Problem 1

$$\min \sum_i \sum_k \pi_{ik} \cdot ||X_i - \mu_k||^2$$

A)

Prove that E step update on membership (π) achieves the minimum objective given the current centroids (μ)

The E-step in K-means clustering updates the cluster membership of each (π) given the centroids (μ). The objective of K-means is to minimize the within cluster sum of squares. The distance between each datapoint and it's assigned centroid. The E-step assigns the new clusters such that the sum of square is minimized for point X_i.

$\frac{\partial J}{\partial \pi_{ik}} = \sum_i \sum_k \pi_{ik} \cdot ||X_i - \mu_k||^2$

-> π_{ik} = {1 if k = $argmin_j ||X_i$ - $\mu_j||^2$, else 0}

This step ensures that within cluster sum of squares is minimized since each data point is assigned to the centroid closest to it. By minimizing the within cluster sum of squares, the E-step updates (π) to achieve minimum objective given current centroid.

B)

Prove that M step update on centroids (μ) achievess the minimum objective given the current memberships (π)

Proof:

O = $\sum_i \sum_k \pi_{ik} \cdot ||X_i - \mu_k||^2$

$\frac{\partial O}{\partial \mu_k}$ = $\frac{\partial(\sum_i \sum_k \pi_{ik} \cdot ||X_i - \mu_k||^2)}{\partial \mu_k}$

$\frac{\partial O}{\partial \mu_k}$ = $(-2 \cdot \sum_i \cdot ||X_i - \mu_k||^2)$

0 = $(-2) \cdot \sum_i \cdot \pi_{ik} \cdot (X_i - \mu_k)$

0 = $2 \cdot \sum_i (\pi_{ik} \cdot X_i) - (\pi_{ik} \cdot \mu_k)$

0 = $2 \sum_i (\pi_{ik} \cdot X_i) - 2 \sum_i (\pi_{ik} \cdot \mu_k)$

$2 \sum_i (\pi_{ik} \cdot \mu_k) = 2 \sum_i (\pi_{ik} \cdot X_i)$

μ_k = $\frac{2 \sum_i (\pi_{ik} \cdot X_i)}{2 \sum_i (\pi_{ik})}$

μ_k = $\frac{\sum_i (\pi_{ik} \cdot X_i)}{\sum_i (\pi_{ik})}$

C)

This is because Kmeans is not necessarily a convex funtion.

There can be many local minimums along the function curve.

When Kmean randomly initialize the mu or the pi, it is not guaranteed that it's initialized adjacent to the globabl minimum.

And since Kmeans can only go down the curve, it's not gauranteed that the curve will reach to the global minimum

It does not guarantee to converge to the global minimum of the objective function, which is the sum of squared distances between samples and their assigned centroid.

This is due to the non-convex nature of the objective function, meaning that it has multiple local minima and a single global minimum, and KMeans can only guarantee convergence to a local minimum, not necessarily the global one.

Problem 2

In [1]:
```
import numpy as np
from sklearn.metrics import euclidean_distances
from sklearn.metrics import pairwise_distances_argmin

def init_random_centroids(data, k):
    centroids = data.copy()
    np.random.shuffle(centroids)
    return centroids[:k]

def assign_cluster(data, centroids):
    distances = pairwise_distances_argmin(data, centroids, metric = 'euclidean')
    return distances

def update_centroids(data, centroids, cluster_assignment):
    new_centroids = np.zeros(centroids.shape)
    for i in range(centroids.shape[0]):
        data_points = data[cluster_assignment == i]
        new_centroids[i] = np.mean(data_points, axis=0)
    return new_centroids

def kmeans(data, k, max_iter = 100):
    centroids = init_random_centroids(data, k)
    for i in range(max_iter):
        cluster_assignment = assign_cluster(data, centroids)
        centroids = update_centroids(data, centroids, cluster_assignment)
    return centroids, cluster_assignment

def evaluate_purity(data, cluster_assignment, labels):
    n_clusters = len(np.unique(cluster_assignment))
    total_purity = 0
    for i in range(n_clusters):
        cluster = data[cluster_assignment == i]
        cluster_labels = labels[cluster_assignment == i]
        unique_labels, counts = np.unique(cluster_labels, return_counts = True)
        total_purity += np.max(counts)
    return total_purity / data.shape[0]

def loss(data, labels, centroids, cluster_assignment):
    distances = euclidean_distances(data, centroids[cluster_assignment])
    return np.sum(np.min(distances, axis = 1))

def run_kmeans(data, labels, k, max_iter = 100):
    centroids, cluster_assignment = kmeans(data, k, max_iter)
    purity = evaluate_purity(data, cluster_assignment, labels)
    loss_val = loss(data, labels, centroids, cluster_assignment)
    return centroids, cluster_assignment, purity, loss_val

def gini_index(cluster_assignment, labels):
    n_clusters = len(np.unique(cluster_assignment))
    gini_index = 0
    for i in range(n_clusters):
        cluster = cluster_assignment == i
        cluster_labels = labels[cluster]
        unique_labels, counts = np.unique(cluster_labels, return_counts = True)
        prob = counts / cluster_labels.shape[0]
        gini_index += 1 - np.sum(prob**2)
    return gini_index / n_clusters
```

