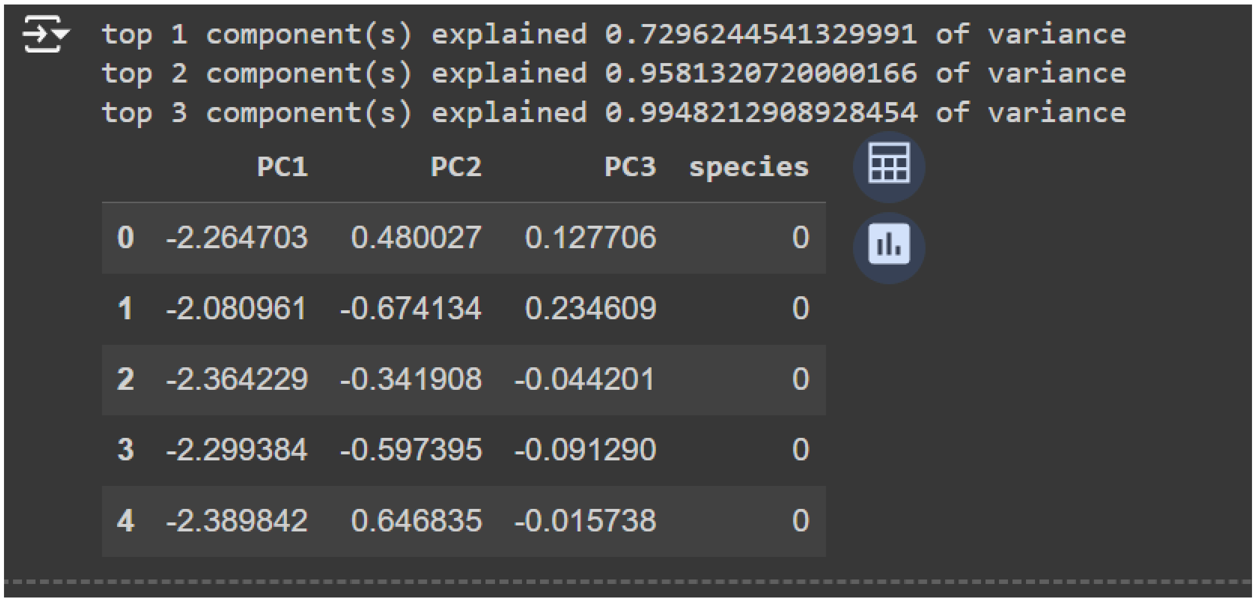
1.4 Short Answer Look the variance explained in just the top 1 and then in the top 3 (this includes top eigenvectors 1,2,3). What do you infer from it? Do you think using just the top 1 will capture the data better or all top 3?



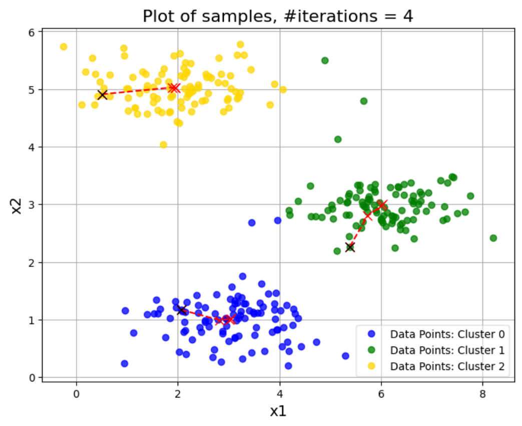
```
top 1 component(s) explained 0.7296244541329991 of variance
top 2 component(s) explained 0.9581320720000166 of variance
top 3 component(s) explained 0.9948212908928454 of variance
```

