Lab 5 : Unsupervised Learning - Searching for a new galaxy

The goal of this lab is to:

- Work on several examples of unsupervised learning.
- Use K-means clustering to perform classification.
- Use Gaussian mixture modeling to for classification and density estimation.
- Determine the optimal number of clusters to use for a problem.
- Identify the remants of a dwarf galaxy that merged with the Milky Way using velocity and metal abundance measurements of stars from the Gaia survey.

Created by: Tuan Do & Bernie Boscoe Last update: Tuan Do

In [1]:
```
# put your imports here
import numpy as np
import pylab as plt
import pandas as pd
```

Part 1 - K-Means Clustering

K-means clustering is a simple clustering algorithm, which contains just a few steps:

1. Guess some cluster centers
2. Repeat the following until converged
    - Expectation step – assign points to the nearest cluster center
    - Maximization step – set the cluster centers to the mean.

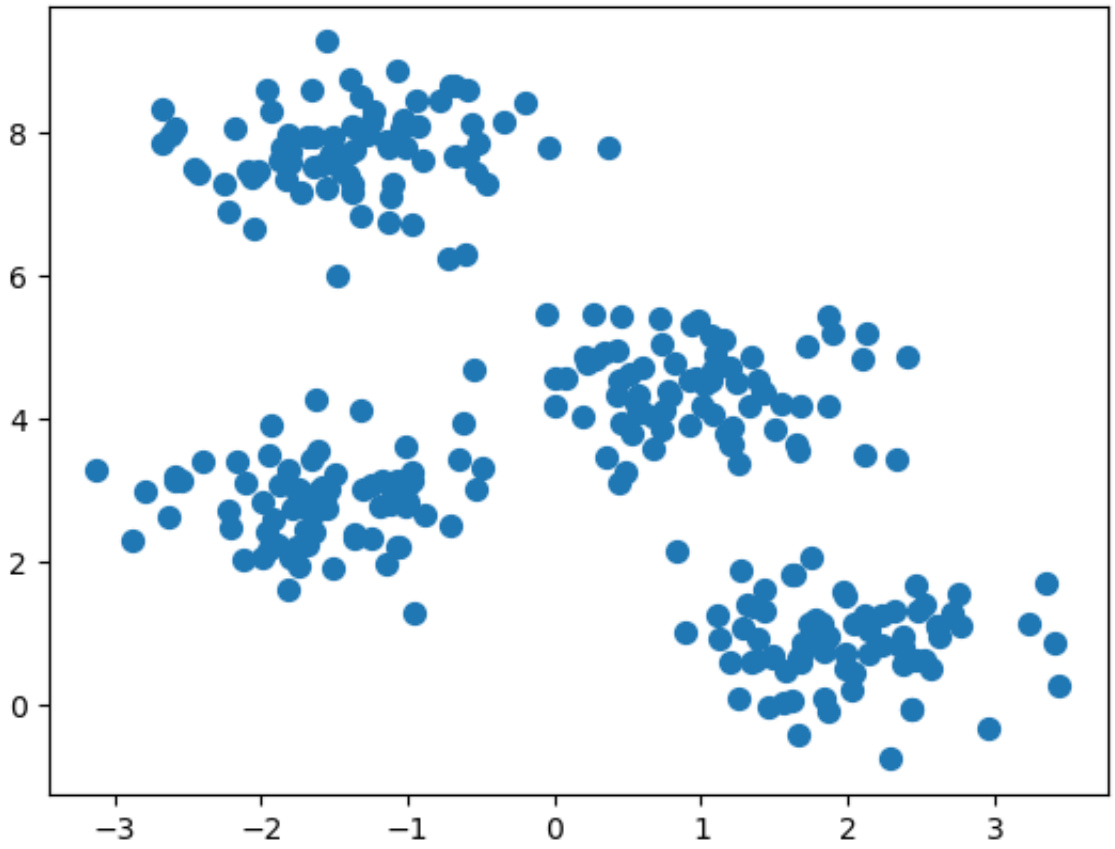
Convergence is defined as when the centroid of the cluster does not change after Step 2.

In this part of the lab, you will generate random clusters and practice using K-means clustering to learn about the model and its performance.

Part 1.1 - Generating random clusters

We will generate first a random set of clusters with 2 features.

In [2]:
```
from sklearn.datasets import make_blobs
X, y_true = make_blobs(n_samples=300, centers=4,
                       cluster_std=0.60, random_state=0)
plt.scatter(X[:, 0], X[:, 1], s=50);
```



Part 1.2 - Building and training a K-means classifier

Question 1

(5 pts)

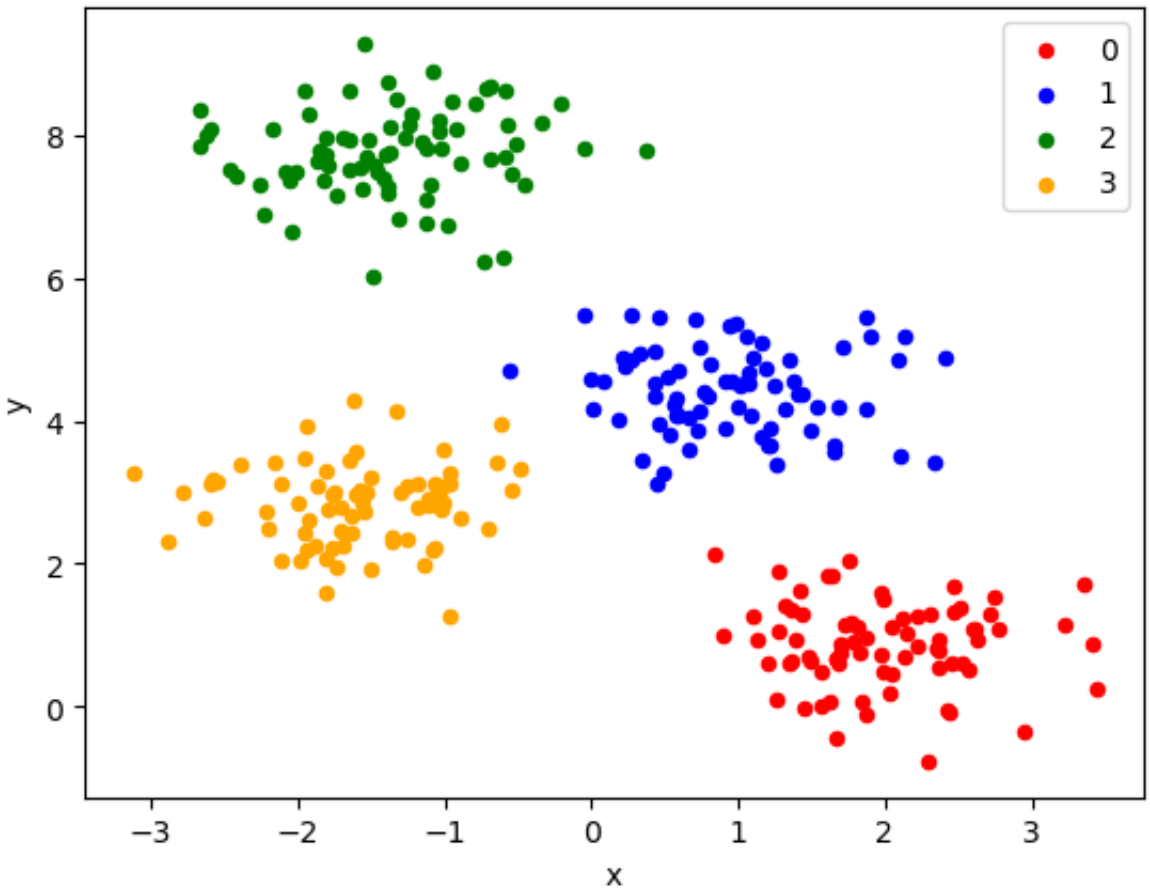
Now, read the K-means documentation in sklearn (https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html) and create a K-means classifier and train it.

In [3]:
```
# put code here
from sklearn.cluster import KMeans
clusterer = KMeans(n_clusters=4, random_state=42, n_init='auto').fit(X)
labels = clusterer.predict(X)
print(labels)
```

```
[0 2 1 2 0 0 3 1 2 2 3 2 1 2 0 1 1 0 3 3 0 0 1 3 3 1 0 1 3 1 2 2 1 2 2 2 2
 2 3 0 1 3 1 1 3 3 2 3 2 0 3 0 2 0 0 3 2 3 2 0 2 1 2 3 3 3 2 0 2 3 1 3 2 3
 3 2 3 1 0 2 0 1 0 0 2 1 0 1 2 2 1 0 2 3 3 1 0 0 1 3 2 0 2 0 1 0 0 1 2 1 3
 3 0 2 0 1 2 0 0 1 3 0 3 0 0 0 0 3 0 3 2 3 3 0 2 3 3 2 1 2 2 3 1 3 1 3 2 1
 2 2 2 1 2 1 0 3 2 3 0 1 2 1 1 0 1 3 3 1 0 1 1 2 0 1 3 2 0 0 1 3 0 1 3 3 1
 1 1 1 0 2 1 3 1 1 3 3 3 1 3 2 1 3 0 3 1 2 3 2 1 2 1 3 1 1 2 3 3 0 0 1 2 0
 0 3 0 3 1 2 2 1 1 2 1 0 3 1 0 3 2 3 0 1 0 2 2 2 2 3 3 2 1 3 0 1 3 3 3 0 0
 2 1 1 3 0 2 3 1 2 1 0 0 3 3 1 0 0 0 1 2 2 0 0 1 0 0 0 2 3 2 1 0 0 2 2 2 0
 0 1 2 3]
```

Visualize your fit - plot the different detected clusters

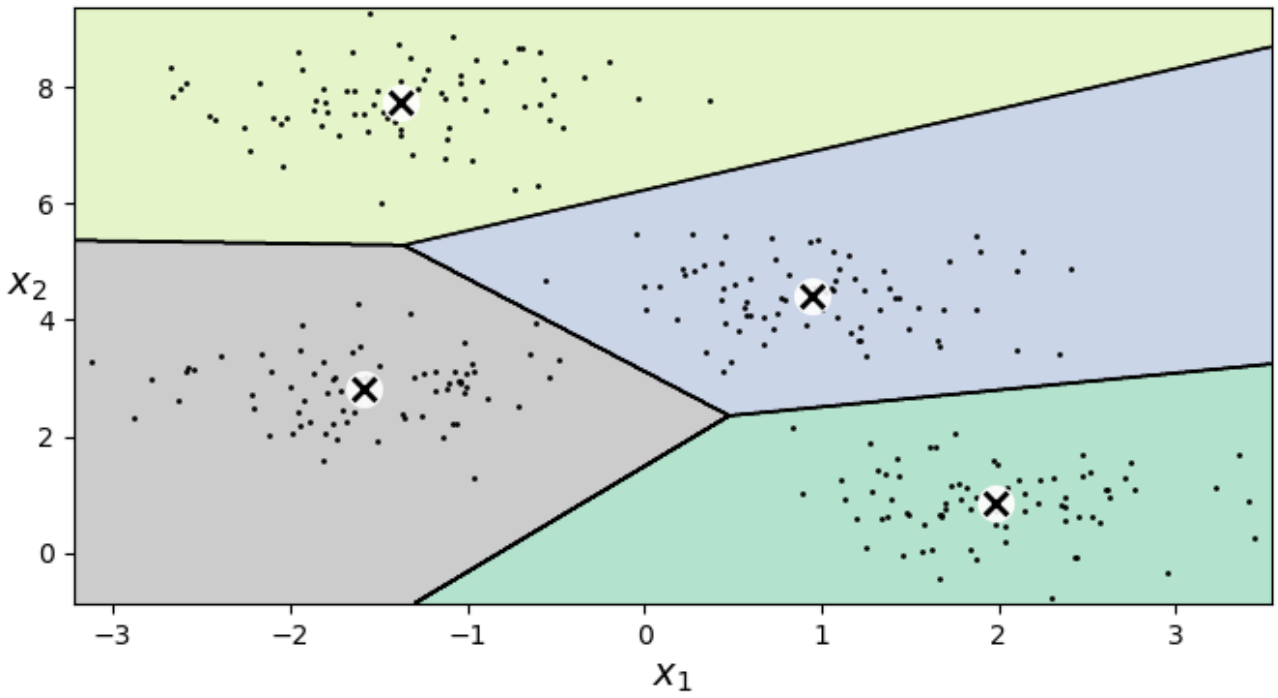
In [4]:
```
# enter code here
# WRITE CODE THAT PLOTS THE SCATTER PLOT BUT WITH COLOR DETERMINED BY PREDIC
import matplotlib.pyplot as plt

df = pd.DataFrame(dict(x=X[:,0], y=X[:,1], label=clusterer.predict(X)))
colors = {0:'red', 1:'blue', 2:'green', 3:'orange'}
fig, ax = plt.subplots()
grouped = df.groupby('label')
for key, group in grouped:
    group.plot(ax=ax, kind='scatter', x='x', y='y', label=key, color=colors[
plt.show()
```



Decision Boundaries below is the code to help you visualize the decision boundaries of your model. Run it and make a plot for your clusters.

