Title: Analyzing Clustering Methods on Multidimensional Data: A Comparative Study of K-Means, PCA, Single-Linkage, and t-SNE.

1. Abstract

This report examines the use of clustering algorithms and dimensionality reduction techniques for high-dimensional data analysis, focusing on methods like k-means clustering, single-linkage clustering, Gaussian Mixture Models (GMM), along with tools such as Principal Component Analysis (PCA) and t-distributed Stochastic Neighbor Embedding (t-SNE). These approaches were applied to a dataset to uncover patterns and improve clustering performance. Key findings uncover that k-means clustering excels with spherical clusters and single-linkage clustering is more effective for irregular shapes. PCA and t-SNE successfully reduced dimensions, facilitating clearer visualizations, though each method has limitations such as initialization sensitivity. The robustness of k-means was tested under different initializations, with results evaluated by comparing each initialization using heatmaps. Metrics like silhouette scores and scree plots are used to find the optimal number of clusters and PCA components. Though the models addressed the specified questions, challenges such as scalability and parameter sensitivity remain. Future work should consider ensemble methods, automated parameter tuning, and domain-specific enhancements to boost performance and applicability. This study highlights the importance of understanding the strengths and weaknesses of unsupervised learning techniques.

2. Introduction

By mimicking human intelligence for learning and decision-making Artificial Intelligence is revolutionizing various industries. A subset of Artificial Intelligence; Machine learning, specializes in enabling computers to learn from data patterns and make predictions without explicit programming (Russell & Norvig, 2021). Machine learning techniques are classified as supervised, unsupervised, and reinforcement learning.

Supervised learning involves training a model on labelled data, where the input-output relationship is predefined, to learn a mapping function to predict outputs for unseen inputs. In this process, the dataset consists of inputs (X) and corresponding outputs (Y), and the model learns by minimizing the error between predicted outputs (Y(hat)) and actual outputs (Y). Common techniques in supervised learning include regression, which is used for predicting continuous values, and classification, which categorizes data into classes. For example, it can be applied to predict house prices based on factors like size and location (regression) or to classify emails as spam or non-spam (classification).

In contrast, unsupervised learning operates on unlabeled data, to uncover hidden patterns or structures without predefined outputs. This approach involves algorithms that identify similarities or differences in the input data, grouping them accordingly. Typical techniques include clustering methods, such as k-means and DBSCAN, and dimensionality reduction techniques like PCA and t-SNE. Unsupervised learning has applications such as customer segmentation in marketing, where similar customers are grouped based on purchasing behaviour, and anomaly detection, which identifies outliers in datasets, such as fraudulent transactions.

Reinforcement learning, on the other hand, trains an agent to make sequential decisions by interacting with an environment. The agent learns through a reward-punishment system, performing actions and receiving feedback in the form of rewards (positive) or penalties (negative) based on those actions. The goal is to maximize cumulative rewards over time by improving the policy that dictates action selection. This approach is used in applications such as training autonomous vehicles to navigate roads safely and

teaching robots to perform tasks like playing chess or assembling objects. Reinforcement learning employs a model based on the interaction between an agent and its environment to enhance decision-making through trial and error. (Goodfellow, Bengio, & Courville, 2016).

Aspect	Supervised Learning	Unsupervised Learning	Reinforcement Learning
Data Requirement	Labeled data	Unlabeled data	Interaction with environment
Goal	Predict output values	Discover hidden patterns	Maximize long-term rewards
Approach	Input-output mapping	Data clustering/grouping	Trial-and-error learning
Examples	Spam detection, stock price prediction	Customer segmentation, anomaly detection	Robotics, game playing

2.1. **Clustering in Machine Learning**

Clustering is a fundamental unsupervised machine learning technique that organizes data into groups based on their similarities as specified by various algorithms. Points within the same cluster have similar characteristics, while points in different clusters exhibit significant differences. This ability is especially beneficial for tasks such as exploratory data analysis, identifying anomalies, and segmenting customers. Among various clustering algorithms, k-means clustering, hierarchical clustering, and Gaussian mixture models are notable for their practical applications and adaptability to diverse data scenarios (Han, Kamber, & Pei, 2011).

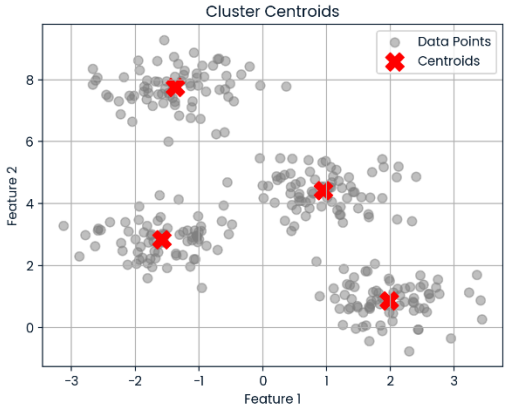


Figure 1 Cluster centroids diagram using dummy data.

2.2. **K-means Clustering**

K-means clustering is a popular centroid-based unsupervised learning algorithm that partitions a dataset into k distinct clusters. It minimizes the variance within each cluster while maximizing the variance between them.

The k-means clustering algorithm iteratively refines cluster centroids and assigns data points based on their proximity to these centroids, resulting in compact, spherical clusters. However, k-means assumes that clusters have a uniform size & shape and can be sensitive to the initial selection of centroids. This sensitivity usually leads to different results in each run of the algorithm. (Lloyd, 1982).

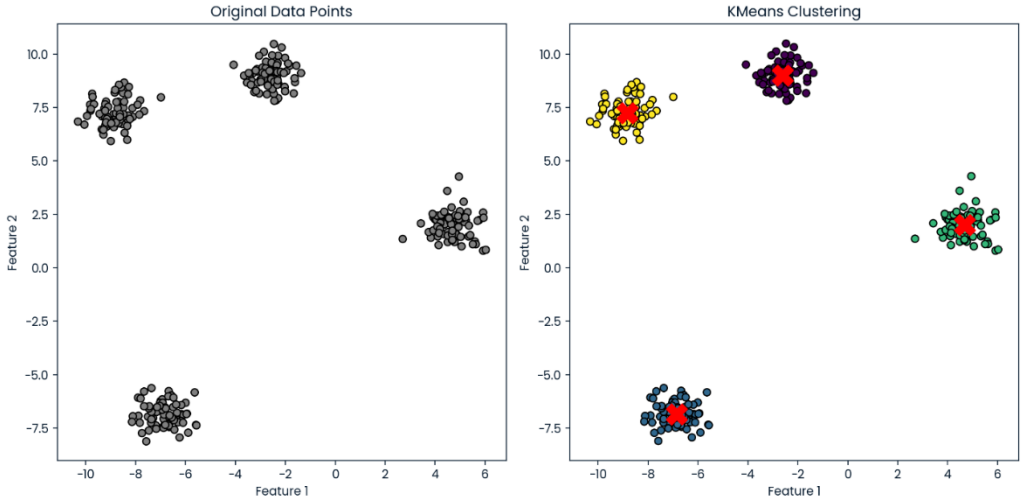


Figure 2 Graphs depicting k-means clustering using dummy data.

2.2.1. Silhouette Score:

The silhouette score is a metric used to evaluate the quality of clustering. It measures how well each data point fits within its assigned cluster compared to other clusters. A higher silhouette score, which ranges from -1 to 1, indicates that the clusters are distinct and well-defined. Silhouette scores were calculated to find the optimal number of clusters.

2.3. **Single-Linkage Clustering**

Single-linkage clustering is a hierarchical algorithm that merges clusters based on the minimum distance between two points in different clusters. This method is especially effective for creating elongated or irregular cluster shapes, setting it apart from centroid-based approaches such as k-means. The results of single-linkage clustering can be visualized using dendrograms, illustrating the hierarchy of cluster mergers (Tan, Steinbach, & Kumar, 2018). This method is especially beneficial when data structures are irregular, as it can detect clusters of any shape.

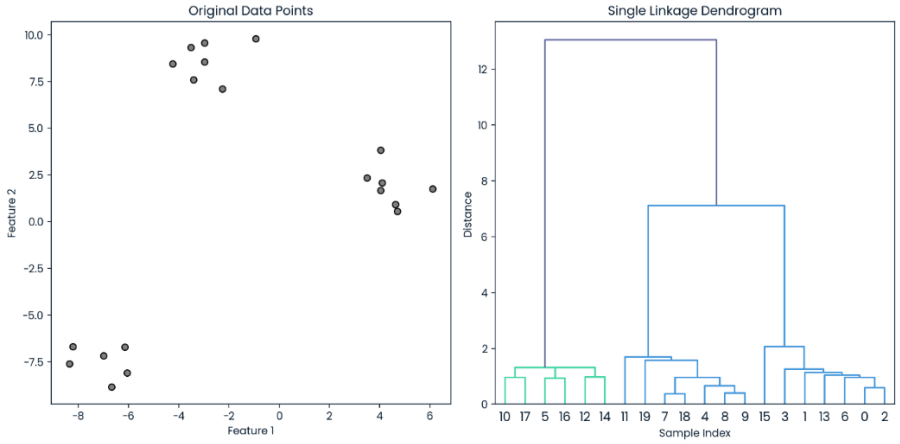


Figure 3 Graphically visualizing single-linkage clustering using dummy data.

2.4. **Dimensionality Reduction Techniques**

The "curse of dimensionality" affects computational efficiency and algorithm performance, which can pose difficulties for clustering when dealing with high-dimensional data. Dimensionality reduction methods such as Principal Component Analysis (PCA) and t-distributed Stochastic Neighbor Embedding (t-SNE) can be used to address these challenges.

2.5. **Principal Component Analysis (PCA)**

PCA transforms data into orthogonal components that capture maximum variance, simplifying datasets while maintaining interpretability. This linear technique is commonly used for preprocessing before clustering, as it reduces redundancy and highlights key patterns (Jolliffe, 2002).

PCA helps to simplify complex data by reducing the number of variables (or features) while keeping the important information. Imagine you have a dataset with many measurements (like height, weight, age, etc.). These measurements can be hard to visualize and analyze. PCA looks for patterns in the data and identifies the main factors that explain the most variation (differences) among the data points. It transforms the original measurements into new dimensions (called principal components) that combine the original data such that it highlights these patterns. Instead of dealing with all the original measurements, PCA allows you to work with just a few new dimensions that capture most of the key information.

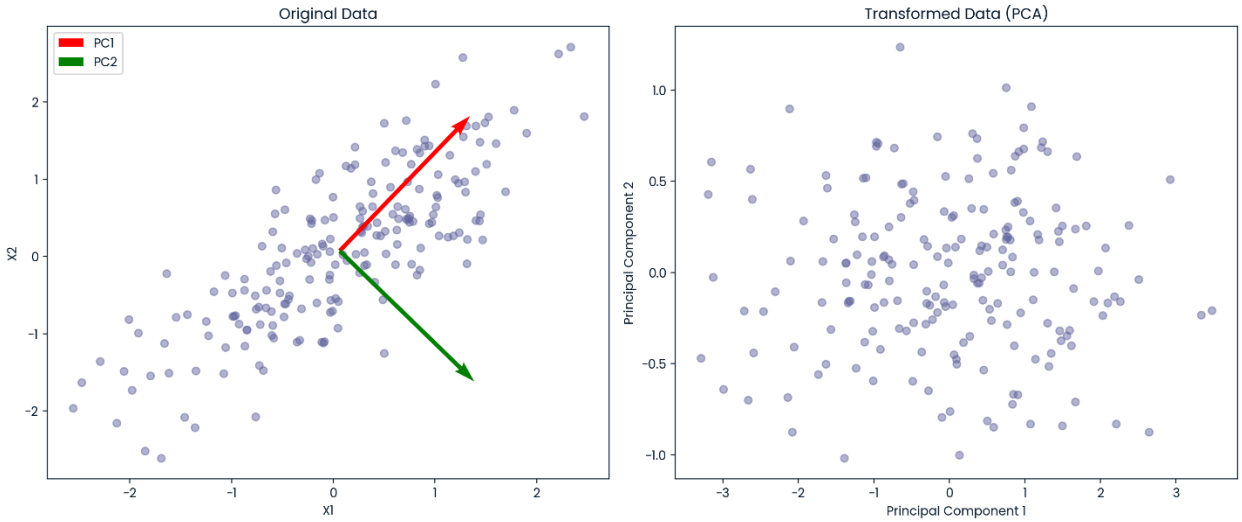


Figure 4 Graphs showing the implementation of PCA on dummy data.

2.5.1. Scree plot:

A scree plot is a visual tool used in Principal Component Analysis (PCA) and other dimensionality reduction techniques. It helps determine the optimal number of components to retain in a dataset by plotting the eigenvalues, representing the variance explained, of each principal component in descending order.

2.6. **t-SNE**

t-SNE (t-Distributed Stochastic Neighbor Embedding) is a non-linear technique for dimensionality reduction, specifically intended for visualizing high-dimensional data in two or three dimensions. Unlike PCA, t-SNE effectively captures non-linear relationships by embedding high-dimensional data into a low-

dimensional space while preserving the structures of neighbourhoods. This technique is beneficial for visualizing complex datasets and uncovering subtle clustering patterns that linear methods might miss (van der Maaten & Hinton, 2008).

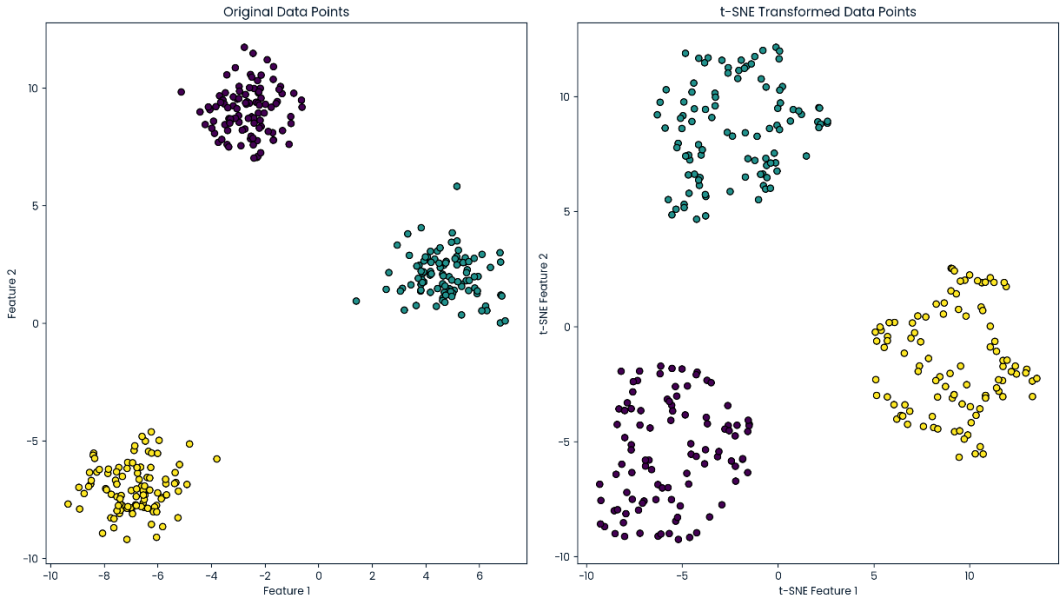


Figure 5 Graphs visualizing t-SNE on dummy dataset.

2.7. **Dataset**

The data was downloaded from the website 'https://russellmilne.com/teaching/gvrd-rangestd.csv'. It has been loaded into a Python program using a function from the Pandas library. This data is organized as a data frame containing only floating-point values. A data frame is a tabular data structure that facilitates easy storage, retrieval, and understanding of the data.

Initially, the dataset had dimensions of (5, 459), indicating 5 rows and 459 columns. However, instead of the typical structure, in this dataset, each column represents a single observation, and each row reflects the values of a specific feature across these observations. To address this, the data was transposed, resulting in new dimensions of (459, 2207), where there are now 459 rows and 2207 columns. In this transposed format, the columns represent the features, and the rows represent the observations.

```
#Transpose the data to have observations as rows and features as columns
data = data.T
Dimensions: (459, 2207)
      0         1    2         3    ...      2203      2204      2205      2206
0  0.048734  0.156177   0.0  0.584464  ...  0.068796  0.068681  0.101562  0.082707
1  0.111323  0.074072   0.0  0.571429  ...  0.194103  0.197802  0.117188  0.150376
2  0.064023  0.080148   0.0  0.473214  ...  0.181818  0.162088  0.195312  0.132832
3  0.058767  0.131366   0.0  0.410714  ...  0.031941  0.024725  0.109375  0.162907
4  0.050645  0.147004   0.0  0.446429  ...  0.140049  0.131868  0.148438  0.170426

[5 rows x 2207 columns]
print(data.head())
print(data.describe())
            0           1      2    ...        2204        2205        2206
count  459.000000  459.000000  459.0  ...  459.000000  459.000000  459.000000
mean     0.080778    0.120589    0.0  ...    0.251107    0.204572    0.196368
std      0.077852    0.125591    0.0  ...    0.159509    0.157142    0.120650
min      0.000000    0.000000    0.0  ...    0.000000    0.000000    0.000000
25%      0.049212    0.050928    0.0  ...    0.127747    0.101562    0.110276
50%      0.061634    0.086378    0.0  ...    0.225275    0.156250    0.177945
75%      0.084090    0.145699    0.0  ...    0.329670    0.242188    0.250627
max      1.000000    1.000000    0.0  ...    1.000000    1.000000    1.000000

[8 rows x 2207 columns]
```

Figure 6 Python code along with a snapshot of the dataset.

2.7.1. Data Processing:

There is no need to scale or standardize the data, as it contains no empty or duplicate values. While some rows or columns may share the same values, such as zeros, removing these uniform values will not affect the clustering results. Additionally, uniform values do not influence the overall clustering outcomes, confirming that scaling is unnecessary.

```
print("Emptyvalues:␣", data.isna().sum().sum())
print("Duplicates:␣", data.duplicated().sum())
```

```
Emptyvalues:␣ 0
Duplicates:␣ 0
```

Figure 7 Python code to check for empty and duplicate values.

2.8. **Problem Description**

The increasing complexity of datasets across various fields, such as retail and healthcare, highlights the importance of understanding intrinsic patterns within high-dimensional data. Clustering is a crucial unsupervised machine-learning technique that explores these datasets and uncovers natural groupings. However, clustering in high-dimensional spaces presents unique challenges, including computational inefficiency, reduced robustness of algorithms, and the possibility of overfitting. Additionally, different clustering methods may yield varying results based on their underlying assumptions about the data's structure.

This project addresses these challenges by applying and comparing multiple clustering techniques on a real-world dataset. The dataset consists of observations characterized as high-dimensional vectors. The study investigates whether dimensionality reduction techniques, such as Principal Component Analysis (PCA), can improve clustering outcomes by simplifying the data while simultaneously preserving essential patterns. It evaluates the robustness of clustering methods against variations in initialization & algorithm parameters, and their ability to adapt to different data structures.

2.9. **Questions to be Addressed**

- How do different clustering algorithms (e.g., k-means, single-linkage) compare in structure?
- How do clusters from full-dimensional versus reduced-dimensional data compare, and does PCA enhance the clustering process?
- How do clusters from reduced-dimensional data align with those from the original data, and do clusters identified in t-SNE correspond with those found by other algorithms?
- Is k-means clustering robust to changes in initial centroids, and what percentage of observation pairs consistently belong to the same cluster across multiple k-means runs?
- How do k-means results compare with hierarchical methods like single-linkage clustering regarding strengths and limitations?
- How does t-SNE facilitate data visualization and clustering, and how effectively do PCA and t-SNE visualize clustering patterns?
- Do Gaussian Mixture Models (GMM) provide insights through soft cluster memberships, and what percentage of observations is assigned to multiple clusters in GMM clustering?

3. Methodology

3.1. K-Means

In the initialization phase of clustering, the user selects the number of clusters, “k”, before the analysis. Subsequently, k initial centroids, which act as cluster centers, are randomly chosen. These centroids are not actual data points but serve as initial estimates for where the clusters may be localized.

In the Cluster Assignment step, each data point in the dataset is assigned to the nearest centroid using a distance metric, usually the Euclidean distance. This involves calculating the distance between a data point (xi) and a centroid (cj) to determine the closest centroid for each point, effectively grouping the data points based on their proximity to the centroids.

$$d(x_i, c_j) = \sqrt{\sum_{m=1}^{M} (x_{i,m} - c_{j,m})^2}$$

Euclidean distance formula, where:

x_i : a point

c_j : a centroid

M: dimensionality of data

After all points have been assigned to their respective clusters, the algorithm recalculates the position of each centroid. This process involves determining the new centroid by taking the mean of all points that have been assigned to each cluster using the following formula:

$$c_j = \frac{1}{n_j} \sum_{x \in C_j} x$$

Where:

n_j : number of points in Cj

The process involves iteratively repeating steps 2 and 3, in which points are reassigned to the nearest updated centroids. This reassignment is followed by recalculating the centroids. The iteration continues until the centroids converge, their positions no longer change significantly, or until a predetermined maximum number of iterations is reached.

K-means seeks to minimize the within-cluster sum of squares (WCSS), defined as:

$$WCSS = \sum_{j=1}^{k} \sum_{x \in C_j} \|x - c_j\|^2$$

The algorithm terminates when WCSS cannot be reduced further.

Applications of K-means clustering include customer segmentation, where customers are grouped according to their purchasing behaviour. It can also be utilized for image compression by reducing the number of colours in an image through the clustering of pixel intensities. Additionally, clustering plays a crucial role in document classification by organizing similar documents based on their term frequencies.

3.2. **Silhouette score**

First pairwise distances are calculated by calculating the distances between each point and all other points in the dataset. For each data point, determine the average distance to all other points in the same cluster to obtain cohesion, denoted as a(i). Additionally, find the average distance from each data point to all points in the nearest cluster to calculate separation, represented as b(i). Next, the silhouette coefficient is calculated.

The silhouette coefficient for a given data point i is defined as:

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$

where:

- $a(i)$: The average distance from point i to all other points in the same cluster (cohesion).
- $b(i)$: The average distance from point i to all points in the nearest cluster (separation).

The interpretation of the silhouette coefficient $s(i)$ is as follows:

- $s(i) \approx 1$: The point i is well-matched to its own cluster and poorly matched to neighboring clusters.
- $s(i) \approx 0$: The point i lies near the boundary between clusters.
- $s(i) < 0$: The point i is likely assigned to the wrong cluster, as it is closer to a neighboring cluster than its own.

Finally, compute the mean s(i) for all data points to obtain the overall silhouette score for the clustering solution.

3.3. **Single Linkage Clustering**

In single linkage clustering, the process begins by treating each data point as its own cluster, resulting in a total of n clusters for n data points. Next, pairwise distances between all points (or clusters) in the dataset are calculated using a suitable metric, such as Euclidean or Manhattan distance. After distance calculation, the algorithm identifies the pair of clusters that have the smallest distance according to the single linkage criterion and merges them into a single cluster. Once merged, the distances between the newly formed cluster and all other clusters are recalculated. This merging and updating process continues until only one cluster remains or until a desired number of clusters is achieved.

Clustering techniques find diverse applications across various fields. In biology, they are used to construct phylogenetic trees to explore evolutionary relationships. In document clustering, similar text documents are grouped for better organization and analysis. Anomaly detection leverages clustering to identify unusual patterns or rare events within datasets. Additionally, geographical analysis employs clustering to categorize locations based on their spatial proximity.

3.4. **Principle Component Analysis**

In the process of Principal Component Analysis (PCA), data is standardized initially. This is crucial because PCA is sensitive to the relative magnitude of different features. To standardize the data, it is ensured that all features are on the same scale. This involves subtracting the mean and dividing it by the standard deviation for each feature, which helps in achieving a uniform scale across the dataset.

To gain insights into how features within a dataset interact, it is important to compute the covariance matrix.

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

To find the eigenvalues and eigenvectors, begin by solving for the eigenvalues and eigenvectors of the covariance matrix, where the eigenvectors indicate the directions of maximum variance, known as principal components, and the eigenvalues signify the amount of variance captured by each principal component.

Next, select the principal components by ranking the eigenvalues in descending order and choosing the top k components, where k reflects the required dimensionality. This process ensures that the new dataset retains the most significant variance. Finally, project the original data into the new lower-dimensional space by multiplying it with the matrix of the selected eigenvectors.

