Unsupervised Learning - Exercise 1

Naama Avni 208523456

Q1

Part A

The function `q_1_a` create the required clusters based on the mentioned parameters using the function `generate_clusters_a`.

The function description:

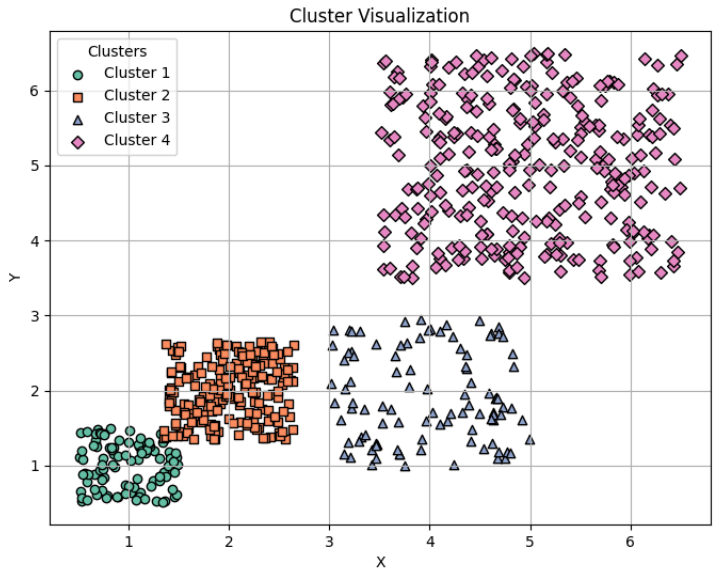
Creates M clusters of points based on Gaussian distribution.

Parameters:
- S (pd.Series): Series of size M with the number of points in each cluster.
- C (pd.DataFrame): DataFrame of size M x D representing the centers of the clusters.
- W (pd.Series): Series of size M with the width (standard deviation) of each cluster.

Returns:
- pd.DataFrame: DataFrame with D+1 columns, containing the coordinates of the points and the cluster number.

The code includes clear documentation and comments that explain the function's logic.

The required output:(also available in the question directory)



Part B

The function `q_1_b` create the required clusters based on the mentioned parameters using the function `generate_clusters_b.`

The function description:

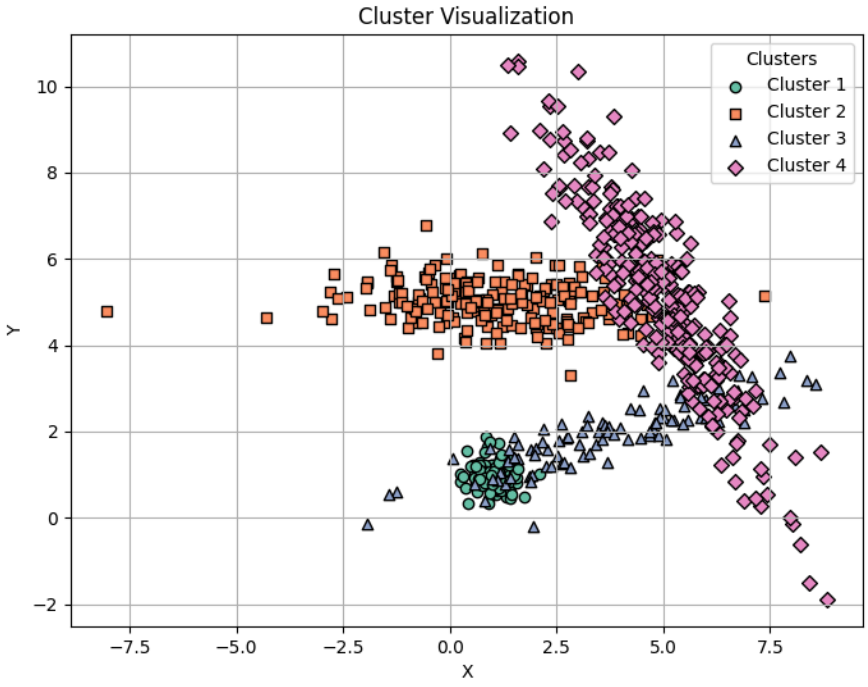
Creates M clusters in Gaussian distribution with the given parameters.