In [5]:
```
#  Code to plot decision boundaries for K-means

def plot_data(X):
    plt.plot(X[:, 0], X[:, 1], 'k.', markersize=2)

def plot_centroids(centroids, weights=None, circle_color='w', cross_color='k
    if weights is not None:
        centroids = centroids[weights > weights.max() / 10]
    plt.scatter(centroids[:, 0], centroids[:, 1],
                marker='o', s=35, linewidths=8,
                color=circle_color, zorder=10, alpha=0.9)
    plt.scatter(centroids[:, 0], centroids[:, 1],
                marker='x', s=2, linewidths=12,
                color=cross_color, zorder=11, alpha=1)

def plot_decision_boundaries(clusterer, X, resolution=1000, show_centroids=T
                             show_xlabels=True, show_ylabels=True):
    mins = X.min(axis=0) - 0.1
    maxs = X.max(axis=0) + 0.1
    xx, yy = np.meshgrid(np.linspace(mins[0], maxs[0], resolution),
                         np.linspace(mins[1], maxs[1], resolution))
    Z = clusterer.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.contourf(Z, extent=(mins[0], maxs[0], mins[1], maxs[1]),
                 cmap="Pastel2")
    plt.contour(Z, extent=(mins[0], maxs[0], mins[1], maxs[1]),
                linewidths=1, colors='k')
    plot_data(X)
    if show_centroids:
        plot_centroids(clusterer.cluster_centers_)

    if show_xlabels:
        plt.xlabel("$x_1$", fontsize=14)
    else:
        plt.tick_params(labelbottom=False)
    if show_ylabels:
        plt.ylabel("$x_2$", fontsize=14, rotation=0)
    else:
        plt.tick_params(labelleft=False)
```

In [6]:
```
# visualize the decision boundaries
plt.figure(figsize=(8, 4))
plot_decision_boundaries(clusterer, X)
```



Part 1.2 - fitting a different number of cluster

The number of clusters is a hyperparameter for the K-means clustering algorithm.

What will happen if we fit 3 clusters instead of 4 clusters?

Question 2

(2 pts)

Before writing code, **predict where you think the centroid of the 3 clusters will be and discuss why.** A rough estimate of the centroid is fine.

In [7]:
```
# put answer here
# I think it will split horizontally into 3 segments combinging the gray and

clusterer_3 = KMeans(n_clusters=3, random_state=42, n_init='auto').fit(X)
labels_3 = clusterer.predict(X)
```

Question 3

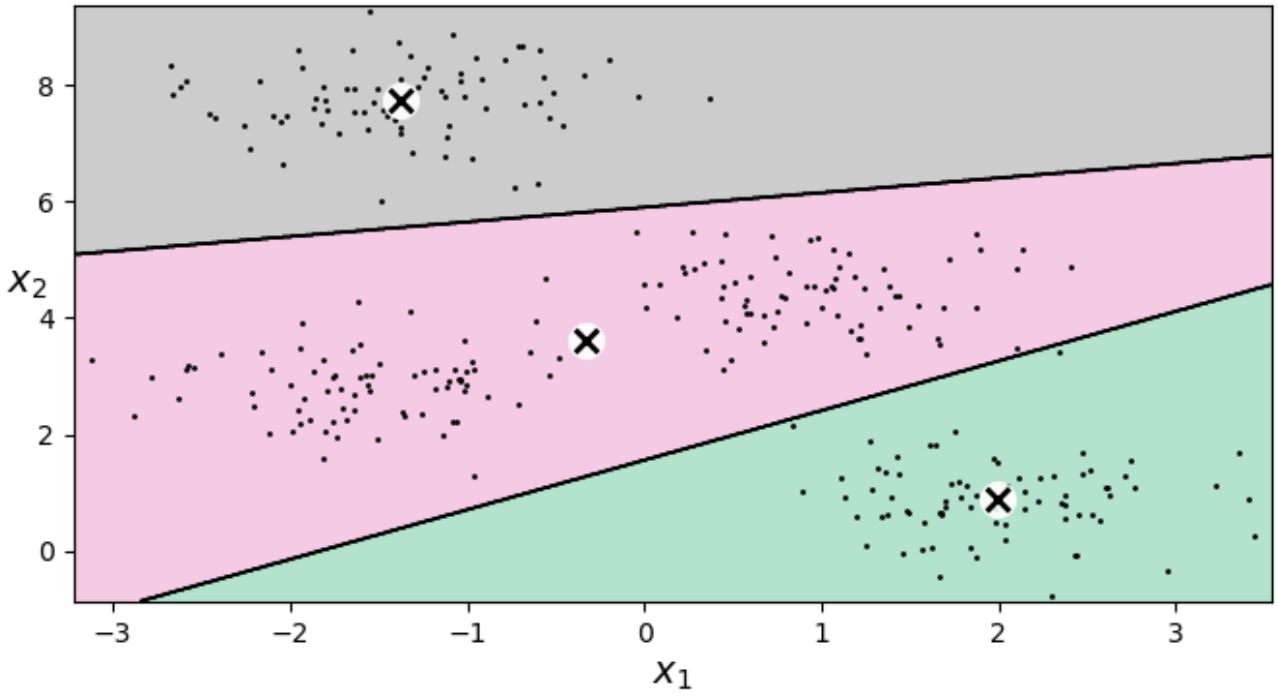
(5 pts)

Below, write the code to fit for 3 clusters instead of 4 to the same dataset as above using the K-means algorithm.

In [8]:
```
# code here
# fit above ^
labels_3 = clusterer.predict(X)
```

Now, visualize the fit with only 3 clusters

In [9]:
```
# visualize the decision boundaries
plt.figure(figsize=(8, 4))
plot_decision_boundaries(clusterer_3, X)
```



Part 1.3 - Determining the Optimal Number of Clusters for K-Means

It can be difficult to determine the correct number of clusters because the definition of a cluster can be pretty vauge. There are a few guidelines we methods that we can use though to help us. We will follow the two methods used in Hands-On Machine Learning 2nd Edition:

1. Look for the 'elbow' in the plot of the number of clusters vs. interia.
2. Look at the silhouette score as a function of the number of clusters.

Part 1.3a - Plot the intertia as a function of number of clusters

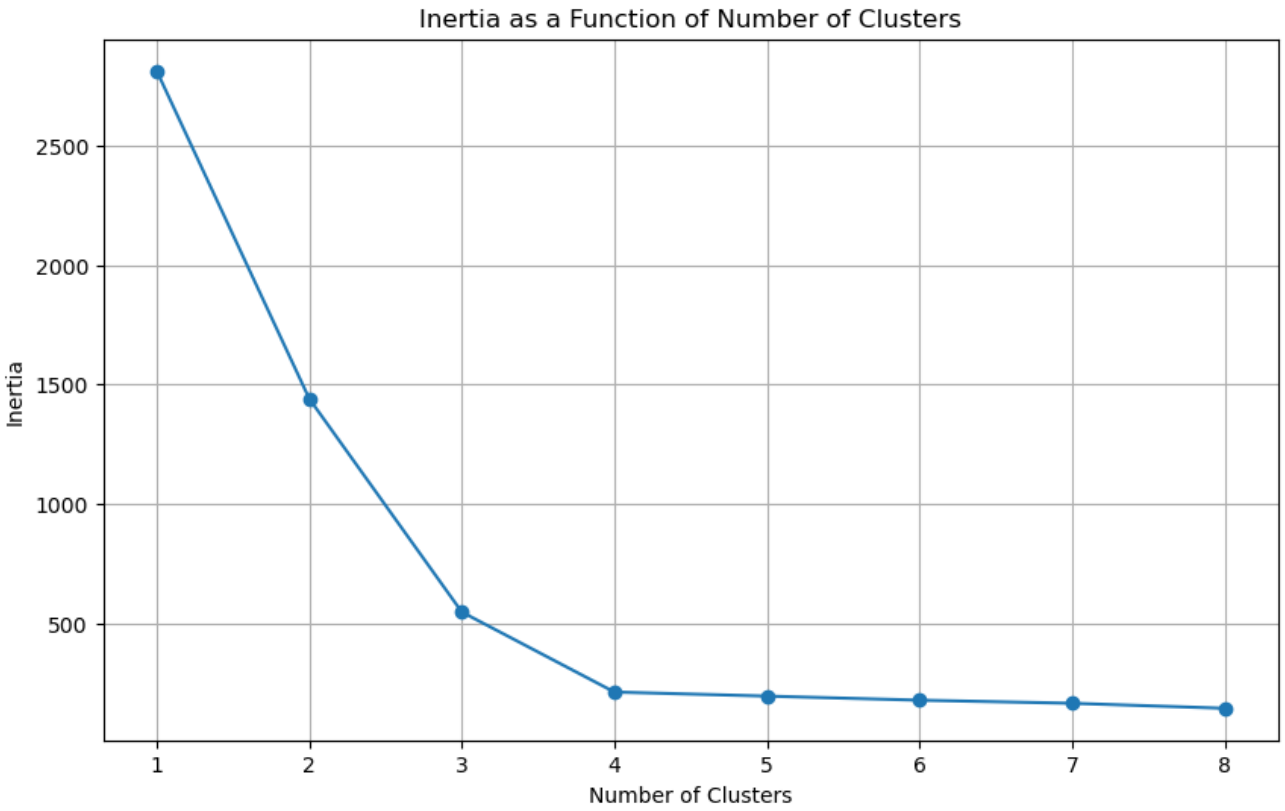
Question 4

(2 pts)

Once the model is trained the kmeans object from sci-kit learn has an `inertia_` attribute that measures the inertia of the fit (see https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html).

Make a plot of the interia as a function of number of clusters (from 1 to 8 clusters)

In [10]:
```
# your code here

# Calculating inertia for a range of k values
k_range = range(1, 9)
inertias = []

for k in k_range:
    clusterer_k = KMeans(n_clusters=k, random_state=0, n_init='auto').fit(X)
    inertias.append(clusterer_k.inertia_)

# Plotting inertia as a function of number of clusters
plt.figure(figsize=(10, 6))
plt.plot(k_range, inertias, '-o')
plt.title('Inertia as a Function of Number of Clusters')
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
plt.xticks(k_range)
plt.grid(True)
plt.show()
```



In the plot above, there should be a point on the plot where it becomes flat. This point is called the elbow and is a good candidate for the optimal number of clusters in your model.

Question 5

(2 pts)

Where is the elbow for your dataset? Explain qualitatively why the curve flatens out (remember the definition of inertia).

In [11]:
```
# you answer here
```

In the plot above, the elbow is at n_clusters = 4.

Inertia measures the sum of squared distances of samples to their closest cluster center. When the number of clusters is too low, adding another cluster significantly reduces the inertia because it better captures the structure of the data, reducing the distances from points to centers. As you increase the number of clusters, each new cluster will only marginally decrease the overall inertia because it's capturing less significant groupings or it's splitting clusters that are already relatively compact.

Part 1.3b - Using the silhouette coefficient

Another method to find the optimal number of clusters is to use the silhoueet coefficient.

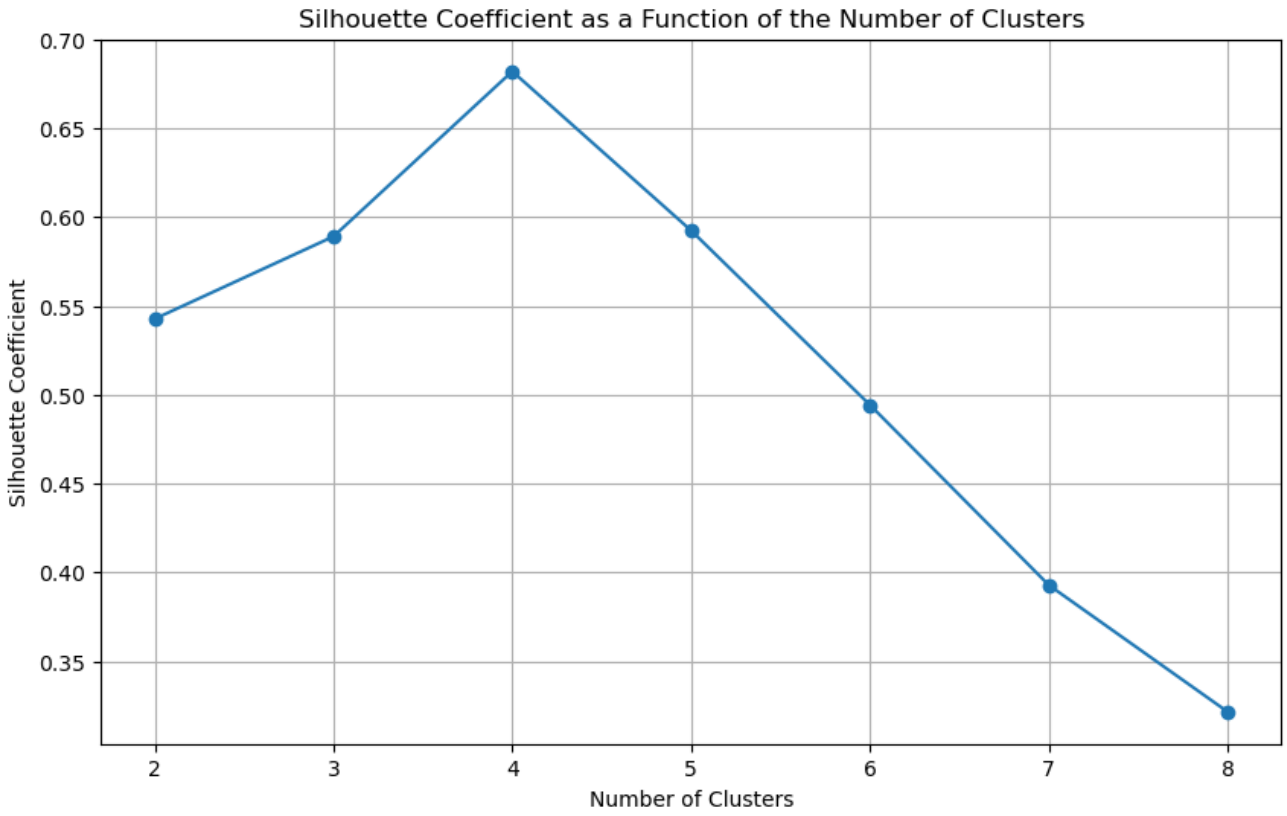
See the documentation here: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html

Question 6

(2 pts)

Plot below the silhouette coefficient for the your dataset as a function of the number of clusters (from 1 to 8)

In [12]:
```
# your code here
from sklearn.metrics import silhouette_score

# Range of clusters to evaluate
cluster_range = range(2, 9)  # Starting from 2 clusters because silhouette s

# List to store the silhouette coefficients
silhouette_coefficients = []

# Calculate silhouette score for each number of clusters
for n_clusters in cluster_range:
    clusterer_sil = KMeans(n_clusters=n_clusters, n_init=10, random_state=42
    score = silhouette_score(X, clusterer_sil.labels_)
    silhouette_coefficients.append(score)

# Plotting the silhouette coefficients as a function of the number of cluste
plt.figure(figsize=(10, 6))
plt.plot(cluster_range, silhouette_coefficients, marker='o')
plt.title('Silhouette Coefficient as a Function of the Number of Clusters')
plt.xlabel('Number of Clusters')
plt.ylabel('Silhouette Coefficient')
plt.xticks(range(2, 9))
plt.grid(True)
plt.show()
```



In the plot above, what is the best sihouette score? Explain qualitatively why the sihouette score peaks where it does.

