Alex Johnson | Assignment 7--IMT 574 | 02.19.23

Objectives:

PART 1 [2 POINTS]:

- EM falls under unsupervised learning. But what specific kind of unsupervised learning setting would you use it for?
  - Put differently, you could use both k-means and EM for clustering, but when will you pick EM over k-means?

PART 2 [8 POINTS]

Under the life-cycle savings hypothesis, as developed by Franco Modigliani, the savings ratio (aggregate personal savings divided by disposable income) is explained by per-capita disposable income, the percentage rate of change in per-capita disposable income, and two demographic variables: The percentage of population less than 15 years old and the percentage of the population over 75 years old. The data are averaged over the decade 1960–1970 to remove the business cycle or other short-term fluctuations.

The dataset contains 50 observations with five variables.

Number	Variable
I	Sr: numeric, aggregate personal savings
II	pop15: numeric, % of the population under 15
III	pop75: numeric, % of the population over 75
IV	dpi: numeric, real per-capita disposable income
V	ddpi: numeric, % growth rate of dpi

INSTRUCTIONS

1. Use EM for clustering “similar” countries.
2. Report how many groups you got and why you chose that number with the help of AIC and BIC.

PART 1:

Based on this week's readings, I see that expectation maximization is great for applying in clustering applications for unsupervised learning tasks. Specifically, the Gaussian Mixture Model is mentioned as a top clustering implemetation with EM methodology in mind.

These general observations of expectation maximization use cases from this week's readings match what ChatGPT says are major advantages and disadvantages of K-Means and Expectation Maximization insofar as their use cases.

Specifically, OpenAI reports:

- EM is a more complex algorithm that can handle a wider range of clustering tasks.
    - EM assumes that the data is generated from a mixture of Gaussian distributions and estimates the parameters of the distributions to cluster the data.
- EM can handle different shapes and sizes of clusters and can also estimate the probability of each data point belonging to a cluster.
    - However, EM is more computationally expensive than K-means and may not scale well to large datasets.
- EM also requires the number of clusters to be specified in advance, although there are extensions that can estimate the number of clusters.
- K-means is a simple and computationally efficient algorithm that works well when the number of clusters is known in advance.
    - K-means assigns each data point to its nearest centroid and iteratively updates the centroids to minimize the sum of squared distances between each data point and its assigned centroid.
- K-means is easy to implement, scales well to large datasets, and can handle high-dimensional data.
    - However, K-means assumes that the clusters are spherical and equally sized, and it may not work well with data that has non-convex shapes.

Overall:

The important thing when considering appropriate clustering techniques like K-Means and Expectation Maximization is exploring our particular dataset and its statistical features, distributions, and the like. In cases where data is flexible and spherical in distribution, K-Means is less computationally expensive than expectation maximization. Where data is nuanced in its distribution, considering expectation maximization is great given potential for point-estimate capabilities. Knowing the cluster size in advance is difficult with certain data thus K-Means is insufficient, thus EM can be a great alternative for spot-on clustering insight. Let it be noted, however, according to ChatGPT, while EM can estimate cluster size for you, it is not an inherent feature of an EM algorithm, so you need to take extra care to identify extensions with capability to estimate cluster size with your data.

PART 2:

Method overview:

Given the literature covered on Expectation Maximization modeling recommends Gaussian Mixture as a base modality of implementation, I use the GMM function for clustering models in rounds 1-3 of the experiment. In round 1, I use K-Means within the PCA to estimate optimal cluster size from the principal components derived from the initial pca gaussian model.

For this investigation, optimal cluster size will thus be determined with accuracy metrics in terms of AIC and BIC for each respective cluster model across all rounds.

- For round 1, I get a sense of the optimal cluster size with a PCA. Also, by exploring the PCA components specifically in round 1, I identify important characteristics of each respective cluster for the model going forward. This gives a good idea of which countries belong to each cluster for sequential rounds.
- For round 2 and 3, I implement an iterative loop of GMM clusterings as demonstrated in Shah with a method that uses K-Means (2020, p. 152).

Data preprocesing

In [1]:

```
# load libraries
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
import pandas as pd
from pandas import read_csv, set_option
from pandas.plotting import scatter_matrix
import seaborn as sns


from sklearn import preprocessing, metrics
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.preprocessing import StandardScaler, MinMaxScaler, Normalizer
from sklearn.metrics import confusion_matrix, accuracy_score, classification_re
from sklearn.datasets import make_blobs
from sklearn.manifold import TSNE
import scipy.cluster.hierarchy as sch
from sklearn.decomposition import PCA
import sys
from sklearn.mixture import GaussianMixture
from scipy.stats import mode
from sklearn.metrics import adjusted_rand_score, silhouette_score
from matplotlib.patches import Ellipse


import csv
import sys
if not sys.warnoptions:
    import os, warnings
    warnings.simplefilter("ignore") # Change the filter in this process
    os.environ["PYTHONWARNINGS"] = "ignore" # Also affect subprocesses
```

In [2]:

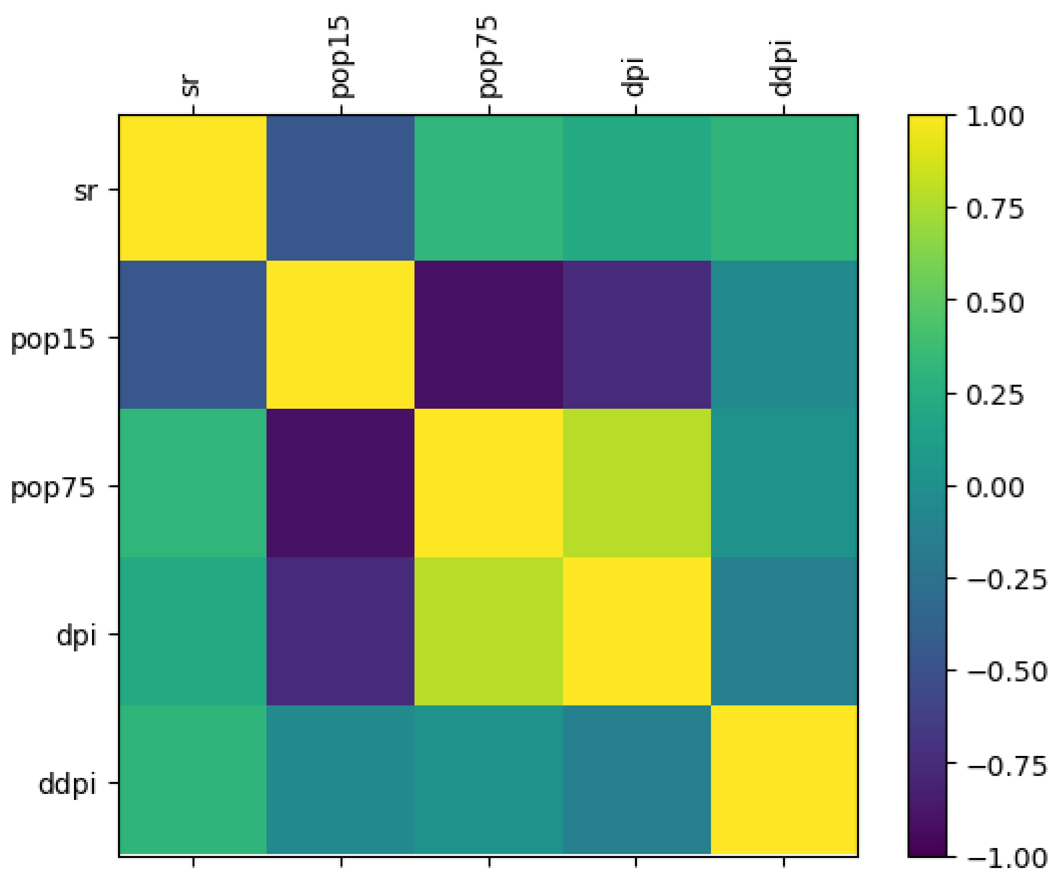
```
# load data
Path = ''

df1 = pd.read_csv(Path)

df = pd.DataFrame(df1)
```

In [3]:
```
# All variables for clustering
X = df[['sr', 'pop15', 'pop75', 'dpi', 'ddpi']]
```

In [4]:
```
scaler = StandardScaler()
XT = scaler.fit_transform(X)
XT = pd.DataFrame(XT)
XT.columns = ['sr', 'pop15', 'pop75', 'dpi', 'ddpi']
```

In [5]:
```
XT2=XT
XT3=XT
```

In [6]:
```
headernames = ['sr', 'pop15', 'pop75', 'dpi', 'ddpi']
correlations = X.corr(method='pearson')

fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(correlations, vmin=-1, vmax=1)
fig.colorbar(cax)
ticks = np.arange(0,5,1)
ax.set_xticks(ticks)
ax.set_yticks(ticks)
ax.set_xticklabels(headernames, rotation=90)
ax.set_yticklabels(headernames)
plt.show()
```



Round 1: PCA Clustering

For round 1 of approaching expectation maximization, I utilize code from:

Biswas, A., (2022, August 12th). Customer segmentation with python (implementing STP framework - part 3/5) [Blog post and code notebook]. Medium.

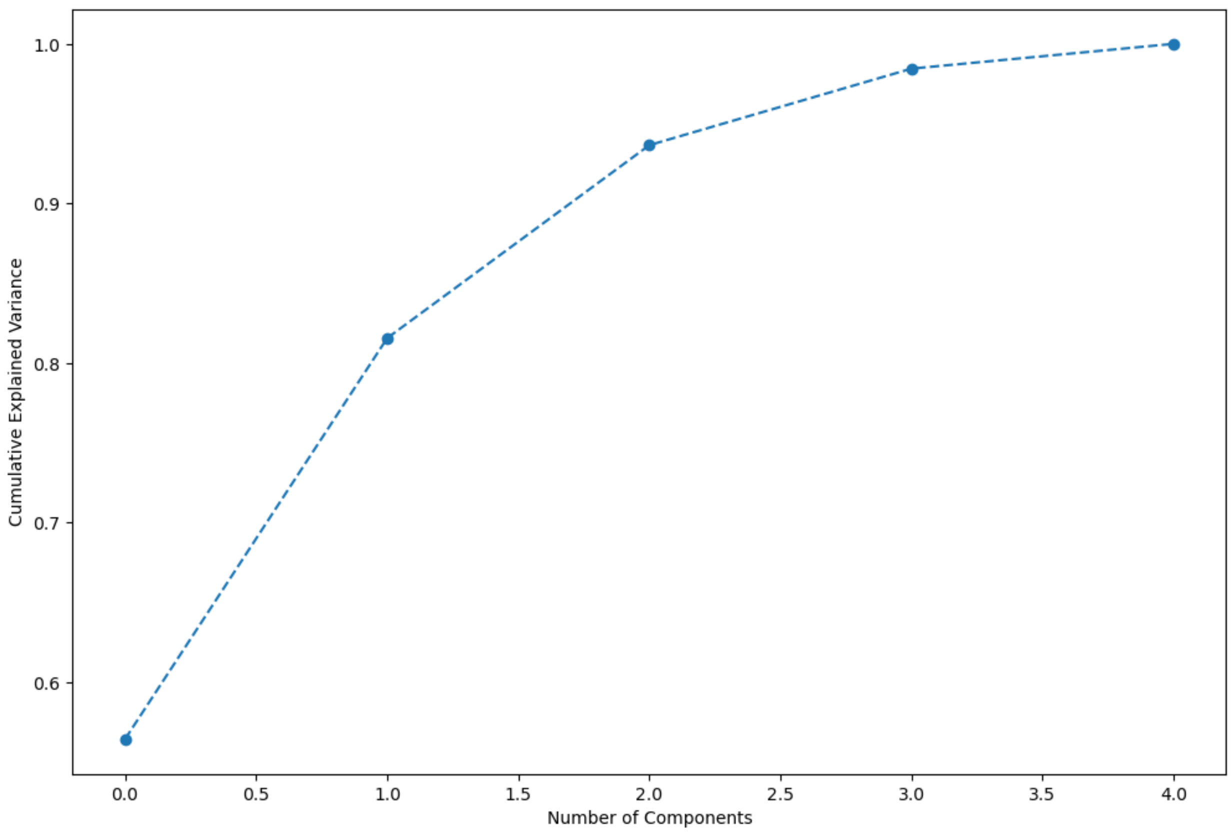
> https://deepnote.com/workspace/asish-biswas-a599-b6cca607-3c12-4ae6-b54d-32861e7e9438/project/Analytic-School-8e6c85bd-e8c9-4387-ba40-0b94fb791066/notebook/notebooks%2Fcustomer_segmentation-2956e191a051443ca73fe314b5cd9568