Principal Component Analysis (PCA) has several applications, including data visualization, where it reduces complex data to 2D or 3D formats for easier exploration/analysis. It also plays a crucial role in noise reduction by eliminating components with low variance, helping to clarify the data. In machine learning, PCA simplifies models by reducing the number of input features, preventing overfitting in high-dimensional datasets. PCA is widely used in compression techniques, such as image compression, allowing for reduced storage requirements while simultaneously maintaining essential details.

We plot the scree plot to find out the optimal number of PCA components but since the question provided specifically instructs that we use only three components, we reduce the data using only three components.

3.5. **t-SNE**

In high-dimensional spaces, t-SNE calculates pairwise similarities between data points by employing a Gaussian distribution. For each point i, it assigns a probability pj|i, which indicates the likelihood of selecting point j as a neighbor of point i. This approach helps to convert the distances between data points into meaningful probabilities, facilitating a more intuitive understanding of the relationships within the data

$$p_{j|i} = \frac{\exp\left(-\|x_i - x_j\|^2 / 2\sigma_i^2\right)}{\sum_{k \neq i} \exp\left(-\|x_i - x_k\|^2 / 2\sigma_i^2\right)}$$

Where:

σi: controls the local neighbourhood size and is adjusted to achieve a desired perplexity.

In lower-dimensional space, pairwise similarities between points are modelled using t-distribution.

$$q_{ij} = \frac{\left(1 + \|y_i - y_j\|^2\right)^{-1}}{\sum_{k \neq l} \left(1 + \|y_k - y_l\|^2\right)^{-1}}$$

t-SNE seeks to align the low-dimensional probability distribution qij with the high-dimensional distribution pij by minimizing the Kullback-Leibler divergence:

$$KL(P\|Q) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

Using gradient descent, the algorithm iteratively adjusts the data point positions in the low-dimensional space to minimize KL divergence. Intuitively:

- Similar points, high pij, are pulled together
- Dissimilar points, low pij, are pushed apart.

The applications of t-SNE are very similar to the applications of PCA.

4. Results and Interpretation

Results and Interpretation:

Question 1a: Perform k-means clustering on the observations in this dataset with a number of clusters k of your choice that is greater than 3. Note which clusters each observation is in. Then, generate an n × n square matrix M, where n is equal to the number of observations in the dataset, and each entry mij is 1 if observations i and j are in the same cluster and 0 if they are in different clusters.

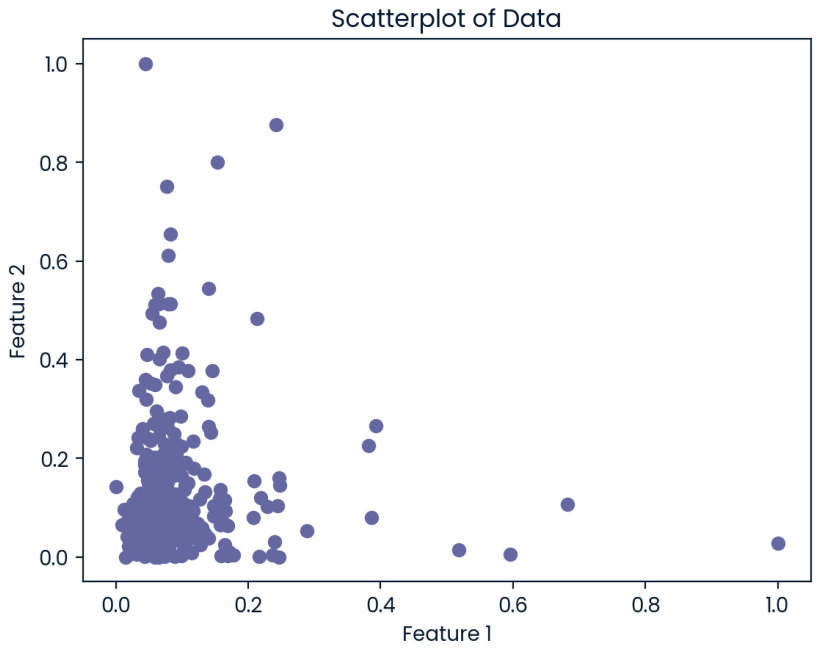


Figure 8 Scatterplot of original data.

```
M
```

```
array([[1, 1, 1, ..., 0, 0, 0],
       [1, 1, 1, ..., 0, 0, 0],
       [1, 1, 1, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 1, 1, 1],
       [0, 0, 0, ..., 1, 1, 1],
       [0, 0, 0, ..., 1, 1, 1]])
```

Using the Silhouette score for checking the number of optimal clusters on original data:

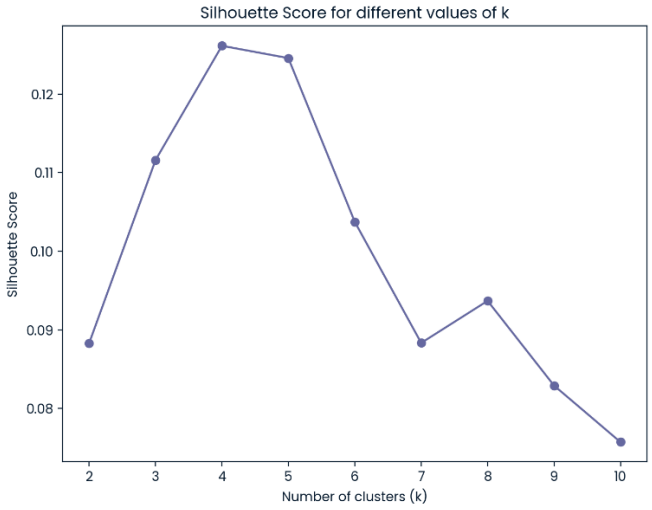


Figure 9 Silhouette score graph of original data.

We observe the optimal number of clusters is 4 using the silhouette score on the original data.

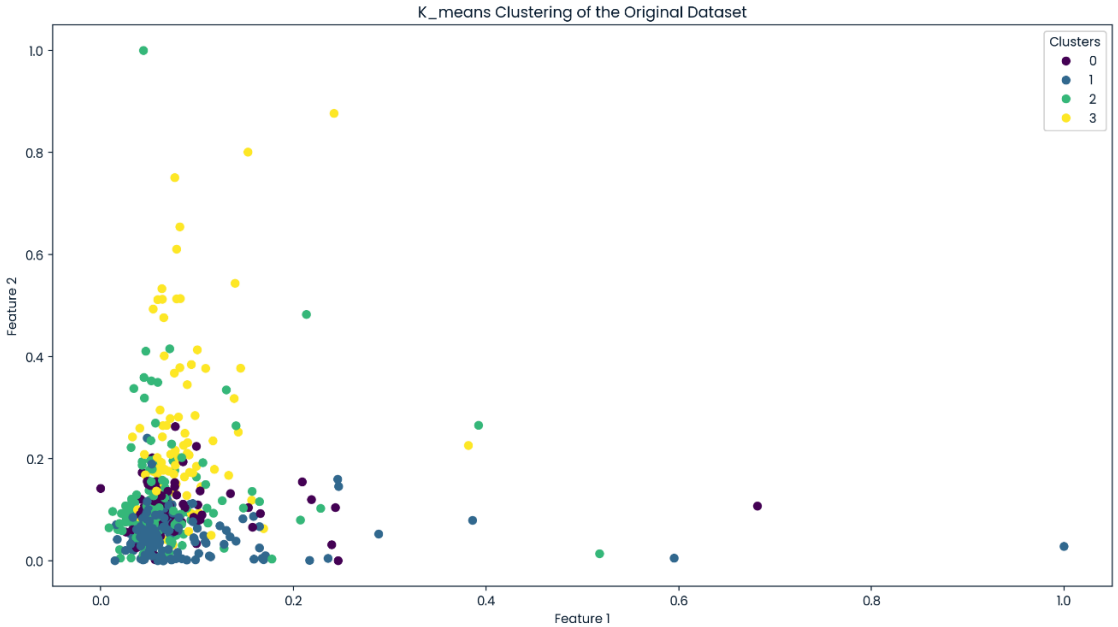


Figure 10 K-means clustering on the original dataset.

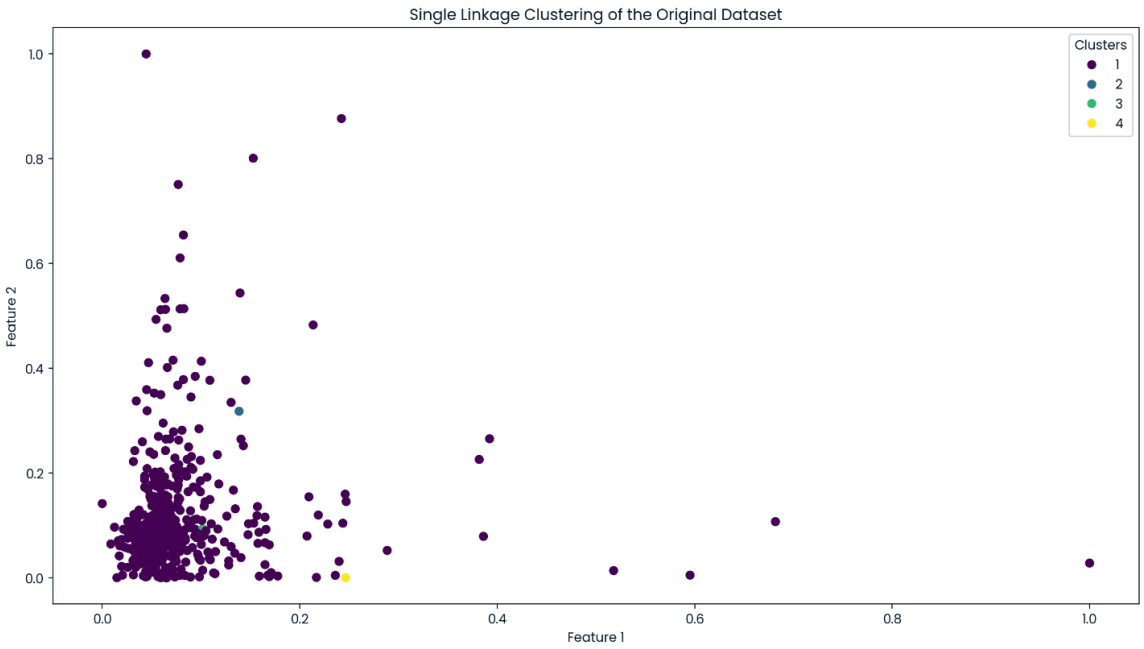


Figure 11 Single-linkage clustering on the original dataset.

Question 1b: Following this, perform principal component analysis on the observations in the dataset, keep only the first three principal components for each observation, and do k-means clustering on the resulting reduced-dimensional data with the same number of clusters that you used previously (i.e. k clusters). Create another n×n matrix P which follows the same rules as M, but for the reduced-dimensional data instead of the full-dimensional data.

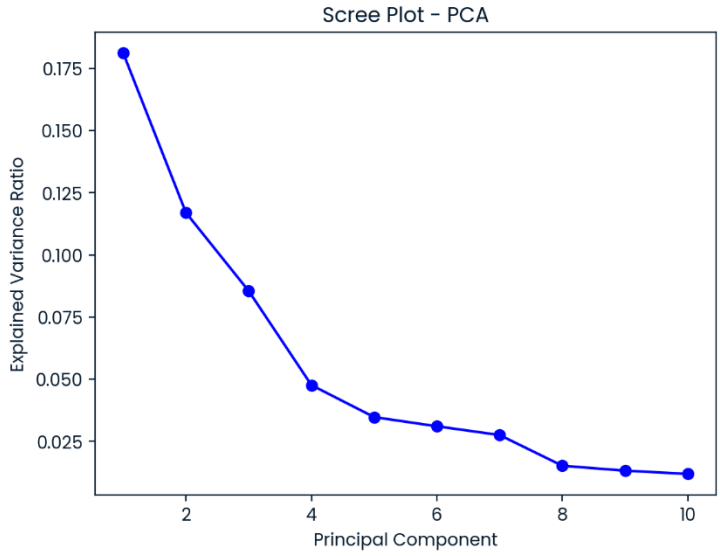


Figure 12 Scree Plot.

In the Scree plot, the first few components (e.g., 2–3) might explain a significant proportion of the variance. But since we are specifically asked to keep the principle components three, we do not consider a higher number.

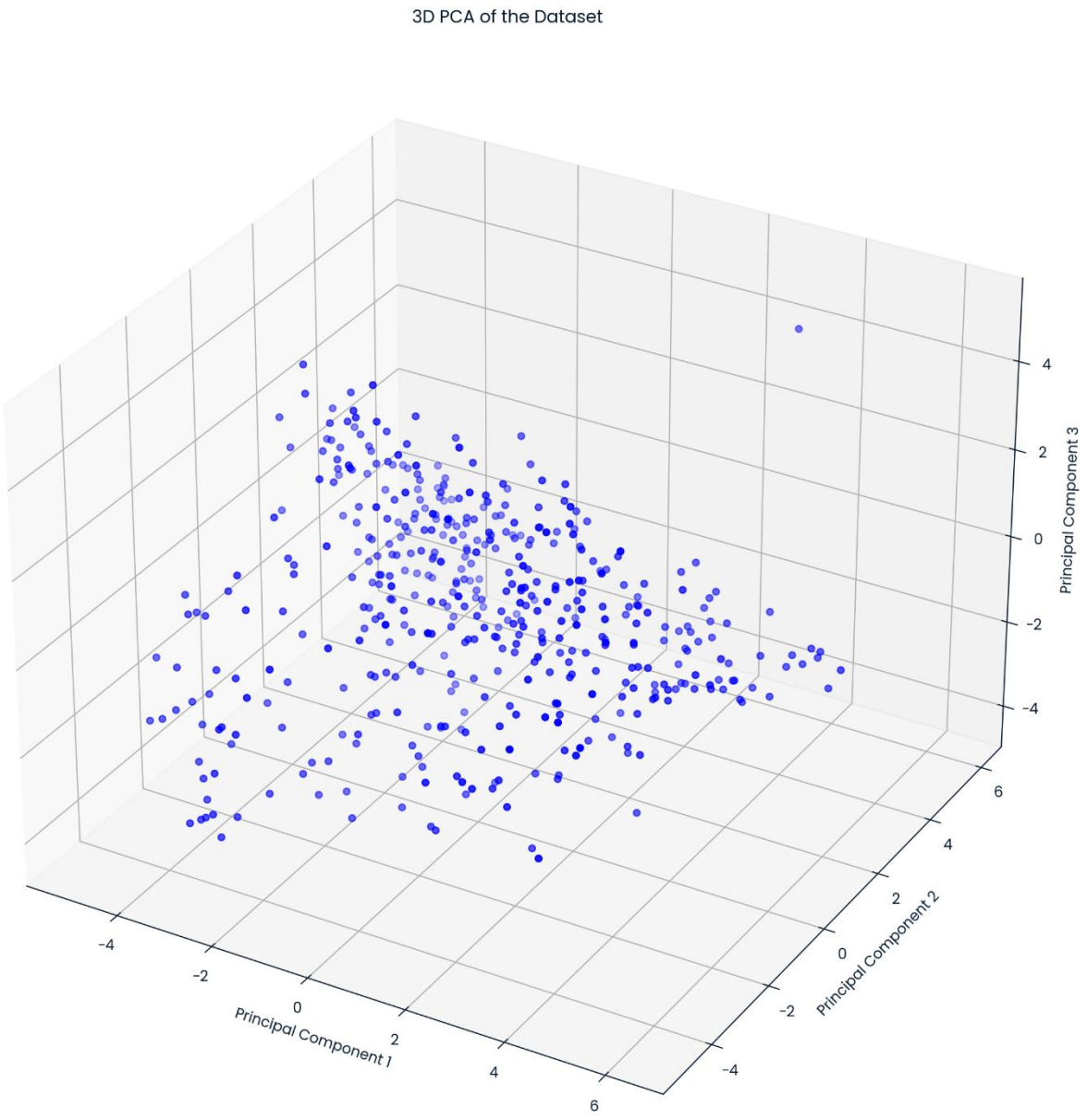


Figure 13 3D visualization of reduced data.

```
P
```

```
array([[1, 1, 1, ..., 0, 0, 0],
       [1, 1, 1, ..., 0, 0, 0],
       [1, 1, 1, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 1, 1, 1],
       [0, 0, 0, ..., 1, 1, 1],
       [0, 0, 0, ..., 1, 1, 1]])
```

Using the Silhouette score for checking the number of optimal clusters on the reduced data:

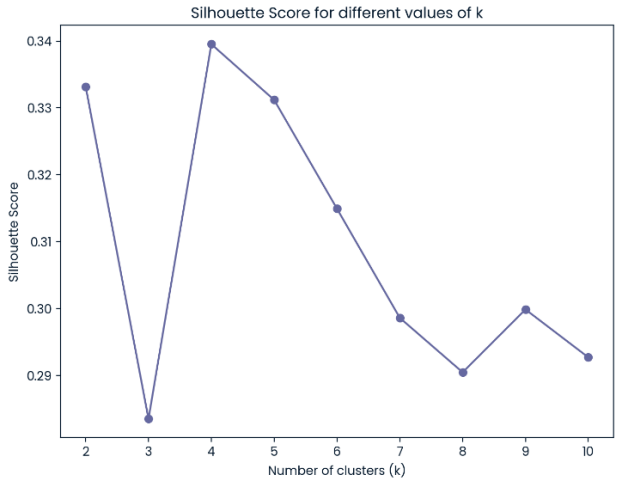


Figure 14 Silhouette score graph of reduced data.

We observe the optimal number of clusters is 4 using the silhouette score on the original data.

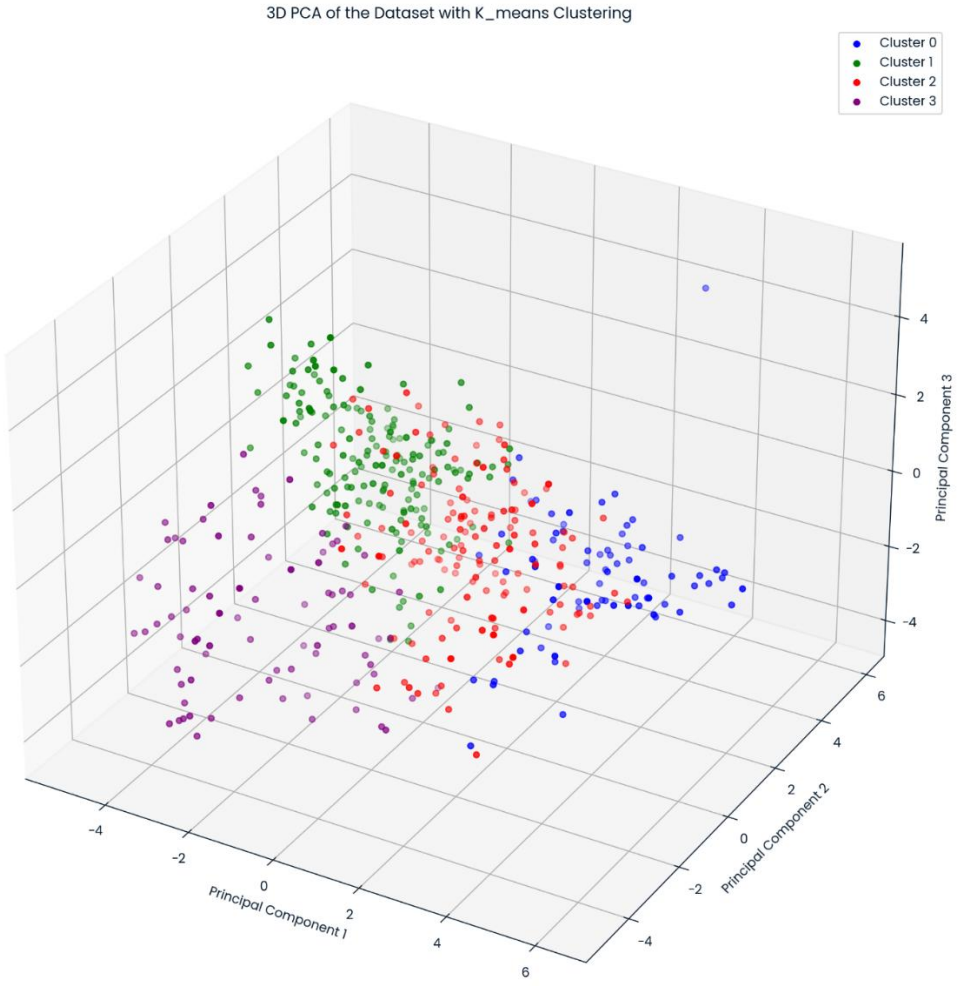


Figure 15 3D visualization of K-means on reduced data.

Question 1c: How many entries are the same between M and P? (In other words, how did performing PCA affect the structure of the data?)

```
# Calculate the number of entries that are the same between M and P
same_entries = np.sum(M == P)
same_entries
```

```
194341
```

194341 reflects the number of entries in the matrices M and P that are the same.

- Matrix M is an n × n matrix where M[i, j] is 1 if the i-th and j-th observations belong to the same cluster based on the original data clustering, and 0 otherwise.

- Matrix P is an n × n matrix where P[i, j] is 1 if the i-th and j-th observations belong to the same cluster based on the PCA-reduced data clustering, and 0 otherwise.

The value 194341 indicates the total number of positions (i, j) where both M and P have the same value (either both 1 or both 0). This can be interpreted as a measure of similarity between the clustering results obtained from the original data and the PCA-reduced data. A higher number suggests that the clustering results are more consistent between the two methods.

Question 2a: Perform k-means on the full-dimensional data ten times, using k clusters each time but ten different sets of initial values for cluster centroids. Generate matrices M1...M10 indicating whether given pairs of observations are in the same cluster, as you did for M and P.

```
M1, M2, M3, M4, M5, M6, M7, M8, M9, M10
```

```
(array([[1, 1, 1, ..., 0, 0, 0],
        [1, 1, 1, ..., 0, 0, 0],
        [1, 1, 1, ..., 0, 0, 0],
        ...,
        [0, 0, 0, ..., 1, 1, 1],
        [0, 0, 0, ..., 1, 1, 1],
        [0, 0, 0, ..., 1, 1, 1]]),
 array([[1, 1, 1, ..., 0, 0, 0],
        [1, 1, 1, ..., 0, 0, 0],
        [1, 1, 1, ..., 0, 0, 0],
        ...,
        [0, 0, 0, ..., 1, 1, 1],
        [0, 0, 0, ..., 1, 1, 1],
        [0, 0, 0, ..., 1, 1, 1]]),
 array([[1, 1, 1, ..., 0, 0, 0],
        [1, 1, 1, ..., 0, 0, 0],
        [1, 1, 1, ..., 0, 0, 0],
        ...,
        [0, 0, 0, ..., 1, 1, 1],
        [0, 0, 0, ..., 1, 1, 1],
        [0, 0, 0, ..., 1, 1, 1]]),
 array([[1, 0, 0, ..., 0, 0, 0],
        [0, 1, 1, ..., 0, 0, 0],
        [0, 1, 1, ..., 0, 0, 0],
        ...,
        [0, 0, 0, ..., 1, 1, 1],
        [0, 0, 0, ..., 1, 1, 1],
        [0, 0, 0, ..., 1, 1, 1]]),
 array([[1, 1, 1, ..., 0, 0, 0],
        [1, 1, 1, ..., 0, 0, 0],
        [1, 1, 1, ..., 0, 0, 0],
        ...,
```

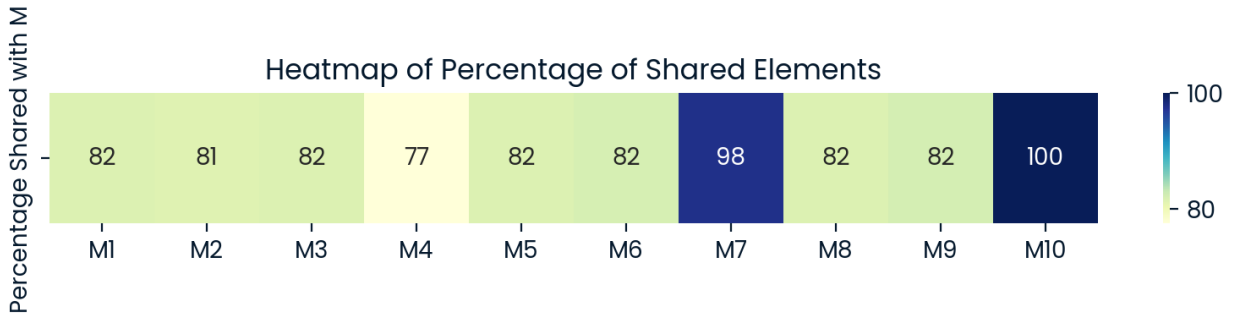


Figure 16 Heatmap comparing M with M1, M2 ... M10.

