3) Generating GMMs

The advantage of using GMMs is it works efficiently especially when the clusters in data are in gaussian distribution. We solve this problem to observe how efficient GMM is in dealing data having gaussian distribution clusters.

For example, this can be a dataset with heights, weights of 3 different animals (dogs, cows, fox). In the case of unsupervised learning, we use clustering techniques to label them. Choosing the right clustering technique is important, in this problem we will find which model performs the best while handling spherical Gaussians between GMM and Kmeans.

Performance metrics :

I am using Adjusted Rand score and normalized_mutual_info_score metrics to compare the model results. The higher these scores are the better the model performance is.

1. Dataset with a mixture of Gaussians (where the covariance matrix is the identity matrix times some positive scalar).

```
#importing required libraries ------------------------------------------------------------
import numpy as np #we use numpy for generating data and other mathematical operations.
import matplotlib.pyplot as plt # for plotting
from sklearn.cluster import KMeans #we are using kmeans from sklearn
from sklearn.mixture import GaussianMixture # we are using Gaussian mixture from sklearn
# I am using Adjusted Rand score and normalized_mutual_info_score metrics to compare the model results.
from sklearn.metrics import adjusted_rand_score, normalized_mutual_info_score


#dataset creation ------------------------------------------------------------------------
# Generating 3 spherical Gaussians (where the covariance matrix is the identity matrix times some positive scalar)
np.random.seed(42)

#Note: When I set the means of clusters too far from each other, There is no overlapped data points, and this will not help us in
#finding which model is better. So I have chosen the closer cluster centres (means) for better performance analysis
# Means for the 3 clusters
m1 = [0, 0] #means generally tell the centers of spherical gaussian distribution
m2 = [3, 3]
m3 = [-2, 2]

# Covariance matrices multiplied by scalars for 3 clusters
#np.eye(2) basically generates identity matrix with dimensions [2,2]. When multiplied by scalar, it explains the spreading
#factor of gaussian distribution
cov1 = np.eye(2) * 0.4  # This means the first Gaussian has equal variance (0.4) in both dimensions.
cov2 = np.eye(2) * 0.9  # 0.9 spread in both the dimensions
cov3 = np.eye(2) * 1.3  # 1.3 spread in both the dimensions

# Generate 100 points for each cluster (we can say 100 samples of dogs, cows and fox based on our example)
x1 = np.random.multivariate_normal(m1, cov1, 100)
x2 = np.random.multivariate_normal(m2, cov2, 100)
x3 = np.random.multivariate_normal(m3, cov3, 100)
# I choose 100 because, when I set the number to less, there are very few overlapped, and this will not help us in
#finding which model is better, so I choose 100.
```

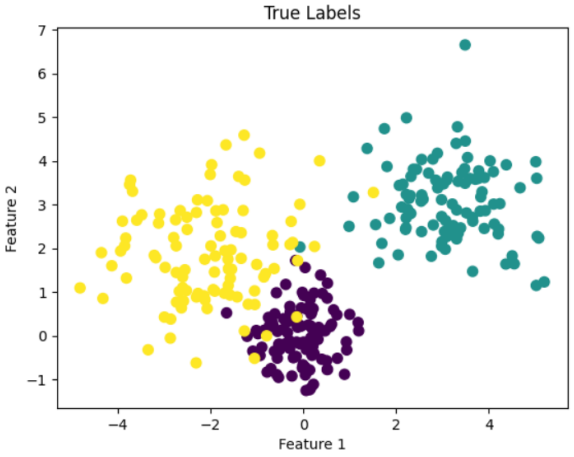
```
# Combine the three clusters into one dataset (from the example, this can be heights and weights(coordiates of data points)
# of all three animals stacking into one column )
X = np.vstack([x1, x2, x3])

#So, X has whole data of the clusters combined,
#Model fitting ----------------------------------------------------------------------------------
#now we perform K-means and GMM.

# K-Means clustering
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans_labels = kmeans.fit_predict(X)

# Gaussian Mixture Model with spherical covariance
gmm = GaussianMixture(n_components=3, covariance_type='spherical', random_state=42)
gmm_labels = gmm.fit_predict(X)

#Performance metrics and analysis ---------------------------------------------------------------
# Compute clustering performance metrics

# Creating the labels, so it can be evaluated with the predicted labels.
#
true_labels = np.array([0]*100 + [1]*100 + [2]*100) #from example, 0 can be dog, 1 can be fox and 2 can be cow.

#I am using two metrics

# a. Adjusted Rand Index
ari_kmeans = adjusted_rand_score(true_labels, kmeans_labels)
ari_gmm = adjusted_rand_score(true_labels, gmm_labels)
# b. Normalized Mutual Information
nmi_kmeans = normalized_mutual_info_score(true_labels, kmeans_labels)
nmi_gmm = normalized_mutual_info_score(true_labels, gmm_labels)
```

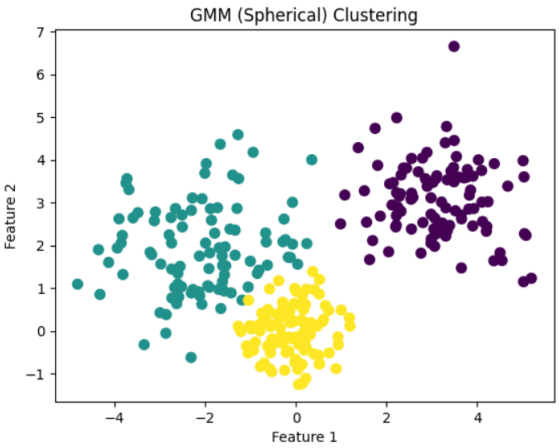
```
# Print Kmeans results
print("K-Means Metrics:")
print(f"ARI: {ari_kmeans}")
print(f"NMI: {nmi_kmeans}")
# Printing GMM results
print("\nGMM Metrics:")
print(f"ARI: {ari_gmm}")
print(f"NMI: {nmi_gmm}")
```

```
K-Means Metrics:
ARI: 0.8236848491385121
NMI: 0.8153563543886516

GMM Metrics:
ARI: 0.9032594416137354
NMI: 0.8649897028119647
```

We can observe both ARI and NMI metrics are higher for GMM, suggesting GMM has better performance in data set where a mixture of 3 spherical Gaussians (where the covariance matrix is the identity matrix times some positive scalar) compare to K-means.

```python
#Plotting the results ------------------------------------------------------------------------
#since i am plotting 3 graphs, original data, k-means cluster result, gmm cluster result,
# i will write the function for plotting and use the same for 3 plots.
#
def plot_clusters(X, labels, title):
    plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', s=50)
    plt.title(title)
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.show()

# Plotting true labels
plot_clusters(X, true_labels, 'True Labels') # from example this gives the plot of heights,
                                             #weights (coordinates of data points) of all 3 animals.

# Plotting K-means cluster result
plot_clusters(X, kmeans_labels, 'K-Means Clustering')

# Plotting GMM cluster result
plot_clusters(X, gmm_labels, 'GMM (Spherical) Clustering')
```





Firstly I observe the clusters in spherical shape, this is due to equal variance in both features (same diagonal elements in covariance matrix of dataset). These plots supports the results of metrics and explains GMM is able to handle this data well compare to K-means.

2. Here we need to construct a data set where a mixture of 3 diagonal Gaussians (where the covariance matrix can have non-zero values on the diagonal, and zeros elsewhere).

Solution.

Instead of having equal spreading by common scalar, the data will be spread in unequal dimension in 2 D space. In the code, I just modified the covariance matrix of data, and added one more model into the code from 1st problem, remaining I put the same.

So, we will evaluate three models. Gaussian mixture with spherical covariance type, Gaussian mixture with diagonal covariance type and K-means. Let's find out how GMMs and K-means work in this case

```
#3.2
#importing required libraries ------------------------------------------------------------
import numpy as np #we use numpy for generating data and other mathematical operations.
import matplotlib.pyplot as plt # for plotting
from sklearn.cluster import KMeans #we are using kmeans from sklearn
from sklearn.mixture import GaussianMixture # we are using Gaussian mixture from sklearn
# I am using Adjusted Rand score and normalized_mutual_info_score metrics to compare the model results.
from sklearn.metrics import adjusted_rand_score, normalized_mutual_info_score


#dataset creation ------------------------------------------------------------------------
# Generating 3 spherical Gaussians (where the covariance matrix can have non-zero values on the diagonal, and zeros elsewhere)
np.random.seed(42)

#Note: When I set the means of clusters too far from each other, There is no overlapped data points, and this will not help us in
#finding which model is better. So I have chosen the closer cluster centres (means) for better performance analysis
# Means for the 3 clusters
m1 = [1, 4] #means generally tell the centers of spherical gaussian distribution
m2 = [5, 5]
m3 = [-2, 2]

# Covariance matrices having non zero values on the diagonal and zeros elsewhere
cov1 = [[1, 0], [0, 6]]  # Wider spread along y-axis
cov2 = [[4, 0], [0, 2 ]]  # Wider spread along x-axis
cov3 = [[3, 0], [0, 5]]  # Wider spread along y-axis
```

```
# Generate 100 points for each cluster (we can say 100 samples of dogs, cows and fox based on our example)
x1 = np.random.multivariate_normal(m1, cov1, 100)
x2 = np.random.multivariate_normal(m2, cov2, 100)
x3 = np.random.multivariate_normal(m3, cov3, 100)
# I choose 100 because, when I set the number to less, there are very few overlapped, and this will not help us in
#finding which model is better, so I choose 100.
# Combine the three clusters into one dataset (from the example, this can be heights and weights(coordiates of data points)
# of all three animals stacking into one column )
X = np.vstack([x1, x2, x3])

#So, X has whole data of the clusters combined,
#Model fitting ----------------------------------------------------------------------------------
#now we perform K-means and GMM.

# K-Means clustering
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans_labels = kmeans.fit_predict(X)
# Gaussian Mixture Model with spherical covariance
gmm = GaussianMixture(n_components=3, covariance_type='spherical', random_state=42)
gmm_labels = gmm.fit_predict(X)
#Gaussian Mixture Model with diagonal covariance
gmm_diag = GaussianMixture(n_components=3, covariance_type='diag', random_state=42)
gmm_diag_labels = gmm_diag.fit_predict(X)
```

