CA2 Assignment Part I - Applying Clustering to Iris Dataset

Contents

Dataset and problem definition

For this dataset, we want to find out the number of species of flowers, based on dimensions of their petals and sepals. However, we must now assume that we **don't know** the number of clusters. We need to apply a clustering algorithm to help us tease out the "natural structure" within the data. Our task therefore is to:

a. Apply a clustering algorithm to predict labels of different classes
b. Determine the best number of classes to assume
c. [In a real life scenario] Apply these labels of k different classes to the business use case for decision making
d. [In this assignment] Check back against the ground truth to see how accurate has our prediction been.

Standard approach that has been used and two visualisations

In practical 5, an example was provided on how k-means clustering is applied to determine the number of possible clusters in the dataset. I am reproducing it here first, before adding on to the approach, by calling up a 3D plot to visualise an additional dimension. I will also try out other clustering algorithms subsequently, and finally compare their predictions against the ground truth (labels in the dataset).

In [1]:

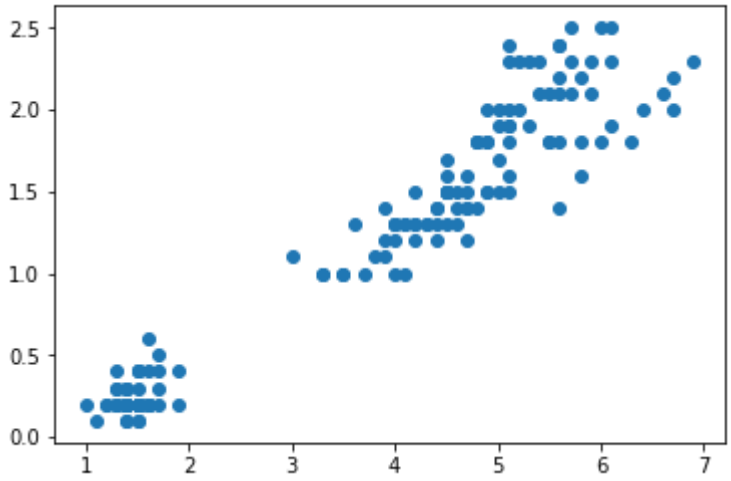
```
#Importing the essential packages
from sklearn.cluster import KMeans
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
%matplotlib inline
```

In [2]:

```
#Iris Dataset
iris = load_iris()
X = iris.data
plt.scatter(X[:, 2], X[:, 3])
```

Out[2]:

```
<matplotlib.collections.PathCollection at 0x20262f64b70>
```



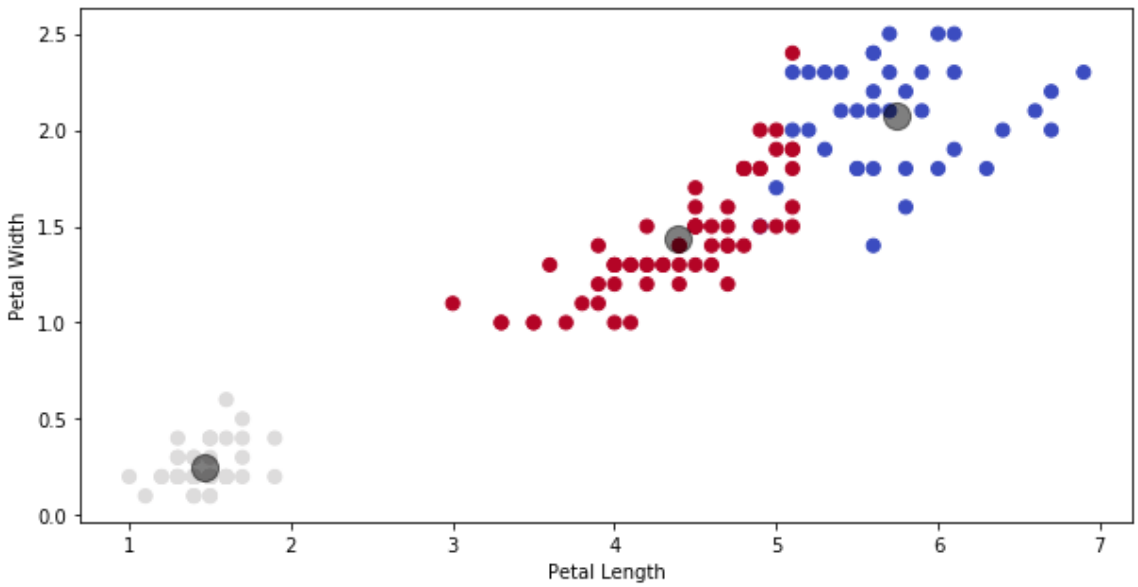
A quick inspection of the data will show us a very clear cluster at the bottom left of the screen. The large group at the top right could contain multiple clusters or one cluster (although through background info, we know that there are 2 clusters in that group, leading to 3 different classes across the dataset). Let us first apply the k-means algorithm with that knowledge, before moving into the next part (where we assume that we do not know k).

