CS - 584 Machine Learning
Assignment 3
Name - Soham Kamble
CWID - A20517098

Problem 1

No. of clusters $K = 2$
Initial centroids = x_3 & x_6

1. 1st Iteration

	x_3	x_6	min	cluster
x_1	3	2	2	c_2
x_2	4	7	4	c_1
x_3	0	5	0	c_1
x_4	4	1	1	c_2
x_5	3	8	3	c_1
x_6	5	0	0	c_2
x_7	6	1	1	c_2
			11	

c_1 : x_2, x_3, x_5
c_2 : x_1, x_4, x_6, x_7

Total cost = 11

To choose new mediods

For c_1 : For x_3 mediods
Total distance from $x_3 \to x_2$ & $x_3 \to x_5$
$= 4 + 3 = 7$

For x_2 mediods : Total distance = $x_2 \to x_3$ & $x_2 \to x_5$ = 4 + 1 = 5

For x_5 mediods : Total distance = $x_5 \to x_2$ & $x_5 \to x_3$ = 1 + 3 = 4

∴ x_5 is the smallest distance so x_5 is the new mediod

IIy. for cluster 2.

For x_1 : Total distance = $x_1 \to x_4 + x_1 \to x_6 + x_1 \to x_7$
= 1 + 2 + 3 = 6

For x_4 : Total distance = 1 + 1 + 2 = 4
For x_6 : Total distance = 2 + 1 + 1 = 4
For x_7 : Total distance = 3 + 2 + 1 = 6

∴ The minimum distances are x_4 or x_6 since x_6 is already the mediod we wont change it.

2. 2nd Iteration

New mediods are : x_5 & x_6

	x_5	x_6	min	cluster
x_1	6	2	2	c_2
x_2	1	7	1	c_1
x_3	3	5	3	c_1
x_4	7	1	1	c_2
x_5	0	8	0	c_1
x_6	8	0	0	c_2
x_7	9	1	1	c_2

Total cost = 8

Therefore the new clusters are

c_1 : x_2, x_3, x_5

c_2 : x_1, x_4, x_6, x_7

Total cost = 8

medoids = x_5, x_6

To check for new medoids

For c_1: No points have changed. Hence x_5 is the new medoid.

For c_2: No points have changed. Hence x_6 is the new medoid

3. Since min distances are same and total cost of 2nd Iteration is low the algorithm converges. Therefore the final medoids and clusters are

clusters = c_1 : x_2, x_3, x_5

c_2 : x_1, x_4, x_6, x_7

medoids = x_5, x_6

Problem 2.

Given in the question the marginal distribution $p(z)$ for the latent variable z is

$$p(z) = \prod_{k=1}^{K} \pi_k^{z_k}$$

We also know that the conditional distribution $p(x|z)$ for the observed variable x is given by

$$p(x|z) = \prod_{k=1}^{K} \mathcal{N}(x|\mu_k, \Sigma_k)^{z_k}$$

where $\sum_{k=1}^{K} \pi_k = 1$; $z = [z_1, z_2, \ldots, z_K]$
& $z_k \in \{0, 1\}$

$$\sum_{k=1}^{K} z_k = 1$$

We now define the joint distribution of latent variable and observed variable as. $p(x, z) = p(x|z) \cdot p(z)$

where we know that x is observed variable & z is latent variable
$p(z)$ is marginal distribution
$p(x|z)$ is the conditional distribution

Now we associate probability with each component z_k

$$p(z_k = 1) = \pi_k$$

where π_k should be between 0 & 1 because of the probability function and we know $\sum_{k=1}^{K} \pi_k = 1$

We know $z_k \in \{0, 1\}$ therefore z_k are mutually exclusive and thus making them independent.