A) MNIST Data

In [2]:
```
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784', version = 1)
data = mnist.data / 255.0
labels = mnist.target.astype(int)
```

In [3]:
```
# For k = 5
centroids, cluster_assignment, purity, loss_val = run_kmeans(data, labels, k = 5, max_iter = 100)
print("Purity:", purity)

gini = gini_index(cluster_assignment, labels)
print("Gini index:", gini)
```

```
Purity: 0.4013285714285714
Gini index: 0.7386591248499118
```

In [4]:
```
# For k = 10
centroids, cluster_assignment, purity, loss_val = run_kmeans(data, labels, k = 10, max_iter = 100)
print("Purity:", purity)

gini = gini_index(cluster_assignment, labels)
print("Gini index:",gini)
```

```
Purity: 0.5987714285714286
Gini index: 0.46830837431349026
```

In [5]:
```
# For k = 20
centroids, cluster_assignment, purity, loss_val = run_kmeans(data, labels, k = 20, max_iter = 100)
print("Purity:", purity)

gini = gini_index(cluster_assignment, labels)
print("Gini index:",gini)
```

```
Purity: 0.7137857142857142
Gini index: 0.3635434133606016
```

B) Fashion Data

In [6]:
```
from sklearn.datasets import fetch_openml
fashion = fetch_openml('Fashion-MNIST', version = 1)
data = fashion.data / 255.0
labels = fashion.target.astype(int)
```

In [7]:
```
# For k = 5
centroids, cluster_assignment, purity, loss_val = run_kmeans(data, labels, k = 5, max_iter = 100)
print("Purity:", purity)

gini = gini_index(cluster_assignment, labels)
print("Gini index:", gini)
```

```
Purity: 0.3823285714285714
Gini index: 0.6973612601329064
```

In [8]:
```
# For k = 10
centroids, cluster_assignment, purity, loss_val = run_kmeans(data, labels, k = 10, max_iter = 100)
print("Purity:", purity)

gini = gini_index(cluster_assignment, labels)
print("Gini index:",gini)
```

```
Purity: 0.5541857142857143
Gini index: 0.47311027118322474
```

In [9]:
```
# For k = 20
centroids, cluster_assignment, purity, loss_val = run_kmeans(data, labels, k = 20, max_iter = 100)
print("Purity:", purity)

gini = gini_index(cluster_assignment, labels)
print("Gini index:",gini)
```

```
Purity: 0.6524142857142857
Gini index: 0.4290915488437658
```

C) 20NG DATA

In [10]:
```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_selection import SelectKBest, chi2

train_set = fetch_20newsgroups(subset = 'train')
train_data = train_set.data
train_label = train_set.target
test_set = fetch_20newsgroups(subset = 'test')
test_data = test_set.data
test_label = test_set.target

# Normalizing the data
vectorizer = TfidfVectorizer(stop_words = 'english')
train_data = vectorizer.fit_transform(train_data)
train_data = np.array(train_data.todense())
```

In [11]:
```
# For k = 5
centroids, cluster_assignment, purity, loss_val = run_kmeans(train_data[:100], train_label[:100], k = 5, max_iter = 1000)
print("Purity:", purity)

gini = gini_index(cluster_assignment, train_label[:100])
print("Gini index:", gini)
```

```
Purity: 0.2
Gini index: 0.871459137498951
```

In [12]:
```
# For k = 10
centroids, cluster_assignment, purity, loss_val = run_kmeans(train_data[:100], train_label[:100], k = 10, max_iter = 1000)
print("Purity:", purity)

gini = gini_index(cluster_assignment, train_label[:100])
print("Gini index:", gini)
```

```
Purity: 0.28
Gini index: 0.7840994687131051
```

In [13]:
```
# For k = 20
centroids, cluster_assignment, purity, loss_val = run_kmeans(train_data[:100], train_label[:100], k = 20, max_iter = 1000)
print("Purity:", purity)

gini = gini_index(cluster_assignment, train_label[:100])
print("Gini index:", gini)
```

```
Purity: 0.4
Gini index: 0.5981527777777778
```

Problem 3

In [1]:
```
import numpy as np
from scipy.special import logsumexp
from scipy.stats import multivariate_normal
from random import randint
import pandas as pd
```