In [13]:
```
# your answer here
```

In [14]:
```
print(max(silhouette_coefficients))
```

```
0.6819938690643478
```

The best score is 0.68 at 4 clusters.

The peak indicates that, at 4 clusters, each data point is, on average, closer to other points in its cluster than to points in other clusters. This is indicative of a good clustering where each cluster is distinct and well-separated from others.

Specifically, from the documentation on the metric:

The Silhouette Coefficient is calculated using the mean intra-cluster distance (a) and the mean nearest-cluster distance (b) for each sample. The Silhouette Coefficient for a sample is (b - a) / max(a, b).

Part 2 - Gaussian Mixture Modeling

Gaussian Mixture Modeling is a lot more flexible than K-means. It can be used for cluster, but also for modeling the probability distribution of the data points. Gaussian mixture modeling can be thought of as fitting N multi-densional Gaussian with M dimensions where M is the nubmer of features in the data.

For this part of the lab, we will explore how to use sci-kit learn's Gaussian model, how to visualize the fitted model, make predictions from the model, and determine the optimial number of clusters.

Part 2.1 - Generating data

Create the dataset by running the code below.

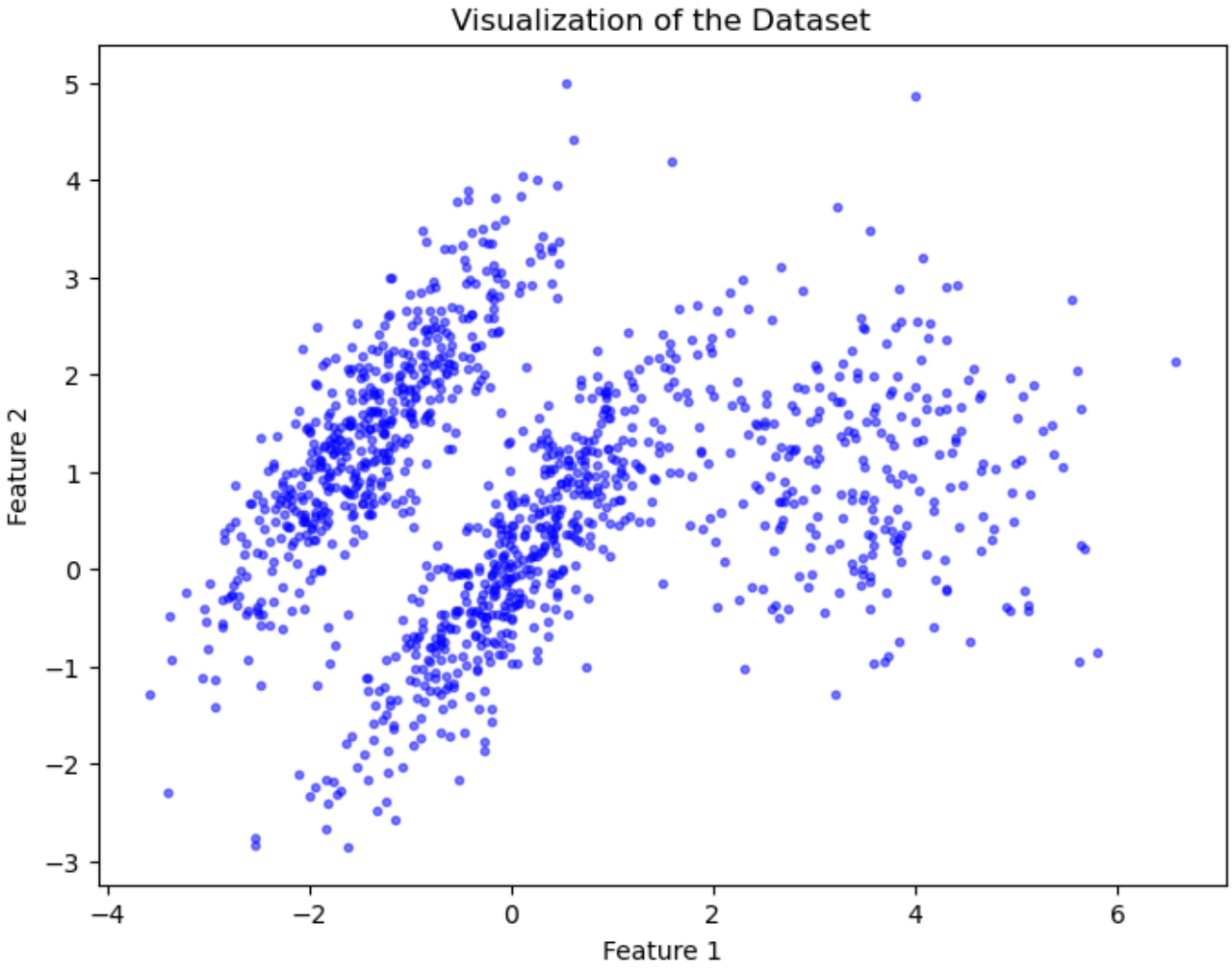
In [15]:
```
# create the dataset
X1, y1 = make_blobs(n_samples=1000, centers=((4, -4), (0, 0)), random_state=
X1 = X1.dot(np.array([[0.374, 0.95], [0.732, 0.598]]))
X2, y2 = make_blobs(n_samples=250, centers=1, random_state=42)
X2 = X2 + [6, -8]
X = np.r_[X1, X2]
y = np.r_[y1, y2]
```

Question 7

(2 pts)

Visualize the dataset - plot the `X` arrays of features. Describe where you would place the cluster center by eye.

In [16]:
```
# enter code here

# Plotting the dataset
plt.figure(figsize=(8, 6))
plt.scatter(X[:, 0], X[:, 1], c='blue', marker='.', alpha=0.5)
plt.title('Visualization of the Dataset')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```



Part 2.2 - Fitting a Gaussian Mixture model

Read the documentation for sci-kit learn's implementation of the Gaussian mixture model:

https://scikit-learn.org/stable/modules/generated/sklearn.mixture.GaussianMixture.html

Question 8

(3 pts)

Use the sci-kit learn Gaussian mixture model to fit the data above.

In [17]:
```
# your code here
from sklearn.mixture import GaussianMixture

# Components by eyeball
```

```
n_components = 3

# Initialize the Gaussian Mixture model
gmm = GaussianMixture(n_components=n_components, random_state=42)

# Fit the model to the data
gmm.fit(X)

# After fitting, you can obtain the cluster labels for each data point
labels = gmm.predict(X)
```

Question 9

(2 pts)

Examine the following attributes of your model and explain what they mean

- `.weights_`
- `.means_`
- `.covariances_`
- `.converged_`
- `.n_iter_`

In [18]:
```
# your answer here

# Examine each of the attributes
print("Weights of each mixture component (.weights_):")
print(gmm.weights_)
print("\nMeans of each mixture component (.means_):")
print(gmm.means_)
print("\nCovariances of each mixture component (.covariances_):")
print(gmm.covariances_)
print("\nConvergence status (.converged_):")
print(gmm.converged_)
print("\nNumber of iterations to converge (.n_iter_):")
print(gmm.n_iter_)
```

```
Weights of each mixture component (.weights_):
[0.39019634 0.2097235  0.40008016]

Means of each mixture component (.means_):
[[ 0.05116693  0.07505701]
 [ 3.39839223  1.05935389]
 [-1.40763382  1.42705028]]

Covariances of each mixture component (.covariances_):
[[[ 0.68778225  0.79591107]
  [ 0.79591107  1.2122181 ]]

 [[ 1.14929846 -0.03260641]
  [-0.03260641  0.95490238]]

 [[ 0.63478593  0.7296996 ]
  [ 0.7296996   1.1610729 ]]]

Convergence status (.converged_):
True

Number of iterations to converge (.n_iter_):
4
```

Meaning of each of the attributes:

`.weights_`

- The weights of each mixture component. In the context of GMMs, a weight represents the proportion of the dataset that is expected to belong to each component. The weights sum up to 1.

`.means_`

- The mean of each mixture component. Since GMMs are probabilistic models that assume each component is Gaussian, each component is defined by a mean and a covariance. The `.means_` attribute gives the centroid (mean vector) of each Gaussian component in the feature space.

`.covariances_`

- The covariance of each mixture component. The covariance matrix of a component describes how its features vary together. In GMMs, this can also indicate the shape and orientation of the component in the feature space.

`.converged_`

- This is a boolean flag that indicates whether or not the algorithm used to fit the GMM converged.

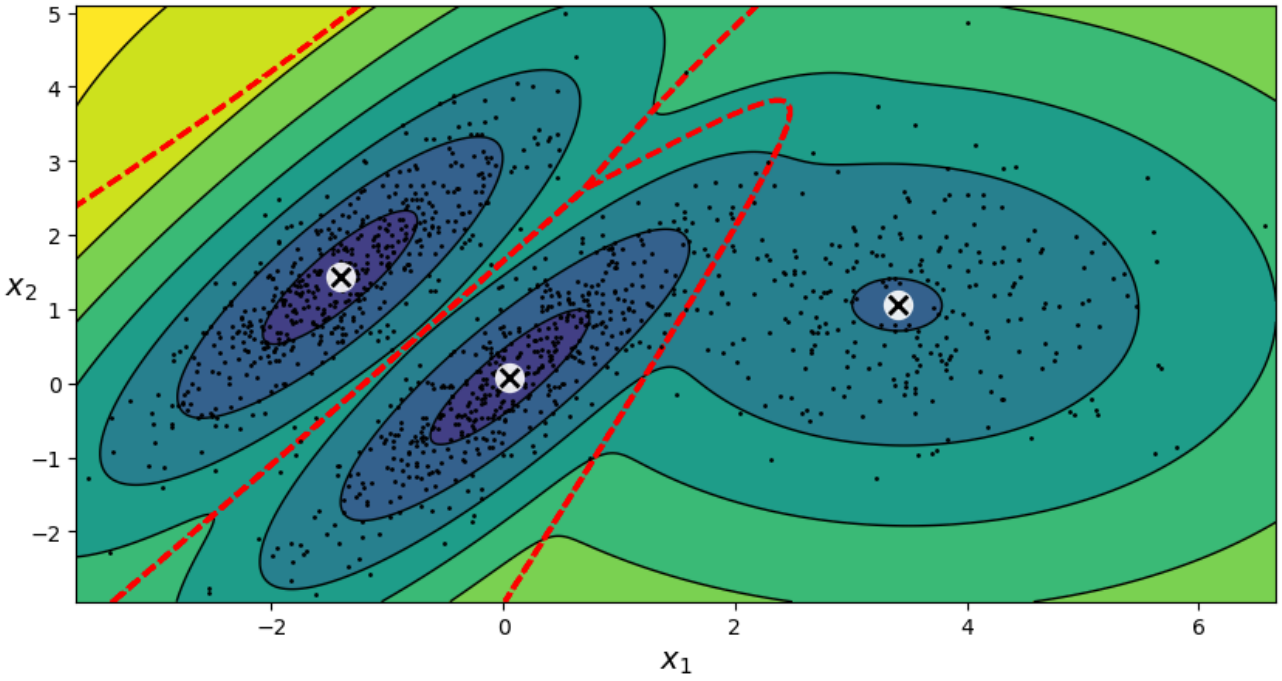
`.n_iter_`

- This attribute shows the number of iterations the EM algorithm ran before termination.

Visualize the model - run the following cells to visualize the predictions from the Gaussian mixture model

In [19]:
```
from matplotlib.colors import LogNorm

def plot_gaussian_mixture(clusterer, X, resolution=1000, show_ylabels=True):
    mins = X.min(axis=0) - 0.1
    maxs = X.max(axis=0) + 0.1
    xx, yy = np.meshgrid(np.linspace(mins[0], maxs[0], resolution),
                         np.linspace(mins[1], maxs[1], resolution))
    Z = -clusterer.score_samples(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.contourf(xx, yy, Z,
                 norm=LogNorm(vmin=1.0, vmax=30.0),
                 levels=np.logspace(0, 2, 12))
    plt.contour(xx, yy, Z,
                norm=LogNorm(vmin=1.0, vmax=30.0),
                levels=np.logspace(0, 2, 12),
                linewidths=1, colors='k')

