Exploratory Data Analysis

Mraihi Abdelhakim

Task1

a) K-Means Clustering on Full Dataset

There is 150 samples and 4 features in the Iris dataset: sepal length, sepal width, petal length, and petal width, across three species: setosa, versicolor, and virginica.

- **The Clustering**: The dataset was clustered into three groups using the K-means algorithm, assuming $k = 3$ for the three species.
- **Evaluation**: The clusters were mapped to the actual labels by identifying the most frequent true label in each cluster. The accuracy of clustering was then computed by comparing these mapped labels with the true labels.
- **Accuracy on Full Dataset**: Classification accuracy is **88.67%**.

b) Factor Analysis and Clustering on Reduced Data

After reducing the Iris dataset to 2D using Factor Analysis (FA), k-means clustering was applied:

- **Factor Analysis**: Reduced the dimensionality from 4 to 2.
- **Evaluation and Visualization**: For visualization, a scatter plot of FA components with true labels was created. The accuracy was calculated similarly by mapping clusters to actual labels. Additionally, the reduced dataset was visualized to show the clustering results alongside the original labels.
- **Accuracy on Reduced Data**: The accuracy after dimension reduction and clustering was **66.67%**.

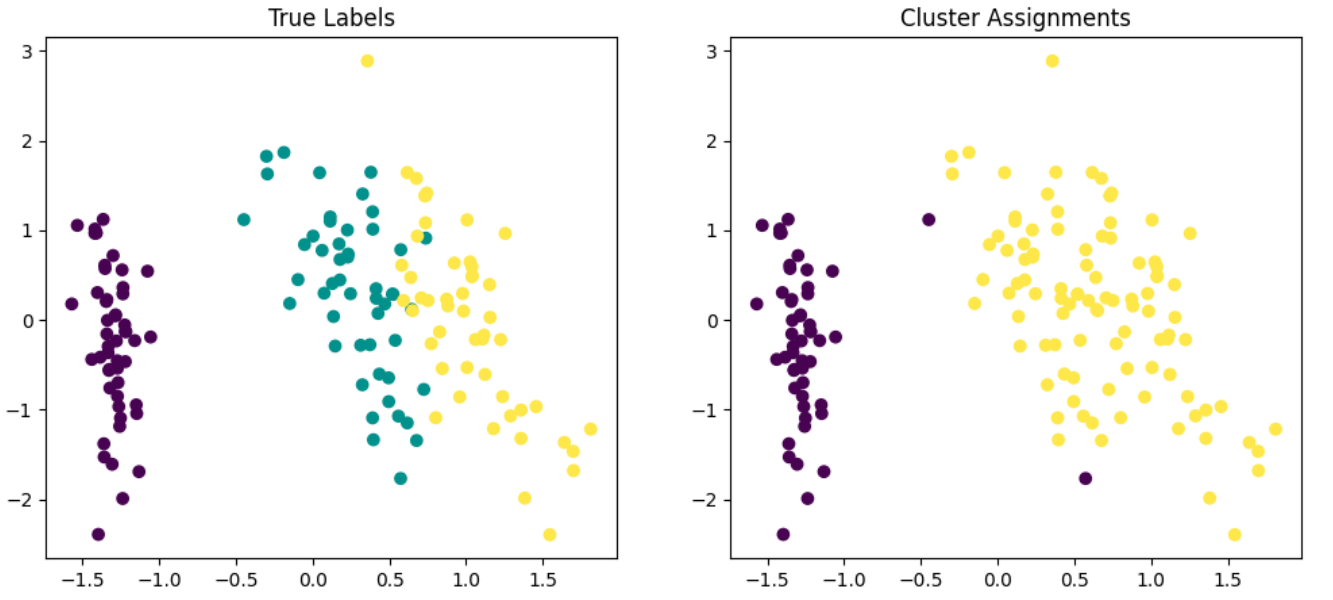


Figure 1: Visualization of the clustered data.

Task2

a)

To tackle the task using the MST method for clustering the lung cancer dataset (LungA), we'll proceed with a step-by-step approach:

Since the DS is stored in a .mat file. I used scipy.io to load it in jupyter lab. The LungA dataset contains 3312 genes (features) and 203 samples (columns).Labeling: Each sample is labeled as one of the following: AD (Adenocarcinoma), SQ (Squamous cell carcinoma), COID (Other carcinoma), S MCL (Small-cell lung carcinoma), and NL (Normal lung tissue). Task Specifications: We are asked to classify into two clusters: Normal (NL) and Cancer (everything else: AD, SQ, COID, SMCL). for that; i converted the labels into binary form: 0 = Normal, 1 = Cancer. Processi: I transposed the data so that rows are samples and columns are gene features. and built a Minimum Spanning Tree (MST) using pairwise distances. Clustering and Evaluation: I cut the MST to form two clusters and finally Compare the cluster labels to ground truth and compute accuracy. The accuracy of correctly classifying samples as Normal vs. Cancer is: **91.63%** This high accuracy indicates that MST-based clustering is quite effective at separating cancerous tissue from normal tissue using gene expression data.

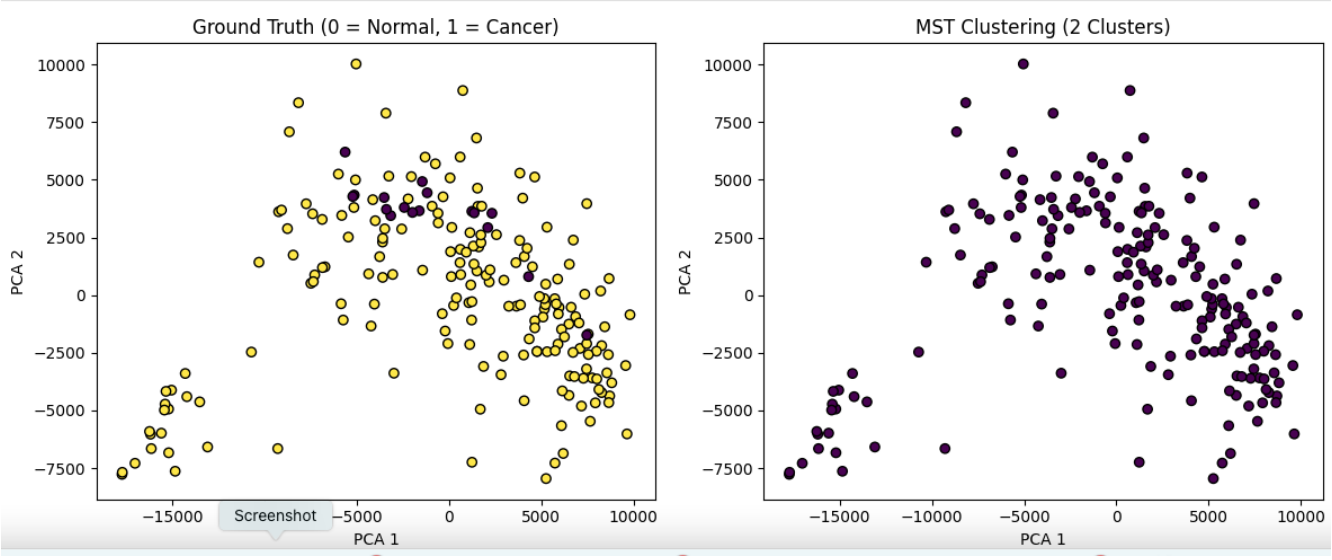


Figure 2: MST clustering

b)

After applying the Minimum Spanning Tree (MST) method to find three clusters and mapping these clusters to the original labels (Normal, Nonsmall cell lung carcinomas, and Small-cell lung carcinomas), the accuracy of correctly clustered genes is approximately **88.67%**

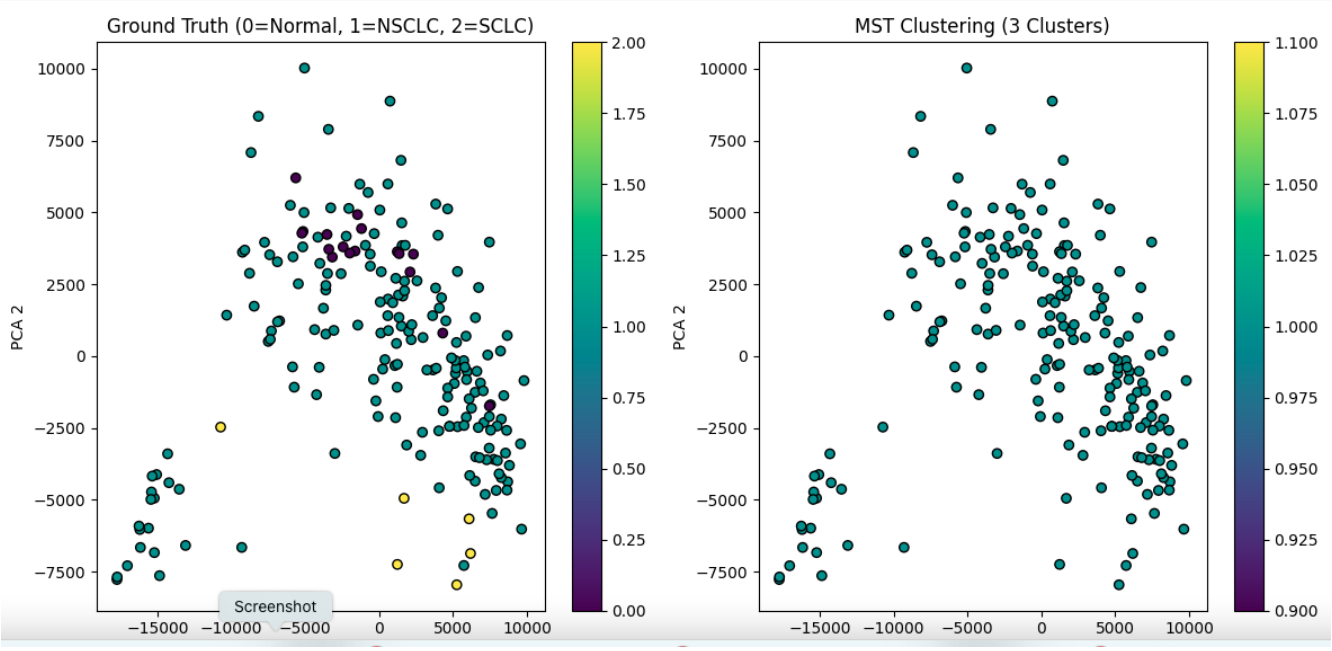


Figure 3: MST Clustering (3 Clusters)