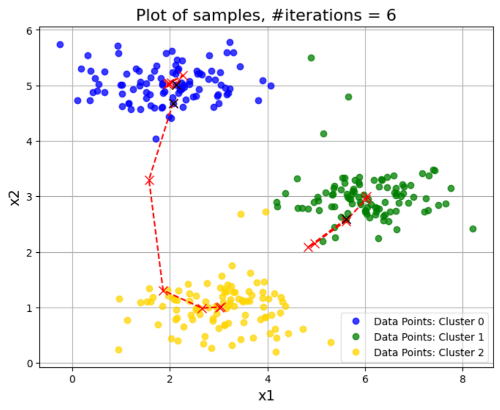
	PC1	PC2	PC3	species
0	-2.264703	0.480027	0.127706	0
1	-2.080961	-0.674134	0.234609	0
2	-2.364229	-0.341908	-0.044201	0
3	-2.299384	-0.597395	-0.091290	0
4	-2.389842	0.646835	-0.015738	0

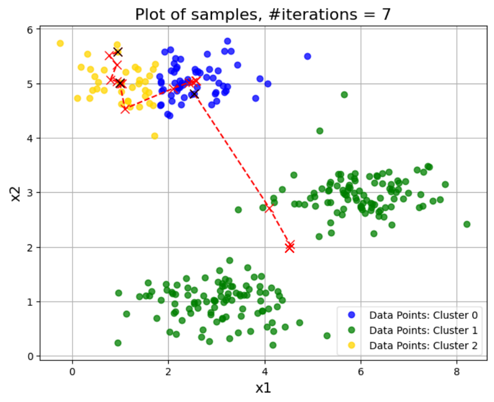
Looking at the variance in just the top 1 and then in the top 3 eigenvectors, it is very clear that the first component contains the vast majority of the variance in the dataset with over 70% of the variance. However, the top 3 eigenvectors together capture over 99% of the variance in the dataset. I believe that using the top 3 components will capture the data better than just using the top component since less information (variance) will be lost. Reducing the data to 3 key features will already greatly reduce the chance of overfitting and avoid the curse of dimensionality, so I believe that using the top 3 features will better capture the data.

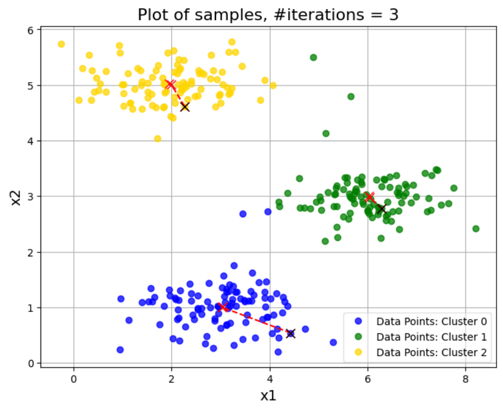
2.3 How the random initialization affects the KMeans algorithms (in terms of number of iterations, cluster assigments)?

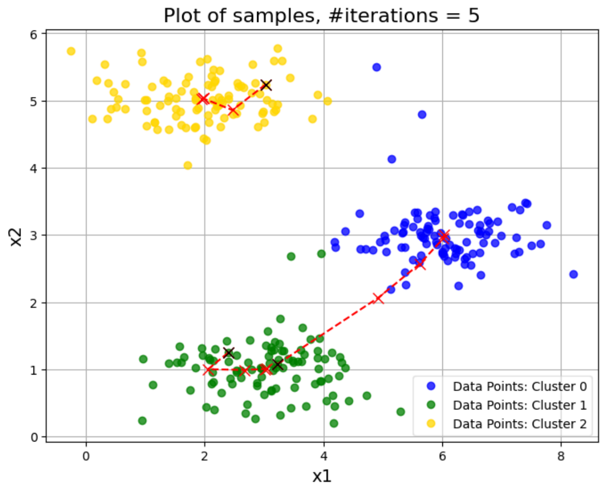
Random initialization greatly affects the results of the KMeans algorithm because KMeans isn't deterministic. If the random initialization chooses outliers as cluster centroids, or chooses two cluster centroids that are very close to each other, it can lead to poor separations of data as KMeans will split one cluster into two different classes and combine two other clusters into a single class (as shown in the image where num_iterations = 7). With random initialization and multiple runs, it seems that on average, KMeans does a good job of separating the clusters into different classes regardless of the number of iterations assigned.