In [3]:

```
#Plotting - a 2D visualisation showing 2 axes
import matplotlib.pyplot as plt

#Fit the k-means algorithm
km = KMeans(n_clusters=3)
km.fit(X)
y_kmeans = km.predict(X)

# scatter plot the petal length (column 2), petal width (column 3)
fig = plt.figure(figsize=(10,5))
plt.scatter(X[:, 2], X[:, 3], c=y_kmeans, s=50, cmap='coolwarm')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

# show centroid centres as grey circle opacity 50%
centers = km.cluster_centers_
plt.scatter(centers[:, 2], centers[:, 3], c='black', s=200, alpha=0.5)
```

Out[3]:

```
<matplotlib.collections.PathCollection at 0x2026341cac8>
```



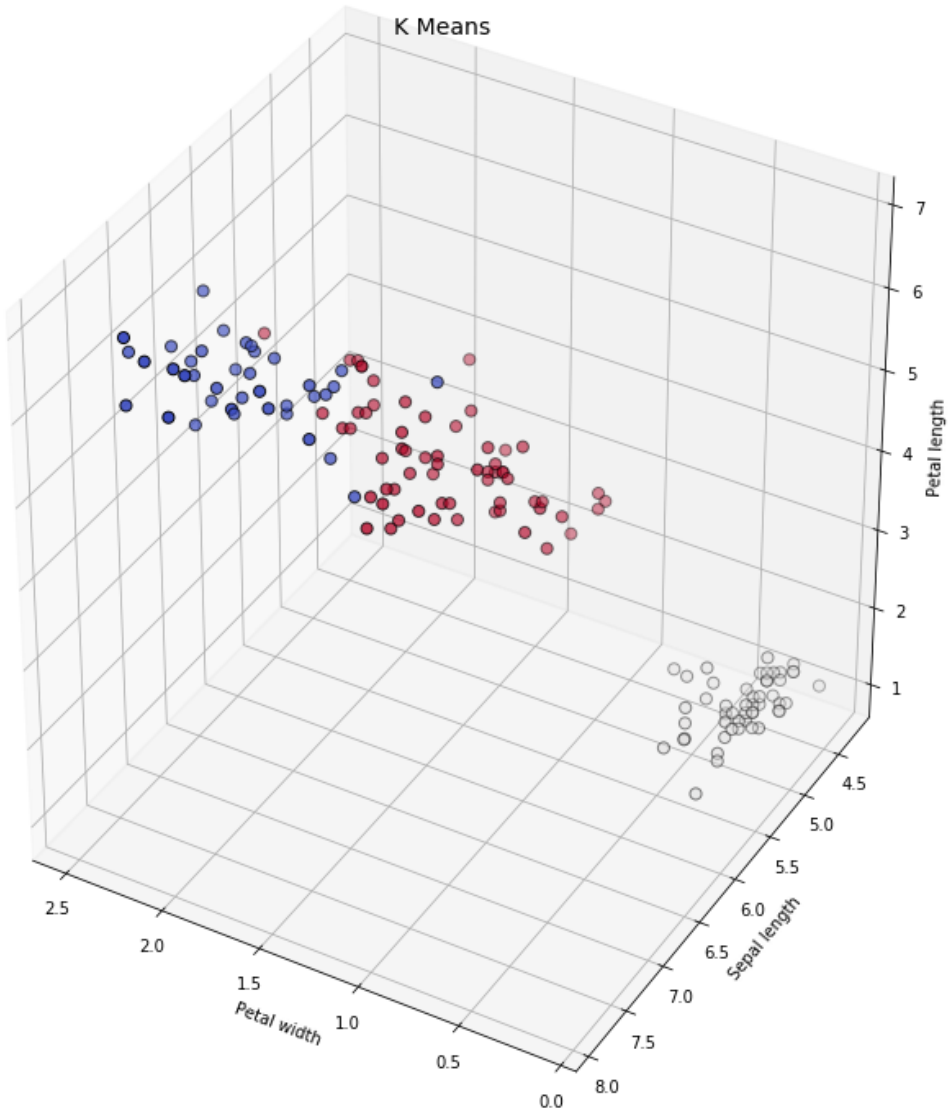
Because the canvas is 2D, we are able to only represent the data on 2 dimensions out of 4. If we want a more nuanced visualisation, there are several ways we can do it. One is to apply PCA to reduce 4 dimensions to 2; the other, which I have tried out below, is to use Matplotlib's 3D axes to create a 3D canvas space, so that I can capture the Euclidean spaces between 3 different features.