    Z = clusterer.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contour(xx, yy, Z,
                linewidths=2, colors='r', linestyles='dashed')

    plt.plot(X[:, 0], X[:, 1], 'k.', markersize=2)
    plot_centroids(clusterer.means_, clusterer.weights_)

    plt.xlabel("$x_1$", fontsize=14)
    if show_ylabels:
        plt.ylabel("$x_2$", fontsize=14, rotation=0)
    else:
        plt.tick_params(labelleft=False)
```

In [20]:
```
plt.figure(figsize=(10, 5))

plot_gaussian_mixture(gmm, X)
```



Part 2.3 - Prediction and Generating new instances using the model

Since Gaussian mixture models are probabilistic, we can use it to determine the probability of belonging to a given cluster for any instance of the dataset. We can also generate new instances of data given a trained model.

Question 10

(4 pts)

Predict the class and probability of each instance of the training data. Compare the predictions with the number of instances you expected to be in each cluster

Hint: use the `.predict()` and `.predict_proba()` methods

In [21]:
```
predictClass = gmm.predict(X)
predictProb = gmm.predict_proba(X).tolist()
newPredict = []
for i in range(len(predictProb)):
    prob = max(predictProb[i]) # cluster label of the instance would be the
    newPredict.append(predictProb[i].index(prob))
newPredict = np.array(newPredict)

uniqueClass, countClass = np.unique(predictClass, return_counts = True)
```

```
uniqueProb, countProb = np.unique(newPredict, return_counts = True)

print(F"predicted number of instances in cluster: {dict(zip(uniqueClass, cou
print(F"expected number of instances in cluster: {dict(zip(uniqueProb, count
```

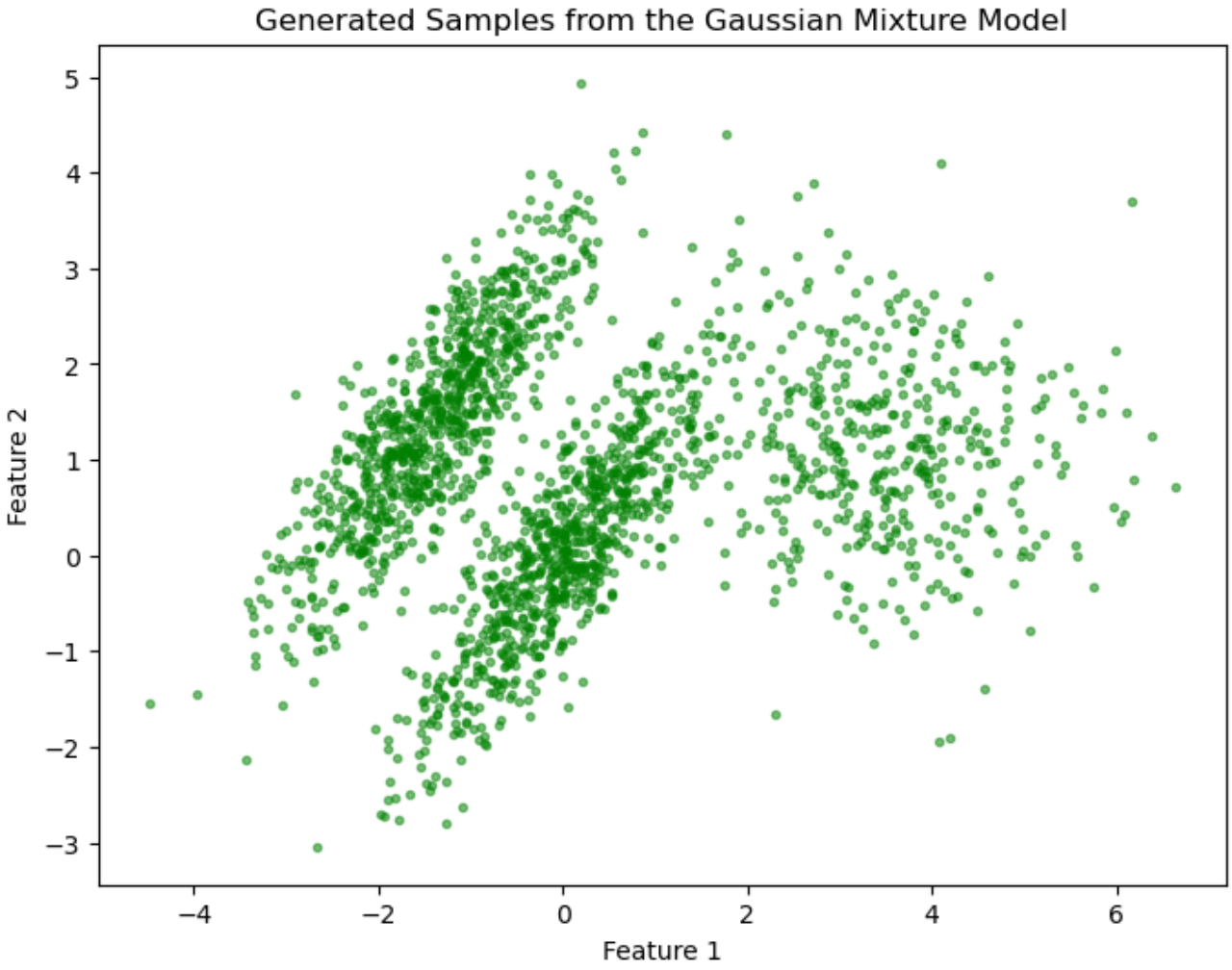
```
predicted number of instances in cluster: {0: 501, 1: 248, 2: 501}
expected number of instances in cluster: {0: 501, 1: 248, 2: 501}
```

Comparison to Expected Number of Instances: We initialized 1000 samples in two clusters and 250 samples in a third cluster. The expected distribution would be two clusters of 500 and one of 250, which is approximately what is found.

Generate 2000 new samples of the dataset and plot them

Hint: use the `.sample` method in the trained model to generate instances.

In [22]:
```
# your code here

# Gen 2000 new samples
generated_data = gmm.sample(n_samples=2000)

# Get samples directly from the returned tuple
new_samples = generated_data[0]

# Plot the generated samples
plt.figure(figsize=(8, 6))
plt.scatter(new_samples[:, 0], new_samples[:, 1], c='green', marker='.', alp
plt.title('Generated Samples from the Gaussian Mixture Model')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```



Part 2.4 Determining the Optimal Number of Clusters

Gaussian mixtures are a probabilistic model, so we can use our Bayesian inference framework for things like model selection. In this case, we want to compare models with different number of clusters and select the one that best explain the data with the fewest parameters.

There are two Bayesian model selection criteria that are built into sci-kit learn:

- Bayesian Information Criterion
- Akaike Information Criterion

Both of these criteria are described in Chapter 9 of Hands-On ML 2nd Edition. For both of these information criteria, you want the model that has the lowest value.

NOTE: you can also use these criteria to select between different kinds of Gaussian models. For example, the sci-kit learn Gaussian mixture model has different choices for

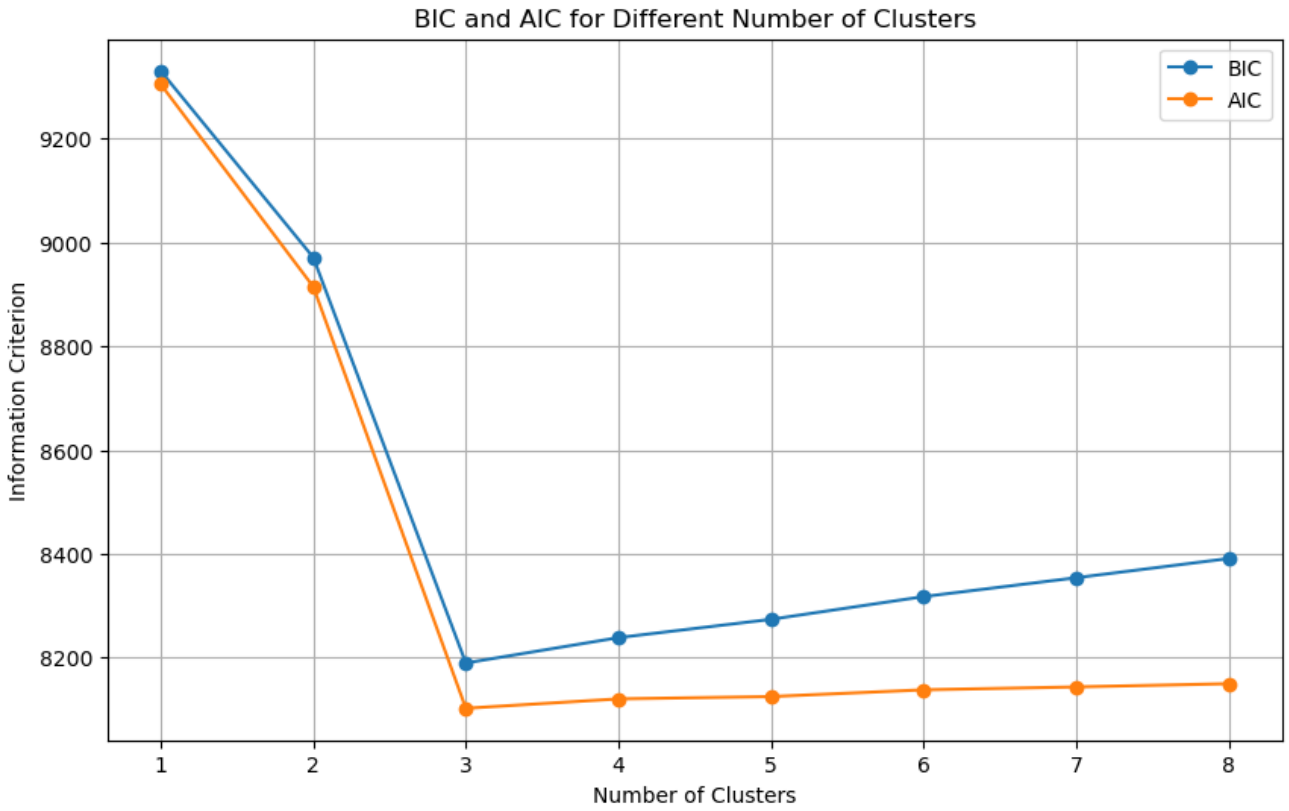
the type of covariance to fit using the `covariance_type` keyword.

Question 11

(4 pts)

Compute the BIC and the AIC for a Gaussian mixture model with between 1 to 8 clusters. What is the optimal number of clusters?

In [23]:
```
# your code here

# Lists to store the BIC and AIC values
bics = []
aics = []

# Range of cluster numbers to evaluate
cluster_range = range(1, 9)

# Compute BIC and AIC for each number of clusters
for n_clusters in cluster_range:
    gmm = GaussianMixture(n_components=n_clusters, random_state=42)
    gmm.fit(X)
    bics.append(gmm.bic(X))
    aics.append(gmm.aic(X))

# Plotting the BIC and AIC values
plt.figure(figsize=(10, 6))
plt.plot(cluster_range, bics, label='BIC', marker='o')
plt.plot(cluster_range, aics, label='AIC', marker='o')
plt.xlabel('Number of Clusters')
plt.ylabel('Information Criterion')
plt.legend()
plt.title('BIC and AIC for Different Number of Clusters')
plt.xticks(cluster_range)
plt.grid(True)
plt.show()

# Finding the optimal number of clusters based on the lowest BIC and AIC val
optimal_clusters_bic = cluster_range[np.argmin(bics)]
optimal_clusters_aic = cluster_range[np.argmin(aics)]
print(f"Optimal number of clusters based on BIC: {optimal_clusters_bic}")
print(f"Optimal number of clusters based on AIC: {optimal_clusters_aic}")
```




```
Optimal number of clusters based on BIC: 3
Optimal number of clusters based on AIC: 3
```

Part 2.5 - Fitting for both the cluster parameters and the number of clusters at the same time

Since we are using a Bayesian framework, we can also fit for the number of clusters directly in our inference for the cluster parameters. In fact, this is generally how you would want to try in real-world data as it tends to be more robust to fit everything together.

Luckily sci-kit learn has this capability for Gaussian mixture modeling through the `BayesianGaussianMixture` model. Read the documentation at: https://scikit-learn.org/stable/modules/generated/sklearn.mixture.BayesianGaussianMixture.html

Question 12

(4 pts)

Fit the dataset above using the `BayesianGaussianMixture` model

Note that to use `BayesianGaussianMixture`, you will need to give it an initial guess

for the number of clusters. Make this initial guess large (like 10).