Plot of samples, #iterations = 4
x1
x2
● Data Points: Cluster 0
● Data Points: Cluster 1
● Data Points: Cluster 2


Plot of samples, #iterations = 6
x1
x2
● Data Points: Cluster 0
● Data Points: Cluster 1
● Data Points: Cluster 2

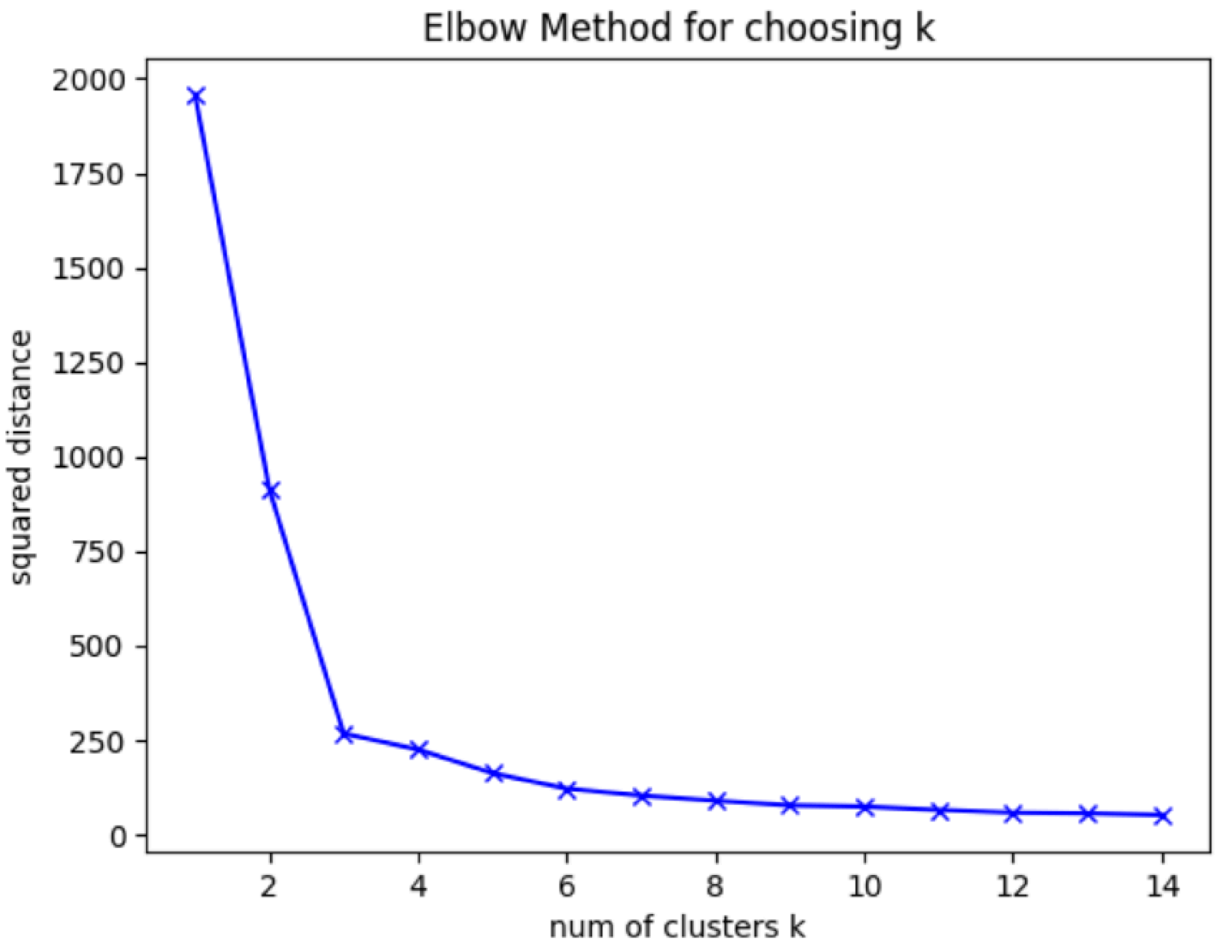

Plot of samples, #iterations = 7
x1
x2
● Data Points: Cluster 0
● Data Points: Cluster 1
● Data Points: Cluster 2