In [4]:

```
#Plotting - a 3D visualisation showing 3 axes

fig = plt.figure(1, figsize=(10,10))
ax = Axes3D(fig, rect=[0, 0, 0.95, 1], elev=35, azim=120)
ax.scatter(X[:, 3], X[:, 0], X[:, 2],
           c=y_kmeans.astype(np.float), edgecolor="k", s=50, cmap='coolwarm')
ax.set_xlabel("Petal width")
ax.set_ylabel("Sepal length")
ax.set_zlabel("Petal length")
plt.title("K Means", fontsize=14)
```

Out[4]:

```
Text(0.5,0.92,'K Means')
```



Now that we have the Euclidean visualisation of the 3 possible clusters, we can call up the predictions based on k-means clustering. This is shown below:

In [5]:

```
print(y_kmeans)
```

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

Going blind - assuming we don't have the label answers

In this unsupervised learning task however, we assme that we don't know how many clusters there are. This will have to be organically "sensed" by running a clustering algorithm (or a set of different ones). We can iteratively find out what is the optimal number of clusters through various metrics.

I will be trying four different algorithms, with my selection based on what I've read in the scikit-learn documentation:

1. K-Means Clustering: This is our first general purpose clustering algorithm, works best on even cluster size, and as long as the number of clusters is not too high.
2. Spectral Clustering: This is able to accomodate non-flat geometry of the points, hence it could be a good additional algorithm to try out in addition to K-Means.
3. Agglomerative Clustering: This algorithm is not affected even if the dataset inherently has many clusters, and can accomodate non-Euclidean distances. It is a hierarchical clustering algorithm.
4. DBSCAN: This algorithm does not need us to specify the number of clusters, and makes its own label predictions based on density. It could be used as a good reference point to see if the other algorithms' optimal number of clusters is way off.