In [2]:
```
def gaussian_data():
    with open("2gaussian.txt", 'r') as f:
        lines = f.readlines()
    data = []
    for line in lines:
        x, y = line.strip().split()
        data.append([float(x), float(y)])
    return np.array(data)
```

In [3]:
```
data = gaussian_data()
print(data)
```

```
[[7.57104365 3.53027417]
 [7.33721752 4.26271316]
 [3.07182783 1.11801871]
 ...
 [5.61639331 3.77793239]
 [8.59215378 3.6349037 ]
 [3.02221288 3.78337346]]
```

In [4]:
```
def gaussian(X, MU, Covariance) -> np.array:
    n = X.shape[1]
    difference = (X - MU).T

    base = 1 / ((2 * np.pi) ** (n / 2) * np.linalg.det(Covariance) ** 0.5)
    exponent_value = -0.5 * np.dot(np.dot(difference.T, np.linalg.inv(Covariance)), difference)
    exponent = np.exp(exponent_value)

    return np.diagonal( base *  exponent).reshape(-1, 1)
```

In [5]:
```
def initialize_clusters(X, k) -> np.array:

    PI = [ 1/k for i in range(0,k) ]
    MU = [ X[randint(0,len(X)-1),:] for i in range(0,k) ]
    Covariance = [ [ np.identity(X.shape[1] ,dtype = np.float64) ] for i in range(0,k) ]

    clusters = []
    for i in range(k):
        cluster = {}
        cluster['PI'] = PI[i]
        cluster['MU'] = MU[i]
        cluster['Covariance'] = Covariance[i]
        clusters.append(cluster)

    return clusters
```

In [6]:
```
def E_step(X, clusters) -> dict:
    expectation = np.zeros((X.shape[0], 1), dtype = np.float64)

    for cluster in clusters:
        PI = cluster['PI']
        MU = cluster['MU']
        Covariance = cluster['Covariance']

        weight = (PI * gaussian(X, MU, Covariance)).astype(np.float64)

        for i in range(X.shape[0]):
            expectation[i] += weight[i]

        cluster['weight'] = weight
        cluster['expectation'] = expectation

    for cluster in clusters:
        cluster['weight'] /= cluster['expectation']

    return cluster
```

In [7]:
```
def M_step(X, clusters) -> dict:
    X_len = float(X.shape[0])

    for cluster in clusters:
        weight = cluster['weight']
        Covariance = np.zeros((X.shape[1], X.shape[1]))
        sum_weights = np.sum(weight, axis=0)
        PI = sum_weights / X_len
        MU = np.sum(weight * X, axis=0) / sum_weights

        for i in range(X.shape[0]):
            difference = (X[i] - MU).reshape(-1, 1)
            Covariance += weight[i] * np.dot(difference, difference.T)
        Covariance = Covariance / sum_weights

        cluster['PI'] = PI
        cluster['MU'] = MU
        cluster['Covariance'] = Covariance

    return clusters
```

In [8]:
```
def get_likelihood(X, clusters) -> list:

    likelihoods = np.log(np.array([cluster['expectation'] for cluster in clusters]))
    sum_log_likelihood = np.sum(likelihoods)

    return [sum_log_likelihood, likelihoods]
```