Now consider $p(z)$ with single & double components

$$p(z_1) = \pi_1^{z_1}$$
$$p(z_1, z_2) = \pi_1^{z_1} \pi_2^{z_2}$$

The conditional probability for a particular component of z would be

$$p(x | z_k = 1) = \mathcal{N}(x | \mu_k, \Sigma_k)$$

which implies $p(x|z)$ can be written in the form

$$p(x|z) = \prod_{k=1}^{K} \mathcal{N}(x | \mu_k, \Sigma_k)^{z_k}$$

By the fact we know that $z_k = 1$ for all product terms except ~~the~~ for one

We know that the marginal distribution of x is then obtained by summing over

all the possible states of z to give

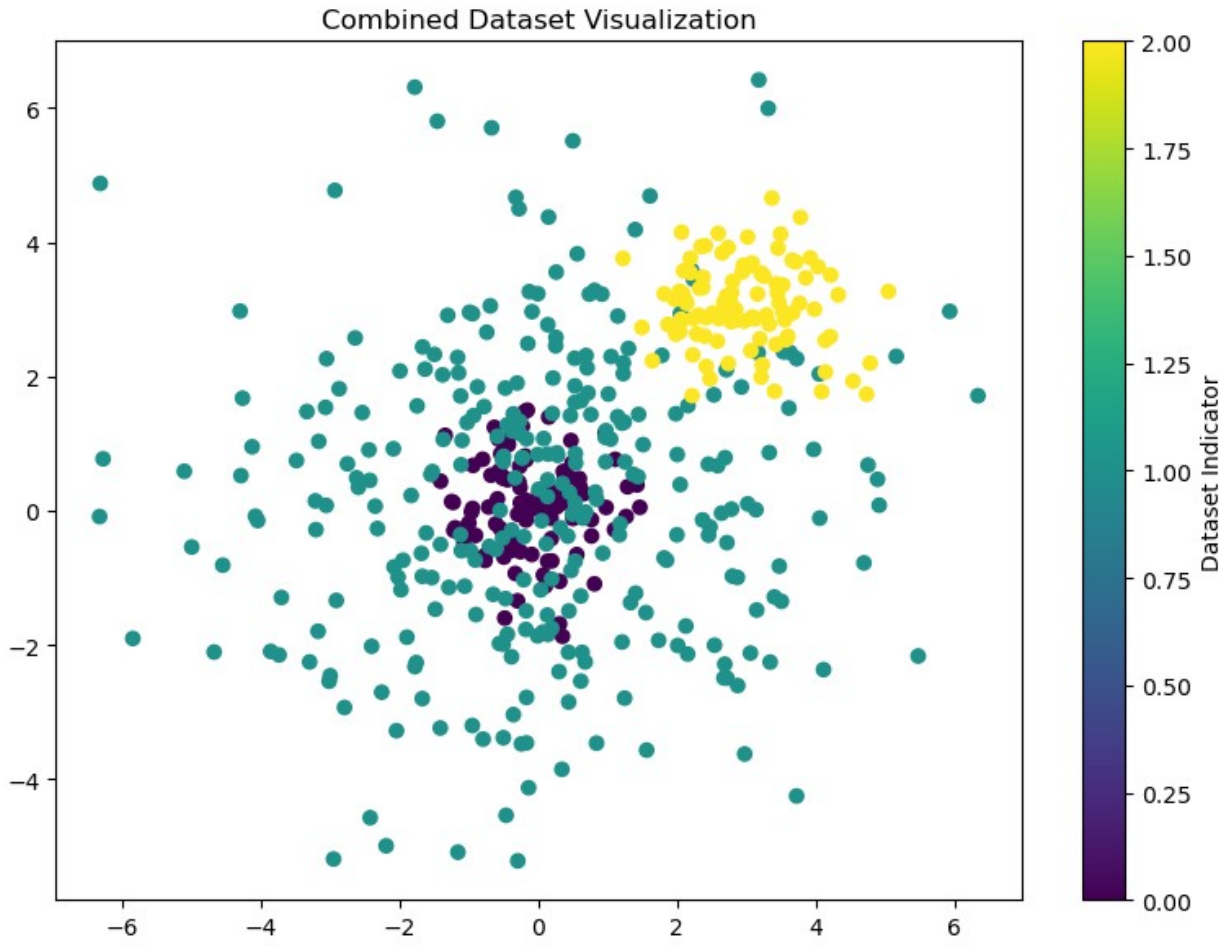
$$p(x) = \sum_z p(z)\, p(x|z)$$

$$= \sum_z \prod_{k=1}^{K} \pi^{z_k} \mathcal{N}(x|\mu_k, \Sigma_k)^{z_k}$$

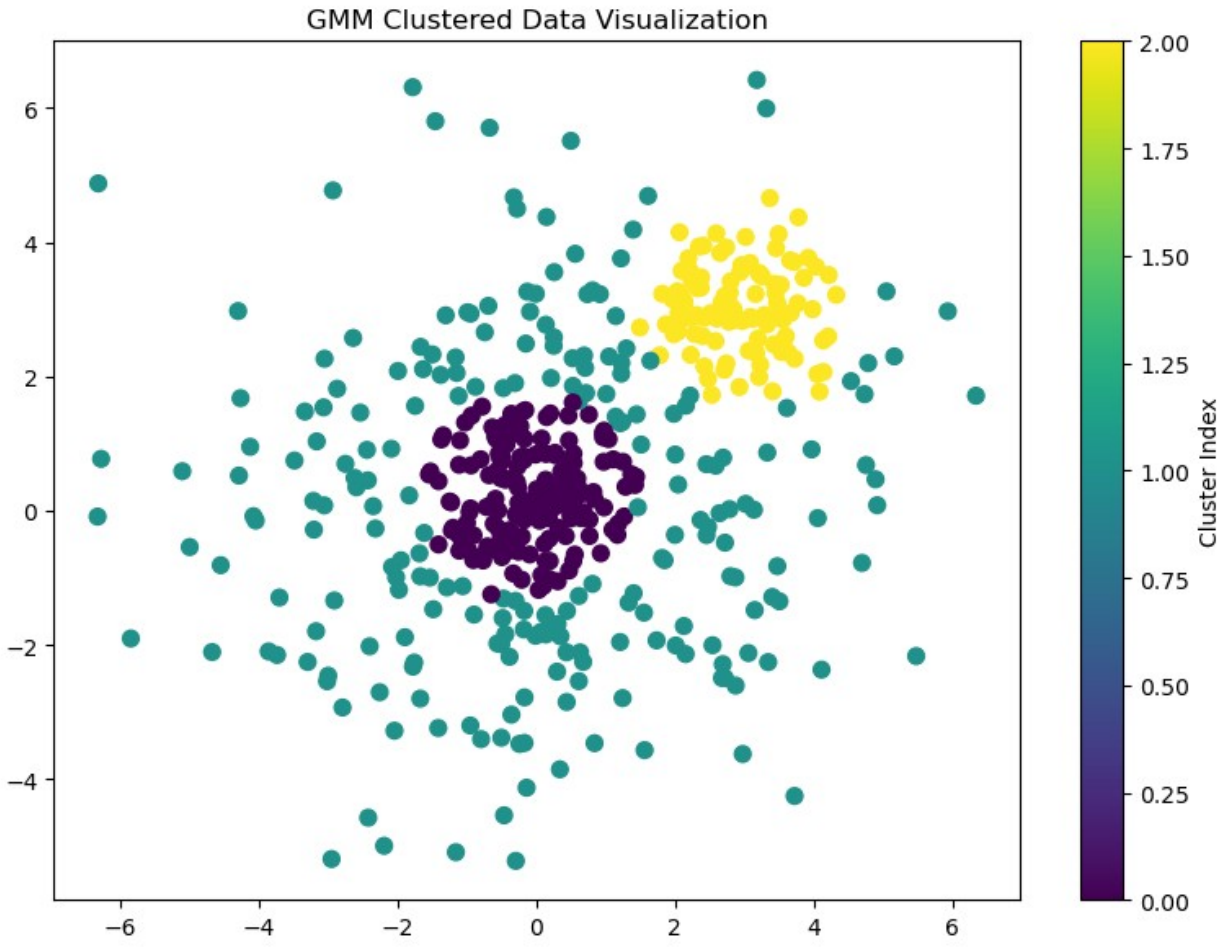
$$= \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k)$$

since $z_k \in \{0, 1\}$

```
import numpy as np
import matplotlib.pyplot as plt

import numpy as np
import matplotlib.pyplot as plt

# DataSet A
mean_A = [0, 0]
covariance_A = np.identity(2) * 0.5
X_A = np.random.multivariate_normal(mean_A, covariance_A, 100)

# DataSet B
mean_B = [0, 0]
covariance_B = np.identity(2) * 5
X_B = np.random.multivariate_normal(mean_B, covariance_B, 300)

# DataSet C
mean_C = [3, 3]
covariance_C = np.identity(2) * 0.5
X_C = np.random.multivariate_normal(mean_C, covariance_C, 100)

# Combined datasets
X_combined = np.concatenate((X_A, X_B, X_C), axis=0)
y_combined = np.concatenate((np.zeros(100), np.ones(300),
(np.ones(100) * 2)), axis=0)


# Create a scatter plot for the combined data
plt.figure(figsize=(8, 6))
plt.scatter(X_combined[:, 0], X_combined[:, 1], c=y_combined,
cmap='viridis')
plt.title("Combined Dataset Visualization")
plt.colorbar().set_label('Dataset Indicator')
plt.tight_layout()
plt.show()
```

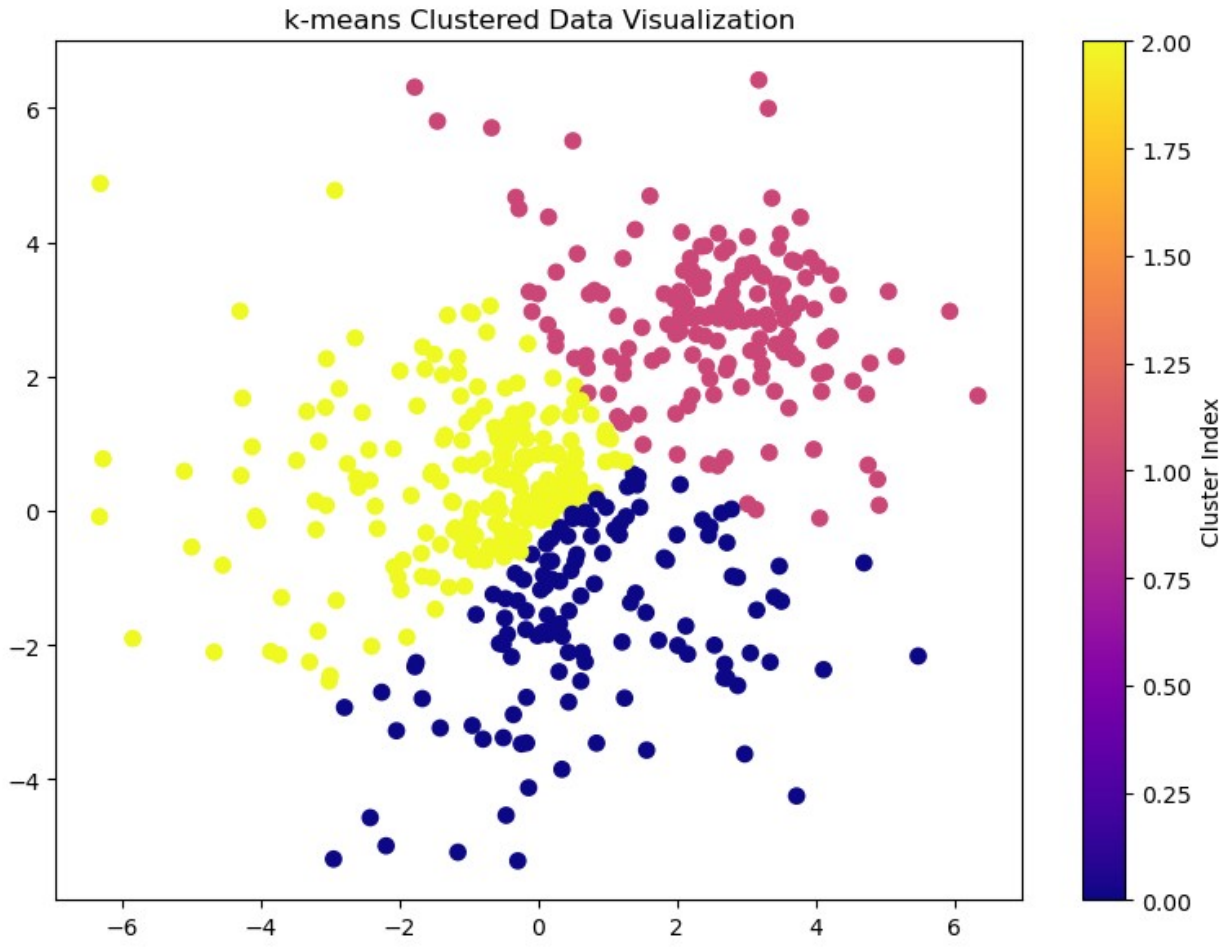



```
from sklearn.mixture import GaussianMixture

gmm_model = GaussianMixture(n_components=3,
covariance_type='spherical').fit(X_combined)
predicted_labels = gmm_model.predict(X_combined)

# Plot the results from the Gaussian Mixture Model
fig, ax = plt.subplots(figsize=(8, 6))
scatter = ax.scatter(X_combined[:, 0], X_combined[:, 1],
c=predicted_labels, s=50, cmap='viridis')
ax.set_title("GMM Clustered Data Visualization")
fig.colorbar(scatter, ax=ax).set_label('Cluster Index')
plt.tight_layout()
plt.show()
```




```
from sklearn.cluster import KMeans

# Fitting the KMeans clustering model
kmeans_model = KMeans(n_clusters=3, random_state=0)
predicted_clusters = kmeans_model.fit_predict(X_combined)

# Visualizing the K-means Clustering outcomes
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(1, 1, 1)
cluster_scatter = ax.scatter(X_combined[:, 0], X_combined[:, 1],
c=predicted_clusters, s=45, cmap='plasma')
ax.set_title("k-means Clustered Data Visualization")
fig.colorbar(cluster_scatter).set_label('Cluster Index')
plt.tight_layout()
plt.show()
```

```
/Users/amaterasu/anaconda3/lib/python3.11/site-packages/sklearn/
cluster/_kmeans.py:1412: FutureWarning: The default value of `n_init`
will change from 10 to 'auto' in 1.4. Set the value of `n_init`
explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```



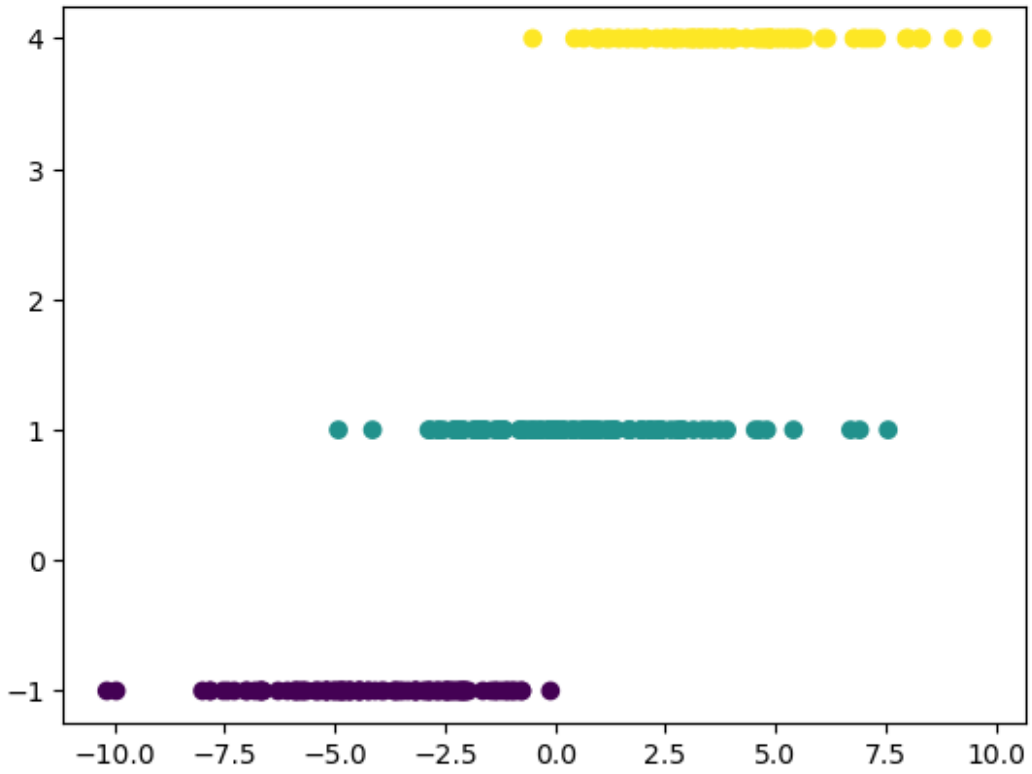