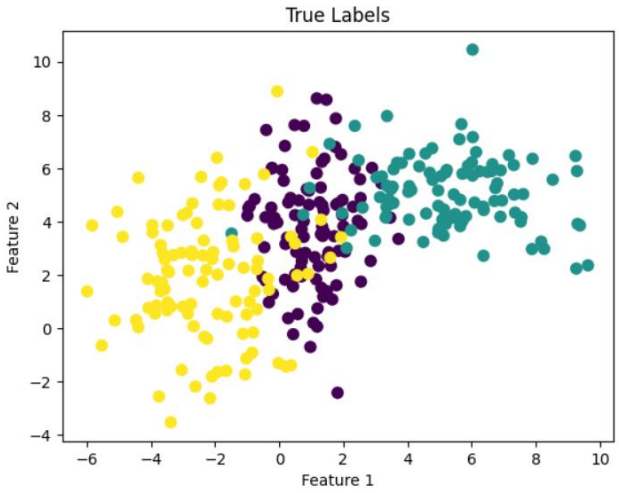
```python
#Performance metrics and analysis ----------------------------------------------------------------------
# Creating the labels, so it can be evaluated with the predicted labels.
true_labels = np.array([0]*100 + [1]*100 + [2]*100) #from example, 0 can be dog, 1 can be fox and 2 can be cow.

#I am using two metrics
# a. Adjusted Rand Index
ari_kmeans = adjusted_rand_score(true_labels, kmeans_labels)
ari_gmm = adjusted_rand_score(true_labels, gmm_labels)
ari_gmm_diag = adjusted_rand_score(true_labels, gmm_diag_labels)
# b. Normalized Mutual Information
nmi_kmeans = normalized_mutual_info_score(true_labels, kmeans_labels)
nmi_gmm = normalized_mutual_info_score(true_labels, gmm_labels)
nmi_gmm_diag = normalized_mutual_info_score(true_labels, gmm_diag_labels)

# Print Kmeans results
print("K-Means Metrics:")
print(f"ARI: {ari_kmeans}")
print(f"NMI: {nmi_kmeans}")
# Printing GMM spherical covariance results
print("\nGMM sphercial Metrics:")
print(f"ARI: {ari_gmm}")
print(f"NMI: {nmi_gmm}")
# Printing GMM diagonal covariance results
print("\nGMM diag Metrics:")
print(f"ARI: {ari_gmm_diag}")
print(f"NMI: {nmi_gmm_diag}")
```

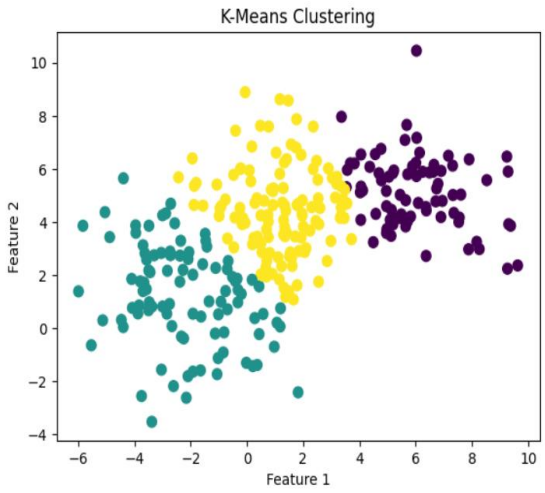
```
K-Means Metrics:
ARI: 0.5316390890023596
NMI: 0.5496583650093707

GMM sphercial Metrics:
ARI: 0.49817285357729907
NMI: 0.5262542429803935

GMM diag Metrics:
ARI: 0.6929010129625742
NMI: 0.6409464082872034
```

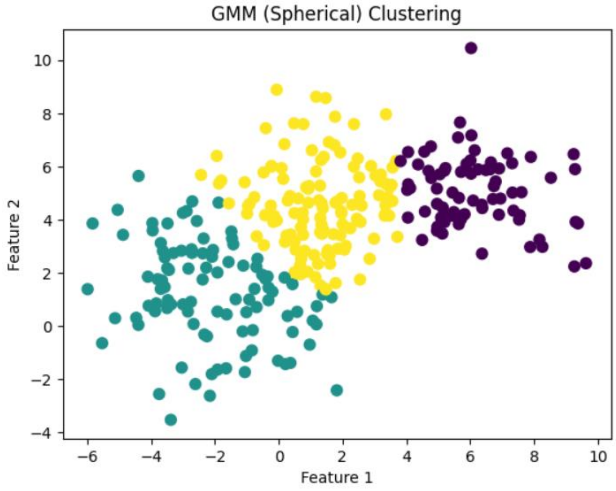
Here the underperformance of Gaussian mixture with spherical covariance type is because it assumes the clusters to have spherical data whereas the data is more elliptical in shape. But here we set different diagonal values in covariance matrix indicating it is not in spherical shapes, hence Gaussian with diagonal type covariance matrix has given the best results with ARI = 0.6929, NMI = 0.6409.

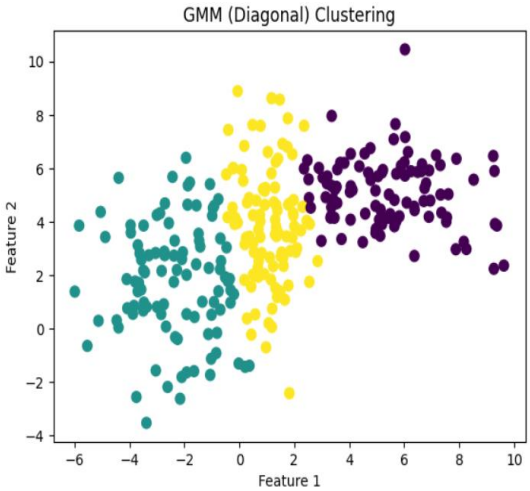
.

```python
#Plotting the results ----------------------------------------------------------------
#since i am plotting 4 graphs, original data, k-means cluster result, gmm spherical cluster result, gmm diagonal cluster.
# i will write the function for plotting and use the same for 3 plots.
#
def plot_clusters(X, labels, title):
    plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', s=50)
    plt.title(title)
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.show()

# Plotting true labels
plot_clusters(X, true_labels, 'True Labels') # from example this gives the plot of heights,
                                             #weights (coordinates of data points) of all 3 animals.

# Plotting K-means cluster result
plot_clusters(X, kmeans_labels, 'K-Means Clustering')

# Plotting GMM cluster result
plot_clusters(X, gmm_labels, 'GMM (Spherical) Clustering')

# Plotting GMM cluster result
plot_clusters(X, gmm_diag_labels, 'GMM (Diagonal) Clustering')
```









Its very evident from plots that the GMM(Diagonal) able to fit the data most efficiently by capturing the elliptical shapes of clusters and the plots show that

3.3 Here we need to construct data set where a mixture of 3 Gaussians with unrestricted covariance matrices and fit with k-means, Diagonal Gaussian mixture and Fully Gaussian mixture.