Plot of samples, #iterations = 3
x1
x2
● Data Points: Cluster 0
● Data Points: Cluster 1
● Data Points: Cluster 2


Plot of samples, #iterations = 5
x1
x2
● Data Points: Cluster 0
● Data Points: Cluster 1
● Data Points: Cluster 2

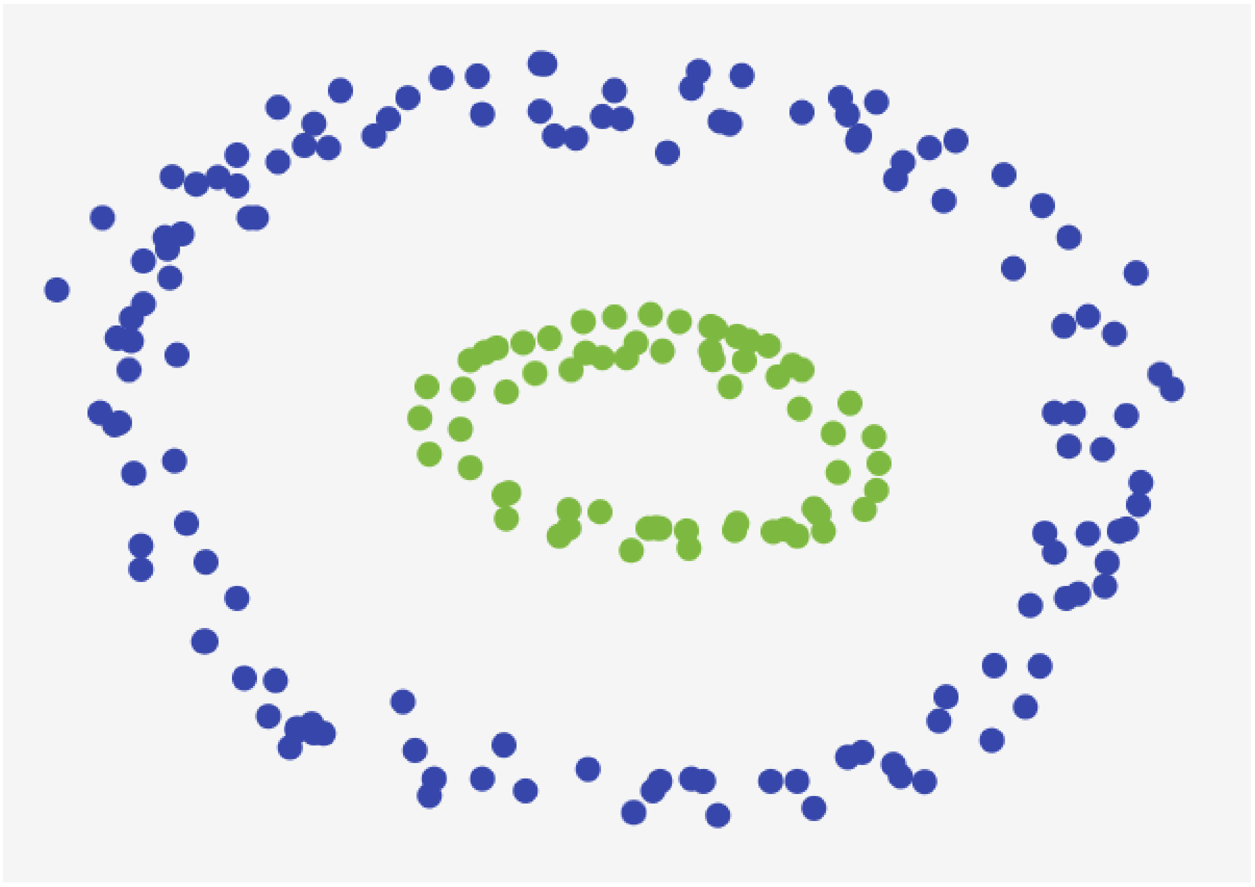
2.4 K-means - Elbow Method

Following the elbow method, k=3 is the best value for k because that is when the sum of square distances to their closest cluster center has decreased the most before there are diminishing returns. Since we want to pick the value for k that provides the lowest sum of square distances before there are diminishing returns on the extra computation, it would not be wise to choose k=14 even if it does have the least sum square distance because the returns are greatly diminished at that point. It isn’t worth the additional cost.



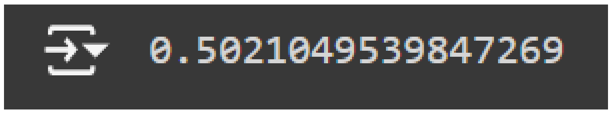
2.5 K-means on sample dataset

If K-means is applied directly to this dataset with k=2, K-means will not cluster the two clusters correctly. This is because K-means clusters based on square distance from the cluster centroid. This means that there tend to be linear separation