```
# Calculate accuracy for Gaussian Mixture Model
gmm_accuracy = (np.sum(predicted_labels == y_combined) /
len(y_combined)) * 100

# Calculate accuracy for KMeans clustering
kmeans_accuracy = (np.sum(predicted_clusters == y_combined) /
len(y_combined)) * 100

# Display the clustering accuracies
print(f"Gaussian Mixture Model Accuracy: {gmm_accuracy}%")
print(f"K-means Clustering Accuracy: {kmeans_accuracy}%")
```

```
Gaussian Mixture Model Accuracy: 80.2%
K-means Clustering Accuracy: 18.0%
```

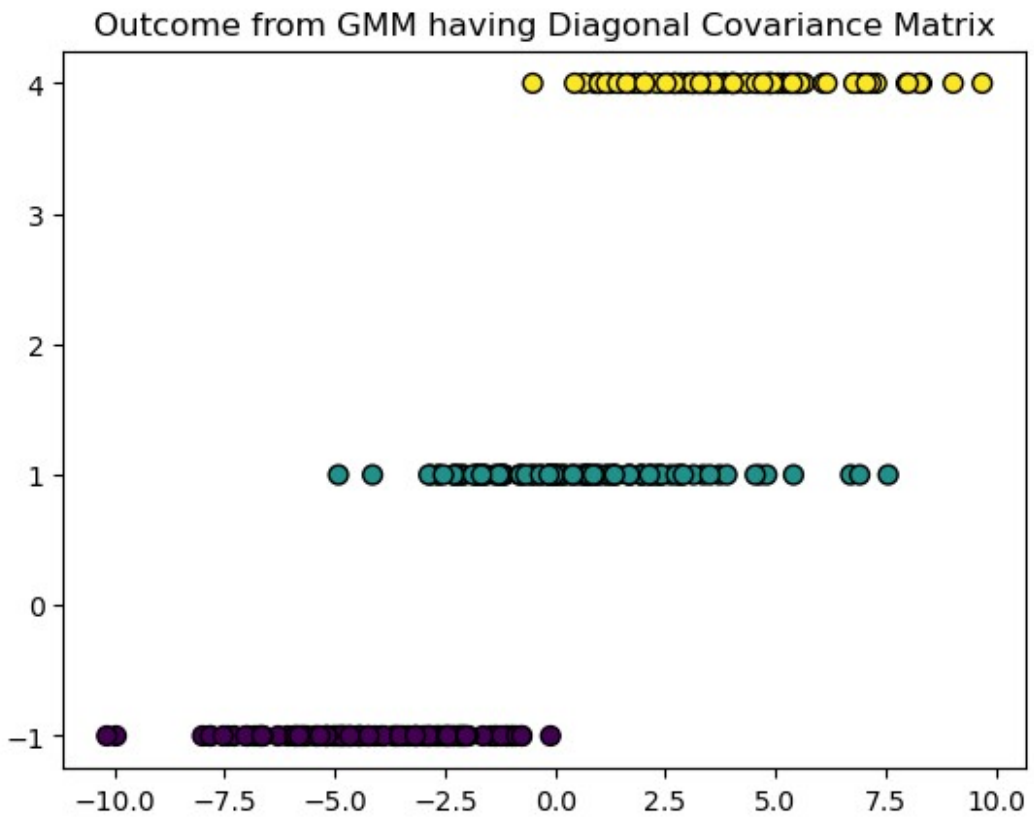
```
print("As observed, the Gaussian mixture model performs well but k-
means does not")
```

```
As observed, the Gaussian mixture model performs well but k-means does
not
```

```
import numpy as np
import matplotlib.pyplot as plt

# Data for Cluster 1
group1_center = [0, 0]
group1_covariance = [[1, 0], [0, 10]]
group1_samples = np.random.multivariate_normal(group1_center,
group1_covariance, 100)

# Data for Cluster 2
group2_center = [3, 3]
group2_covariance = [[10, 0], [0, 1]]
group2_samples = np.random.multivariate_normal(group2_center,
group2_covariance, 100)

# Data for Cluster 3
group3_center = [6, 6]
group3_covariance = [[8, 0], [0, 8]]
group3_samples = np.random.multivariate_normal(group3_center,
group3_covariance, 100)

