Q1: Gaussian Mixture Model(GMM)

(a) GMM learning with Expectation Maximization

GMM with EM on 1D data

In [1]:
```
import matplotlib.pyplot as plt
from sklearn import cluster, datasets, mixture
import numpy as np
from scipy.stats import multivariate_normal
```

Creating the 1d dataset

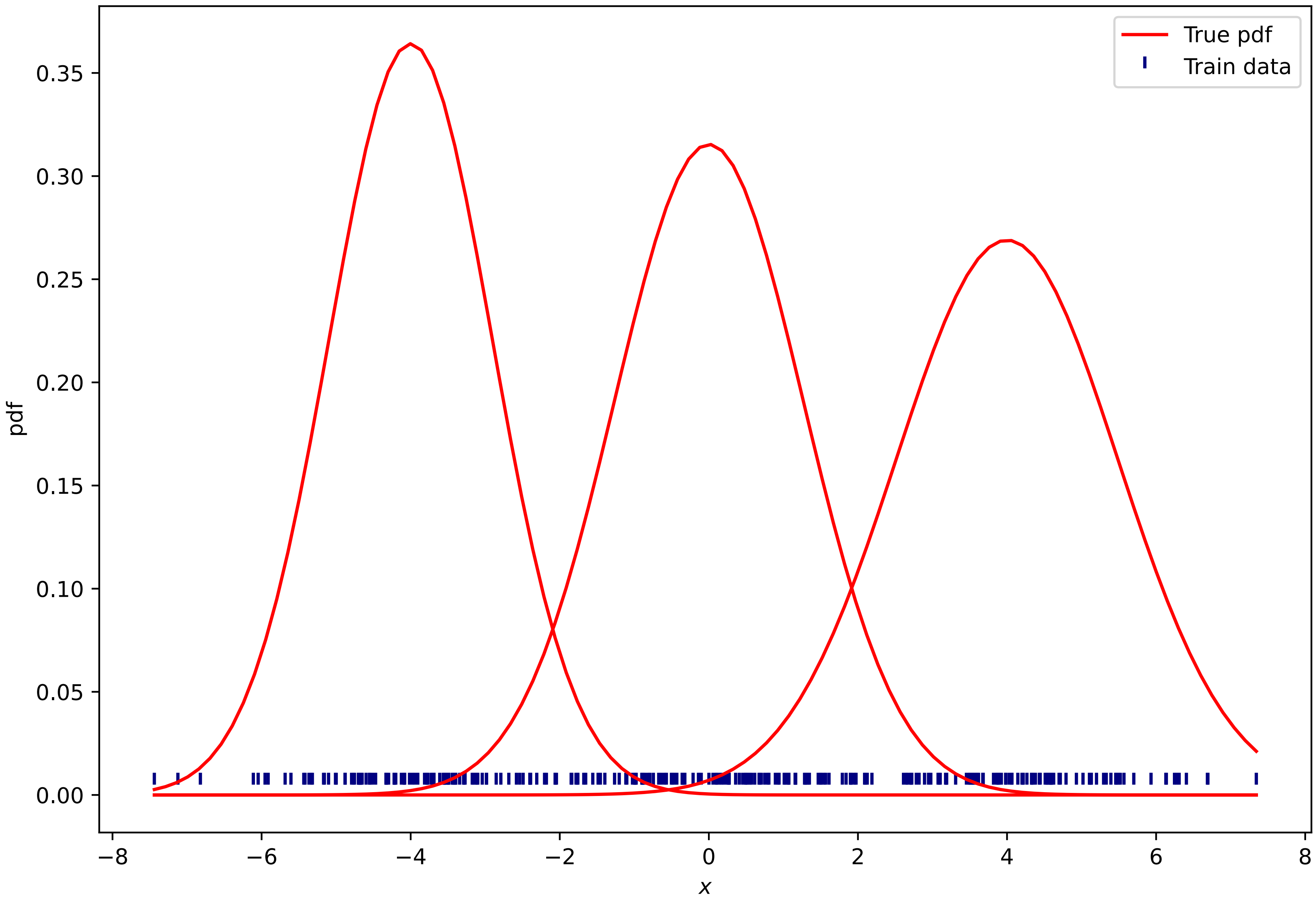
In [63]:
```
# define the number of points
n_samples = 100
mu1, sigma1 = -4, 1.2 # mean and variance
mu2, sigma2 = 4, 2.2 # mean and variance
mu3, sigma3 = 0, 1.6 # mean and variance

x1 = np.random.normal(mu1, np.sqrt(sigma1), n_samples)
x2 = np.random.normal(mu2, np.sqrt(sigma2), n_samples)
x3 = np.random.normal(mu3, np.sqrt(sigma3), n_samples)

X = np.array(list(x1) + list(x2) + list(x3))
np.random.shuffle(X)
print("Dataset shape:", X.shape)
```

```
Dataset shape: (300,)
```

In [64]:
```
def pdf(data, mean: float, variance: float):
  # A normal continuous random variable.
  # Enter your code here 1

  return np.exp(-(data-mean)**2/(2*variance))/(np.sqrt(2*np.pi*variance)+eps)
```

In [65]:
```
# visualize the training data
bins = np.linspace(np.min(X),np.max(X),100)

plt.figure(figsize=(10,7))
plt.xlabel("$x$")
plt.ylabel("pdf")
plt.scatter(X, [0.005] * len(X), color='navy', s=30, marker=2, label="Train data")

plt.plot(bins, pdf(bins, mu1, sigma1), color='red', label="True pdf")
plt.plot(bins, pdf(bins, mu2, sigma2), color='red')
plt.plot(bins, pdf(bins, mu3, sigma3), color='red')

plt.legend()
plt.plot()
```

Out[65]: []