c)

After applying the Minimum Spanning Tree (MST) method to find five clusters and comparing these clusters with the specific labels (AD, COID, SQ, NL, SCLC), the accuracy of correctly clustered genes is approximately **68.47%**

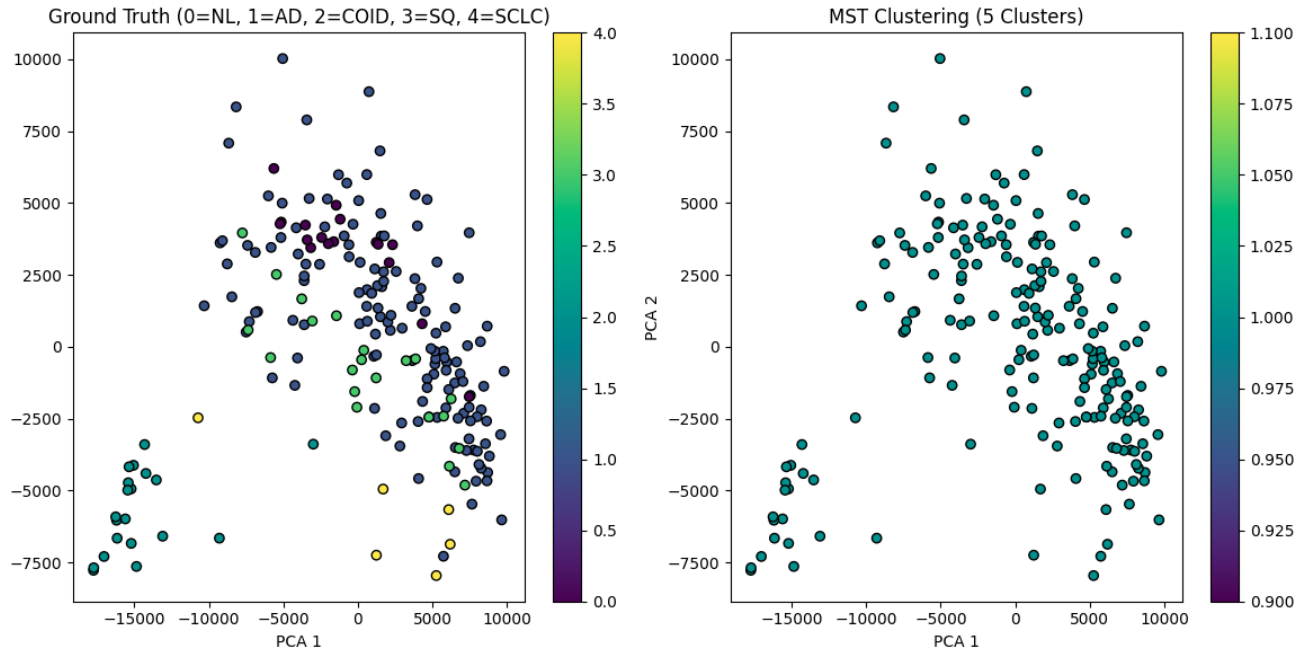


Figure 4: 5 clusters

General Observations:

- Increasing Complexity: As we move from two clusters to five clusters, the accuracy of MST-based clustering decreases. This is a common pattern when clustering data into more categories—while two broad categories

like Normal vs. Cancer are easier to separate, finer distinctions between subtypes (such as NSCLC vs. SCLC vs. AD) introduce more complexity.

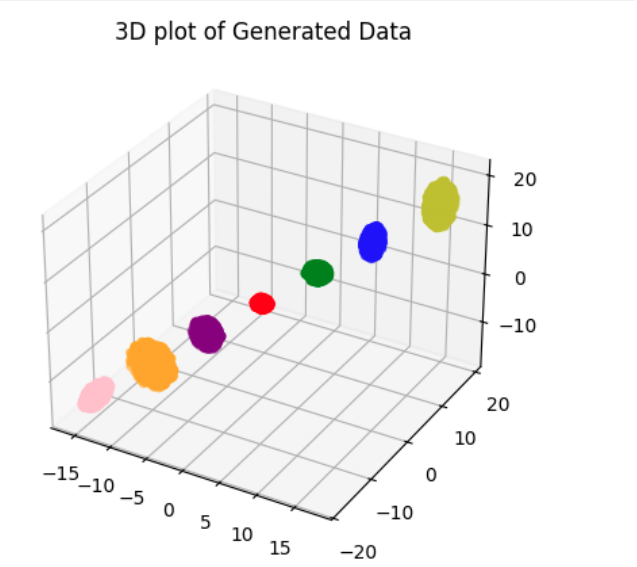
- MST's Sensitivity: MST works well for separating data that has clear distinctions, but when clusters overlap or are very close together (as in the case of the five clusters), the method may struggle to maintain high accuracy.
- These results highlight the importance of considering the level of granularity for clustering. While MST was very effective at the two-cluster level, the accuracy decrease as we moved to more clusters suggests that the data may benefit from dimensionality reduction or a different clustering approach.
- Visualization and Overlap: The PCA plots in each case help us visually inspect how well the data separates into clusters. As the number of clusters increases, the data points become more scattered, which aligns with the drop in accuracy. The visual overlap between certain clusters indicates that gene expression profiles for these subtypes are similar.
- In summary, while MST performed well for two clusters, its effectiveness decreased with more clusters, revealing the challenges of capturing finer distinctions in gene expression data. Further tuning or alternative clustering methods might help improve the results at higher levels of granularity.

Task3

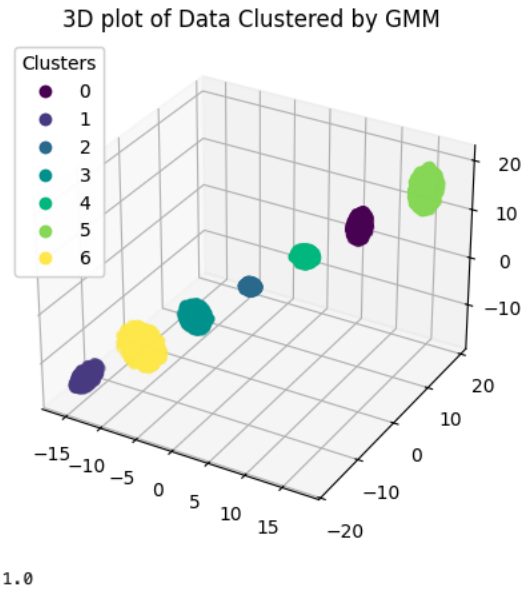
a)

For this TaskFirst i generate three types of datasets as specified: 2 balls of different sizes,2 ellipsoids of different sizes with axes parallel to the coordinate axes. and 3 ellipsoids of different sizes with arbitrary symmetry axes and position them so they are far enough apart to be considered disjoint, later im going to apply GMM .

Below is the 3D plot of the generated data for the model-based clustering task. There is: 2 balls (red and green) of different sizes, 2 ellipsoids (blue and yellow) of different sizes with axes parallel to the coordinate axes. and 3 ellipsoids (purple, orange, and pink) with arbitrary axes and different sizes.



Next, I'll apply model-based clustering using Gaussian Mixture Models (GMM) to identify clusters :



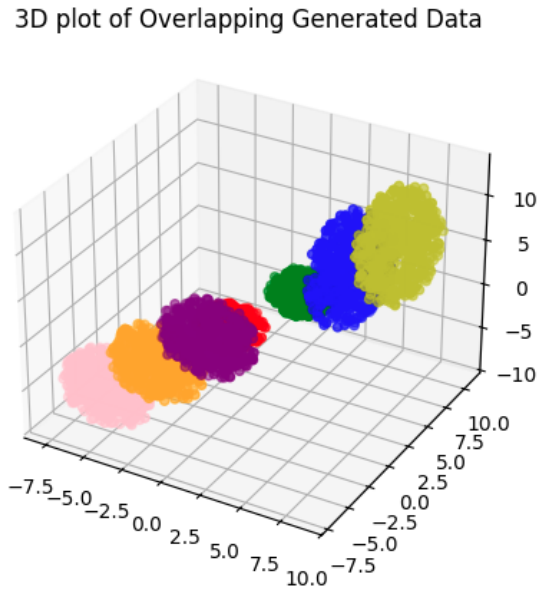
The 3D plot above displays the clusters identified by the Gaussian Mixture Model (GMM), with each color representing a different cluster. The GMM successfully distinguished the seven groups we originally generated.