In [7]:
```
pca = PCA()

pca.fit(XT)

pca.explained_variance_ratio_
```

Out[7]:
```
array([0.56441556, 0.25121331, 0.12090509, 0.04792899, 0.01553704])
```

In [8]:
```
plt.figure(figsize=(12, 8))
plt.plot(range(0, 5), pca.explained_variance_ratio_.cumsum(), marker='o', lines
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
```

Out[8]:
```
Text(0, 0.5, 'Cumulative Explained Variance')
```



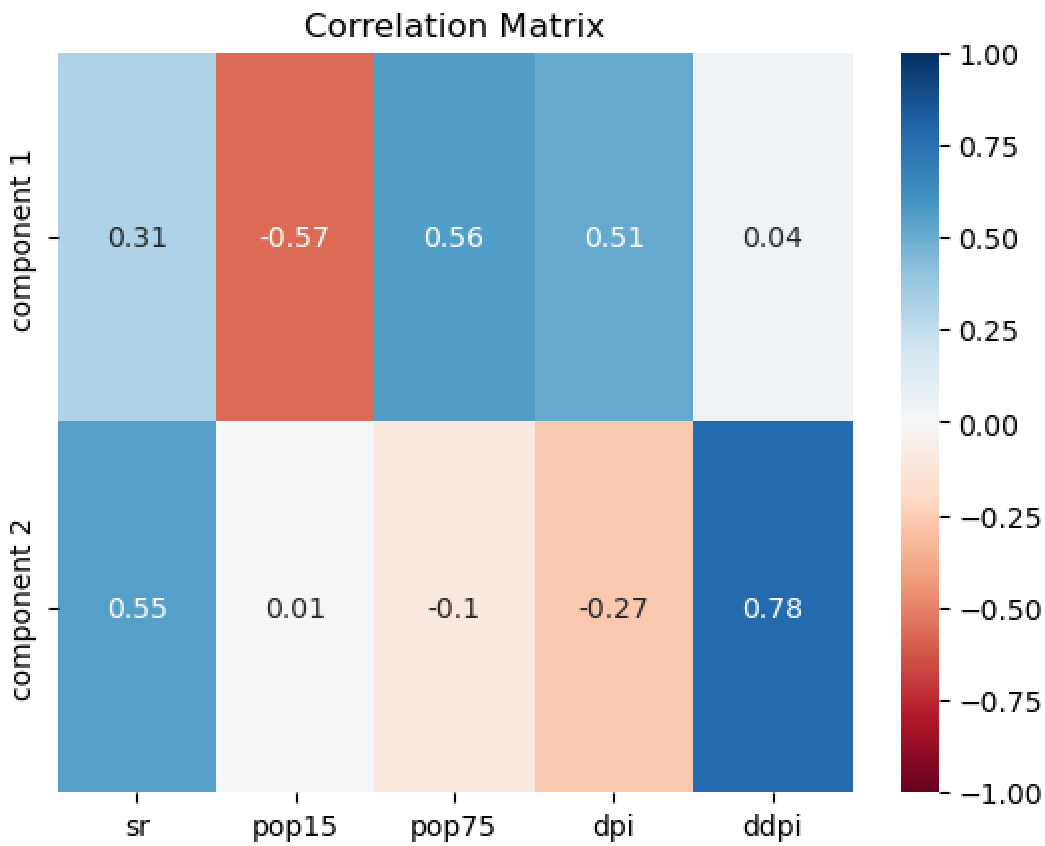
In [9]:
```
pca = PCA(n_components=2)

pca.fit(XT)

df_pca_components = pd.DataFrame(
    data=pca.components_.round(2),
    columns=XT.columns.values,
```

```
    index=['component 1', 'component 2'])

df_pca_components
```

Out[9]:

	sr	pop15	pop75	dpi	ddpi
component 1	0.31	-0.57	0.56	0.51	0.04
component 2	0.55	0.01	-0.10	-0.27	0.78

In [10]:
```
s = sns.heatmap(
    df_pca_components,
    vmin=-1,
    vmax=1,
    cmap='RdBu',
    annot=True
)
plt.title('Correlation Matrix')
plt.show()
```

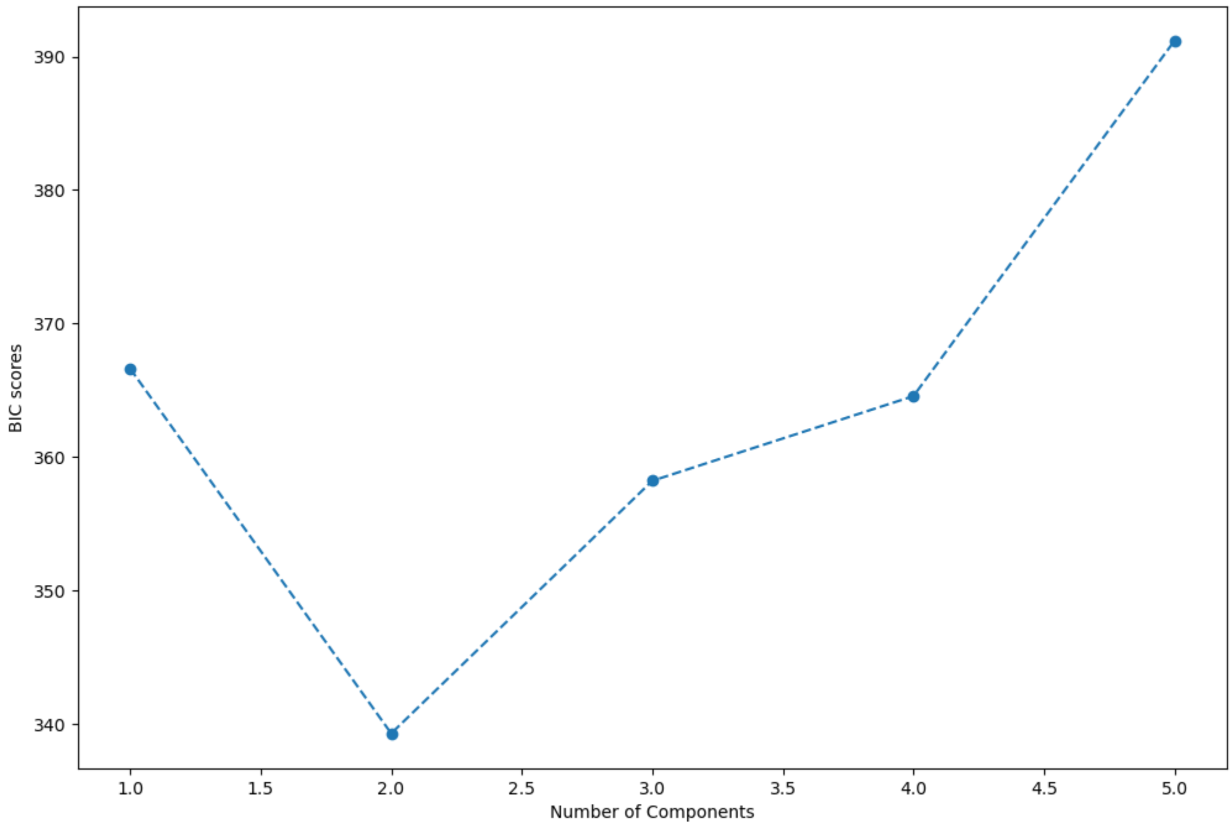


Note from the figure below how BIC is minimized with 2 components

Also note in the case where n_components = 5, each principal component would consist of one unique variable from our dataset in a 1:1 fashion--this is unfeasible

In [11]:
```
pca_scores = pca.transform(XT)

results = {}

for i in range(1, 6):
```

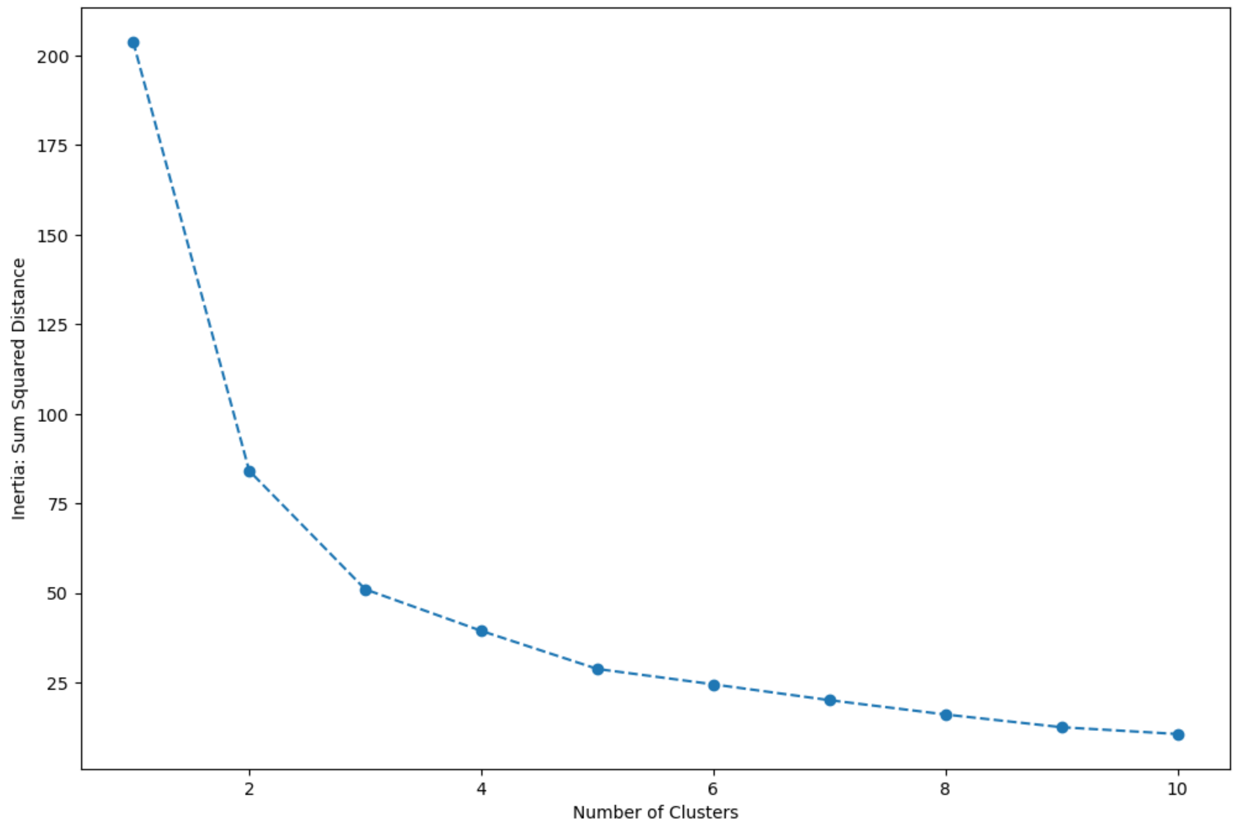
```
    gmm = GaussianMixture(n_components=i, random_state=42)
    gmm.fit(pca_scores)
    results[i] = gmm.bic(pca_scores)

plt.figure(figsize=(12, 8))
plt.plot(results.keys(), results.values(), marker='o', linestyle='--')
plt.xlabel('Number of Components')
plt.ylabel('BIC scores')
plt.show()
```



