Hierarchical Clustering on a Simple Dataset

1. Importing necessary libraries

First, we need to import the libraries required for our analysis: `numpy` for numerical operations, `matplotlib.pyplot` for visualisations, `scipy` for implementation of hierarchical clustering, and `sklearn` for evaluation of clustering performance.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import linkage, dendrogram, fcluster
from sklearn.datasets import make_blobs
from sklearn.metrics import silhouette_score
```

2. Generating a simple dataset

We create a simple dataset using the `make_blobs` function. The dataset consists of samples grouped into clusters. A random state is provided for reproducibility.

```
def generate_dataset(n_samples=100, n_clusters=3, random_state=42):
  X, _ = make_blobs(n_samples=n_samples, centers=n_clusters, random_state=random_state)
  return X
```

3. Implementing hierarchical clustering algorithms

Then, we define a function `hierarchical_clustering` that takes the dataset `X` and a `method` (linkage criterion) as input. It uses the `linkage` function from `scipy.cluster.hierarchy` to perform hierarchical clustering based on the specified method.

```
def hierarchical_clustering(X, method):
  Z = linkage(X, method=method)
  return Z
```

4. Visualising the dendrogram

The `visualize_dendrogram` function takes the linkage matrix `Z` and the `method` used, and generates a dendrogram. The dendrogram provides a visual representation of the hierarchical clustering, showing how clusters are formed at various distances.

```
def visualize_dendrogram(Z, method):
  plt.figure(figsize=(10,6))
  dendrogram(Z)
  plt.title(f'Dendrogram using {method} linkage')
  plt.xlabel('Samples')
  plt.ylabel('Distance')
  plt.show()
```

5. Evaluating clustering performance

Next, we evaluate the performance of each clustering method using the silhouette score. The silhouette score measures how similar an object is to its own cluster compared to other clusters. The higher the score, the better the clustering. We use the `fcluster` function to extract cluster labels for a specified number of clusters.

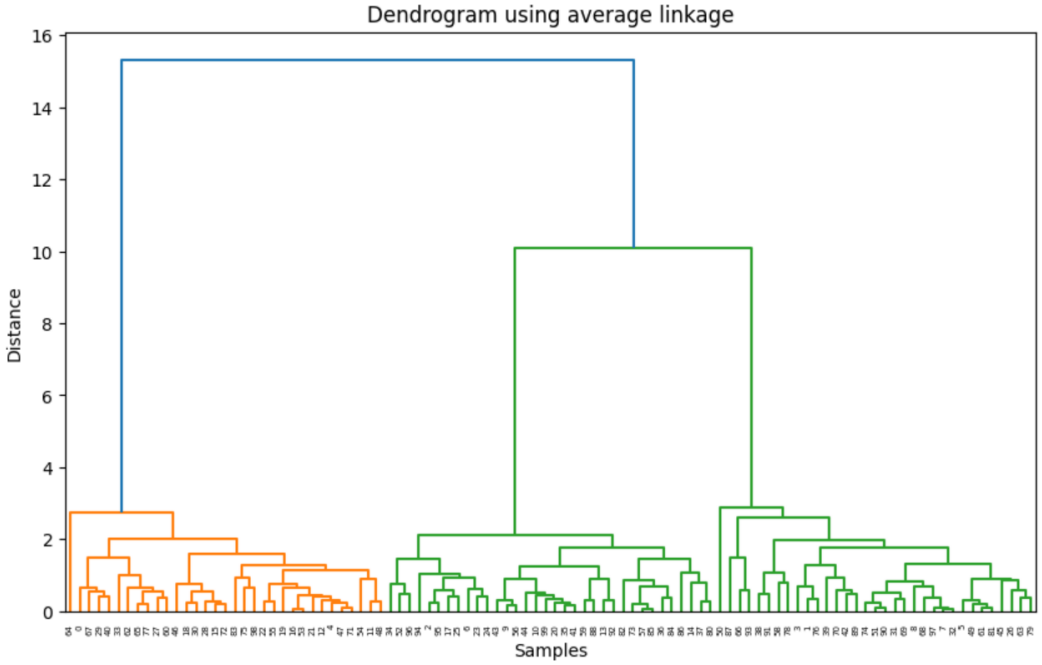
```
def evaluate_clustering(X):
  methods = ['single', 'complete', 'average', 'centroid']
  scores = {}