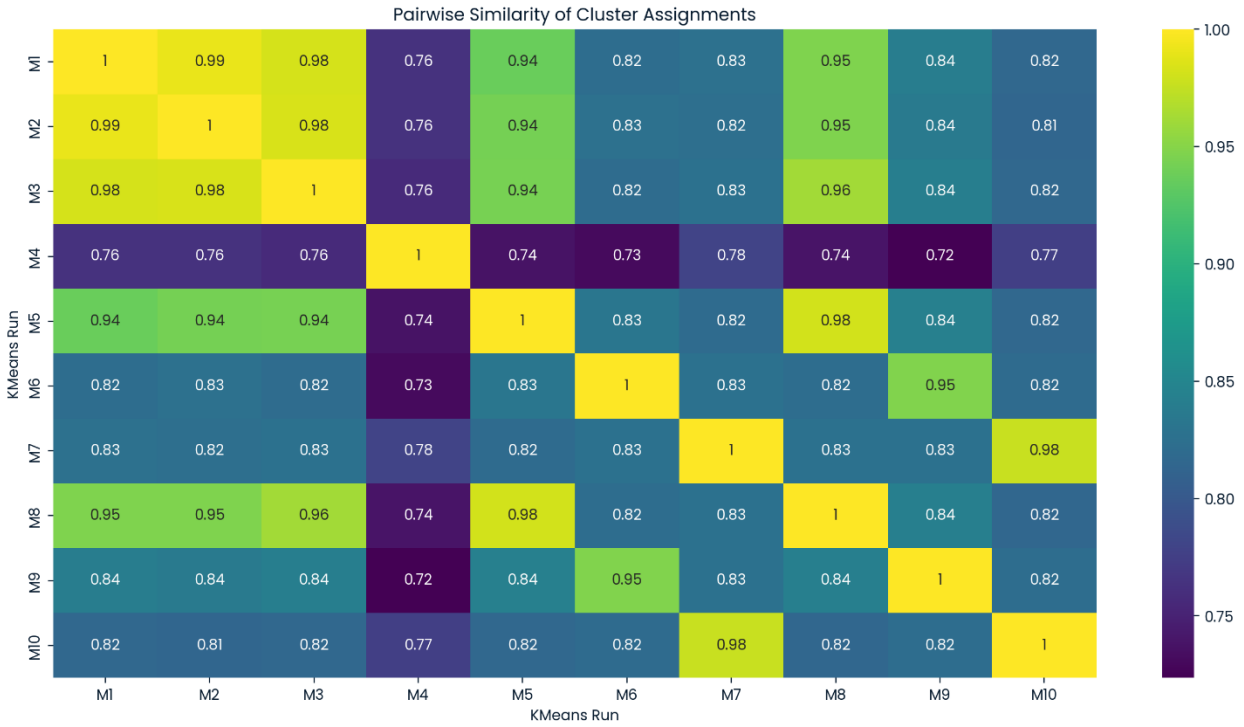


Figure 17 Heatmap comparing M1, M2, ... M10 with each other.

Question 2b: Are the results of k-means on this dataset robust to the initial cluster centroid values chosen?

The average similarity between different k-means runs 0.86. The results of k-means on this dataset are not robust to the initial cluster centroid values chosen, assuming we set the threshold for robustness at 90%.

Q2c: Out of all of the pairs of observations that were recorded to be in the same cluster in at least one run, what percentage were in the same cluster in all ten?

58.56%

Q3a, Perform clustering on the reduced-dimensional data using the single linkage clustering algorithm with k clusters and create a matrix S for the result in the same way that you did M and P.

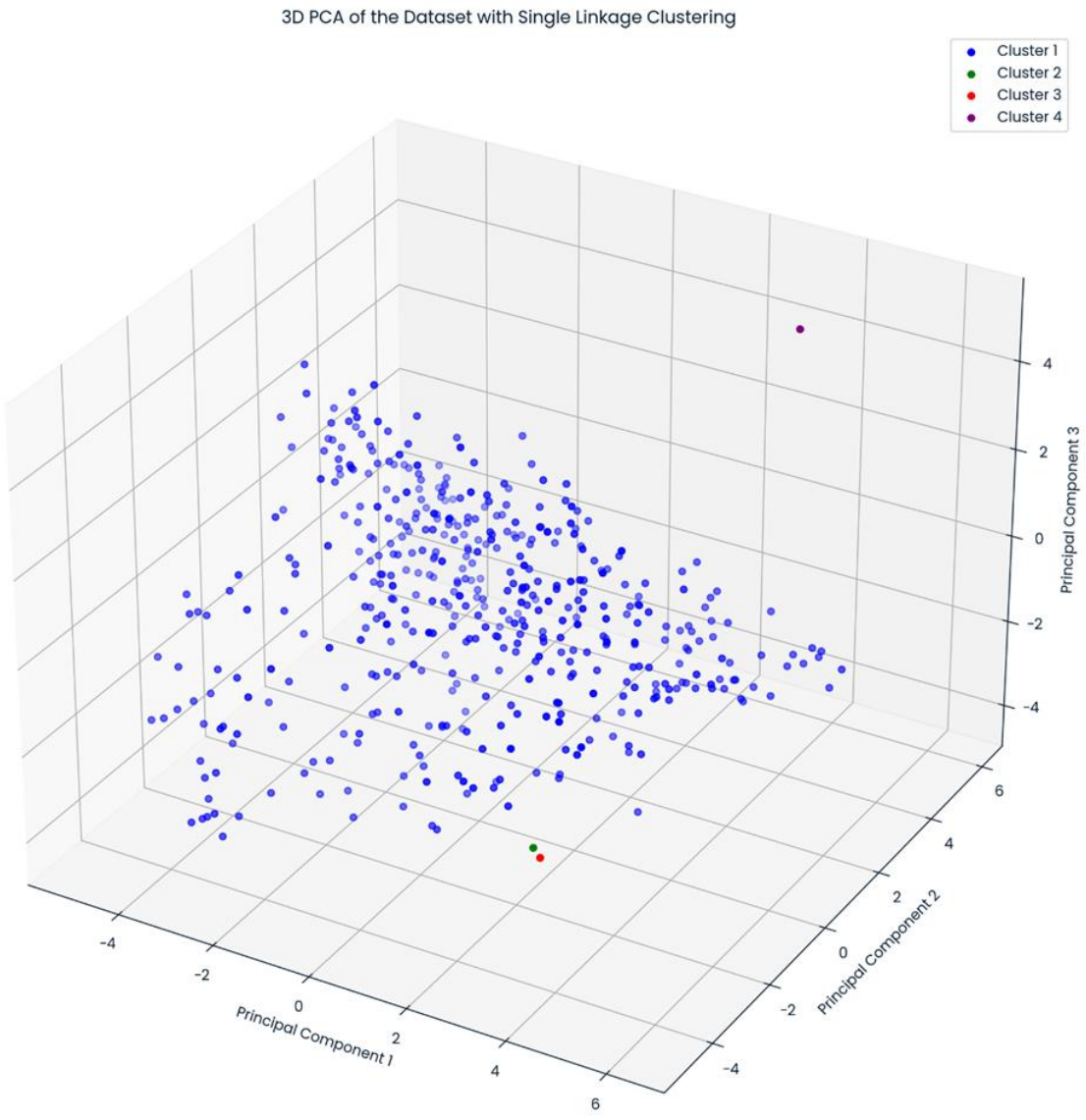


Figure 18 3D visualization of single-linkage clustering on reduced data.

```
S_matrix
```

```
array([[1, 1, 1, ..., 1, 1, 1],
       [1, 1, 1, ..., 1, 1, 1],
       [1, 1, 1, ..., 1, 1, 1],
       ...,
       [1, 1, 1, ..., 1, 1, 1],
       [1, 1, 1, ..., 1, 1, 1],
       [1, 1, 1, ..., 1, 1, 1]])
```

Question 3b: Determine how many entries S has in common with P.

Entries common: 57147

Percentage of entries common: 27.12%

Question 3c: Additionally, create two scatterplots of the observations in the reduced-dimensional dataset, each with the first two principal components on the axes. The points corresponding to the observations should be coloured according to which cluster they were assigned to by k-means in the first scatterplot, and single-linkage clustering in the second scatterplot.

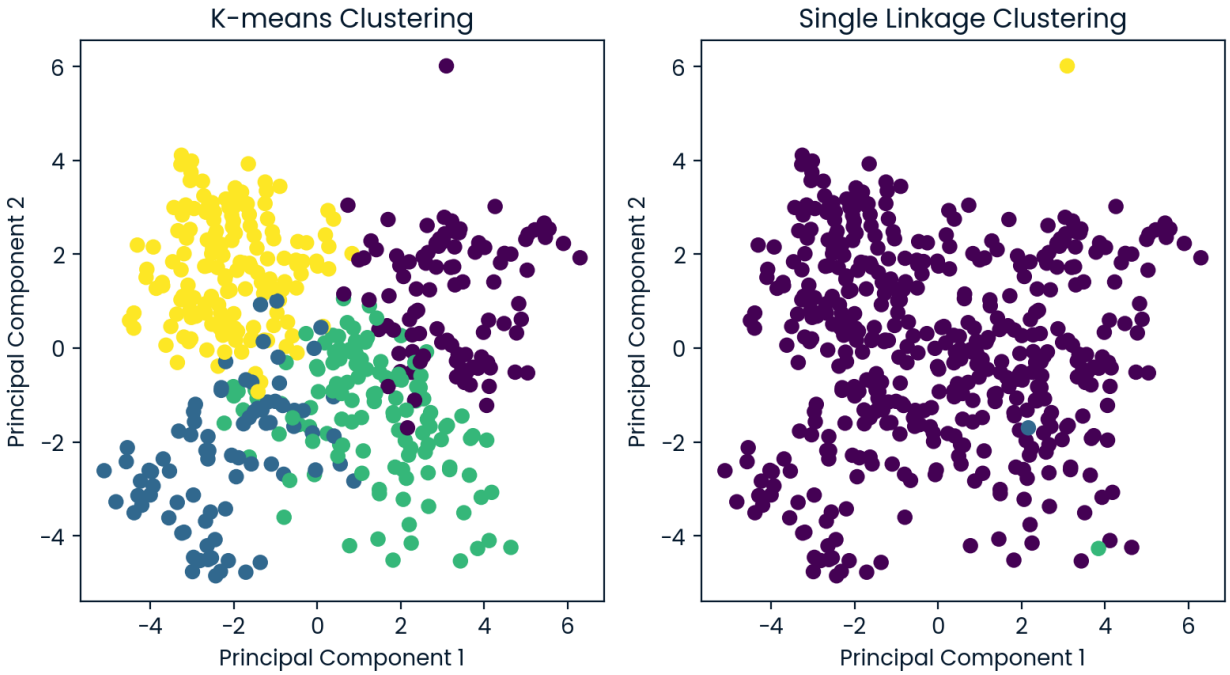


Figure 19 2D graphs comparing K-means and Single-linkage clustering on PCA reduced data.

Question 3d, What kinds of cluster shapes do k-means and single-linkage clustering tend to produce?

1. K-means Clustering:

- Cluster Shape: K-means clustering tends to produce clusters that are roughly spherical or circular. This is because the algorithm minimizes the variance within each cluster, effectively creating clusters that are equidistant from the cluster centroid.
- Limitations: K-means may struggle with clusters of varying sizes and densities, and it is not well-suited for identifying clusters with irregular shapes.

2. Single-Linkage Clustering:

- Cluster Shape: Single-linkage clustering, also known as the nearest-neighbor method, can produce clusters of arbitrary shapes. This method links clusters based on the minimum distance between points in different clusters, allowing them to form elongated or irregularly shaped clusters.
- Limitations: Single-linkage clustering is sensitive to noise and outliers, which can lead to the chaining phenomenon where clusters are formed by linking points in a chain-like manner.

In summary, K-means clustering is best for spherical clusters, while single-linkage clustering can identify clusters with more complex shapes but may be affected by noise and outliers.

Question 4a: In a Gaussian mixture model, each point in a dataset is assumed to have been drawn from one of many different Gaussian distributions. Hence, these distributions can be thought of as overlapping clusters. The expectation-maximization algorithm can be used to estimate the means and standard deviations of the distributions in a Gaussian mixture model (the centroids and sizes of the clusters, to put it a different way). Use the expectation-maximization algorithm with k clusters to probabilistically assign observations in the reduced-dimensional data to distributions/clusters. Create a new matrix E whose entries eij take values in [0,1]: for observations i and j, eij is the probability that i and j are in the same cluster.

```
print(E)
print(E.shape)
# E is now the matrix where E[i, j] is the probability that observations i and j are in the same cluster
```

```
[[9.80072507e-01 1.35645643e-02 2.13703202e-09 ... 1.00650519e-02
  1.00661729e-02 1.00650519e-02]
 [1.35645643e-02 9.92882699e-01 3.73593351e-11 ... 9.96428595e-01
  9.96427459e-01 9.96428595e-01]
 [2.13703202e-09 3.73593351e-11 9.99999231e-01 ... 2.97568826e-11
  2.97593180e-11 2.97568827e-11]
 ...
 [1.00650519e-02 9.96428595e-01 2.97568826e-11 ... 1.00000000e+00
  9.99998856e-01 1.00000000e+00]
 [1.00661729e-02 9.96427459e-01 2.97593180e-11 ... 9.99998856e-01
  9.99997712e-01 9.99998856e-01]
 [1.00650519e-02 9.96428595e-01 2.97568827e-11 ... 1.00000000e+00
  9.99998856e-01 1.00000000e+00]]
(2207, 2207)
```

Question 4b: What percentage of observations have at least two different clusters that they are assigned to with probability >1%?

25.51%. This percentage is a measure of how many data points are not strongly associated with a single cluster, indicating some level of uncertainty or overlap between clusters. This percentage can provide insights into the clustering quality. A high percentage might indicate that many data points are not assigned to a single cluster, suggesting that the clusters are not well separated. Conversely, a low percentage would indicate that most data points are strongly associated with a single cluster, suggesting well-defined clusters.

Question 5a: t-distributed stochastic neighbour embedding (t-SNE) is arguably a tool for data visualization more than clustering, but it can still be used to find interesting patterns in data that would ordinarily be hard to uncover. One advantage of t-SNE is that it can find clusters in high-dimensional datasets whose members are similar to one another but do not fall neatly into one specific region of the vector space that they are in. Run t-SNE on the full-dimensional dataset (not the reduced-dimensional one) with tuning parameters of your choice. (If you use Python, you can find an implementation of t-SNE in the sci-kit-learn library.) Plot the results.

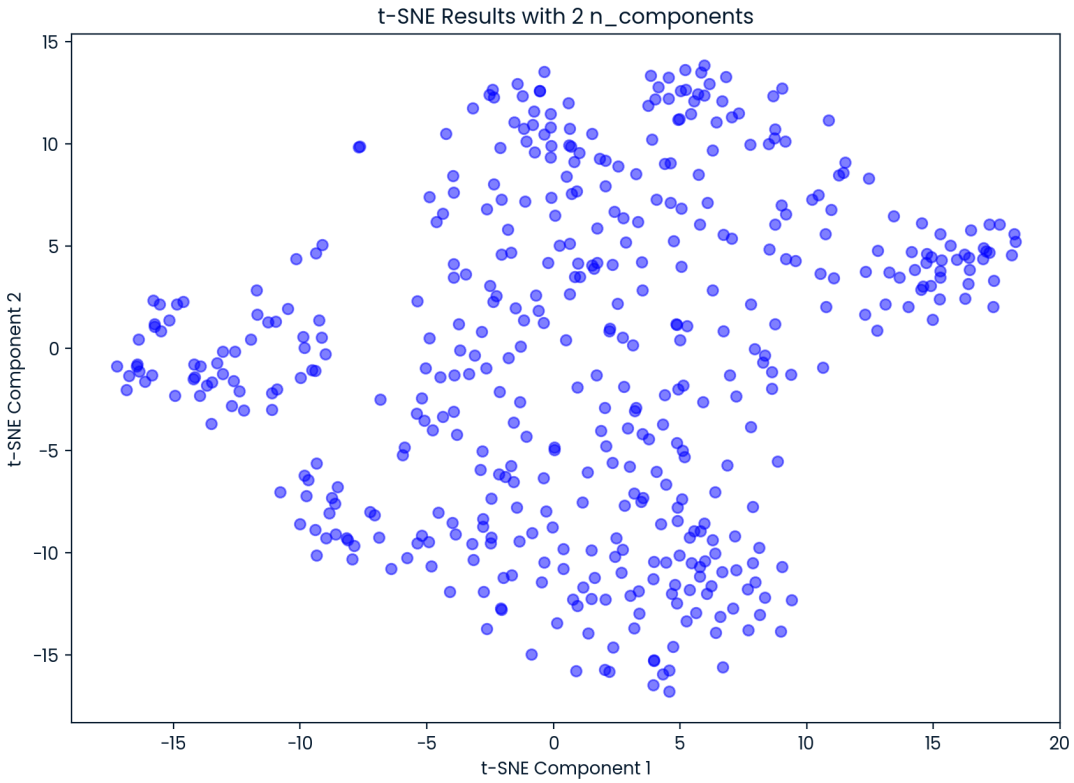


Figure 20 Graph of reduced data to 2 components using t-SNE.

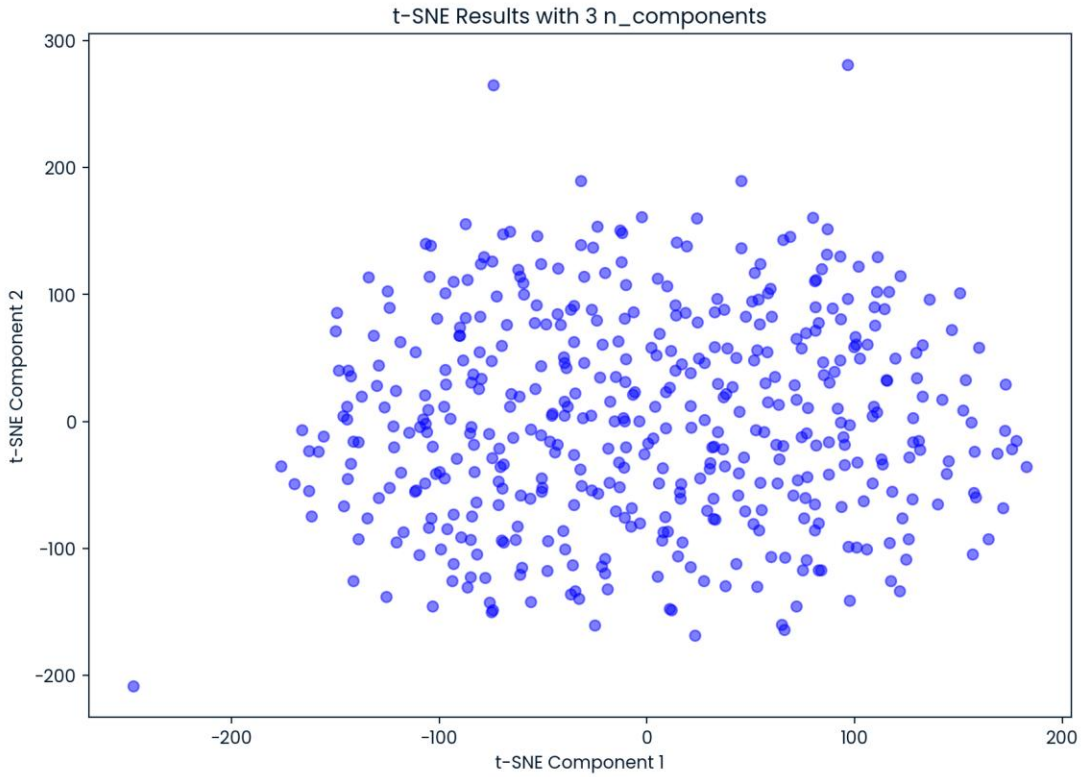


Figure 21 Graph of reduced data to 3 components using t-SNE.

Question 5b: How similar does the plot look to the ones that you generated in part c? Do the observations that are located in the same part of the t-SNE plot also tend to cluster together when other algorithms are used?

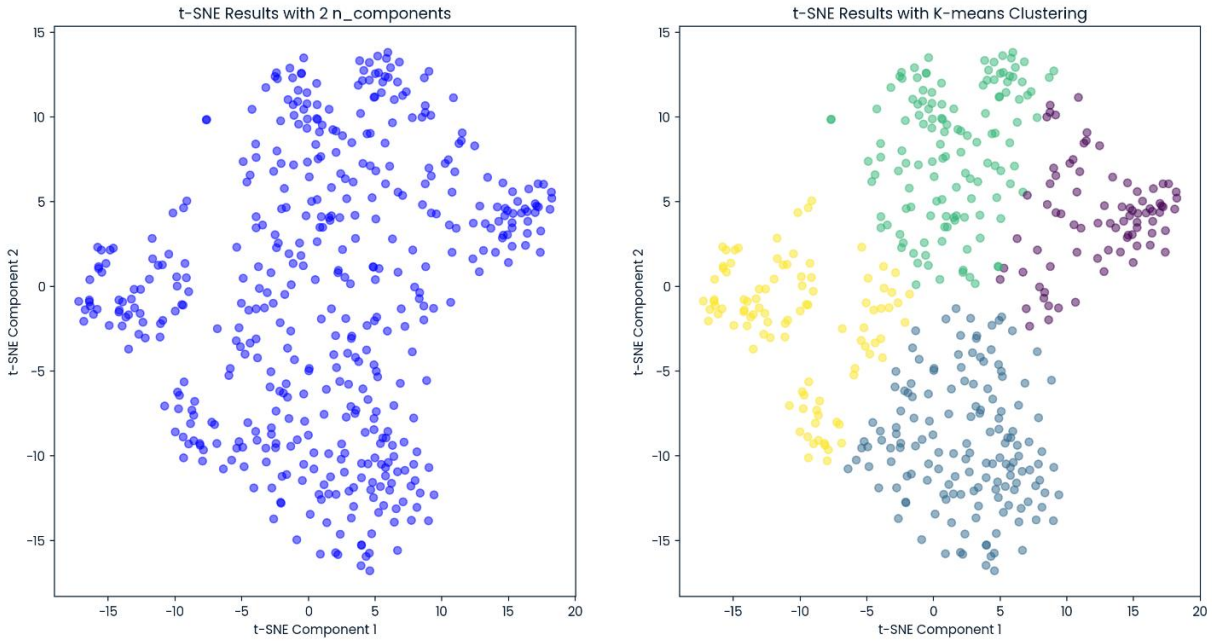


Figure 22 Graphs of reduced data to 2 components using t-SNE and K-means Clustering.

We observe more refined results of K-means clustering with t-SNE reduced data with 2 components. We can clearly observe properly divided clusters.

