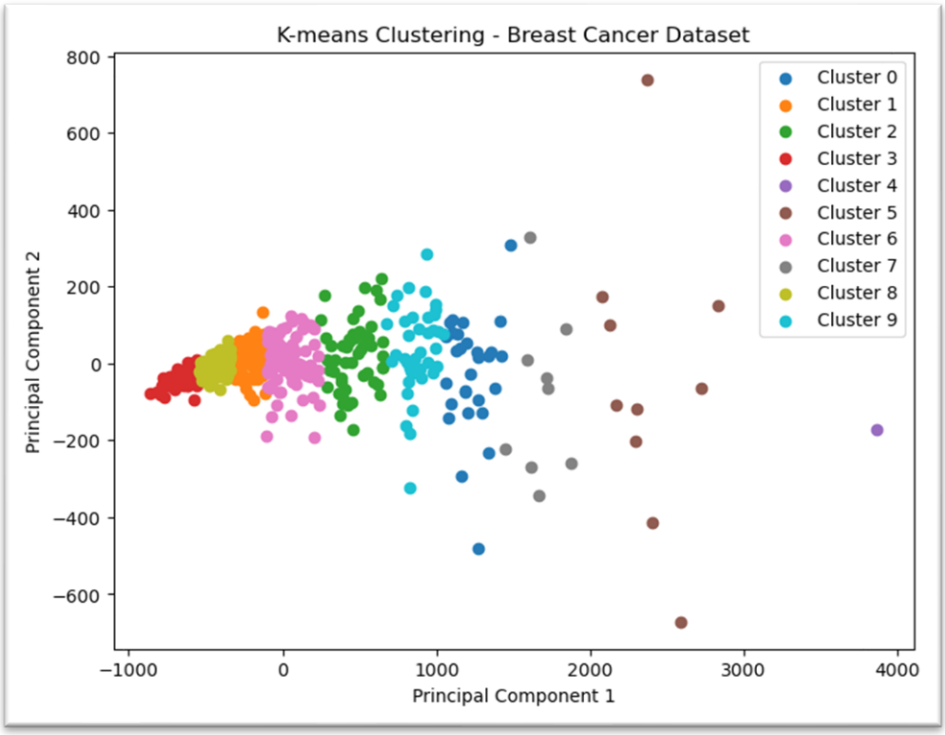
Final Project

Introduction:

In this clustering analysis, we explored three popular clustering methods - k-means clustering, density-based clustering (DBSCAN), and hierarchical clustering. We applied these methods on three different datasets - breast cancer dataset, digits dataset, and iris dataset.

Part 1: K-means Clustering - Breast Cancer Dataset

```
# Import required libraries

import matplotlib.pyplot as plt

from sklearn.datasets import load_breast_cancer

from sklearn.decomposition import PCA

from sklearn.cluster import KMeans

# Load the breast cancer dataset

data = load_breast_cancer().data

# Reduce the dimensions of the dataset to 2 features using PCA

pca = PCA(n_components=2)

df = pca.fit_transform(data)

# Create an object of KMeans with n_clusters = 10

kmeans = KMeans(n_clusters=10)

# Fit and predict the transformed dataset

labels = kmeans.fit_predict(df)

# Plot all the observations on a 2D coordinate system

plt.figure(figsize=(8, 6))

for i in range(10):

    plt.scatter(df[labels == i, 0], df[labels == i, 1], label=f'Cluster {i}')

plt.title('K-means Clustering - Breast Cancer Dataset')

plt.xlabel('Principal Component 1')

plt.ylabel('Principal Component 2')

plt.legend()

plt.show()
```

