Labwork 2 – Report

Member:
Phan Manh Tung – USTHBI8-160
Vu Tuan Phong – USTHBI8-139

T2 - Clustering

Data: For this labwork, we choose 2 dataset from UCI Machine Learning Repository namely Breast Cancer Wisconsin dataset and Iris dataset.

Tool: Python is our chosen programming language and we opt for google colab (a free python interactive environment offered by Google) for strong computing power and convenient coding.

I) K-means

1.1 Breast Cancer Wisconsin dataset

First, we read the breast cancer data into a dataframe using pandas (a package for exploring data in python).

```
# Breast cancer dataset

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

column = ["Sample code number", "Clump Thickness", "Uniformity of Cell Size", "Uniformity of Cell Shape", "Marginal Adhesion", "Single Epithelial Cell Size",
          "Bare Nuclei", "Bland Chromatin", "Normal Nucleoli", "Mitoses", "Class"]
df = pd.read_csv("/content/gdrive/My Drive/breast-cancer-wisconsin.data", sep=',', header=None, skipinitialspace=True)
df.columns = column

# Save the class information
myclass = df["Class"]
# class: 2 for benign, 4 for malignant

# Set the ID number as index
df.set_index("Sample code number", inplace=True)

# Get rid of class column for statistical analysis and PCA, append it latter
df.drop(['Class'],axis=1, inplace=True)

df.head()
```

Sample code number	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses
1000025	5	1	1	1	2	1	3	1	1
1002945	5	4	4	5	7	10	3	2	1
1015425	3	1	1	1	2	2	3	1	1
1016277	6	8	8	1	3	4	3	7	1
1017023	4	1	1	3	2	1	3	1	1

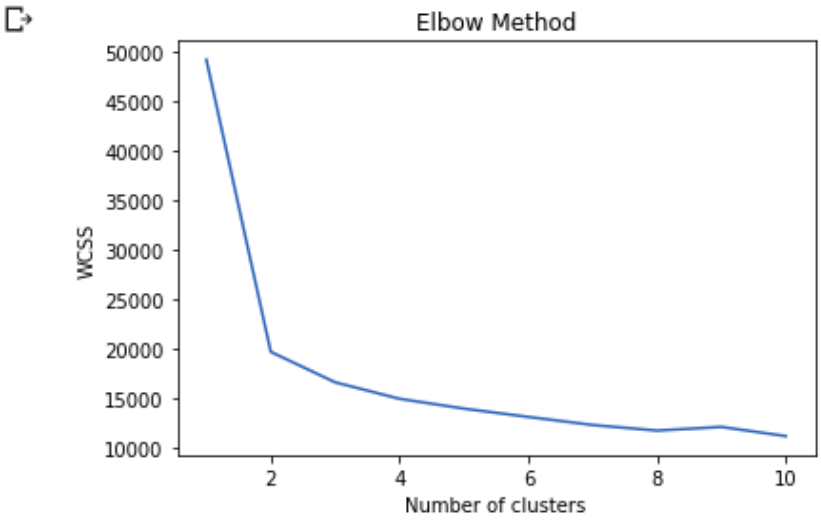
This dataset consists of 8 attributes (integers from 1 to 10) and ID number. For clustering-unsupervised problem, we first remove the class/labels of the data points and save it into “myclass” for later evaluation. ID number is set to be the index column. (Class: 2 for benign, 4 for malignant)

Because this is a 9-dimensional dataset, it is difficult to visualize, thus hard to estimate the optimum number of cluster.

A common method to solve this problem is “Elbow Method” which simply tries out different number of cluster, then quantifies the “badness” of each option using Within Cluster Sum of Square (WCSS) – basically the sum of total variation squared.