In [24]:
```
# your code here
from sklearn.mixture import BayesianGaussianMixture

bgmm = BayesianGaussianMixture(n_components=10,
                               covariance_type='full',
                               max_iter=1000, # Increase the number of itera
                               tol=1e-3, # Adjust the convergence threshold
                               random_state=42)
bgmm.fit(X)
```

Out[24]:
```
BayesianGaussianMixture
BayesianGaussianMixture(max_iter=1000, n_components=10, random_stat
e=42)
```

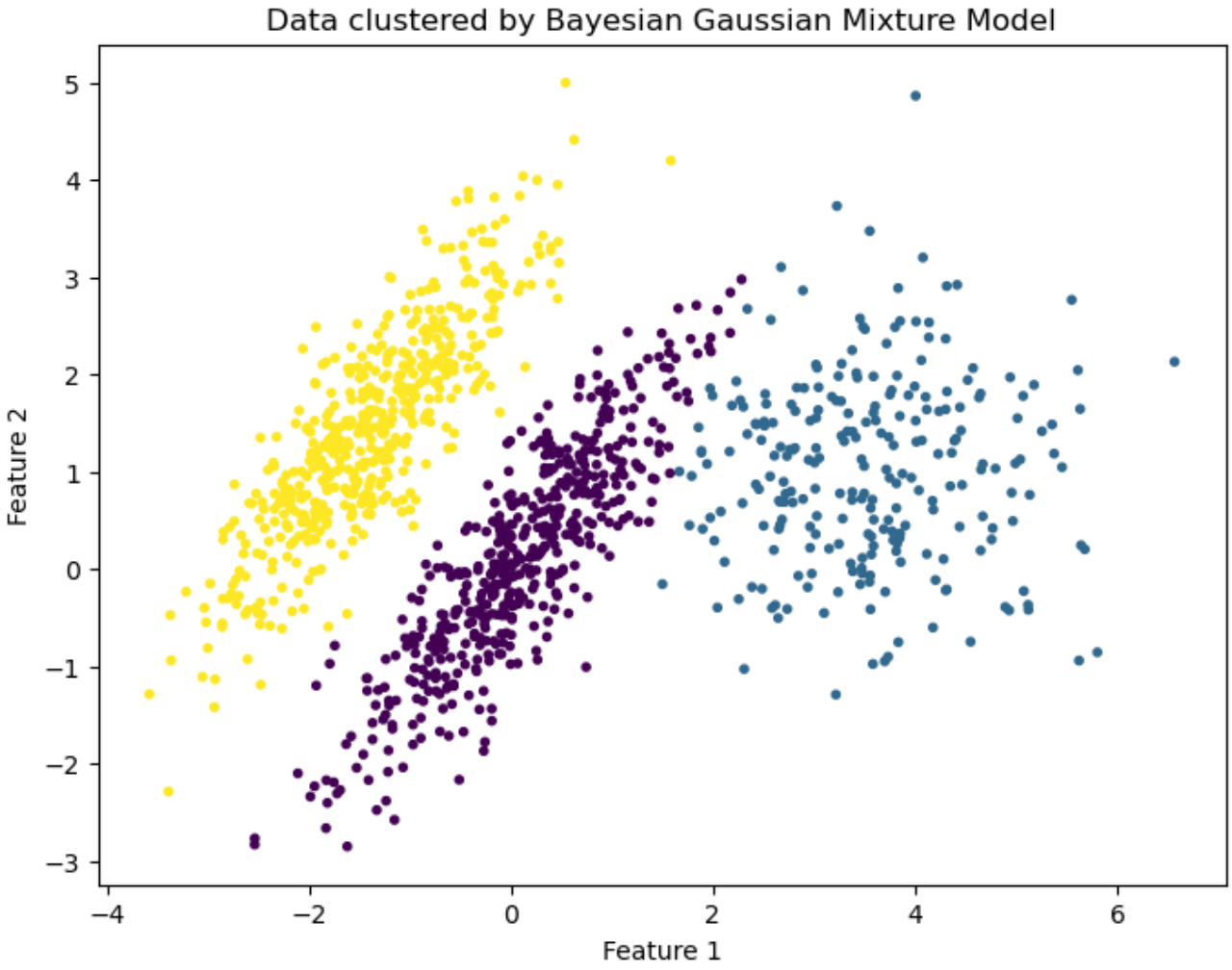
Examine the `.weights_` attribute after fitting. What does it mean?

In [25]:
```
# your code here
print(bgmm.weights_)
```

```
[3.95174673e-01 2.05046196e-01 8.51794745e-04 3.98847828e-01
 7.22802686e-05 6.57091743e-06 5.97356129e-07 5.43051027e-08
 4.93682752e-09 4.48802501e-10]
```

In [26]:
```
# Components with very small weights can be ignored, indicating fewer cluste
effective_clusters = np.sum(bgmm.weights_ > 0.01) # Ignore small weights

print(f"Effective number of clusters: {effective_clusters}")

predicted_labels = bgmm.predict(X)
plt.figure(figsize=(8, 6))
plt.scatter(X[:, 0], X[:, 1], c=predicted_labels, cmap='viridis', marker='.'
plt.title('Data clustered by Bayesian Gaussian Mixture Model')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```

```
Effective number of clusters: 3
```



Part 3 - Discovering a new dwarf galaxy embedded in the Milky Way

The Gaia spacecraft and the all sky survey that it is producing is changing many aspects of astronomy. The Gaia catalog now contains over 2 billion sources with position and velocity measurements.

One of the big discoveries of Gaia is the detection of the remnant of a dwarf galaxy embedded in the Milky Way that was previously hidden from view because the stars are mixed with the Milky Way stars. This dwarf galaxy is called Gaia-Enceladus (also known as the Gaia Sausage). More details can be found in the discovery paper from Helmi et al. (2018): https://arxiv.org/abs/1806.06038

In this part of the lab, you will use a subset of the Gaia dataset to see if you can detect Gaia-Enceladus using unsuperivsed clustering methods. Unlike the other clustering

excercises you did in other parts of the lab, this time, you'll use more than two features.

3.1 Big Picture

The science question we would like to answer is: "How did the components of the Milky Way form?" To answer this question, we can test models of the formation of the Milky Way. One model is that the Milky Way halo is made up of the mergers of smaller galaxies. Evidence of past mergers would support this model.

3.2 Get the data

In this lab's directory, there's a file called `gaia_apogee_match_noflag.csv`. This file contains data from the Gaia survey with the kinematic information on the stars. It contains data from the APOGEE survey, which is a spectroscopic survey from the Sloan Digital Sky Survey. APOGEE provides measurements of the elemental abundances for these stars.

The columns are:

- source_id - ID number of the star in Gaia
- vx - velocity of the star in the direction of the Galactic center
- vy - tangential velocity of the star in the Milky Way disk of the Milky Way
- vz - perpendicular velocity to the Milky Way disk
- En - total energy of the star
- Lz - angular moment of the stars
- apogee - the star's name in the APOGEE survey
- feh - [Fe/H] the logarithm of the iron abundance of the star as a fraction of hydrogen, scaled by the sun. ([Fe/H] = 0 for the Sun)
- alpha - [alpha/Fe] the logarithm of the ratio of alpha-elements to iron in (scaled by the sun), [alpha/Fe]= 0 for the Sun.

Question 13

(2 pts)

Load the data into a pandas data frame and examine it

In [27]:
```
# code here
import pandas as pd
```

```
# Load the data into a pandas DataFrame
df = pd.read_csv('gaia_apogee_match_noflag.csv')
```

3.3 Explore the data

Question 14

(5 pts)

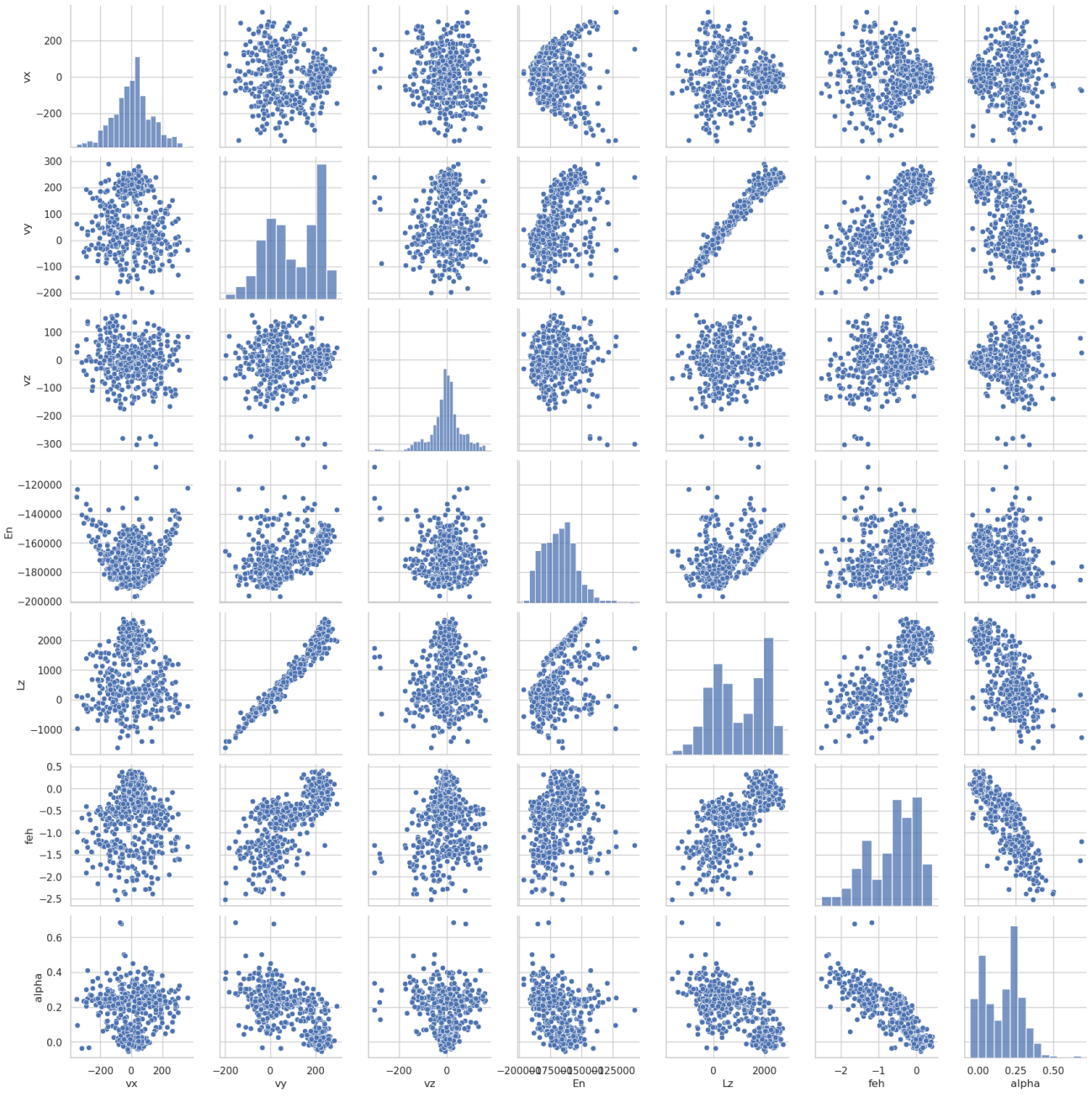
Check the shape, and head, plot, visualize, and have a look at the data with whatever ways you think will help your task. Comment throughout on observations. Are you there potential clustering that you see in some of these features?

In [28]:
```
# code here
print('---Shape---\n', df.shape)
print('---Head---\n', df.head())
print('---Describe---\n', df.describe())
print('---Is Null---\n', df.isnull().sum())
```

```
---Shape---
 (526, 9)
---Head---
          source_id_1          vx          vy         vz             En  \
0  61047157615088256  -25.761714  222.054260  29.777083 -155526.229018
1  65723380567903104   -1.410748  222.424836  27.887632 -152480.050683
2  66570932234426112  -52.639903  213.206085  10.694683 -159300.952613
3  67139104867760384   23.515292  232.149817  34.817371 -150036.436055
4  73621275666889344 -122.512619  -15.886698   7.277072 -176382.521936

            Lz               apogee       feh     alpha
0  2298.725882  2M03253760+2050536 -0.312015  0.081689
1  2477.465061  2M03531303+2352226 -0.276759  0.057058
2  2120.135959  2M03532875+2407032  0.263341  0.026393
3  2578.785212  2M03592264+2531442 -0.250059  0.056364
4  -119.022473  2M02101132+1222103 -0.985187  0.093934
---Describe---
         source_id_1          vx          vy          vz             En  \
count   5.260000e+02  526.000000  526.000000  526.000000     526.000000
mean    2.102764e+18    7.275274   96.949364   -5.325371 -167971.268549
std     1.309843e+18  121.910853  115.644161   64.978286   13133.246109
min     6.104716e+16 -352.357185 -199.859889 -302.312380 -196516.693412
25%     1.269537e+18  -65.534483    2.101660  -30.951782 -178254.104527
50%     1.597441e+18   14.644768   93.587644   -2.765396 -167769.209163
75%     3.373111e+18   73.839982  209.344272   24.721122 -159373.840061
max     6.917218e+18  359.676197  290.882264  159.071067 -107879.448995

                Lz         feh       alpha
count   526.000000  526.000000  526.000000
mean    949.951179   -0.628290    0.162382
std    1010.721486    0.651759    0.126699
min   -1612.701797   -2.521534   -0.053512
25%     101.954732   -1.180192    0.034605
50%     887.899437   -0.481386    0.190760
75%    1938.974793   -0.097929    0.250312
max    2720.889619    0.413277    0.686474
---Is Null---
 source_id_1    0
vx              0
vy              0
vz              0
En              0
Lz              0
apogee          0
feh             0
alpha           0
dtype: int64
```

In [29]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

```
sns.set(style="whitegrid")