# Combining samples from all clusters
combined_samples = np.concatenate((group1_samples, group2_samples,
group3_samples), axis=0)
group_labels = np.concatenate((np.zeros(100), np.ones(100),
(np.ones(100)*2)), axis=0)

plt.scatter(all_samples[:, 0], all_samples[:, 1], c=cluster_labels,
cmap='viridis')
plt.show()
```



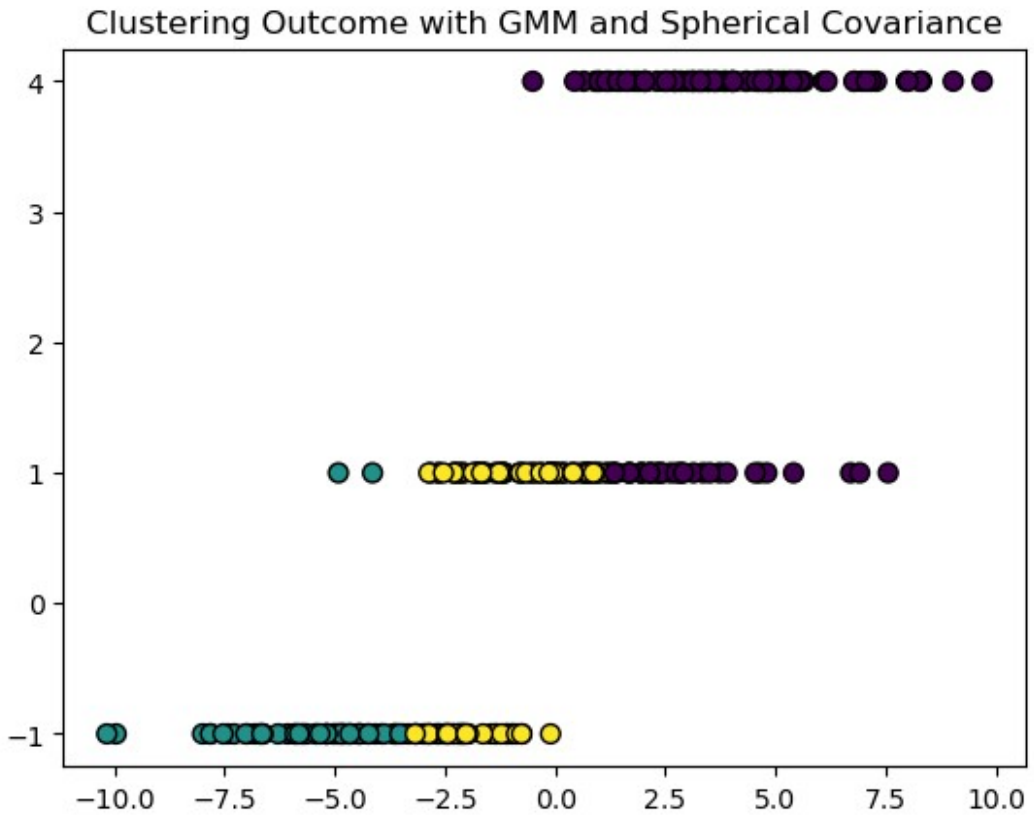

```
from sklearn.mixture import GaussianMixture

# Create and fit a Gaussian Mixture Model with 3 components and
diagonal covariance
gmm_model = GaussianMixture(n_components=3,
covariance_type='diag').fit(all_samples)

# Predict cluster labels using the GMM
predicted_labels = gmm_model.predict(all_samples)

# Plot the results
plt.title("Outcome from GMM having Diagonal Covariance Matrix")
plt.scatter(all_samples[:, 0], all_samples[:, 1], c=predicted_labels,
s=50, edgecolor='k')
plt.show()
```




```
print("As we can see this is very similar to the original plot")
```

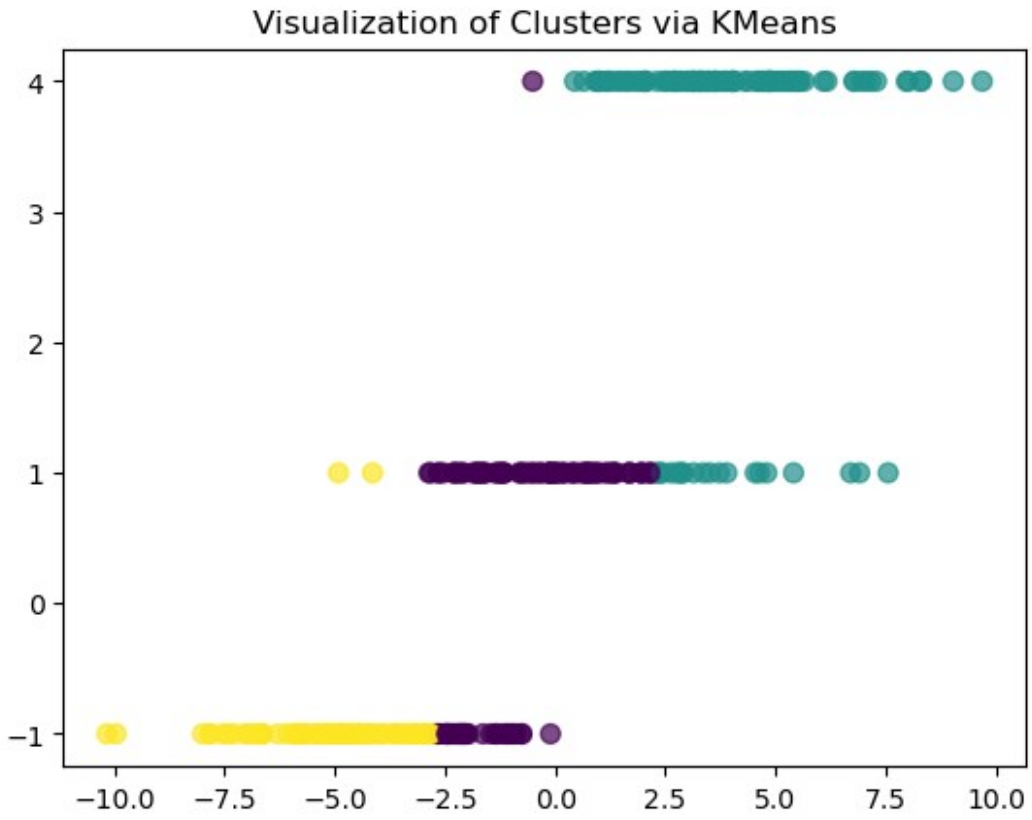
```
As we can see this is very similar to the original plot
```

```
# Initialize and fit a Gaussian Mixture Model with 3 components and
spherical covariance
spherical_gmm = GaussianMixture(n_components=3,
covariance_type='spherical').fit(all_samples)

# Obtain the cluster assignments using the spherical GMM
spherical_labels = spherical_gmm.predict(all_samples)

# Display the clustering outcome
plt.title("Clustering Outcome with GMM and Spherical Covariance")
plt.scatter(all_samples[:, 0], all_samples[:, 1], c=spherical_labels,
s=50, edgecolor='k')
plt.show()
```




```
from sklearn.cluster import KMeans

# Initialize and fit a KMeans clustering with 3 clusters
cluster_model = KMeans(n_clusters=3, random_state=0)
kmeans_clusters = cluster_model.fit_predict(all_samples)

# Visualize the clustering results
# Display clustering visualization
plt.title("Visualization of Clusters via KMeans")
plt.scatter(all_samples[:, 0], all_samples[:, 1], c=kmeans_clusters,
s=50, alpha=0.7)
plt.show()
```

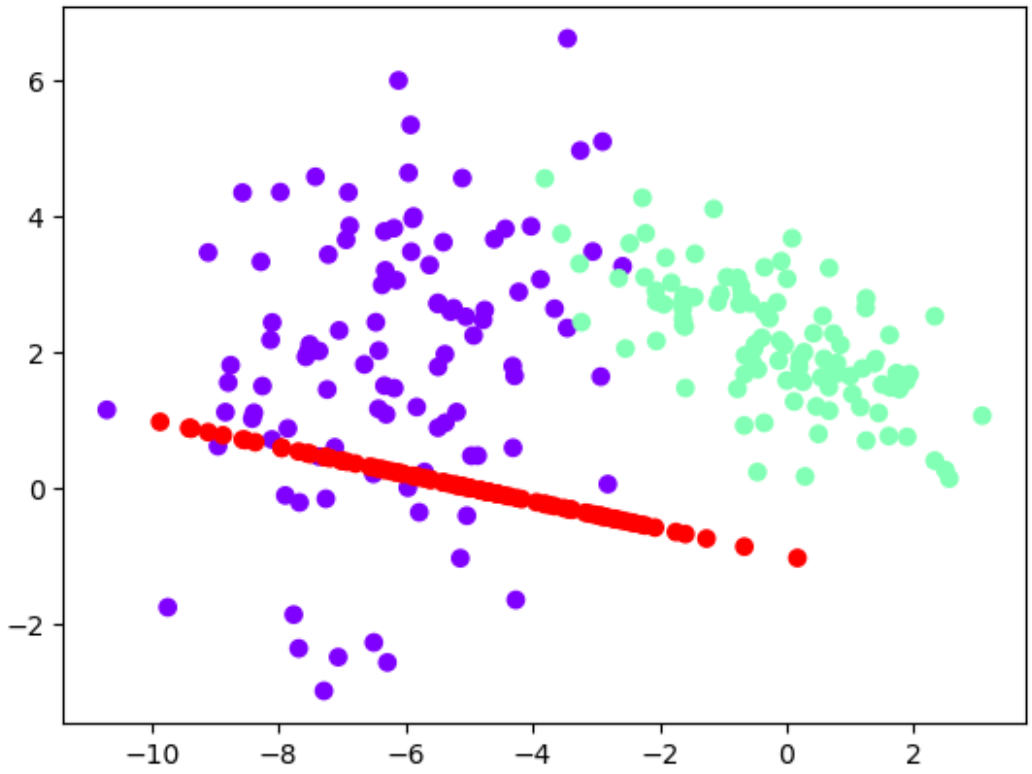
```
/Users/amaterasu/anaconda3/lib/python3.11/site-packages/sklearn/
cluster/_kmeans.py:1412: FutureWarning: The default value of `n_init`
will change from 10 to 'auto' in 1.4. Set the value of `n_init`
explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```