With non zero values in all the elements of covariance matrix, we can observe a correlation between x and y features that is the cluster will no longer be aligned to and x and y axis, instead there will be a presence of correlation between them. So, I am using the same code, but changing the covariance matrix values and adding Fully Gaussian mixture model and compare it with diagonal GMM and K-means.

```
#3.3
#importing required libraries ------------------------------------------------------------
import numpy as np #we use numpy for generating data and other mathematical operations.
import matplotlib.pyplot as plt # for plotting
from sklearn.cluster import KMeans #we are using kmeans from sklearn
from sklearn.mixture import GaussianMixture # we are using Gaussian mixture from sklearn
# I am using Adjusted Rand score and normalized_mutual_info_score metrics to compare the model results.
from sklearn.metrics import adjusted_rand_score, normalized_mutual_info_score


#dataset creation ------------------------------------------------------------------------
# Generating 3 spherical Gaussians (where a mixture of 3 Gaussians with unrestricted covariance matrices)
np.random.seed(42)

#Note: When I set the means of clusters too far from each other, There is no overlapped data points, and this will not help us in
#finding which model is better. So I have chosen the closer cluster centres (means) for better performance analysis
# Means for the 3 clusters
m1 = [1, 8] #means generally tell the centers of spherical gaussian distribution
m2 = [5, 5]
m3 = [-2, 2]

# Covariance matrices with off-diagonal values (representing correlation between features)
cov1 = [[3, 2], [2, 5]]  # Correlated features
cov2 = [[7, -3], [-3, 4]]  # Negative correlation
cov3 = [[5, 1.5], [1.5, 3]]  # Correlated features
```

```
# Generate 100 points for each cluster (we can say 100 samples of dogs, cows and fox based on our example)
x1 = np.random.multivariate_normal(m1, cov1, 100)
x2 = np.random.multivariate_normal(m2, cov2, 100)
x3 = np.random.multivariate_normal(m3, cov3, 100)
# I choose 100 because, when I set the number to less, there are very few overlapped, and this will not help us in
#finding which model is better, so I choose 100.
# Combine the three clusters into one dataset (from the example, this can be heights and weights(coordiates of data points)
# of all three animals stacking into one column )
X = np.vstack([x1, x2, x3])

#So, X has whole data of the clusters combined,
#Model fitting ----------------------------------------------------------------------------------
#now we perform K-means and GMM.

# K-Means clustering
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans_labels = kmeans.fit_predict(X)
#Gaussian Mixture Model with diagonal covariance
gmm_diag = GaussianMixture(n_components=3, covariance_type='diag', random_state=42)
gmm_diag_labels = gmm_diag.fit_predict(X)
# Gaussian Mixture Model with full covariance
gmm_full = GaussianMixture(n_components=3, covariance_type='full', random_state=42)
gmm_full_labels = gmm_full.fit_predict(X)
```

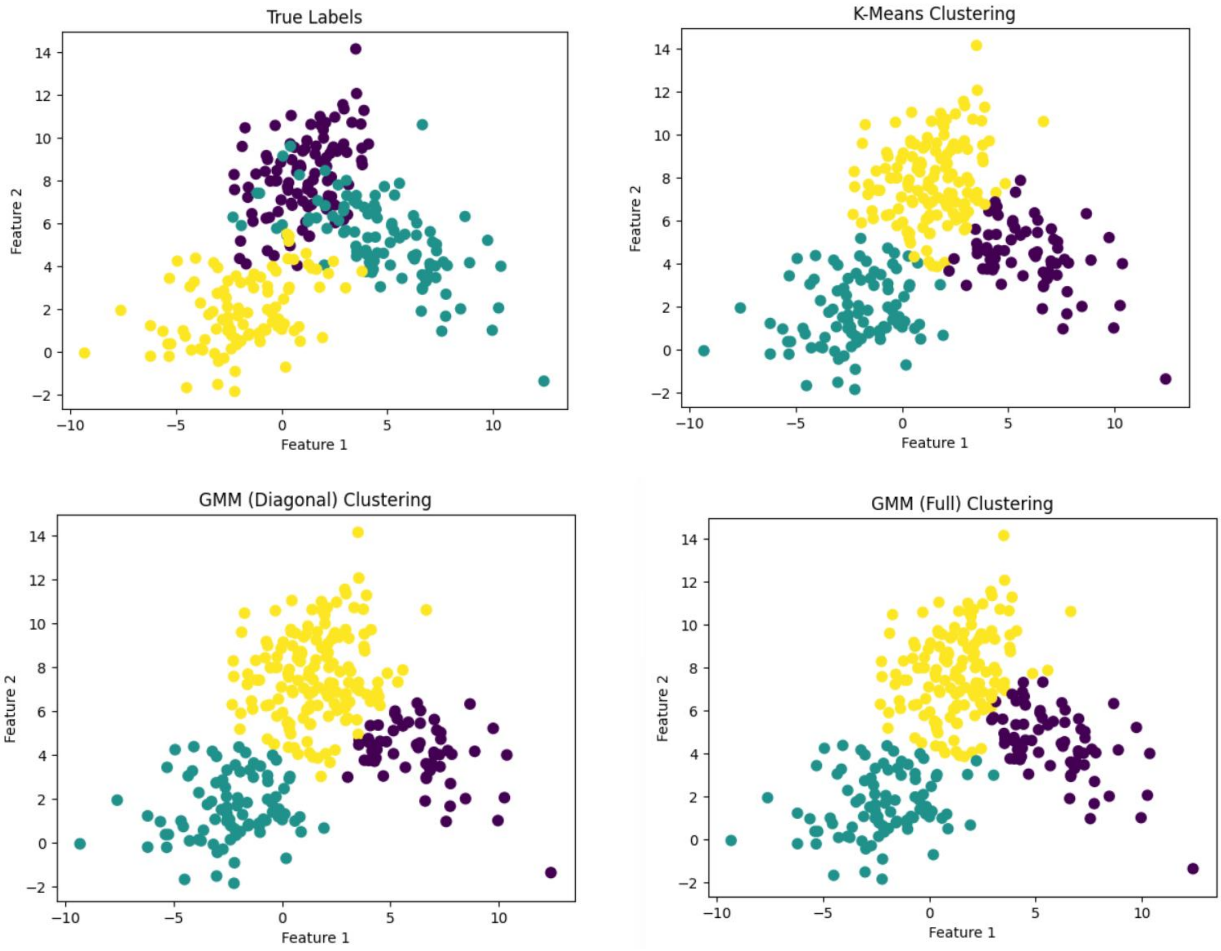
```python
#Performance metrics and analysis ----------------------------------------------------------
# Creating the labels, so it can be evaluated with the predicted labels.
true_labels = np.array([0]*100 + [1]*100 + [2]*100) #from example, 0 can be dog, 1 can be fox and 2 can be cow.

#I am using two metrics
# a. Adjusted Rand Index
ari_kmeans = adjusted_rand_score(true_labels, kmeans_labels)
ari_gmm_full = adjusted_rand_score(true_labels, gmm_full_labels)
ari_gmm_diag = adjusted_rand_score(true_labels, gmm_diag_labels)
# b. Normalized Mutual Information
nmi_kmeans = normalized_mutual_info_score(true_labels, kmeans_labels)
nmi_gmm_diag = normalized_mutual_info_score(true_labels, gmm_diag_labels)
nmi_gmm_full = normalized_mutual_info_score(true_labels, gmm_full_labels)

# Print metrics for comparison
print("K-Means Metrics:")
print(f"ARI: {ari_kmeans}")
print(f"NMI: {nmi_kmeans}")

print("\nGMM (Diagonal) Metrics:")
print(f"ARI: {ari_gmm_diag}")
print(f"NMI: {nmi_gmm_diag}")

print("\nGMM (Full) Metrics:")
print(f"ARI: {ari_gmm_full}")
print(f"NMI: {nmi_gmm_full}")
```

```
K-Means Metrics:
ARI: 0.5599509377277903
NMI: 0.5783329524263763

GMM (Diagonal) Metrics:
ARI: 0.5176906166539922
NMI: 0.5765957931151816

GMM (Full) Metrics:
ARI: 0.604013246674675
NMI: 0.6261055082994317
```

As expected, From the metrics we observe Fully GMM has given the best performance compare to other two models, because Fully GMM assumes a correlation between the features which allows it to fit well with the data and clusters efficiently.

Lets look at the plot:

```python
# Plotting the results ----------------------------------------------------------------------
def plot_clusters(X, labels, title):
    plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', s=50)
    plt.title(title)
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.show()

# Plotting true labels
plot_clusters(X, true_labels, 'True Labels')

# Plotting K-means cluster result
plot_clusters(X, kmeans_labels, 'K-Means Clustering')

# Plotting GMM with diagonal covariance result
plot_clusters(X, gmm_diag_labels, 'GMM (Diagonal) Clustering')

# Plotting GMM with full covariance result
plot_clusters(X, gmm_full_labels, 'GMM (Full) Clustering')
```



Firstly we observe the clusters not aligned with x, y axis indicating there is correlation between x and y. When the non diagonal elements in covariance matrix are positive, there is positive correlation, and negative correlation when the non diagonal elements are negative. It looks like Fully GMM > K-means> Diagonal GMM.

4. Implementing K-Means and Spectral Clustering.

Before performing clustering algorithms lets **understand the given dataset** and transform the labels, data points into dataframe to ease the clustering.