In [9]:
```
k = 2
cycles = 4000
X_partition = []
X = data

clusters = initialize_clusters(X, k = 2)
likelihoods = np.zeros((cycles, ))

updated_likelihood = 0
for i in range(cycles):
  E_step(X, clusters)
  M_step(X, clusters)

  result = get_likelihood(X, clusters)
  likelihood, sample_likelihoods = result[0], result[1]

  if likelihood == updated_likelihood: break
  else:
    updated_likelihood = likelihood
    print('Cycle: ', i + 1, '  |  Likelihood: ', likelihood)


n = 0
clusters
for cluster in clusters:
  n += 1
  print('\nCluster  :  ', n )
  PI =  cluster['PI']
  MU =  cluster['MU']
  Covariance = cluster['Covariance']

  print('PI : ', PI)
  print('Mean : ', MU)
  print('Covariance Matrix : \n', np.array(Covariance))
```

```
Cycle:  1   |  Likelihood:  -21759.283746140267
Cycle:  2   |  Likelihood:  -45391.91427481979
Cycle:  3   |  Likelihood:  -44986.21124153352
Cycle:  4   |  Likelihood:  -44640.75026177596
Cycle:  5   |  Likelihood:  -44361.739777584706
Cycle:  6   |  Likelihood:  -44079.794823029406
Cycle:  7   |  Likelihood:  -43705.95194052593
Cycle:  8   |  Likelihood:  -43221.08633291509
Cycle:  9   |  Likelihood:  -42789.717951574545
Cycle:  10   |  Likelihood:  -42531.28534627899
Cycle:  11   |  Likelihood:  -42373.647344868565
Cycle:  12   |  Likelihood:  -42262.504019602355
Cycle:  13   |  Likelihood:  -42180.60659853055
Cycle:  14   |  Likelihood:  -42122.50013982032
Cycle:  15   |  Likelihood:  -42084.62906798239
Cycle:  16   |  Likelihood:  -42062.12165213494
Cycle:  17   |  Likelihood:  -42049.75389655398
Cycle:  18   |  Likelihood:  -42043.34857348301
Cycle:  19   |  Likelihood:  -42040.16694358488
Cycle:  20   |  Likelihood:  -42038.6307812887
Cycle:  21   |  Likelihood:  -42037.9030107265
Cycle:  22   |  Likelihood:  -42037.56254138199
Cycle:  23   |  Likelihood:  -42037.40459251016
Cycle:  24   |  Likelihood:  -42037.331727191086
Cycle:  25   |  Likelihood:  -42037.29823874928
Cycle:  26   |  Likelihood:  -42037.2828864106
Cycle:  27   |  Likelihood:  -42037.27586025005
Cycle:  28   |  Likelihood:  -42037.27264832117
Cycle:  29   |  Likelihood:  -42037.27118115374
Cycle:  30   |  Likelihood:  -42037.270511318304
Cycle:  31   |  Likelihood:  -42037.270205611916
Cycle:  32   |  Likelihood:  -42037.270066123434
Cycle:  33   |  Likelihood:  -42037.27000248743
Cycle:  34   |  Likelihood:  -42037.269973459195
Cycle:  35   |  Likelihood:  -42037.26996021862
Cycle:  36   |  Likelihood:  -42037.26995417953
Cycle:  37   |  Likelihood:  -42037.26995142516
Cycle:  38   |  Likelihood:  -42037.26995016895
Cycle:  39   |  Likelihood:  -42037.26994959602
Cycle:  40   |  Likelihood:  -42037.269949334725
Cycle:  41   |  Likelihood:  -42037.26994921556
Cycle:  42   |  Likelihood:  -42037.269949161215
Cycle:  43   |  Likelihood:  -42037.26994913643
Cycle:  44   |  Likelihood:  -42037.269949125126
Cycle:  45   |  Likelihood:  -42037.26994911997
Cycle:  46   |  Likelihood:  -42037.269949117624
Cycle:  47   |  Likelihood:  -42037.26994911655
Cycle:  48   |  Likelihood:  -42037.26994911605
Cycle:  49   |  Likelihood:  -42037.269949115835
Cycle:  50   |  Likelihood:  -42037.26994911574
Cycle:  51   |  Likelihood:  -42037.26994911568
Cycle:  52   |  Likelihood:  -42037.26994911567
Cycle:  53   |  Likelihood:  -42037.26994911566
Cycle:  54   |  Likelihood:  -42037.26994911565

Cluster  :   1
PI :  [0.66520423]
Mean :  [7.01314832 3.98313419]
Covariance Matrix :
 [[0.97475892 0.4974703 ]
 [0.4974703  1.00114259]]

Cluster  :   2
PI :  [0.33479577]
Mean :  [2.99413183 3.0520966 ]
Covariance Matrix :
 [[1.01023427 0.02719139]
 [0.02719139 2.93782296]]
```

In [10]:

```python
n1 = n2 = 0

for i in range(X.shape[0]):
    if clusters[0]['weight'][i][0] >= clusters[1]['weight'][i][0]:
        n1 += 1
    else:
        n2 += 1

print("Number of data points in Cluster 1:", n1)
print("Number of data points in Cluster 2:", n2)
```

```
Number of data points in Cluster 1: 4009
Number of data points in Cluster 2: 1991
```

In [11]:
```
def gaussian_data():
    with open("3gaussian.txt", 'r') as f:
        lines = f.readlines()
    data = []
    for line in lines:
        x, y = line.strip().split()
        data.append([float(x), float(y)])
    return np.array(data)
```

In [12]:
```
data = gaussian_data()
print(data)
```

```
[[2.94693347 3.16222499]
 [5.98399602 4.84671738]
 [5.30142995 8.16811309]
 ...
 [6.27055168 2.83700248]
 [5.27935185 7.87197636]
 [7.26196796 4.58568396]]
```