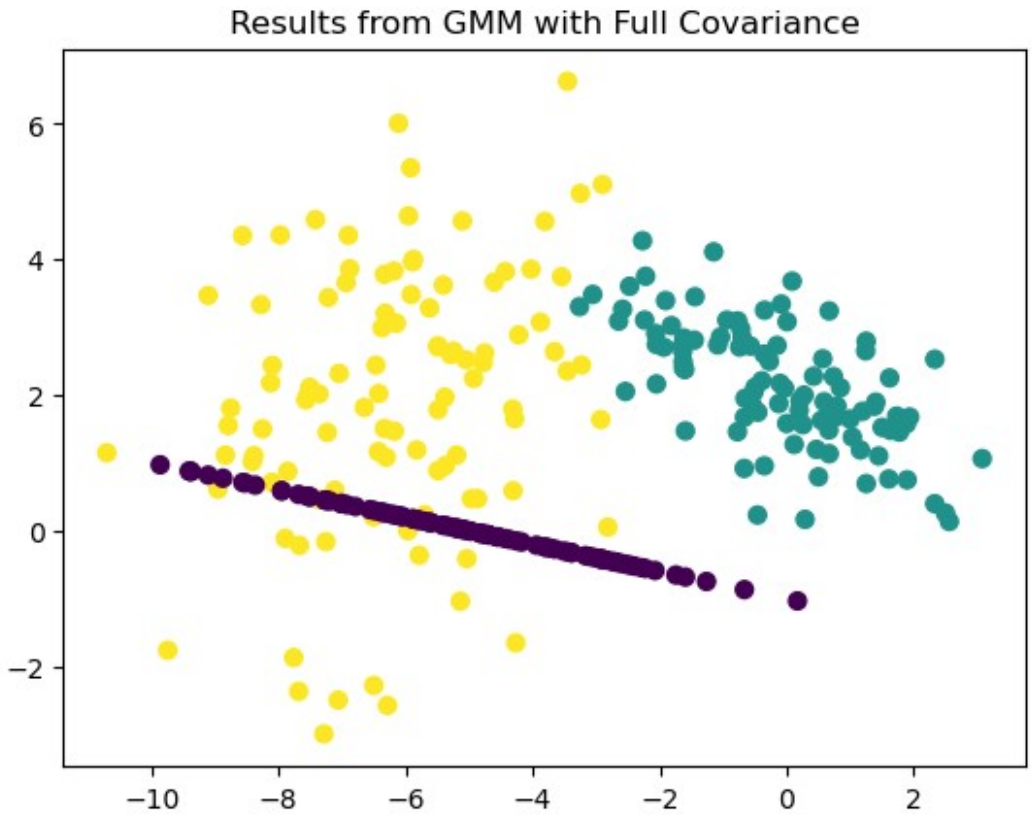

```
print("KMeans and GMM with spherical covariance don't yield optimal results.")
```

```
KMeans and GMM with spherical covariance don't yield optimal results.
```

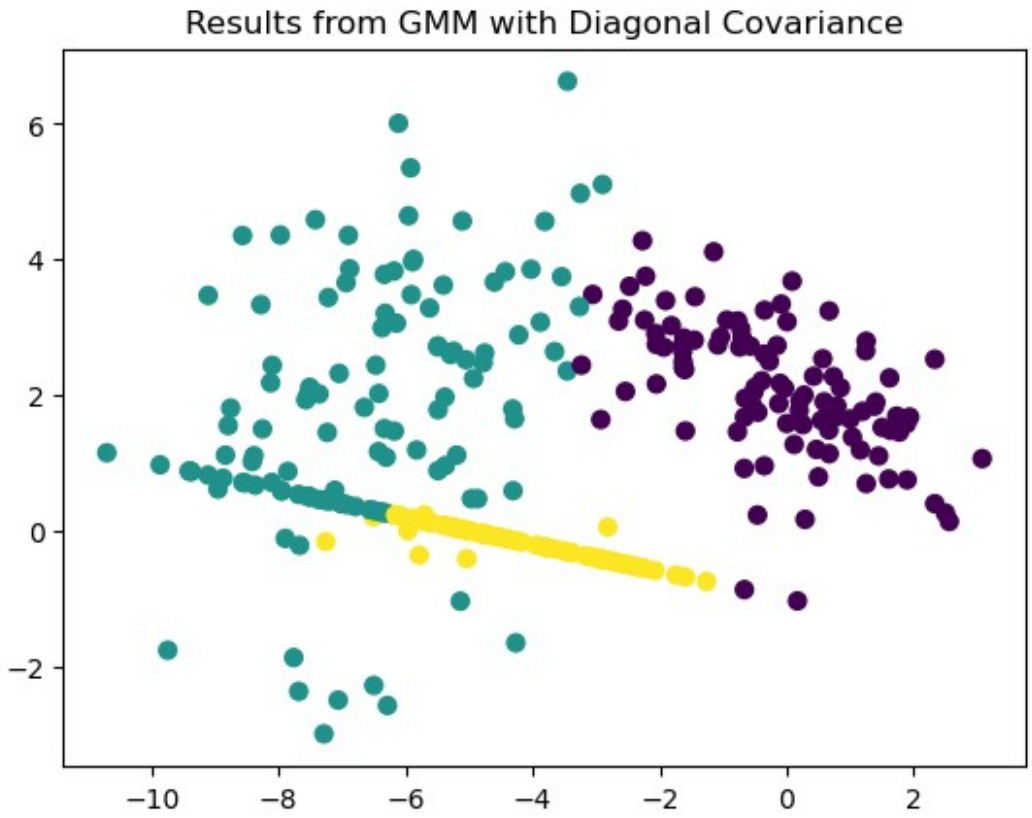
```
import numpy as np
import matplotlib.pyplot as plt

# Data for Cluster 1
group1_center = [-6, 2]
group1_covariance = [[3, 0], [0, 3]]  # High variance in y-axis, low
in x-axis
group1_samples = np.random.multivariate_normal(group1_center,
group1_covariance, 100)

# Data for Cluster 2
group2_center = [0, 2]
group2_covariance = [[2, -1], [-1, 1]]  # High variance in x-axis, low
in y-axis
group2_samples = np.random.multivariate_normal(group2_center,
group2_covariance, 100)

# Data for Cluster 3
group3_center = [-5, 0]
group3_covariance = [[5, -1], [-1, 0.2]]  # Moderate variance in both
x and y axes
group3_samples = np.random.multivariate_normal(group3_center,
group3_covariance, 100)

# Combining samples from all clusters
combined_samples = np.concatenate((group1_samples, group2_samples,
group3_samples), axis=0)
group_labels = np.concatenate((np.zeros(100), np.ones(100),
(np.ones(100)*2)), axis=0)

plt.scatter(combined_samples[:,0], combined_samples[:,1],
c=group_labels, cmap='rainbow')
plt.show()
```




```
from sklearn.mixture import GaussianMixture

# Train a GMM model
gmm_model = GaussianMixture(3,
covariance_type='full').fit(combined_samples)
predicted_labels = gmm_model.predict(combined_samples)

# Visualize the results
plt.title("Results from GMM with Full Covariance")
plt.scatter(combined_samples[:, 0], combined_samples[:, 1],
c=predicted_labels, s=40)
plt.show()
```




```
# Train a GMM model with diagonal covariance
gmm_diagonal = GaussianMixture(3,
covariance_type='diag').fit(combined_samples)
predicted_diagonal_labels = gmm_diagonal.predict(combined_samples)

# Visualize the results
plt.title("Results from GMM with Diagonal Covariance")
plt.scatter(combined_samples[:, 0], combined_samples[:, 1],
c=predicted_diagonal_labels, s=40)
plt.show()
```




```
from sklearn.cluster import KMeans

# Initialize and train KMeans model
kmeans_model = KMeans(3, random_state=0)
kmeans_labels = kmeans_model.fit_predict(combined_samples)

# Visualize clustering results
plt.scatter(combined_samples[:, 0], combined_samples[:, 1],
c=kmeans_labels, s=40)
plt.title("KMeans Clustering Results")
plt.show()
```

```
/Users/amaterasu/anaconda3/lib/python3.11/site-packages/sklearn/
cluster/_kmeans.py:1412: FutureWarning: The default value of `n_init`
will change from 10 to 'auto' in 1.4. Set the value of `n_init`
explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

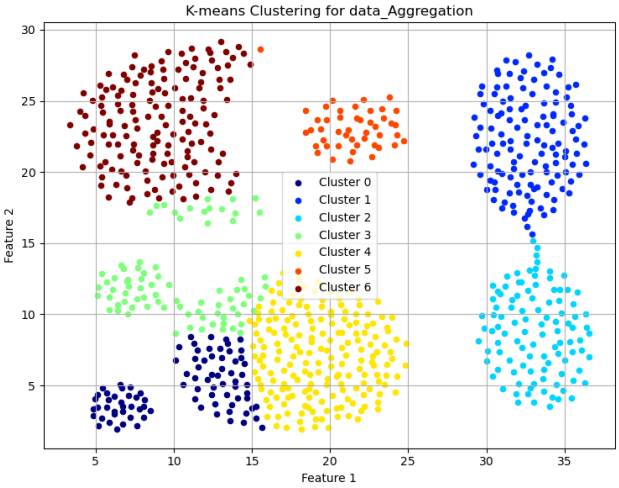