Additionally, the Adjusted Rand Index (ARI) for this clustering is 1.0, indicating a perfect match between the clusters found by the GMM and the original labels. This suggests that model-based clustering effectively identified the geometrical shapes in the data as separate clusters.

b)

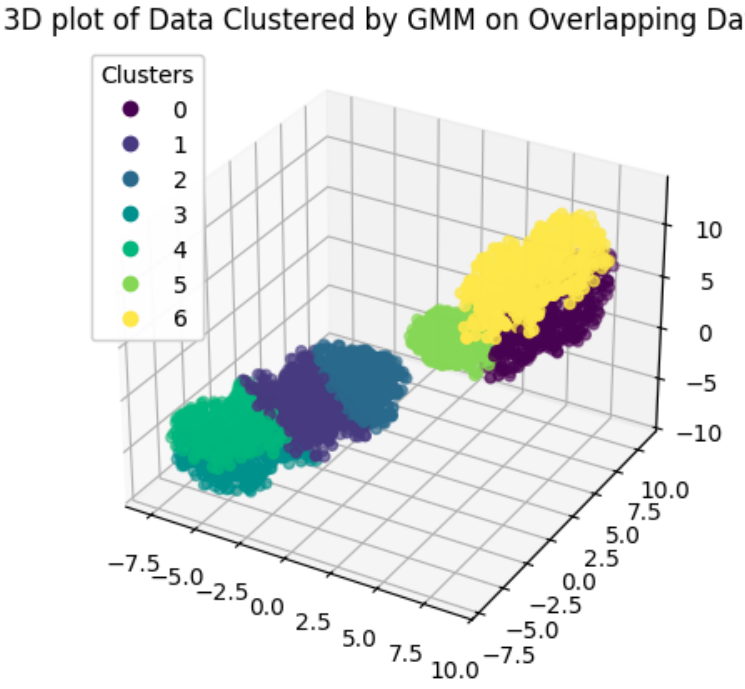
In this task, we are going to generate Overlapping Data by adjusting the centers and sizes of the balls and ellipsoids so that they overlap partially and later applying a Model-Based Clustering: by using Gaussian Mixture Models (GMM) again to identify clusters in this overlapping scenario.

First I'll update the centers and sizes (in the code) for overlap and then visualize the result:



Here's the 3D plot : it includes two balls and five ellipsoids, with modifications to their centers and sizes to ensure some degree of overlap.

Next, I'll apply the Gaussian Mixture Model (GMM) for clustering this overlapping data and then evaluate the performance by comparing the estimated clusters with the original labels:



[40]: 0.5275965033077729

The clustering results on the overlapping data are visualized above:

The Adjusted Rand Index (ARI) for this overlapping data scenario is approximately 0.5275 While not perfect (as in the non-overlapping case), this score still indicates a moderate level of agreement between the original labels and the clusters identified by GMM. The decrease in ARI compared to the non-overlapping scenario reflects the added complexity and ambiguity introduced by the overlapping regions.

This outcome demonstrates that while GMM can still effectively identify distinct groups within the data, the presence of overlaps does pose challenges and may lead to less distinct clustering results.

Task4

a)

- The unemployment data for EU countries was sourced from the Eurostat database:`https://ec.europa.eu/eurostat/databrowser/view/une_rt_a/default/table?lang=en&category=labour.employ.lfsi.une`
- Inflation rate data for EU countries : `https://ec.europa.eu/eurostat/databrowser/view/tec00118/default/table?lang=en`

I will proceed by preparing the data in a suitable format for clustering and then perform hierarchical clustering. First, i will pivot the data to have countries as rows and years as columns, which is suitable for time series clustering.

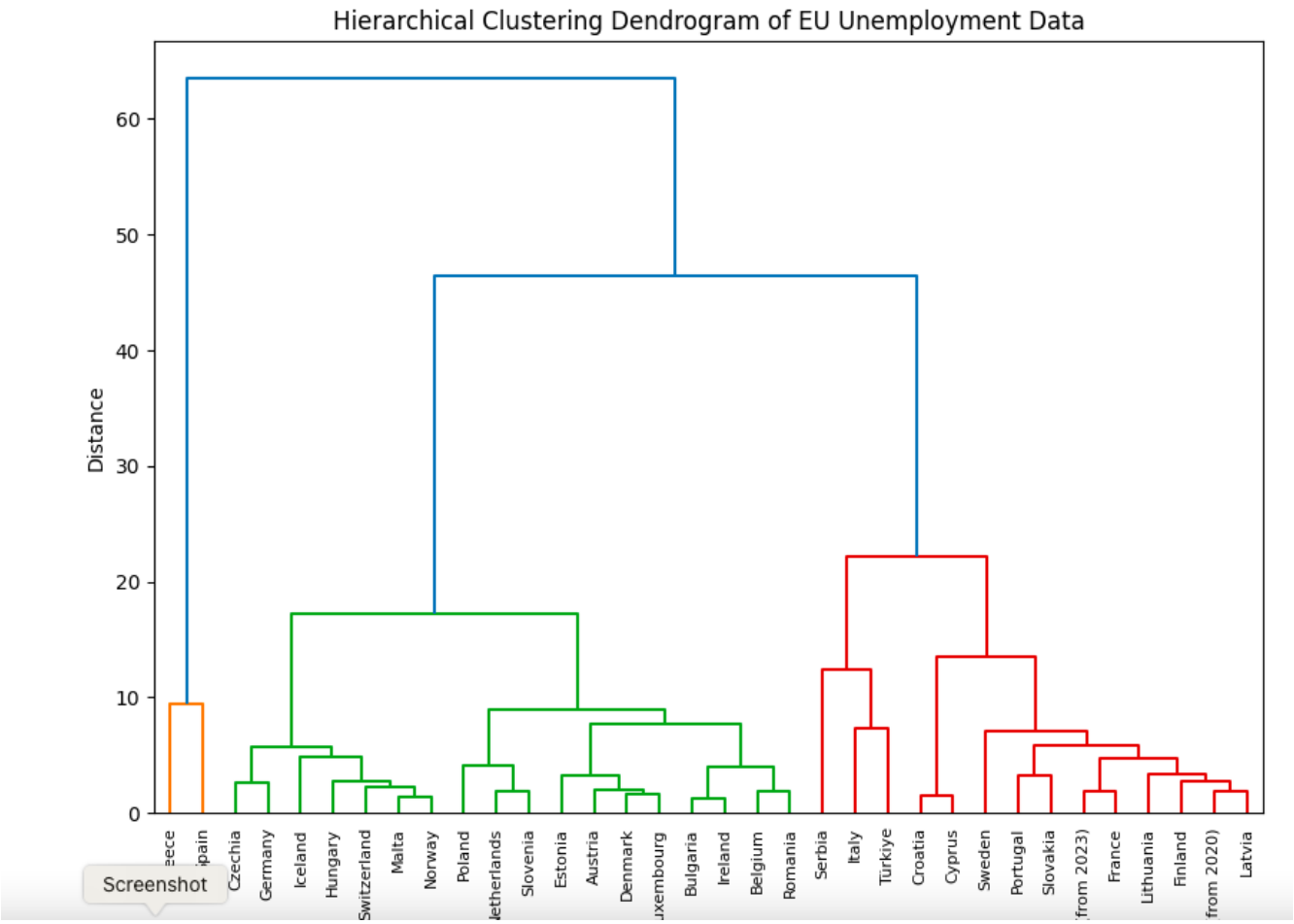
```
<class 'pandas.core.frame.DataFrame'>
Index: 34 entries, Austria to Türkiye
Data columns (total 10 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   2015    34 non-null     float64
 1   2016    34 non-null     float64
 2   2017    34 non-null     float64
 3   2018    34 non-null     float64
 4   2019    34 non-null     float64
 5   2020    34 non-null     float64
 6   2021    34 non-null     float64
 7   2022    34 non-null     float64
 8   2023    34 non-null     float64
 9   2024    34 non-null     float64
dtypes: float64(10)
memory usage: 2.9+ KB
```

[6]:

```
(None,
 TIME_PERIOD  2015  2016  2017  2018  2019  2020  2021  2022  2023  2024
 geo
 Austria       6.1   6.5   5.9   5.2   4.8   6.0   6.2   4.8   5.1   5.2
 Belgium       8.7   7.9   7.2   6.0   5.5   5.8   6.3   5.6   5.5   5.7
 Bulgaria     10.1   8.6   7.2   6.2   5.2   6.1   5.2   4.2   4.3   4.2
 Croatia      16.2  13.0  11.1   8.3   6.6   7.4   7.5   6.8   6.1   5.0
 Cyprus       15.0  13.0  11.1   8.4   7.2   7.6   7.2   6.3   5.8   4.9)
```

The data has been successfully pivoted and now includes annual unemployment rates for 34 countries from 2015 to 2024. Next, I will perform hierarchical clustering on this dataset to identify patterns in unemployment across these countries over the specified years.

