc[:, [3, 4]].values`: This line selects specific columns (column indices 3 and 4) from the DataFrame "df" and converts them to a NumPy array, which is often used for numerical computations.

9. `plt.scatter(df[:,0], df[:,1], s=10, c= "blue")`: This line creates a scatter plot using Matplotlib, where the first column of the NumPy array is used as the x-axis, the second column as the y-axis, and data points are plotted as blue dots with a size of 10.

10. `from sklearn.cluster import KMeans`: This line imports the KMeans clustering algorithm from the scikit-learn (sklearn) library. KMeans is a popular method for clustering data.

11. `from sklearn.cluster import DBSCAN`: This line imports the DBSCAN (Density-Based Spatial Clustering of Applications with Noise) clustering algorithm from scikit-learn. DBSCAN is another clustering algorithm.

12. `dbscan = DBSCAN(eps=5, min_samples=5)`: This line creates an instance of the DBSCAN clustering algorithm with specific hyperparameters: "eps" is the maximum distance between two samples for one to be considered as in the neighborhood of the other, and "min_samples" is the number of samples in a neighborhood for a point to be considered a core point.

13. `labels = dbscan.fit_predict(df)`: This line applies the DBSCAN clustering algorithm to the data (the NumPy array "df") and assigns cluster labels to each data point. The labels are stored in the "labels" variable.

14. `np.unique(labels)`: This line uses NumPy to find the unique cluster labels assigned by the DBSCAN algorithm.

15. `plt.scatter(df[labels == -1, 0], df[labels == -1, 1], s = 10, c = 'black')`: This line plots data points belonging to the cluster labeled as -1 in black color with a size of 10.

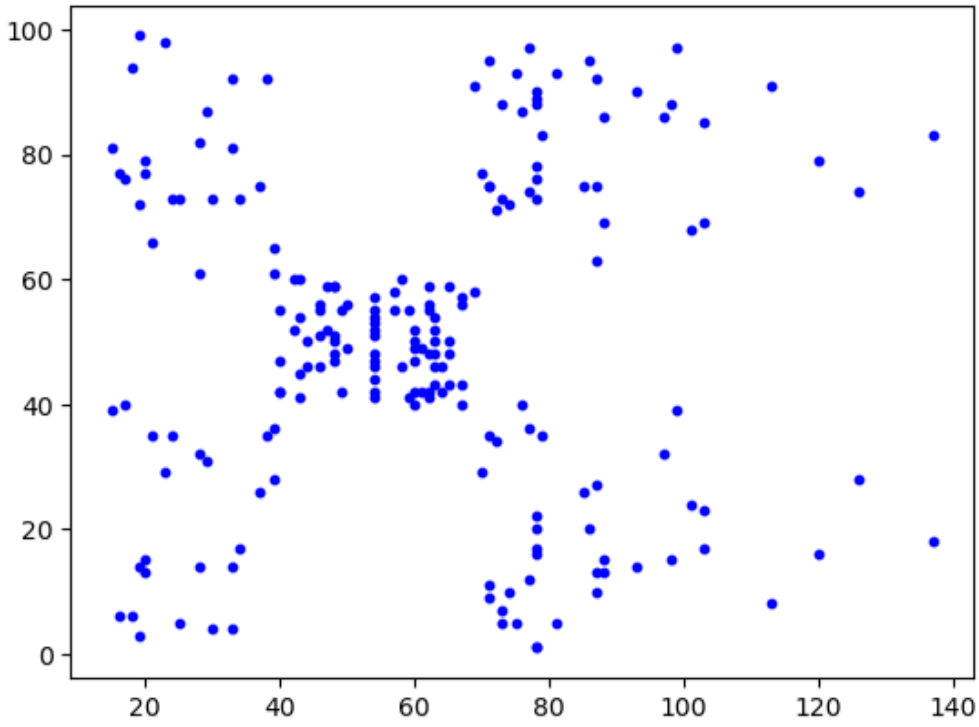
16. The next several lines (e.g., `plt.scatter(df[labels == 0, 0], df[labels == 0, 1], s = 10, c = 'purple')`, `plt.scatter(df[labels == 1, 0], df[labels == 1, 1], s = 10, c = 'red')`) plot data points belonging to different clusters with different colors.

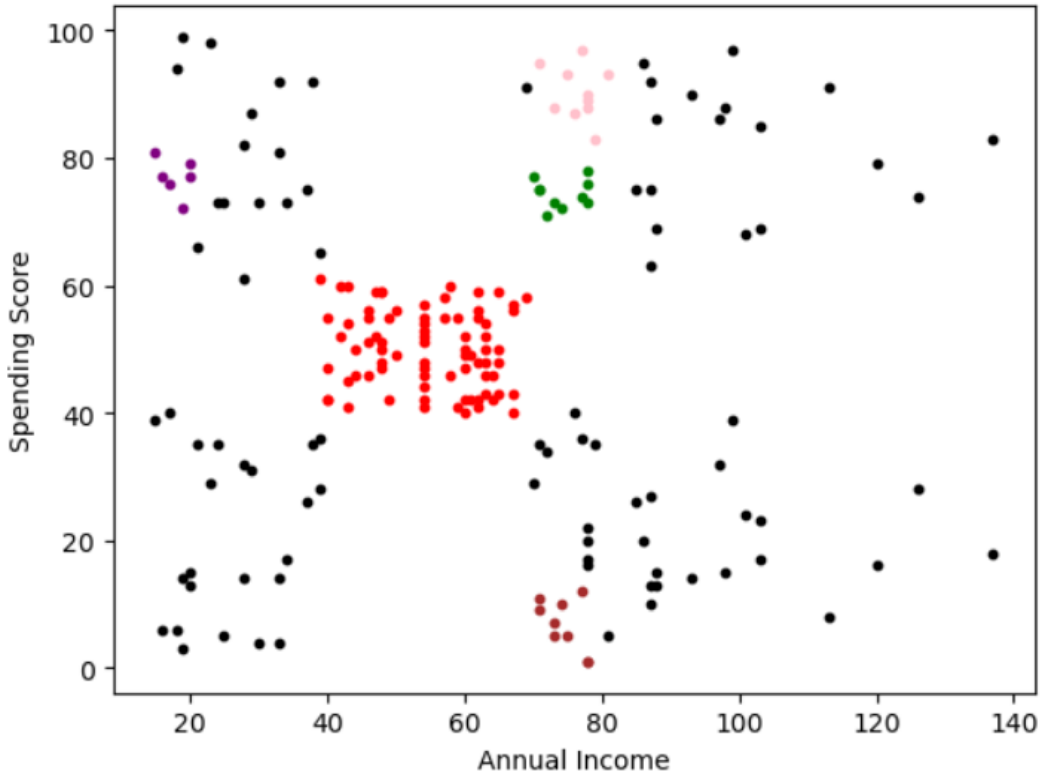
17. `plt.xlabel('Annual Income')`: This line sets the label for the x-axis of the plot as "Annual Income."

18. `plt.ylabel('Spending Score')`: This line sets the label for the y-axis of the plot as "Spending Score."

19. `plt.show()`: This line displays the final scatter plot with data points colored by their respective clusters based on the DBSCAN algorithm.

Result:


100
80
60
40
20
0
20
40
60
80
100
120
140


Spending Score
Annual Income
100
80
60
40
20
0
20
40
60
80
100
120
140

PART (D)

Github Link: https://github.com/dm-khalid/clustering