boundaries between different clusters (where the distance between a data point between two cluster centroids is extremely similar). Since K-means tends to separate clusters linearly (using a distance function), it is unlikely that it will be able to separate the two clusters even if there is a clear circular boundary between the two clusters.



3.4 What is the chance (random) accuracy here and are we doing better than it? Is K-means best suited for this task, or would you use some other algorithm?

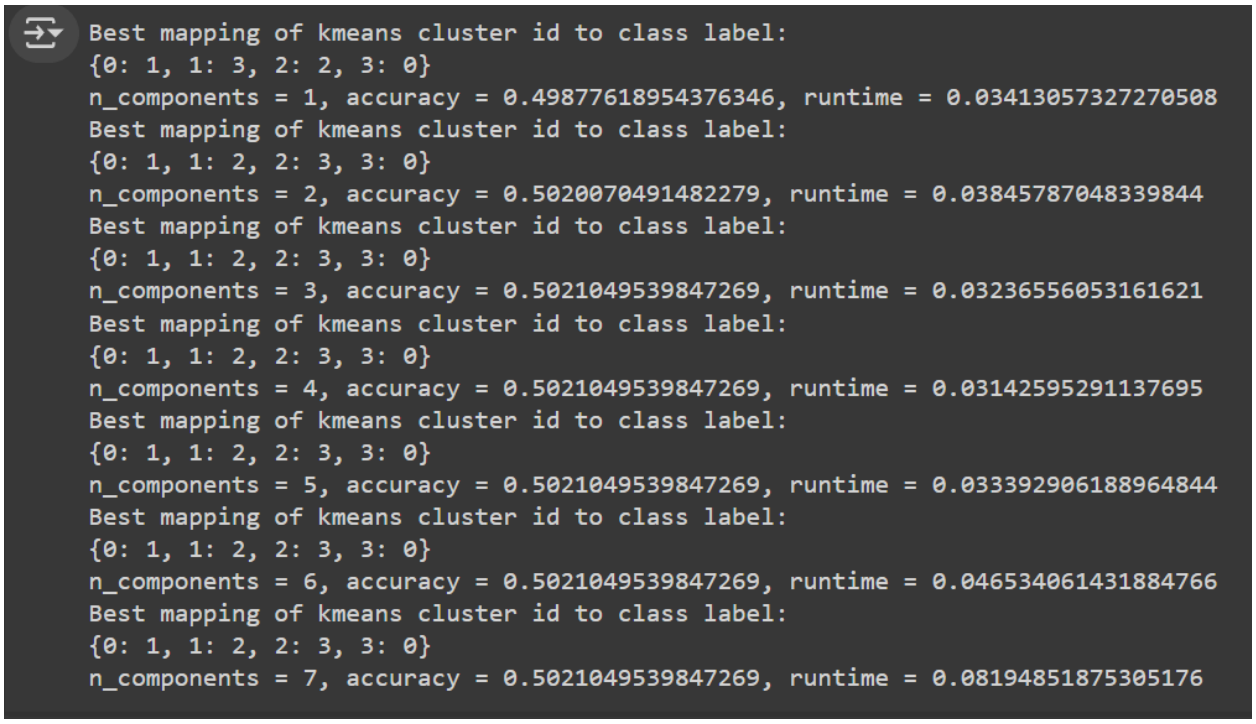
Since we know there are 4 possible classes, the random accuracy would be 25%. KMeans provides an accuracy score of 50.2% which is significantly better than random accuracy (performing twice as well as random choice). However, K-means is still getting the data wrong about half the time which shows that K-means may not be best suited for this task. This is because K-means cannot accurately separate groups that have unique relationships with each other which may not be separated well by K-means (especially when groups are not directly clustered together as shown in question 2.5).



```
0.5021049539847269
```