# Generate the pair plot for all features
sns.pairplot(df[['vx', 'vy', 'vz', 'En', 'Lz', 'feh', 'alpha']])
plt.show()
```



3.4 Prepare Data

The data is mostly prepared already.

3.5 Select model and train

Question 15

(4 pts)

Choose a clustering algorithm and try to find clusters in the data. You can use any clustering algorithm that you would like. Since this is unsupervised learning, we will not be splitting our data into training and testing.

In [30]:
```
# code here

# Selecting features for clustering
features = df[['vx', 'vy', 'vz', 'En', 'Lz', 'feh', 'alpha']]

# Fit the Gaussian Mixture Model
gmm = GaussianMixture(n_components=3, random_state=1)
gmm.fit(features)

# Predict the clusters
gmm_labels = gmm.predict(features)
```

Question 16

(4 pts)

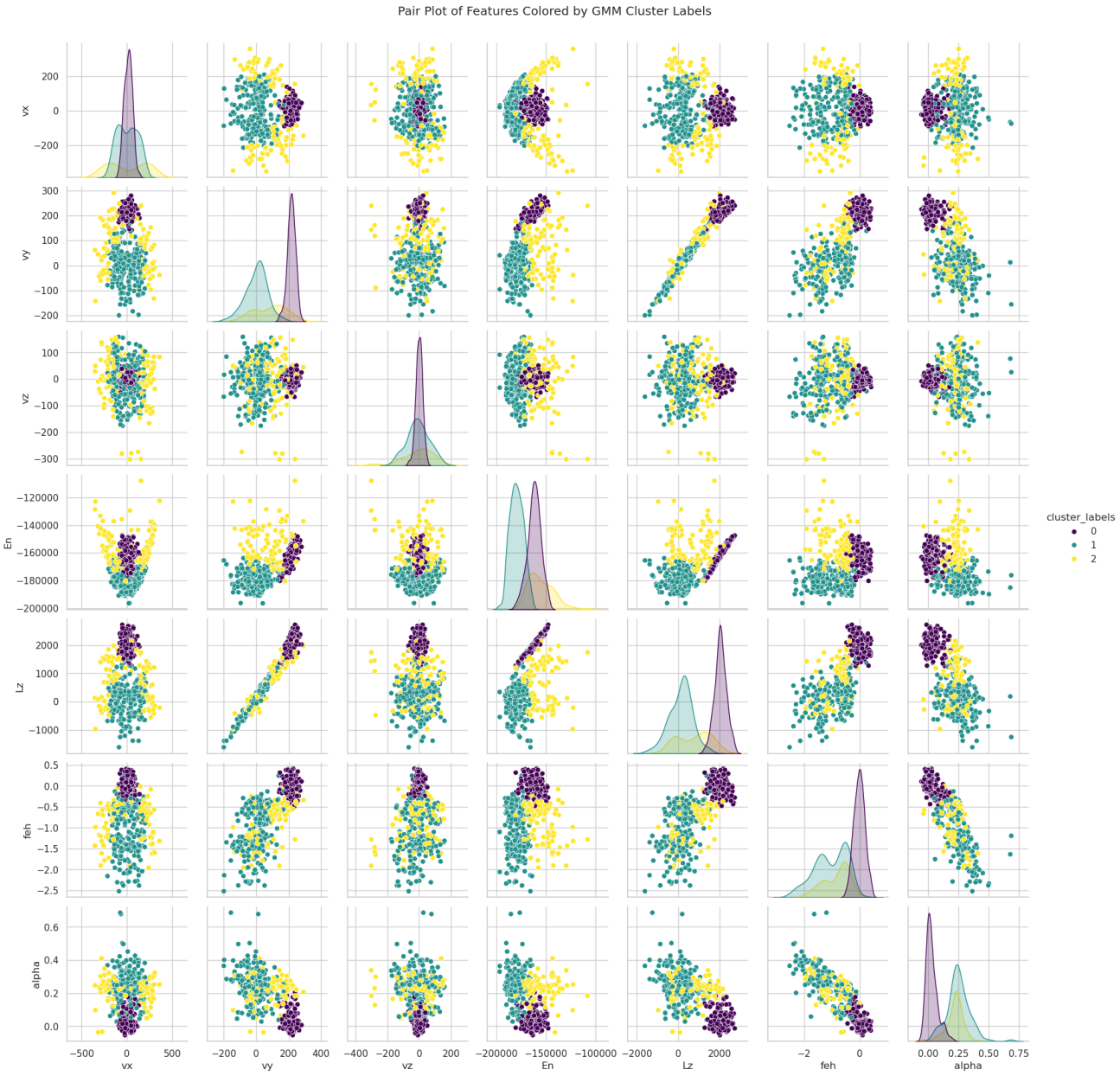
Discuss the properties of the clusters that the algorithm picked out. Visualize the clusters using the different features

I chose to use Gaussian Mixture Modelling after doing some searching online. My impression is that they are often used in astrophysics for modeling celestial objects and that they have flexibility in modeling different covariance structures which makes GMMs useful for capturing complex, overlapping clusters. I considered also, K-Means for its simplicity, and efficiency but feel its use will be limited if clusters are not spherical.

In [31]:
```
# Add labels to df
df['cluster_labels'] = gmm_labels
print(df.head())
```

```
         source_id_1          vx          vy         vz             En  \
0  61047157615088256  -25.761714  222.054260  29.777083 -155526.229018
1  65723380567903104   -1.410748  222.424836  27.887632 -152480.050683
2  66570932234426112  -52.639903  213.206085  10.694683 -159300.952613
3  67139104867760384   23.515292  232.149817  34.817371 -150036.436055
4  73621275666889344 -122.512619  -15.886698   7.277072 -176382.521936

            Lz              apogee       feh     alpha  cluster_labels
0  2298.725882  2M03253760+2050536 -0.312015  0.081689               0
1  2477.465061  2M03531303+2352226 -0.276759  0.057058               0
2  2120.135959  2M03532875+2407032  0.263341  0.026393               0
3  2578.785212  2M03592264+2531442 -0.250059  0.056364               0
4  -119.022473  2M02101132+1222103 -0.985187  0.093934               1
```

In [32]:
```
# Generate a pair plot for all features, colored by cluster labels
sns.pairplot(df, vars=['vx', 'vy', 'vz', 'En', 'Lz', 'feh', 'alpha'], hue='c
plt.suptitle('Pair Plot of Features Colored by GMM Cluster Labels', y=1.02)
plt.show()
```



Properties of Clusters found by Model: The model has seperated the stars into three clusters. Based on the pairplot above, vy, En, Lz, and feh appear to be reasonably good seperators with most scatter plots involving these features forming relatively distinct clusters. Of the classes found in this dataset, a minority of stars are of class 1. The distributions of vy and Lz are most distinct with each class having seperate peaks and less overlap than other features.

Question 17

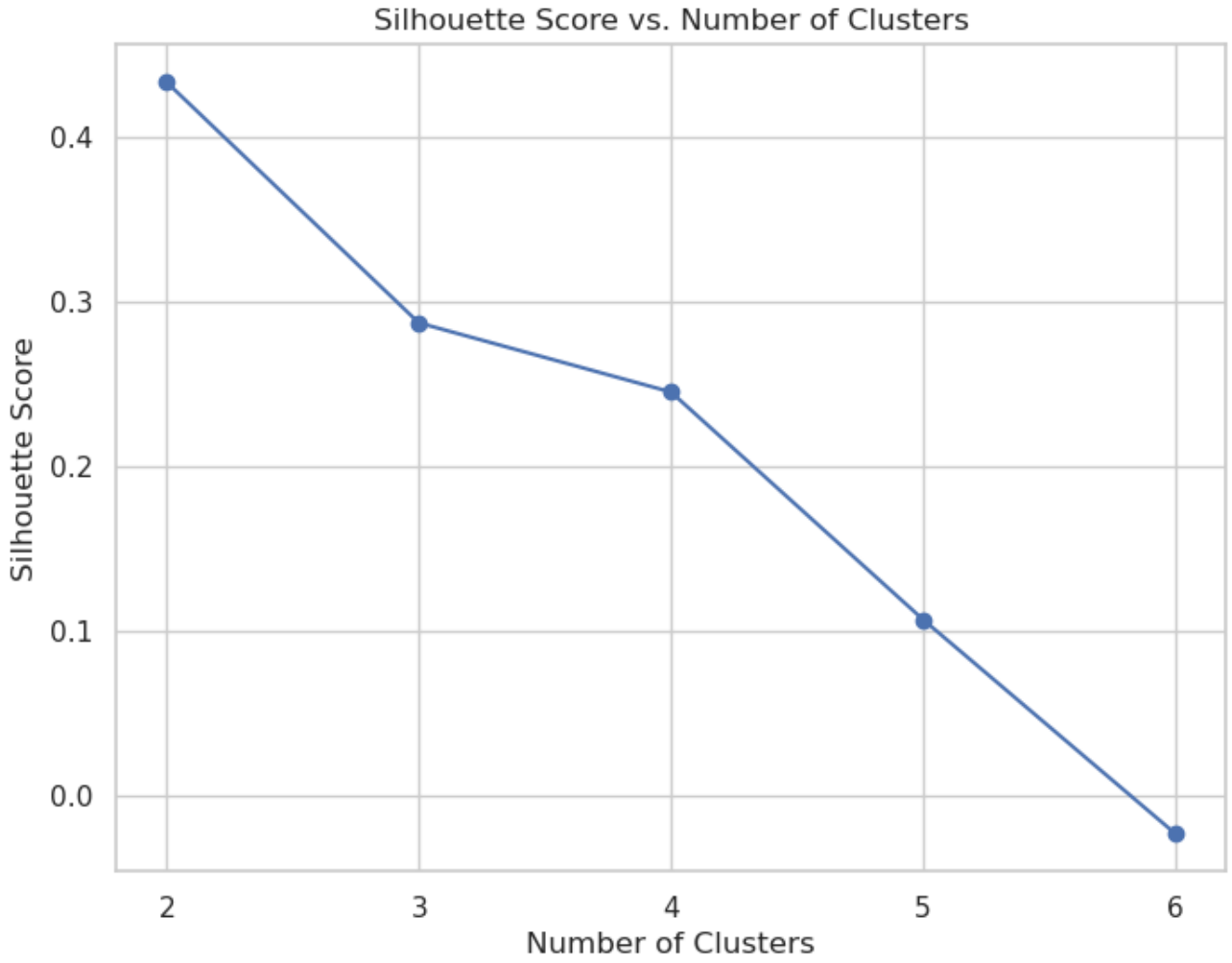
(4 pts)

Test the clustering performance. Discuss what how you determined the optimal number of clusters

Hint: the Milky Way should have two components: stars in the disk of the Milky Way and stars in the halo. Stars in the Milky Way disk have iron abundance and alpha-elemental abundance close to that of the Sun. Stars in the Milky Way Halo tend to have low iron abundance ([Fe/H] < 0). The velocities of the disk and halo are also different. The image below shows some components of the Milky Way. The central region is called the bulge (there are no stars in this sample from that component).

disk and halo

In [33]:
```
silhouette_scores = []
n_clusters_list = range(2, 7)

for n_clusters in n_clusters_list:
    gmm = GaussianMixture(n_components=n_clusters, random_state=42)
    labels = gmm.fit_predict(features)
    silhouette_scores.append(silhouette_score(features, labels))

# Plotting the silhouette scores
plt.figure(figsize=(8, 6))
plt.plot(n_clusters_list, silhouette_scores, marker='o')
plt.title('Silhouette Score vs. Number of Clusters')
plt.xlabel('Number of Clusters')
plt.ylabel('Silhouette Score')
plt.xticks(n_clusters_list)
plt.grid(True)
plt.show()
```



The Silhouette Score was used to determine the optimal number of clusters. Intuitively from the plots and from the description of the problem it makes sense to expect two clusters. The plot above shows loss in performance as cluster count further increases. Despite the Silhouette score indicating two clusters are prefered and the knowledge that stars and the galactic disk and stars in the galactic halo should naturally form two distinct clusters we look for 3 clusters in an effort to identify stars from a dwarf galaxy.

3.6 Fine tune model

Question 18

(5 pts)

Try another clustering algorithm and compare its performance with the previous method. Are there differences in how the clustering works with different number of features? What features are most helpful to distinguish clusters?

Hint: you can use the original paper Helmi et al. (2018): https://arxiv.org/abs/1806.06038 as a guide for how to think about the features. The original paper did not use clustering algorithms, but rather knowledge about the Milky Way's properties to find an extra component that did not belong.

In [34]:
```
# code here

# Let's do K-Means
# Starting with the same number of clusters assumed for GMM
kmeans = KMeans(n_clusters=3, random_state=42, n_init='auto')
kmeans_labels = kmeans.fit_predict(features)

# Calculate silhouette score for K-Means
kmeans_silhouette = silhouette_score(features, kmeans_labels)
print(f"K-Means Silhouette Score: {kmeans_silhouette}")
```