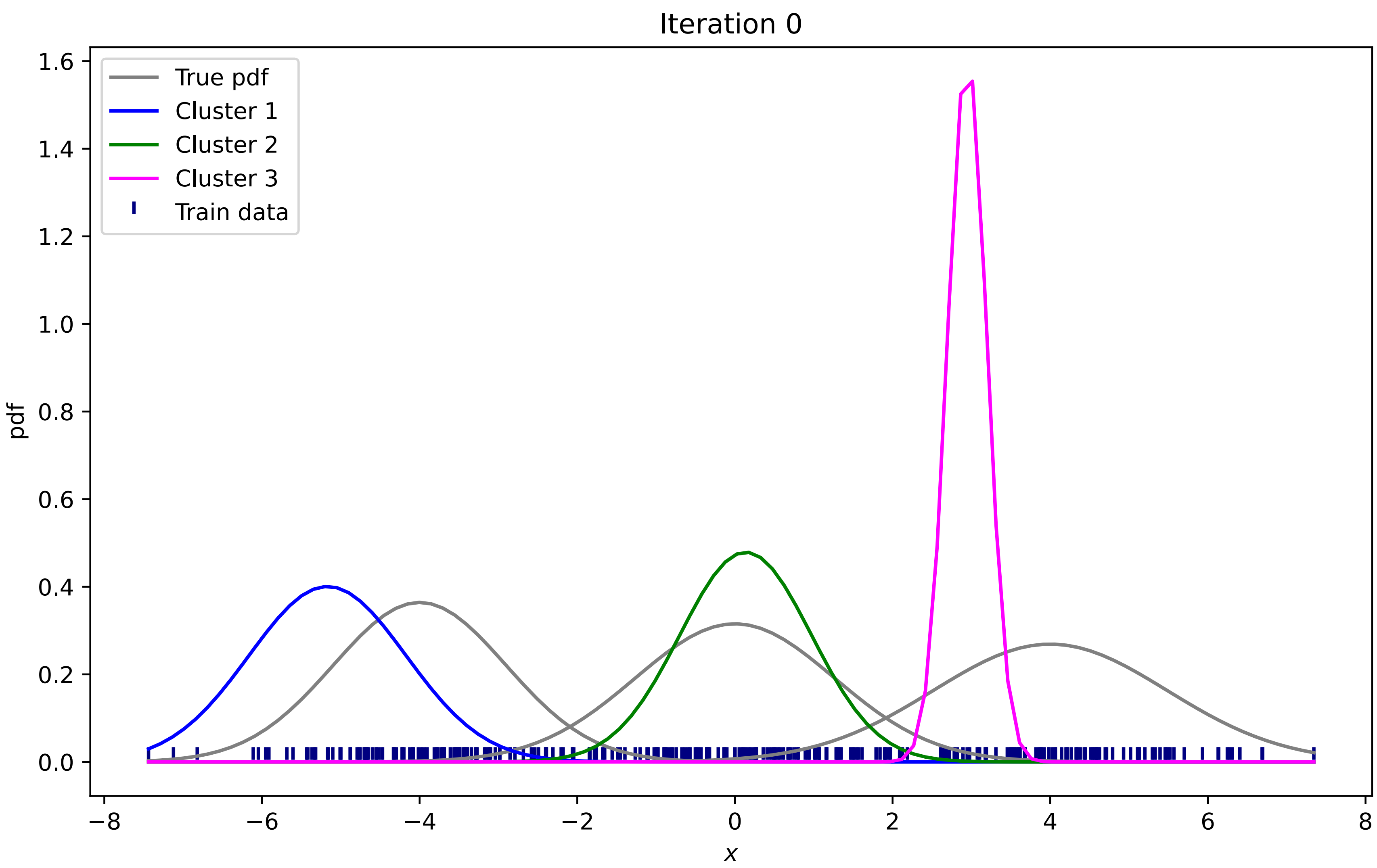
In [90]:
```
# define the number of clusters to be learned
k = 3
weights = np.ones((k)) / k
means = np.random.choice(X, k)
variances = np.random.random_sample(size=k)
print(means, variances)
```

```
[-5.16860852  0.13606389  2.94698077] [0.99164432 0.69381628 0.0612628 ]
```

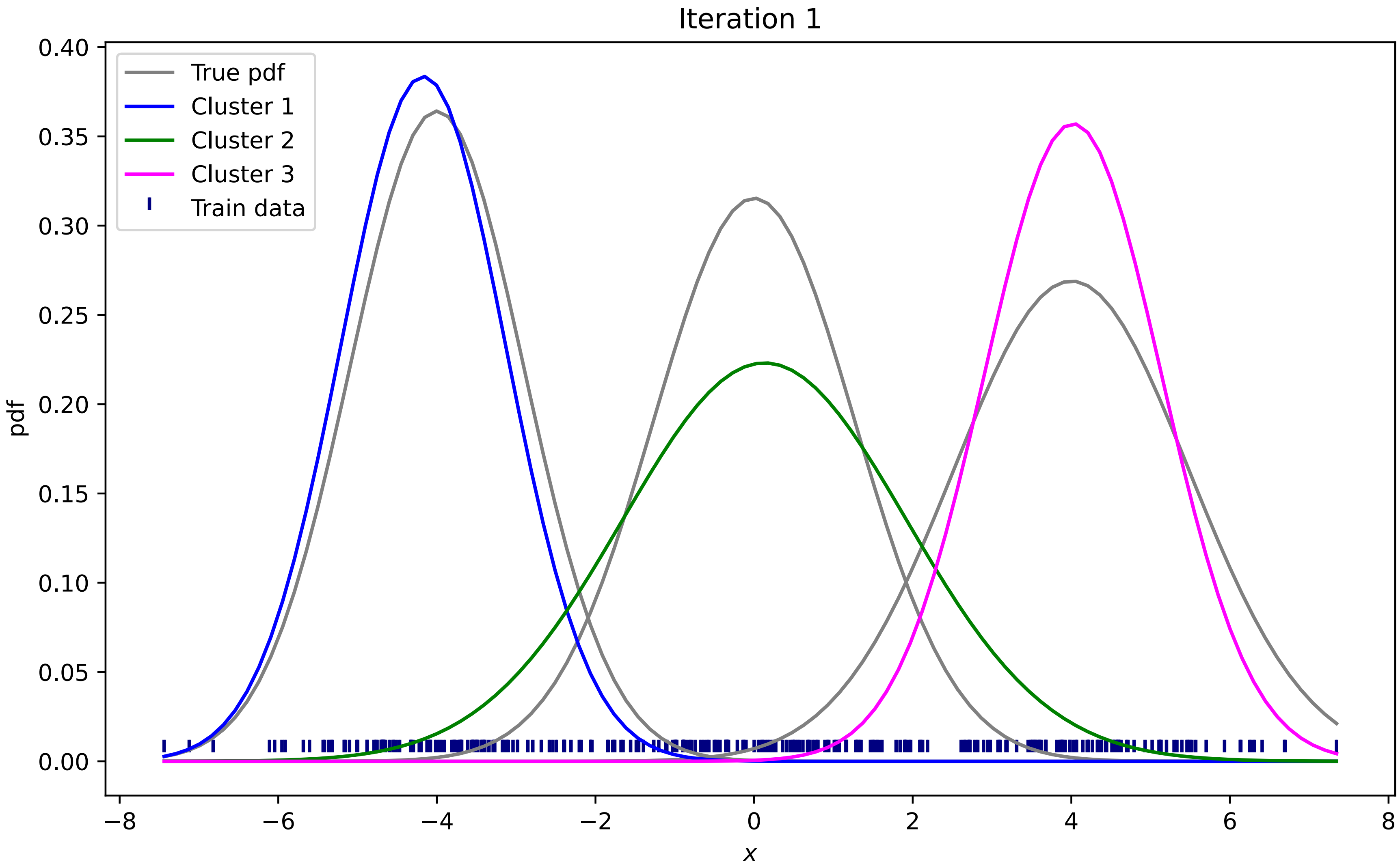
Actual implementation of EM

In [91]:

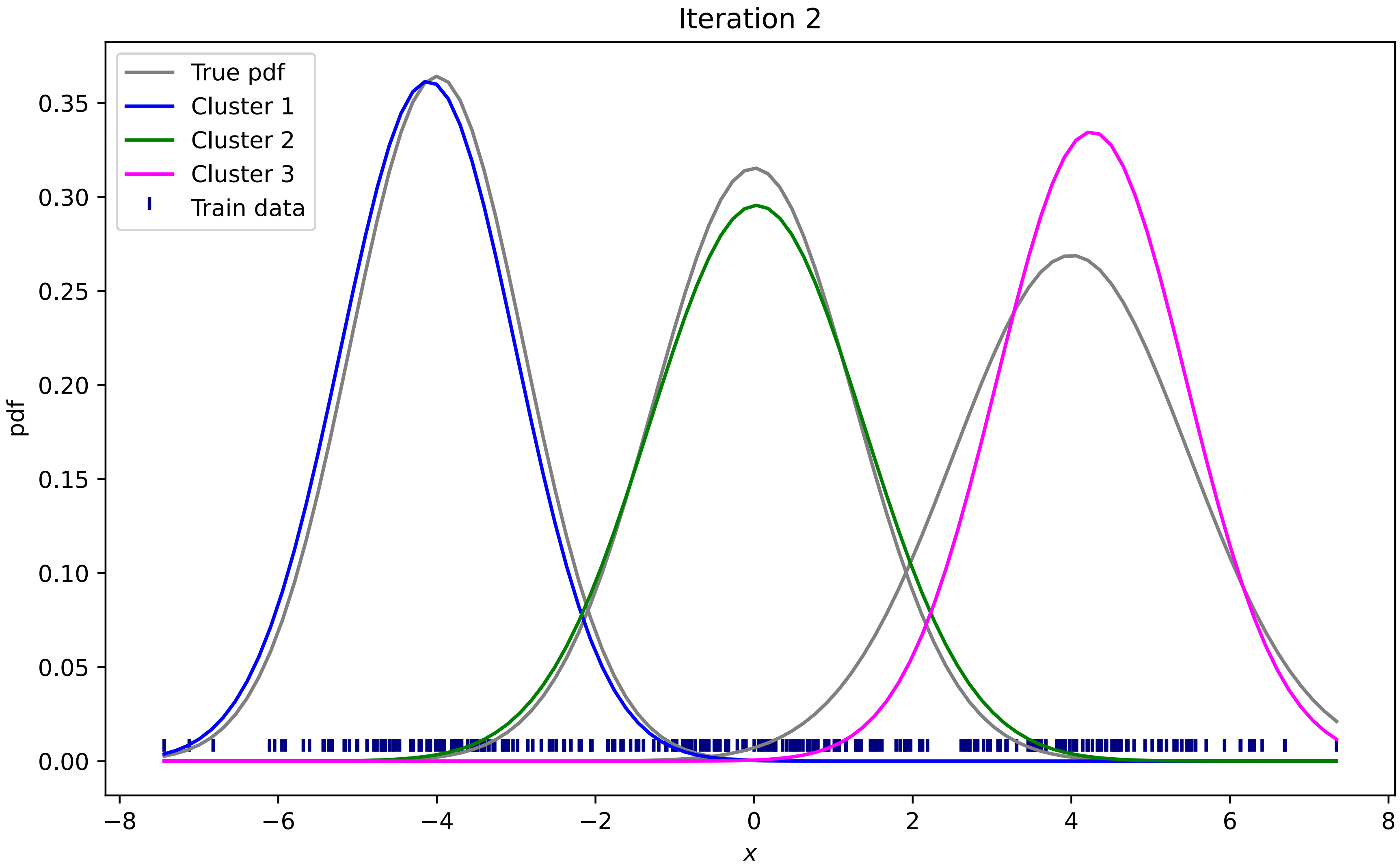
```
eps=1e-8
for step in range(10):

  if step % 1 == 0:
    plt.figure(figsize=(10,6))
    axes = plt.gca()
    plt.xlabel("$x$")
    plt.ylabel("pdf")
    plt.title("Iteration {}".format(step))
    plt.scatter(X, [0.005] * len(X), color='navy', s=30, marker=2, label="Train data")

    plt.plot(bins, pdf(bins, mu1, sigma1), color='grey', label="True pdf")
    plt.plot(bins, pdf(bins, mu2, sigma2), color='grey')
    plt.plot(bins, pdf(bins, mu3, sigma3), color='grey')

    plt.plot(bins, pdf(bins, means[0], variances[0]), color='blue', label="Cluster 1")
    plt.plot(bins, pdf(bins, means[1], variances[1]), color='green', label="Cluster 2")
    plt.plot(bins, pdf(bins, means[2], variances[2]), color='magenta', label="Cluster 3")

    plt.legend(loc='upper left')

    plt.savefig("img_{0:02d}".format(step), bbox_inches='tight')
    plt.show()

  # calculate the maximum likelihood of each observation xi
  likelihood = np.empty((300,0))


  # Expectation step
  for j in range(k):
    Pdf = pdf(X.reshape(-1,1), means[j], np.sqrt(variances[j]))
    likelihood = np.hstack((likelihood,Pdf))
  # likelihood = np.array(likelihood)
  print('likelihood',likelihood.shape)

  # b = []
  # Maximization step
  # Enter your code here 2
  a = likelihood*weights
  Delimiter = np.sum(a,axis = 1).reshape(-1,1)
  b = a/Delimiter
  means = (X.reshape(1,-1)@b/np.sum(b,axis = 0))[0]
  variances = np.sum(((X.reshape(-1,1)- means)**2)*b,axis = 0)/np.sum(b,axis = 0)
  weights = np.sum(b,axis = 0)/300
  print("means: ",means)
  print("variances: ",variances)
  print("weights: ",weights)
```



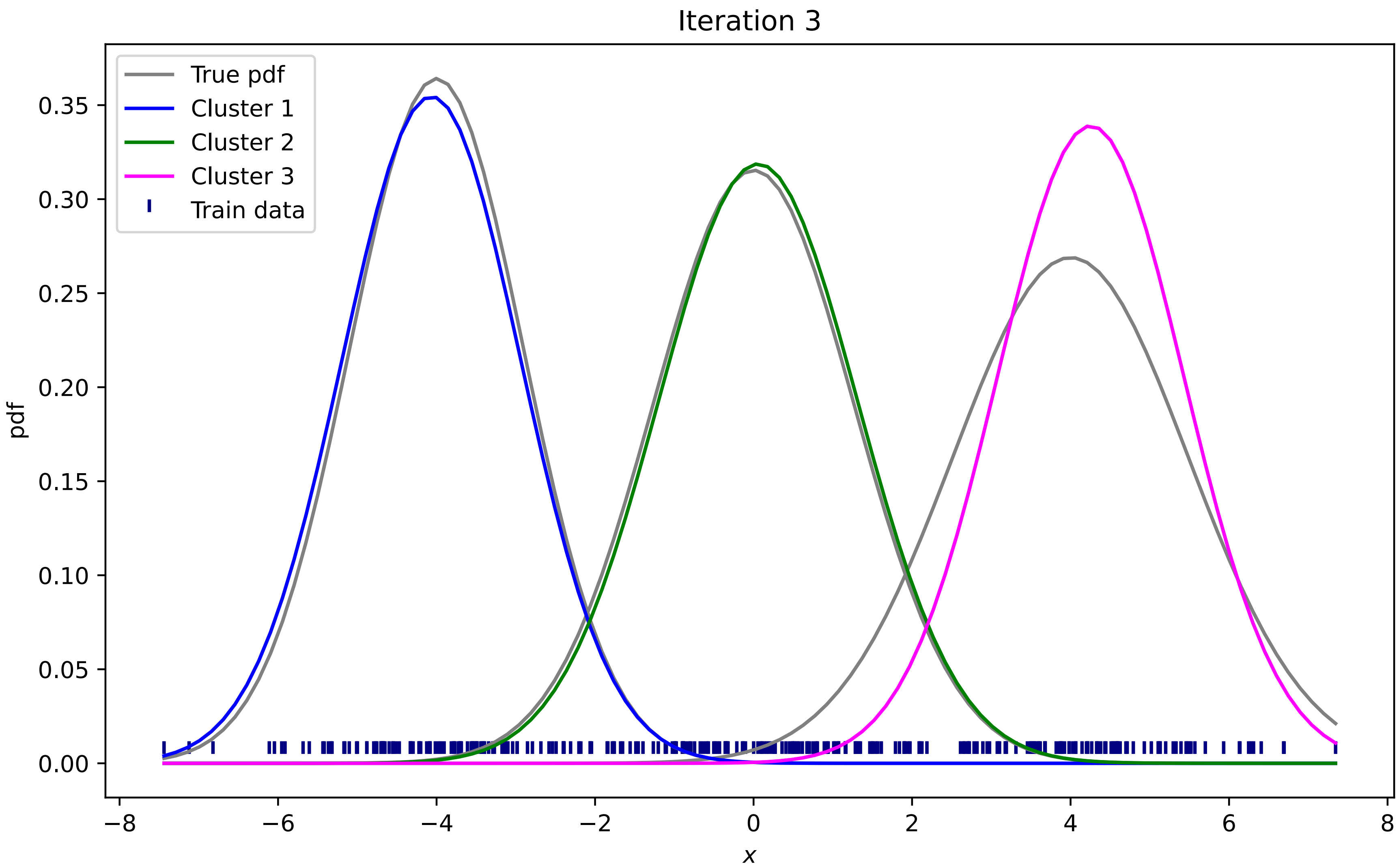

```
likelihood (300, 3)
means:  [-4.17187097  0.13129718  4.0200934 ]
variances:  [1.08147581 3.19699756 1.2478432 ]
weights:  [0.31893197 0.36723472 0.31383331]
```