After clustering, I'll generate a Silhouette plot to assess the quality of the clusters formed. Let's proceed with hierarchical clustering :



I selected 5 clusters to illustrate the clustering process. The resulting Silhouette score for this configuration is approximately 0.352. This score, while not particularly high, suggests a moderate level of separation between the clusters. It indicates that there are some distinctions between the groups, but the boundaries between some clusters might not be very clear or might contain mixed patterns. **Major Clusters:**

- Green Cluster: Includes countries like Greece, Spain, and Portugal among others, suggesting a higher unemployment rate .
- Red Cluster: Contains countries like Italy, Croatia, Cyprus, suggesting a medium range of unemployment rates.
- Blue Cluster: This cluster seems to include countries like Sweden, Finland, and Lithuania, which typically have lower or different economic characteristics affecting unemployment.
- The large blue cluster that stands alone at the top of the dendrogram indicates a significant dissimilarity with other countries in terms of unemployment metrics.

  The division into clusters suggests varying economic conditions, labor market policies, and perhaps different economic shocks affecting these regions.

b)

Fir this task, I'll perform the following steps:

1- Loading and Prepare the Data: Loading both datasets and prepare them for analysis by ensuring they're properly aligned and combined into a single dataset.

2- Applying k-means clustering to the combined dataset.

3- Dunn Index Evaluation: at the end i will be computing the Dunn index to evaluate the clustering quality.

For that we need to:

- Align the Data: Ensure that both datasets are filtered for the same countries and years, and then merge them based on these dimensions.
- Prepare the Combined Dataset: Create a dataset that includes both unemployment and inflation rates.

2019_unemployment	2020_unemployment	2021_unemployment	2022_unemployment	2023_unemployment	2024_unemployment
4.8	6.0	6.2	4.8	5.1	5.2
5.5	5.8	6.3	5.6	5.5	5.7
5.2	6.1	5.2	4.2	4.3	4.2
6.6	7.4	7.5	6.8	6.1	5.0
7.2	7.6	7.2	6.3	5.8	4.9

Figure 5: Combined data

The plot below visualizes the clustering of EU data, combining unemployment and inflation, using PCA and k-means clustering :

```
plt.show()
```

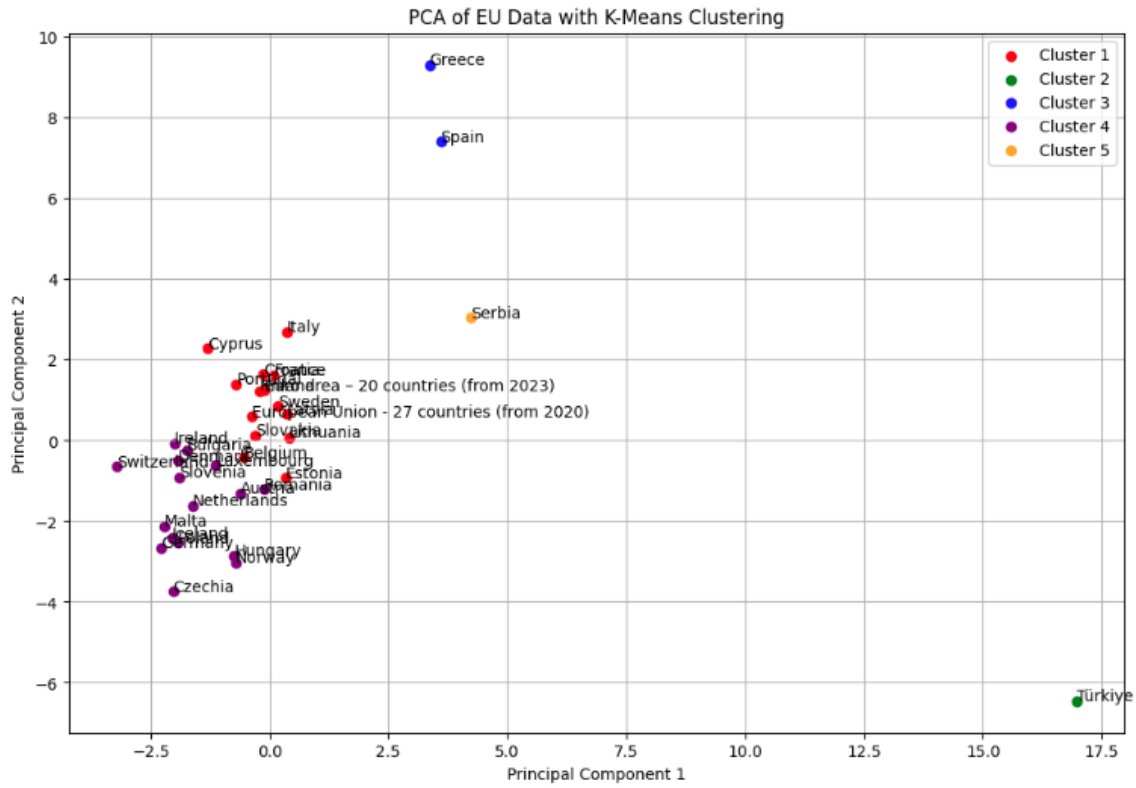
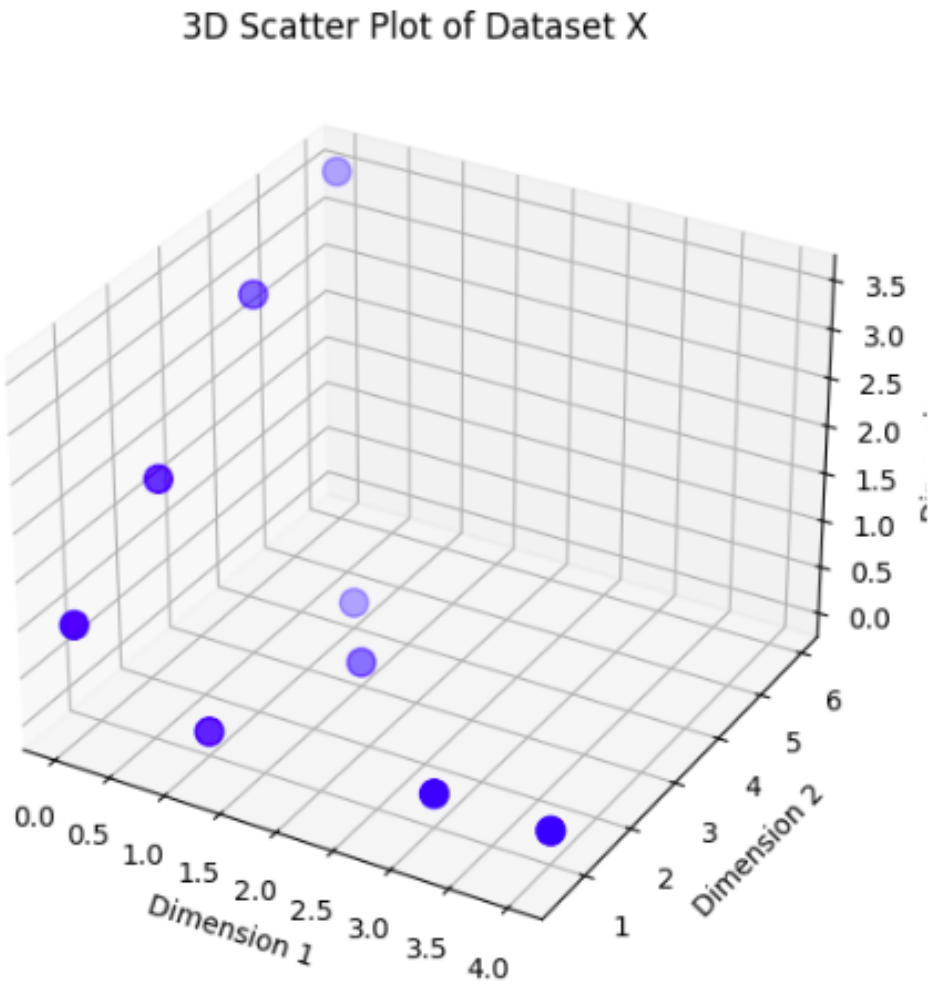


Figure 6: unemployment and inflation in the EU

The k-means clustering has been applied to the combined dataset of EU inflation and unemployment rates. The Dunn index for the clustering result is approximately 0.189(the compactness). Thus, the value of approximately 0.189 suggests moderate separation but indicates that clusters might be overlapping or not distinctly apart.