K-Means Clustering: Determining the Optimal Number of Clusters

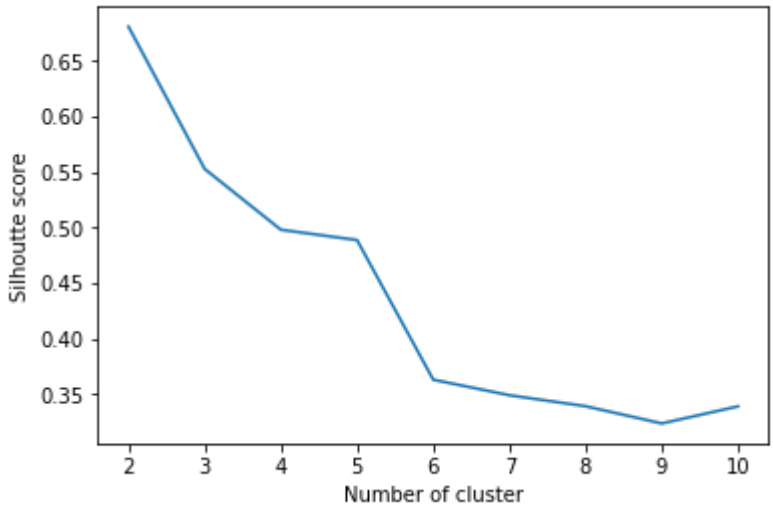
There are several metrics to determine the optimal number of clusters. I focus on 2 here:

1. Silhoutte score: a measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation). The higher the score, the greater the degree of cohesion amongst the clusters. This was covered in Practical 5. However, we saw that it was not very effective, due to the overlap of 2 out of 3 clusters
2. Sum of squared errors: a measure of how internally coherent clusters are, by minimising the sum of squares in relation to an assigned centroid. The lower the score, the lower the error and hence the greater the degree of cohesion amongst the clusters. This measure was not covered in practical 5, but is a very useful attribute of the K-Means algorithm, hence I decided to give it a try.

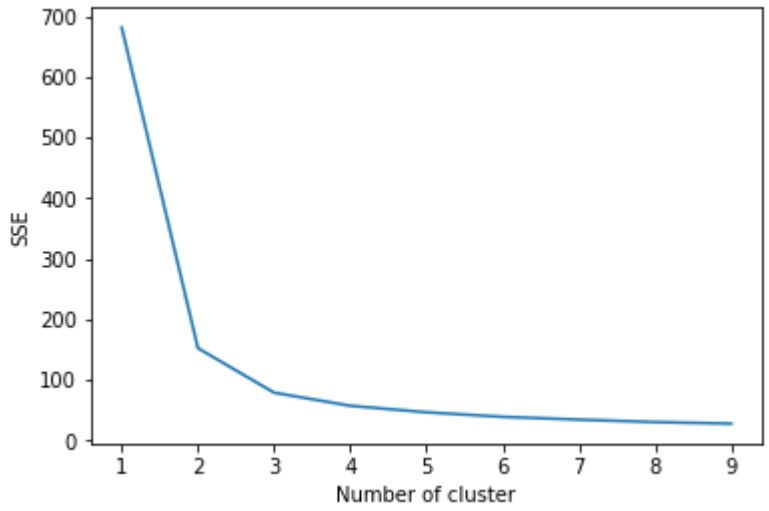
In [6]:

```
#Importing the silhoutte score function
from sklearn.metrics import silhouette_score
from sklearn.metrics import accuracy_score
from sklearn.cluster import KMeans

#Silhoutte scores for different clusters and visualisation
sil = {}
for n_cluster in range(2, 11):
    kmeans = KMeans(n_clusters=n_cluster, max_iter=1000).fit(X)
    label = kmeans.labels_
    sil[n_cluster] = silhouette_score(X, label, metric='euclidean')
    print("For n_clusters={}, The Silhouette Coefficient is {}.".format(
            n_cluster, sil[n_cluster]))
plt.figure()
plt.plot(list(sil.keys()), list(sil.values()))
plt.xlabel("Number of cluster")
plt.ylabel("Silhoutte score")
plt.show()

print()

#Inertia/SSE for different clusters and visualisation
sse = {}
for k in range(1, 10):
    kmeans = KMeans(n_clusters=k, max_iter=1000).fit(X)
    sse[k] = kmeans.inertia_ # Inertia: Sum of distances of samples to their closest cl
uster center
    print("For n_clusters={}, the sum of squared errors is {}.".format(
            k, sse[k]))
plt.figure()
plt.plot(list(sse.keys()), list(sse.values()))
plt.xlabel("Number of cluster")
plt.ylabel("SSE")
plt.show()
```

```
For n_clusters=2, The Silhouette Coefficient is 0.681046169211746.
For n_clusters=3, The Silhouette Coefficient is 0.5528190123564091.
For n_clusters=4, The Silhouette Coefficient is 0.4980505049972866.
For n_clusters=5, The Silhouette Coefficient is 0.4887488870931048.
For n_clusters=6, The Silhouette Coefficient is 0.36295529183027064.
For n_clusters=7, The Silhouette Coefficient is 0.348950840496828.
For n_clusters=8, The Silhouette Coefficient is 0.3390450925992608.
For n_clusters=9, The Silhouette Coefficient is 0.32352028503142977.
For n_clusters=10, The Silhouette Coefficient is 0.3389256908245164.
```




```
For n_clusters=1, the sum of squared errors is 681.3706.
For n_clusters=2, the sum of squared errors is 152.34795176035792.
For n_clusters=3, the sum of squared errors is 78.85144142614601.
For n_clusters=4, the sum of squared errors is 57.25600931571815.
For n_clusters=5, the sum of squared errors is 46.472230158730156.
For n_clusters=6, the sum of squared errors is 39.03998724608725.
For n_clusters=7, the sum of squared errors is 34.40900974025974.
For n_clusters=8, the sum of squared errors is 30.33777154862682.
For n_clusters=9, the sum of squared errors is 27.78726218956482.
```