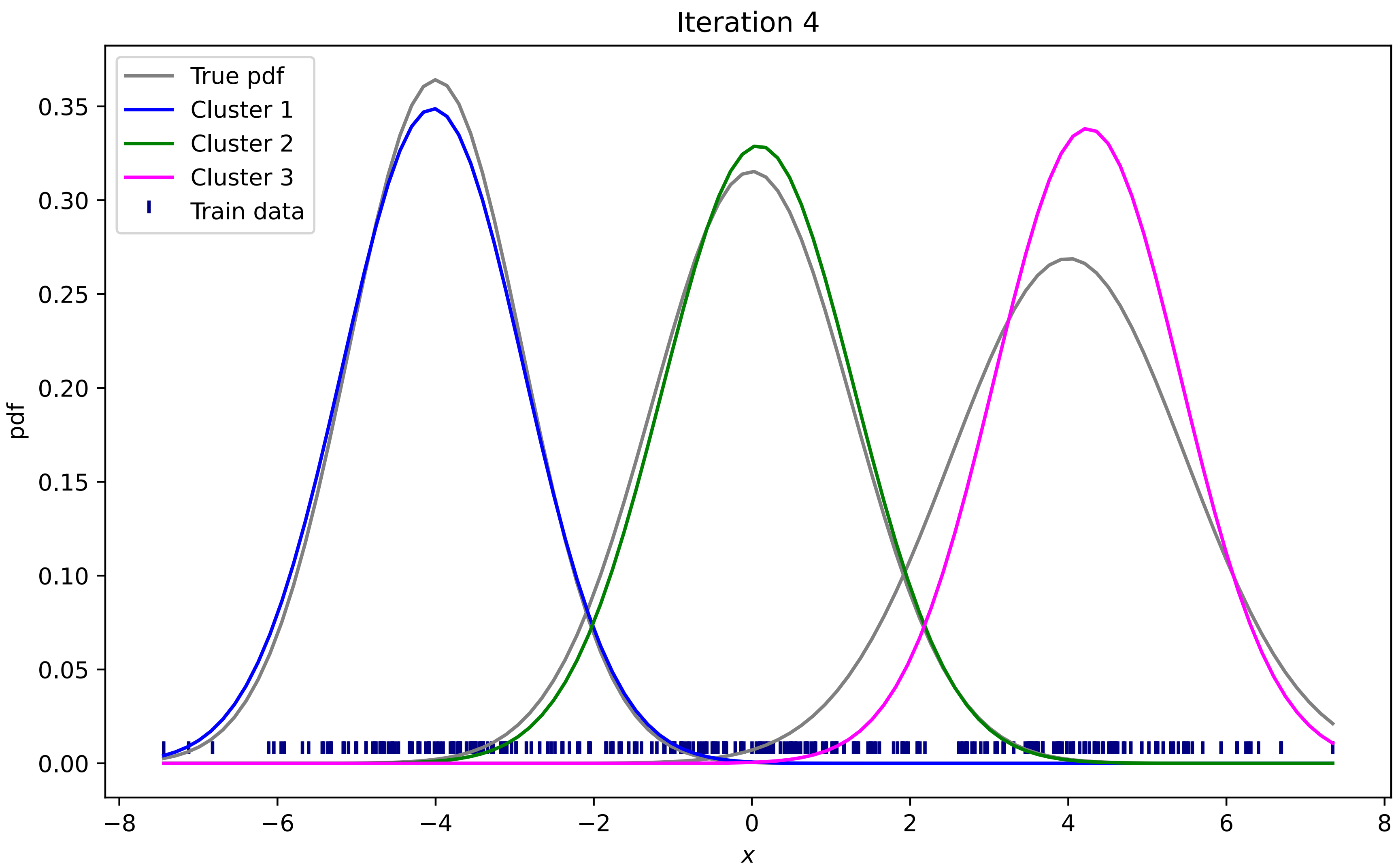

```
likelihood (300, 3)
means:  [-4.10872796  0.03154118  4.25508907]
variances:  [1.21739936 1.82147395 1.42109413]
weights:  [0.32517156 0.36843446 0.30639398]
```



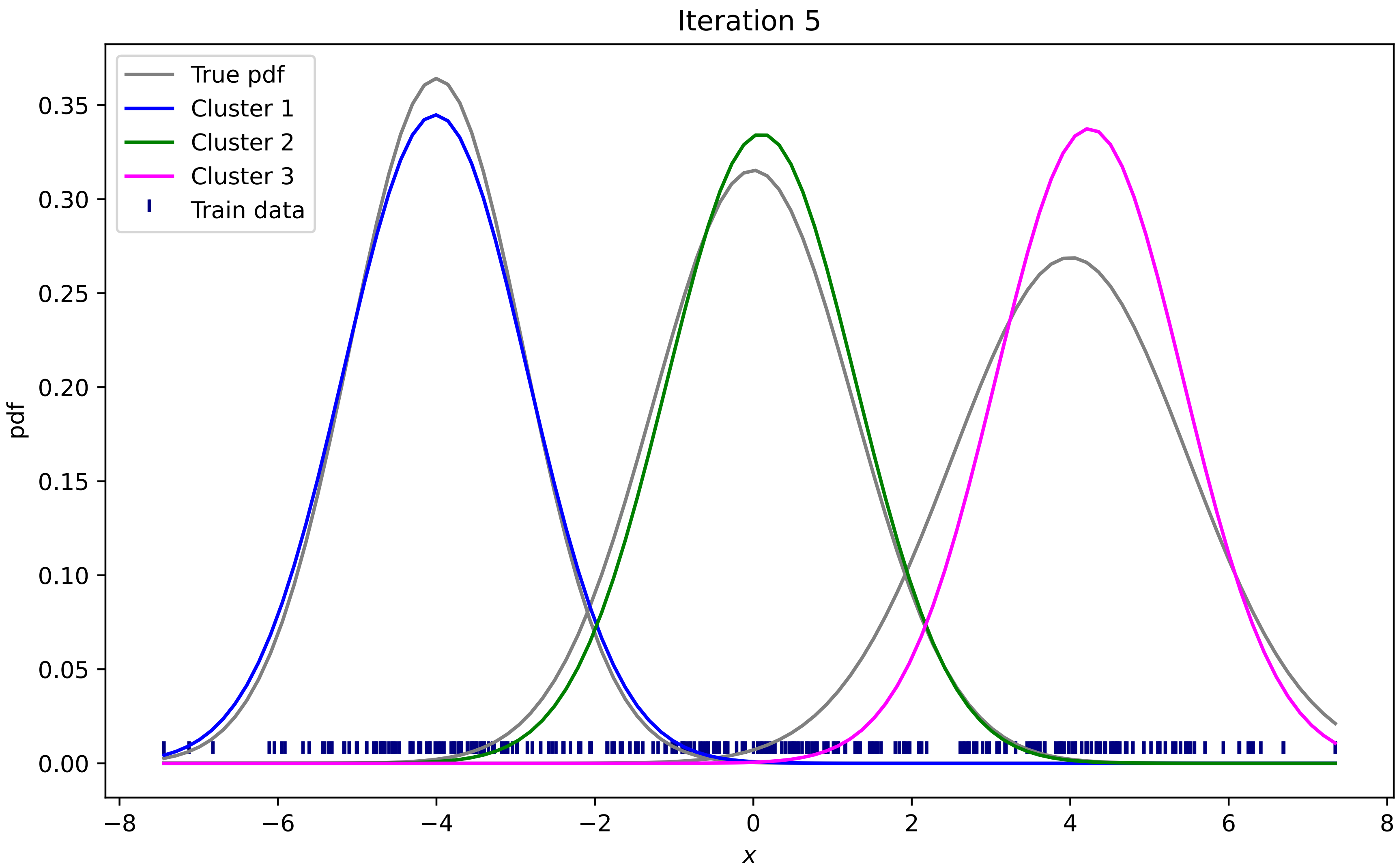

```
likelihood (300, 3)
means:  [-4.06564106  0.0566804   4.25328061]
variances:  [1.26579359 1.56647465 1.38497272]
weights:  [0.3341808  0.35598904 0.30983016]
```