Read the data using loadmat:

```
#Lets understand the data before applying clustering algorithms

import scipy.io
from my_kmeans import my_kmeans
from my_spectralclustering import my_spectralclustering
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
# Loading the 7.m files
m1 = scipy.io.loadmat('D:/Projects/assignments/toydata/cluster/data_Aggregation')
m2 = scipy.io.loadmat('D:/Projects/assignments/toydata/cluster/data_Bridge')
m3 = scipy.io.loadmat('D:/Projects/assignments/toydata/cluster/data_Compound')
m4 = scipy.io.loadmat('D:/Projects/assignments/toydata/cluster/data_Flame')
m5 = scipy.io.loadmat('D:/Projects/assignments/toydata/cluster/data_Jain')
m6 = scipy.io.loadmat('D:/Projects/assignments/toydata/cluster/data_Spiral')
m7 = scipy.io.loadmat('D:/Projects/assignments/toydata/cluster/data_TwoDiamonds')
```

```
#finding out the column headers
print("The column headers are:")
print(m1.keys(), "\n", m2.keys(),"\n",m3.keys(),"\n", m4.keys(),
    "\n", m5.keys(), "\n", m6.keys(), "\n", m7.keys())
```

```
The column headers are:
dict_keys(['__header__', '__version__', '__globals__', 'L', 'D'])
 dict_keys(['__header__', '__version__', '__globals__', 'L', 'D'])
 dict_keys(['__header__', '__version__', '__globals__', 'L', 'D'])
 dict_keys(['__header__', '__version__', '__globals__', 'D', 'L'])
 dict_keys(['__header__', '__version__', '__globals__', 'D', 'L'])
 dict_keys(['__header__', '__version__', '__globals__', 'D', 'L'])
 dict_keys(['__header__', '__version__', '__globals__', 'L', 'D'])
```

lets transform our data from m files into dataframe from the question we know ”D” is the data matrix and ”L” is the ground truth label matrix. So we will save d in x variable, and l in y variable(target or label)

```
#Labels:
y_m1, y_m2, y_m3, y_m4, y_m5, y_m6, y_m7 = m1['L'], m2['L'], m3['L'], m4['L'], m5['L'], m6['L'], m7['L']

#Data points:
x_m1, x_m2, x_m3, x_m4, x_m5, x_m6, x_m7 = m1['D'], m2['D'], m3['D'], m4['D'], m5['D'], m6['D'], m7['D']

#Lets print the data together for one cluster file
print("Data in m1 file:", "\n")
print(x_m1, y_m1)
```

```
 Data in m1 file:

 [[15.55 28.65]
  [14.9  27.55]
  [14.45 28.35]
  ...
  [ 8.5   3.25]
  [ 8.1   3.55]
  [ 8.15  4.  ]] [[2]
  [2]
  [2]
  [2]
  [2]
  [2]
  [2]
```

```
#---------y[labels]--------------
y_m1, y_m2, y_m3, y_m4, y_m5, y_m6, y_m7 = pd.DataFrame(y_m1), pd.DataFrame(y_m2), pd.DataFrame(y_m3), pd.DataFrame(y_m4), pd.DataFrame(y_m5), pd.DataFrame(y_m6),
#------------x[Data points]-----------------------------
x_m1, x_m2, x_m3, x_m4, x_m5, x_m6, x_m7 = pd.DataFrame(x_m1), pd.DataFrame(x_m2), pd.DataFrame(x_m3), pd.DataFrame(x_m4), pd.DataFrame(x_m5), pd.DataFrame(x_m6),


#take a preview of the dataset
print("cluster 1 data \n", x_m1.head())
print(x_m1.info(), "\n", y_m1.info())
```

Preview of x1[datapoints of a]

```
 cluster 1 data
          0      1
 0  15.55  28.65
 1  14.90  27.55
 2  14.45  28.35
 3  14.15  28.80
 4  13.75  28.05
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 788 entries, 0 to 787
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   0       788 non-null    float64
 1   1       788 non-null    float64
dtypes: float64(2)
memory usage: 12.4 KB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 788 entries, 0 to 787
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   0       788 non-null    uint8
```

In (a) data set there 788 samples in total.

Lets see number of labels in each data set:

```
print(y_m1[0].unique(),  "\n", y_m2[0].unique(),  "\n", y_m3[0].unique(),
     "\n", y_m4[0].unique() ,  "\n", y_m5[0].unique(),  "\n", y_m6[0].unique(),
     "\n", y_m7[0].unique() )
```

```
[2 7 4 3 6 1 5]
 [3 6]
 [1 2 3 4 5 6]
 [1 2]
 [2 1]
 [3 1 2]
 [1 2]
```

1.

Defining K-means:

```python
# defining my_kmeans function:
import numpy as np


def my_kmeans(data, K):  2 usages
    # Initializing cluster centroids by randomly selecting K data points
    count = 0
    np.random.seed(42)
    data = data.to_numpy()
    # choosing random points as centroids in k means.
    centroids_indices = np.random.choice(len(data), K, replace=False)

    centroids = data[centroids_indices]

    while True:
        # Assigning each data point to the nearest centroid
        labels = np.argmin(np.linalg.norm(data[:, np.newaxis] - centroids, axis=2), axis=1) + 1  # Add 1 to shift labels

        # Calculating new centroids as the mean of data points in each cluster
        new_centroids = np.array([data[labels == k].mean(axis=0) for k in range(1, K + 1)])  # Start from 1

        # Checking for convergence
        if np.all(centroids == new_centroids):
            break

        centroids = new_centroids
        count = count + 1  # the final count will be the number of iterations.
    return labels, count
```

To meet the condition of iterating till convergence I set the function to break only when new centroids same as previous iteration centroids. The function returns the predicted labels and number of iterations it took (count). The function for k-means is saved in my_kmeans.py

Lets explore the given data and perform the kmeans clustering technique in Run_clustering.py:

I am defining function to perform the k-means on all the 7 datasets, instead of rewriting every time the arguments for function will be input data(for ex: x_m1), labels(for ex: y_m1), k and my_kmeans.

```python
def kmeans_clustering(data, true_labels, K, kmeans_clus):
    # Calling the provided kmeans function
    cluster_labels, iteration = kmeans_clus(data, K)
    # Creating a DataFrame for visualization
    df = pd.DataFrame(data)
    df['true_labels'] = true_labels
    df['cluster_labels'] = cluster_labels
    # Scatter plot for actual clusters
    plt.figure(figsize=(12, 5))
    plt.subplot( *args: 1, 2, 1)
    plt.scatter(df[0], df[1], c=df['true_labels'], cmap='viridis')
    plt.title('Actual Clusters')
    plt.xlabel('x-axis')
    plt.ylabel('y-axis')
    # Scatter plot for predicted clusters (K-means)
    plt.subplot( *args: 1, 2, 2)
    plt.scatter(df[0], df[1], c=df['cluster_labels'], cmap='viridis')
    plt.title('Predicted Clusters (K-means)')
    plt.xlabel('x-axis')
    plt.ylabel('y-axis')
    plt.tight_layout()
    plt.show()
    # Print the cluster labels and the number of iterations
    print(cluster_labels)
    print("Number of iterations:", iteration)
```

```python
# (a) Aggregation (K=7)
K = 7
kmeans_clustering(x_m1, y_m1, K, my_kmeans)
# (b) Bridge (K=2)
K = 2
kmeans_clustering(x_m2, y_m2, K, my_kmeans)
# (c) Compound (K=6)
K = 6
kmeans_clustering(x_m3, y_m3, K, my_kmeans)
# (d) Flame (K=2)
K = 2
kmeans_clustering(x_m4, y_m4, K, my_kmeans)
# (e) Jain (K=2)
K = 2
kmeans_clustering(x_m5, y_m5, K, my_kmeans)
# (f) Spiral (K=3)
K = 3
kmeans_clustering(x_m6, y_m6, K, my_kmeans)
# (g) TwoDiamond (K=2)
K = 2
kmeans_clustering(x_m7, y_m7, K, my_kmeans)
```

a)

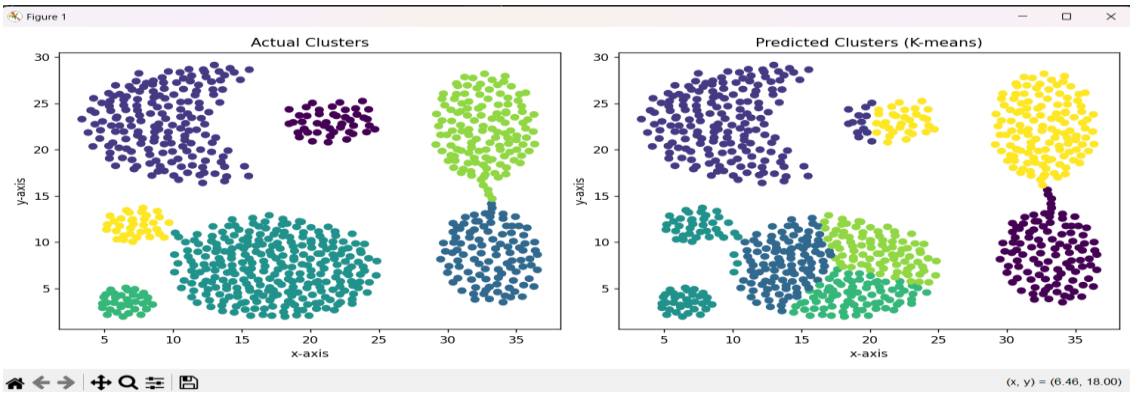


Figure 1
Actual Clusters
Predicted Clusters (K-means)
y-axis
x-axis
(x, y) = (6.46, 18.00)

b)