Silhoutte score showed that the optimal number of clusters was 2.
Inertia/SSE showed that the "elbow" where the error reduction occurs is around 2 to 3 clusters.

Spectral Clustering and Agglomerative Clustering: Determining the Optimal Number of Clusters

For these two algorithms, while we could use the Silhoutte score, we have already talked about the inherent limitations above. It would be more valuable to try out another metric to assess the effectiveness of the algorithm for different numbers of clusters.

I had initially wanted to try out inertia, but discovered that not all algorithms have that attribute, and also based on the scikit learn documentation inertia may not be a good measurement of effectiveness for some algorithms. Looking further within the documentation, I decided to try out the **Calinski Harabaz score**.

The Calinski-Harabaz score is given as the ratio of the between-clusters dispersion mean and the within-cluster dispersion:

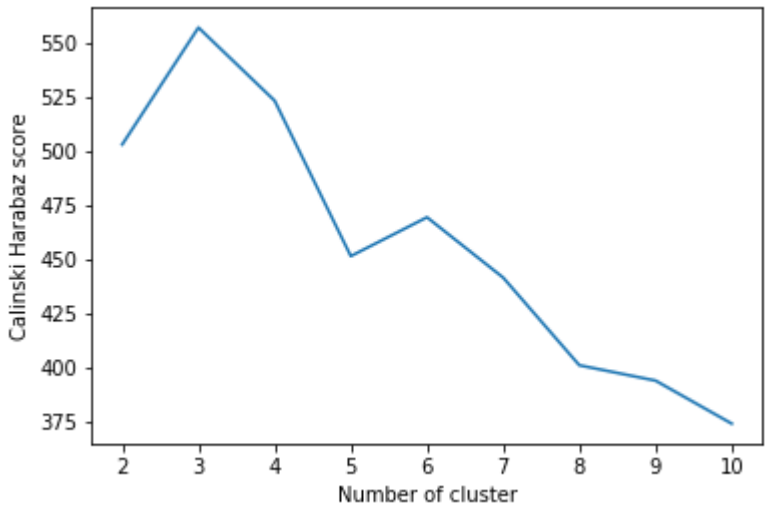
$$s(k) = \frac{\mathrm{Tr}(B_k)}{\mathrm{Tr}(W_k)} \times \frac{N - k}{k - 1}$$