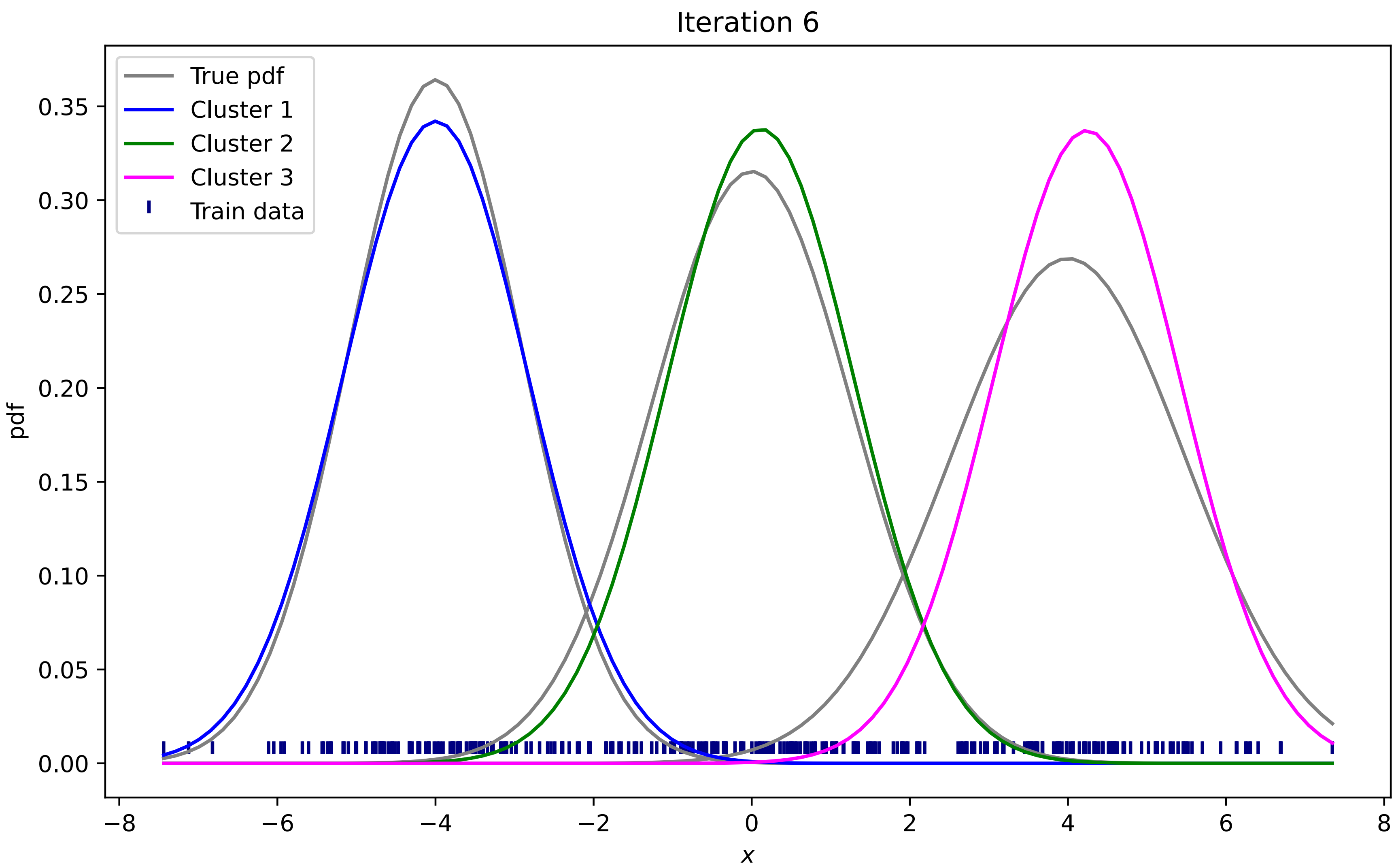

```
likelihood (300, 3)
means:  [-4.03418774  0.08217868  4.2453569 ]
variances:  [1.30773111 1.46961666 1.3909682 ]
weights:  [0.34000701 0.34851753 0.31147545]
```



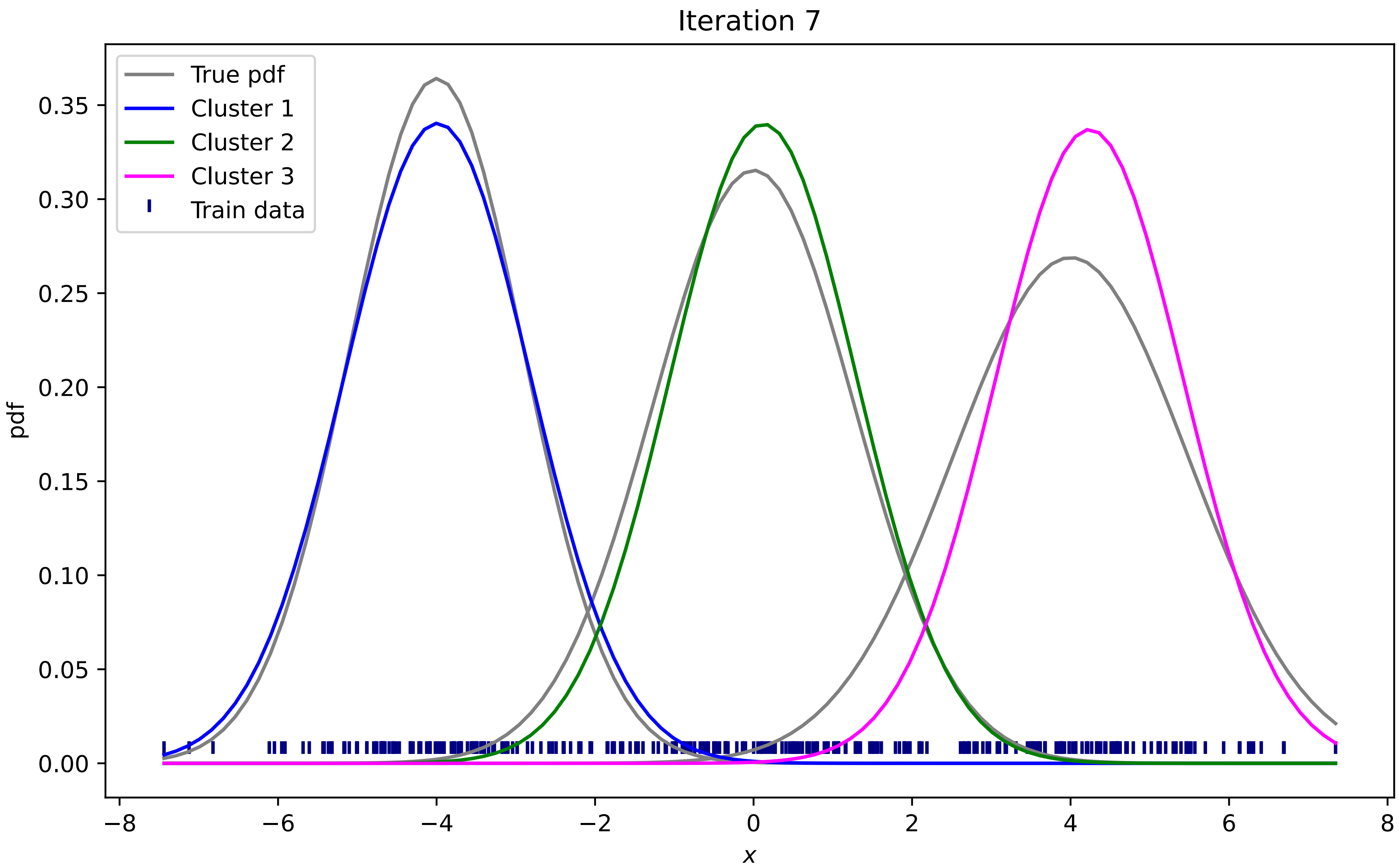

```
likelihood (300, 3)
means:  [-4.01298117  0.10060161  4.24084261]
variances:  [1.33854001 1.42083011 1.39713468]
weights:  [0.34374127 0.34402579 0.31223294]
```