```
print("From the observations, it's evident that the GMM with
unrestricted covariance succeeds, while the others fall short.")
```

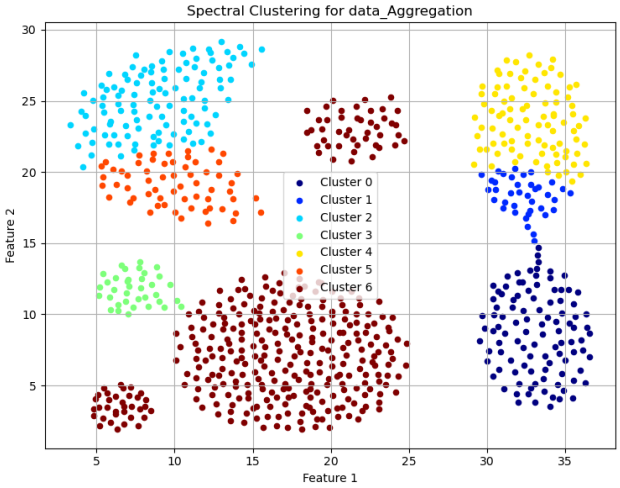
```
From the observations, it's evident that the GMM with unrestricted
covariance succeeds, while the others fall short.
```

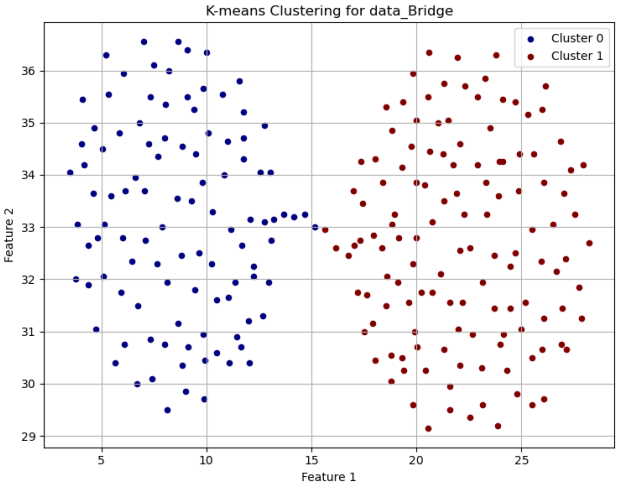
```
import numpy as np
import scipy.io
import matplotlib.pyplot as plt
from scipy.spatial.distance import cdist
from scipy.linalg import eigh
from sklearn.metrics import accuracy_score


def my_spectralclustering(data, K, sigma):
    # 1. Create Affinity Matrix using Gaussian Kernel
    pairwise_distances = cdist(data, data, 'euclidean')
    affinity_matrix = np.exp(-pairwise_distances ** 2 / (2 * sigma **
2))

    # 2. Calculate the Laplacian Matrix
    diagonal_matrix = np.diag(np.sum(affinity_matrix, axis=1))
    laplacian_matrix = diagonal_matrix - affinity_matrix

    # 3. Extract the first K eigenvectors
    eigenvalues, eigenvectors = eigh(laplacian_matrix)
    features = eigenvectors[:, :K]

    # 4. Use these eigenvectors as features to cluster data points
using K-means
    labels = my_kmeans(features, K)

    return labels

def my_kmeans(data, K):
    # Initialize centroids by picking K random samples from the data
    centroids = data[np.random.choice(data.shape[0], K,
replace=False)]

    labels = np.zeros(data.shape[0], dtype=int)
    while True:
        # Assign each data point to the closest centroid
        distances = np.linalg.norm(data[:, np.newaxis] - centroids,
axis=2)
        new_labels = np.argmin(distances, axis=1)

        # If labels don't change, we've reached convergence
        if np.array_equal(labels, new_labels):
            break

        labels = new_labels

        # Update the centroids
        for k in range(K):
            centroids[k] = np.mean(data[labels == k], axis=0)
```

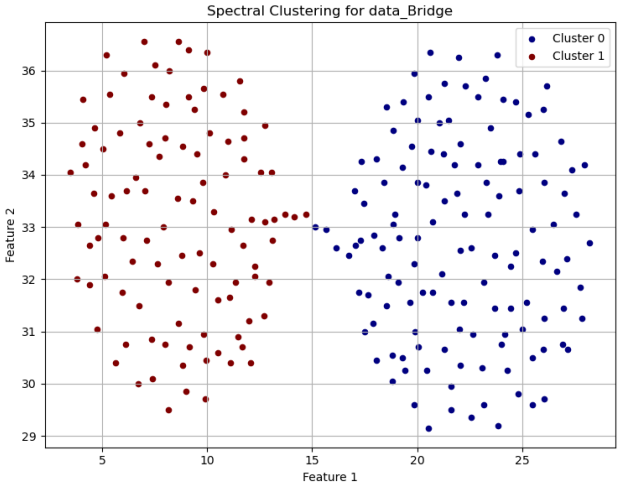
```
    return labels

def plot_clustering(data, labels, dataset_name):
    # Get the number of unique clusters
    unique_labels = np.unique(labels)
    colors = plt.cm.jet(np.linspace(0, 1, len(unique_labels)))

    plt.figure(figsize=(10, 6))
    for i, label in enumerate(unique_labels):
        cluster_data = data[labels == label]
        plt.scatter(cluster_data[:, 0], cluster_data[:, 1],
color=colors[i], label=f"Cluster {label}", s=20)

    plt.title(f"Clustering results for {dataset_name}")
    plt.xlabel("Feature 1")
    plt.ylabel("Feature 2")
    plt.legend(loc="best")
    plt.grid(True)

    # Save the figure to a file before displaying it
    file_name = dataset_name.split('.')[0] + "_clustering_result.png"
    plt.savefig(file_name, dpi=300, bbox_inches='tight')

    plt.show()

def plot_side_by_side(data, labels_kmeans, labels_spectral,
dataset_name):
    unique_labels_kmeans = np.unique(labels_kmeans)
    unique_labels_spectral = np.unique(labels_spectral)

    colors = plt.cm.jet(np.linspace(0, 1,
max(len(unique_labels_kmeans), len(unique_labels_spectral))))

    plt.figure(figsize=(15, 6))

    # K-means clustering plot
    plt.subplot(1, 2, 1)
    for i, label in enumerate(unique_labels_kmeans):
        cluster_data = data[labels_kmeans == label]
        plt.scatter(cluster_data[:, 0], cluster_data[:, 1],
color=colors[i], label=f"Cluster {label}", s=20)
    plt.title(f"K-means Clustering for {dataset_name}")
    plt.xlabel("Feature 1")
    plt.ylabel("Feature 2")
    plt.legend(loc="best")
    plt.grid(True)

    # Spectral clustering plot
    plt.subplot(1, 2, 2)
```

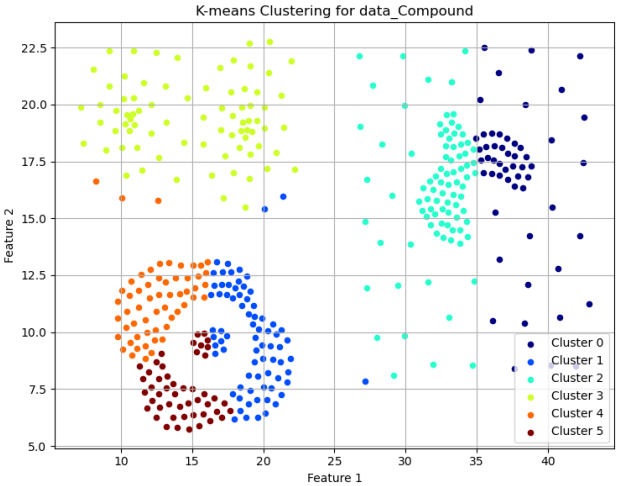
```
    for i, label in enumerate(unique_labels_spectral):
        cluster_data = data[labels_spectral == label]
        plt.scatter(cluster_data[:, 0], cluster_data[:, 1],
color=colors[i], label=f"Cluster {label}", s=20)
    plt.title(f"Spectral Clustering for {dataset_name}")
    plt.xlabel("Feature 1")
    plt.ylabel("Feature 2")
    plt.legend(loc="best")
    plt.grid(True)