We visualize the result via an Elbow plot. It is clear that after the point of 2 clusters, WCSS begins to go down slowly, that turning point is the critical value we attempt to find.

```
[3] from sklearn.cluster import KMeans

    X = df.replace("?", 1).values

    wcss = []
    for i in range(1, 11):
        kmeans = KMeans(n_clusters=i, init='random', max_iter=300, n_init=10)
        kmeans.fit(X)
        wcss.append(kmeans.inertia_)
    plt.plot(range(1, 11), wcss)
    plt.title('Elbow Method')
    plt.xlabel('Number of clusters')
    plt.ylabel('WCSS')
    plt.show()
```



This plot also answers the third question: with different values of k, sum of total variation within clusters would decline (eventually to 0 as k=n).

Then, we implement the k-means algorithm with 2 clusters and evaluate that with 2 metrics in scikit-learn: accuracy and rand index.

Note: Our initialization:
+ n_clusters=2: number of clusters is 2
+ init='random': choose k observations at random from data for the initial centroids
+ max_iter=300: maximum number of iterations of the k-mean algorithm for a single run

```
kmeans = KMeans(n_clusters=2, init='random', max_iter=300, n_init=10)
# Explain the initialization:
# n_clusters=2: number of clusters is 2
# init='random': choose k observations at random from data for the initial centroids
# max_iter=300: Maximum number of iterations of the k-means algorithm for a single run

pred_y = kmeans.fit_predict(X)

# Turn 0, 1 values into 2, 4 values
my_pred_y = []
for i in pred_y:
  if(i==0):
    my_pred_y.append(2)
  else:
    my_pred_y.append(4)

from sklearn import metrics
# Accuracy reflects the proportion of objects that were correctly assigned
print("Accuracy: ", metrics.accuracy_score(myclass.values, my_pred_y))

# Rand index computes how similar the obtained clusters are to the benchmark classifications
print("Rand index: ", metrics.adjusted_rand_score(myclass.values, my_pred_y))
```

```
Accuracy:  0.9585121602288984
Rand index:  0.839053994351602
```

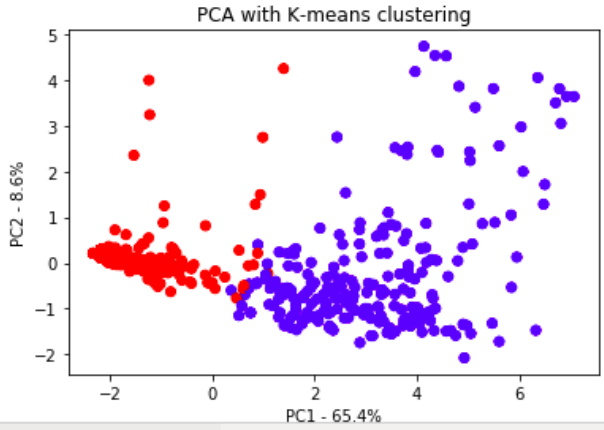
Accuracy (0.9585) and Rand index (0.839) values are close to 1, which means our model is quite accurate in assigning labels for the data points and 2 obtained clusters are similar to the benchmark classification.

Finally, we apply PCA to the dataset and using the predicted classification of k-means values for 2D visualization.At the same time, we plot the data with the benchmark classification for a clear comparison.