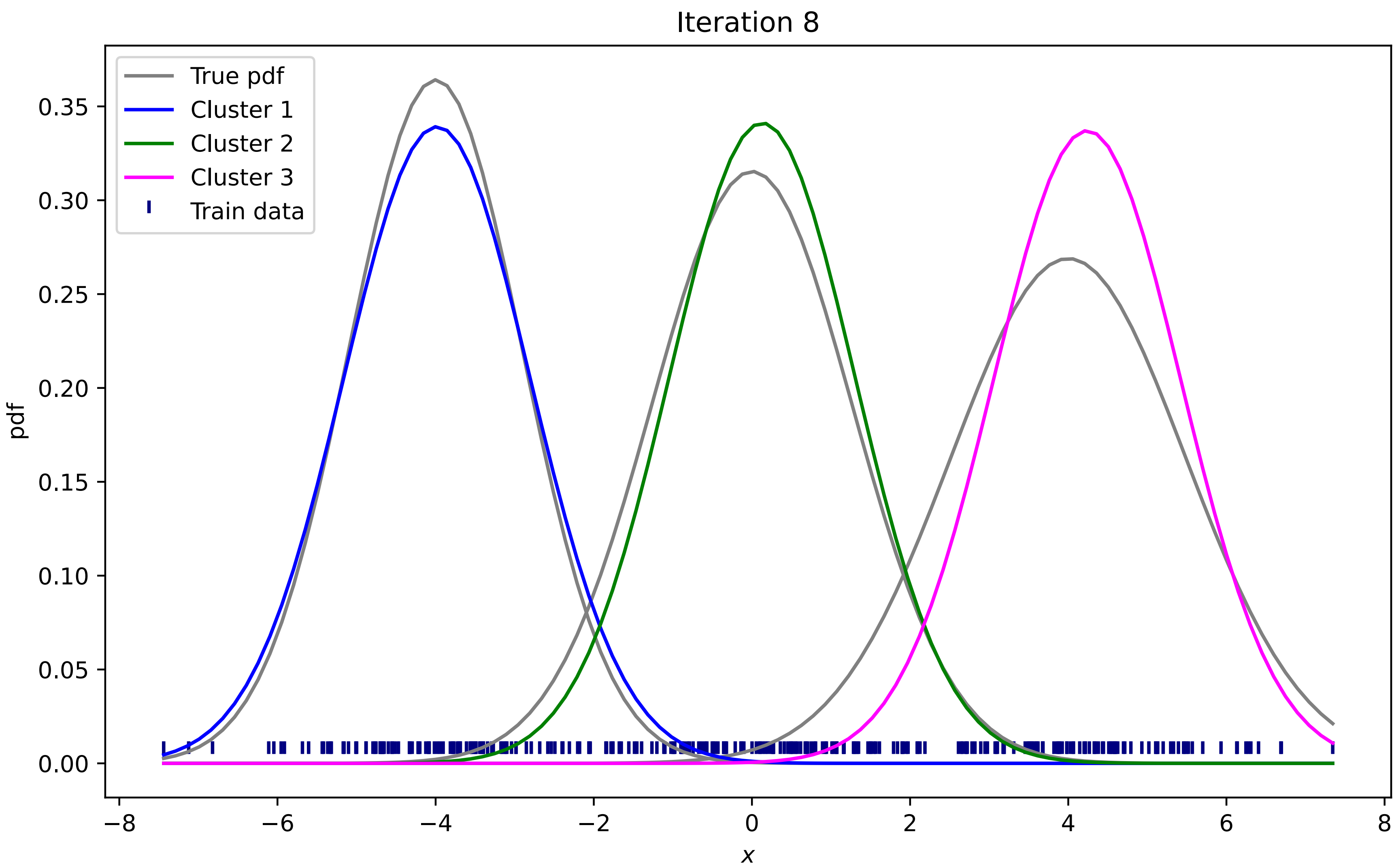

```
likelihood (300, 3)
means:  [-3.99901235  0.11343548  4.23890387]
variances:  [1.3596408  1.39326567 1.40002673]
weights:  [0.34614828 0.34130677 0.31254495]
```