Parameters:
- S (pd.Series): Series of size M with the number of points in each cluster.
- C (pd.DataFrame): DataFrame of size M x (D + D^2), where the first D columns represent the cluster centers
  and the next D^2 columns represent the flattened covariance matrices.
- D (int): The number of dimensions for the clusters.

Returns:
- pd.DataFrame: DataFrame with D+1 columns (coordinates and cluster label).

The code includes clear documentation and comments that explain the function's logic.

The required output:(also available in the question directory)



Part C

The function `q_1_c` create the required clusters based on the mentioned parameters using the function `generate_clusters_c.`

The function description:

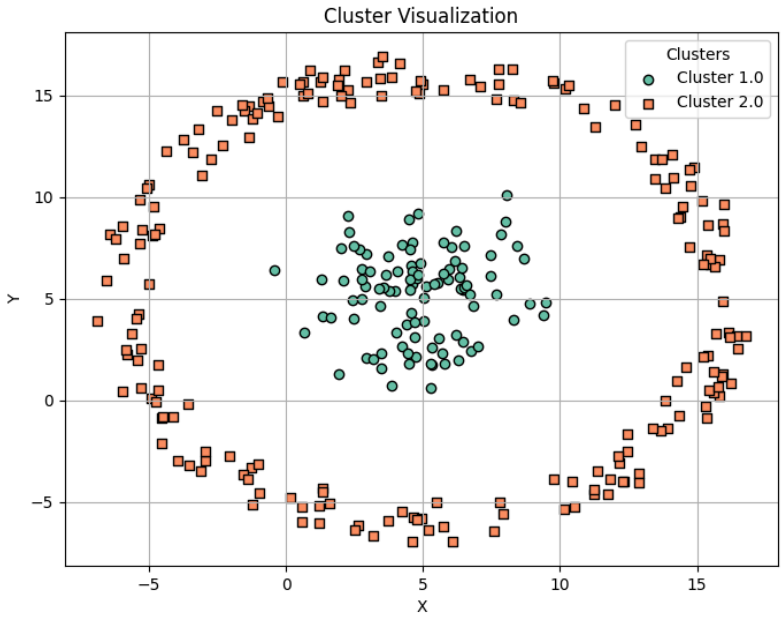
Generate a Gaussian cluster and a ring cluster around it.

Parameters:
- center: Tuple (x, y), center of both clusters
- sigma: Standard deviation (not variance) of the Gaussian cluster
- Ng: Number of points in the Gaussian cluster
- inner_radius: Inner radius of the ring cluster
- ring_width: Width of the ring
- Nr: Number of points in the ring cluster

Returns:
- A (Ng + Nr, 3) numpy array: [x, y, label]

The code includes clear documentation and comments that explain the function's logic.

The required output:(also available in the question directory)



Q2

The required logic implemented in the function `inner_cluster_similarity`

What the function does:

This function calculates how tight or compact each cluster is by measuring how close the points are to each other inside each cluster.

Step-by-step explanation:

1. Input:
    - A dataset with 2D points (as a pandas DataFrame), and
    - The cluster labels (which point belongs to which group).
2. For each cluster (based on the labels you gave):
    - It looks at all the points in that cluster,
    - Then, it calculates the distance between every possible pair of points in that cluster,
    - And it adds up all those distances.
3. Finally, it adds up the distances from all clusters, and returns that total.
4. Output:
    - A single number that tells you how "spread out" your clusters are in total.
      A lower number means the clusters are tighter and more compact (which is usually better).