In [13]:
```
k = 3
cycles = 4000
X_partition = []
X = data

clusters = initialize_clusters(X, k = 3)
likelihoods = np.zeros((cycles, ))

updated_likelihood = 0
for i in range(cycles):
  E_step(X, clusters)
  M_step(X, clusters)

  result = get_likelihood(X, clusters)
  likelihood, sample_likelihoods = result[0], result[1]

  if likelihood == updated_likelihood: break
  else:
    updated_likelihood = likelihood
    print('Cycle: ', i + 1, '  |  Likelihood: ', likelihood)

n = 0
clusters
for cluster in clusters:
  n += 1
  print('\nCluster  :  ', n )
  PI =  cluster['PI']
  MU =  cluster['MU']
  Covariance = cluster['Covariance']

  print('PI : ', PI)
  print('Mean : ', MU)
  print('Covariance Matrix : \n', np.array(Covariance))
```

```
Cycle:  1   |  Likelihood:  -125169.26396770614
Cycle:  2   |  Likelihood:  -117363.83200370363
Cycle:  3   |  Likelihood:  -116336.36189120801
Cycle:  4   |  Likelihood:  -115839.98478240208
Cycle:  5   |  Likelihood:  -115510.99791576609
Cycle:  6   |  Likelihood:  -115284.41768730324
Cycle:  7   |  Likelihood:  -115124.56867900268
Cycle:  8   |  Likelihood:  -115007.55604407785
Cycle:  9   |  Likelihood:  -114918.5030418912
Cycle:  10   |  Likelihood:  -114848.45150494791
Cycle:  11   |  Likelihood:  -114791.80806195675
Cycle:  12   |  Likelihood:  -114744.83680019964
Cycle:  13   |  Likelihood:  -114704.90419575069
Cycle:  14   |  Likelihood:  -114670.10720811533
Cycle:  15   |  Likelihood:  -114639.06936005372
Cycle:  16   |  Likelihood:  -114610.81226993594
Cycle:  17   |  Likelihood:  -114584.66712532956
Cycle:  18   |  Likelihood:  -114560.20923136428
Cycle:  19   |  Likelihood:  -114537.20339013917
Cycle:  20   |  Likelihood:  -114515.5516953353
Cycle:  21   |  Likelihood:  -114495.24212379252
Cycle:  22   |  Likelihood:  -114476.30225769812
Cycle:  23   |  Likelihood:  -114458.76324385204
Cycle:  24   |  Likelihood:  -114442.63603403815
Cycle:  25   |  Likelihood:  -114427.89904286804
Cycle:  26   |  Likelihood:  -114414.49518481645
Cycle:  27   |  Likelihood:  -114402.33615576813
Cycle:  28   |  Likelihood:  -114391.31198657924
Cycle:  29   |  Likelihood:  -114381.3039441348
Cycle:  30   |  Likelihood:  -114372.19859110659
Cycle:  31   |  Likelihood:  -114363.90018000347
Cycle:  32   |  Likelihood:  -114356.338069299
Cycle:  33   |  Likelihood:  -114349.46673717414
Cycle:  34   |  Likelihood:  -114343.25885009079
Cycle:  35   |  Likelihood:  -114337.69505166504
Cycle:  36   |  Likelihood:  -114332.75496473334
Cycle:  37   |  Likelihood:  -114328.4119517584
Cycle:  38   |  Likelihood:  -114324.63163281936
Cycle:  39   |  Likelihood:  -114321.37284295177
Cycle:  40   |  Likelihood:  -114318.58967395505
Cycle:  41   |  Likelihood:  -114316.23372648103
Cycle:  42   |  Likelihood:  -114314.25614940186
Cycle:  43   |  Likelihood:  -114312.60931154268
Cycle:  44   |  Likelihood:  -114311.24807470913
Cycle:  45   |  Likelihood:  -114310.130686447
Cycle:  46   |  Likelihood:  -114309.21933145834
Cycle:  47   |  Likelihood:  -114308.4803913475
Cycle:  48   |  Likelihood:  -114307.88446843742
Cycle:  49   |  Likelihood:  -114307.40623104674
Cycle:  50   |  Likelihood:  -114307.024134804
Cycle:  51   |  Likelihood:  -114306.72006812986
Cycle:  52   |  Likelihood:  -114306.47896137554
Cycle:  53   |  Likelihood:  -114306.28838976455
Cycle:  54   |  Likelihood:  -114306.13819142291
Cycle:  55   |  Likelihood:  -114306.02011413372
Cycle:  56   |  Likelihood:  -114305.92749833548
Cycle:  57   |  Likelihood:  -114305.85499931942
Cycle:  58   |  Likelihood:  -114305.79834840343
Cycle:  59   |  Likelihood:  -114305.75415082586
Cycle:  60   |  Likelihood:  -114305.7197169424
Cycle:  61   |  Likelihood:  -114305.69292278407
Cycle:  62   |  Likelihood:  -114305.67209593893
Cycle:  63   |  Likelihood:  -114305.65592289779
Cycle:  64   |  Likelihood:  -114305.64337433584
Cycle:  65   |  Likelihood:  -114305.63364520977
Cycle:  66   |  Likelihood:  -114305.62610697266
Cycle:  67   |  Likelihood:  -114305.62026962145
Cycle:  68   |  Likelihood:  -114305.61575166642
Cycle:  69   |  Likelihood:  -114305.61225644684
Cycle:  70   |  Likelihood:  -114305.60955350426