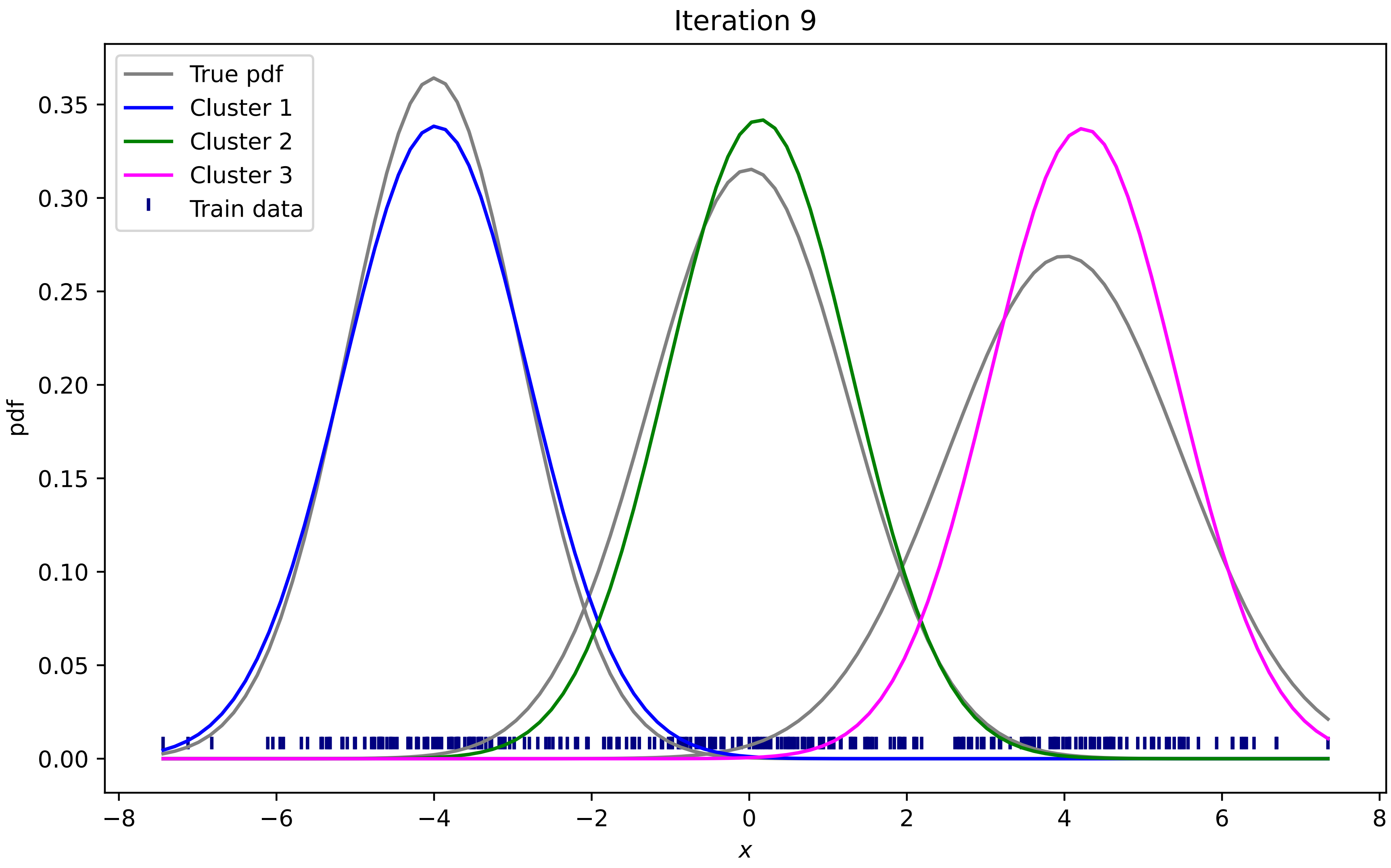

```
likelihood (300, 3)
means:  [-3.98985141  0.1223494   4.23834836]
variances:  [1.3737631  1.37700041 1.40084175]
weights:  [0.34770982 0.33965252 0.31263765]
```




```
likelihood (300, 3)
means:  [-3.98383508  0.12854391  4.23844037]
variances:  [1.38314827 1.36721716 1.4006491 ]
weights:  [0.348729   0.33864147 0.31262952]
```




```
likelihood (300, 3)
means:  [-3.97987145  0.13284944  4.23878688]
variances:  [1.38937716 1.36126811 1.40006091]
weights:  [0.34939785 0.3380208  0.31258135]
```




```
likelihood (300, 3)
means:  [-3.97725066  0.13584095  4.23919365]
variances:  [1.39351551 1.35762205 1.39938458]
weights:  [0.34983897 0.33763809 0.31252294]
```

(b) GMM with sklearn library

In [92]:

```
from sklearn.mixture import GaussianMixture
gm = GaussianMixture(n_components=3, random_state=0).fit(X.reshape(-1,1))
means = gm.means_
variances = gm.covariances_
print("means: ",means)
print("variances: ",variances)
```

```
means:  [[ 4.24667335]
 [-3.95912856]
 [ 0.16076671]]
variances:  [[[1.43473826]]

 [[1.46749592]]

 [[1.55910445]]]
```

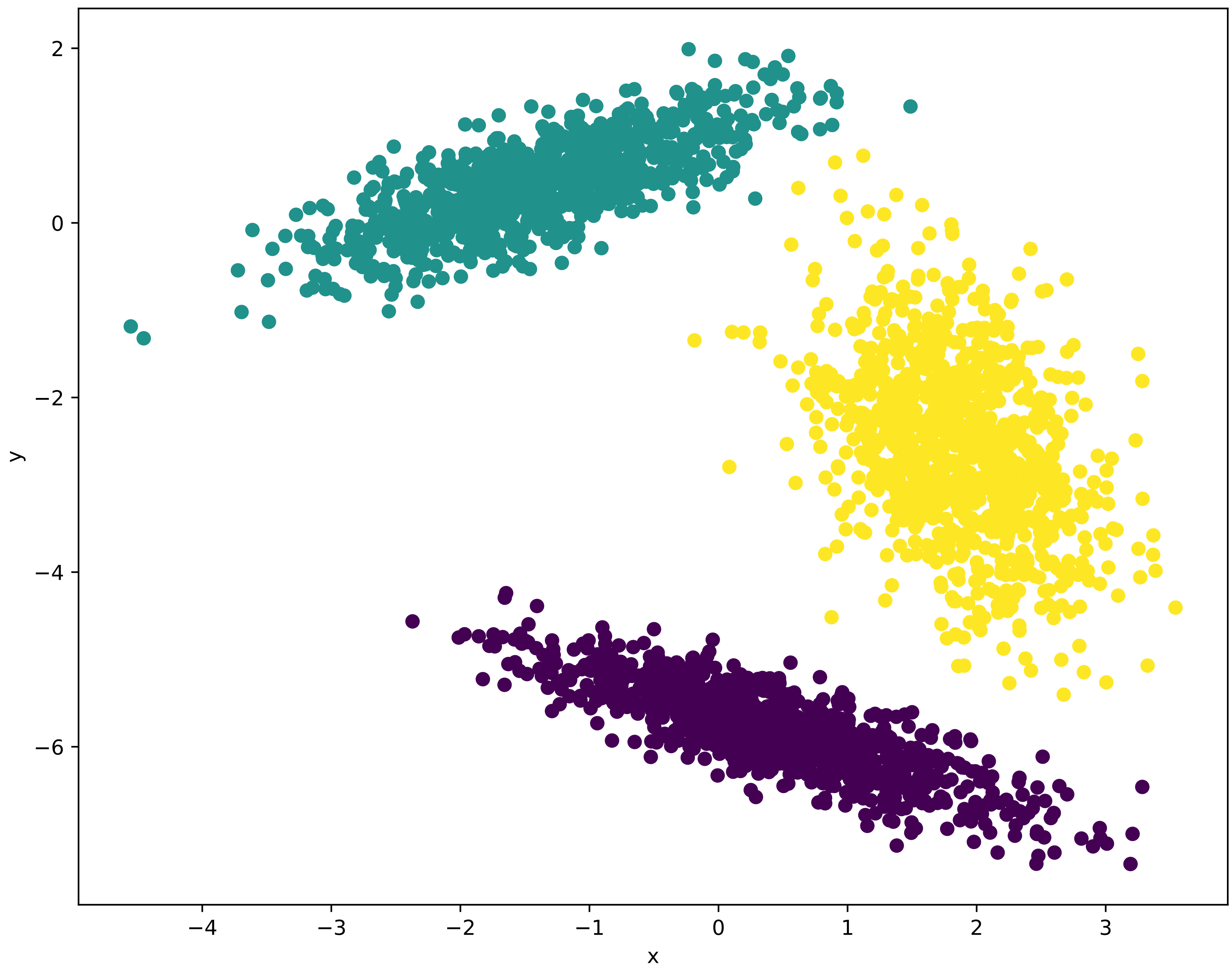
I get similar values compared to a. If we have more iterations in (a), we can get more robust values.