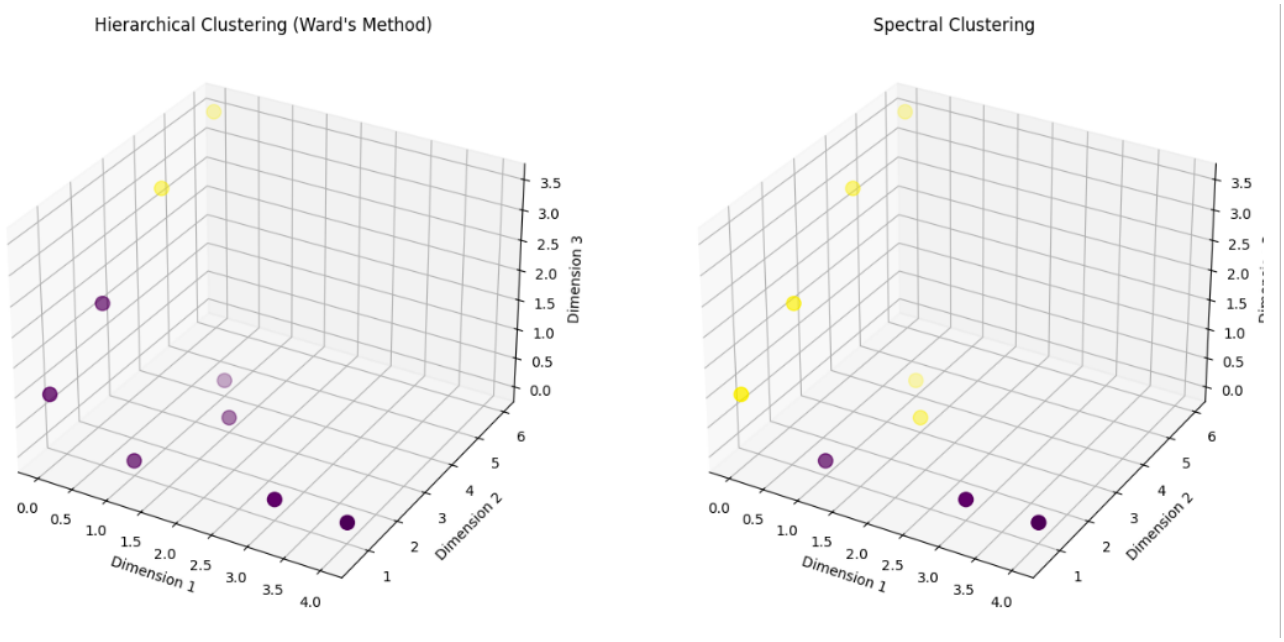
Task5

a)



b)

For this question i will apply both methods (using python) to the dataset X and visualize the results to see how well each method performs in identifying the two described clusters:



- Hierarchical Clustering (Ward's Method) appears to have successfully identified two distinct clusters, potentially matching the groups described (one on the xy-plane and the other on the yz-plane).
- Spectral Clustering also seems to delineate two clusters effectively, showing its capability to identify complex patterns, which might be based on the neighborhood graph constructed from the 3 nearest neighbors.

These visualizations suggest that both methods may be effective in recognizing the two clusters you hypothesized exist within the data.

Task6

a)

For this Taskwe need to perform the Non-negative Matrix Factorization of the matrix X using the Multiplicative Update Algorithm. I assumed a rank-2 factorization for illustration (i.e., W is n×2 and H is 2×m), which seems reasonable given the description of the data likely containing two clusters. I implimented this in python and found :

```
(array([[4.72098143e-01, 3.17393916e-01],
        [3.45178839e-01, 8.46028048e-02],
        [4.29989585e-01, 3.19075989e-01],
        [5.78434792e-01, 3.57746981e-01],
        [5.20249888e-01, 2.81260927e-01],
        [6.84987269e-01, 2.41964624e-01],
        [2.20587455e-03, 4.64694278e-01],
        [4.15825680e-01, 2.15994359e-01],
        [9.06724702e-01, 1.92817715e-04],
        [3.99897209e-01, 5.47496712e-01]]),
 array([[1.08766535, 0.13786585, 0.08039983],
        [0.58020272, 1.60866269, 1.03832275]]))
```

Figure 7: W and H

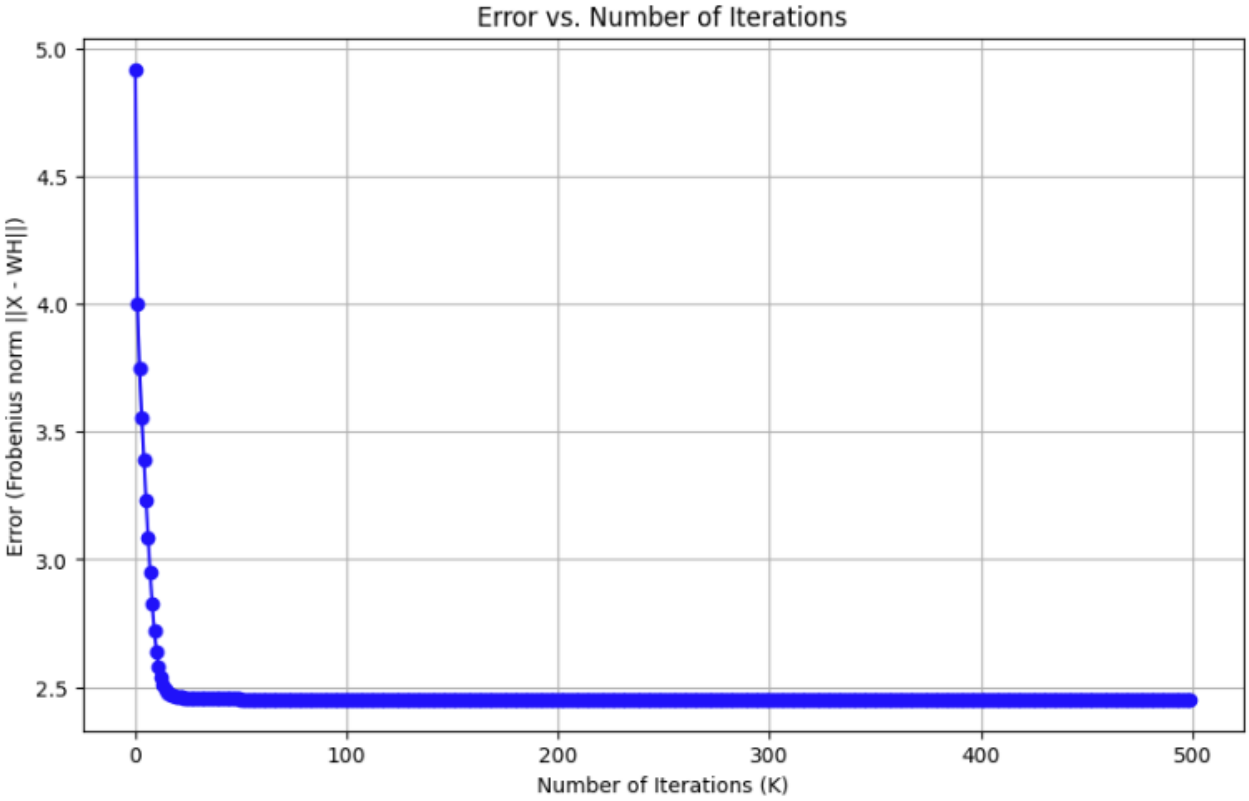
The solver='mu' parameter in the provided algorithm specifically sets the solver to use **the multiplicative update algorithm**.

b)

Here, I will implement a custom loop to perform the NMF using the Multiplicative Update rules and track the error at each iteration until the maximum number of iterations K is reached. In the python file is the implementation using *numpy* for matrix operations.

- Initialization: W and H are initialized randomly.
- Updates: The H and W matrices are updated using the Multiplicative Update rules.
- Error Calculation: The Frobenius norm $\|X - WH\|$ is calculated after each update to monitor convergence.
- Convergence Criteria: The loop can break early if the error goes below a certain threshold, which is useful for ensuring that we don't continue iterating when unnecessary.

i)



It appears that the error quickly decreases in the initial iterations and then stabilizes, indicating that the Multiplicative Update algorithm converges fairly rapidly for this dataset and then reaches a plateau.

The error stabilizes after about 25 to 50 iterations, suggesting that additional iterations do not significantly improve the fit.

ii)

we analyze the convergence of the Multiplicative Update Algorithm (MULT) for Non-Negative Matrix Factorization (NMF).

ii) Threshold Analysis

For tolerance $\varepsilon = 10^{-4}$, we find:

$$\boxed{\text{No integer } K \leq 1000 \text{ satisfies } f(K) < 10^{-4}}$$

This indicates that the algorithm either converges to a local minimum or requires more iterations to achieve the desired tolerance.

Task7

In Principal Component Analysis (PCA), when the covariance matrix is used, the principal component scores are uncorrelated. The proof follows from the eigenvalue decomposition of the covariance matrix.

Let X represent the $n \times p$ data matrix with n observations and p features. The covariance matrix Σ is a $p \times p$ matrix, where:

$$\Sigma = \text{Cov}(X) = \frac{1}{n-1} X^T X$$

Step 1: Eigenvalue Decomposition of the Covariance Matrix Perform the eigenvalue decomposition of the covariance matrix Σ:

$$\Sigma = V \Lambda V^T$$

where V is the matrix of eigenvectors and Λ is the diagonal matrix of eigenvalues.