Cycle:  71   |  Likelihood:  -114305.60746397005
Cycle:  72   |  Likelihood:  -114305.6058491262
Cycle:  73   |  Likelihood:  -114305.60460146723
Cycle:  74   |  Likelihood:  -114305.603637728
Cycle:  75   |  Likelihood:  -114305.60289345236
Cycle:  76   |  Likelihood:  -114305.60231876782
Cycle:  77   |  Likelihood:  -114305.60187510174
Cycle:  78   |  Likelihood:  -114305.60153263198
Cycle:  79   |  Likelihood:  -114305.60126830894
Cycle:  80   |  Likelihood:  -114305.60106432265
Cycle:  81   |  Likelihood:  -114305.60090691503
Cycle:  82   |  Likelihood:  -114305.60078546028
Cycle:  83   |  Likelihood:  -114305.60069175341
Cycle:  84   |  Likelihood:  -114305.6006194597
Cycle:  85   |  Likelihood:  -114305.6005636891
Cycle:  86   |  Likelihood:  -114305.60052066734
```

```
Cycle:  87   |  Likelihood:  -114305.60048748151
Cycle:  88   |  Likelihood:  -114305.60046188386
Cycle:  89   |  Likelihood:  -114305.60044213993
Cycle:  90   |  Likelihood:  -114305.60042691158
Cycle:  91   |  Likelihood:  -114305.6004151663
Cycle:  92   |  Likelihood:  -114305.6004061077
Cycle:  93   |  Likelihood:  -114305.60039912131
Cycle:  94   |  Likelihood:  -114305.60039373321
Cycle:  95   |  Likelihood:  -114305.60038957783
Cycle:  96   |  Likelihood:  -114305.60038637317
Cycle:  97   |  Likelihood:  -114305.60038390175
Cycle:  98   |  Likelihood:  -114305.60038199581
Cycle:  99   |  Likelihood:  -114305.60038052601
Cycle:  100   |  Likelihood:  -114305.6003793925
Cycle:  101   |  Likelihood:  -114305.60037851839
Cycle:  102   |  Likelihood:  -114305.60037784431
Cycle:  103   |  Likelihood:  -114305.60037732449
Cycle:  104   |  Likelihood:  -114305.60037692363
Cycle:  105   |  Likelihood:  -114305.60037661449
Cycle:  106   |  Likelihood:  -114305.6003763761
Cycle:  107   |  Likelihood:  -114305.60037619228
Cycle:  108   |  Likelihood:  -114305.6003760505
Cycle:  109   |  Likelihood:  -114305.60037594117
Cycle:  110   |  Likelihood:  -114305.60037585687
Cycle:  111   |  Likelihood:  -114305.60037579187
Cycle:  112   |  Likelihood:  -114305.60037574175
Cycle:  113   |  Likelihood:  -114305.60037570307
Cycle:  114   |  Likelihood:  -114305.60037567327
Cycle:  115   |  Likelihood:  -114305.60037565028
Cycle:  116   |  Likelihood:  -114305.60037563257
Cycle:  117   |  Likelihood:  -114305.60037561889
Cycle:  118   |  Likelihood:  -114305.60037560835
Cycle:  119   |  Likelihood:  -114305.60037560022
Cycle:  120   |  Likelihood:  -114305.60037559396
Cycle:  121   |  Likelihood:  -114305.60037558911
Cycle:  122   |  Likelihood:  -114305.60037558539
Cycle:  123   |  Likelihood:  -114305.60037558252
Cycle:  124   |  Likelihood:  -114305.60037558031
Cycle:  125   |  Likelihood:  -114305.6003755786
Cycle:  126   |  Likelihood:  -114305.60037557727
Cycle:  127   |  Likelihood:  -114305.60037557626
Cycle:  128   |  Likelihood:  -114305.60037557546
Cycle:  129   |  Likelihood:  -114305.60037557487
Cycle:  130   |  Likelihood:  -114305.6003755744
Cycle:  131   |  Likelihood:  -114305.60037557405
Cycle:  132   |  Likelihood:  -114305.60037557378
Cycle:  133   |  Likelihood:  -114305.60037557354
Cycle:  134   |  Likelihood:  -114305.60037557338
Cycle:  135   |  Likelihood:  -114305.60037557327
Cycle:  136   |  Likelihood:  -114305.60037557317
Cycle:  137   |  Likelihood:  -114305.60037557309
Cycle:  138   |  Likelihood:  -114305.60037557302
Cycle:  139   |  Likelihood:  -114305.60037557298
Cycle:  140   |  Likelihood:  -114305.60037557295
Cycle:  141   |  Likelihood:  -114305.60037557292
Cycle:  142   |  Likelihood:  -114305.60037557289
Cycle:  143   |  Likelihood:  -114305.6003755729
Cycle:  144   |  Likelihood:  -114305.60037557289
Cycle:  145   |  Likelihood:  -114305.60037557286

Cluster  :   1
PI :  [0.29843661]
Mean :  [7.02156142 4.01546065]
Covariance Matrix :
 [[0.99041327 0.50095954]
 [0.50095954 0.99564873]]

Cluster  :   2
PI :  [0.49596835]
Mean :  [5.0117217  7.00146622]
Covariance Matrix :
 [[0.97972162 0.18516295]
 [0.18516295 0.97455232]]

Cluster  :   3
PI :  [0.20559504]
Mean :  [3.03968827 3.04847409]
Covariance Matrix :
 [[1.02849913 0.02681589]
 [0.02681589 3.38466417]]
```