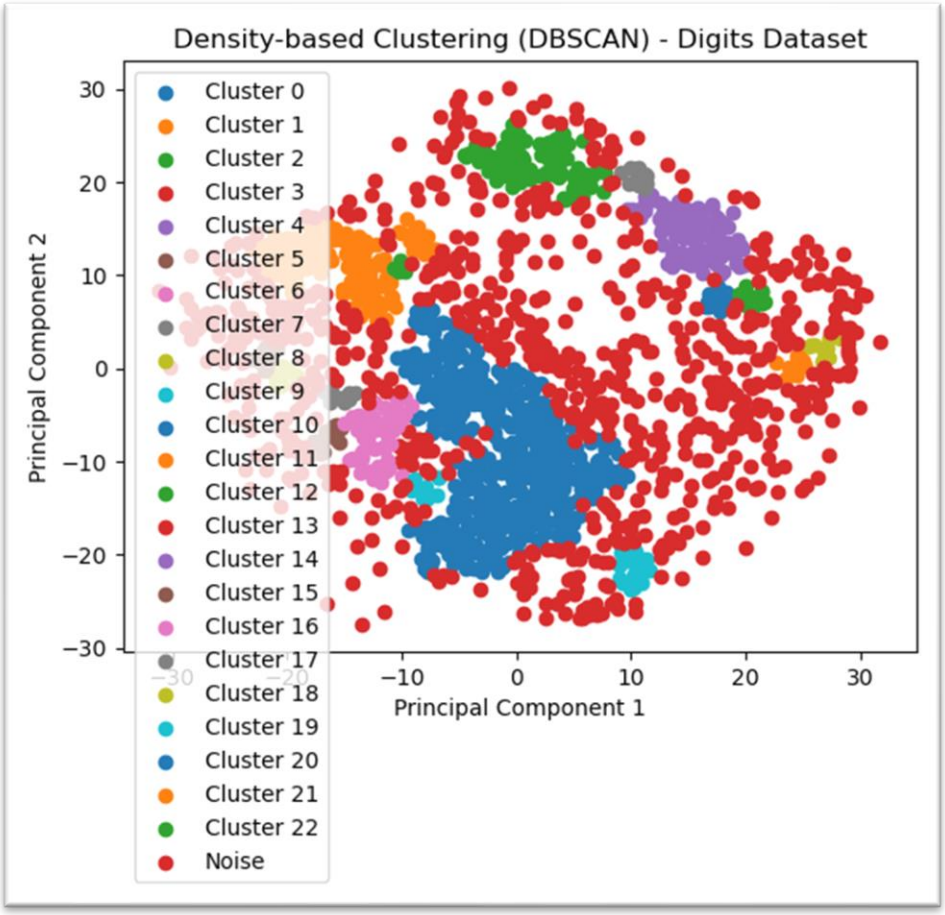


In Part 1, we applied k-means clustering to the breast cancer dataset. After reducing the dataset to 2 dimensions using PCA, we used the KMeans algorithm with 10 clusters. The resulting plot displayed 10 clusters, each marked with a different color, showcasing how k-means partitions the data into distinct groups based on similarity. The plot indicates that the k-means algorithm was able to separate the data into well-defined clusters.

Part 2: Density-based Clustering (DBSCAN) - Digits Dataset

```
#part2

# Import required libraries

from sklearn.datasets import load_digits

from sklearn.decomposition import PCA

from sklearn.cluster import DBSCAN

import matplotlib.pyplot as plt

# Load the digits dataset

data = load_digits().data

# Reduce dimensions to 2 using PCA

pca = PCA(2)

df = pca.fit_transform(data)

# Create a DBSCAN clustering object with min_samples = 10 and eps = 1.5

dbscan = DBSCAN(min_samples=10, eps=1.5)
```