3.5 K-means with PCA

In question 3.3 we achieved an accuracy score of 50.2% while in this question (with n_components=7) we achieved an accuracy score of 50.2%. This shows that even when the number of components is increased, the accuracy score doesn't seem to rise. This further reinforces the hypothesis that K-means doesn't fit this dataset very well and that the relationship between data points in this dataset may not be purely based on pure clustered groups.



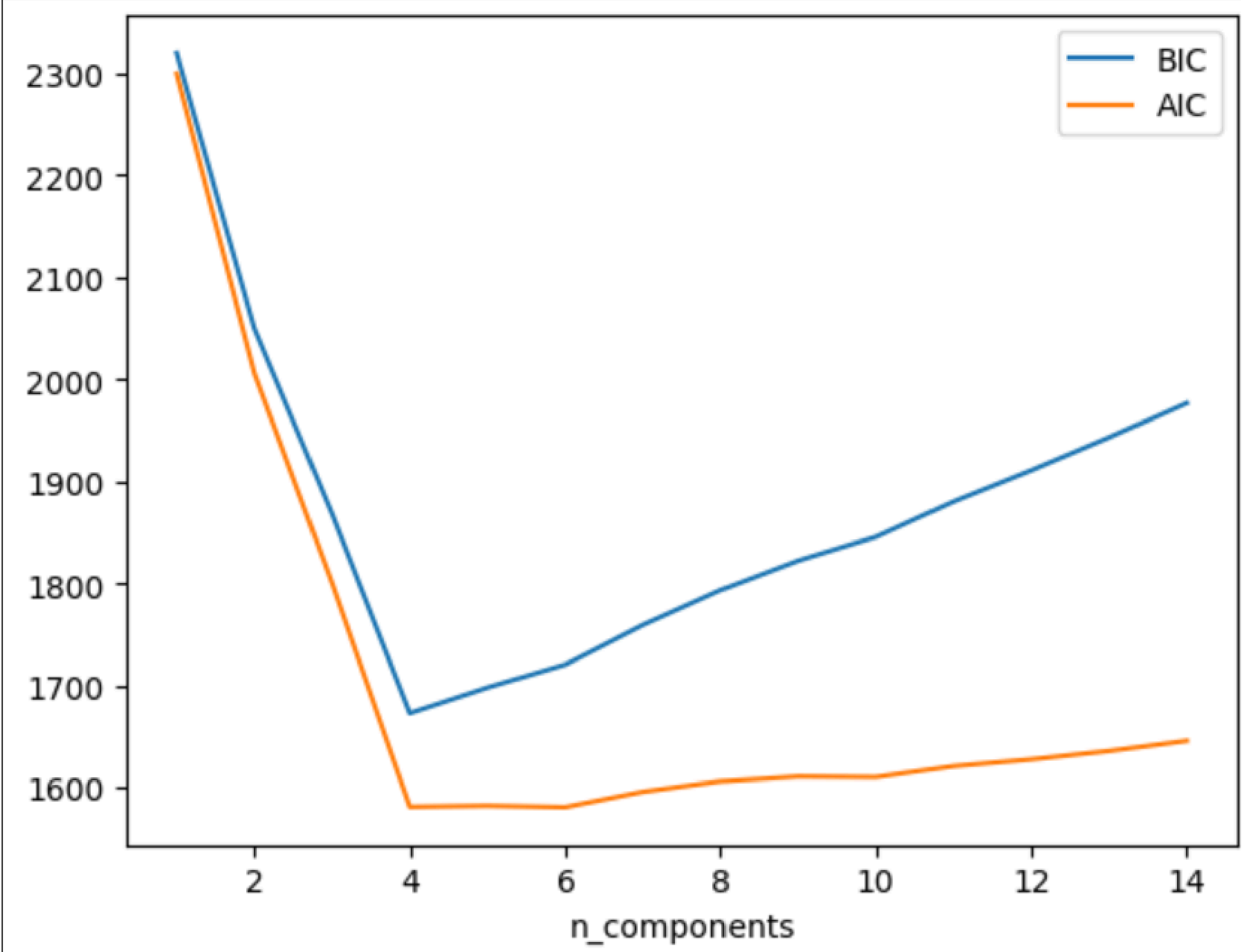
```
Best mapping of kmeans cluster id to class label:
{0: 1, 1: 3, 2: 2, 3: 0}
n_components = 1, accuracy = 0.49877618954376346, runtime = 0.03413057327270508
Best mapping of kmeans cluster id to class label:
{0: 1, 1: 2, 2: 3, 3: 0}
n_components = 2, accuracy = 0.5020070491482279, runtime = 0.03845787048339844
Best mapping of kmeans cluster id to class label:
{0: 1, 1: 2, 2: 3, 3: 0}
n_components = 3, accuracy = 0.5021049539847269, runtime = 0.03236556053161621
Best mapping of kmeans cluster id to class label:
{0: 1, 1: 2, 2: 3, 3: 0}
n_components = 4, accuracy = 0.5021049539847269, runtime = 0.03142595291137695
Best mapping of kmeans cluster id to class label:
{0: 1, 1: 2, 2: 3, 3: 0}
n_components = 5, accuracy = 0.5021049539847269, runtime = 0.033392906188964844
Best mapping of kmeans cluster id to class label:
{0: 1, 1: 2, 2: 3, 3: 0}
n_components = 6, accuracy = 0.5021049539847269, runtime = 0.046534061431884766
Best mapping of kmeans cluster id to class label:
{0: 1, 1: 2, 2: 3, 3: 0}
n_components = 7, accuracy = 0.5021049539847269, runtime = 0.08194851875305176
```

4.3 Soft clusters vs Hard clusters

Hard clustering is when every data point in the dataset is assigned a definitive class label (or cluster label). Instead of assigning a single class label to each data point, soft clustering assigns each point a probability that the data point belongs to that class label and uses the class label with the highest probability as the predicted class label. K-means clustering uses hard clustering since each data point is assigned a definitive class label. GMM clustering uses soft clustering since it identifies probability distributions for each class and assigns these probabilities to each data point.