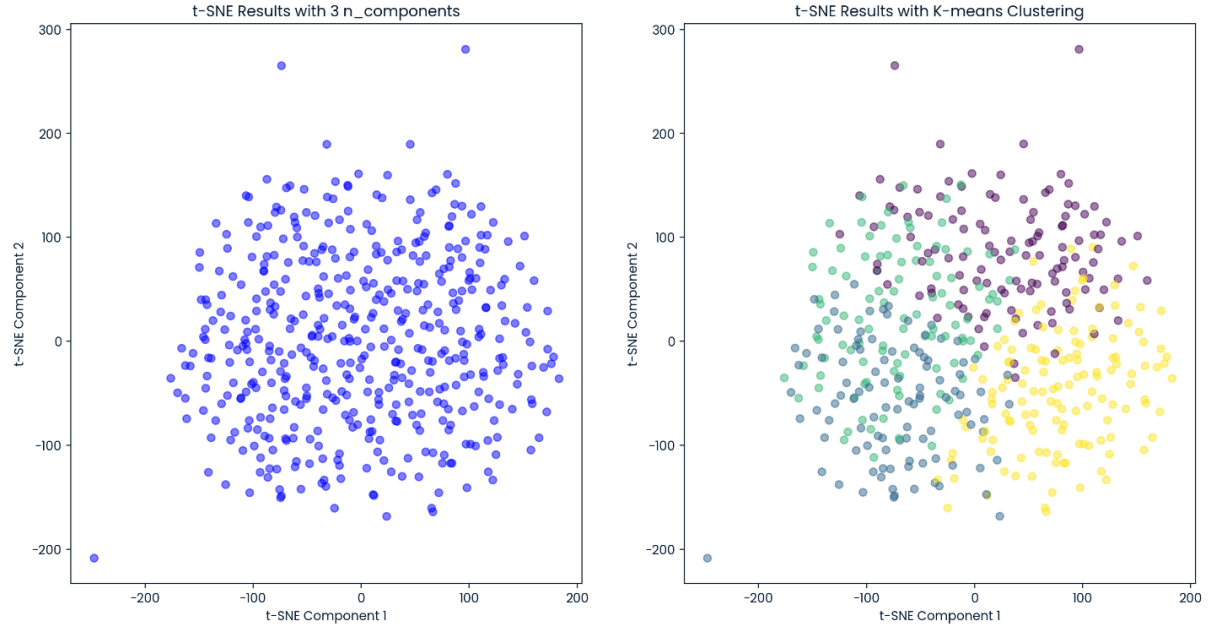


Figure 23 Graphs of reduced data to 3 components using t-SNE and K-means Clustering.

The results with t-SNE reduced data with 3 components are not favorable as we observe overlapping clusters.

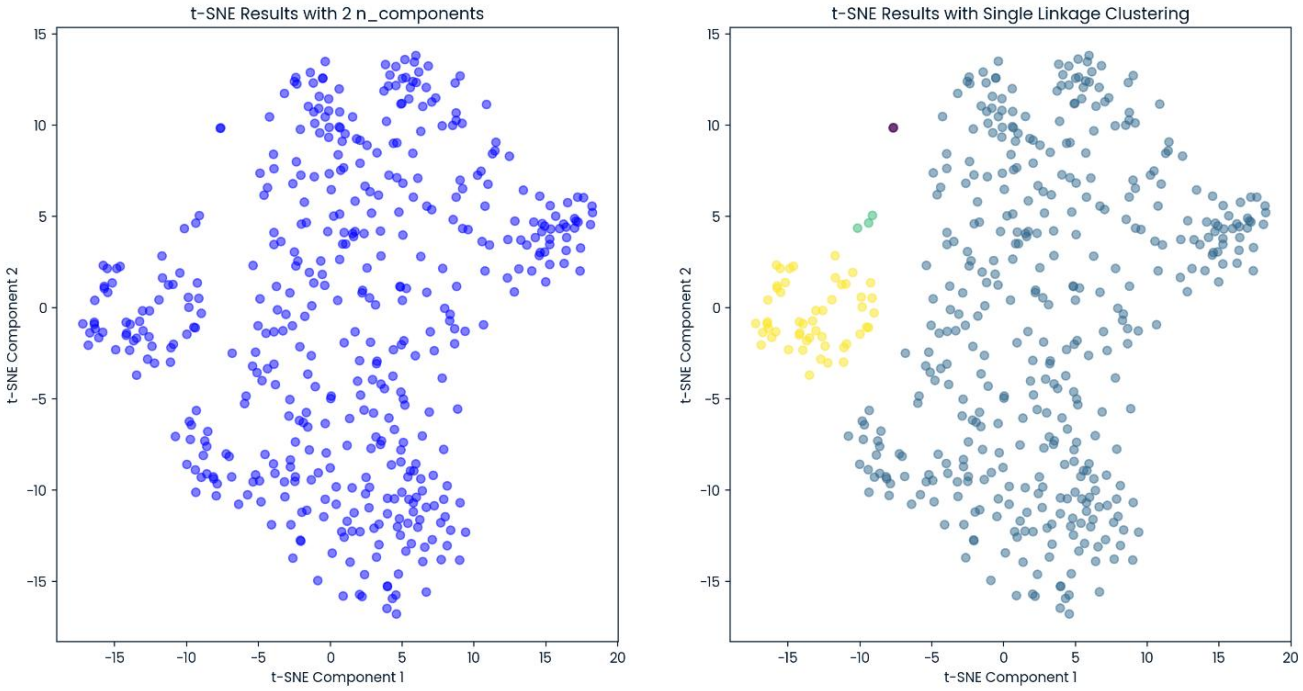


Figure 24 Graphs of reduced data to 2 components using t-SNE and Single-Linkage Clustering.

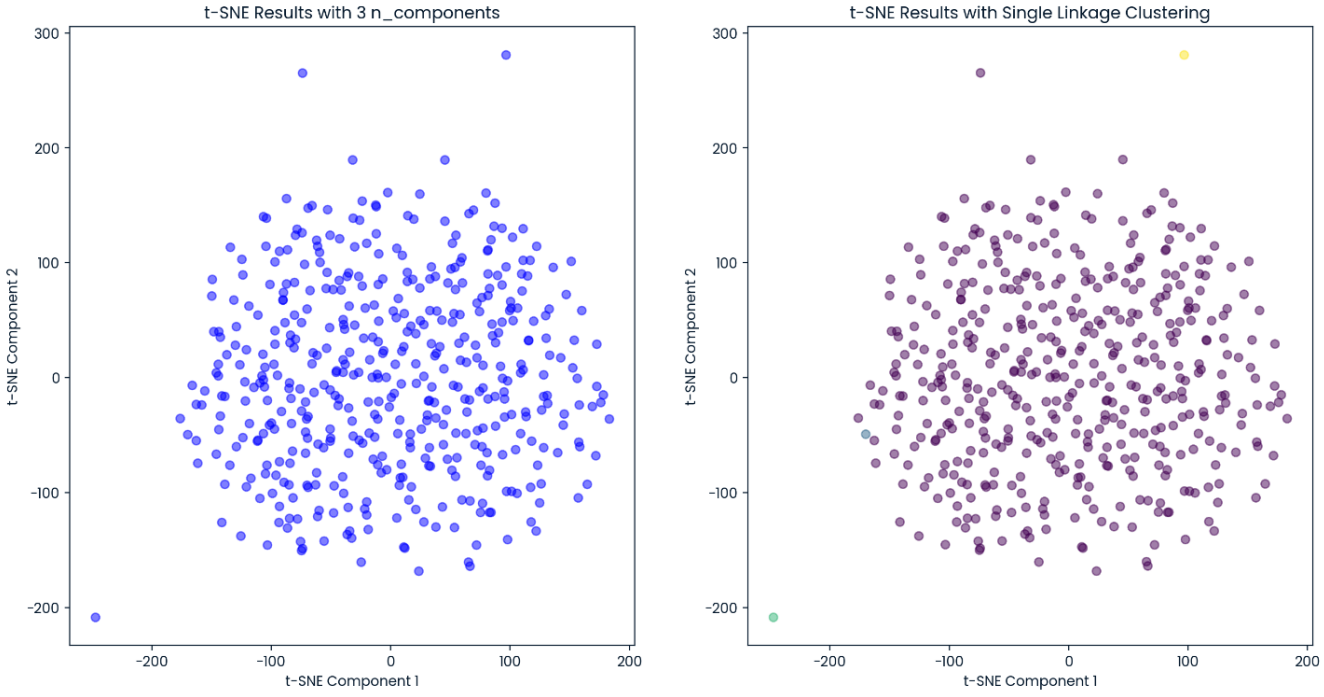


Figure 25 Graphs of reduced data to 3 components using t-SNE and Single-Linkage Clustering.

Similar to K-means clustering, results with t-SNE data reduced to 2 components produced much better single-linkage clustering results compared to t-SNE data reduced to 3 components.

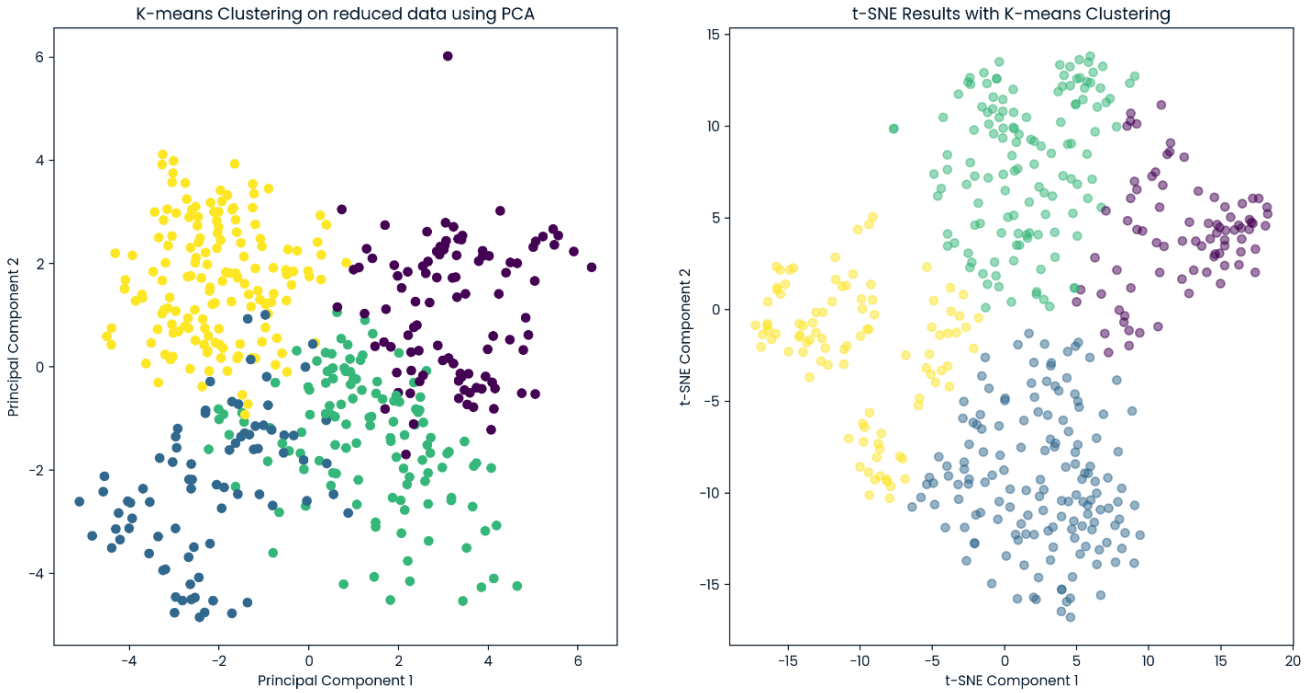


Figure 26 Graphs of K-means clustering on PCA and t-SNE reduced data.

The observations that are located in the same part of the t-SNE plot often tend to cluster together when K-means is applied.

This is evident from the visualization above:

In the PCA plot with K-means clustering (left), the data points are divided into distinct clusters based on principal components. In the t-SNE plot with K-means clustering (right), a similar clustering pattern is visible, although t-SNE emphasizes preserving local similarities in the data.

Despite the dimensionality reduction and projection differences between PCA and t-SNE, both methods maintain enough of the structure of the data to show that points grouped together in one method tend to stay together in the other. Nonetheless, exact groupings and boundaries might differ slightly due to how each algorithm processes the data.

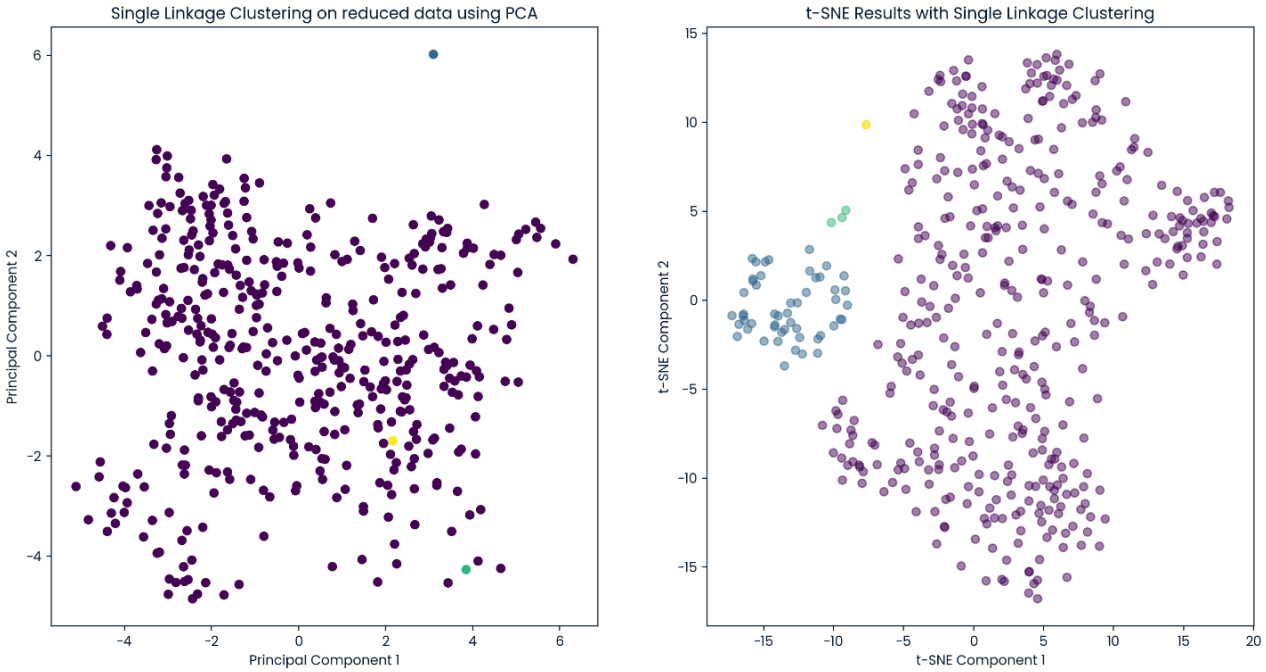


Figure 27 Graphs of Single-Linkage clustering on PCA and t-SNE reduced data.

Observations in the same region of the t-SNE plot do not consistently form distinct clusters when single-linkage clustering is applied. This method often results in one large cluster with a few outliers, as demonstrated in both the PCA and t-SNE visualizations, due to its chaining effect. As a result, the local groupings identified in t-SNE are not accurately represented in the clustering.

In conclusion, When K-means is applied, observations that appear in the same area of the t-SNE plot typically tend to group effectively. In contrast, when single-linkage is used, observations within the same region of the t-SNE plot do not cluster together as efficiently. This is because single-linkage clustering often has difficulty managing datasets that do not contain well-separated groups.

5. Discussion

The techniques and models employed in this project successfully addressed the primary questions regarding clustering, dimensionality reduction, and their effectiveness in exposing the structure within the data.

The effectiveness of the models was evident through satisfactory results, as k-means clustering, single-linkage clustering, Gaussian Mixture Models (GMM), and t-SNE successfully identified clusters in both original and reduced-dimensional datasets. PCA and t-SNE proved effective in reducing dimensionality while retaining essential patterns (such as variance), leading to interpretable visualizations and improved computational efficiency. The elbow method and scree plot were successful in determining the optimal number of clusters and components. Comparison matrices such as M, P, and S demonstrated variations in data structure and clustering outcomes across different algorithms, highlighting their consistency and robustness.

5.1. Limitations

The limitations of clustering models involve sensitivity to parameters, as algorithms like k-means can yield varying results based on centroid initialization as observed in our results. While PCA effectively

reduces dimensionality, it assumes linear relationships and may miss nonlinear patterns. Moreover, k-means presumes spherical cluster shapes, which can hinder performance on non-convex datasets, and single-linkage clustering may suffer from chaining effects, resulting in overly elongated clusters. Additionally, algorithms like t-SNE and GMM can be computationally expensive when applied to larger datasets, posing scalability challenges.

5.2. **Possible improvements if more time was available**

Incorporating advanced techniques such as density-based clustering algorithms like DBSCAN can effectively address irregular cluster shapes and sensitivity to noise, while deep learning approaches like autoencoders may better capture complex patterns. Automating hyperparameter selection for clustering algorithms is recommended for increased robustness. Using advanced cluster validation metrics, such as the Davies-Bouldin Index or Adjusted Rand Index, could further refine the analysis. Exploring hybrid models that combine methods like k-means with hierarchical clustering or ensemble techniques could yield more robust results. Finally, effective handling of outliers through preprocessing steps is essential, especially for density-sensitive methods.

5.3. **Future Research**

To enhance our work, we could explore alternative datasets from various domains, such as customer behaviour and genomics, to assess the generalizability of our models. Integrating domain-specific insights could improve the interpretability and relevance of our clustering results. Developing real-time clustering pipelines for dynamic datasets would broaden the applicability of these methods in fields like e-commerce and healthcare. Finally, creating interactive visualizations with tools like Tableau or Plotly could significantly enhance usability for stakeholders involved in cluster analysis.

In conclusion, although the models used in this project effectively addressed the specified questions and provided valuable insights, they do have limitations. Improving these weaknesses by incorporating advanced methods and adapting the models to specific domains would greatly enhance their scalability, robustness, and utility in real-world applications.

6. Conclusion

This report examined the effectiveness of clustering algorithms and dimensionality reduction techniques on high-dimensional datasets to uncover relationships and patterns. Utilizing methods like k-means clustering, single-linkage clustering, and Gaussian Mixture Models (GMM), alongside dimensionality reduction tools such as PCA and t-SNE, we identified clusters and visualized data structures.

The analysis revealed that while clustering algorithms can group similar data points effectively, their performance is influenced by data characteristics such as cluster shape and density. For example, k-means works best with spherical clusters, while single-linkage clustering is adept at identifying irregular shapes. PCA and t-SNE aid in visualizing high-dimensional data but come with limitations like sensitivity to initialization as observed in our results.

Issues such as scalability and parameter sensitivity indicate potential for further enhancement. Future work could explore real-time clustering systems, domain-specific models, and diverse datasets to broaden the applicability of these techniques in data-driven decision-making across fields like genomics and e-commerce.

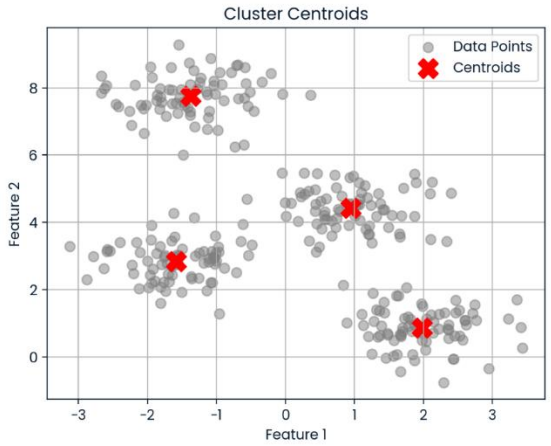
7. References

1. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.
2. Han, J., Kamber, M., & Pei, J. (2011). *Data Mining: Concepts and Techniques*. Elsevier.
3. Jolliffe, I. T. (2002). *Principal Component Analysis*. Springer.
4. Lloyd, S. (1982). Least squares quantization in PCM. *IEEE Transactions on Information Theory, 28*(2), 129-137.
5. Scikit-learn Documentation: K-Means and PCA Implementation. Retrieved from https://scikit-learn.org
6. Tan, P.-N., Steinbach, M., & Kumar, V. (2018). *Introduction to Data Mining*. Pearson.
7. van der Maaten, L., & Hinton, G. (2008). Visualizing Data Using t-SNE. *Journal of Machine Learning Research*.

8. Appendices

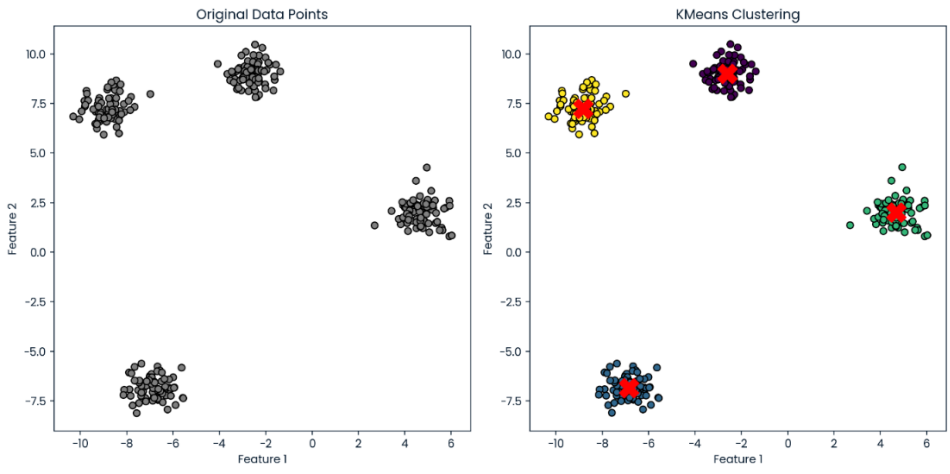
Python code for making Cluster Centroids diagram using dummy data:

```
[19]
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans

# Generate sample data
X, _ = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=0)

# Fit KMeans
kmeans = KMeans(n_clusters=4)
kmeans.fit(X)
centroids = kmeans.cluster_centers_

# Plot the data points
plt.scatter(X[:, 0], X[:, 1], s=50, c='gray', marker='o', alpha=0.5, label='Data
Points')

# Plot the centroids
plt.scatter(centroids[:, 0], centroids[:, 1], s=200, c='red', marker='X',
label='Centroids')

# Add labels and title
plt.title('Cluster Centroids')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.grid(True)
plt.show()
```



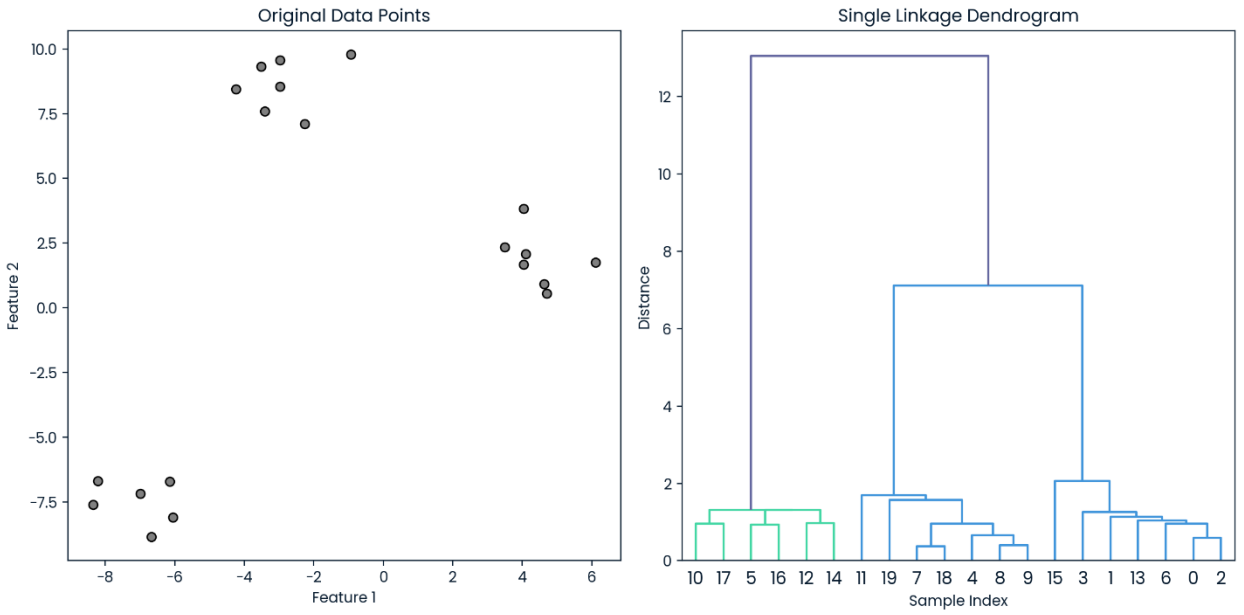
Python code for making graphs depicting k-means clustering using dummy data:

```python
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans

# Generate a synthetic dataset for clustering
X, _ = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=42)

# Initialize KMeans clustering algorithm
kmeans = KMeans(n_clusters=4, random_state=42)

# Fit the KMeans algorithm and predict the cluster labels
y_pred = kmeans.fit_predict(X)

# Plot the original data points
plt.figure(figsize=(12, 6))

# Plot the data points with cluster assignments
plt.subplot(1, 2, 1)
plt.scatter(X[:, 0], X[:, 1], c='gray', edgecolor='k', marker='o')
plt.title('Original Data Points')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')

# Plot the clustered data points
plt.subplot(1, 2, 2)
plt.scatter(X[:, 0], X[:, 1], c=y_pred, cmap='viridis', edgecolor='k', marker='o')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=300, c='red', marker='X')
plt.title('KMeans Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')

plt.tight_layout()
plt.show()
```