where BK is the between group dispersion matrix, WK is the within-cluster dispersion matrix, and N is the number of data points. In essence, to give a quick intuition, this metric functions somewhat like ANOVA, where there is a within-group and between-group mean, and when these means diverge sufficiently the null hypothesis is rejected.

In [7]:

```
def warn(*args, **kwargs):
    pass
import warnings
warnings.warn = warn


from sklearn.cluster import SpectralClustering
from sklearn.metrics import calinski_harabaz_score

#Silhoutte scores for different clusters and visualisation
chs = {}
for n_cluster in range(2, 11):
    spectral = SpectralClustering(n_clusters=n_cluster, affinity='nearest_neighbors').f
it(X)
    label = spectral.labels_
    chs[n_cluster] = calinski_harabaz_score(X, label)
    print("For n_clusters={}, The Calinski Harabaz score is {}.".format(
            n_cluster, chs[n_cluster]))
plt.figure()
plt.plot(list(chs.keys()), list(chs.values()))
plt.xlabel("Number of cluster")
plt.ylabel("Calinski Harabaz score")
plt.show()
```

```
For n_clusters=2, The Calinski Harabaz score is 502.82156350235897.
For n_clusters=3, The Calinski Harabaz score is 556.8795419179528.
For n_clusters=4, The Calinski Harabaz score is 523.0982068284292.
For n_clusters=5, The Calinski Harabaz score is 451.2219241858261.
For n_clusters=6, The Calinski Harabaz score is 469.17746699794236.
For n_clusters=7, The Calinski Harabaz score is 441.2962024124759.
For n_clusters=8, The Calinski Harabaz score is 400.7565814970956.
For n_clusters=9, The Calinski Harabaz score is 393.66440161422.
For n_clusters=10, The Calinski Harabaz score is 373.777967917726.
```

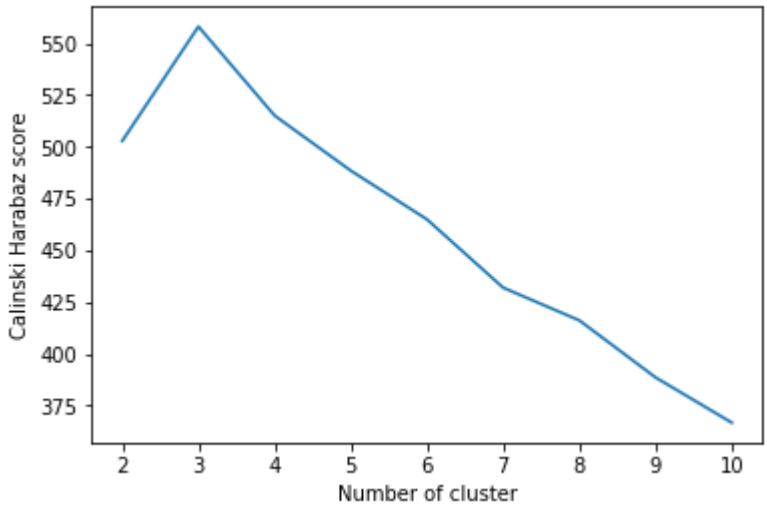


The highest Calinski Harabaz score occured when k = 3.