Step 2: Transformation of Data In PCA, the data is projected onto the eigenvectors of the covariance matrix to form the principal component scores. Let Z denote the matrix of principal component scores, which is obtained by multiplying the original data matrix X by the matrix of eigenvectors V:

$$Z = XV$$

Step 3: Covariance of the Principal Component Scores To prove that the principal components are uncorrelated, we compute the covariance matrix of Z. The covariance matrix of Z is given by:

$$\text{Cov}(Z) = \frac{1}{n} Z^T Z = \frac{1}{n} (XV)^T (XV)$$

Expanding the above expression:

$$\text{Cov}(Z) = \frac{1}{n} V^T X^T X V$$

Since $X^T X = \Sigma$, we substitute this into the equation:

$$\text{Cov}(Z) = \frac{1}{n} V^T \Sigma V$$

Step 4: Substitution with Eigenvalue Decomposition Now, using the eigenvalue decomposition of Σ, where $\Sigma = V \Lambda V^T$, we substitute into the equation:

$$\text{Cov}(Z) = \frac{1}{n} V^T (V \Lambda V^T) V$$

Since $V^T V = I$ (the identity matrix), this simplifies to:

$$\text{Cov}(Z) = \Lambda$$

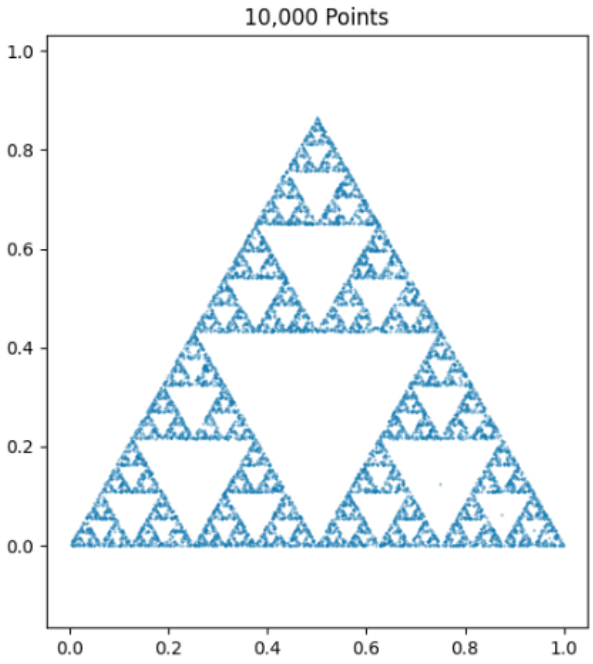
Thus, the covariance matrix of the principal component scores is a diagonal matrix Λ, where the diagonal elements are the eigenvalues of the covariance matrix Σ. **Conclusion** Since Λ is a diagonal matrix, the off-diagonal elements are zero. This means that the correlation between any pair of principal components is zero. Therefore, the principal component scores are uncorrelated.

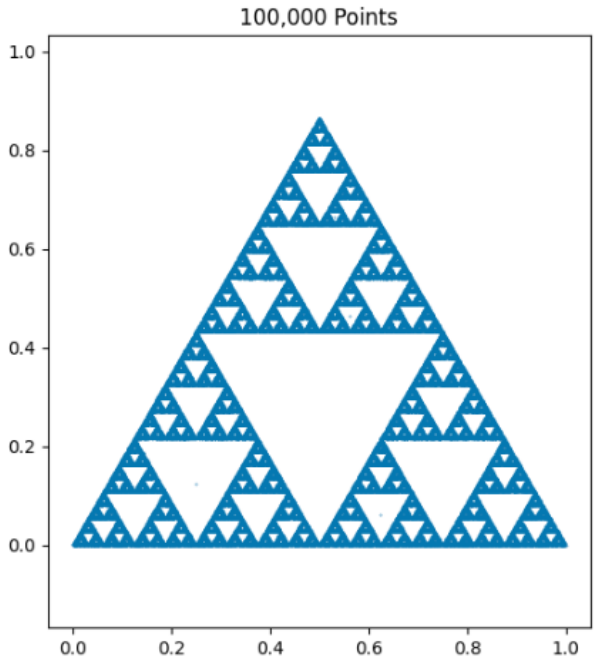
Task8

a)

The python code provided effectively simulates the "chaos game" method to generate points that fall within the Sierpinski triangle, adhering to its fractal structure.

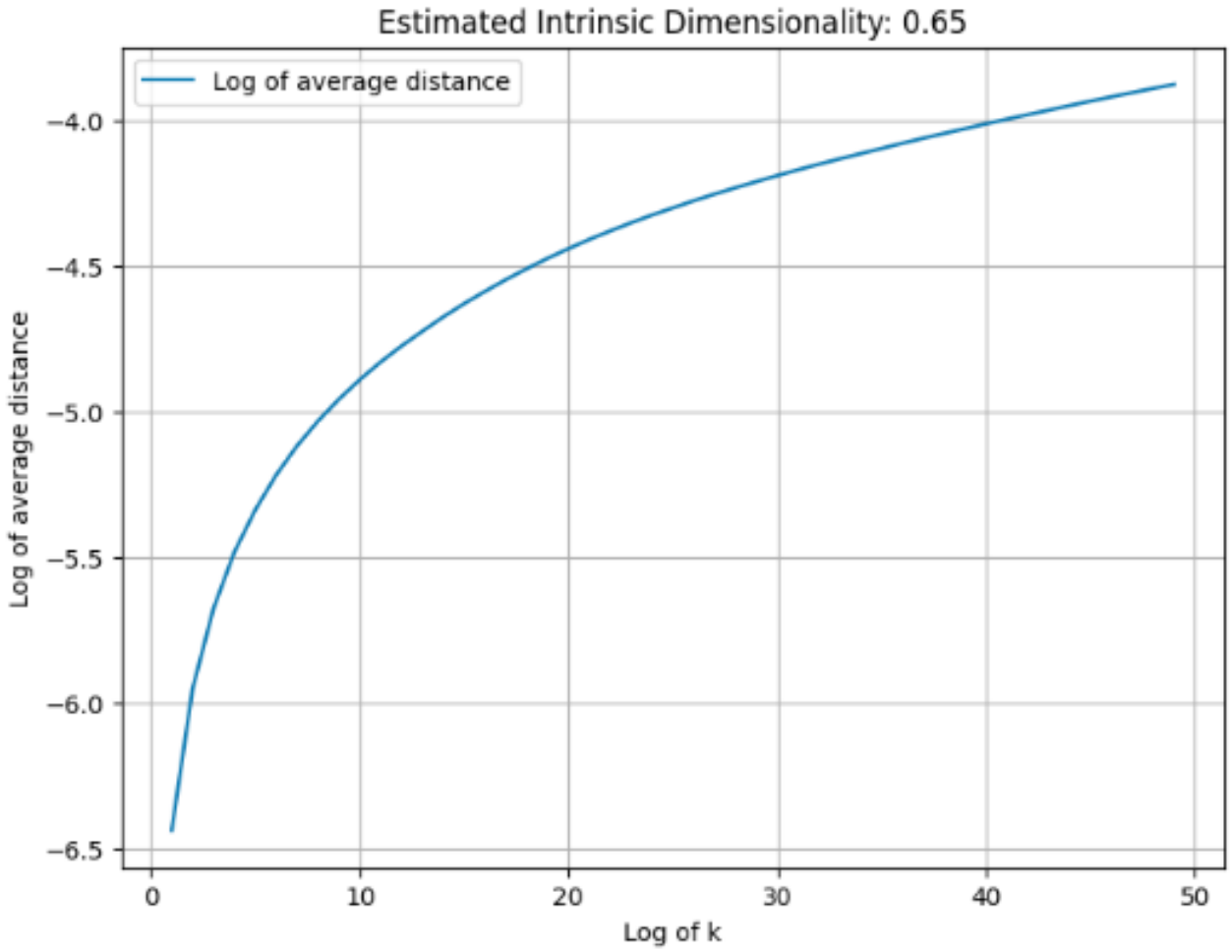
The output :





b)

To estimate the intrinsic dimensionality i will be using the k-nearest neighbor (k-NN) method, which provides a measure based on how the distances between data points scale as the number of neighbors increases:




```
Estimated Intrinsic Dimensionality: 0.65
```

c)