Python code for graphically visualizing single-linkage clustering using dummy data:

```python
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.datasets import make_blobs

# Generate a synthetic dataset for clustering
X, _ = make_blobs(n_samples=20, centers=3, cluster_std=1.00, random_state=42)

# Perform single linkage hierarchical clustering
Z = linkage(X, method='single')

# Plot the original data points
plt.figure(figsize=(12, 6))

# Plot the data points
plt.subplot(1, 2, 1)
plt.scatter(X[:, 0], X[:, 1], c='gray', edgecolor='k', marker='o')
plt.title('Original Data Points')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')

# Plot the dendrogram
plt.subplot(1, 2, 2)
dendrogram(Z)
plt.title('Single Linkage Dendrogram')
plt.xlabel('Sample Index')
plt.ylabel('Distance')

plt.tight_layout()
plt.show()
```



Python code along with a snapshot of the dataset:

```
#Transpose the data to have observations as rows and features as columns
data = data.T
print(data.head())
```

```
Dimensions: (459, 2207)
          0         1    2         3     ...      2203      2204      2205      2206
0  0.048734  0.156177  0.0  0.504464  ...  0.068796  0.068681  0.101562  0.082707
1  0.111323  0.074072  0.0  0.571429  ...  0.194103  0.197802  0.117188  0.150376
2  0.064023  0.080148  0.0  0.473214  ...  0.181818  0.162088  0.195312  0.132832
3  0.058767  0.131366  0.0  0.410714  ...  0.031941  0.024725  0.109375  0.162907
4  0.050645  0.147004  0.0  0.446429  ...  0.140049  0.131868  0.148438  0.170426

[5 rows x 2207 columns]
```

```
print(data.describe())
```

```
                0           1      2     ...        2204        2205        2206
count  459.000000  459.000000  459.0  ...  459.000000  459.000000  459.000000
mean     0.080778    0.120589    0.0  ...    0.251107    0.204572    0.196368
std      0.077852    0.125591    0.0  ...    0.159509    0.157142    0.120650
min      0.000000    0.000000    0.0  ...    0.000000    0.000000    0.000000
25%      0.049212    0.050928    0.0  ...    0.127747    0.101562    0.110276
50%      0.061634    0.086378    0.0  ...    0.225275    0.156250    0.177945
75%      0.084090    0.145699    0.0  ...    0.329670    0.242188    0.250627
max      1.000000    1.000000    0.0  ...    1.000000    1.000000    1.000000

[8 rows x 2207 columns]
```

Python code to check for empty and duplicate values:

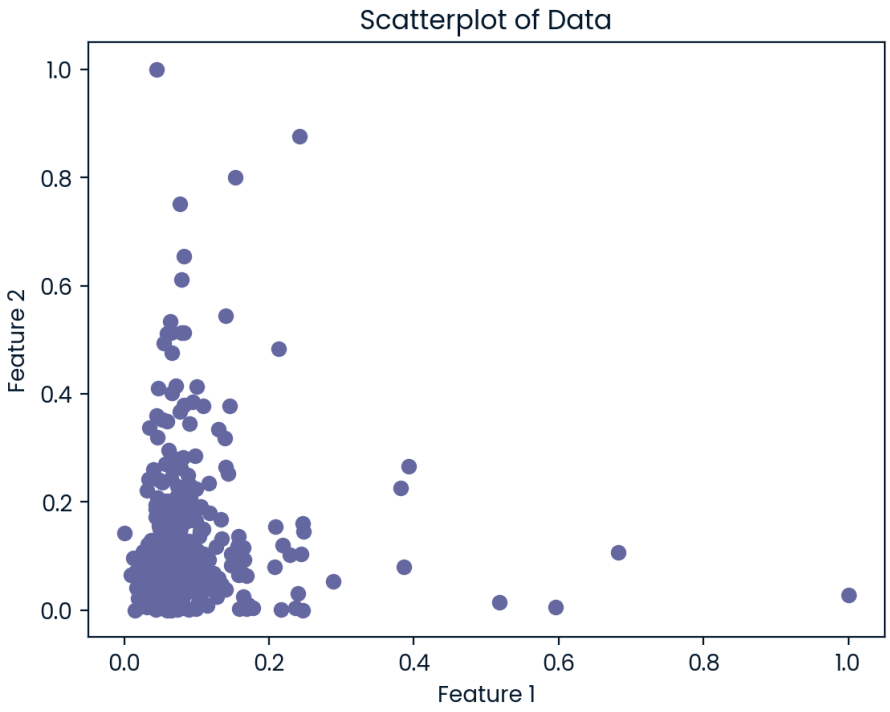
```
print("Emptyvalues:␣", data.isna().sum().sum())
print("Duplicates:␣", data.duplicated().sum())
```

```
Emptyvalues:␣ 0
Duplicates:␣ 0
```

Python code for visualizing the data using a scatter plot:

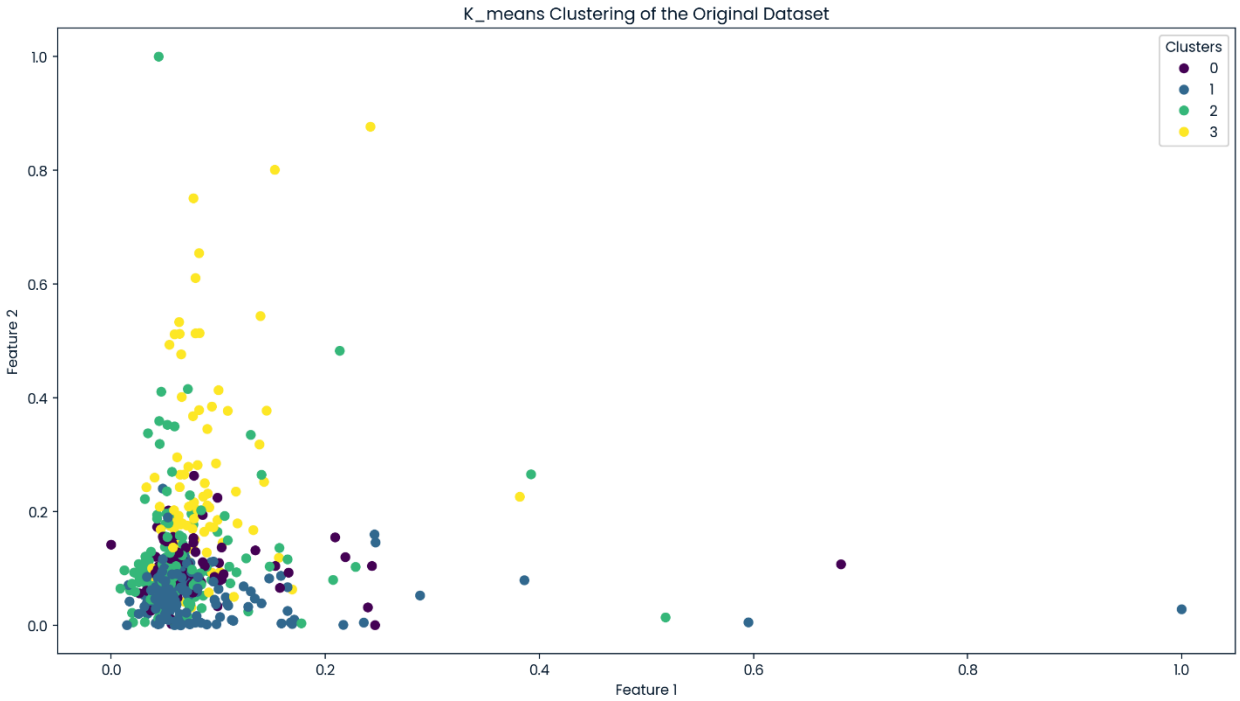
```
import matplotlib.pyplot as plt

plt.scatter(data.iloc[:, 0], data.iloc[:, 1]) # for naively visualizing the entire data in 2 dimensional space
plt.title('Scatterplot of Data')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```

Run cancelled



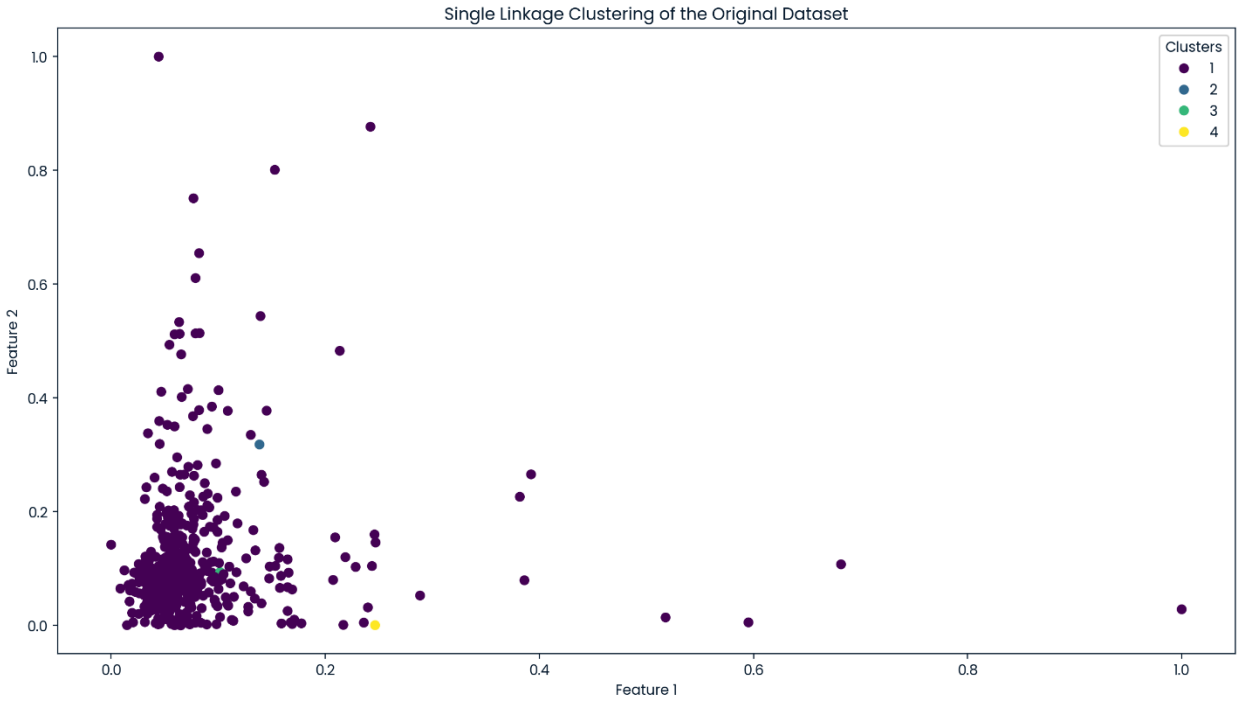
Python code for k-means clustering on the original data set:

```python
import matplotlib.pyplot as plt

# Assuming 'data' is the original dataset and 'clusters' contains the cluster labels
fig, ax = plt.subplots(figsize=(15, 8))

# Scatter plot of the original data with cluster coloring
scatter = ax.scatter(data.iloc[:, 0], data.iloc[:, 1], c=clusters, cmap='viridis', marker='o')

# Add legend
legend1 = ax.legend(*scatter.legend_elements(), title="Clusters")
ax.add_artist(legend1)

ax.set_title('K_means Clustering of the Original Dataset')
ax.set_xlabel('Feature 1')
ax.set_ylabel('Feature 2')

plt.show()
```



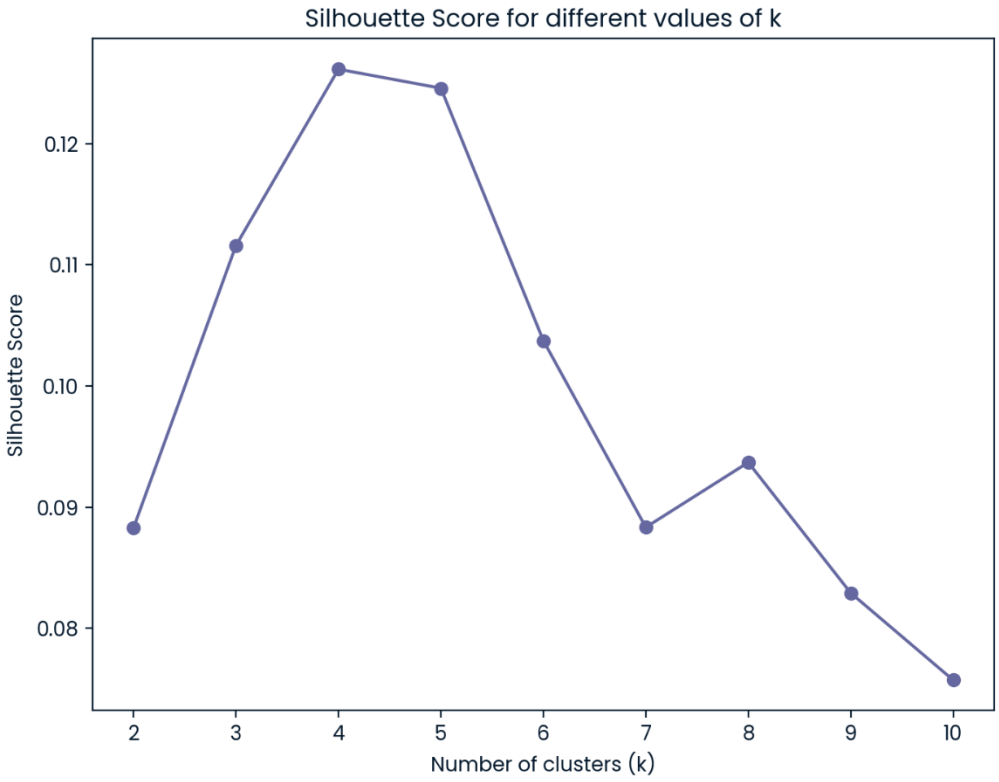
Python code for single-linkage clustering on the original data set:

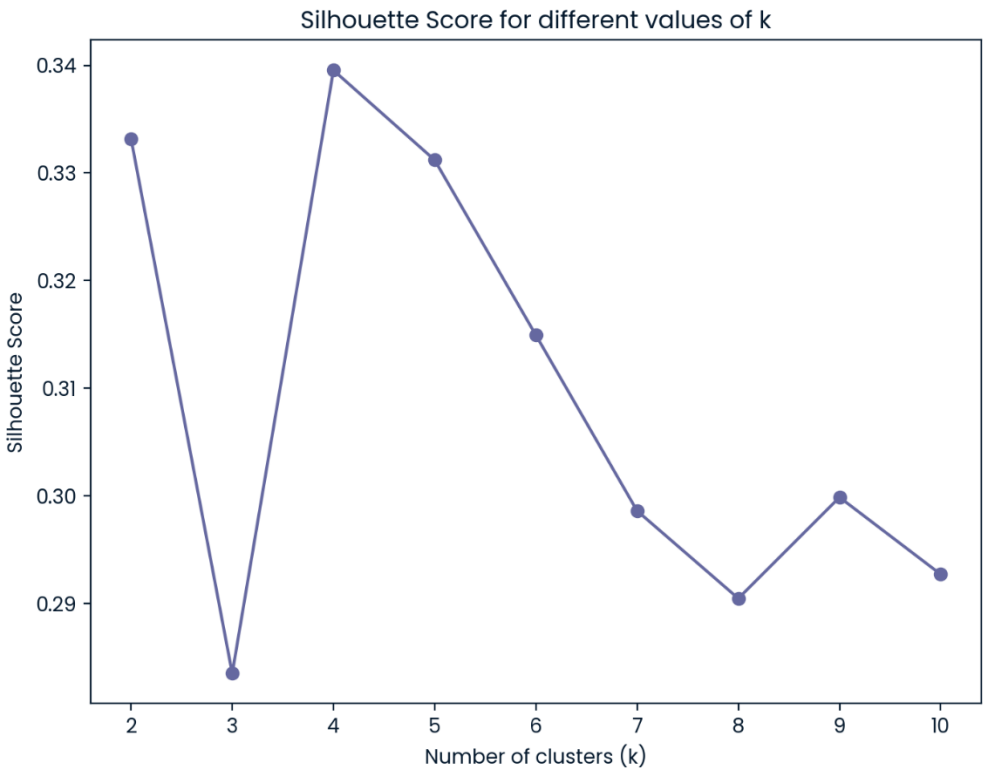
```
import matplotlib.pyplot as plt                                                    [6]
from scipy.cluster.hierarchy import linkage, fcluster
import pandas as pd
import numpy as np
# Load the data
file_path = 'gvrd-rangestd.csv'
data = pd.read_csv(file_path, header=None)
# Transpose the data to have observations as rows and features as columns
data = data.T
# Perform single linkage clustering
Z = linkage(data, method='single')
clusters = fcluster(Z, t=4, criterion='maxclust')

# Add the cluster labels to the original data
data_with_clusters = data.copy()
data_with_clusters['Cluster'] = clusters

# Generate an n × n square matrix M
n = data.shape[0]
M = np.zeros((n, n), dtype=int)
for i in range(n):
    for j in range(n):
        if clusters[i] == clusters[j]:
            M[i, j] = 1

# Scatter plot of the original data with cluster coloring
fig, ax = plt.subplots(figsize=(15, 8))
scatter = ax.scatter(data.iloc[:, 0], data.iloc[:, 1], c=clusters, cmap='viridis', marker='o')

# Add legend
legend1 = ax.legend(*scatter.legend_elements(), title="Clusters")
ax.add_artist(legend1)

ax.set_title('Single Linkage Clustering of the Original Dataset')
ax.set_xlabel('Feature 1')
ax.set_ylabel('Feature 2')
plt.show()
```



Python code for Silhouette Score graphs to determine optimal cluster size:

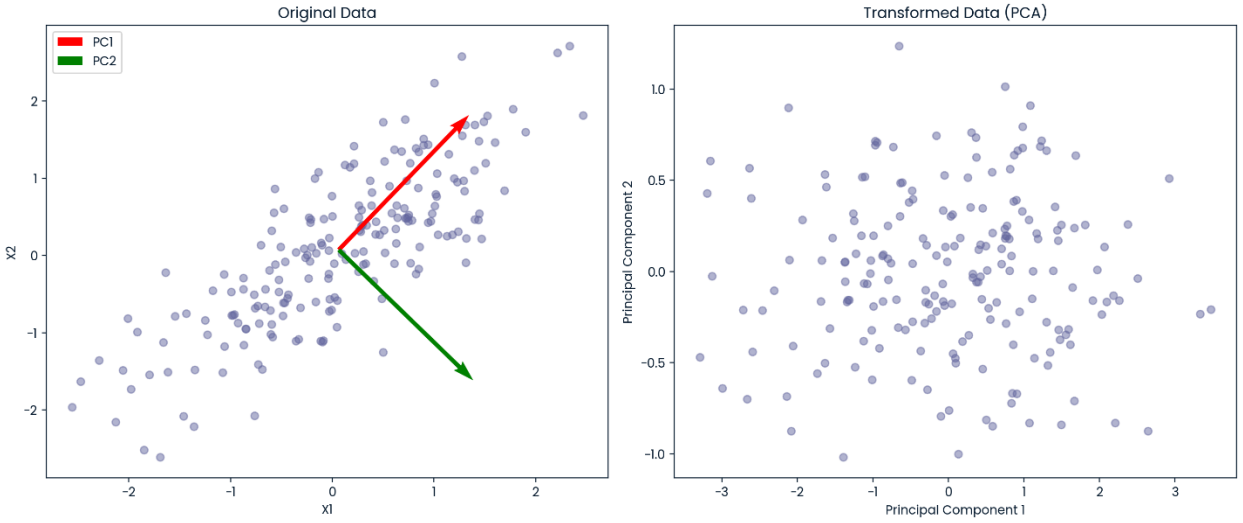
```
import matplotlib.pyplot as plt
from sklearn.metrics import silhouette_score

# Function to calculate the silhouette score for different values of k
def find_optimal_clusters(data, max_k):
    iters = range(2, max_k+1)
    s_scores = []

    for k in iters:
        kmeans = KMeans(n_clusters=k, random_state=0)
        kmeans.fit(data)
        s_scores.append(silhouette_score(data, kmeans.labels_))

    plt.figure(figsize=(8, 6))
    plt.plot(iters, s_scores, marker='o')
    plt.xlabel('Number of clusters (k)')
    plt.ylabel('Silhouette Score')
    plt.title('Silhouette Score for different values of k')
    plt.show()

# Find the optimal number of clusters for the original data
find_optimal_clusters(data, 10)

# Find the optimal number of clusters for the reduced-dimensional data
find_optimal_clusters(data_reduced, 10)
```


Silhouette Score for different values of k
Silhouette Score
0.12
0.11
0.10
0.09
0.08
2
3
4
5
6
7
8
9
10
Number of clusters (k)


Silhouette Score for different values of k
Silhouette Score
0.34
0.33
0.32
0.31
0.30
0.29
2
3
4
5
6
7
8
9
10
Number of clusters (k)

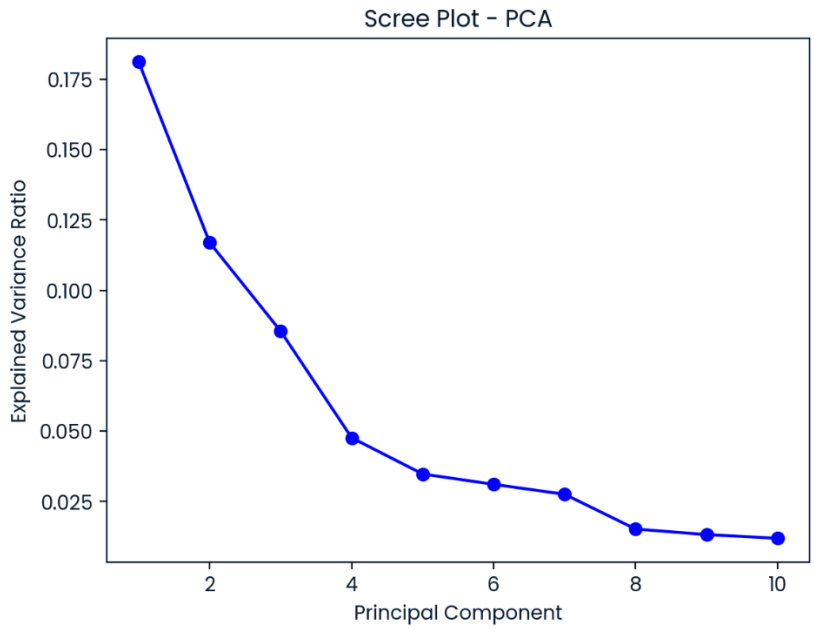
Python code for graphs showing the implementation of PCA on dummy data:

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# Generate synthetic data
np.random.seed(0)
mean = [0, 0]
cov = [[1, 0.8], [0.8, 1]]  # diagonal covariance
data = np.random.multivariate_normal(mean, cov, 200)

# Perform PCA
pca = PCA(n_components=2)
pca.fit(data)
transformed_data = pca.transform(data)

# Plot original data
plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
plt.scatter(data[:, 0], data[:, 1], alpha=0.5)
plt.title('Original Data')
plt.xlabel('X1')
plt.ylabel('X2')

# Plot principal components
origin = np.mean(data, axis=0)
plt.quiver(*origin, pca.components_[0, 0], pca.components_[0, 1], color='r', scale=3, label='PC1')
plt.quiver(*origin, pca.components_[1, 0], pca.components_[1, 1], color='g', scale=3, label='PC2')
plt.legend()

# Plot transformed data
plt.subplot(1, 2, 2)
plt.scatter(transformed_data[:, 0], transformed_data[:, 1], alpha=0.5)
plt.title('Transformed Data (PCA)')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')

plt.tight_layout()
plt.show()
```