- So we have 2 components as the optimal component composition, both of which were determined from PCA and the resulting BIC scores assigned to each PCA model's tested component size.
- Next, let's see how many clusters we should use to plot our 2 components.
- Because we cannot currently estimate the number of clusters for which optimal component clustering results using our GMM predictions, I utilize K-Means and its cluster predictions to quickly estimate cluster size for this first round.

In [12]:
```
pca_scores0 = pca.transform(XT)

results = {}

for i in range(1, 11):
    kmeans_pca = KMeans(n_clusters=i, init='k-means++', random_state=42)
    kmeans_pca.fit(pca_scores0)   # pca_scores are standarzied by default
    results[i] = kmeans_pca.inertia_

plt.figure(figsize=(12, 8))
plt.plot(results.keys(), results.values(), marker='o', linestyle='--')
plt.xlabel('Number of Clusters')
```

```
plt.ylabel('Inertia: Sum Squared Distance')
plt.show()
```



K-Means suggests 3 or 4 clusters and 2 principal components will minimize error as estimated by intertia scores, so for my Gaussian models in rounds 2 and 3, I will compare performance of both 3 and 4 clusters to determine optimal cluster size that minimizes AIC and BIC.

In [13]:
```
gmm_pca = GaussianMixture(n_components=4, covariance_type='full', random_state=
gmm_pca.fit(pca_scores)
```

Out[13]:
```
GaussianMixture(n_components=4, random_state=42)
```

In [14]:
```
# DataFrame showing row-by-row cluster assignment across countries in the datas

df_segm_pca = pd.concat([XT.reset_index(drop=True), pd.DataFrame(pca_scores)],
df_segm_pca.columns.values[-2:] = ['component 1', 'component 2']

gmm_pca = GaussianMixture(n_components=4, covariance_type='full', random_state=
gmm_pca.fit(pca_scores)

df_segm_pca['GMM PCA'] = gmm_pca.predict(pca_scores)

df_segm_pca

pd.DataFrame(df_segm_pca)
df_segm_pca.head()
```

Out[14]:

	sr	pop15	pop75	dpi	ddpi	component 1	component 2	GMM PCA
0	0.396584	-0.633528	0.451558	1.246721	-0.312422	1.365290	-0.410149	0
1	0.540879	-1.299109	1.656756	0.409040	0.060682	2.049022	0.054519	0
2	0.788885	-1.246127	1.672408	1.021206	0.021964	2.416945	-0.002233	0
3	-0.884029	0.750617	-0.487557	-0.935487	-1.245184	-1.501810	-1.155836	3
4	0.723501	0.783730	-1.144938	-0.385650	0.282433	-1.053062	0.850125	1

In [15]:

```
# transform each row with respect to the mean of each gaussian

df_segm_pca_analysis = df_segm_pca.groupby(['GMM PCA']).mean().round(4)
df_segm_pca_analysis
```

Out[15]:

GMM PCA	sr	pop15	pop75	dpi	ddpi	component 1	component 2
0	0.3760	-1.0591	1.2195	1.2541	-0.2223	2.0394	-0.4360
1	0.5228	0.9173	-0.8773	-0.7898	1.7150	-1.1944	1.9415
2	0.5067	-0.6257	0.2657	-0.0616	0.3015	0.6420	0.4983
3	-0.7963	1.0449	-0.9497	-0.7846	-0.4921	-1.7957	-0.5081

In [16]:

```
# produce dataframe to show count and % data per each cluster

df_segm_pca_analysis['Count'] = df_segm_pca[['GMM PCA', 'sr']].groupby(['GMM PC
df_segm_pca_analysis['%'] = df_segm_pca_analysis['Count'] / df_segm_pca_analysi
df_segm_pca_analysis.rename(index={
    0: '0',
    1: '1',
    2: '2',
    3: '3'
}, inplace=True)

df_segm_pca_analysis
```

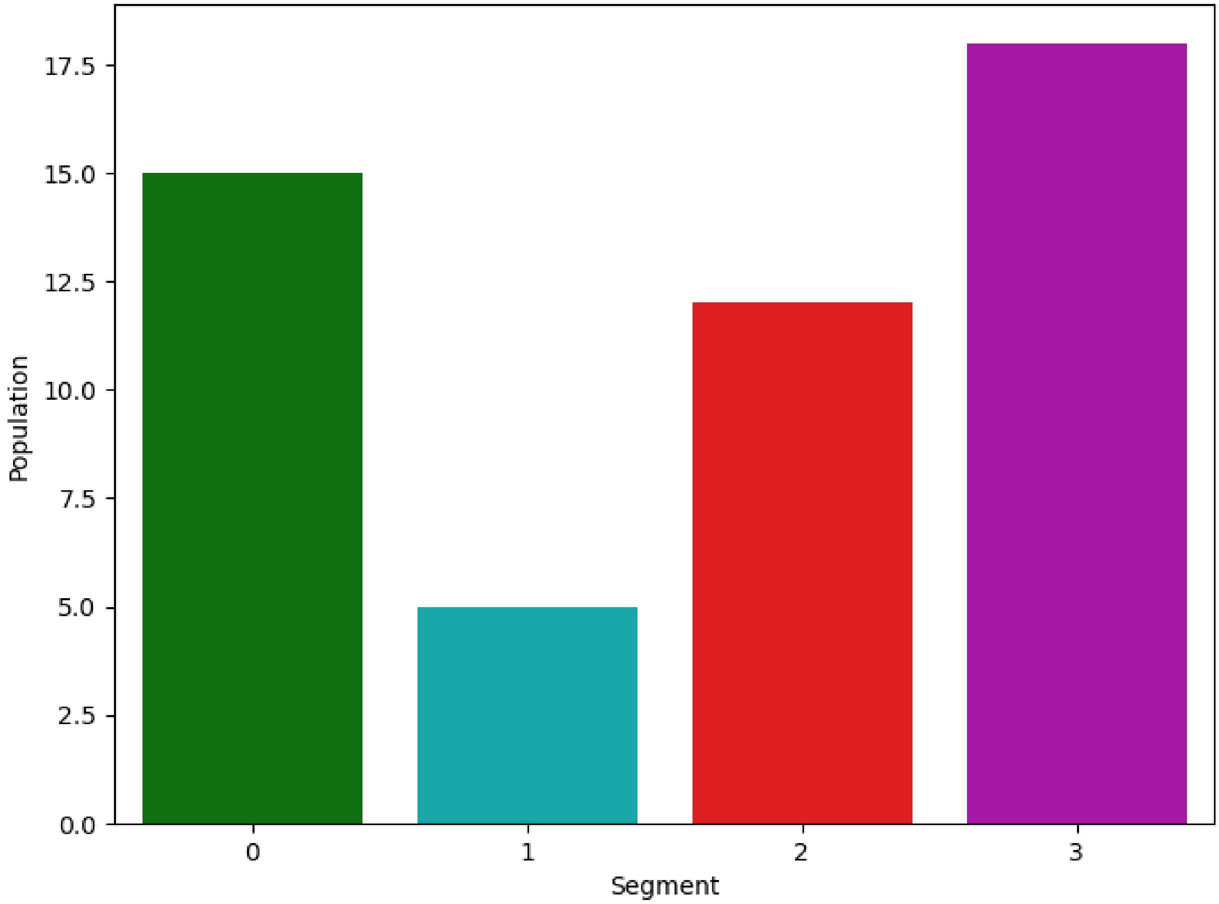
Out[16]:

GMM PCA	sr	pop15	pop75	dpi	ddpi	component 1	component 2	Count	%
0	0.3760	-1.0591	1.2195	1.2541	-0.2223	2.0394	-0.4360	15	0.30
1	0.5228	0.9173	-0.8773	-0.7898	1.7150	-1.1944	1.9415	5	0.10
2	0.5067	-0.6257	0.2657	-0.0616	0.3015	0.6420	0.4983	12	0.24
3	-0.7963	1.0449	-0.9497	-0.7846	-0.4921	-1.7957	-0.5081	18	0.36

In [17]:

```
plt.figure(figsize=(8, 6))
s = sns.barplot(data=df_segm_pca_analysis, x=df_segm_pca_analysis.index, y='Cou
plt.xlabel('Segment')
```

```
plt.ylabel('Population')
plt.show()
```