In [8]:

```
from sklearn.cluster import AgglomerativeClustering

chsam = {}
for n_cluster in range(2, 11):
    agglom = AgglomerativeClustering(n_clusters=n_cluster).fit(X)
    label = agglom.labels_
    chsam[n_cluster] = calinski_harabaz_score(X, label)
    print("For n_clusters={}, The Calinski Harabaz score is {}.".format(
            n_cluster, chsam[n_cluster]))
plt.figure()
plt.plot(list(chsam.keys()), list(chsam.values()))
plt.xlabel("Number of cluster")
plt.ylabel("Calinski Harabaz score")
plt.show()
```

```
For n_clusters=2, The Calinski Harabaz score is 502.82156350235897.
For n_clusters=3, The Calinski Harabaz score is 558.0580408128307.
For n_clusters=4, The Calinski Harabaz score is 515.0789062430442.
For n_clusters=5, The Calinski Harabaz score is 488.4849040365162.
For n_clusters=6, The Calinski Harabaz score is 464.94939152139295.
For n_clusters=7, The Calinski Harabaz score is 431.9818198792292.
For n_clusters=8, The Calinski Harabaz score is 416.18448738825094.
For n_clusters=9, The Calinski Harabaz score is 388.6499175669273.
For n_clusters=10, The Calinski Harabaz score is 366.8296134141234.
```



The highest Calinski Harabaz score occured when k = 3.

DBSCAN

For DBSCAN, there was no need to specify the number of clusters. It would automatically make its own label predictions, and from there we can then find out what was the optimal number of clusters that the algorithm had "sussed out"

In [13]:

```
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler

X1 = StandardScaler().fit_transform(X)
db = DBSCAN()
db.fit(X1)
label = db.labels_
n_clusters_ = len(set(label)) - (1 if -1 in label else 0)
print('Estimated number of clusters: %d' % n_clusters_)
```

```
Estimated number of clusters: 2
```

Checking back against the "answers"

Now that we have all the optimal number of clusters for each clustering algorithm, let's reinstantiate them, and make predictions. Also, since we have the ground truth (the "answers" in the form of actual labels), we can measure the effectivness of each algorithm.

The measure that I will use is the Adjusted Rand Index (ARI). I chose this measure because it **ignores permutations**, hence the name of the labels (1 vs 2 or 0 vs 1) does not matter. Also, it is **symmetric**, as swapping the argument does not change the score. This makes it useful since we are only interested in the similarity of the assignments.

In [14]:

```
#Reinstantiating and fitting algorithms with their optimal k

#K-Means Clustering
km = KMeans(n_clusters=3).fit(X)
kmlabel = km.labels_

#Spectral Clustering
spectral = SpectralClustering(n_clusters=3, affinity='nearest_neighbors').fit(X)
spectrallabel = spectral.labels_

#Agglomerative Clustering
agglom = AgglomerativeClustering(n_clusters=3).fit(X)
agglomlabel = agglom.labels_

#DBSCAN
X1 = StandardScaler().fit_transform(X)
db = DBSCAN()
db.fit(X1)
dblabel = db.labels_
```

In [22]:

```
from sklearn import metrics
ARIkm = metrics.adjusted_rand_score(iris.target, kmlabel)
ARIspectral = metrics.adjusted_rand_score(iris.target, spectrallabel)
ARIagglom = metrics.adjusted_rand_score(iris.target, agglomlabel)
ARIdb = metrics.adjusted_rand_score(iris.target, dblabel)

print("The ARI for K-Means Clustering is {}.".format(ARIkm))
print("The ARI for Spectral Clustering is {}.".format(ARIspectral))
print("The ARI for Agglomerative Clustering is {}.".format(ARIagglom))
print("The ARI for DBSCAN is {}.".format(ARIdb))
```

```
The ARI for K-Means Clustering is 0.7302382722834697.
The ARI for Spectral Clustering is 0.7591987071071522.
The ARI for Agglomerative Clustering is 0.7311985567707745.
The ARI for DBSCAN is 0.44209866858859237.
```

The highest ARI was from Spectral Clustering, with k=3 at an ARI of 0.75. K-Means and Agglomerative come close behind, at 0.73.