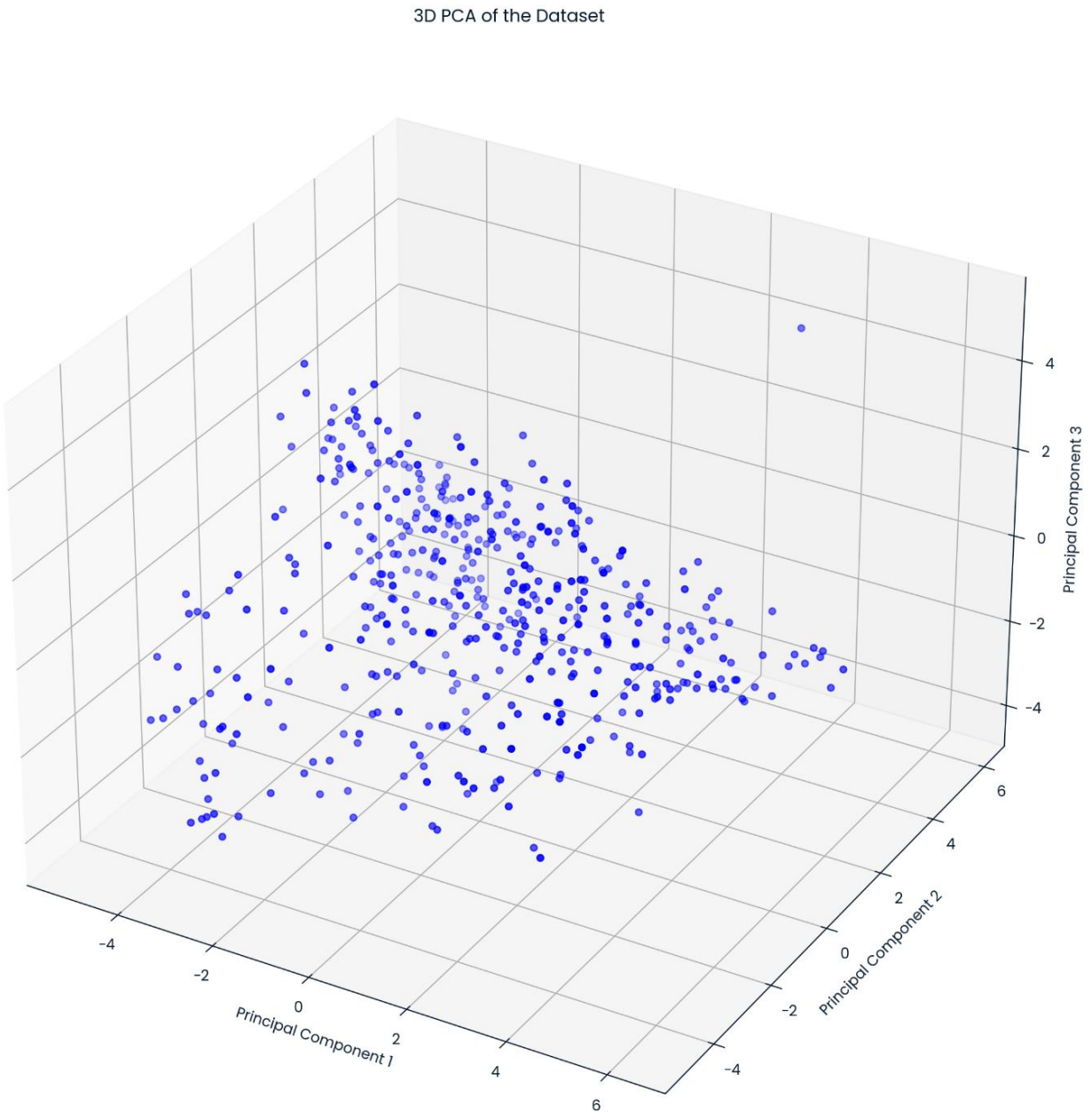
Python code for Scree Plot:

```
pca = PCA(n_components=10)
reduced_data = pca.fit_transform(data) # principal component analysis on the data; fit model and reduce dimensions

plt.plot(np.arange(1, len(pca.explained_variance_ratio_) + 1), pca.explained_variance_ratio_, 'bo-')
plt.xlabel('Principal Component')
plt.ylabel('Explained Variance Ratio')
plt.title('Scree Plot - PCA')
plt.show()
```



Python code for 3D PCA of Dataset

```
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

file_path = 'gvrd-rangestd.csv'
data = pd.read_csv(file_path, header=None)

#Transpose the data to have observations as rows and features as columns
data = data.T


# Assuming 'data' is the dataset you want to reduce
# Perform PCA to reduce the data to 3 dimensions
pca = PCA(n_components=3)
reduced_data = pca.fit_transform(data)

# Create a larger 3D scatter plot
fig = plt.figure(figsize=(14, 14))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(reduced_data[:, 0], reduced_data[:, 1], reduced_data[:, 2], c='blue', marker='o')

ax.set_title('3D PCA of the Dataset')
ax.set_xlabel('Principal Component 1')
ax.set_ylabel('Principal Component 2')
ax.set_zlabel('Principal Component 3')

plt.show()
```


3D PCA of the Dataset
Principal Component 1
Principal Component 2
Principal Component 3
-4
-2
0
2
4
6

Python code for PCA on dataset along with making M and P matrixes:

```
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.metrics import pairwise_distances
# Load the dataset
file_path = 'gvrd-rangestd.csv'
data = pd.read_csv(file_path, header=None)
#Transpose the data to have observations as rows and features as columns
data = data.T
print("Dimensions:", data.shape)
print(data.head())
print(data.describe()) # Details of the data

# Perform k-means clustering on the original data
k = 4  # Because of Sillhoutte Score
kmeans = KMeans(n_clusters=k, random_state=42)
clusters_original = kmeans.fit_predict(data)

# Generate the n x n matrix M
n = data.shape[0]
M = np.zeros((n, n), dtype=int)
for i in range(n):
    for j in range(n):
        if clusters_original[i] == clusters_original[j]:
            M[i, j] = 1

# Perform PCA and keep the first three principal components
pca = PCA(n_components=3)
data_reduced = pca.fit_transform(data)

# Perform k-means clustering on the reduced-dimensional data
kmeans_reduced = KMeans(n_clusters=k, random_state=42)
clusters_reduced = kmeans_reduced.fit_predict(data_reduced)

# Generate the n x n matrix P
P = np.zeros((n, n), dtype=int)
for i in range(n):
    for j in range(n):
        if clusters_reduced[i] == clusters_reduced[j]:
            P[i, j] = 1

# Count the number of entries that are the same between M and P
same_entries = np.sum(M == P)
same_entries
```

Output of the code:

```
Dimensions: (459, 2207)
       0         1     2         3     ...      2203      2204      2205      2206
0  0.048734  0.156177   0.0  0.504464  ...  0.068796  0.068681  0.101562  0.082707
1  0.111323  0.074072   0.0  0.571429  ...  0.194103  0.197802  0.117188  0.150376
2  0.064023  0.080148   0.0  0.473214  ...  0.181818  0.162088  0.195312  0.132832
3  0.058767  0.131366   0.0  0.410714  ...  0.031941  0.024725  0.109375  0.162907
4  0.050645  0.147004   0.0  0.446429  ...  0.140049  0.131868  0.148438  0.170426

[5 rows x 2207 columns]
             0           1      2     ...        2204        2205        2206
count  459.000000  459.000000  459.0  ...  459.000000  459.000000  459.000000
mean     0.080778    0.120589    0.0  ...    0.251107    0.204572    0.196368
std      0.077852    0.125591    0.0  ...    0.159509    0.157142    0.120650
min      0.000000    0.000000    0.0  ...    0.000000    0.000000    0.000000
25%      0.049212    0.050928    0.0  ...    0.127747    0.101562    0.110276
50%      0.061634    0.086378    0.0  ...    0.225275    0.156250    0.177945
75%      0.084090    0.145699    0.0  ...    0.329670    0.242188    0.250627
max      1.000000    1.000000    0.0  ...    1.000000    1.000000    1.000000

[8 rows x 2207 columns]
```

```
194341
```

Python code for generating M1, M2, … M10:

```
[14]
from sklearn.cluster import KMeans
import numpy as np
file_path = 'gvrd-rangestd.csv'
data = pd.read_csv(file_path, header=None)

#Transpose the data to have observations as rows and features as columns
data = data.T
# Number of clusters
k = 4  # Adjust k as needed

# Number of times to run k-means with different initial centroids
n_init = 10

# Perform k-means clustering
kmeans_models = []
for i in range(n_init):
    kmeans = KMeans(n_clusters=k, n_init=1, init='random', random_state=i)
    kmeans.fit(data)
    kmeans_models.append(kmeans)

# Extract the cluster labels from each run
cluster_labels_list = [model.labels_ for model in kmeans_models]

# Generate matrices M1,...,M10 indicating whether given pairs of observations are in the same cluster
n_samples = data.shape[0]
M_matrices = []

for idx, labels in enumerate(cluster_labels_list):
    M = np.zeros((n_samples, n_samples), dtype=int)
    for i in range(n_samples):
        for j in range(n_samples):
            if labels[i] == labels[j]:
                M[i, j] = 1
    globals()[f'M{idx+1}'] = M
    M_matrices.append(M)
# M1, M2, ..., M10 matrices are now available as separate variables
M1, M2, M3, M4, M5, M6, M7, M8, M9, M10
```

Python code for calculating the shared entries in M1, M2, … M10:

```
[15]
import numpy as np

# Assuming M and M1, M2, ..., M10 are defined in the previous cells
M_list = [M1, M2, M3, M4, M5, M6, M7, M8, M9, M10]

for i in range(10):
    shared_entries = np.sum(M_list[i] == M)
    total_elements = M.shape[0] * M.shape[1]
    probability_shared = shared_entries / total_elements
    percentage_shared = probability_shared * 100

    print(f"The number of entries that M{i+1} and M share together are: {shared_entries}")
    print(f"The total elements that M{i+1} and M each has: {total_elements}")
    print(f"The probability of elements that are shared among in M{i+1} and M is: {shared_entries}/{total_elements}=
{probability_shared}")
    print(f"The percentage of elements that are shared among in M{i+1} and M is: {percentage_shared}%")
    print()
    print()
```

Python code for out of all of the pairs of observations that were recorded to be in the same cluster in at least one run, the percentage of the ones in the same cluster in all ten:

```
[16]
# Assuming M and M1, M2, ..., M10 are defined in the previous cells
M_list = [M1, M2, M3, M4, M5, M6, M7, M8, M9, M10]

# Initialize a matrix to count the number of times each pair is in the same cluster
pair_count = np.zeros(M.shape)

# Count the number of times each pair is in the same cluster across all runs
for M_i in M_list:
    pair_count += (M_i == M)

# Find pairs that were in the same cluster in at least one run
at_least_one = np.sum(pair_count > 0)

# Find pairs that were in the same cluster in all ten runs
all_ten = np.sum(pair_count == 10)

# Calculate the percentage
percentage_all_ten = (all_ten / at_least_one) * 100

percentage_all_ten
```

```
58.55630075801804
```

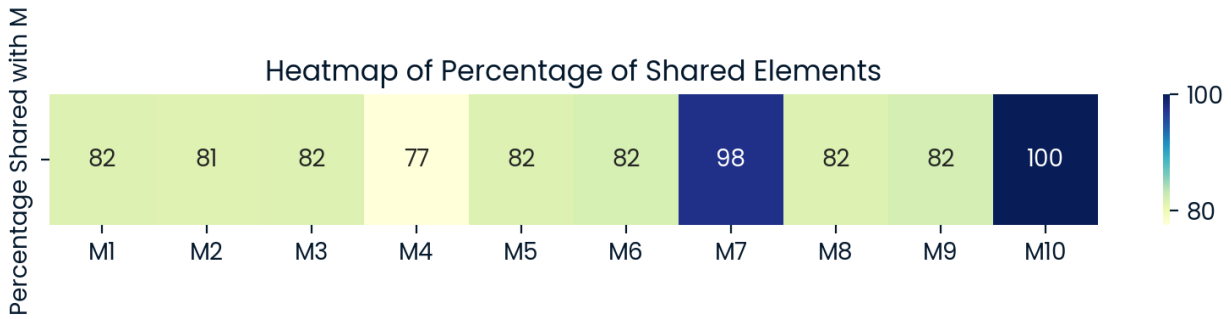
Python for checking robustness:

```
[17]
import numpy as np

# Calculate the pairwise similarity between the M matrices
def calculate_similarity(M1, M2):
    return np.sum(M1 == M2) / (M1.shape[0] * M1.shape[1])

# Calculate the average similarity across all pairs of M matrices
similarities = []
for i in range(len(M_matrices)):
    for j in range(i + 1, len(M_matrices)):
        similarity = calculate_similarity(M_matrices[i], M_matrices[j])
        similarities.append(similarity)

average_similarity = np.mean(similarities)
print(f"Average similarity between different k-means runs: {average_similarity:.4f}")

# Determine robustness based on average similarity
threshold = 0.9  # Define a threshold for robustness
if average_similarity >= threshold:
    print("The results of k-means on this dataset are robust to the initial cluster centroid values chosen.")
else:
    print("The results of k-means on this dataset are not robust to the initial cluster centroid values chosen.")
```

```
Average similarity between different k-means runs: 0.8470
The results of k-means on this dataset are not robust to the initial cluster centroid values chosen.
```

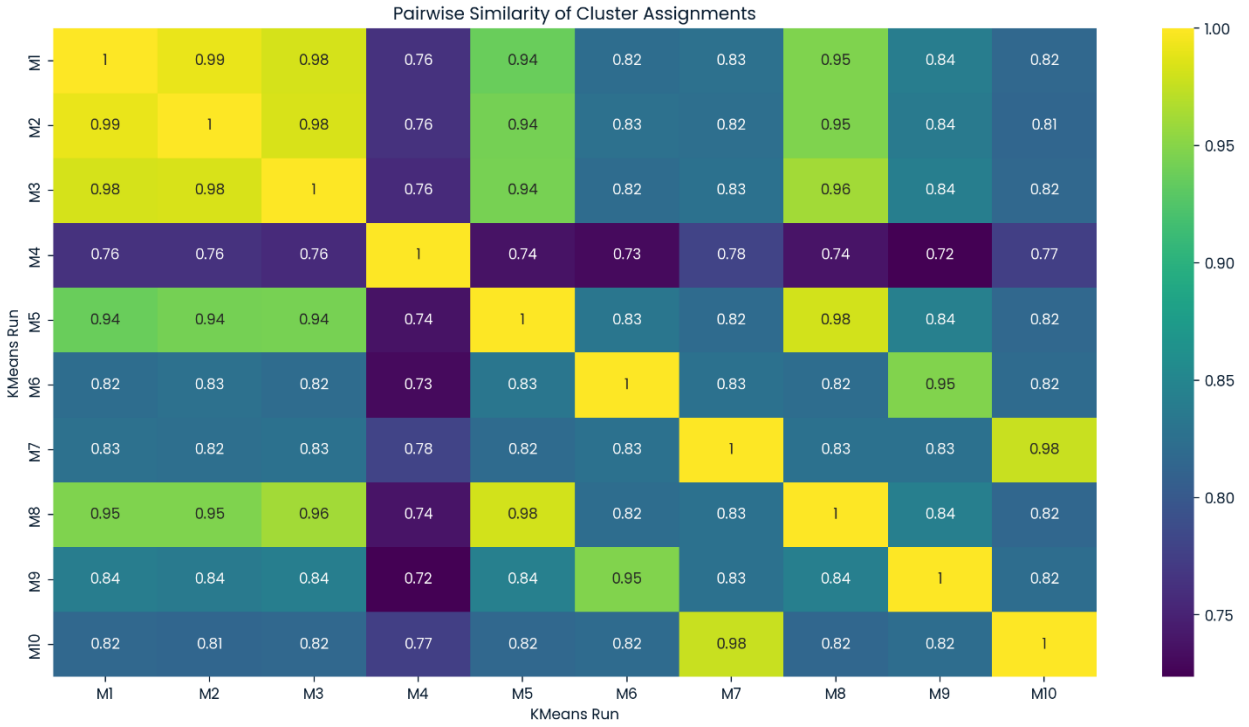
Python code for heatmap comparing M with M1, M2 … M10:

```
[17]
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming M and M1, M2, ..., M10 are defined in the previous cells
M_list = [M1, M2, M3, M4, M5, M6, M7, M8, M9, M10]

# Initialize a list to store the percentage of shared elements
percentage_shared_list = []

for i in range(10):
    shared_entries = np.sum(M_list[i] == M)
    total_elements = M.shape[0] * M.shape[1]
    probability_shared = shared_entries / total_elements
    percentage_shared = probability_shared * 100
    percentage_shared_list.append(percentage_shared)

# Create a heatmap
plt.figure(figsize=(10, 1))
sns.heatmap([percentage_shared_list], annot=True, cmap="YlGnBu", cbar=True, xticklabels=[f'M{i+1}' for i in range(10)],
yticklabels=['Percentage Shared with M'], fmt='.0f')
plt.title('Heatmap of Percentage of Shared Elements')
plt.show()
```



Python code for heatmap comparing M1, M2, … M10 with each other:

```
[18]
import matplotlib.pyplot as plt
import seaborn as sns

# Calculate the pairwise similarity between the cluster label matrices
similarity_matrix = np.zeros((n_init, n_init))

for i in range(n_init):
    for j in range(n_init):
        similarity_matrix[i, j] = np.sum(globals()[f'M{i+1}'] == globals()[f'M{j+1}']) / (n_samples * n_samples)

# Plot the similarity matrix
plt.figure(figsize=(16, 8))
sns.heatmap(similarity_matrix, annot=True, cmap='viridis', cbar=True, xticklabels=[f'M{i+1}' for i in range(n_init)],
yticklabels=[f'M{i+1}' for i in range(n_init)])
plt.title('Pairwise Similarity of Cluster Assignments')
plt.xlabel('KMeans Run')
plt.ylabel('KMeans Run')
plt.show()

# Calculate the average similarity
average_similarity = np.mean(similarity_matrix)
average_similarity
```


Pairwise Similarity of Cluster Assignments
KMeans Run
M1 1 0.99 0.98 0.76 0.94 0.82 0.83 0.95 0.84 0.82
M2 0.99 1 0.98 0.76 0.94 0.83 0.82 0.95 0.84 0.81
M3 0.98 0.98 1 0.76 0.94 0.82 0.83 0.96 0.84 0.82
M4 0.76 0.76 0.76 1 0.74 0.73 0.78 0.74 0.72 0.77
M5 0.94 0.94 0.94 0.74 1 0.83 0.82 0.98 0.84 0.82
M6 0.82 0.83 0.82 0.73 0.83 1 0.83 0.82 0.95 0.82
M7 0.83 0.82 0.83 0.78 0.82 0.83 1 0.83 0.83 0.98
M8 0.95 0.95 0.96 0.74 0.98 0.82 0.83 1 0.84 0.82
M9 0.84 0.84 0.84 0.72 0.84 0.95 0.83 0.84 1 0.82
M10 0.82 0.81 0.82 0.77 0.82 0.82 0.98 0.82 0.82 1
M1 M2 M3 M4 M5 M6 M7 M8 M9 M10
KMeans Run
1.00
0.95
0.90
0.85
0.80
0.75

Python code for generating S matrix:

```
[23]
from scipy.cluster.hierarchy import linkage, fcluster
import pandas as pd
import numpy as np
from sklearn.decomposition import PCA

file_path = 'gvrd-rangestd.csv'
data = pd.read_csv(file_path, header=None)


#Transpose the data to have observations as rows and features as columns
data = data.T

pca = PCA(n_components=3)
data_reduced = pca.fit_transform(data)

# Perform single linkage clustering
Z = linkage(data_reduced, method='single')
k = 4  # Define the number of clusters
clusters = fcluster(Z, k, criterion='maxclust')

# Create the S matrix
n_observations = data_reduced.shape[0]
S_matrix = np.zeros((n_observations, n_observations), dtype=int)

for i in range(n_observations):
    for j in range(n_observations):
        if clusters[i] == clusters[j]:
            S_matrix[i, j] = 1

S_matrix
```

```
array([[1, 1, 1, ..., 1, 1, 1],
       [1, 1, 1, ..., 1, 1, 1],
       [1, 1, 1, ..., 1, 1, 1],
       ...,
       [1, 1, 1, ..., 1, 1, 1],
       [1, 1, 1, ..., 1, 1, 1],
       [1, 1, 1, ..., 1, 1, 1]])
```

Python code for checking common entries between P and S:

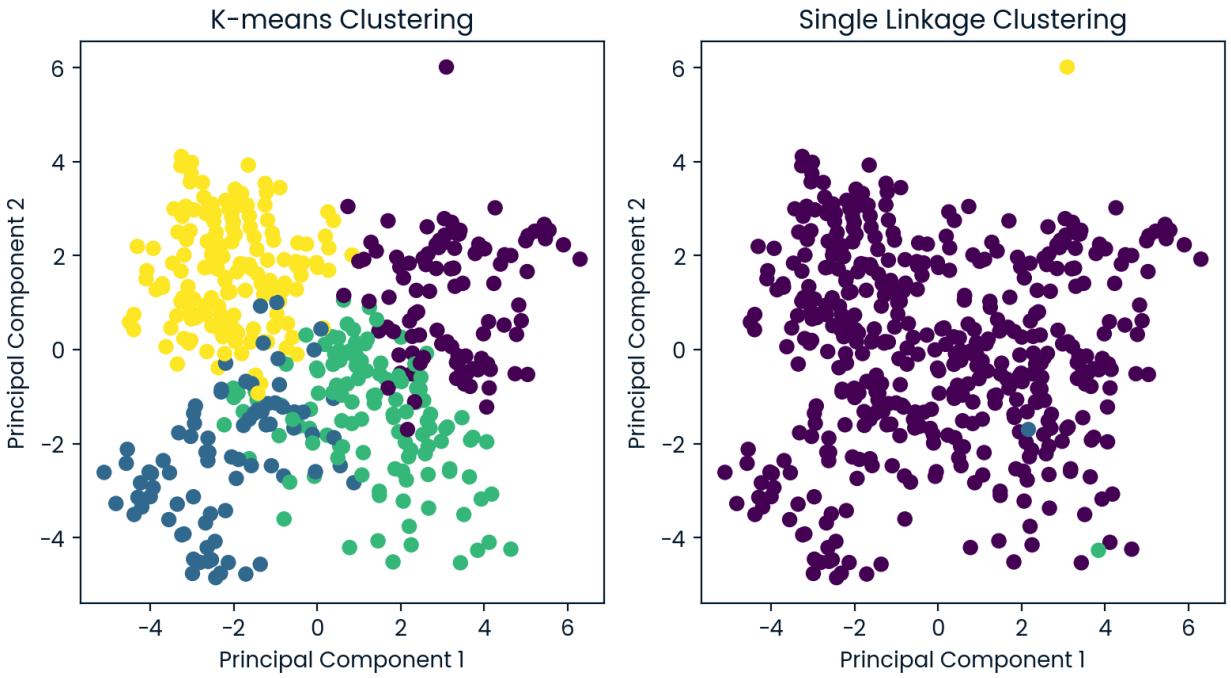
```
[22]
# Determine the number of entries S_matrix has in common with P
common_entries = np.sum(S_matrix == P)
common_entries
print(common_entries)

# Calculate the percentage of elements that are shared between S_matrix and P
total_elements = S_matrix.size
percentage_shared_between_S_and_P = (common_entries / total_elements) * 100
percentage_shared_between_S_and_P
```

```
57147
```

```
27.12489498341094
```