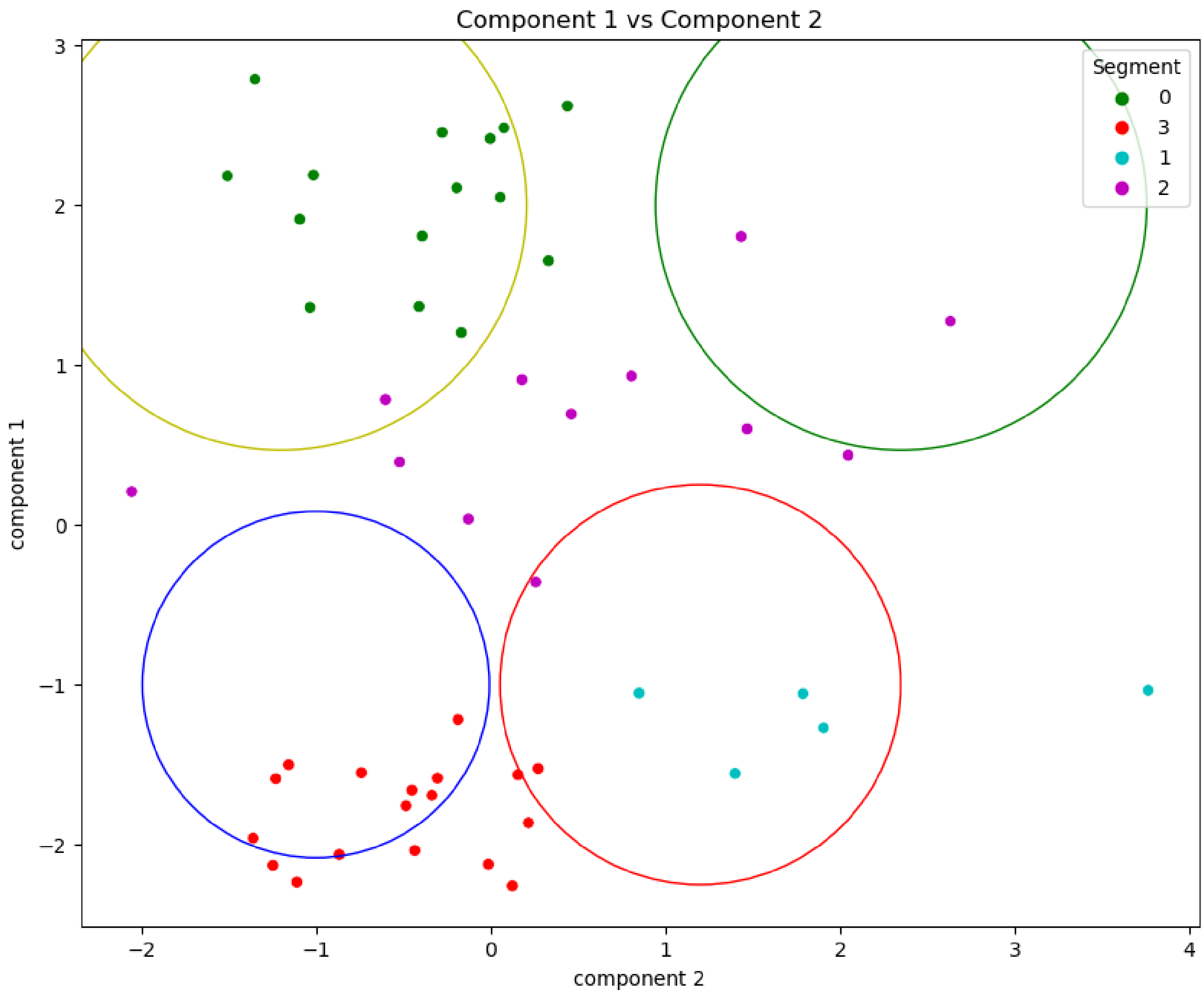
In [18]:
```
df_segm_pca['Segment'] = df_segm_pca['GMM PCA'].map({
    0: '0',
    1: '1',
    2: '2',
    3: '3'
})
pd.DataFrame(df_segm_pca)
df_segm_pca.head()
```

Out[18]:

	sr	pop15	pop75	dpi	ddpi	component 1	component 2	GMM PCA	Segn
0	0.396584	-0.633528	0.451558	1.246721	-0.312422	1.365290	-0.410149	0	
1	0.540879	-1.299109	1.656756	0.409040	0.060682	2.049022	0.054519	0	
2	0.788885	-1.246127	1.672408	1.021206	0.021964	2.416945	-0.002233	0	
3	-0.884029	0.750617	-0.487557	-0.935487	-1.245184	-1.501810	-1.155836	3	
4	0.723501	0.783730	-1.144938	-0.385650	0.282433	-1.053062	0.850125	1	

In [19]:
```
# plotting our clusters

plt.figure(figsize=(10, 8))
sns.scatterplot(
    x=df_segm_pca['component 2'],
    y=df_segm_pca['component 1'],
```

```
    hue=df_segm_pca['Segment'],
    palette=['g','r','c','m']
)
plt.scatter(2.35, 2 , s=60000, facecolors='none', edgecolors='g' )
plt.scatter(-1.2, 2 , s=60000, facecolors='none', edgecolors='y' )
plt.scatter(-1, -1 , s=30000, facecolors='none', edgecolors='b' )
plt.scatter(1.2, -1 , s=40000, facecolors='none', edgecolors='r' )

plt.title('Component 1 vs Component 2')
plt.show()
```



Round 1 Recap:

PCA recommends 4 clusters based on 2 principal components. All features from the data were used to cluster for this approach.

Please note round 2 and 3 below will also utilize the Gaussian Mixture Model (GMM) to inform its PCA and dimension reduction to provide clustering results based on these same components. However, for round 1, clustering was estimated and plotted based on K-Means distributions.

The round 2 and 3 below, in contrast, will likely predict a different cluster size than the PCA above because we explore GMM based clustering that calculate cluster distribution based on spherical covariance calculation as opposed to K-Means' circular covariance.

To get an idea what this means in terms of a specific datum's (i.e. country) cluster assignment, take a look at the table below:

In [20]:
```
# TABLE 1: K-Means PCA clustering
pca = PCA(n_components=5)
transform = pca.fit_transform(XT)
kmeans = KMeans(n_clusters=4)
X_clustered = kmeans.fit_predict(transform)
X_clustered_2d = X_clustered.reshape(-1, 1) # Reshape X_clustered as a column v
country_pairs = np.concatenate((X_clustered_2d, df['Country'].values.reshape(-1
df_clustered = pd.DataFrame(country_pairs, columns=['KM_cluster_id', 'country']


# TABLE 2: GMM PCA clustering
pca = PCA(n_components=5)
transform2 = pca.fit_transform(XT)
gmm = GaussianMixture(n_components=4)
gmm.fit(transform2)
X_clustered2 = gmm.predict(transform2)
# Create a DataFrame with cluster_id and country columns
df_clustered2 = pd.DataFrame({'GMM_cluster_id': X_clustered2, 'country': df['Co


# Cluster comparison between GMM and K-Means
df_clustered_compare =  pd.merge(df_clustered, df_clustered2, how='outer', on='
df_clustered_compare.sort_values(by='GMM_cluster_id')
```

Out[20]:

	KM_cluster_id	country	GMM_cluster_id
0	0	Australia	0
47	0	Uruguay	0
43	0	United States	0
42	0	United Kingdom	0
39	0	Switzerland	0
38	0	Sweden	0
35	3	South Africa	0
28	0	New Zealand	0
26	0	Norway	0
21	0	Italy	0
20	0	Ireland	0
18	0	Iceland	0
14	0	Germany	0
24	0	Luxembourg	0
12	0	Finland	0
1	0	Austria	0
2	0	Belgium	0
10	0	Denmark	0
5	0	Canada	0
13	0	France	0
44	3	Venezuela	1
41	1	Tunisia	1
40	1	Turkey	1
6	1	Chile	1
30	1	Panama	1
29	1	Nicaragua	1
31	1	Paraguay	1
49	1	Malaysia	1
16	1	Guatamala	1
17	1	Honduras	1
23	1	Korea	1
11	1	Ecuador	1
8	1	Colombia	1
45	3	Zambia	2
3	1	Bolivia	2

	KM_cluster_id	country	GMM_cluster_id
19	3	India	2
4	3	Brazil	2
9	3	Costa Rica	2
36	3	South Rhodesia	2
33	3	Philippines	2
32	3	Peru	2
7	3	China	2
22	2	Japan	3
34	2	Portugal	3
48	2	Libya	3
25	2	Malta	3
46	2	Jamaica	3
15	2	Greece	3
37	2	Spain	3
27	2	Netherlands	3

In [ ]:

In [21]:
```
#df2 = pd.DataFrame(transform2)
#df2 = df2[[0,1,2]] # only want to visualise relationships between first 3 proj
#df2['X_cluster'] = X_clustered
```

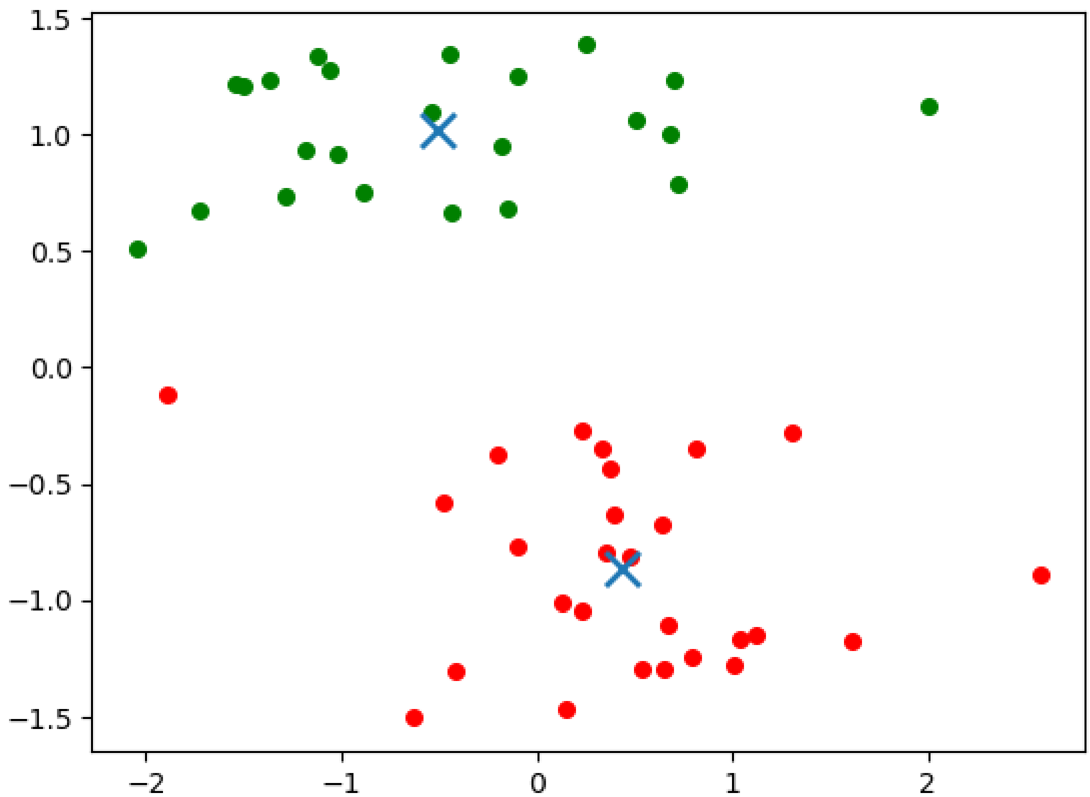
In [22]:
```
#sns.pairplot(df2, hue='X_cluster', palette= 'Dark2', diag_kind='kde',size=1.85
```