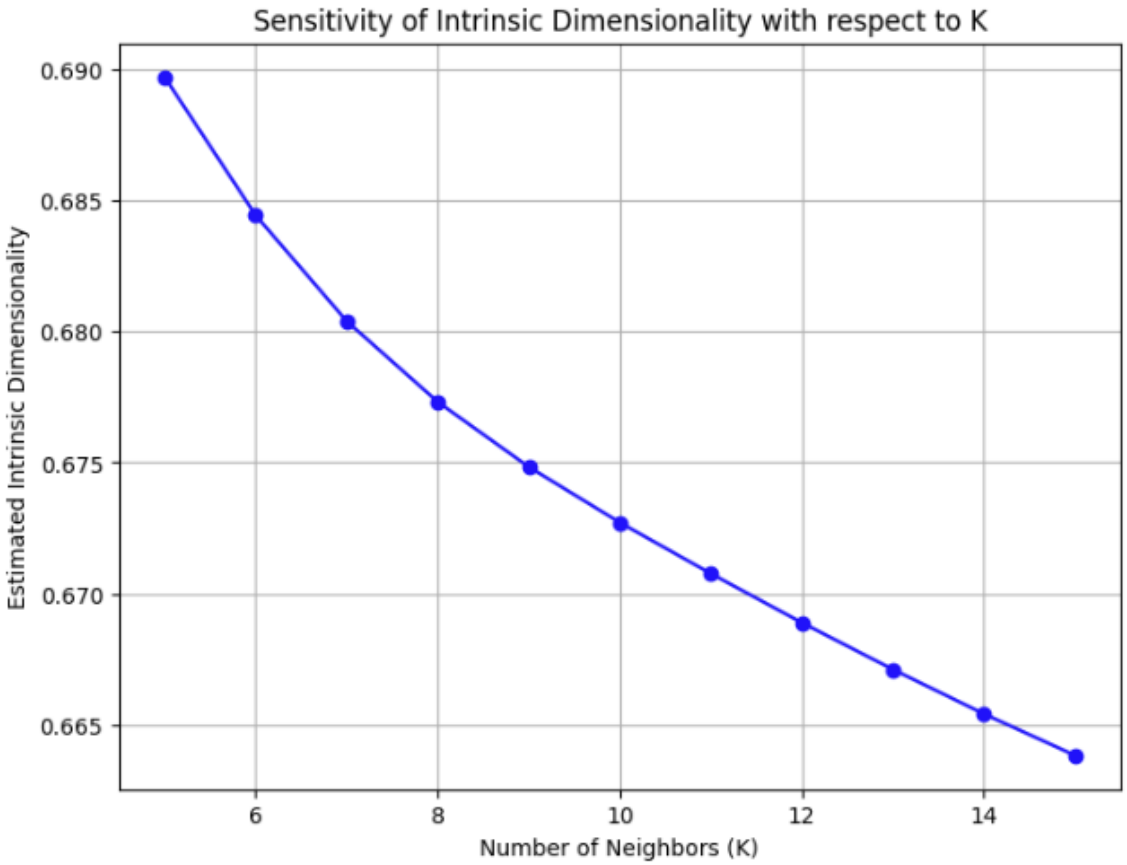
To study the sensitivity of the intrinsic dimensionality estimation with respect to the number of neighbors K, we need to run the dimensionality estimation procedure for various values of K within the range $5 \leq K \leq 15$. We will then plot the estimated dimensionality as a function of K and analyze the behavior of these estimates.

Steps:

- **Generating the Sierpinski Triangle:** I will use the provided MATLAB code to generate the Sierpinski triangle (in python)
- **Estimating the Dimensionality for Different K:**Than compute the intrinsic dimensionality for different values of K using the k-nearest neighbor method.
- **Plotting the Sensitivity:** plotting the intrinsic dimensionality estimates as a function of K for $K \in [5, 15]$.

Steps in Python:

First i generate the Sierpinski triangle with 1,000,000 points as given in the MATLAB code, and then estimate the intrinsic dimensionality for K values between 5 and 15..(please find the code in the second document)



The graph shows a decreasing trend in the estimated intrinsic dimensionality as the number of neighbors K increases. Starting from around 0.690 for $K = 5$ and decreasing to about 0.665 for $K = 15$.This suggests that larger neighborhood sizes might be smoothing out local complexities, leading to lower estimates of intrinsic dimensionality.
Implications:

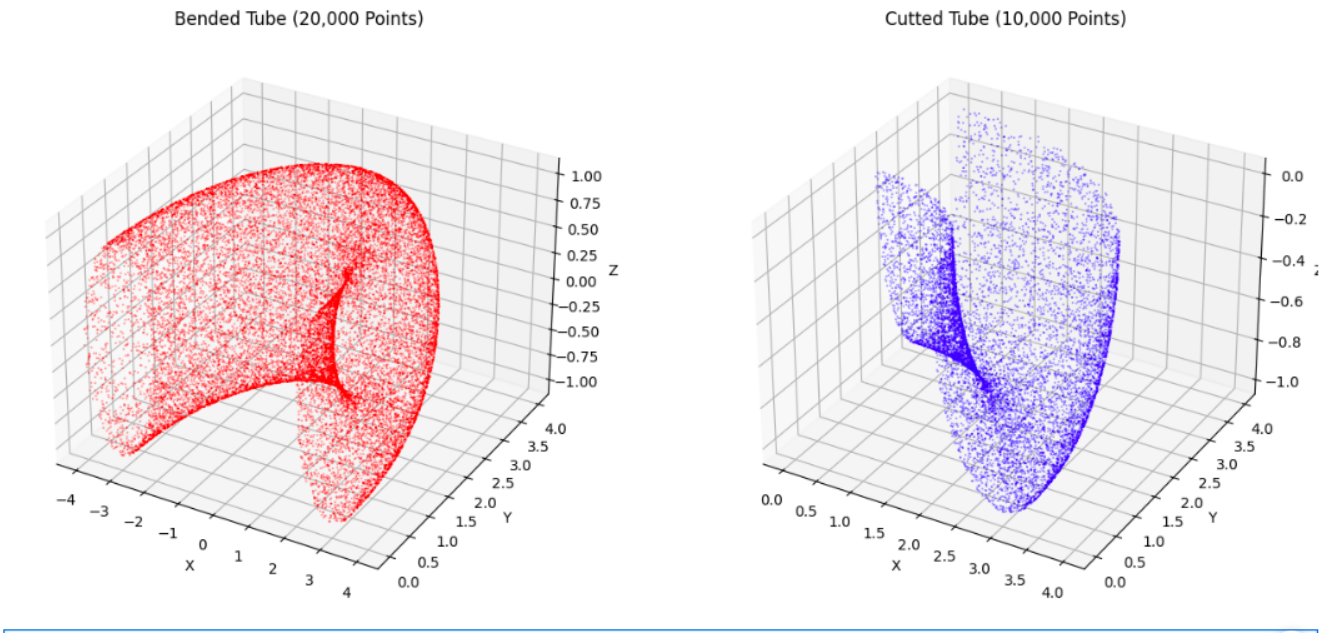
Local vs. Global Structure: Lower K values likely capture more of the local fractal structure, which is more sensitive to finer details, whereas higher K values average these details over larger neighborhoods, reflecting broader data trends.

Parameter Sensitivity: The choice of K significantly impacts the estimation, highlighting the method's sensitivity to this parameter.

Task9

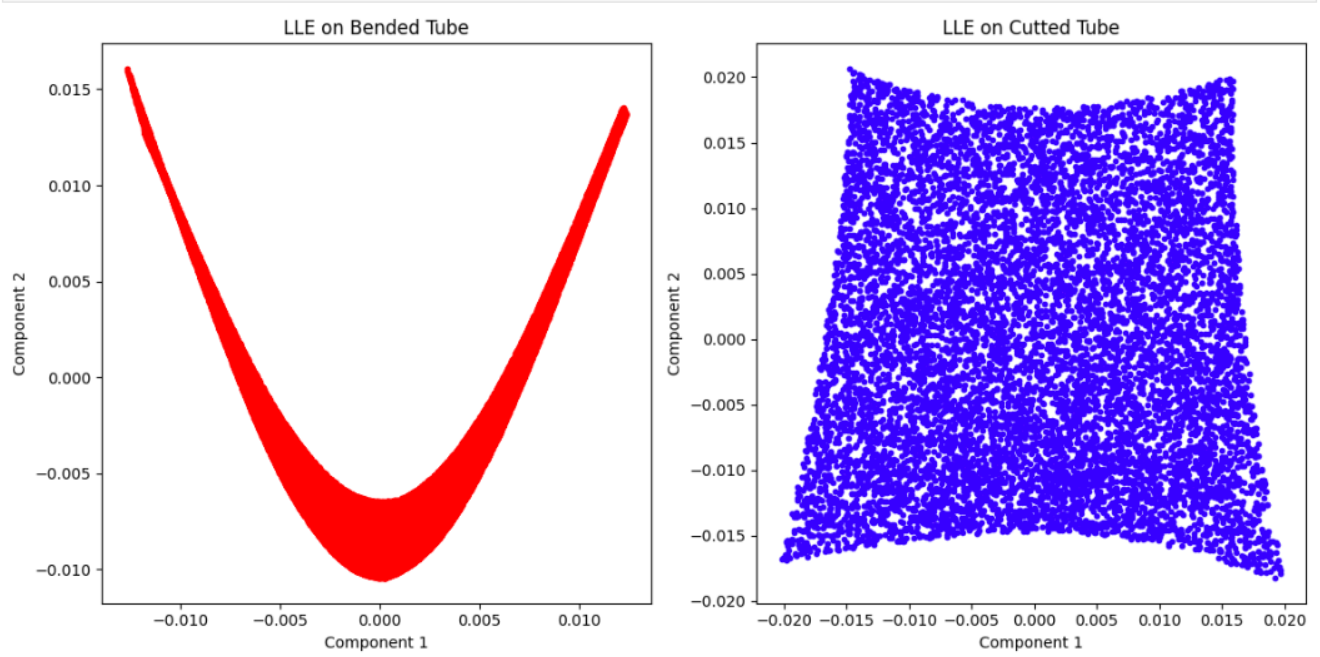
a)

To generate sample points for the bended tube X (20,000 points) and cutted tube Z (10,000 points) in Python, Im gonna replicate the logic from the provided MATLAB code tube.m (the code is in the implimentation file)
- The plot :



b)

To apply LLE to the bended and cutted tubes (datasets XX and ZZ), we'll use Python's "sklearn.manifold.LocallyLinearEmbeddin and analyze how the method performs on these different topological structures.