Q3

This script runs **K-Means clustering** on three different datasets using **three initialization strategies**:

- Random initialization
- Farthest-point initialization
- Initialization based on DBSCAN

Then, for each result:

- It calculates how tight the clusters are (using inner-cluster similarity),
- It plots the clusters, and
- It shows how clustering quality changes as the number of clusters `k` increases.

`k_means()` Function

This function does the actual **K-Means clustering**:

- It can start with **random centers**, **farthest points**, or **cluster centers estimated by DBSCAN**.
- It loops, assigning points to the closest center and updating those centers, until it either converges or hits the iteration limit.
- It returns the cluster **labels** (which cluster each point belongs to).

`plot_similarities_scores()` Function

- This ta**kes a list of similarity scores (how tightly packed each cluster is) for each k.**
- **It prints** them and plots a line graph of **similarity vs number of clusters (`k`)**.

`main` Block

This part runs the full clustering analysis when the script is executed:
For each dataset:

1. It runs K-Means **7 times**, with `k=1 to 7`, using:
    - **Random initialization**
    - **Farthest-point initialization**
2. For each run:
    - It clusters the data.
    - Plots the results.
    - Calculates and stores the **inner-cluster similarity** (a score showing how compact the clusters are).

3. Then it plots how similarity changes with `k` (for both init strategies).
4. After that, it runs **K-Means initialized by DBSCAN** (which auto-detects k) and plots the result.

The text output available in the appendix, and all plots available in the question directory.

Q4

PAM Function

This implements **PAM (Partitioning Around Medoids)**:

1. It randomly picks `k` medoids (representative data points).
2. Then, it assigns each data point to the nearest medoid.
3. After that, for each cluster, it tries to find a better medoid (the most central point in the cluster).
4. It keeps updating medoids until they stop changing.
5. Returns: the medoids and the cluster assignments (labels).

CLARA Support Function: `pam_on_subset`

This runs PAM **on a small sample** of the full data (used inside CLARA). Once it finds medoids on the sample, it:

- Assigns **all** points in the full dataset to the closest medoid.
- Calculates the **total cost** (how far points are from their medoids).

CLARA Function

This implements **CLARA (Clustering Large Applications)**:

1. It runs `pam_on_subset` multiple times (on different random samples).

2. It keeps track of the best result (lowest cost).
3. Finally, it returns the medoids and labels from the best sample.

Main Script

1. Load 3 datasets from `Q1`:

They're likely different cluster structures to test on.

2. Run PAM:

- For each dataset:
    - Try `k = 1` to `7` clusters
    - Run PAM
    - Measure how tightly packed the clusters are (**inner-cluster similarity**)
    - Plot the clusters
    - Plot how similarity changes as `k` increases
- Also, measure total **execution time** for PAM.

3. Run CLARA:

- Same process as PAM, but using the **CLARA** method (more scalable)
- Again, measure similarity and runtime

4. Compare Runtime:

Print how long PAM and CLARA took so you can compare speed.

The text output available in the appendix, and all plots available in the question directory.

Appendix:

Q3 text output:

```
Running K-means clustering on datasets:
Running K-means with k=1 on dataset 1 on random initialization
Running K-means with k=1 on dataset 1 on farthest initialization
Running K-means with k=2 on dataset 1 on random initialization
Running K-means with k=2 on dataset 1 on farthest initialization
Running K-means with k=3 on dataset 1 on random initialization
Running K-means with k=3 on dataset 1 on farthest initialization
Running K-means with k=4 on dataset 1 on random initialization
Running K-means with k=4 on dataset 1 on farthest initialization
Running K-means with k=5 on dataset 1 on random initialization
Running K-means with k=5 on dataset 1 on farthest initialization
Running K-means with k=6 on dataset 1 on random initialization
Running K-means with k=6 on dataset 1 on farthest initialization
Running K-means with k=7 on dataset 1 on random initialization
Running K-means with k=7 on dataset 1 on farthest initialization

Similarity scores:, dataset 1, K-Means Random Initialization
  k=1: similarity = 721185.6546
  k=2: similarity = 189135.5070
  k=3: similarity = 116775.6080
  k=4: similarity = 86600.9415
  k=5: similarity = 46892.0848
  k=6: similarity = 39497.4418
  k=7: similarity = 36033.5754

Similarity scores:, dataset 1, K-Means Farthest Initialization
  k=1: similarity = 721185.6546
  k=2: similarity = 189135.5070
  k=3: similarity = 116775.6080
  k=4: similarity = 77081.1478
  k=5: similarity = 64878.0908
  k=6: similarity = 34703.4243
  k=7: similarity = 30794.6381

Running K-means with DBSCAN initialization on dataset 1

Running K-means with k=1 on dataset 2 on random initialization
Running K-means with k=1 on dataset 2 on farthest initialization
Running K-means with k=2 on dataset 2 on random initialization
```

Running K-means with k=2 on dataset 2 on farthest initialization
Running K-means with k=3 on dataset 2 on random initialization
Running K-means with k=3 on dataset 2 on farthest initialization
Running K-means with k=4 on dataset 2 on random initialization
Running K-means with k=4 on dataset 2 on farthest initialization
Running K-means with k=5 on dataset 2 on random initialization
Running K-means with k=5 on dataset 2 on farthest initialization
Running K-means with k=6 on dataset 2 on random initialization
Running K-means with k=6 on dataset 2 on farthest initialization
Running K-means with k=7 on dataset 2 on random initialization
Running K-means with k=7 on dataset 2 on farthest initialization

Similarity scores:, dataset 2, K-Means Random Initialization
  k=1: similarity = 1035579.5350
  k=2: similarity = 381920.3211
  k=3: similarity = 218408.4098
  k=4: similarity = 113221.7302
  k=5: similarity = 78842.6497
  k=6: similarity = 59961.4275
  k=7: similarity = 53400.1135

Similarity scores:, dataset 2, K-Means Farthest Initialization
  k=1: similarity = 1035579.5350
  k=2: similarity = 381920.3211
  k=3: similarity = 198189.2247
  k=4: similarity = 113105.3801
  k=5: similarity = 78987.8955
  k=6: similarity = 66451.2130
  k=7: similarity = 48934.3383

Running K-means with DBSCAN initialization on dataset 2

Running K-means with k=1 on dataset 3 on random initialization
Running K-means with k=1 on dataset 3 on farthest initialization
Running K-means with k=2 on dataset 3 on random initialization
Running K-means with k=2 on dataset 3 on farthest initialization
Running K-means with k=3 on dataset 3 on random initialization
Running K-means with k=3 on dataset 3 on farthest initialization
Running K-means with k=4 on dataset 3 on random initialization
Running K-means with k=4 on dataset 3 on farthest initialization
Running K-means with k=5 on dataset 3 on random initialization
Running K-means with k=5 on dataset 3 on farthest initialization
Running K-means with k=6 on dataset 3 on random initialization
Running K-means with k=6 on dataset 3 on farthest initialization

Running K-means with k=7 on dataset 3 on random initialization
Running K-means with k=7 on dataset 3 on farthest initialization

Similarity scores:, dataset 3, K-Means Random Initialization
  k=1: similarity = 521434.6253
  k=2: similarity = 221197.0661
  k=3: similarity = 118762.6807
  k=4: similarity = 81545.6783
  k=5: similarity = 43346.6120
  k=6: similarity = 34532.2046
  k=7: similarity = 30022.8133

Similarity scores:, dataset 3, K-Means Farthest Initialization
  k=1: similarity = 521434.6253
  k=2: similarity = 212046.7813
  k=3: similarity = 118551.4926
  k=4: similarity = 76837.3451
  k=5: similarity = 43092.1169
  k=6: similarity = 33563.6705
  k=7: similarity = 29399.4981