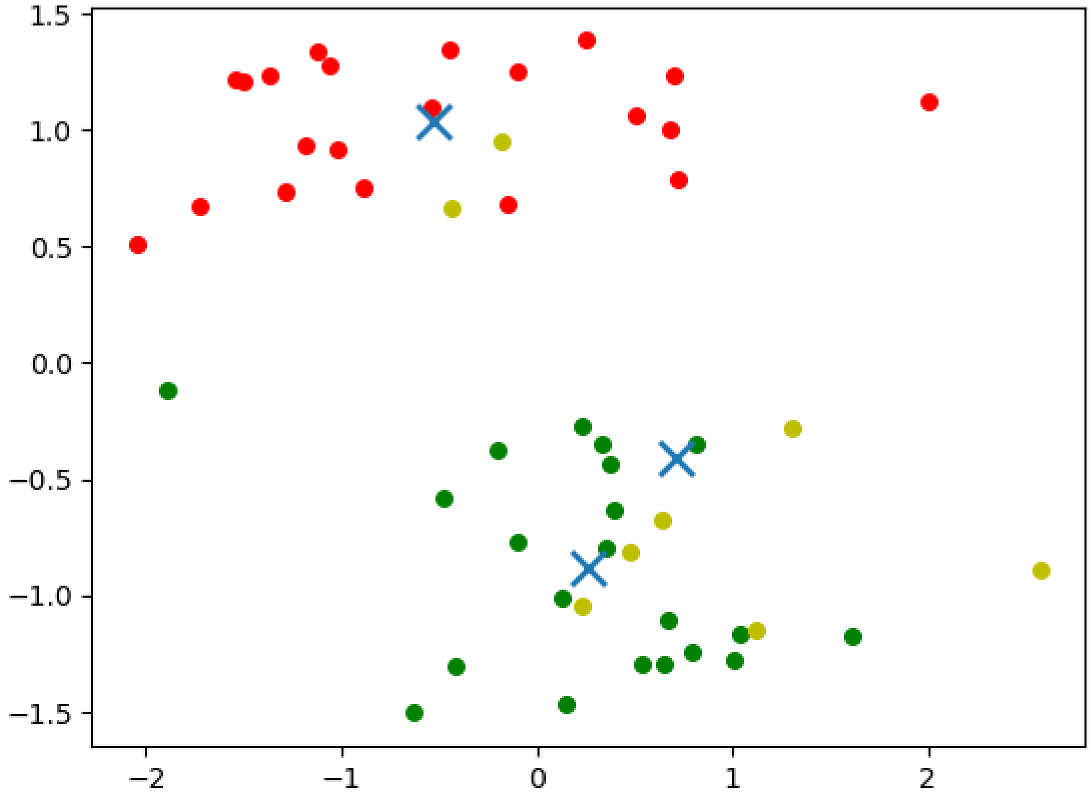
Round 2:

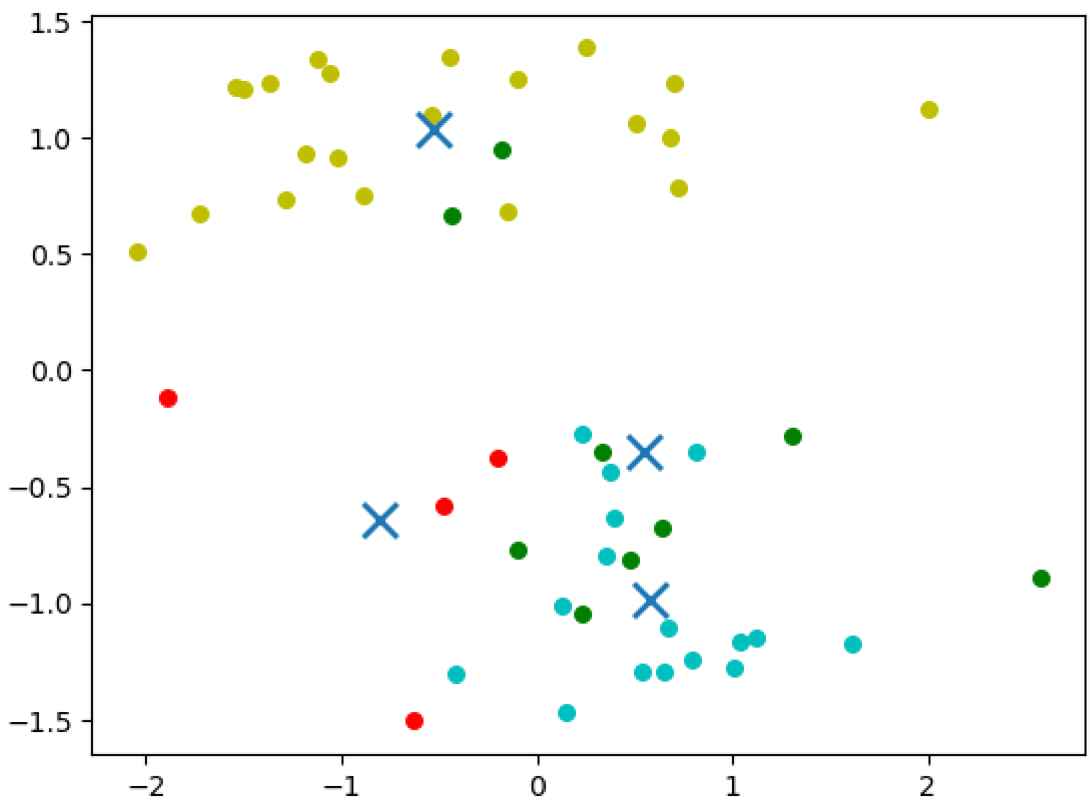
In [23]:
```
XT2 = XT2.values
```

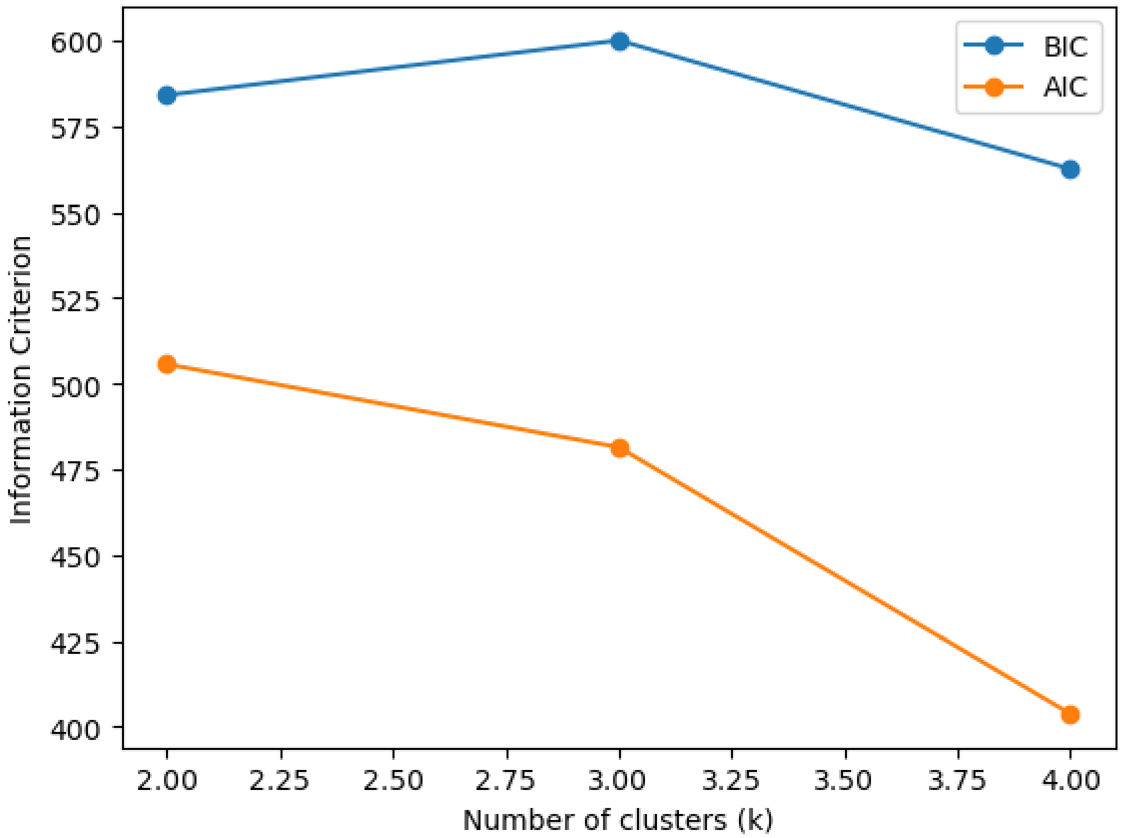
In [24]:
```
def gmm_cluster2(XT2, k_min=2, k_max=5):
    bic_errors = []
    aic_errors = []
    for k in range(k_min, k_max):
        gmm = GaussianMixture(n_components=k)
        gmm.fit(XT2)

        centroids = gmm.means_

        labels = gmm.predict(XT2)

        # print(centroids)
        # print(labels)

        colors = ['r','g','y','c']

        for i in range(k):
```

```
            plt.scatter(XT2[labels == i, 0], XT2[labels == i, 1], color=colors[

        # Plot centroids
        plt.scatter(centroids[:,0], centroids[:,1], marker= "x", s = 150, linew

        plt.show()

        bic_errors.append(gmm.bic(XT2))
        aic_errors.append(gmm.aic(XT2))

    plt.plot(range(k_min, k_max), bic_errors, marker='o', label='BIC')
    plt.plot(range(k_min, k_max), aic_errors, marker='o', label='AIC')
    plt.xlabel('Number of clusters (k)')
    plt.ylabel('Information Criterion')
    plt.legend()
    plt.show()

gmm_cluster2(XT2, k_min=2, k_max=5)
```




1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2
−1
0
1
2


1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2
−1
0
1
2



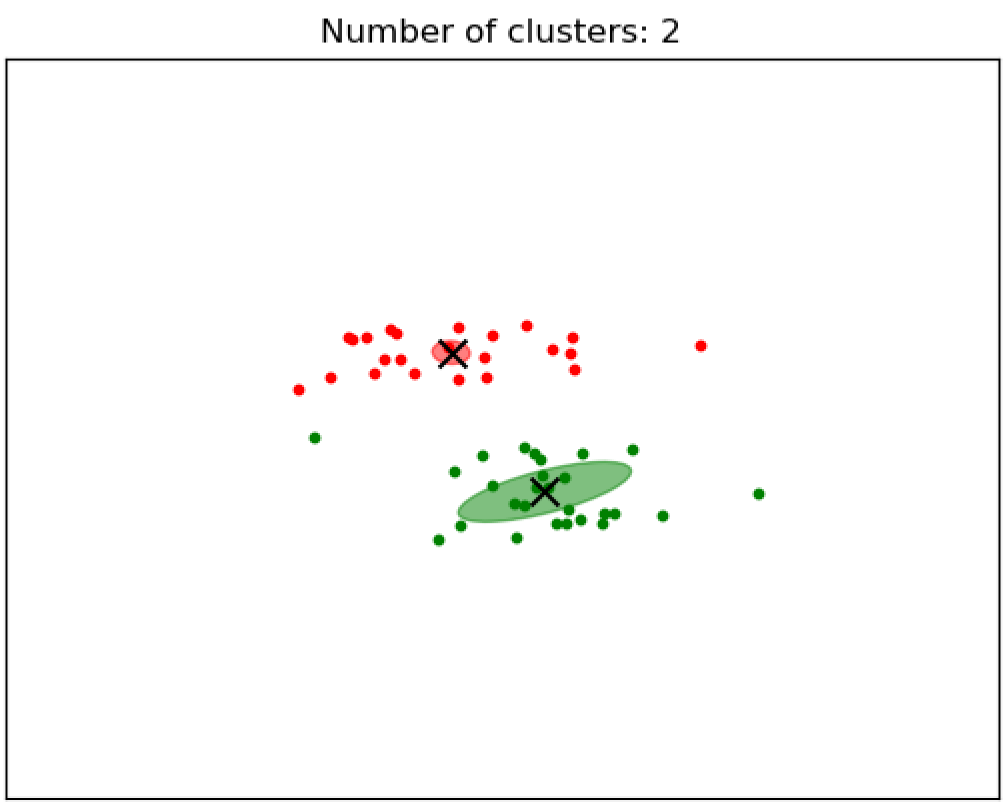
Notice:

Observe the clean clustering acheived with K = 3 clusters. Compared to the model with K = 4, there is a clear demarcation with K = 3 for each data point belonging more clearly to a specific cluster.