DBSCAN's ARI is the lowest at 0.44. This is because the natural number of clusters (2 clusters) sussed out by the algorithm was already different from ground truth (3 clusters), hence confirming what we know from background knowledge that the true value of k=3.

Summary - the value of the process

Going through this process of trying out various algorithms and evaluation metrics is useful because:

- It helps us *narrow down* on the useful range of what the k-value could be. In a real-life business case, where we do not know the optimal k, what we have established is a better sense of possible values, rather than directly taking the Silhoutte score.
- Going a step further, the process may help us to *triangulate* the best value of k. Even in this example alone, after we have narrowed down k to two possible values of 2 and 3, intuition lends itself more strongly towards 3, since out of 5 evaluation metrics, three of them point to k=3 (inertia for K-means and Calinski-Harabaz for Spectral and Agglomerative), and two of them point to k=2 (Silhoutte for K-means and DBSCAN). There could even be a formalised voting system coded into a script if we want to make this automatic.
- In other datasets, the structure of the data may not best lend itself to a Euclidean mode of clustering, or where the number of clusters is naturally very large and K-means clustering may not be suitable. Some clusters could be naturally occurring in *density* too. Going through this process may help us detect that -- if the predictions from different algorithms turn out to be *extremely* different, then we know that one of these mechanisms would have kicked into effect, and we need to be careful when selecting algorithms.

What we need to remember:

- In finding the best algorithm and k-value, we cannot totally remove the element of trial and error -- since we don't have the answers for unsupervised learning. But having a good process of trying different algorithms and metrics, as well as having a good sense of the data, will help us cut down the range of that trial and error and let us arrive at the best solution more quickly
- In StackOverflow, someone said to always seek the ground-truth manually if the business case is too important for failure (one example I can think of clustering of medical ailments for treatment). These unsupervised algorithms and evaluation metrics should always be used as a guide, and the business user needs to know the room to maneuver in the absence of ground truth.

References

1. Scikit Learn Documentation, 2.3: Clustering. https://scikit-learn.org/stable/modules/clustering.html (https://scikit-learn.org/stable/modules/clustering.html)
2. Photios A. Stavrou, Answer to Researchgate question "What is the difference between convex and non-convex optimization problems?"https://www.researchgate.net/post/What_is_the_difference_between_convex_and_non-convex_optimization_problems (https://www.researchgate.net/post/What_is_the_difference_between_convex_and_non-convex_optimization_problems)
3. Wikipedia, Cluster Analysis: https://en.wikipedia.org/wiki/Cluster_analysis (https://en.wikipedia.org/wiki/Cluster_analysis)
4. Jure Leskovec, Anand Rajaraman and Jeffrey Ullman, Stanford University. Mining of Massive Datasets Chpt 7: Clustering: http://infolab.stanford.edu/~ullman/mmds/ch7.pdf (http://infolab.stanford.edu/~ullman/mmds/ch7.pdf)
5. Michael J Garbade, Understanding K-means Clustering in Machine Learning. https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1 (https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1)
6. Kelvin Salton do Prado, How DBSCAN works and why should we use it? https://towardsdatascience.com/how-dbscan-works-and-why-should-i-use-it-443b4a191c80 (https://towardsdatascience.com/how-dbscan-works-and-why-should-i-use-it-443b4a191c80)
7. Amine Aoullay, Spectral Clustering for Beginners. https://towardsdatascience.com/spectral-clustering-for-beginners-d08b7d25b4d8 (https://towardsdatascience.com/spectral-clustering-for-beginners-d08b7d25b4d8)
8. Chaitanya Reddy, Understanding the Concept of Hierarchical Clustering Technique. https://towardsdatascience.com/understanding-the-concept-of-hierarchical-clustering-technique-c6e8243758ec (https://towardsdatascience.com/understanding-the-concept-of-hierarchical-clustering-technique-c6e8243758ec)