    # Save the figure to a file before displaying it
    file_name = dataset_name.replace(" ", "_") +
"_clustering_results.png"
    plt.savefig(file_name, dpi=300, bbox_inches='tight')

    plt.tight_layout()
    plt.show()

def Run_clustering(datasets, sigma_values):
    results_kmeans = {}
    results_spectral = {}

    for dataset, sigma in zip(datasets, sigma_values):
        # Load data from the .mat file
        data = scipy.io.loadmat(dataset)["D"]
        K = len(np.unique(scipy.io.loadmat(dataset)["L"]))

        # Cluster using K-means
        labels_kmeans = my_kmeans(data, K)
        results_kmeans[dataset] = labels_kmeans

        # Cluster using Spectral Clustering
        labels_spectral = my_spectralclustering(data, K, sigma)
        results_spectral[dataset] = labels_spectral

        # Plot results side by side
        dataset_name = dataset.split('.')[0]
        plot_side_by_side(data, labels_kmeans, labels_spectral,
dataset_name)

    return results_kmeans, results_spectral

datasets = ["data_Aggregation.mat", "data_Bridge.mat",
"data_Compound.mat", "data_Flame.mat", "data_Jain.mat",
"data_Spiral.mat", "data_TwoDiamonds.mat"]
sigma_values = [0.5, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3]  # Sample sigma
values; adjust them for best results for each dataset.
Run_clustering(datasets, sigma_values)
```

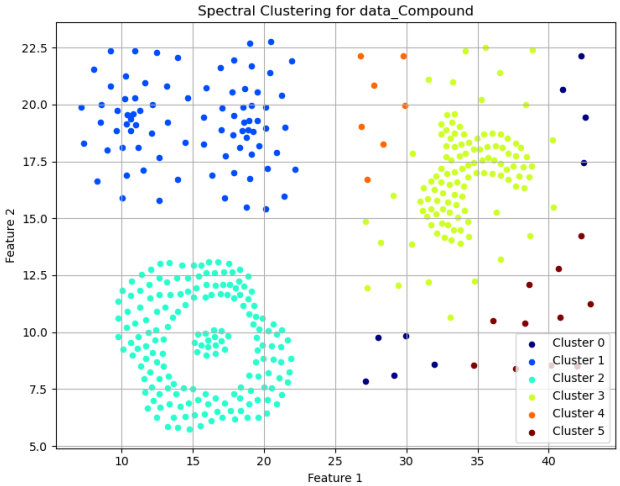

K-means Clustering for data_Aggregation
Feature 1
Feature 2
Cluster 0
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
Cluster 6

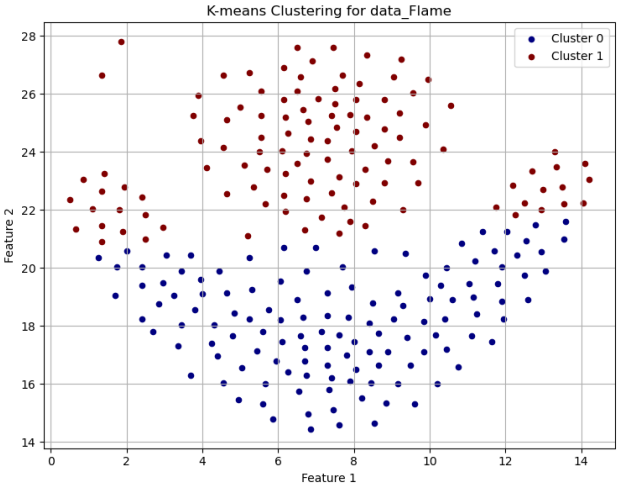

Spectral Clustering for data_Aggregation
Feature 1
Feature 2
Cluster 0
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
Cluster 6

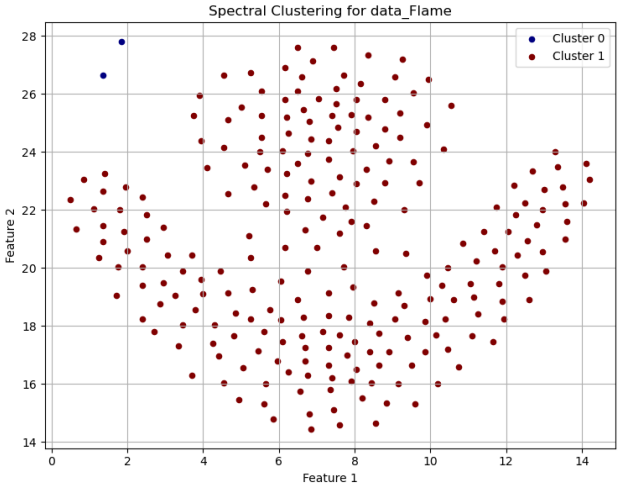

K-means Clustering for data_Bridge
Feature 1
Feature 2
Cluster 0
Cluster 1

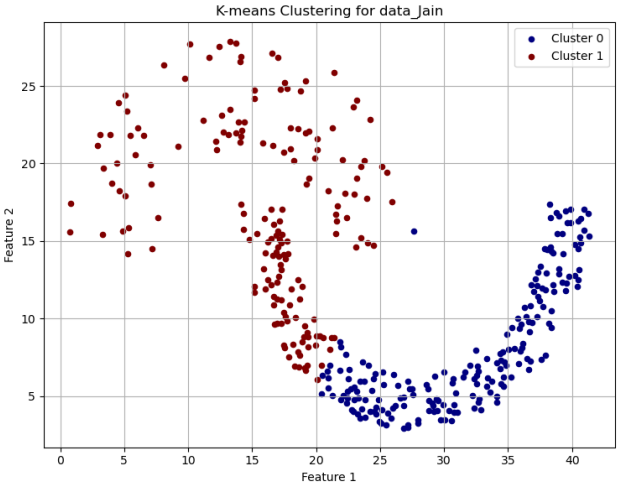

Spectral Clustering for data_Bridge
Feature 1
Feature 2
Cluster 0
Cluster 1

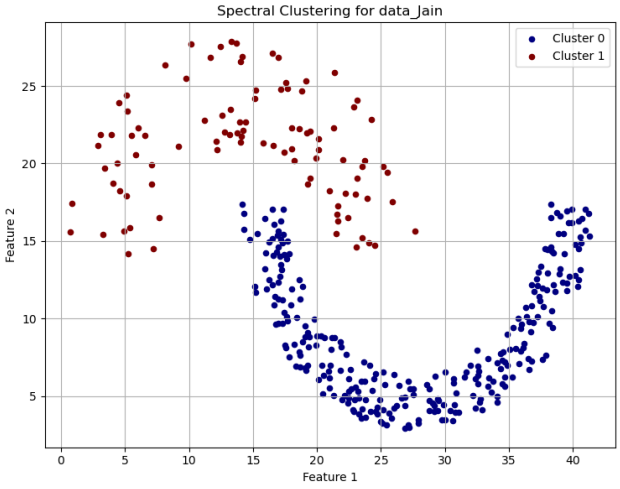

K-means Clustering for data_Compound
Feature 1
Feature 2
Cluster 0
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5

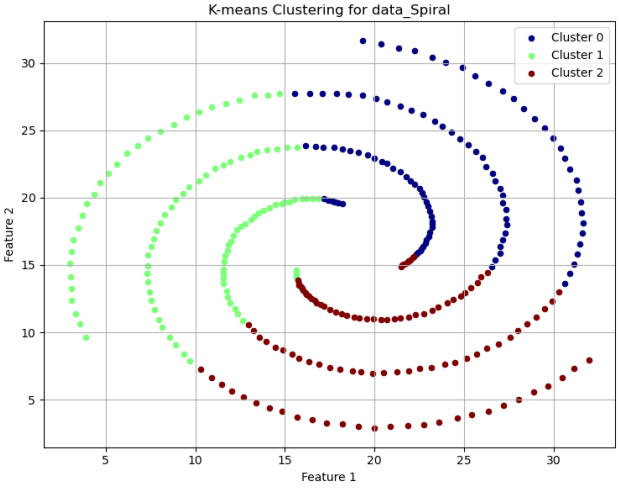

Spectral Clustering for data_Compound
Feature 1
Feature 2
Cluster 0
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5

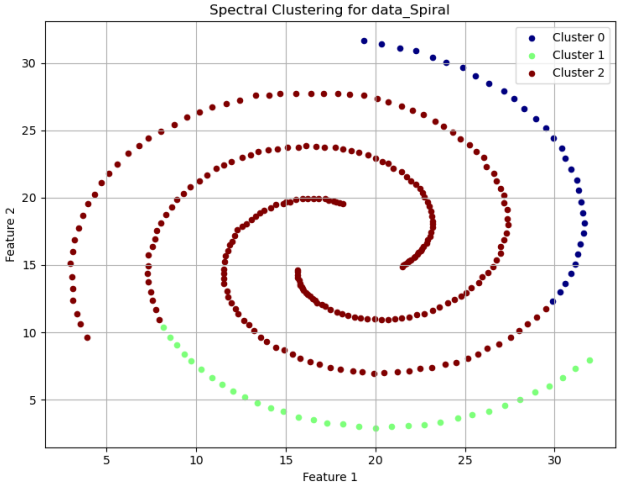

K-means Clustering for data_Flame
Cluster 0
Cluster 1
Feature 1
Feature 2

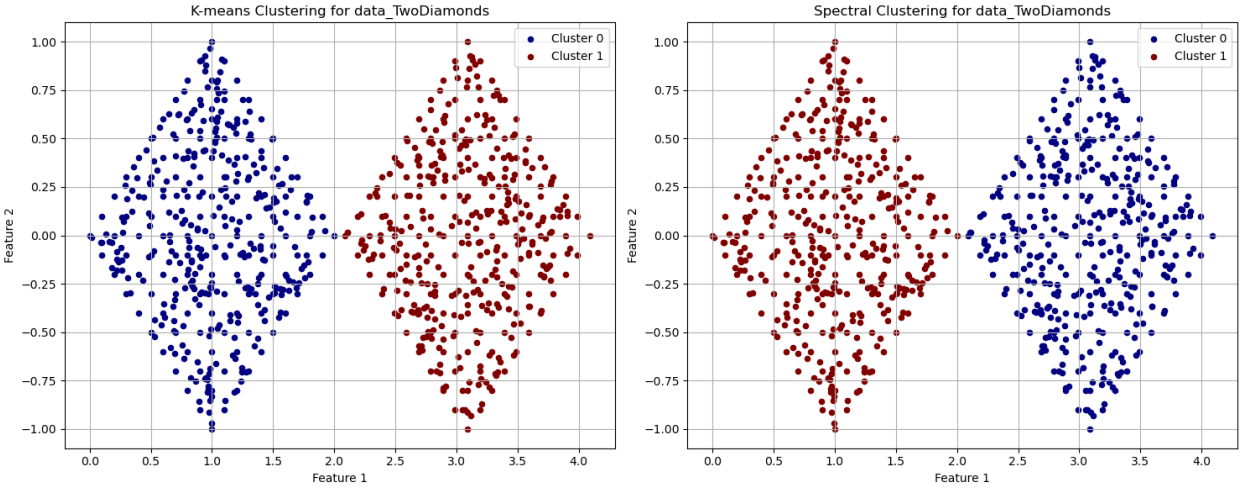

Spectral Clustering for data_Flame
Cluster 0
Cluster 1
Feature 1
Feature 2


K-means Clustering for data_Jain
Cluster 0
Cluster 1
Feature 1
Feature 2


Spectral Clustering for data_Jain
Cluster 0
Cluster 1
Feature 1
Feature 2


K-means Clustering for data_Spiral
Cluster 0
Cluster 1
Cluster 2
Feature 1
Feature 2


Spectral Clustering for data_Spiral
Cluster 0
Cluster 1
Cluster 2
Feature 1
Feature 2