In [21]:
```
n1 = n2 = n3 = 0

for i in range(X.shape[0]):
    if clusters[0]['weight'][i][0] >= clusters[1]['weight'][i][0] and clusters[0]['weight'][i][0] >= clusters[2]['weight'][i][0]:
        n2 += 1
    elif clusters[1]['weight'][i][0] >= clusters[0]['weight'][i][0] and clusters[1]['weight'][i][0] >= clusters[2]['weight'][i][(
        n3 += 1
    else:
        n1 += 1

print("Number of data points in Cluster 1:", n1)
print("Number of data points in Cluster 2:", n2)
print("Number of data points in Cluster 3:", n3)
```

```
Number of data points in Cluster 1: 1964
Number of data points in Cluster 2: 3004
Number of data points in Cluster 3: 5032
```

In [ ]:

Problem 4

In [1]:
```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

In [2]:
```
data_spam = pd.read_csv('spambase.data', header = None)
data_spam.rename(columns = {57 : 'spam'}, inplace = True)
```

In [3]:
```
data_spam
```

Out[3]:

	0	1	2	3	4	5	6	7	8	9	...	48	49	50	51	52	53	54	55	56	spam
0	0.00	0.64	0.64	0.0	0.32	0.00	0.00	0.00	0.00	0.00	...	0.000	0.000	0.0	0.778	0.000	0.000	3.756	61	278	1
1	0.21	0.28	0.50	0.0	0.14	0.28	0.21	0.07	0.00	0.94	...	0.000	0.132	0.0	0.372	0.180	0.048	5.114	101	1028	1
2	0.06	0.00	0.71	0.0	1.23	0.19	0.19	0.12	0.64	0.25	...	0.010	0.143	0.0	0.276	0.184	0.010	9.821	485	2259	1
3	0.00	0.00	0.00	0.0	0.63	0.00	0.31	0.63	0.31	0.63	...	0.000	0.137	0.0	0.137	0.000	0.000	3.537	40	191	1
4	0.00	0.00	0.00	0.0	0.63	0.00	0.31	0.63	0.31	0.63	...	0.000	0.135	0.0	0.135	0.000	0.000	3.537	40	191	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4596	0.31	0.00	0.62	0.0	0.00	0.31	0.00	0.00	0.00	0.00	...	0.000	0.232	0.0	0.000	0.000	0.000	1.142	3	88	0
4597	0.00	0.00	0.00	0.0	0.00	0.00	0.00	0.00	0.00	0.00	...	0.000	0.000	0.0	0.353	0.000	0.000	1.555	4	14	0
4598	0.30	0.00	0.30	0.0	0.00	0.00	0.00	0.00	0.00	0.00	...	0.102	0.718	0.0	0.000	0.000	0.000	1.404	6	118	0
4599	0.96	0.00	0.00	0.0	0.32	0.00	0.00	0.00	0.00	0.00	...	0.000	0.057	0.0	0.000	0.000	0.000	1.147	5	78	0
4600	0.00	0.00	0.65	0.0	0.00	0.00	0.00	0.00	0.00	0.00	...	0.000	0.000	0.0	0.125	0.000	0.000	1.250	5	40	0

4601 rows × 58 columns

In [4]:
```
X = data_spam.drop(['spam'], axis = 1)
y = data_spam['spam']
```

In [5]:
```
X
```

Out[5]:

	0	1	2	3	4	5	6	7	8	9	...	47	48	49	50	51	52	53	54	55	56
0	0.00	0.64	0.64	0.0	0.32	0.00	0.00	0.00	0.00	0.00	...	0.0	0.000	0.000	0.0	0.778	0.000	0.000	3.756	61	278
1	0.21	0.28	0.50	0.0	0.14	0.28	0.21	0.07	0.00	0.94	...	0.0	0.000	0.132	0.0	0.372	0.180	0.048	5.114	101	1028
2	0.06	0.00	0.71	0.0	1.23	0.19	0.19	0.12	0.64	0.25	...	0.0	0.010	0.143	0.0	0.276	0.184	0.010	9.821	485	2259
3	0.00	0.00	0.00	0.0	0.63	0.00	0.31	0.63	0.31	0.63	...	0.0	0.000	0.137	0.0	0.137	0.000	0.000	3.537	40	191
4	0.00	0.00	0.00	0.0	0.63	0.00	0.31	0.63	0.31	0.63	...	0.0	0.000	0.135	0.0	0.135	0.000	0.000	3.537	40	191
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4596	0.31	0.00	0.62	0.0	0.00	0.31	0.00	0.00	0.00	0.00	...	0.0	0.000	0.232	0.0	0.000	0.000	0.000	1.142	3	88
4597	0.00	0.00	0.00	0.0	0.00	0.00	0.00	0.00	0.00	0.00	...	0.0	0.000	0.000	0.0	0.353	0.000	0.000	1.555	4	14
4598	0.30	0.00	0.30	0.0	0.00	0.00	0.00	0.00	0.00	0.00	...	0.0	0.102	0.718	0.0	0.000	0.000	0.000	1.404	6	118
4599	0.96	0.00	0.00	0.0	0.32	0.00	0.00	0.00	0.00	0.00	...	0.0	0.000	0.057	0.0	0.000	0.000	0.000	1.147	5	78
4600	0.00	0.00	0.65	0.0	0.00	0.00	0.00	0.00	0.00	0.00	...	0.0	0.000	0.000	0.0	0.125	0.000	0.000	1.250	5	40

4601 rows × 57 columns

In [6]:
```
y
```

Out[6]:
```
0       1
1       1
2       1
3       1
4       1
       ..
4596    0
4597    0
4598    0
4599    0
4600    0
Name: spam, Length: 4601, dtype: int64
```

In [7]:
```
# Scale the features, as the original values have wide ranges
X = StandardScaler().fit_transform(X)
```

In [8]:
```
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size = 0.2, stratify = y)
```

In [9]:
```
import numpy as np
from sklearn.mixture import GaussianMixture
from sklearn.metrics import accuracy_score