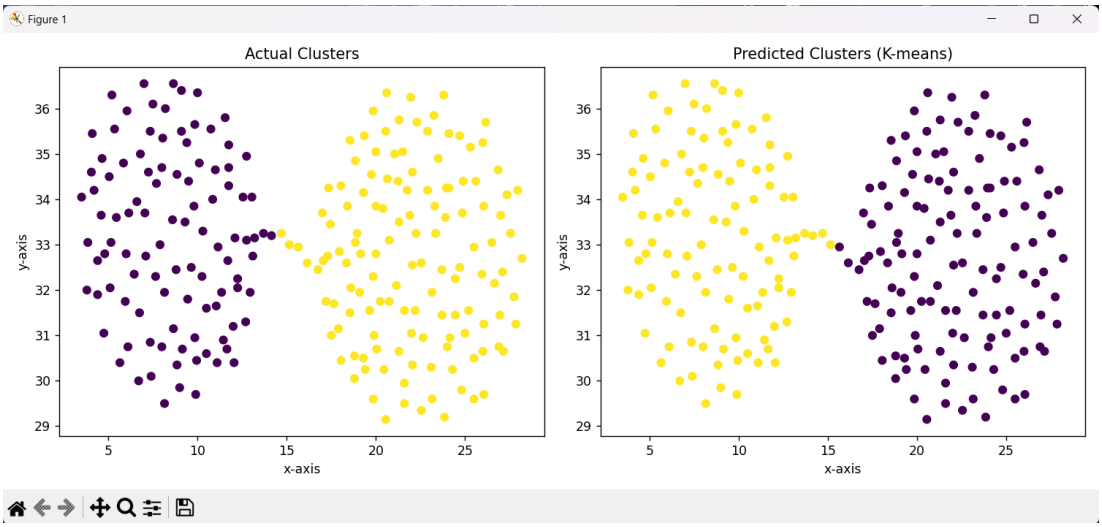


Figure 1
Actual Clusters
Predicted Clusters (K-means)
y-axis
x-axis

c)

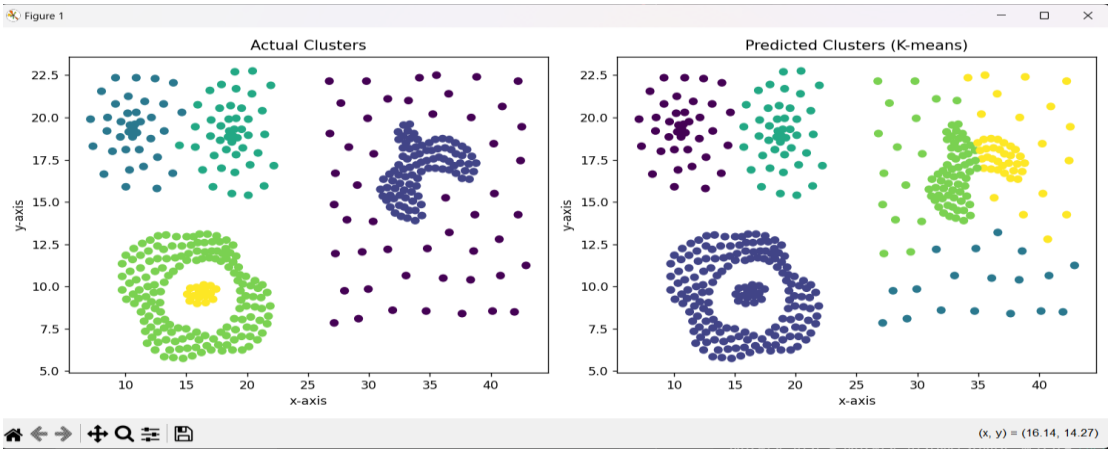


Figure 1
Actual Clusters
Predicted Clusters (K-means)
y-axis
x-axis
(x, y) = (16.14, 14.27)

d)

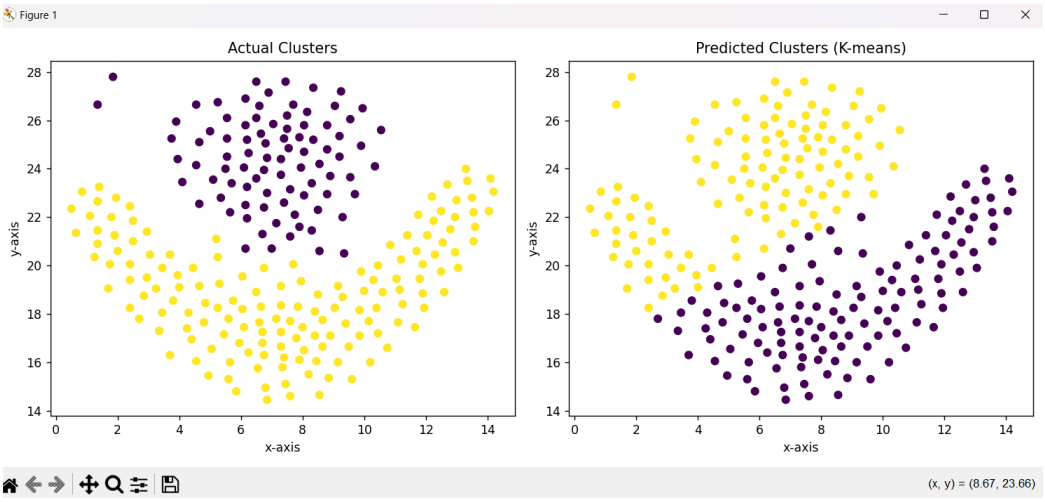


Figure 1
Actual Clusters
Predicted Clusters (K-means)
x-axis
y-axis
(x, y) = (8.67, 23.66)

e)

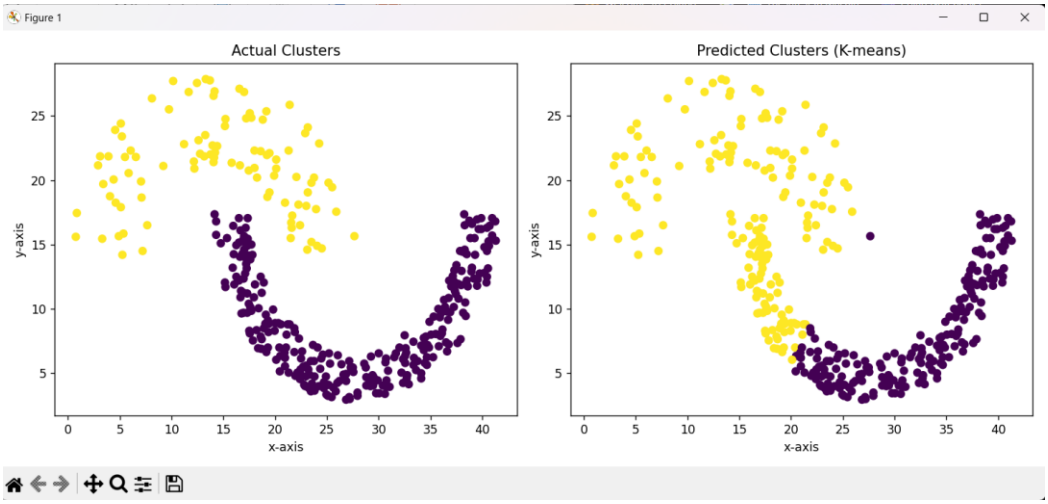


Figure 1
Actual Clusters
Predicted Clusters (K-means)
x-axis
y-axis

f)