  for method in methods:
    Z = hierarchical_clustering(X, method)
    clusters = fcluster(Z, t=3, criterion='maxclust')    # Assuming a threshold for cutting the dendrogram
    score = silhouette_score(X, clusters)
    scores[method] = score

  return scores
```

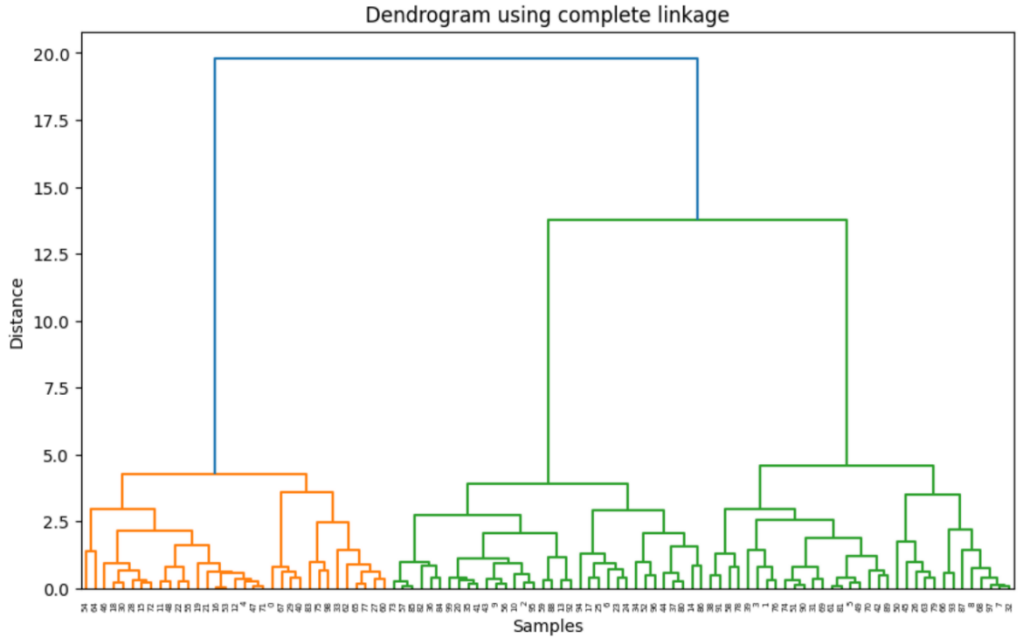
6. Main exeuction

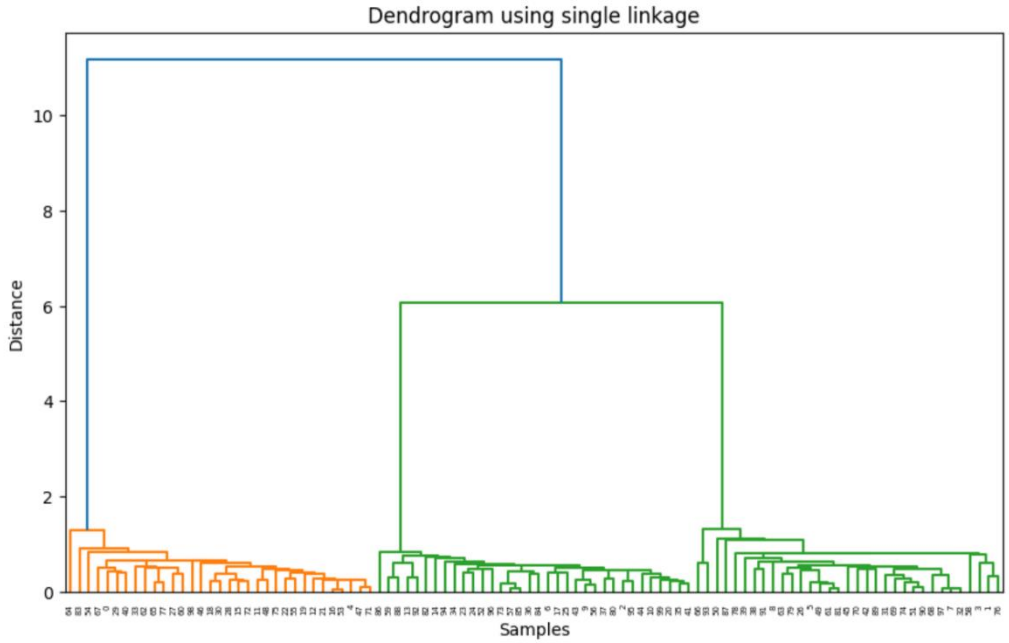
Finally, the main block of the program generates the dataset, visualises the dendrograms for each clustering method, and evaluates their performance by printing the silhouette scores.

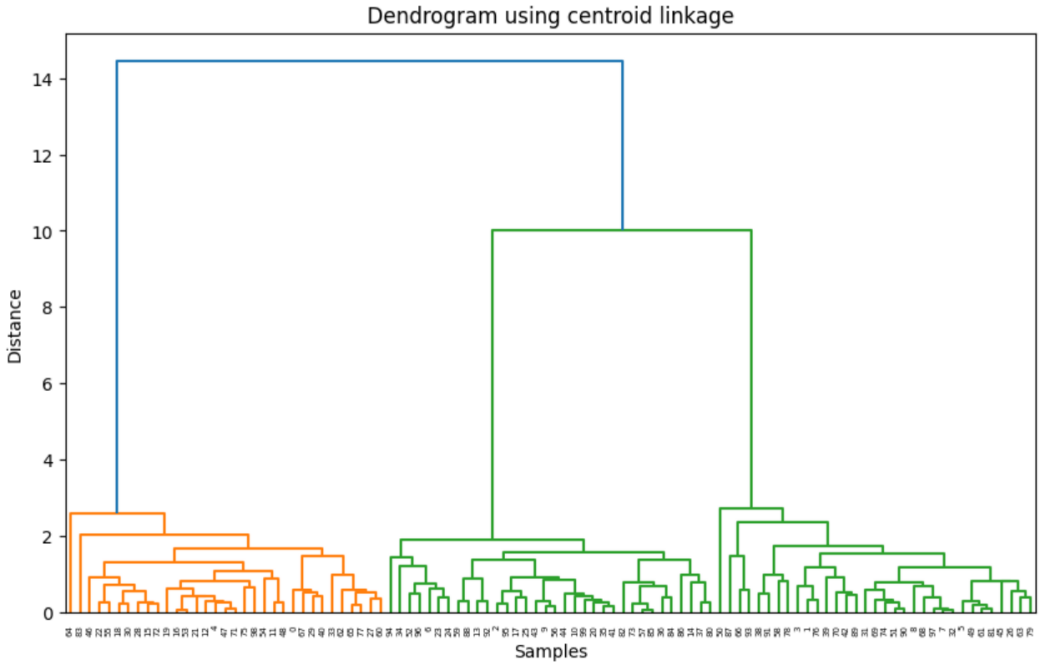
```
if __name__ == "__main__":
  X = generate_dataset()

  methods = ['single', 'complete', 'average', 'centroid']
  for method in methods:
    Z = hierarchical_clustering(X, method)
    visualize_dendrogram(Z, method)

  scores = evaluate_clustering(X)
  print('Silhouette scores for each method:')
  for method, score in scores.items():
    print(f'{method.capitalize()} linkage: {score:.4f}')
```

Show hidden output


Dendrogram using average linkage
Distance
Samples


Dendrogram using complete linkage
Distance
Samples






```
Silhouette scores for each method:
Single linkage: 0.8470
Complete linkage: 0.8470
Average linkage: 0.8470
Centroid linkage: 0.8470
```