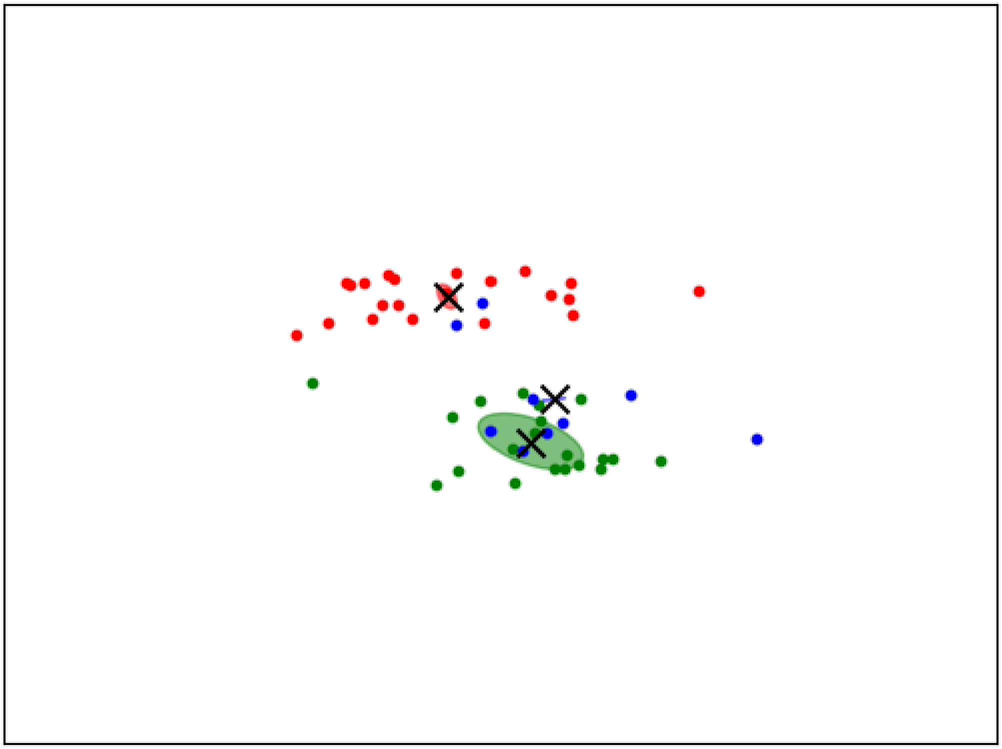
Let's plot these clusters with their entire cluster boundaries.

In [25]:
```
def plot_gmm(gmm, X, label=True, ax=None):
    if ax is None:
        ax = plt.gca()
    if label:
        labels = gmm.predict(X)
    else:
        labels = np.zeros(X.shape[0], dtype=int)
    color_iter = ['r', 'g', 'b', 'c']
    for i, (mean, covar, color) in enumerate(zip(
            gmm.means_, gmm.covariances_, color_iter)):
        v, w = np.linalg.eigh(covar)
        u = w[0] / np.linalg.norm(w[0])
        angle = np.arctan2(u[1], u[0])
        angle = 180 * angle / np.pi  # convert to degrees
        v *= 9
        ell = Ellipse(xy=mean, width=v[0], height=v[1], angle=angle,
                      color=color)
        ell.set_alpha(0.5)
        ax.add_artist(ell)
        if label:
            ax.scatter(X[labels == i, 0], X[labels == i, 1], color=color, s=10)
        else:
```

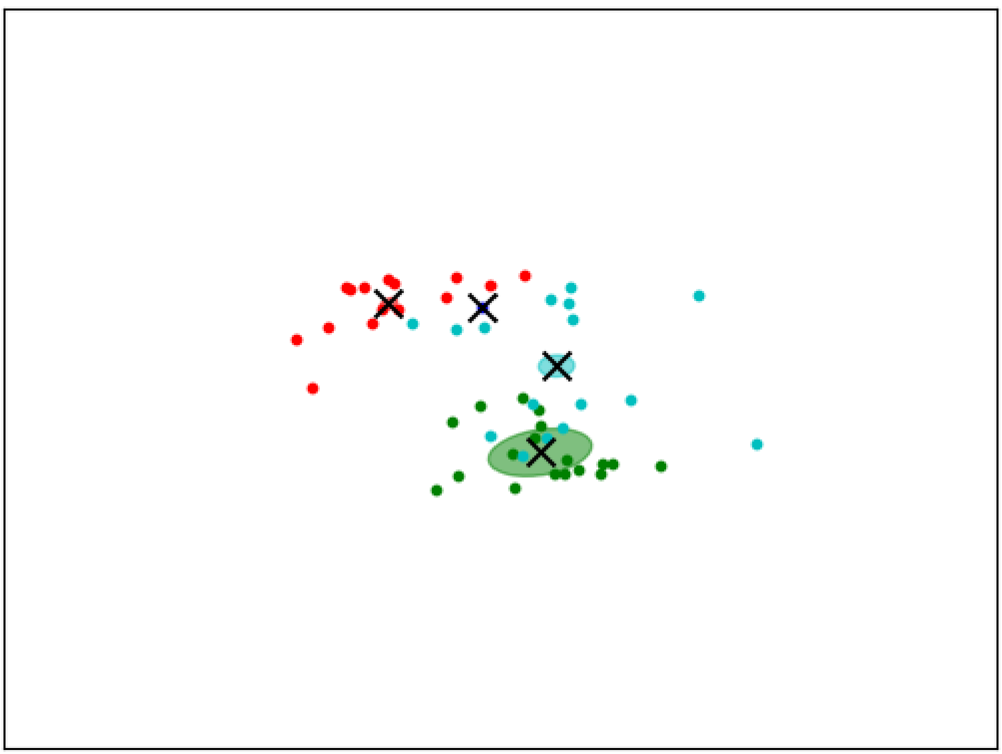
```
            ax.scatter([], [], color=color)
    ax.set_xlim(-5, 5)  # set x limits
    ax.set_ylim(-5, 5)  # set y limits
    ax.set_xticks(())
    ax.set_yticks(())
    plt.title("GMM")

def gmm_cluster3(XT2, k_min=2, k_max=5):
    bic_errors = []
    aic_errors = []
    for k in range(k_min, k_max):
        gmm = GaussianMixture(n_components=k)
        gmm.fit(XT2)

        bic_errors.append(gmm.bic(XT2))
        aic_errors.append(gmm.aic(XT2))

        fig, ax = plt.subplots()
        plot_gmm(gmm, XT2, ax=ax)
        ax.scatter(gmm.means_[:, 0], gmm.means_[:, 1], marker='x', s=100, color
        plt.title("Number of clusters: {}".format(k))
        plt.show()

    plt.plot(range(k_min, k_max), bic_errors, marker='o', label='BIC')
    plt.plot(range(k_min, k_max), aic_errors, marker='o', label='AIC')
    plt.xlabel('Number of clusters (k)')
    plt.ylabel('Information Criterion')
    plt.legend()
    plt.show()

gmm_cluster3(XT2, k_min=2, k_max=5)
```

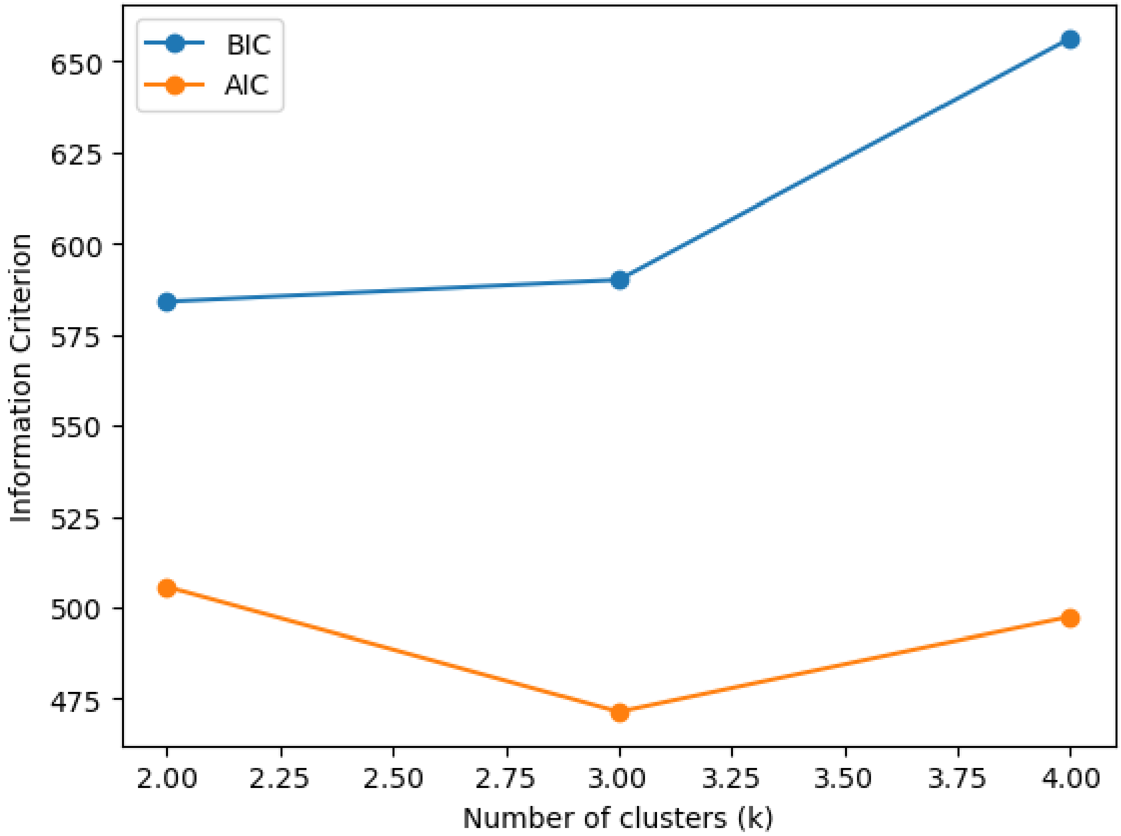


Number of clusters: 3



Number of clusters: 4





Round 2 Recap:

Based on the information score interpretation criteria discussed in CodeAcademy (see bibliograpy), the optimal cluster size is that for which the clusters v. information curve from the figure above elbows.

For round 2, the optimal cluster size based on GMM is 3.

For round 3, therefore, I will again test 3 and 4 clusters respectively and compare their performance in terms of AIC and BIC to determine the optimal clustering size for this data. However, I will adjust hyperparameters for the GMM function to see if scores can be optimized.