Running K-means with DBSCAN initialization on dataset 3

Q4 text output:

Running PAM with k=1 on dataset 1
Running PAM with k=2 on dataset 1
Running PAM with k=3 on dataset 1
Running PAM with k=4 on dataset 1
Running PAM with k=5 on dataset 1
Running PAM with k=6 on dataset 1
Running PAM with k=7 on dataset 1

Similarity scores:
  k=1: similarity = 731943.2026
  k=2: similarity = 195106.1767
  k=3: similarity = 120087.9554
  k=4: similarity = 89461.0936
  k=5: similarity = 48168.3289
  k=6: similarity = 42226.3253
  k=7: similarity = 37129.1750
Running PAM with k=1 on dataset 2
Running PAM with k=2 on dataset 2
Running PAM with k=3 on dataset 2
Running PAM with k=4 on dataset 2
Running PAM with k=5 on dataset 2
Running PAM with k=6 on dataset 2
Running PAM with k=7 on dataset 2

Similarity scores:
  k=1: similarity = 1017308.7599
  k=2: similarity = 373919.3937
  k=3: similarity = 203557.5116
  k=4: similarity = 166211.1378
  k=5: similarity = 81255.8152
  k=6: similarity = 60539.8269
  k=7: similarity = 47412.6549
Running PAM with k=1 on dataset 3
Running PAM with k=2 on dataset 3
Running PAM with k=3 on dataset 3
Running PAM with k=4 on dataset 3
Running PAM with k=5 on dataset 3
Running PAM with k=6 on dataset 3
Running PAM with k=7 on dataset 3

Similarity scores:
  k=1: similarity = 519762.3578
  k=2: similarity = 257384.7126
  k=3: similarity = 135736.5703
  k=4: similarity = 71605.9053
  k=5: similarity = 42228.2342
  k=6: similarity = 33280.2697
  k=7: similarity = 29258.1144
Running CLARA with k=1 on dataset 1
Running CLARA with k=2 on dataset 1
Running CLARA with k=3 on dataset 1

Running CLARA with k=4 on dataset 1
Running CLARA with k=5 on dataset 1
Running CLARA with k=6 on dataset 1
Running CLARA with k=7 on dataset 1

Similarity scores:
  k=1: similarity = 731943.2026
  k=2: similarity = 195447.0053
  k=3: similarity = 118504.7472
  k=4: similarity = 93587.9922
  k=5: similarity = 51159.7702
  k=6: similarity = 59778.3495
  k=7: similarity = 31465.6946
Running CLARA with k=1 on dataset 2
Running CLARA with k=2 on dataset 2
Running CLARA with k=3 on dataset 2
Running CLARA with k=4 on dataset 2
Running CLARA with k=5 on dataset 2
Running CLARA with k=6 on dataset 2
Running CLARA with k=7 on dataset 2

Similarity scores:
  k=1: similarity = 1017308.7599
  k=2: similarity = 436305.4798
  k=3: similarity = 236751.0609
  k=4: similarity = 113937.5717
  k=5: similarity = 88459.1417
  k=6: similarity = 69773.7235
  k=7: similarity = 51433.7902
Running CLARA with k=1 on dataset 3
Running CLARA with k=2 on dataset 3
Running CLARA with k=3 on dataset 3
Running CLARA with k=4 on dataset 3
Running CLARA with k=5 on dataset 3
Running CLARA with k=6 on dataset 3
Running CLARA with k=7 on dataset 3

Similarity scores:
  k=1: similarity = 519762.3578
  k=2: similarity = 268119.2827
  k=3: similarity = 161653.5703
  k=4: similarity = 72158.3275
  k=5: similarity = 43972.0287
  k=6: similarity = 35684.6053

```
  k=7: similarity = 26426.0821
PAM runtime:   18.5664 seconds
CLARA runtime: 12.5892 seconds
```