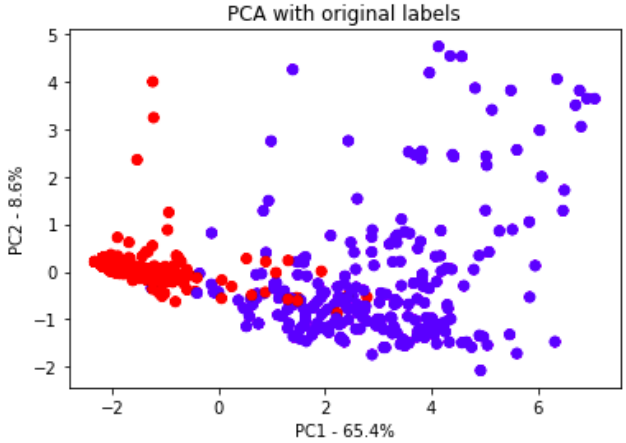
We can see that 2 pictures are roughly equivalent, thus our k-means model works out quite well.

```
from sklearn.decomposition import PCA
from sklearn import preprocessing

# Fill unknown value with 1
df.replace("?", 1, inplace=True)

scaled_data = preprocessing.scale(df)

pca = PCA(n_components=2) # create a PCA object
pca.fit(scaled_data) # do the math
pca_data = pca.transform(scaled_data) # get PCA coordinates for scaled_data

per_var = np.round(pca.explained_variance_ratio_* 100, decimals=1)
print(per_var)
labels = ['PC' + str(x) for x in range(1, len(per_var)+1)]

pca_df = pd.DataFrame(pca_data, columns=labels)

# Label class using my_pred_y instead of myclass
pca_df['Class'] = my_pred_y

colors = {2:'red', 4:'blue'}

plt.scatter(pca_df.PC1, pca_df.PC2, c=pca_df['Class'].apply(lambda x: colors[x]))
plt.title('PCA with K-means clustering')
plt.xlabel('PC1 - {0}%'.format(per_var[0]))
plt.ylabel('PC2 - {0}%'.format(per_var[1]))
plt.show()
```

```
[65.4  8.6]
```




```
pca_df['Class'] = myclass

colors = {2:'red', 4:'blue'}

plt.scatter(pca_df.PC1, pca_df.PC2, c=pca_df['Class'].apply(lambda x: colors[x]))
plt.title('PCA with original labels')
plt.xlabel('PC1 - {0}%'.format(per_var[0]))
plt.ylabel('PC2 - {0}%'.format(per_var[1]))
plt.show()
```



1.2 Iris dataset

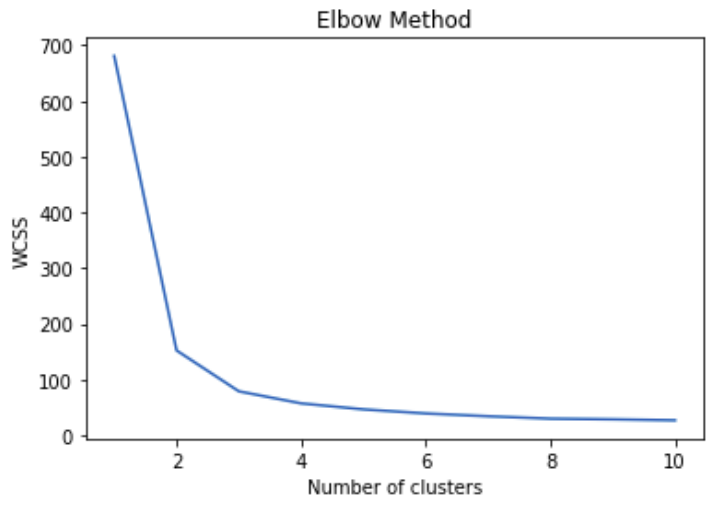
```
[35] column = ["sepal length", "sepal width", "petal length", "petal width", "class"]
     df = pd.read_csv("/content/gdrive/My Drive/iris.data", sep=',', header=None)
     df.columns = column
     myclass = df["class"]
     df.head(5)
```

	sepal length	sepal width	petal length	petal width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Again, we read the dataset using pandas, there are 4 attributes with 3 categories in Iris dataset.

Then, we illustrate the Elbow plot to find the optimum value for k.

```
[36] X = df.drop(["class"],axis=1).values

     wcss = []
     for i in range(1, 11):
         kmeans = KMeans(n_clusters=i, init='random', max_iter=300, n_init=10)
         kmeans.fit(X)
         wcss.append(kmeans.inertia_)
     plt.plot(range(1, 11), wcss)
     plt.title('Elbow Method')
     plt.xlabel('Number of clusters')
     plt.ylabel('WCSS')
     plt.show()
```