Q2: Clustering Algorithms

I. GMM

In [103…

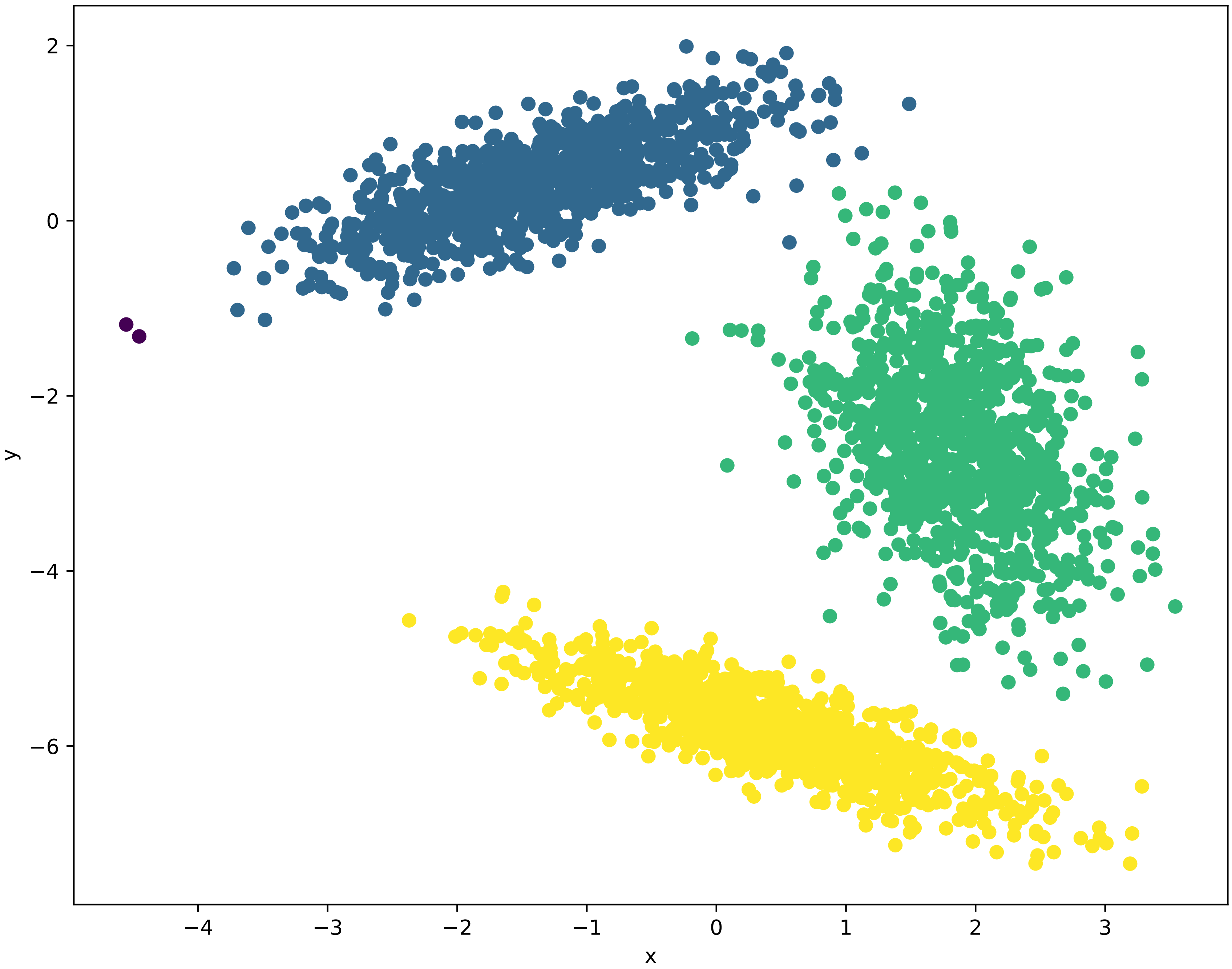
```
X_ = np.loadtxt("threeblobs.txt")
gm = GaussianMixture(n_components=3, random_state=0).fit(X_)
labels = gm.predict(X_)
# Plotting
plt.figure(figsize=(10, 8))
plt.scatter(X_[:,0],X_[:,1],c = labels)
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```



II. DBSCAN

In [118…

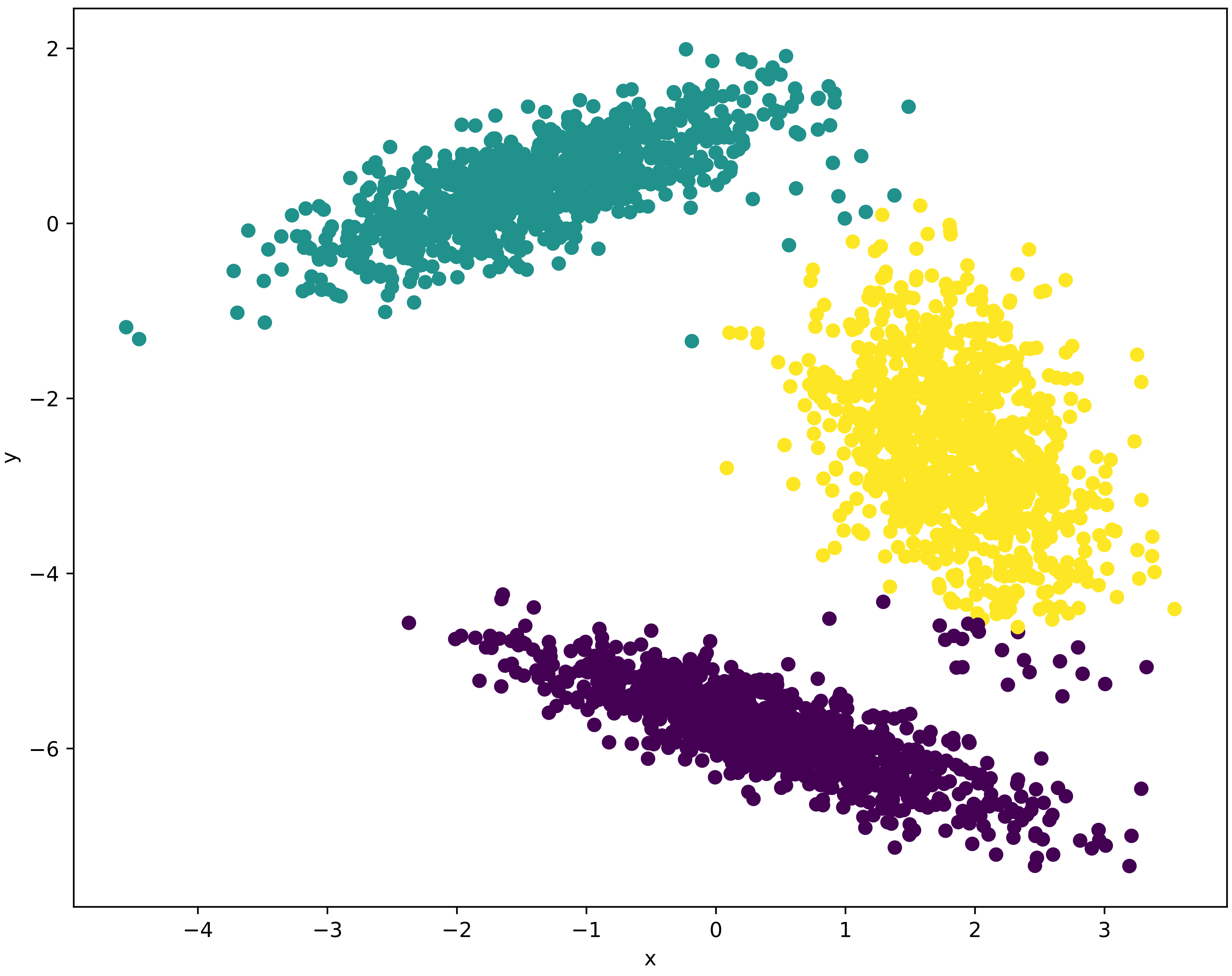
```
from sklearn.cluster import DBSCAN
clustering = DBSCAN(eps=0.6, min_samples=10).fit(X_)
labels = clustering.labels_
# Plotting
plt.figure(figsize=(10, 8))
plt.scatter(X_[:,0],X_[:,1],c = labels)
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```



III. KMeans

In [119…

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=3, random_state=0).fit(X_)
labels = kmeans.labels_
# Plotting
plt.figure(figsize=(10, 8))
plt.scatter(X_[:,0],X_[:,1],c = labels)
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```


2
0
−2
−4
−6
y
−4
−3
−2
−1
0
1
2
3
x