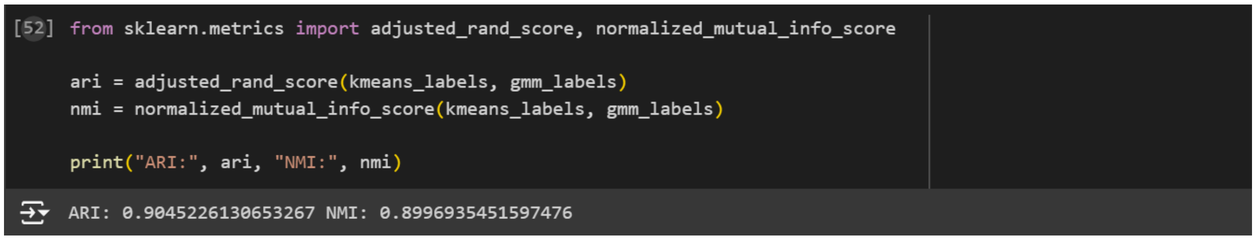
4.4 Number of Components

The value that should be chosen for the n_components is 4. Since better models have better scores for both AIC and BIC, n_components should be chosen based on which value has the lowest AIC and BIC scores. Referencing the graph below, we can see that AIC and BIC sharply drop until n_components equals 4 before starting to rise again when n_components increase past 4 slowly. Therefore, 4 is the best choice for the number of components.



4.7 ARI and NMI

We use the Adjusted Rand Index and the Normalized Mutual Information score as metrics for evaluating clustering performance because these metrics show how similar two sets of labels are and how much information overlap exists between them. The Adjusted Rand Index compares how similar two sets of labels are while accounting for random chance. This shows more information about how similar the two sets of predicted labels are and how much better the predictions line up with each other than just random predictions. Normalized Mutual Information shows how much information is shared between the two sets of labels. We are comparing the predictions of two different clustering models, so the NMI is useful to see how much original information was preserved in both predicted sets of labels from the original dataset.

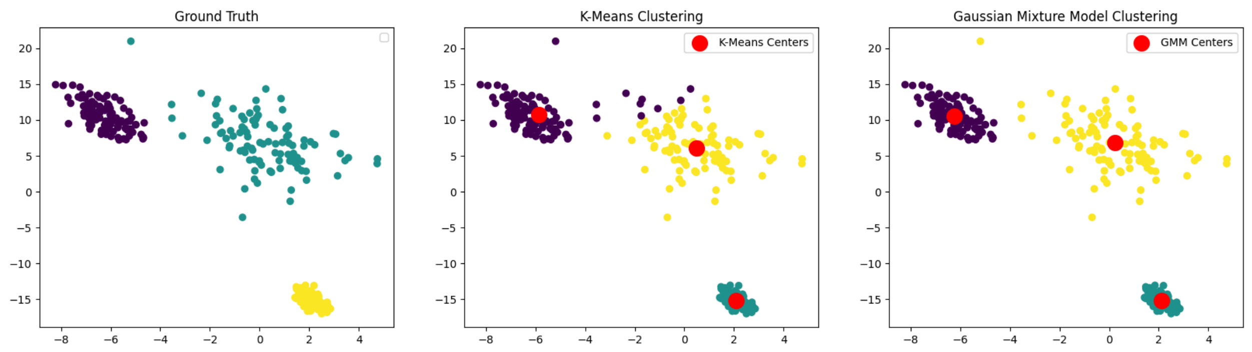


```
[52] from sklearn.metrics import adjusted_rand_score, normalized_mutual_info_score

     ari = adjusted_rand_score(kmeans_labels, gmm_labels)
     nmi = normalized_mutual_info_score(kmeans_labels, gmm_labels)

     print("ARI:", ari, "NMI:", nmi)
```

```
ARI: 0.9045226130653267 NMI: 0.8996935451597476
```

4.8 K-means vs GMM

Comparing the performance of K-means with GMM, I believe that Gaussian Mixture Models perform better between the two algorithms because it can account for different distributions of clusters. K-means directly labels data based on its proximity to the closest cluster centroid, which makes it unable to account for the fact that different clusters may have different properties. The flexibility from the probability distribution allows GMMs to better model different clusters that follow different distributions.


Ground Truth
K-Means Clustering
K-Means Centers
Gaussian Mixture Model Clustering
GMM Centers