def train_gmm(X, y , n_components = 7):
    gmms = []
    classes = np.unique(y)
    for class_id in classes:
        class_data = X[y == class_id]
        gmm = GaussianMixture(n_components = 7)
        gmm.fit(class_data)
        gmms.append(gmm)
    return gmms

def predict_gmm(X, gmms):
    n_samples, _ = X.shape
    n_classes = len(gmms)
    posteriors = np.zeros((n_samples, n_classes))
    for class_id, gmm in enumerate(gmms):
        class_posteriors = gmm.score_samples(X)
        posteriors[:, class_id] = class_posteriors
    return np.argmax(posteriors, axis = 1)

def supervised_gmm(X_train, y_train, X_test, K = 7):
    gmms = train_gmm(X_train, y_train, n_components = 7)
    y_pred = predict_gmm(X_test, gmms)
    return y_pred

y_pred = supervised_gmm(X_train, y_train, X_test)
acc = accuracy_score(y_test, y_pred)
print("Accuracy:", acc)
```

```
Accuracy: 0.8870792616720955
```

In [10]:
```
import numpy as np
from sklearn.mixture import GaussianMixture
from sklearn.metrics import accuracy_score
from sklearn.datasets import fetch_openml
fashion = fetch_openml('Fashion-MNIST', version = 1)
X = fashion.data / 255.0
y = fashion.target.astype(int)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, stratify = y)


def train_gmm(X, y, n_components = 5):
    gmms = []
    classes = np.unique(y)
    for class_id in classes:
        class_data = X[y == class_id]
        gmm = GaussianMixture(n_components = 5)
        gmm.fit(class_data)
        gmms.append(gmm)
    return gmms

def predict_gmm(X, gmms):
    n_samples, _ = X.shape
    n_classes = len(gmms)
    posteriors = np.zeros((n_samples, n_classes))
    for class_id, gmm in enumerate(gmms):
        class_posteriors = gmm.score_samples(X)
        posteriors[:, class_id] = class_posteriors
    return np.argmax(posteriors, axis = 1)

def supervised_gmm(X_train, y_train, X_test, K = 5):
    gmms = train_gmm(X_train, y_train, n_components = 5)
    y_pred = predict_gmm(X_test, gmms)
    return y_pred

y_pred = supervised_gmm(X_train, y_train, X_test)
acc = accuracy_score(y_test, y_pred)
print("Accuracy:", acc)
```

```
Accuracy: 0.7590714285714286
```