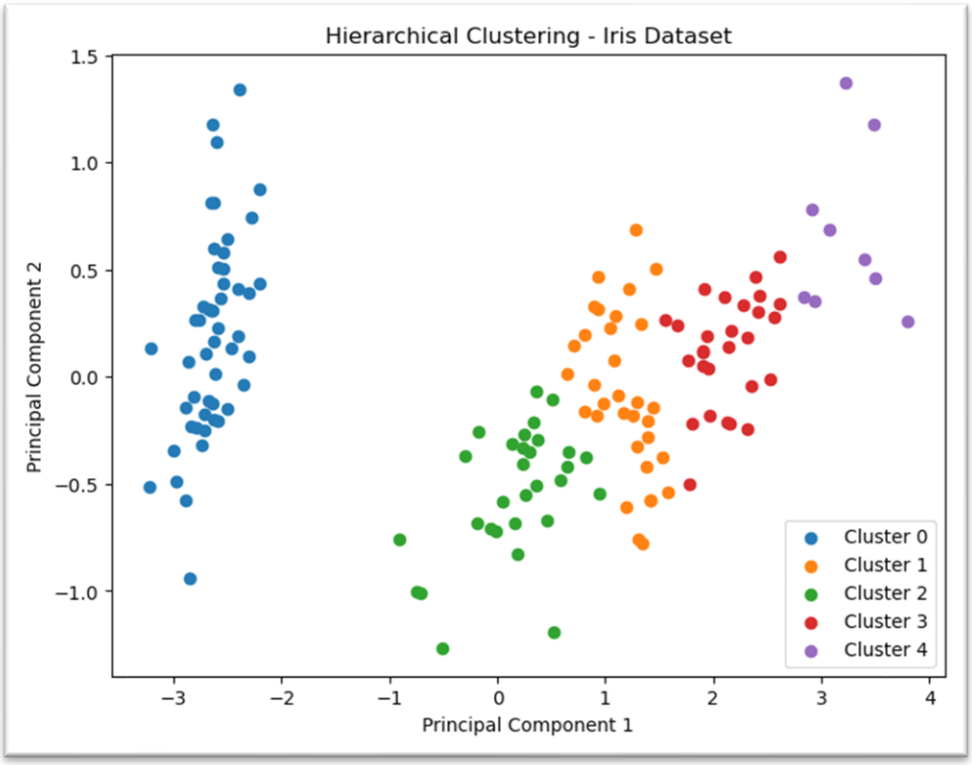
```
# Fit and predict the transformed dataset
labels = dbscan.fit_predict(df)
# Get unique labels
unique_labels = set(labels)
# Plot the observations for each label
for label in unique_labels:
    if label == -1:
        plt.scatter(df[labels == label, 0], df[labels == label, 1], label='Noise')
    else:
        plt.scatter(df[labels == label, 0], df[labels == label, 1], label='Cluster ' + str(label))
plt.title('Density-based Clustering (DBSCAN) - Digits Dataset')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
plt.show()
```



In Part 2, we utilized the DBSCAN algorithm on the digits dataset. Unlike k-means, DBSCAN does not require specifying the number of clusters beforehand and identifies noise points as well. We set the minimum samples and epsilon values to 10 and 1.5, respectively. The plot showed various clusters, each labeled, and some points marked as "Noise," indicating data points that did not belong to any cluster. DBSCAN's ability to detect noise and handle clusters of varying shapes and sizes makes it a powerful tool for certain types of datasets.

Part 3: Hierarchical Clustering - Iris Dataset

```
#part3

# Import required libraries

import matplotlib.pyplot as plt

from sklearn.datasets import load_iris

from sklearn.decomposition import PCA

from sklearn.cluster import AgglomerativeClustering

# Load the iris dataset

data = load_iris().data

# Reduce the dimensions of the dataset to 2 features using PCA

pca = PCA(n_components=2)

df = pca.fit_transform(data)

# Create an object of AgglomerativeClustering with n_clusters = 5

agg_clustering = AgglomerativeClustering(n_clusters=5)

# Fit and predict the transformed dataset

labels = agg_clustering.fit_predict(df)

# Plot all the observations on a 2D coordinate system

plt.figure(figsize=(8, 6))

for i in range(5):

    plt.scatter(df[labels == i, 0], df[labels == i, 1], label=f'Cluster {i}')

plt.title('Hierarchical Clustering - Iris Dataset')

plt.xlabel('Principal Component 1')

plt.ylabel('Principal Component 2')

plt.legend()

plt.show()
```