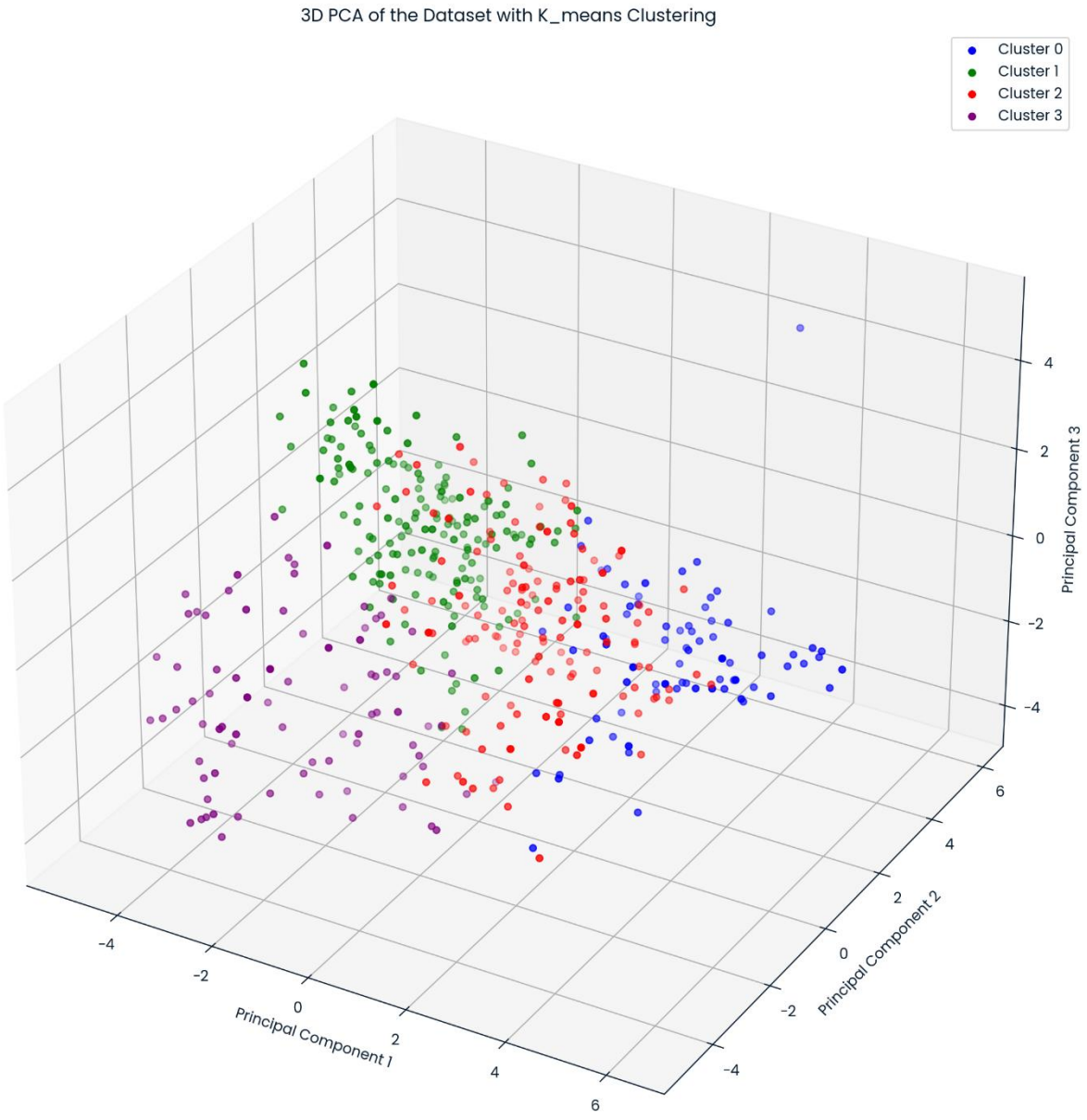
Python code for 2D graphs comparing K-means and Single-linkage clustering on PCA reduced data:

```
import matplotlib.pyplot as plt                                                    [25]
from sklearn.cluster import KMeans

# Perform k-means clustering
kmeans = KMeans(n_clusters=k, random_state=0).fit(data_reduced)
kmeans_clusters = kmeans.labels_

# Create scatter plot for k-means clustering
plt.figure(figsize=(8, 4.5))

plt.subplot(1, 2, 1)
plt.scatter(data_reduced[:, 0], data_reduced[:, 1], c=kmeans_clusters, cmap='viridis', marker='o')
plt.title('K-means Clustering')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')

# Create scatter plot for single linkage clustering
plt.subplot(1, 2, 2)
plt.scatter(data_reduced[:, 0], data_reduced[:, 1], c=clusters, cmap='viridis', marker='o')
plt.title('Single Linkage Clustering')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')

plt.tight_layout()
plt.show()
```



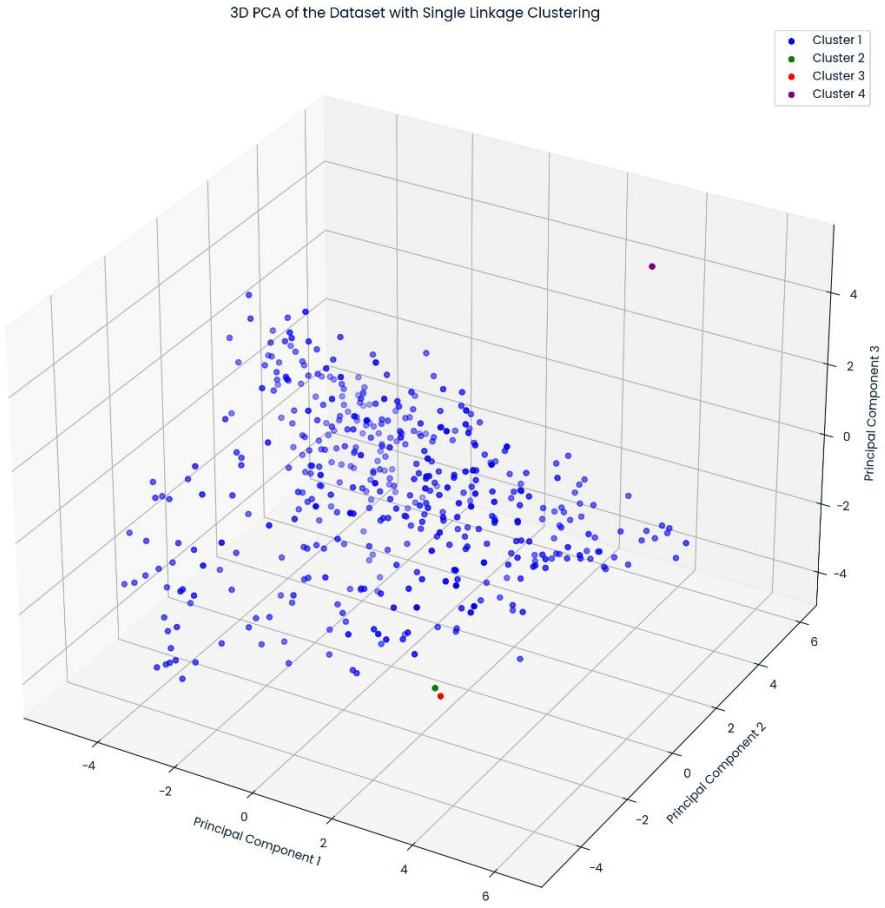
Python code for visualizing K-means clustering on PCA-reduced data:

```
# Visualize the clustering in 3D using the reduced data from PCA
fig = plt.figure(figsize=(14, 14))
ax = fig.add_subplot(111, projection='3d')

# Use different colors for each cluster
colors = ['blue', 'green', 'red', 'purple']
for cluster in range(k):
    cluster_points = reduced_data[clusters == cluster]
    ax.scatter(cluster_points[:, 0], cluster_points[:, 1], cluster_points[:, 2],
               c=colors[cluster], label=f'Cluster {cluster}', marker='o')

ax.set_title('3D PCA of the Dataset with K_means Clustering')
ax.set_xlabel('Principal Component 1')
ax.set_ylabel('Principal Component 2')
ax.set_zlabel('Principal Component 3')
ax.legend()

plt.show()
```


3D PCA of the Dataset with K_means Clustering
Cluster 0
Cluster 1
Cluster 2
Cluster 3
Principal Component 1
Principal Component 2
Principal Component 3
-4
-2
0
2
4
6

Python code for visualizing single-linkage clustering on PCA-reduced data:

```
[9]
from scipy.cluster.hierarchy import linkage, fcluster
from mpl_toolkits.mplot3d import Axes3D

# Perform single linkage clustering
Z = linkage(reduced_data, method='single')
clusters = fcluster(Z, t=k, criterion='maxclust')

# Visualize the clustering in 3D using the reduced data from PCA
fig = plt.figure(figsize=(14, 14))
ax = fig.add_subplot(111, projection='3d')

# Use different colors for each cluster
colors = ['blue', 'green', 'red', 'purple']
for cluster in range(1, k+1):
    cluster_points = reduced_data[clusters == cluster]
    ax.scatter(cluster_points[:, 0], cluster_points[:, 1], cluster_points[:, 2],
               c=colors[cluster-1], label=f'Cluster {cluster}', marker='o')

ax.set_title('3D PCA of the Dataset with Single Linkage Clustering')
ax.set_xlabel('Principal Component 1')
ax.set_ylabel('Principal Component 2')
ax.set_zlabel('Principal Component 3')
ax.legend()

plt.show()
```


3D PCA of the Dataset with Single Linkage Clustering
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Principal Component 1
Principal Component 2
Principal Component 3

Python code for generating E matrix:

```
[26]
import numpy as np
from sklearn.mixture import GaussianMixture
import pandas as pd
from sklearn.decomposition import PCA

file_path = 'gvrd-rangestd.csv'
data = pd.read_csv(file_path, header=None)

pca = PCA(n_components=3)
reduced_data = pca.fit_transform(data)

# Define the number of clusters
k = 4

# Fit the Gaussian Mixture Model
gmm = GaussianMixture(n_components=k, random_state=42)
gmm.fit(reduced_data)

# Get the probabilities of each point belonging to each cluster
probabilities = gmm.predict_proba(reduced_data)

# Create the matrix E
n_samples = reduced_data.shape[0]
E = np.zeros((n_samples, n_samples))

for i in range(n_samples):
    for j in range(n_samples):
        E[i, j] = np.dot(probabilities[i], probabilities[j])

print(E)
print(E.shape)
# E is now the matrix where E[i, j] is the probability that observations i and j are in the same cluster
```

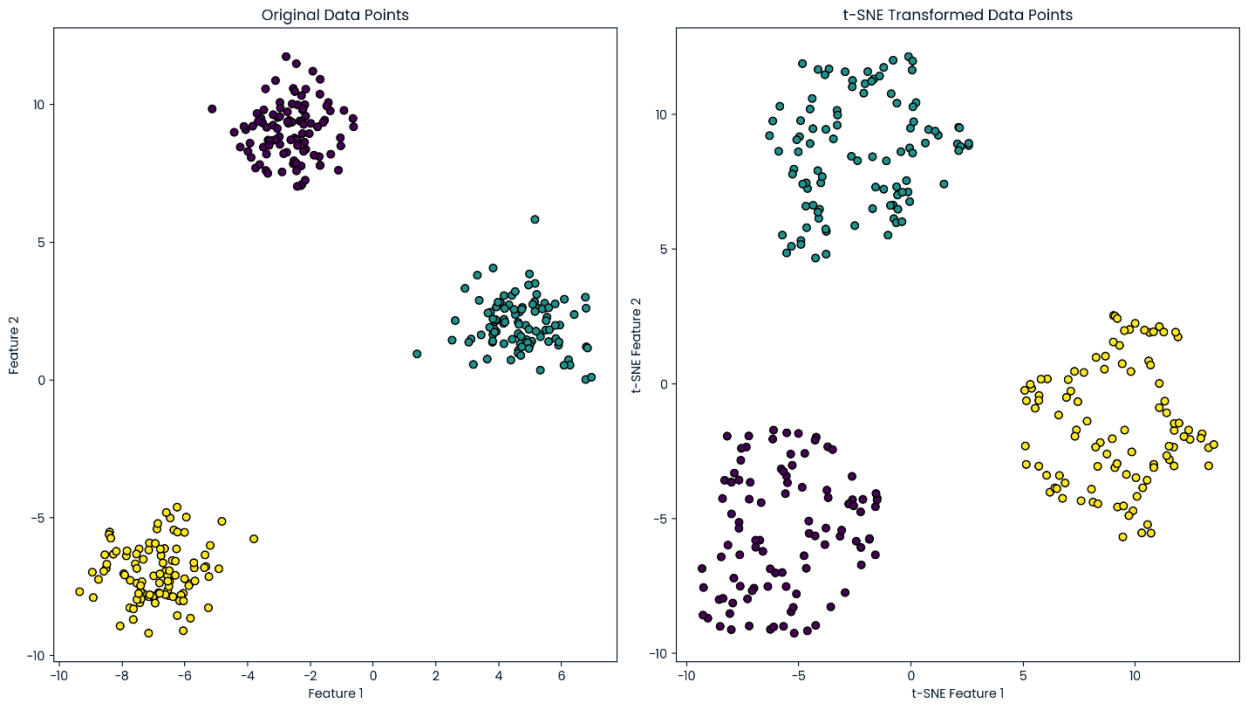
Python code to answer What percentage of observations have at least two different clusters that they are assigned to with probability >1%?

```
[27]
# Calculate the percentage of observations that have at least two different clusters
# with a probability greater than 1%

# Threshold for the probability
threshold = 0.01

# Count the number of observations that meet the criteria
count = 0
for prob in probabilities:
    if np.sum(prob > threshold) >= 2:
        count += 1

# Calculate the percentage
percentage = (count / n_samples) * 100

percentage
```

```
25.509741730856366
```

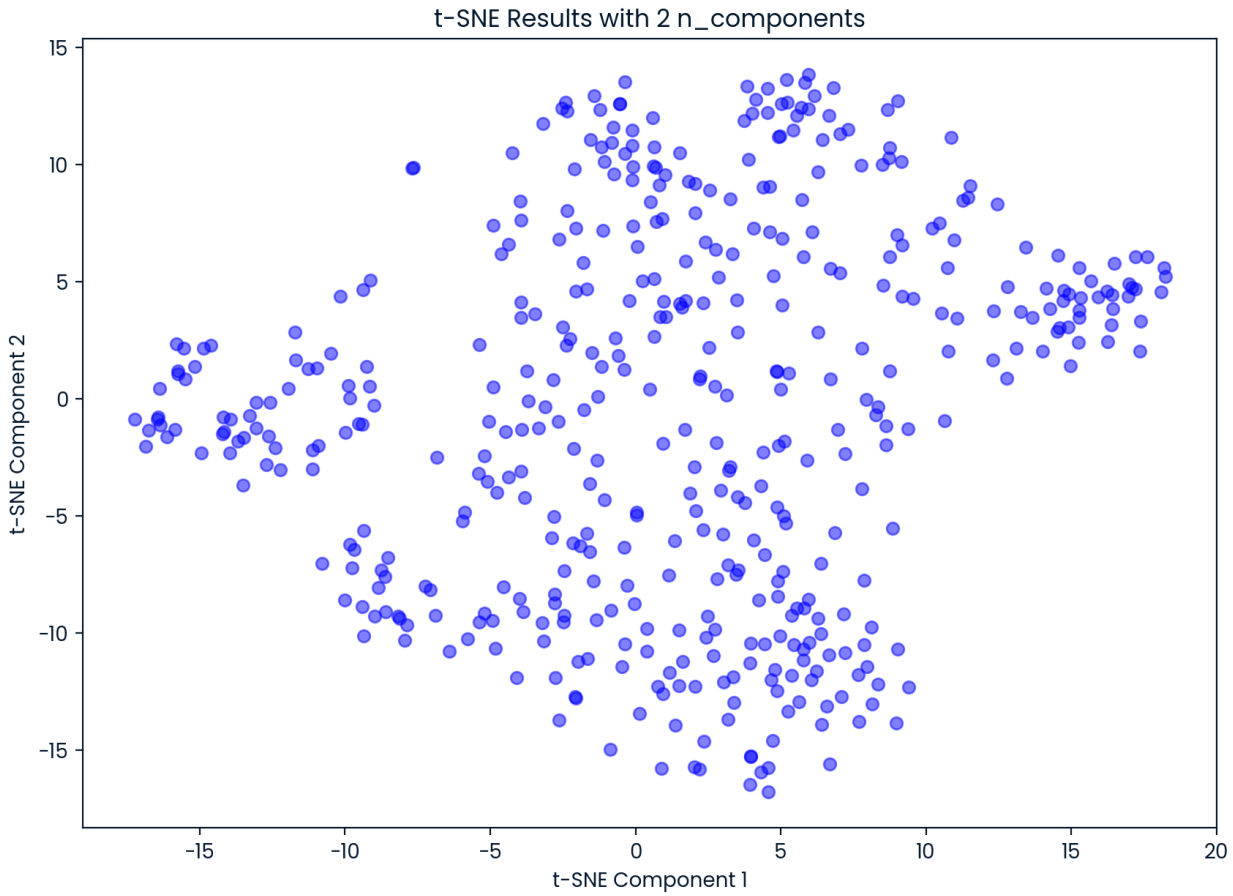
Python code for graphs visualizing t-SNE on dummy dataset:

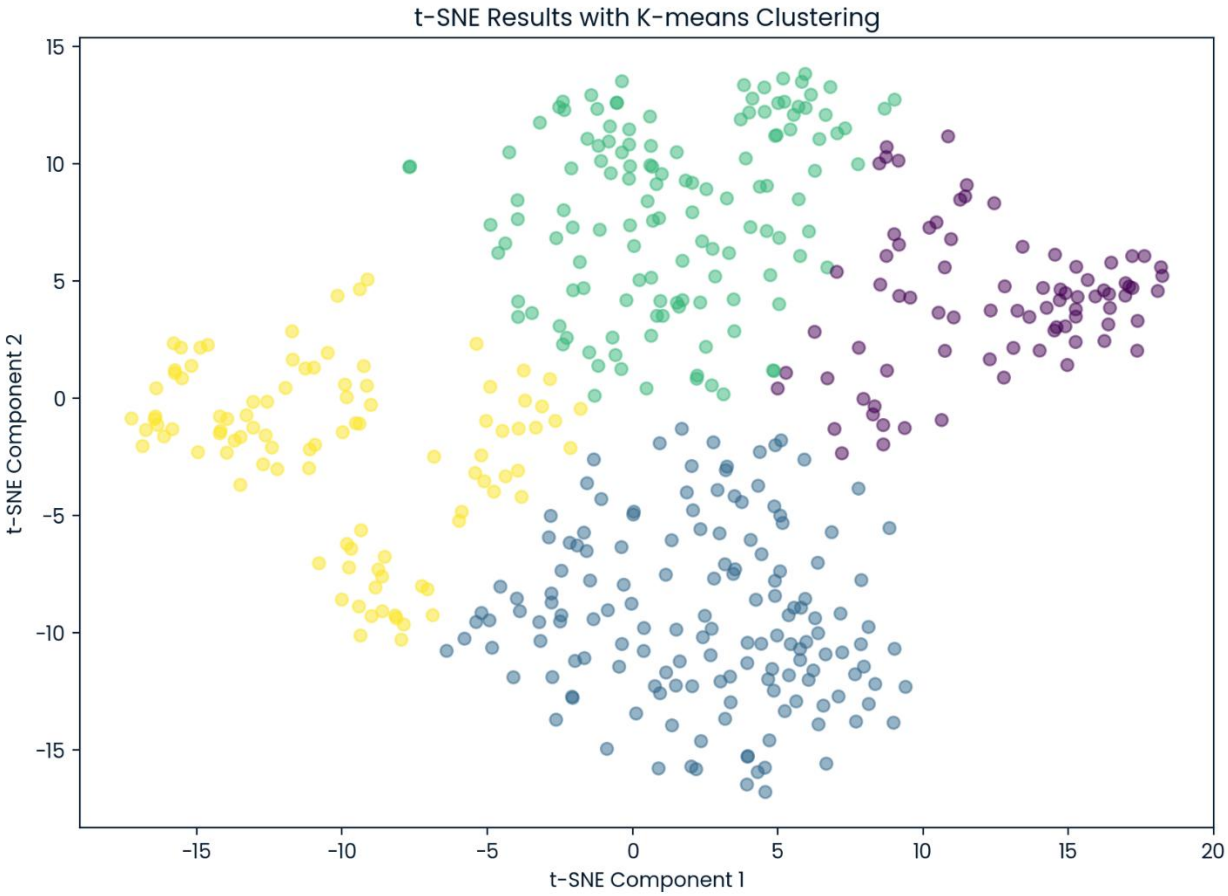
```python
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.manifold import TSNE

# Generate a synthetic dataset for demonstration
X, y = make_blobs(n_samples=300, centers=3, cluster_std=1.00, random_state=42)

# Perform t-SNE
tsne = TSNE(n_components=2, perplexity=30, n_iter=300)
X_tsne = tsne.fit_transform(X)

# Plot the original data points
plt.figure(figsize=(14, 8))

# Plot the original high-dimensional data points
plt.subplot(1, 2, 1)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis', edgecolor='k', marker='o')
plt.title('Original Data Points')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')

# Plot the t-SNE transformed data points
plt.subplot(1, 2, 2)
plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=y, cmap='viridis', edgecolor='k', marker='o')
plt.title('t-SNE Transformed Data Points')
plt.xlabel('t-SNE Feature 1')
plt.ylabel('t-SNE Feature 2')

plt.tight_layout()
plt.show()
```



Python code for reducing the data using t-SNE with 2 components and doing K-means clustering:

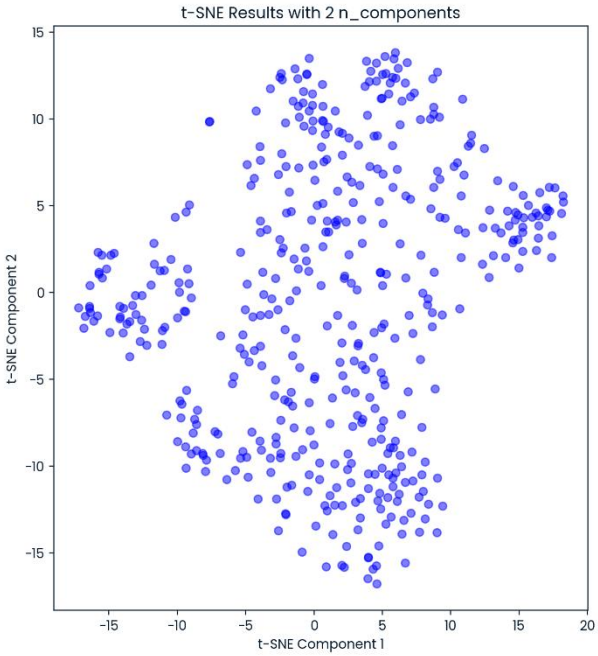
```python
# Import necessary libraries
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE

file_path = 'gvrd-rangestd.csv'
data = pd.read_csv(file_path, header=None)

#Transpose the data to have observations as rows and features as columns
data = data.T

# Run t-SNE on the full-dimensional dataset with new tuning parameters
tsne = TSNE(n_components=2, perplexity=40, n_iter=1500, learning_rate=200, random_state=0)
tsne_results = tsne.fit_transform(data)

# Plot the results
plt.figure(figsize=(10, 7))
plt.scatter(tsne_results[:, 0], tsne_results[:, 1], c='blue', alpha=0.5)
plt.title('t-SNE Results with 2 n_components')
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
plt.show()

from sklearn.cluster import KMeans

# Perform K-means clustering on the t-SNE results
kmeans = KMeans(n_clusters=4, random_state=0)
kmeans_labels = kmeans.fit_predict(tsne_results)

# Plot the clustered results
plt.figure(figsize=(10, 7))
plt.scatter(tsne_results[:, 0], tsne_results[:, 1], c=kmeans_labels, cmap='viridis', alpha=0.5)
plt.title('t-SNE Results with K-means Clustering')
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
plt.show()
```