```
K-Means Silhouette Score: 0.5491166963048498
```

Differences in Clustering In comparing the performance of K-Means and the Gaussian Mixture Model (GMM) on a given dataset, it seems that K-Means achieved a higher silhouette score, suggesting more defined clusters. There are inherent differences between the two algorithms. K-Means, with its simplicity and assumption of spherical clusters of similar size, excels when the data naturally forms distinct, non-overlapping groups. Its objective to minimize within-cluster variance directly aligns with the silhouette score's emphasis on tight, well-separated clusters. On the other hand, GMM offers more flexibility by accommodating clusters of various shapes and densities, thanks to its probabilistic approach and allowance for elliptical cluster shapes. However, this flexibility may not always translate to higher silhouette scores.

What Features are Most Important As described above, The model has seperated the stars into clusters. Based on the pairplot above, vy, En, Lz, and feh appear to be reasonably good seperators with most scatter plots involving these features forming relatively distinct clusters. The distributions of vy and Lz are most distinct with each class having peaks with minimal overlap.

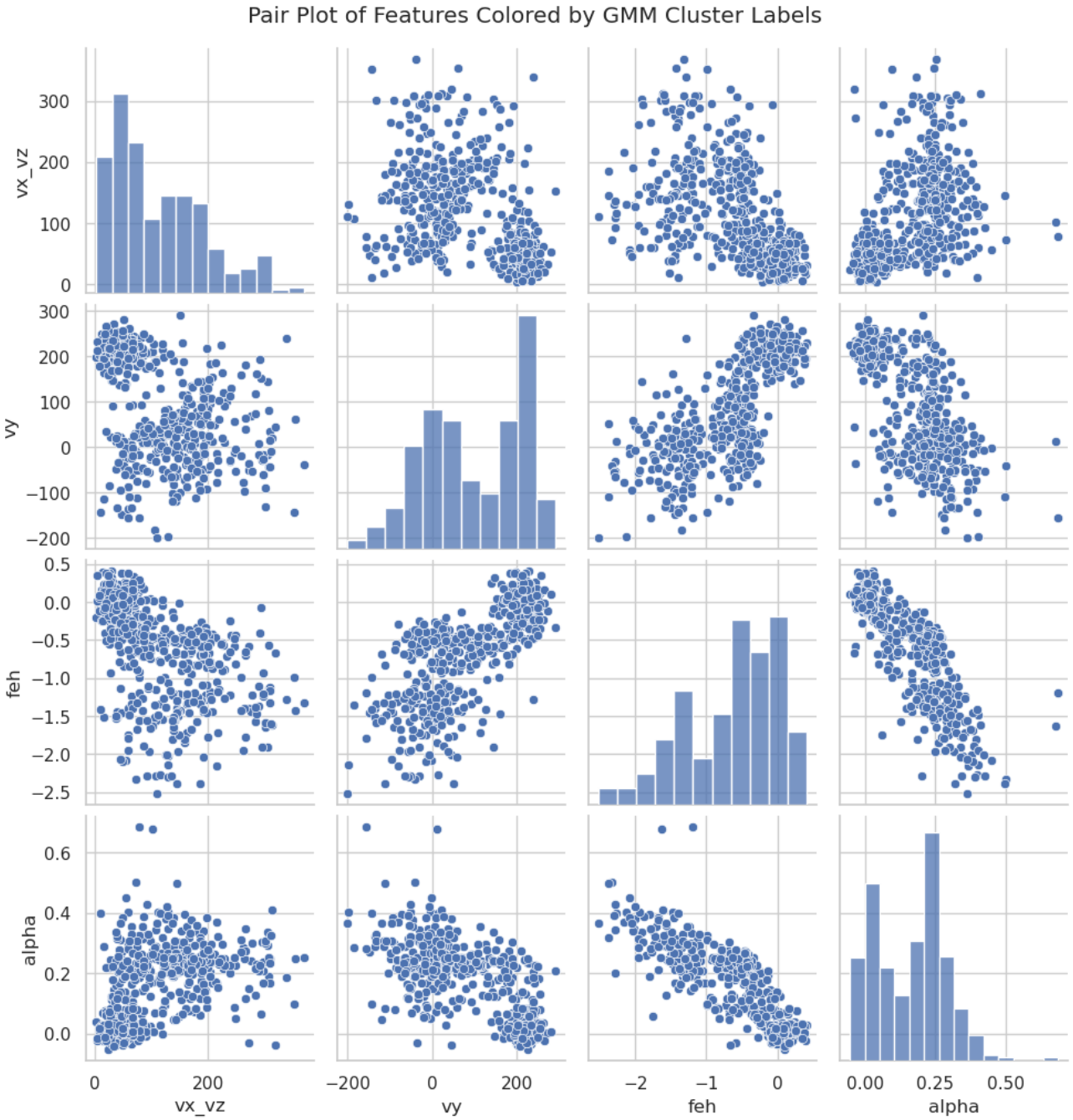
In [35]:
```
df['vx_vz'] = np.sqrt(df['vx']**2 + df['vz']**2)
df.head()
```

Out[35]:

	source_id_1	vx	vy	vz	En	
0	61047157615088256	-25.761714	222.054260	29.777083	-155526.229018	2298.72
1	65723380567903104	-1.410748	222.424836	27.887632	-152480.050683	2477.46
2	66570932234426112	-52.639903	213.206085	10.694683	-159300.952613	2120.13
3	67139104867760384	23.515292	232.149817	34.817371	-150036.436055	2578.78
4	73621275666889344	-122.512619	-15.886698	7.277072	-176382.521936	-119.02

In [36]:

```
# Generate pair plots for the combos of features shown in the paper
df['vx_vz'] = np.sqrt(df['vx']**2 + df['vz']**2)
tuned_features = ['vx_vz', 'vy', 'feh', 'alpha']

sns.pairplot(df, vars=tuned_features)
plt.suptitle('Pair Plot of Features Colored by GMM Cluster Labels', y=1.02)
plt.show()
```



In [37]:
```
#Train model based on a subset of features Identified in paper

# Add the new feature 'vx_vz' The non y velocity
df['vx_vz'] = np.sqrt(df['vx']**2 + df['vz']**2)

# Specify and extract the features to be used for clustering
features_for_clustering = ['vx_vz', 'vy', 'feh', 'alpha']
tuned_features = df[features_for_clustering]

# Fit and predict using Gaussian Mixture Model
gmm = GaussianMixture(n_components=3, random_state=42)
gmm_labels = gmm.fit_predict(tuned_features)
```

```
# Fit and predict using K-Means
kmeans = KMeans(n_clusters=3, random_state=42, n_init='auto')
kmeans_labels = kmeans.fit_predict(tuned_features)
```

In [38]:
```
# Generate a pair plot for all features, colored by cluster labels
df['gmm_labels'] = gmm_labels
df['kmeans_labels'] = kmeans_labels

sns.pairplot(df, vars=tuned_features, hue='gmm_labels', palette='viridis')
plt.suptitle('Pair Plot of Features Colored by GMM Cluster Labels', y=1.02)
plt.show()
```

3.7 Present solution

Question 19

(2 pts)

Discuss whether you can detect the presense of a dwarf galaxy in the data?

Identifying a Dwarf Galaxy: If clustering analysis reveals a group of stars with both distinct kinematic properties and chemical compositions that differ from the bulk of the Milky Way's stars, this could indicate the presence of a dwarf galaxy.

Based on clustering results and analysis of kinematic and chemical properties, one can assess the likelihood of a dwarf galaxy's presence within the dataset. The distinct motion and chemical composition of stars from such a galaxy manifest as clusters that are separable from those of the Milky Way's own stars.

Question 20

(2 pts)

How does your clustering solution compared with the original paper?

In the lab directory there's a file called `gaia_apogee_match_flag.csv` that is the same as the file you loaded, except there is a new column called `flag`. This flag is based on the identification of the stars from the Helmi et al. (2018) paper.

The flag values are: 0 = Milky Way disk stars 1 = Milky Way halo stars 2 = stars from the dwarf galaxy

Note that the identification from the Helmi et al. (2018) paper is based on some specific thresholds in energy and angular momentum, so a clustering algorithm may give more nuanced assignments to the component that each star belongs to.

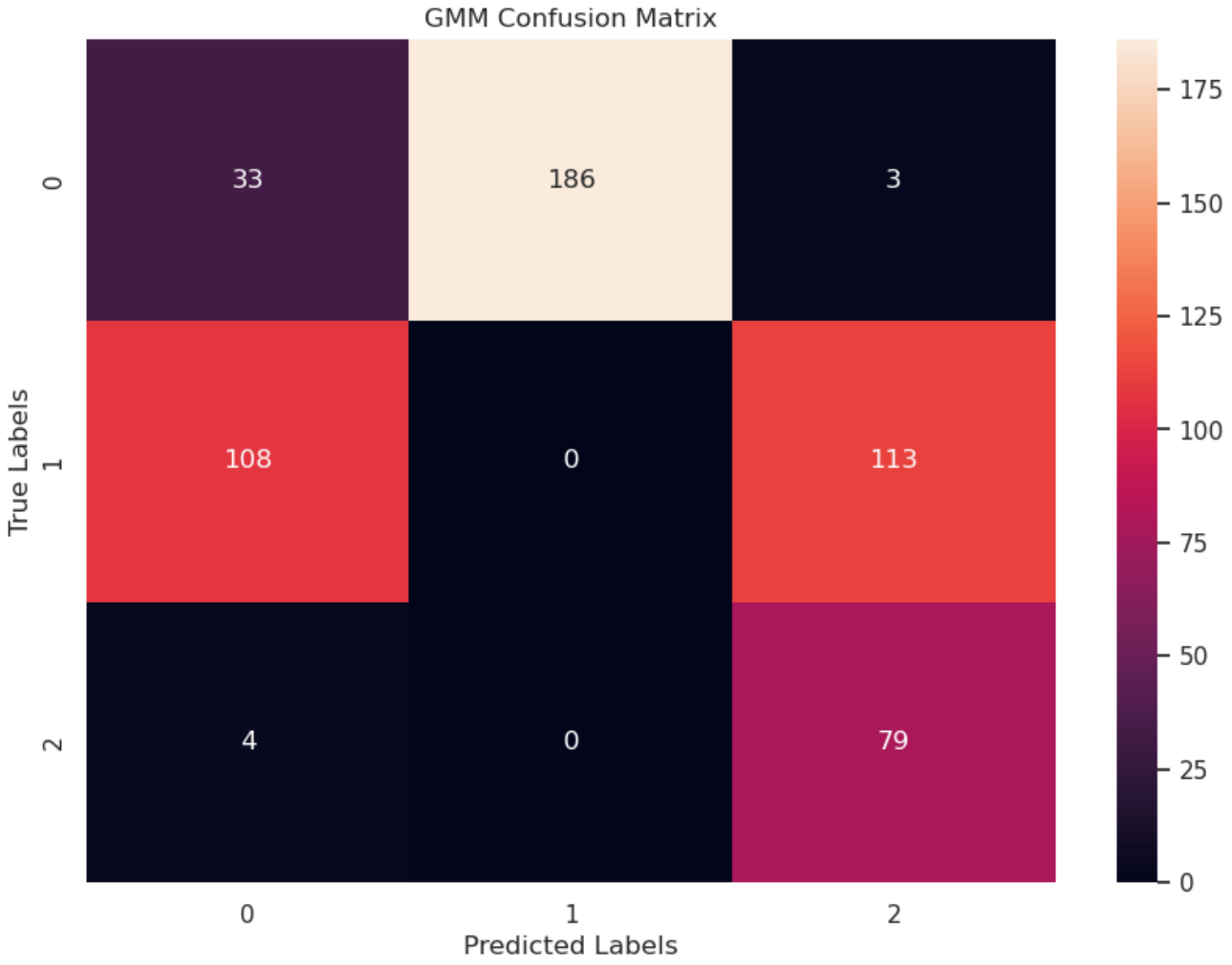
In [39]:
```
df = pd.read_csv('gaia_apogee_match_flag.csv')
df['gmm_labels'] = gmm_labels
df['kmeans_labels'] = kmeans_labels
df.head()
```

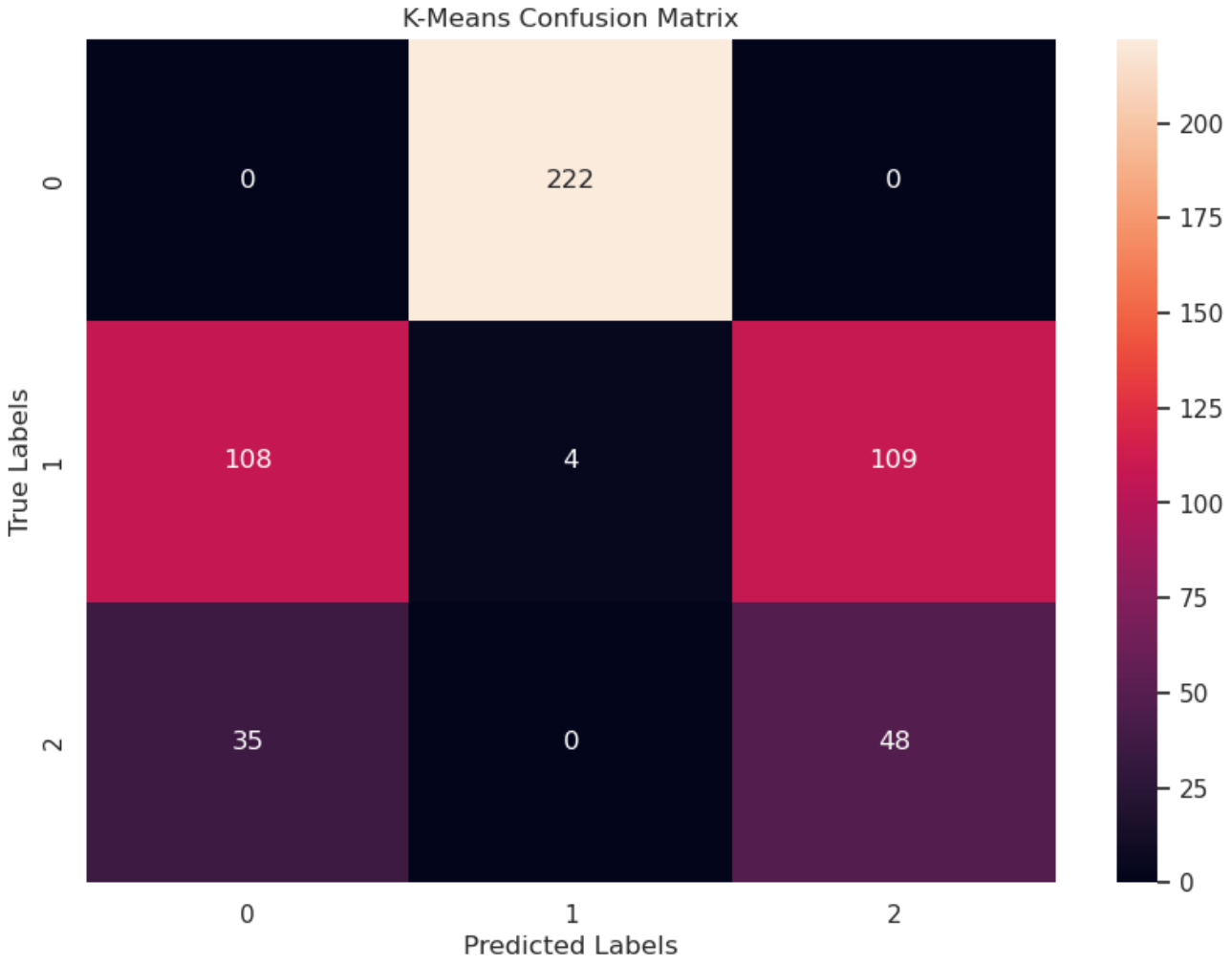
Out[39]:

	source_id_1	vx	vy	vz	En	
0	61047157615088256	-25.761714	222.054260	29.777083	-155526.229018	2298.72
1	65723380567903104	-1.410748	222.424836	27.887632	-152480.050683	2477.46
2	66570932234426112	-52.639903	213.206085	10.694683	-159300.952613	2120.13
3	67139104867760384	23.515292	232.149817	34.817371	-150036.436055	2578.78
4	73621275666889344	-122.512619	-15.886698	7.277072	-176382.521936	-119.02

In [40]:

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Plotting confusion matrix for GMM labels
plt.figure(figsize=(10, 7))
sns.heatmap(confusion_matrix(df['flag'], df['gmm_labels']), annot=True, fmt=
plt.title('GMM Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()

# Plotting confusion matrix for k-means labels
plt.figure(figsize=(10, 7))
sns.heatmap(confusion_matrix(df['flag'], df['kmeans_labels']), annot=True, f
plt.title('K-Means Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()

plt.show()
```