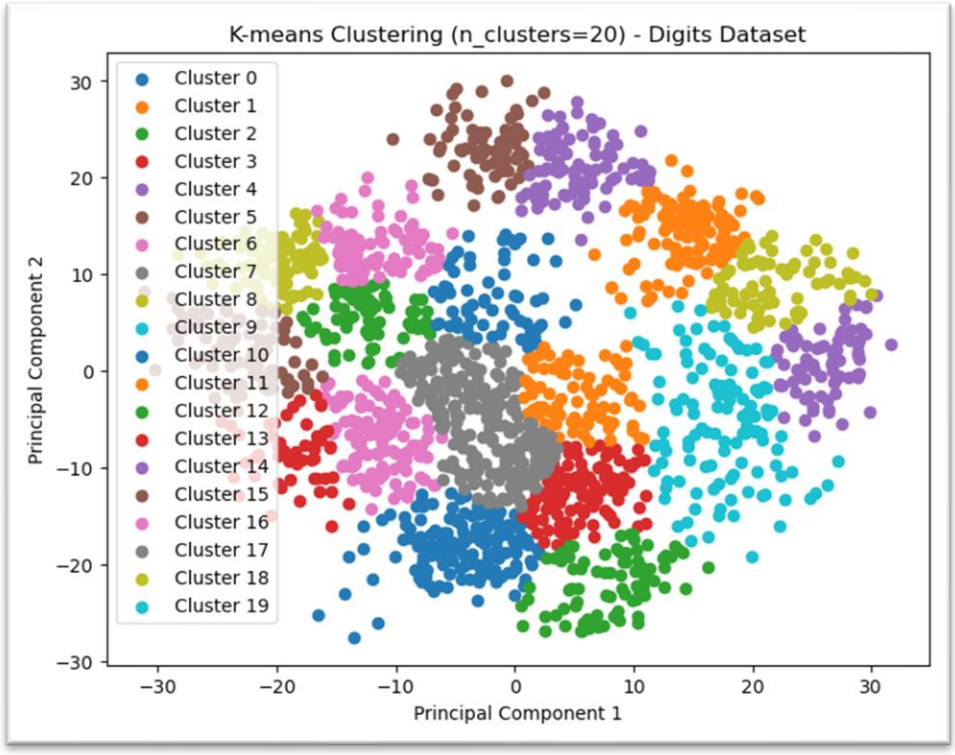
In Part 3, we used the AgglomerativeClustering technique to accomplish hierarchical clustering on the iris dataset with 5 clusters. The figure highlighted the hierarchical nature of this approach, showing how data points are organized into clusters and subclusters until the required number of clusters is reached. Hierarchical clustering provides a clear hierarchical view of the clusters, which can be valuable when interpretability and the understanding of subgroups are important.

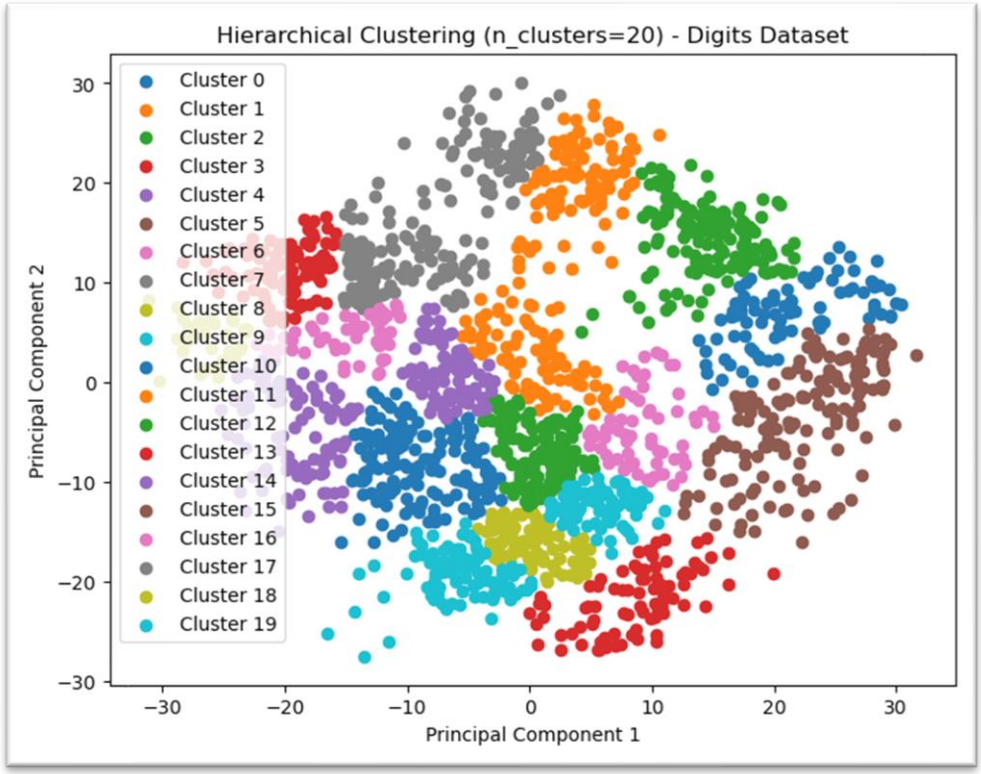
Part 4: Higher n_cluster for k-means and hierarchical clustering on digits dataset

```
# Part 1 (K-means clustering) with n_clusters = 20 and digits dataset

# Import required libraries

import matplotlib.pyplot as plt

from sklearn.datasets import load_digits

from sklearn.decomposition import PCA

from sklearn.cluster import KMeans

# Load the digits dataset

data = load_digits().data

# Reduce the dimensions of the dataset to 2 features using PCA

pca = PCA(n_components=2)

df = pca.fit_transform(data)

# Create an object of KMeans with n_clusters = 20

kmeans = KMeans(n_clusters=20)
```

```
# Fit and predict the transformed dataset
labels = kmeans.fit_predict(df)
# Plot all the observations on a 2D coordinate system
plt.figure(figsize=(8, 6))
for i in range(20):
    plt.scatter(df[labels == i, 0], df[labels == i, 1], label=f'Cluster {i}')
plt.title('K-means Clustering (n_clusters=20) - Digits Dataset')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
plt.show()
# Part 3 (Hierarchical Clustering) with n_clusters = 20 and digits dataset
# Import required libraries
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits
from sklearn.decomposition import PCA
from sklearn.cluster import AgglomerativeClustering
# Load the digits dataset
data = load_digits().data
# Reduce the dimensions of the dataset to 2 features using PCA
pca = PCA(n_components=2)
df = pca.fit_transform(data)
# Create an object of AgglomerativeClustering with n_clusters = 20
agg_clustering = AgglomerativeClustering(n_clusters=20)
# Fit and predict the transformed dataset
labels = agg_clustering.fit_predict(df)
# Plot all the observations on a 2D coordinate system
plt.figure(figsize=(8, 6))
for i in range(20):
    plt.scatter(df[labels == i, 0], df[labels == i, 1], label=f'Cluster {i}')
plt.title('Hierarchical Clustering (n_clusters=20) - Digits Dataset')
```

```
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
plt.show()
```