Round 3: Gaussian Mixture Model

Based on round 2, we see 3 clusters is likely the optimal size per the Gaussian Mixture model--but let's take a look at how hyperparameter tuning affects AIC and BIC models for the 3 and 4 cluster models, respectively:

In [26]:
```
# Took this code from
#<https://towardsdatascience.com/gmm-gaussian-mixture-models-how-to-successful

# Set the model and its parameters - 4 clusters
model3 = GaussianMixture(n_components= 4, # this is the number of clusters
                         covariance_type='spherical', # {'full', 'tied', 'diag
                         max_iter=100, # the number of EM iterations to perforn
```

```
                         n_init=1, # the number of initializations to perform.
                         init_params='kmeans', # the method used to initialize
                         verbose=0, # default 0, {0,1,2}
                         random_state=1 # for reproducibility
                        )

# Fit the model and predict labels
clust3 = model3.fit(XT3)
labels3 = model3.predict(XT3)

# Generate 10,000 new samples based on the model
smpl=model3.sample(n_samples=10000)

# Print model summary
print('*************** 4 Cluster Model ***************')
#print('Weights: ', clust3.weights_)
#print('Means: ', clust3.means_)
#print('Covariances: ', clust3.covariances_)
#print('Precisions: ', clust3.precisions_)
#print('Precisions Cholesky: ', clust3.precisions_cholesky_)
print('Converged: ', clust3.converged_)
print(' No. of Iterations: ', clust3.n_iter_)
#print('Lower Bound: ', clust3.lower_bound_)
```

```
*************** 4 Cluster Model ***************
Converged:  True
 No. of Iterations:  18
```

In [27]:

```
AIC = model3.aic(XT3)
BIC = model3.bic(XT3)

print('AIC: ', AIC)
print('BIC: ', BIC)
```

```
AIC:  574.9682244083019
BIC:  626.5928455548619
```

In [28]:

```
# Took this code from
#<https://towardsdatascience.com/gmm-gaussian-mixture-models-how-to-successfull

# Set the model and its parameters - 3 clusters
model4 = GaussianMixture(n_components= 3, # this is the number of clusters
                         covariance_type='spherical', # {'full', 'tied', 'diag
                         max_iter=100, # the number of EM iterations to perforn
                         n_init=1, # the number of initializations to perform.
                         init_params='kmeans', # the method used to initialize
                         verbose=0, # default 0, {0,1,2}
                         random_state=1 # for reproducibility
                        )

# Fit the model and predict labels
clust4 = model4.fit(XT3)
labels4 = model4.predict(XT3)

# Generate 10,000 new samples based on the model
smpl=model4.sample(n_samples=10000)

# Print model summary
print('*************** 3 Cluster Model ***************')
#print('Weights: ', clust3.weights_)
#print('Means: ', clust3.means_)
```

```
#print('Covariances: ', clust3.covariances_)
#print('Precisions: ', clust3.precisions_)
#print('Precisions Cholesky: ', clust3.precisions_cholesky_)
print('Converged: ', clust3.converged_)
print(' No. of Iterations: ', clust3.n_iter_)
#print('Lower Bound: ', clust3.lower_bound_)
```

```
*************** 3 Cluster Model ***************
Converged:  True
 No. of Iterations:  18
```

In [29]:
```
AIC = model4.aic(XT3)
BIC = model4.bic(XT3)

print('AIC: ', AIC)
print('BIC: ', BIC)
```

```
AIC:  622.5013170915375
BIC:  660.7417772001005
```

Round 3 Recap & Conclusion:

The Gaussian Modeling algorithm covered in this round suggests, based on the goal of minimizing AIC and BIC while also avoiding overfitting and entropy, that 3 clusters is indeed the optimal size.

Note that while the GMM with 4 clusters achieves a technically lower AIC and BIC score than the GMM with 3 clusters, 3 clusters is still the optimal size because AIC/BIC is not significantly higher than the GMM with 3 clusters and it still achieves a better balance between AIC/BIC and information gain.

Bibliography

Biswas, A., (2022, August 12th). Customer segmentation with python (implementing STP framework - part 3/5) [Blog post and code notebook]. Medium.

> https://deepnote.com/workspace/asish-biswas-a599-b6cca607-3c12-4ae6-b54d-32861e7e9438/project/Analytic-School-8e6c85bd-e8c9-4387-ba40-0b94fb791066/notebook/notebooks%2Fcustomer_segmentation-2956e191a051443ca73fe314b5cd9568

Causevic, S., (2020, November 26th). Implement expectation-maximization algorithm (EM) in python from scratch. towardsdatascience.com.

> https://towardsdatascience.com/implement-expectation-maximization-em-algorithm-in-python-from-scratch-f1278d1b9137

Codecademy. (n.d.). Machine Learning: Clustering Cheatsheet. Codecademy. https://www.codecademy.com/learn/machine-learning/modules/dspath-clustering/cheatsheet

Kamal, S., (2019, n.d.). Principal component analysis with kmeans. Kaggle.

> https://www.kaggle.com/code/sanikamal/principal-component-analysis-with-kmeans

OpenAI. (2021). ChatGPT. OpenAI. https://openai.com/api-docs/models/gpt-3/ (Accessed on February 17, 2023). Use cases: asked GPT on major advantages and disadvantages of K-Means and Expectation Maximization algorithms, respectively

Shah, C. (2020, p. 150-52). A hands-on introduction to data science. Cambridge University Press.

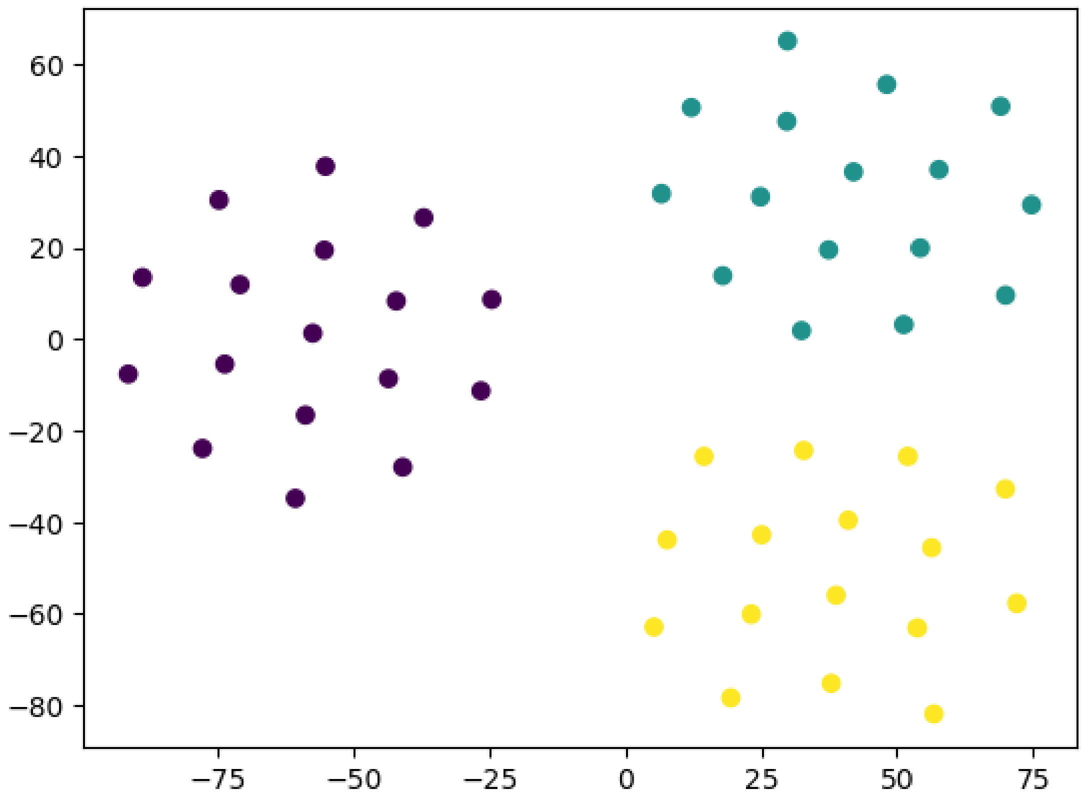
VanderPlas, J. (2023). Python data science handbook : essential tools for working with data (Second edition). O'Reilly Media, Incorporated.

Trial Round (comparing with another method that considers log-liklihood scores

In [30]:
```
# Chat GPT made this algorithm from another I found at
#<https://cmdlinetips.com/2021/03/gaussian-mixture-models-with-scikit-learn-in-

# XT3 is 5-dimensional from its features, thus
# generate some 5-dimensional data
XT3, y = make_blobs(n_samples=49, centers=3, random_state=0, n_features=5)

# Fit the Gaussian Mixture model to the data
gmm2 = GaussianMixture(n_components=3) # specify the number of clusters n
gmm2.fit(XT3)

# Predict the cluster labels for each data point
labels = gmm2.predict(XT3)

# reduce to 2D using t-SNE
XT3_embedded = TSNE(n_components=2).fit_transform(XT3) #maintains via principal

# Check that the lengths match
assert len(labels) == len(XT3_embedded)

# Create a dataframe from the clustered data
clustered_data = pd.DataFrame({'cluster': labels, 'x0': XT3_embedded[:, 0], 'x1

# Display the points in each cluster
print(clustered_data.head())

# Group the data by cluster label
grouped_data = clustered_data.groupby('cluster').count()

# Display the count of points in each cluster
print(grouped_data.head())

# plot the 2D data points colored by their cluster label
plt.scatter(XT3_embedded[:, 0], XT3_embedded[:, 1], c=y, cmap='viridis')
```

```
   cluster         x0         x1
0        0  24.877855  31.063091
1        1 -57.466995   1.291225
2        2  41.034187 -39.605911
3        1 -70.866371  11.903442
4        2  51.973980 -25.649666
         x0  x1
cluster
0        16  16
1        17  17
2        16  16
```