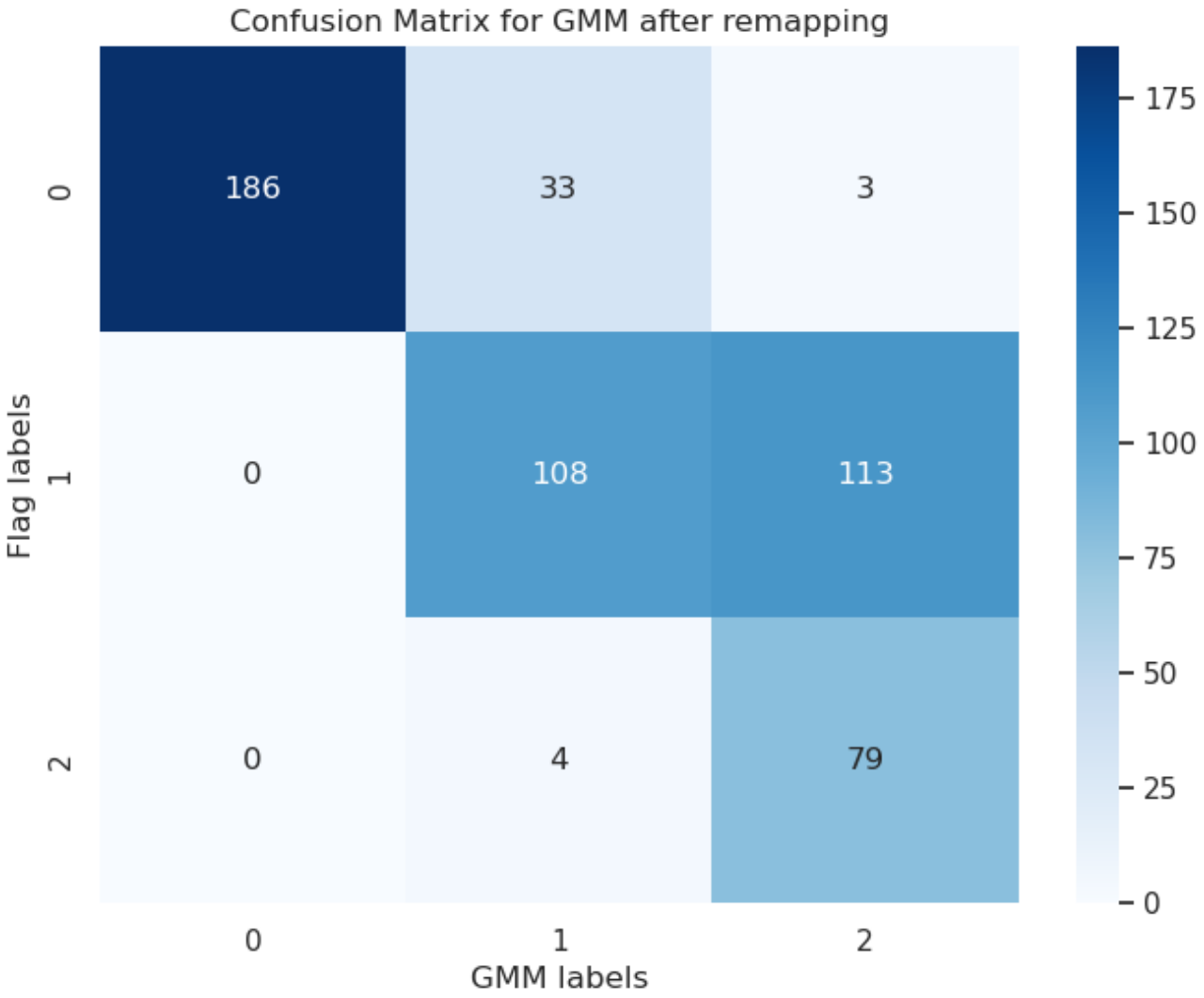

GMM Confusion Matrix
True Labels
Predicted Labels
0
1
2
33
186
3
108
0
113
4
0
79
175
150
125
100
75
50
25
0



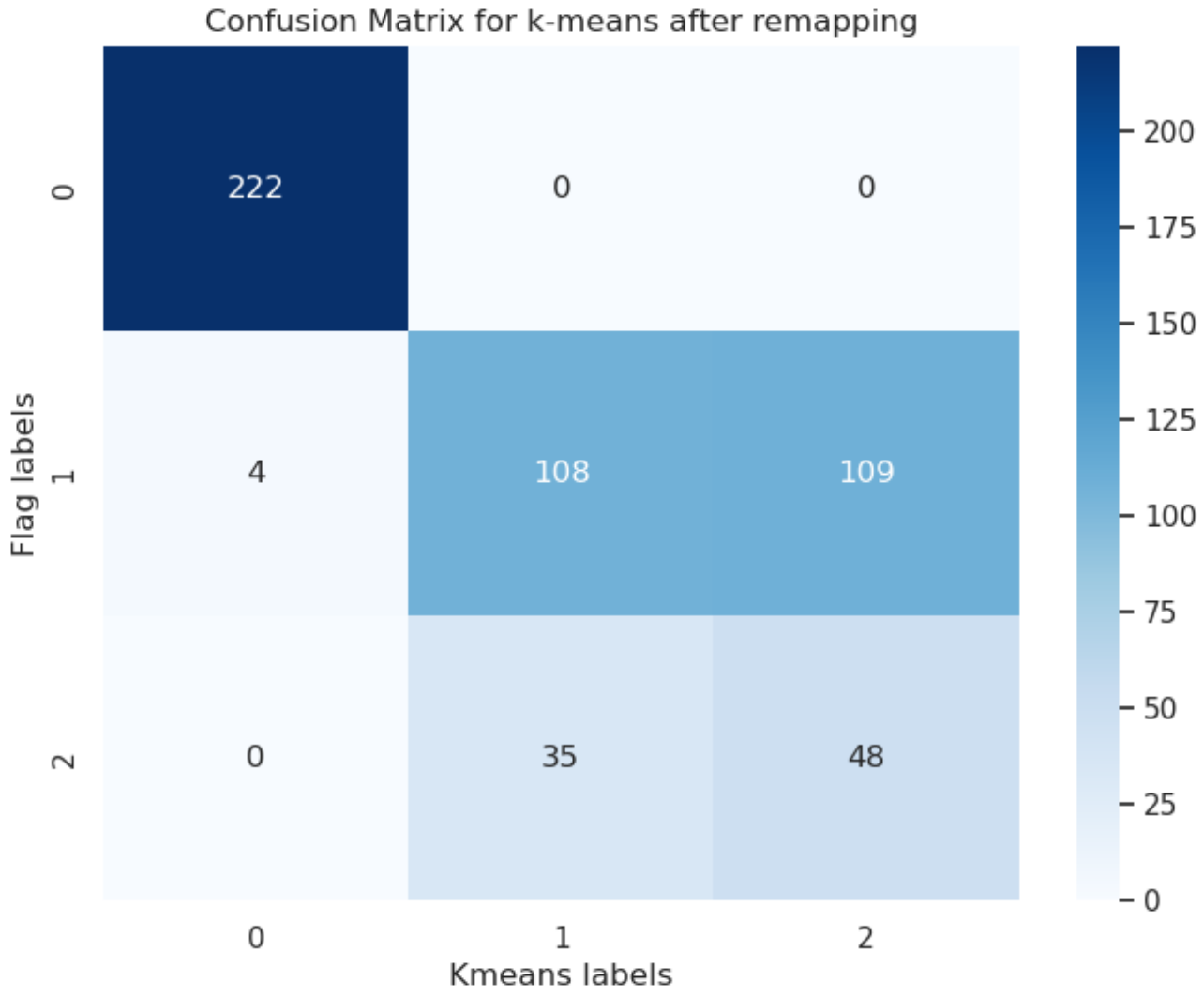
In [41]:
```
from sklearn.metrics import classification_report

# Assuming df is your DataFrame and the remapping rules you provided:
gmm_remap = {0: 1, 1: 0, 2: 2}
kmeans_remap = {0: 1, 1: 0, 2: 2}

# Remapping labels
df['gmm_labels_remapped'] = df['gmm_labels'].map(gmm_remap)
df['kmeans_labels_remapped'] = df['kmeans_labels'].map(kmeans_remap)

# Evaluation for GMM after remapping
print("Classification Report for GMM after remapping:")
print(classification_report(df['flag'], df['gmm_labels_remapped']))

print("Confusion Matrix for GMM after remapping:")
gmm_confusion_matrix = confusion_matrix(df['flag'], df['gmm_labels_remapped'
fig, ax = plt.subplots(figsize=(8, 6))
sns.heatmap(gmm_confusion_matrix, annot=True, fmt='d', cmap='Blues', ax=ax)
ax.set_xlabel('GMM labels')
ax.set_ylabel('Flag labels')
ax.set_title('Confusion Matrix for GMM after remapping')
```

```
plt.show()

# Evaluation for k-means after remapping
print("Classification Report for k-means after remapping:")
print(classification_report(df['flag'], df['kmeans_labels_remapped']))

print("Confusion Matrix for k-means after remapping:")
kmeans_confusion_matrix = confusion_matrix(df['flag'], df['kmeans_labels_rem
fig, ax = plt.subplots(figsize=(8, 6))
sns.heatmap(kmeans_confusion_matrix, annot=True, fmt='d', cmap='Blues', ax=a
ax.set_xlabel('Kmeans labels')
ax.set_ylabel('Flag labels')
ax.set_title('Confusion Matrix for k-means after remapping')
plt.show()
```

```
Classification Report for GMM after remapping:
              precision    recall  f1-score   support

           0       1.00      0.84      0.91       222
           1       0.74      0.49      0.59       221
           2       0.41      0.95      0.57        83

    accuracy                           0.71       526
   macro avg       0.72      0.76      0.69       526
weighted avg       0.80      0.71      0.72       526

Confusion Matrix for GMM after remapping:
```




```
Classification Report for k-means after remapping:
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       222
           1       0.76      0.49      0.59       221
           2       0.31      0.58      0.40        83

    accuracy                           0.72       526
   macro avg       0.68      0.69      0.66       526
weighted avg       0.78      0.72      0.73       526

Confusion Matrix for k-means after remapping:
```



From the confusion matrices and reports above we see some evidence of a distinct third cluster could represent a dwarf galaxy. Agreement with the paper is not complete however with the both GMM and Kmeans drawing a fuzzier boundary between flag labels 1 and 2 (Milky way halo stars and stars from a dwarf galaxy, respectively).

More experimentation with fine tuning is possible. But similarities between stars in the galactic halo and stars from a dwarf galaxy seem reasonable features similar to those utilized in the paper were employed.

Lab Survey and Submit

Please take this lab survey to give us feedback to improve the lab! Survey Link: https://forms.gle/28yywKq2tYV15YXE9

Remeber to restart your kernel and run all cells before submitting!