In Part 4, we repeated k-means and hierarchical clustering on the digits dataset, increasing the number of clusters to 20. The resulting plots displayed 20 clusters, highlighting how the increased number of clusters leads to finer partitioning of the data. K-means demonstrated improved separation of the data into more distinct groups, while hierarchical clustering exhibited more detailed hierarchical structures with more subclusters.

Comparing the three methods under different circumstances.

- **Well-Separated Spherical Clusters:**

For datasets with well-separated spherical clusters, K-means clustering is a suitable choice. K-means efficiently partitions the data into distinct clusters and is computationally efficient, making it suitable for large datasets. However, it is essential to specify the number of clusters in advance, which may not always be feasible or known in real-world scenarios.

DBSCAN may also perform well in such cases, but its strength lies in identifying clusters of different shapes and sizes. For well-separated spherical clusters, DBSCAN may still correctly identify clusters but may not offer significant advantages over K-means in terms of accuracy.

Hierarchical clustering, while capable of handling any cluster shape, may not be the most efficient choice for well-separated spherical clusters due to its dendrogram representation, which can be computationally intensive and less straightforward to interpret for such datasets.

- **Clusters with Varying Shapes and Sizes:**

When dealing with datasets that have clusters with varying shapes and sizes, DBSCAN stands out as a robust choice. DBSCAN does not require specifying the number of clusters in advance and can automatically determine the clusters based on data density. It can effectively identify clusters of any shape, making it versatile for a wide range of datasets.

K-means, on the other hand, assumes convex and isotropic clusters, which may not be ideal for datasets with varying cluster shapes. It could result in suboptimal cluster assignments and may not accurately capture the underlying structure of the data.

Hierarchical clustering may be able to handle clusters with varying shapes, but its scalability and dendrogram representation may become limiting factors, especially for large and complex datasets.

- **Interpretable Hierarchical Structure:**

For datasets where interpretability and a hierarchical view of the clusters are important, hierarchical clustering shines. Hierarchical clustering provides a dendrogram representation that allows researchers to explore the data's hierarchical structure and uncover subclusters at different levels.

DBSCAN and K-means do not naturally provide a hierarchical view of the clusters. While hierarchical clustering can be insightful for smaller datasets with a clear hierarchical organization, it may not be feasible for very large datasets due to its computational complexity.

Conclusion:

The choice of the clustering method depends on the specific characteristics of the dataset and the goals of the analysis. K-means is a good option for datasets with well-separated spherical clusters, while DBSCAN excels in datasets with varying cluster shapes and sizes and handles noise points effectively. Hierarchical clustering is valuable when interpretability and a hierarchical view of the clusters are desired.