- LLE on the Bended Tube (X):

  The LLE result shows a clear "U" shape, which suggests that LLE has successfully unfolded the bended tube into a two-dimensional space.

  Local neighborhoods are preserved, and the intrinsic 2D geometry is successfully unfolded.

  Analysis: LLE performs well here because the tube is a continuous manifold without discontinuities. The algorithm reconstructs local linear relationships and maps the half-torus to a 2D space faithfully.

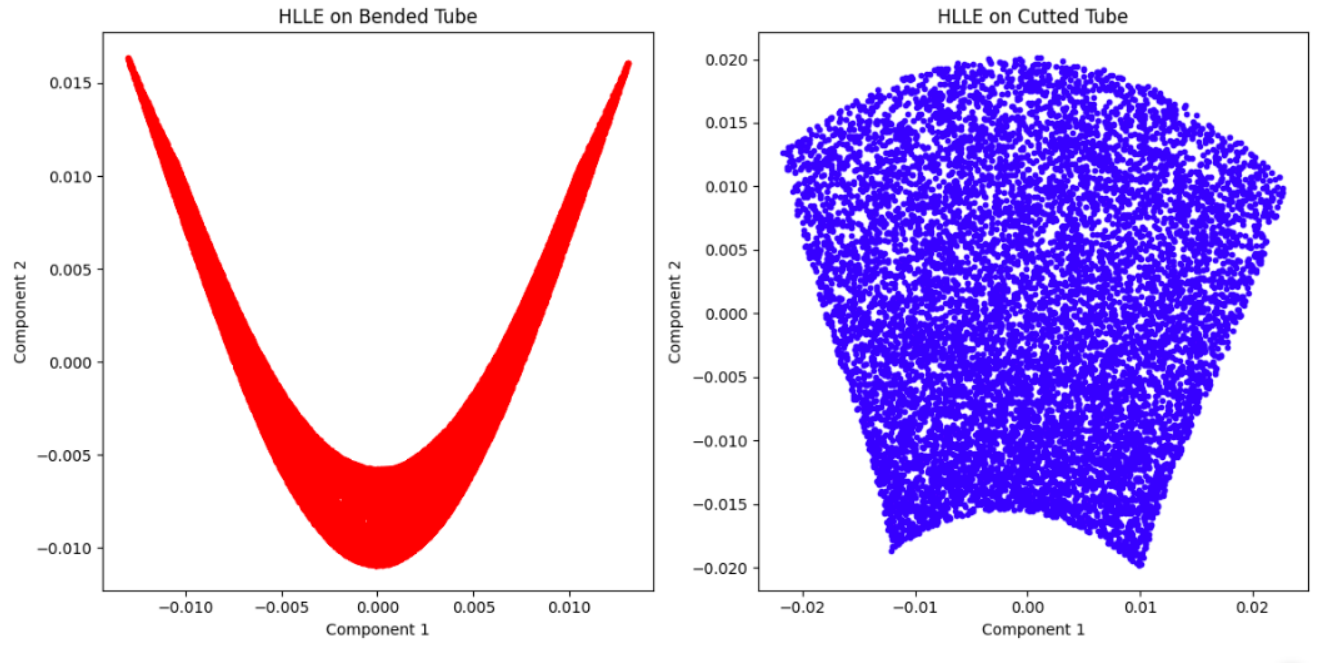
- On the Cutted Tube:

  The result for the cutted tube appears as a dense, almost square-like scatter of points. This indicates a loss of the clear structural integrity that we might expect given the cut and open ends of the tube.

  The dense plot suggests that LLE struggled to maintain a meaningful representation of the topological changes due to the cut. Instead of showing an open structure or any indication of the ends, it spreads out the points, likely trying to maintain local neighbor relations without recognizing the overall topological form.

c)

For this task i also used the same Python library *LocallyLinearEmbedding* from *sklearn.manifold* (the computation is in the implimentation file)



- Bended Tube HLLE seems to perform better than LLE by providing a cleaner, more precise visualization of the tube's unrolled form. This indicates superior handling of the manifold's curvature and connectivity.
- Cutted Tube Both LLE and HLLE struggle with the cutted tube, but HLLE provides a somewhat clearer representation of the spread, albeit without effectively capturing the cut. It shows the global structure but fails to delineate the open ends and the distinct topology introduced by the cut.

Task10

a)

Implimentation in Python (live in the file dedicated for the code) :

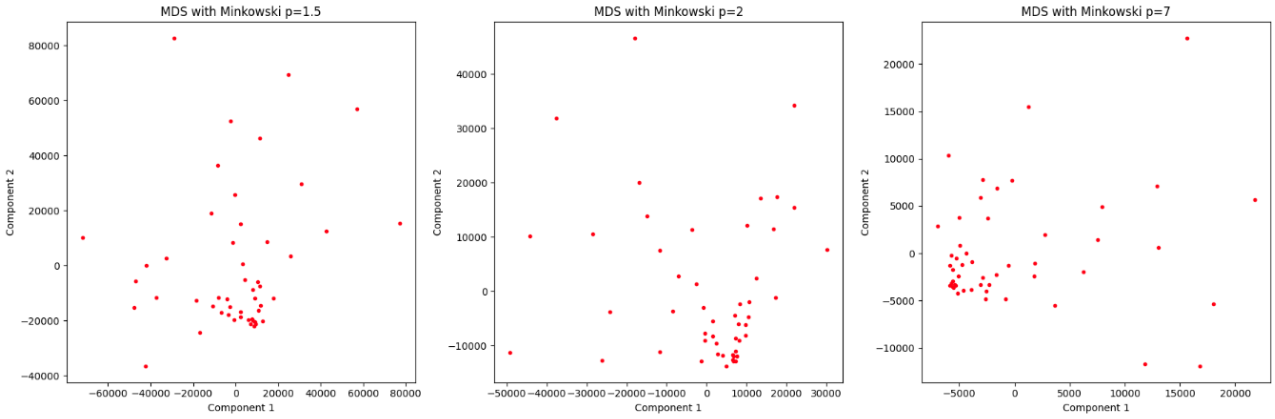
```python
import numpy as np

def guttman_transform(Z, delta):
    """here i performed the Guttman transformation."""
    n = Z.shape[0]
    D_Z = np.linalg.norm(Z[:, np.newaxis] - Z, axis=2)  # Distance matrix in the lower dimension
    B = np.zeros((n, n))
    np.fill_diagonal(B, -B.sum(axis=0))
    np.divide(delta, D_Z, out=B, where=D_Z!=0)  # Avoid division by zero
    B *= -1
    np.fill_diagonal(B, -B.sum(axis=1))
    X = (1/n) * B @ Z
    return X

def smacof(delta, d=2, max_iter=100, tol=1e-4):
    """The SMACOF algorithm for multidimensional scaling."""
    n = delta.shape[0]
    X = np.random.rand(n, d)  # Initial configuration
    stress_old = np.inf

    for i in range(max_iter):
        X = guttman_transform(X, delta)
        D_X = np.linalg.norm(X[:, np.newaxis] - X, axis=2)
        stress_new = np.sum((D_X - delta)**2)

        if abs(stress_new - stress_old) < tol:
            break
        stress_old = stress_new

    return X

# Example of usage / testing the output
# delta: nxn dissimilarity matrix
n = 10  # Number of points
delta = np.random.rand(n, n)
delta = (delta + delta.T) / 2  # i made it symetric
np.fill_diagonal(delta, 0)  # Distance from a point to itself should be 0

X = smacof(delta, d=2, max_iter=100, tol=1e-4)
print("Final Configuration:\n", X)
```

```
Final Configuration:
 [[ 0.18412735  0.05101799]
 [-0.02346107  0.37773208]
 [-0.3203672   0.14765708]
 [ 0.32306185  0.25410326]
 [ 0.37866677  0.10510103]
 [-0.3550441  -0.16932254]
 [-0.23713303 -0.01636173]
```

Figure 8: SMACOF alg. implimentation

b)

Using the leukemia data set



The Multidimensional Scaling (MDS) results for different Minkowski p-values reveal distinct patterns in point distribution and scaling:

- $p = 1.5$: The points display a wide spread with a large range, indicating that the distance metric is sensitive to variations in all dimensions, resulting in a diverse dispersion.
- $p = 2$ (Euclidean Distance): Points are more centrally clustered, suggesting that the Euclidean metric provides a balanced sensitivity to differences across dimensions, leading to a more concentrated distribution.

- $p = 7$: There is a pronounced central concentration of points with a reduced range in components, showing that higher p-values focus on the maximum differences and diminish the impact of smaller variations, causing the points to cluster tightly around the center.

Conclusion: As p increases, the distribution of points becomes more centralized and the scale compresses, reflecting a focus on maximum disparities and a reduction in the influence of smaller dimensional variations. These observations suggest that the choice of p significantly impacts the perceived structure and diversity in MDS analysis.