```
({'data_Aggregation.mat': array([5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6,
        6, 6, 6, 6, 6, 3, 3, 3, 3, 3, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6,
        6, 6, 3, 3, 3, 3, 3, 3, 6, 6, 6, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3,
        3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3,
        3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
4, 3,
        3, 4, 4, 3, 3, 3, 3, 3, 3, 3, 3, 0, 0, 0, 0, 0, 3, 3, 3, 0,
0, 0,
        0, 0, 0, 0, 0, 3, 3, 4, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 3, 0,
4, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 4,
4, 4,
        4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
4, 4,
        4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
4, 4,
        4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
4, 4,
```

```
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
4, 4,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
4, 4,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
4, 4,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
4, 4,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 2, 2, 2, 2, 2,
2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 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, 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, 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, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1,
       1, 1, 1, 1, 1, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
5, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
5, 5,
       5, 5, 5, 5, 5, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]),
 'data_Bridge.mat': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 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, 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,
        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, 1, 1, 1, 1, 1, 1,
1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]),
 'data_Compound.mat': array([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 1, 2, 2,
        2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 2, 2, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2,
        2, 2, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3,
        3, 3, 4, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 4, 3,
3, 3,
        3, 3, 3, 3, 4, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3,
        3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3,
        3, 3, 1, 1, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
4, 4,
        4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
4, 4,
        4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 4, 5, 5, 5, 5,
5, 5,
        5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
5, 5,
        5, 5, 5, 5, 5, 5, 5, 4, 4, 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, 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, 5, 5, 1, 1, 1, 1, 1, 1, 1, 1, 1, 5, 5, 5,
5, 5,
        5, 5, 5]),
 'data_Flame.mat': array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 0, 0, 0, 1, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
0, 0,
        0, 0, 1, 0, 0, 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, 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, 1, 1, 1, 1, 1, 1, 1, 1, 1]),
  'data_Jain.mat': 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, 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, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1,
        1, 1, 1, 1, 0, 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, 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, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 0,
        1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0]),
  'data_Spiral.mat': array([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2,
        2, 2, 2, 2, 2, 2, 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, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 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, 0, 0, 0, 0, 0, 0, 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, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2,
        1, 1, 1, 1]),
  'data_TwoDiamonds.mat': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 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, 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, 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, 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, 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, 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,
        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, 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, 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,
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, 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, 1, 1, 1, 1])},
 {'data_Aggregation.mat': array([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
         2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2,
         2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2,
         2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2,
         2, 2, 2, 5, 5, 2, 2, 2, 2, 2, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
5, 5,
         5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2,
         2, 2, 2, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
5, 5,
         5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 3, 3, 3, 3,
3, 3,
         3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3,
         3, 3, 3, 3, 3, 3, 3, 3, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6,
         6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6,
         6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6,
         6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6,
         6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6,
         6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6,
         6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6,
         6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6,
         6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6,
         6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6,
         6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6,
```

```
         6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 0, 0, 0, 0, 0,
0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1,
         1, 1, 4, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1,
         1, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
4, 4,
         4, 4, 4, 4, 4, 1, 1, 1, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
4, 4,
         4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
4, 4,
         4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
4, 4,
         4, 4, 4, 4, 4, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6,
         6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6,
         6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6,
         6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6]),
  'data_Bridge.mat': 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, 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, 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, 0, 0, 0, 0, 0,
0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
```

```
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]),
  'data_Compound.mat': array([4, 4, 3, 4, 4, 4, 4, 3, 4, 3, 3, 3, 3,
3, 3, 3, 3, 0, 0, 0, 0, 0,
          5, 3, 3, 3, 3, 3, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 0, 3, 0, 0,
0, 3,
          3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3,
          3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3,
          3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3,
          3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3,
          3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 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, 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, 1, 1, 1, 1,
1, 1,
          1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2,
          2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2,
          2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2,
          2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2,
          2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2,
          2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2,
          2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2,
          2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2,
          2, 2, 2]),
  'data_Flame.mat': array([0, 0, 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, 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, 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,
          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, 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, 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,
1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]),
  'data_Jain.mat': 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, 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, 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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0]),
  'data_Spiral.mat': 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, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
```

```
2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2,
        2, 2, 2, 2]),
  'data_TwoDiamonds.mat': 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, 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, 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,
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, 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, 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, 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, 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, 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, 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, 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,
            1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
            0, 0, 0, 0, 0, 0, 0, 0])})
```

On the toy dataset, Spectral Clustering outperforms K-means in capturing intricate data correlations with complex, non-globular clusters. However, it requires a lot of processing power and its performance depends on choosing the right sigma values. K-means is simpler and faster, but it assumes spherical clusters, which can be inaccurate for datasets like "data_Spiral.mat." Inconsistent outcomes may also result from its centroid initialization. K-means is successful and efficient for clear, spherical clusters, however Spectral Clustering could be favored for complex data structures despite its overheads. The specific dataset structure will therefore determine the best option.