Out[30]: <matplotlib.collections.PathCollection at 0x7faeb0be3ca0>

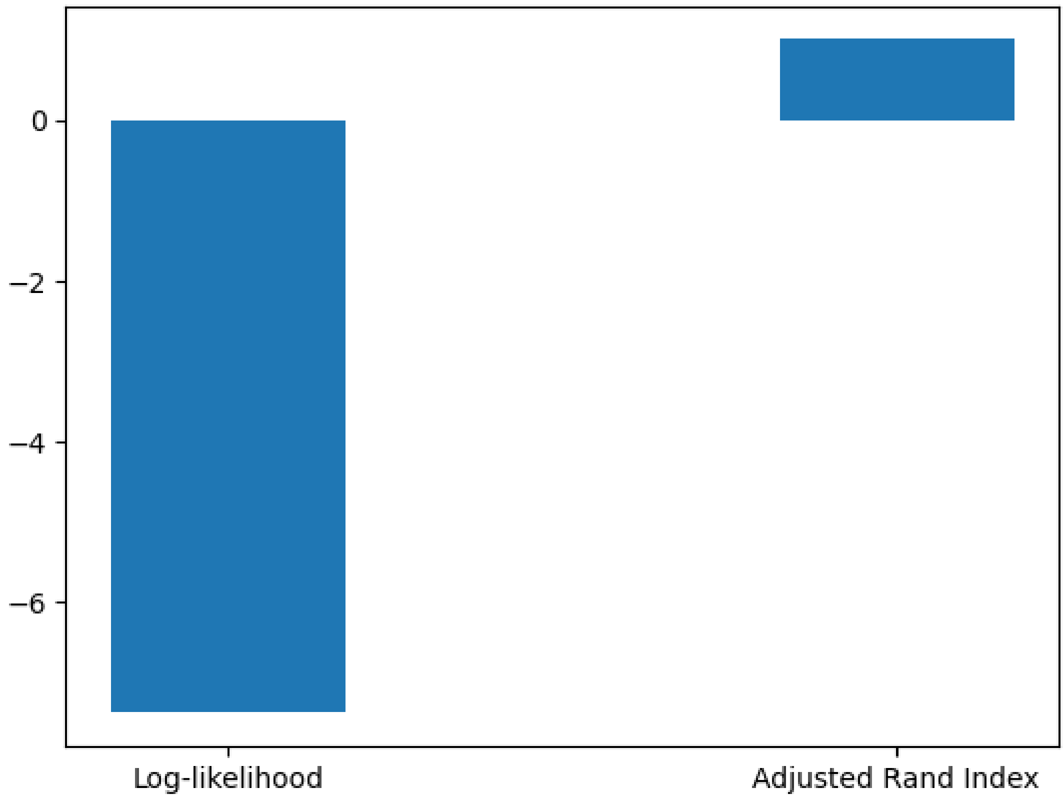


In [31]:
```
AIC = gmm2.aic(XT3)
BIC = gmm2.bic(XT3)

print('AIC: ', AIC)
print('BIC: ', BIC)
```

```
AIC:  847.7469373442896
BIC:  965.0397958271484
```

In [32]:
```
# Predict the cluster labels for each data point
labels = gmm2.predict(XT3)

# Calculate the log-likelihood of the data given the model
log_likelihood = gmm2.score_samples(XT3).mean()

# Calculate the adjusted Rand index
ari = adjusted_rand_score(y, labels)

# Create a bar chart to display the results
bar_width = 0.35
index = [0, 1]
```

```
bar_labels = ['Log-likelihood', 'Adjusted Rand Index']
metrics = [log_likelihood, ari]

plt.bar(index, metrics, bar_width)
plt.xticks(index, bar_labels)
plt.show()

# Display the log-likelihood and ARI
print("Log-likelihood:", log_likelihood)
print("Adjusted Rand Index:", ari)
```



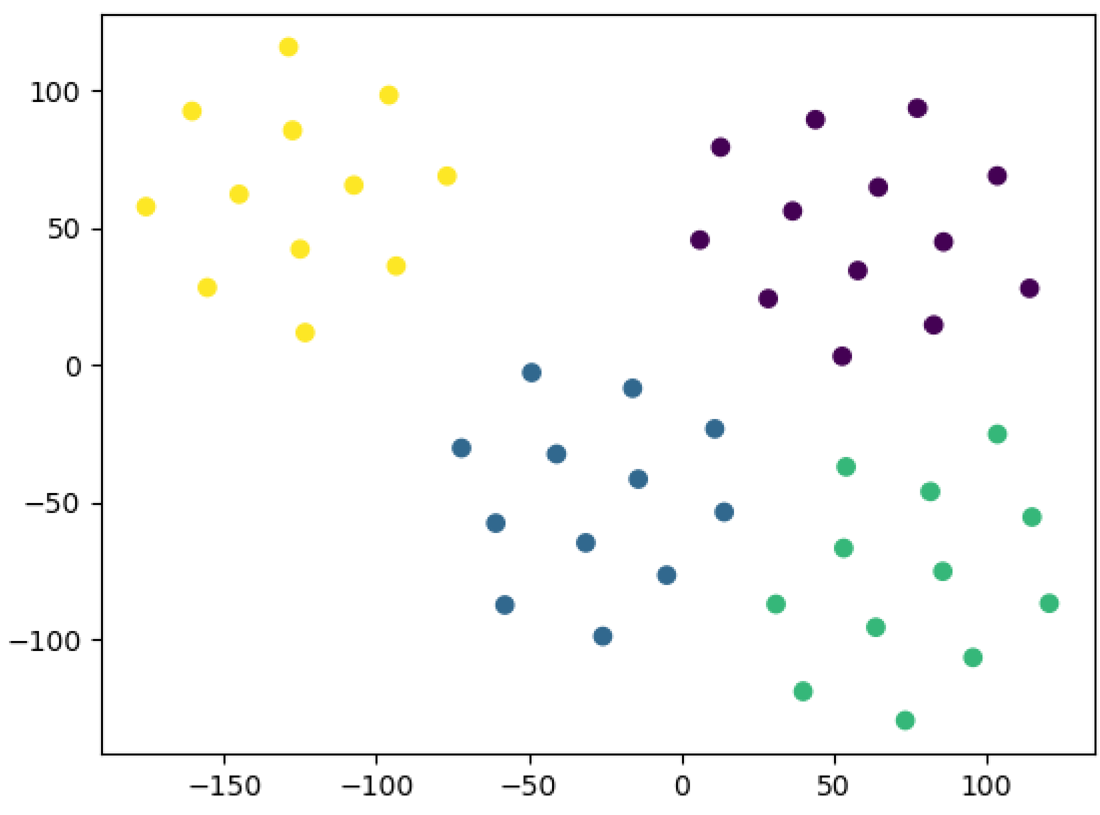

```
Log-likelihood: -7.385172830043771
Adjusted Rand Index: 1.0
```

In [33]:
```
# Chat GPT made this algorithm from another I found at
#<https://cmdlinetips.com/2021/03/gaussian-mixture-models-with-scikit-learn-in-

# XT3 is 5-dimensional from its features, thus
# generate some 5-dimensional data
XT3, y = make_blobs(n_samples=49, centers=4, random_state=0, n_features=5)

# Fit the Gaussian Mixture model to the data
gmm3 = GaussianMixture(n_components=4) # specify the number of clusters n
gmm3.fit(XT3)

# Predict the cluster labels for each data point
labels = gmm3.predict(XT3)

# reduce to 2D using t-SNE
XT3_embedded = TSNE(n_components=2).fit_transform(XT3) #maintains via principal

# Check that the lengths match
assert len(labels) == len(XT3_embedded)

# Create a dataframe from the clustered data
```

```
clustered_data = pd.DataFrame({'cluster': labels, 'x0': XT3_embedded[:, 0], 'x1

# Display the points in each cluster
print(clustered_data.head())

# Group the data by cluster label
grouped_data = clustered_data.groupby('cluster').count()

# Display the count of points in each cluster
print(grouped_data.head())

# plot the 2D data points colored by their cluster label
plt.scatter(XT3_embedded[:, 0], XT3_embedded[:, 1], c=y, cmap='viridis')
```

```
   cluster         x0         x1
0        2  52.847660   3.068260
1        2  82.782478  14.488420
2        1 -76.803337  68.803551
3        2   6.103323  45.507977
4        3 -31.371382 -64.793907
         x0  x1
cluster
0        12  12
1        12  12
2        13  13
3        12  12
```

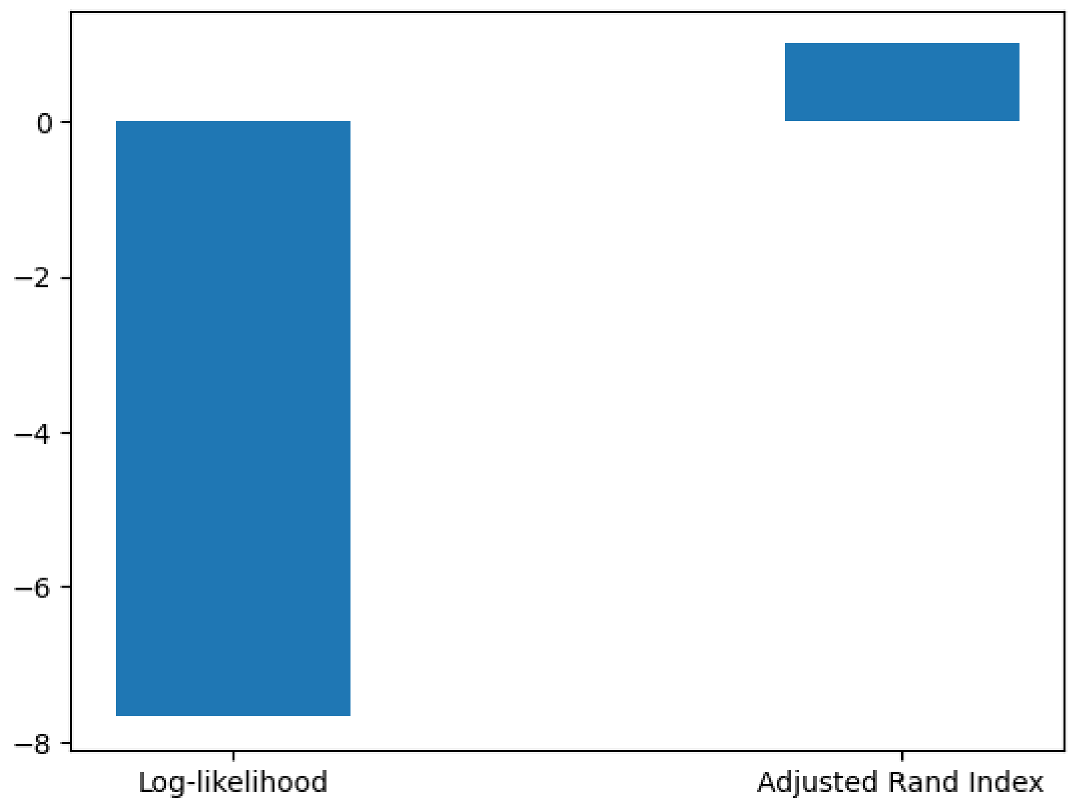
Out[33]: <matplotlib.collections.PathCollection at 0x7faeb10957f0>



In [34]:
```
AIC = gmm3.aic(XT3)
BIC = gmm3.bic(XT3)

print('AIC: ', AIC)
print('BIC: ', BIC)
```

```
AIC:  918.6045124307975
BIC:  1075.6255971739795
```

In [35]:
```
# Predict the cluster labels for each data point
labels = gmm3.predict(XT3)

# Calculate the log-likelihood of the data given the model
log_likelihood = gmm3.score_samples(XT3).mean()

# Calculate the adjusted Rand index
ari = adjusted_rand_score(y, labels)

# Create a bar chart to display the results
bar_width = 0.35
index = [0, 1]
bar_labels = ['Log-likelihood', 'Adjusted Rand Index']
metrics = [log_likelihood, ari]

plt.bar(index, metrics, bar_width)
plt.xticks(index, bar_labels)
plt.show()

# Display the log-likelihood and ARI
print("Log-likelihood:", log_likelihood)
print("Adjusted Rand Index:", ari)
```




```
Log-likelihood: -7.679637881946913
Adjusted Rand Index: 1.0
```

This trial round more clearly demonstrates 3 clusters is the optimal size per its AIC, BIC, and log-liklihood score.