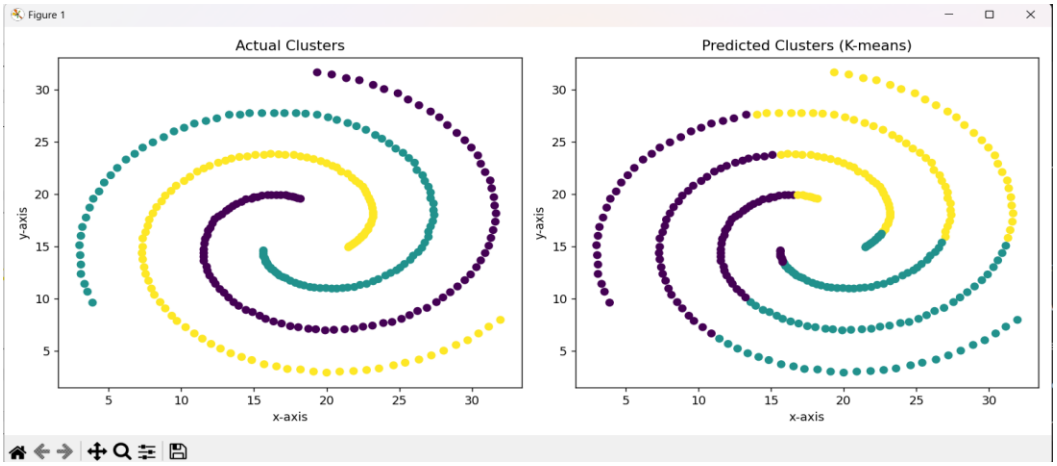
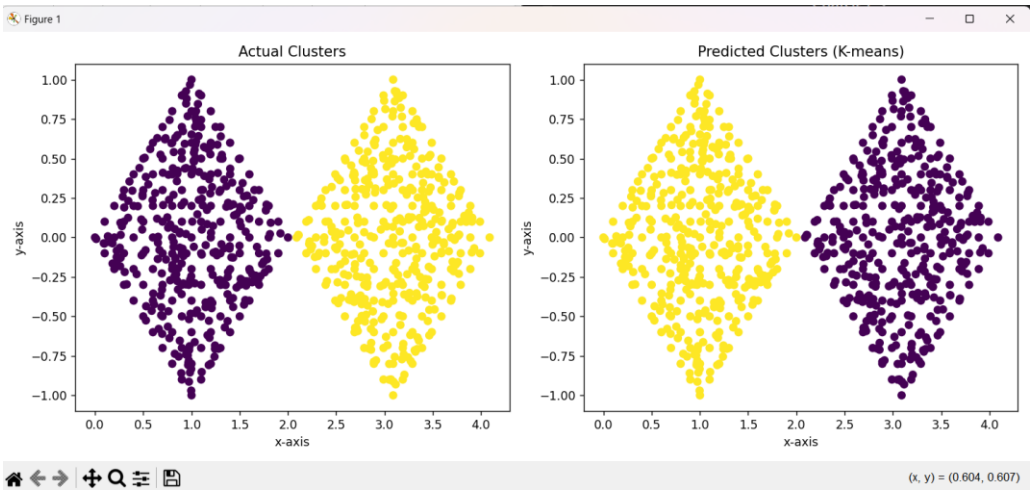


Figure 1
Actual Clusters
Predicted Clusters (K-means)
x-axis
y-axis

g)



2.

Defining Spectral clustering:

Structure of spectral clustering: Constructing similarity graph(affinity matrix) Gaussian kernel → Calculate laplacian matrix (Degree matrix - affinity matrix) → Eigen vectors of laplacian matrix → normalizing eigenvectors to make sure the data is on same scale before clustering → k-means used on the new space formed by eigenvectors.

In finding the similarity graph, I mean affinity matrix we rely on Gaussian kernels where sigma is used. Lets see how the sigma effects the results in the problem.

The function for Spectral clustering is saved in my_spectralclustering.py

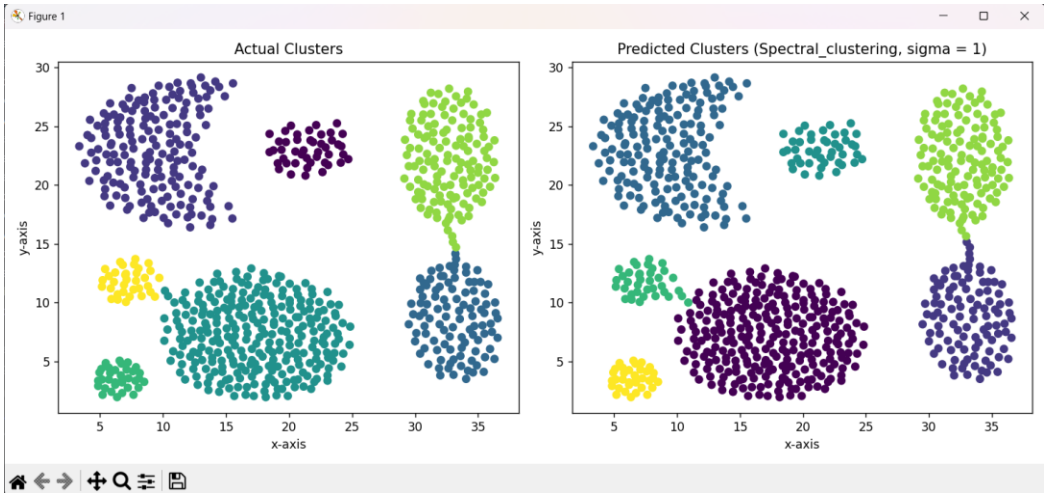
```python
from sklearn.cluster import KMeans
from sklearn.metrics import pairwise_distances

from scipy.sparse.linalg import eigs

def my_spectralclustering(data, K, sigma):  4 usages
    # Calculate the pairwise distance matrix based on Gaussian kernel
    pairwise_dist = pairwise_distances(data, metric='euclidean', squared=True)
    aff_matrix = np.exp(-pairwise_dist / (2 * sigma**2))  #affinity matrix

    # Creating a graph Laplacian matrix
    degree_matrix = np.diag(np.sum(aff_matrix, axis=1))
    laplacian_matrix = degree_matrix - aff_matrix  #(D-A)

    # Computing the first K eigenvectors of the Laplacian matrix
    _, eigenvectors = eigs(laplacian_matrix, k=K, which='SM')
    # Using only the real part of the eigenvector matrix
    eigenvectors = np.real(eigenvectors)
    # Normalizing the rows of the eigenvector matrix
    normalized_eigenvectors = eigenvectors / np.linalg.norm(eigenvectors, axis=1)[:, np.newaxis]

    # Performing K-means clustering on the normalized eigenvectors
    km = KMeans(n_clusters=K, random_state=42)
    cluster_labels = km.fit_predict(normalized_eigenvectors) + 1  # added 1 to start cluster labels from 1.

    return cluster_labels
```

I would have used same kmeans function to plot spectral clustering, but since we have additonal argument sigma, to avoid confusion, I am defining a new function.

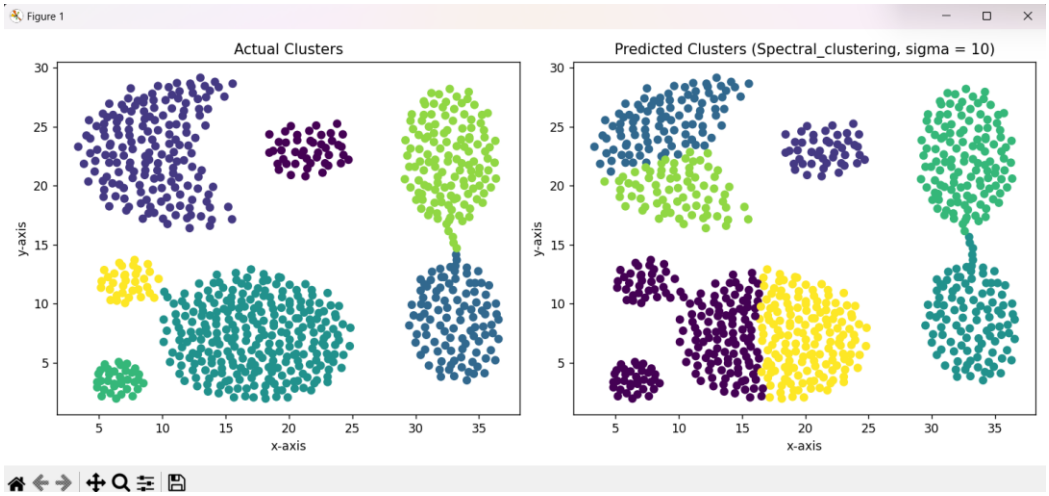
```python
def spectral_clustering(data, true_labels, K, spect_clus, sigma):  3 usages
    cluster_labels = spect_clus(data, K, sigma)
    # Creating a DataFrame for visualization
    df = pd.DataFrame(data)
    df['true_labels'] = true_labels
    df['cluster_labels'] = cluster_labels
    # Scatter plot for actual clusters
    plt.figure(figsize=(12, 5))
    plt.subplot( *args: 1, 2, 1)
    plt.scatter(df[0], df[1], c=df['true_labels'], cmap='viridis')
    plt.title('Actual Clusters')
    plt.xlabel('x-axis')
    plt.ylabel('y-axis')
    # Scatter plot for predicted clusters (K-means)
    plt.subplot( *args: 1, 2, 2)
    plt.scatter(df[0], df[1], c=df['cluster_labels'], cmap='viridis')
    plt.title(f'Predicted Clusters (Spectral_clustering, sigma = {sigma})')
    plt.xlabel('x-axis')
    plt.ylabel('y-axis')
    plt.tight_layout()
    plt.show()
    # Print the cluster labels and the number of iterations
    print(cluster_labels)
```

Case 1:

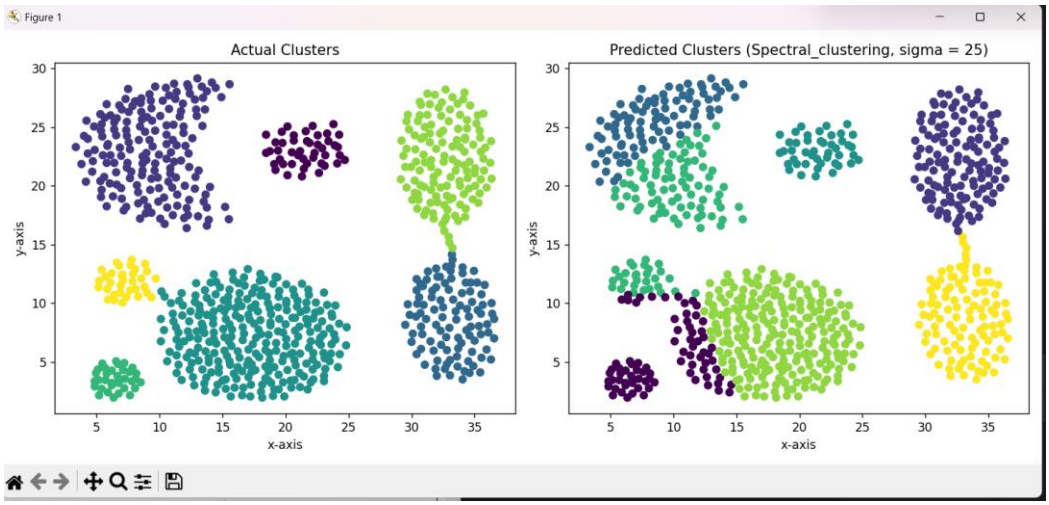
Sigma = 1



Case 2: Sigma = 10



Case 3: Sigma = 25



So I tried 3 different sigma values, I realized when the sigma (1) is small, only very close points are considered similar, while increasing the sigma (10, 25), I observed many data points blended into same cluster. So choosing the appropriate sigma is effective especially in complex and non linear data distribution. I am trying different sigma values for other datasets and attaching the best results only.

(b)

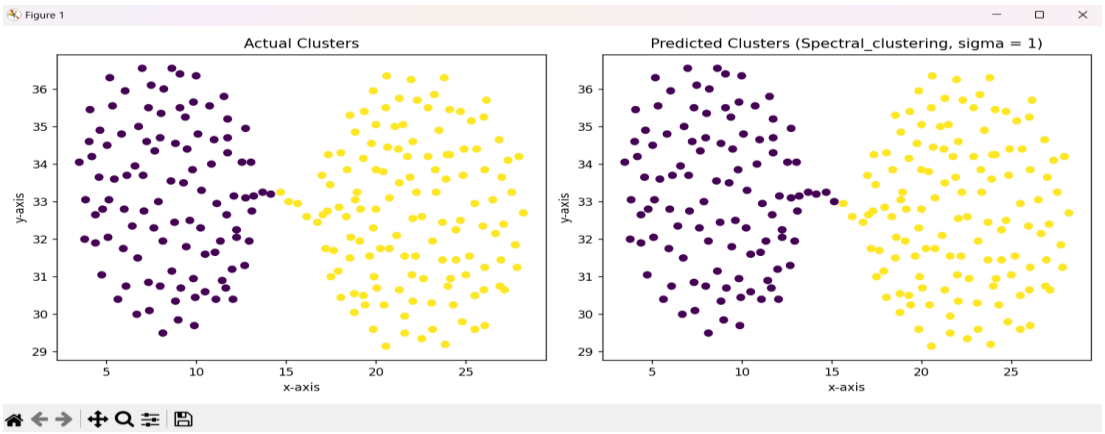


Figure 1
Actual Clusters
Predicted Clusters (Spectral_clustering, sigma = 1)
y-axis
x-axis

(c)

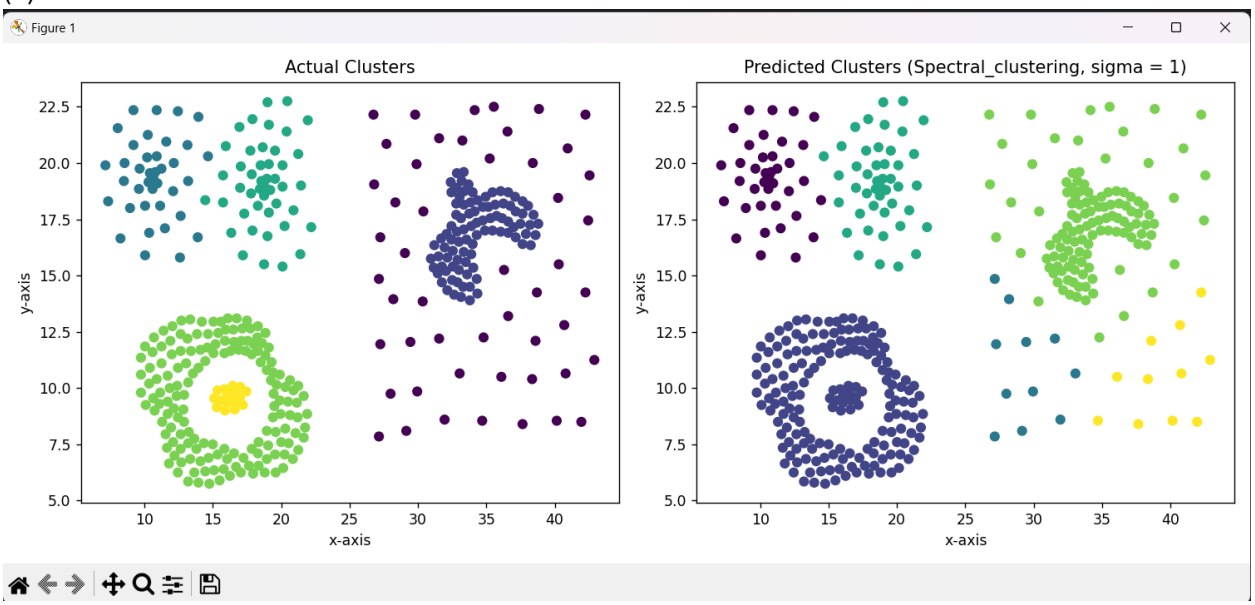


Figure 1
Actual Clusters
Predicted Clusters (Spectral_clustering, sigma = 1)
y-axis
x-axis

d)

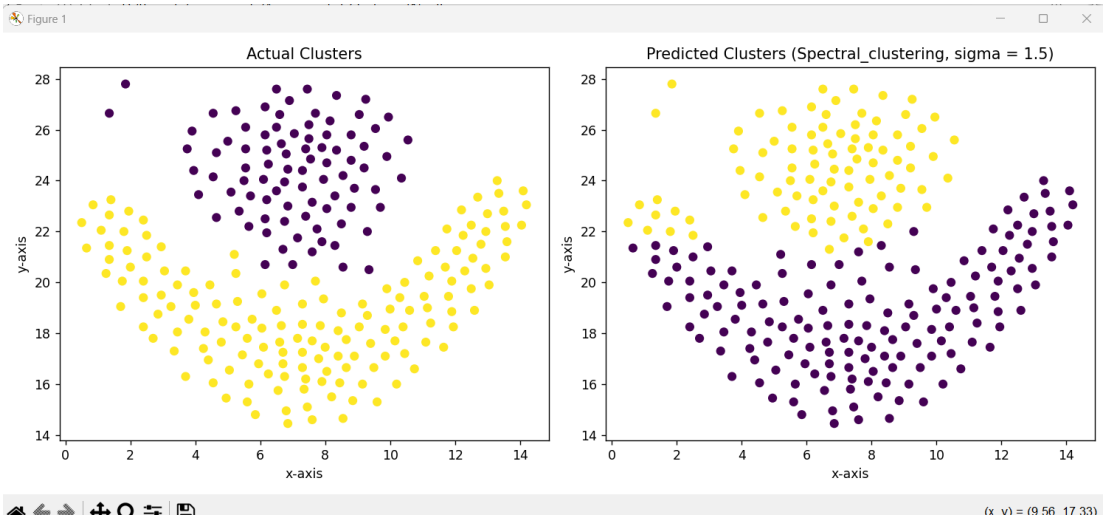


Figure 1
Actual Clusters
Predicted Clusters (Spectral_clustering, sigma = 1.5)
y-axis
x-axis

e)