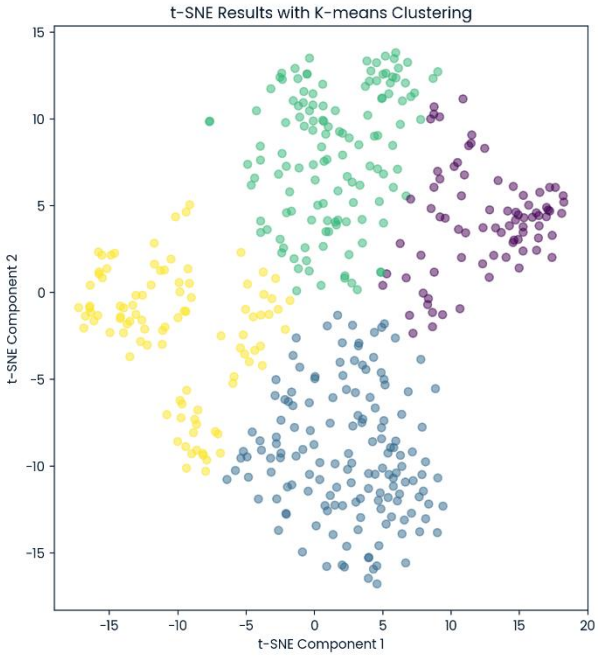




Python code for reducing the data using t-SNE with 2 components and K-means clustering:

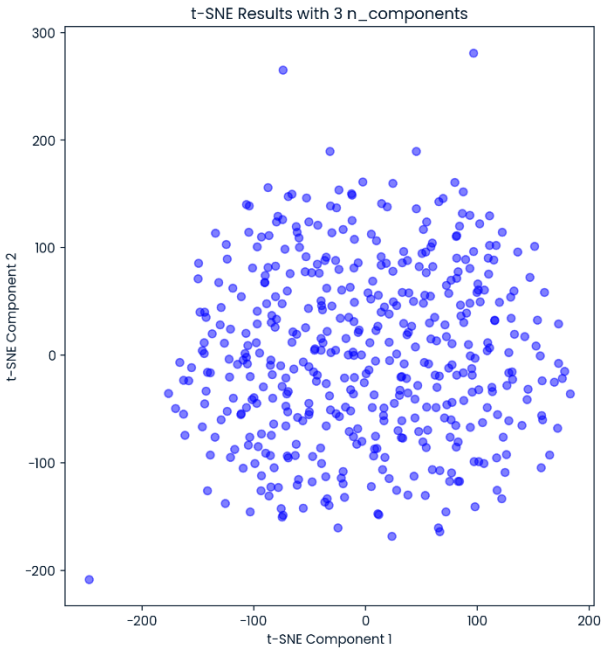
```python
# Import necessary libraries
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE
import pandas as pd

file_path = 'gvrd-rangestd.csv'
data = pd.read_csv(file_path, header=None)

# Transpose the data to have observations as rows and features as columns
data = data.T

# Run t-SNE on the full-dimensional dataset with new tuning parameters
tsne = TSNE(n_components=2, perplexity=40, n_iter=1500, learning_rate=200, random_state=0)
tsne_results = tsne.fit_transform(data)

from sklearn.cluster import KMeans

# Perform K-means clustering on the t-SNE results
kmeans = KMeans(n_clusters=4, random_state=0)
kmeans_labels = kmeans.fit_predict(tsne_results)

# Plot the results
fig, ax = plt.subplots(1, 2, figsize=(16, 8))

# Plot t-SNE results
ax[0].scatter(tsne_results[:, 0], tsne_results[:, 1], c='blue', alpha=0.5)
ax[0].set_title('t-SNE Results with 2 n_components')
ax[0].set_xlabel('t-SNE Component 1')
ax[0].set_ylabel('t-SNE Component 2')

# Plot t-SNE results with K-means clustering
ax[1].scatter(tsne_results[:, 0], tsne_results[:, 1], c=kmeans_labels, cmap='viridis', alpha=0.5)
ax[1].set_title('t-SNE Results with K-means Clustering')
ax[1].set_xlabel('t-SNE Component 1')
ax[1].set_ylabel('t-SNE Component 2')

plt.show()
```

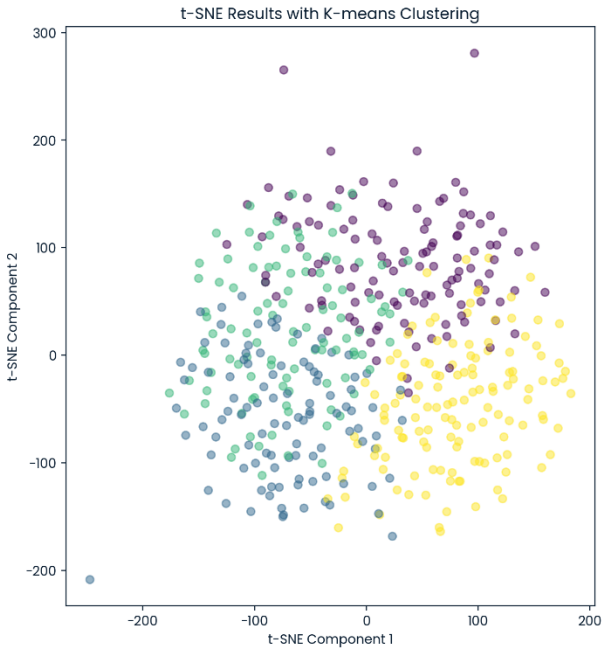




Python code for reducing the data using t-SNE with 3 components and K-means clustering:

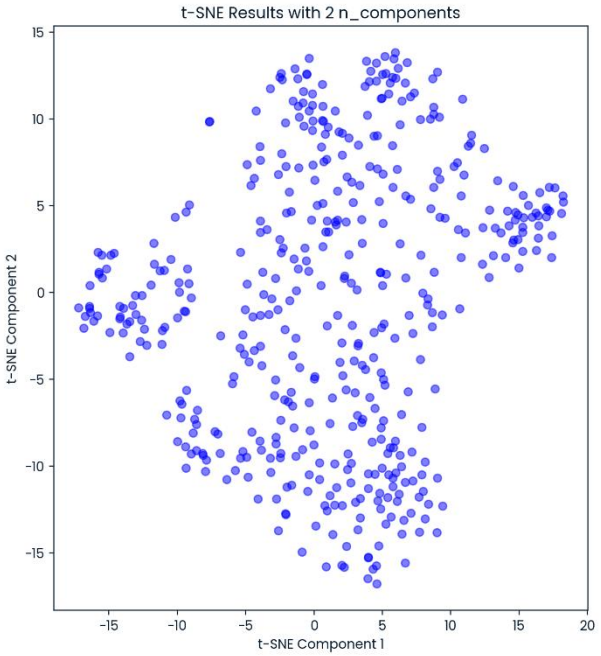
```
# Import necessary libraries
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE
import pandas as pd

file_path = 'gvrd-rangestd.csv'
data = pd.read_csv(file_path, header=None)

# Transpose the data to have observations as rows and features as columns
data = data.T

# Run t-SNE on the full-dimensional dataset with new tuning parameters
tsne = TSNE(n_components=3, perplexity=40, n_iter=1500, learning_rate=200, random_state=0)
tsne_results = tsne.fit_transform(data)

from sklearn.cluster import KMeans

# Perform K-means clustering on the t-SNE results
kmeans = KMeans(n_clusters=4, random_state=0)
kmeans_labels = kmeans.fit_predict(tsne_results)

# Plot the results
fig, ax = plt.subplots(1, 2, figsize=(16, 8))

# Plot t-SNE results
ax[0].scatter(tsne_results[:, 0], tsne_results[:, 1], c='blue', alpha=0.5)
ax[0].set_title('t-SNE Results with 3 n_components')
ax[0].set_xlabel('t-SNE Component 1')
ax[0].set_ylabel('t-SNE Component 2')

# Plot t-SNE results with K-means clustering
ax[1].scatter(tsne_results[:, 0], tsne_results[:, 1], c=kmeans_labels, cmap='viridis', alpha=0.5)
ax[1].set_title('t-SNE Results with K-means Clustering')
ax[1].set_xlabel('t-SNE Component 1')
ax[1].set_ylabel('t-SNE Component 2')

plt.show()
```

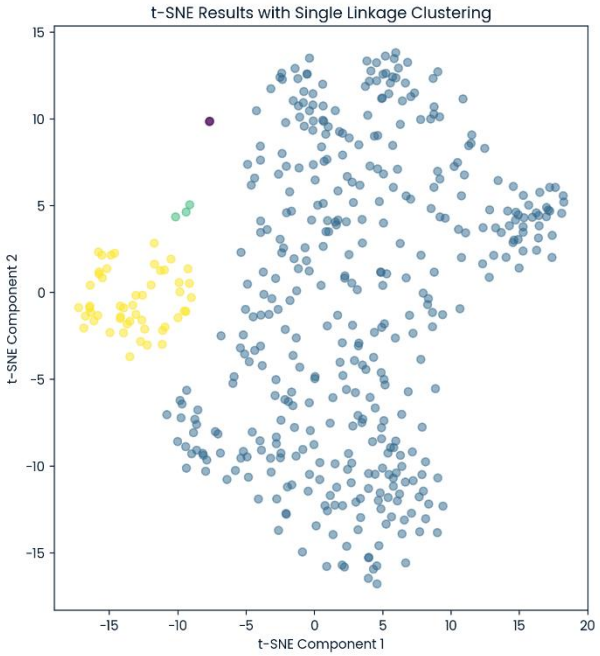




Python code for reducing the data using t-SNE with 2 components and single-linkage clustering:

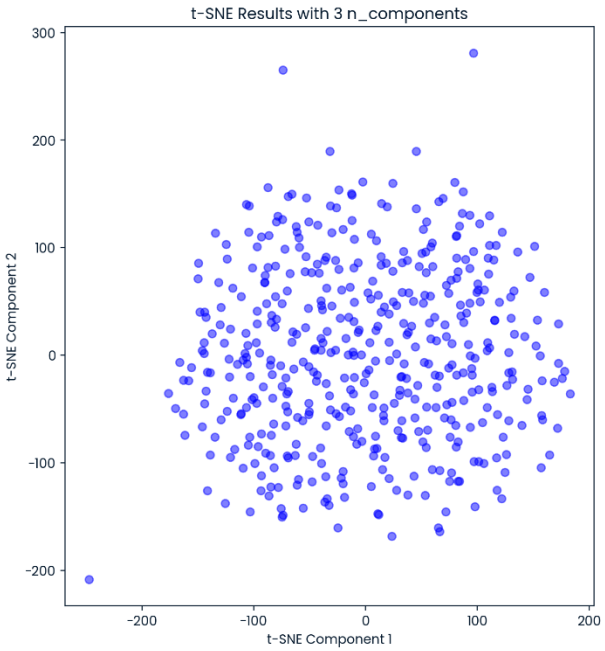
```
[34]
# Import necessary libraries
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE
import pandas as pd
from scipy.cluster.hierarchy import single, fcluster
from scipy.spatial.distance import pdist

file_path = 'gvrd-rangestd.csv'
data = pd.read_csv(file_path, header=None)

# Transpose the data to have observations as rows and features as columns
data = data.T

# Run t-SNE on the full-dimensional dataset with new tuning parameters
tsne = TSNE(n_components=2, perplexity=40, n_iter=1500, learning_rate=200, random_state=0)
tsne_results = tsne.fit_transform(data)

# Perform single linkage clustering on the t-SNE results
distance_matrix = pdist(tsne_results)
linkage_matrix = single(distance_matrix)
single_linkage_labels = fcluster(linkage_matrix, t=4, criterion='maxclust')

# Plot the results
fig, ax = plt.subplots(1, 2, figsize=(16, 8))

# Plot t-SNE results
ax[0].scatter(tsne_results[:, 0], tsne_results[:, 1], c='blue', alpha=0.5)
ax[0].set_title('t-SNE Results with 2 n_components')
ax[0].set_xlabel('t-SNE Component 1')
ax[0].set_ylabel('t-SNE Component 2')

# Plot t-SNE results with single linkage clustering
ax[1].scatter(tsne_results[:, 0], tsne_results[:, 1], c=single_linkage_labels, cmap='viridis', alpha=0.5)
ax[1].set_title('t-SNE Results with Single Linkage Clustering')
ax[1].set_xlabel('t-SNE Component 1')
ax[1].set_ylabel('t-SNE Component 2')

plt.show()
```

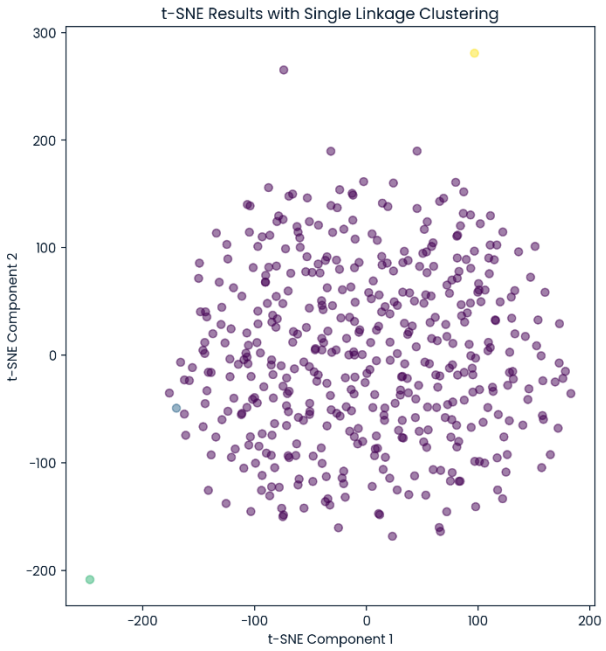




Python code for reducing the data using t-SNE with 3 components and single-linkage clustering:

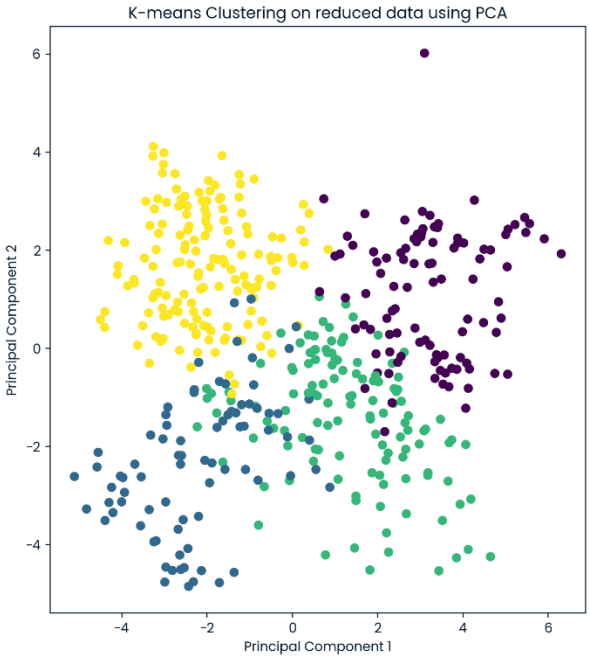
```python
# Import necessary libraries
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE
import pandas as pd
from scipy.cluster.hierarchy import single, fcluster
from scipy.spatial.distance import pdist

file_path = 'gvrd-rangestd.csv'
data = pd.read_csv(file_path, header=None)

# Transpose the data to have observations as rows and features as columns
data = data.T

# Run t-SNE on the full-dimensional dataset with new tuning parameters
tsne = TSNE(n_components=3, perplexity=40, n_iter=1500, learning_rate=200, random_state=0)
tsne_results = tsne.fit_transform(data)

# Perform single linkage clustering on the t-SNE results
distance_matrix = pdist(tsne_results)
linkage_matrix = single(distance_matrix)
single_linkage_labels = fcluster(linkage_matrix, t=4, criterion='maxclust')

# Plot the results
fig, ax = plt.subplots(1, 2, figsize=(16, 8))

# Plot t-SNE results
ax[0].scatter(tsne_results[:, 0], tsne_results[:, 1], c='blue', alpha=0.5)
ax[0].set_title('t-SNE Results with 3 n_components')
ax[0].set_xlabel('t-SNE Component 1')
ax[0].set_ylabel('t-SNE Component 2')

# Plot t-SNE results with single linkage clustering
ax[1].scatter(tsne_results[:, 0], tsne_results[:, 1], c=single_linkage_labels, cmap='viridis', alpha=0.5)
ax[1].set_title('t-SNE Results with Single Linkage Clustering')
ax[1].set_xlabel('t-SNE Component 1')
ax[1].set_ylabel('t-SNE Component 2')

plt.show()
```

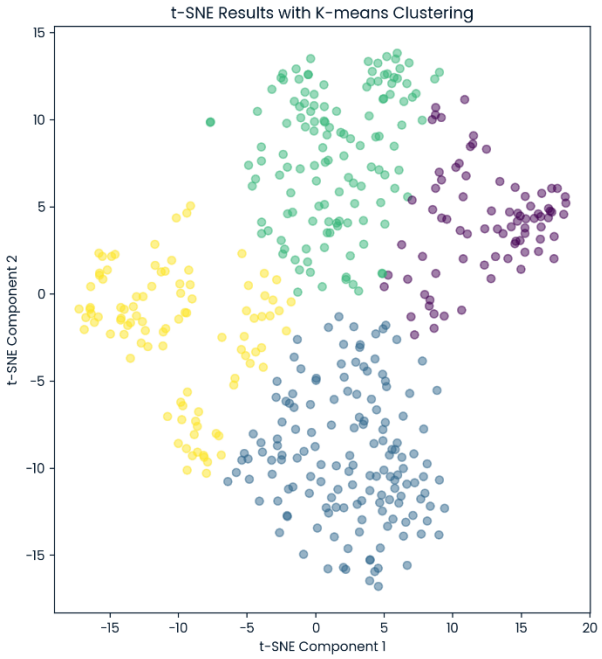




Python code for graphs comparing K-means:

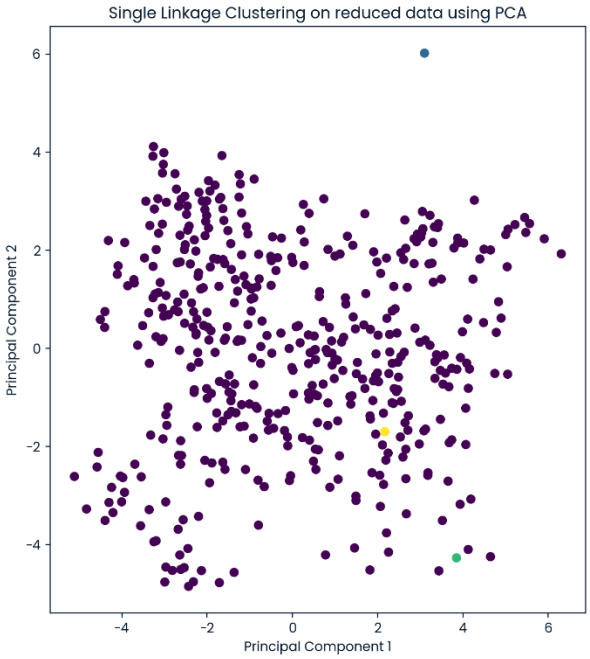
```python
# Import necessary libraries
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE
from sklearn.cluster import KMeans
import pandas as pd

file_path = 'gvrd-rangestd.csv'
data = pd.read_csv(file_path, header=None)

# Transpose the data to have observations as rows and features as columns
data = data.T

# Perform k-means clustering on reduced data (assuming data_reduced is defined elsewhere)
kmeans = KMeans(n_clusters=4, random_state=0).fit(data_reduced)
kmeans_clusters = kmeans.labels_

# Run t-SNE on the full-dimensional dataset with new tuning parameters
tsne = TSNE(n_components=2, perplexity=40, n_iter=1500, learning_rate=200, random_state=0)
tsne_results = tsne.fit_transform(data)

# Perform K-means clustering on the t-SNE results
kmeans_tsne = KMeans(n_clusters=4, random_state=0)
kmeans_tsne_labels = kmeans_tsne.fit_predict(tsne_results)

# Create combined scatter plots
fig, axs = plt.subplots(1, 2, figsize=(16, 8))

# Scatter plot for k-means clustering on reduced data
axs[0].scatter(data_reduced[:, 0], data_reduced[:, 1], c=kmeans_clusters, cmap='viridis', marker='o')
axs[0].set_title('K-means Clustering on reduced data using PCA')
axs[0].set_xlabel('Principal Component 1')
axs[0].set_ylabel('Principal Component 2')

# Scatter plot for t-SNE results with k-means clustering
axs[1].scatter(tsne_results[:, 0], tsne_results[:, 1], c=kmeans_tsne_labels, cmap='viridis', alpha=0.5)
axs[1].set_title('t-SNE Results with K-means Clustering')
axs[1].set_xlabel('t-SNE Component 1')
axs[1].set_ylabel('t-SNE Component 2')

plt.show()
```

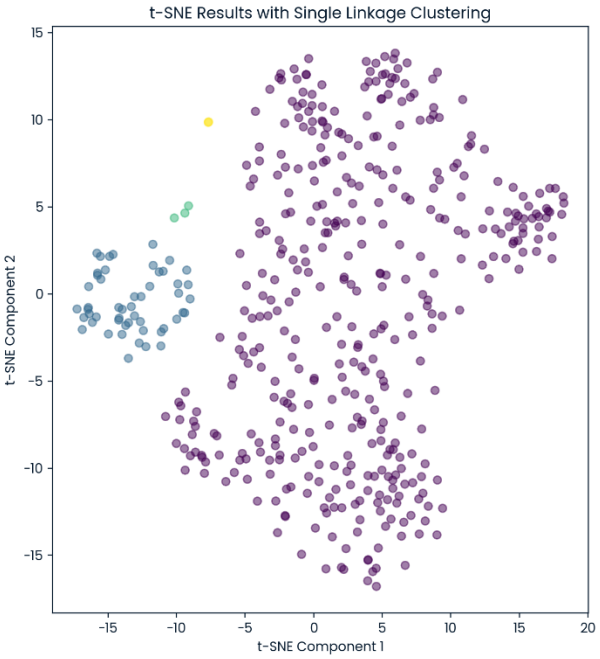




Python code for graphs comparing single-linkage clustering:

```python
# Import necessary libraries
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE
from sklearn.cluster import AgglomerativeClustering
import pandas as pd

file_path = 'gvrd-rangestd.csv'
data = pd.read_csv(file_path, header=None)

# Transpose the data to have observations as rows and features as columns
data = data.T

# Perform single linkage clustering on reduced data (assuming data_reduced is defined elsewhere)
single_linkage = AgglomerativeClustering(n_clusters=4, linkage='single')
single_linkage_clusters = single_linkage.fit_predict(data_reduced)

# Run t-SNE on the full-dimensional dataset with new tuning parameters
tsne = TSNE(n_components=2, perplexity=40, n_iter=1500, learning_rate=200, random_state=0)
tsne_results = tsne.fit_transform(data)

# Perform single linkage clustering on the t-SNE results
single_linkage_tsne = AgglomerativeClustering(n_clusters=4, linkage='single')
single_linkage_tsne_labels = single_linkage_tsne.fit_predict(tsne_results)

# Create combined scatter plots
fig, axs = plt.subplots(1, 2, figsize=(16, 8))

# Scatter plot for single linkage clustering on reduced data
axs[0].scatter(data_reduced[:, 0], data_reduced[:, 1], c=single_linkage_clusters, cmap='viridis', marker='o')
axs[0].set_title('Single Linkage Clustering on reduced data using PCA')
axs[0].set_xlabel('Principal Component 1')
axs[0].set_ylabel('Principal Component 2')

# Scatter plot for t-SNE results with single linkage clustering
axs[1].scatter(tsne_results[:, 0], tsne_results[:, 1], c=single_linkage_tsne_labels, cmap='viridis', alpha=0.5)
axs[1].set_title('t-SNE Results with Single Linkage Clustering')
axs[1].set_xlabel('t-SNE Component 1')
axs[1].set_ylabel('t-SNE Component 2')

plt.show()
```


Single Linkage Clustering on reduced data using PCA
Principal Component 2
Principal Component 1
6
4
2
0
-2
-4
-4
-2
0
2
4
6


t-SNE Results with Single Linkage Clustering
t-SNE Component 2
t-SNE Component 1
15
10
5
0
-5
-10
-15
-15
-10
-5
0
5
10
15
20