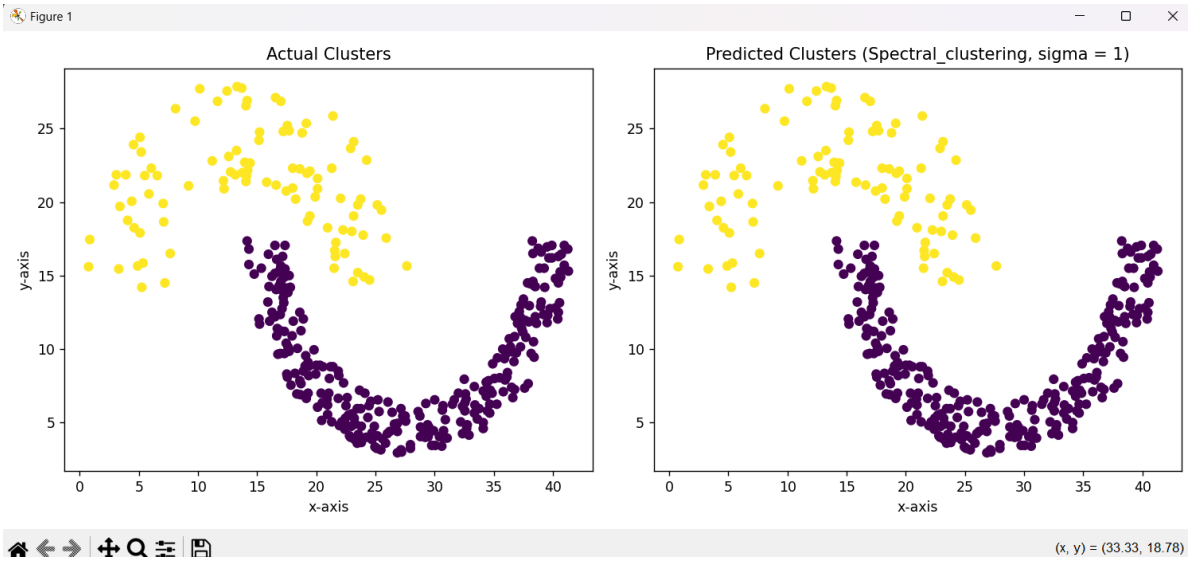


Figure 1
Actual Clusters
Predicted Clusters (Spectral_clustering, sigma = 1)
y-axis
x-axis
(x, y) = (33.33, 18.78)

f)

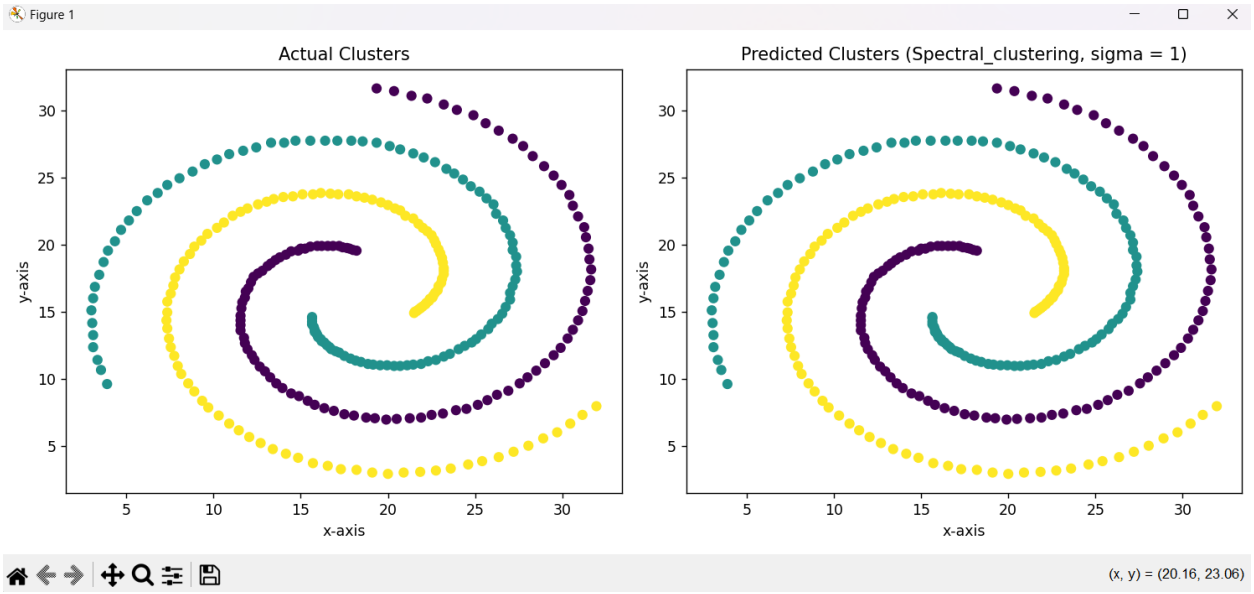
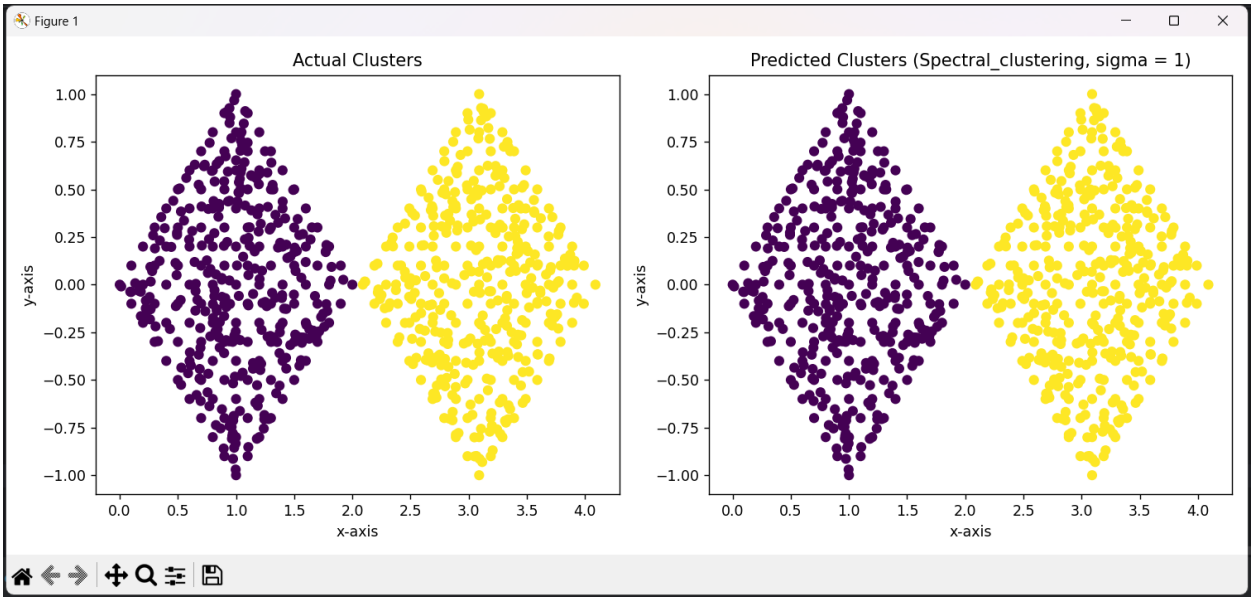


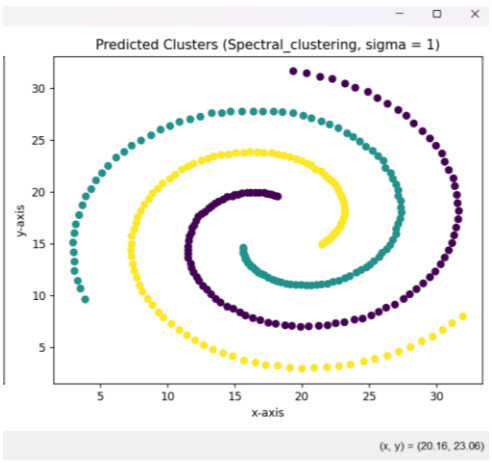
Figure 1
Actual Clusters
Predicted Clusters (Spectral_clustering, sigma = 1)
y-axis
x-axis
(x, y) = (20.16, 23.06)

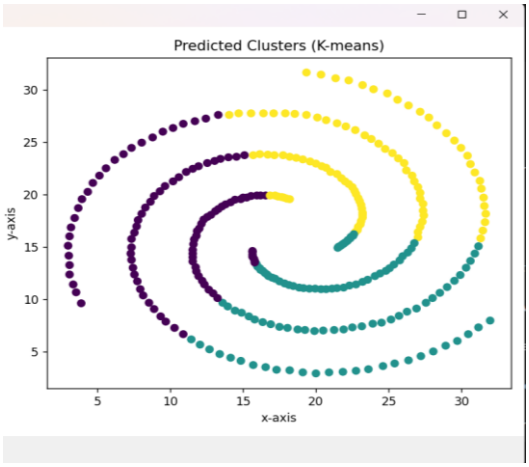
g)



3. Compare your spectral clustering results with k-means. It is natural that on certain hard toy example, both method won't generate perfect results. In your report, briefly analyze what is the advantage or disadvantage of spectral clustering over k-means. Why it is the case? (You do not need to mathematically prove it but just need to give answers in your own language.)

Answer: In my analysis, I am most surprised with clustering result of spiral dataset(f, k=3) from spectral clustering model.





It is very evident that Spectral clustering is able to cluster the data distributed in spiral which is not same case for k-means. This is the best advantage of using Spectral clustering, the eigen vectors of laplace matrix create a new space for the data points so k means can cluster with ease. The value of gamma plays a vital role in performance of spectral clustering.