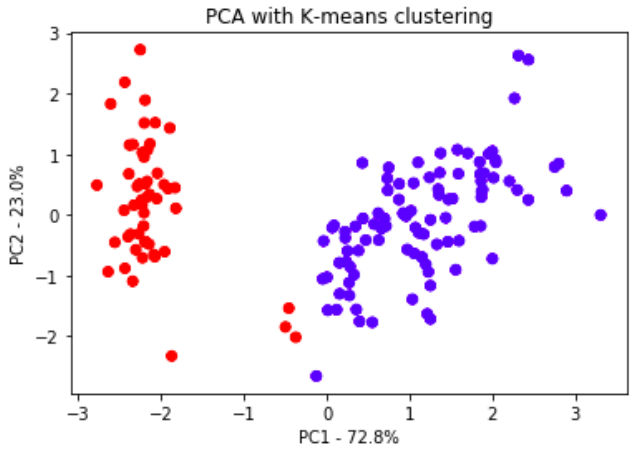


It is evident that both k=2 and k=3 are good choices for number of cluster for this dataset. We choose k=2 because it is a more obvious turning point.

We apply the k-means algorithm with the same initialization, then apply PCA to visualize the results in 2D.

```
from sklearn.decomposition import PCA
from sklearn import preprocessing

df.drop(['class'], axis=1, inplace=True)
kmeans = KMeans(n_clusters=2, init='random', max_iter=300, n_init=10)
pred_y = kmeans.fit_predict(X)

scaled_data = preprocessing.scale(df)

pca = PCA(n_components=2) # create a PCA object
pca.fit(scaled_data) # do the math
pca_data = pca.transform(scaled_data) # get PCA coordinates for scaled_data

per_var = np.round(pca.explained_variance_ratio_* 100, decimals=1)
print(per_var)
labels = ['PC' + str(x) for x in range(1, len(per_var)+1)]

pca_df = pd.DataFrame(pca_data, columns=labels)

# Label class using my_pred_y instead of myclass
pca_df['Class'] = pred_y

colors = {0:'red', 1:'blue'}

plt.scatter(pca_df.PC1, pca_df.PC2, c=pca_df['Class'].apply(lambda x: colors[x]))
plt.title('PCA with K-means clustering')
plt.xlabel('PC1 - {0}%'.format(per_var[0]))
plt.ylabel('PC2 - {0}%'.format(per_var[1]))
plt.show()
```

```
[72.8 23. ]
```



Our k-means has done a great job in separating the data into 2 distinct clusters.

Finally, because we know for the fact that there are 3 categories in the original class, so to evaluate the model, we need to use some metrics which require no ground truth.

```
print(metrics.davies_bouldin_score(X, pred_y))

print(metrics.calinski_harabasz_score(X, pred_y))
```

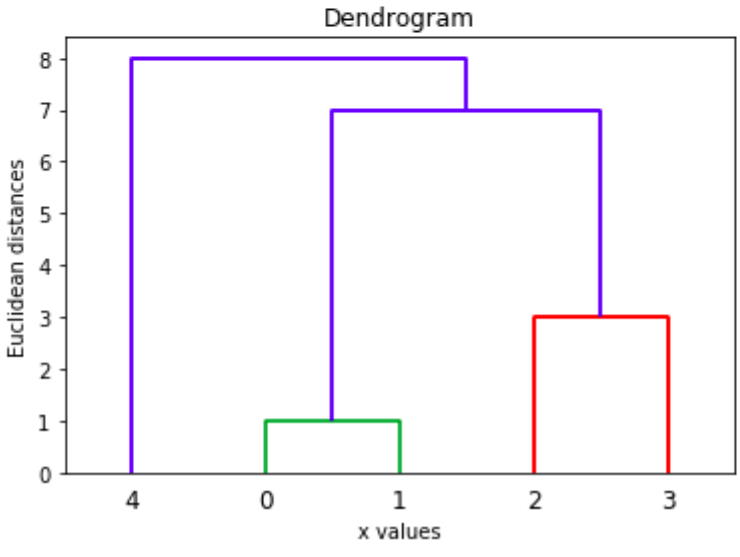
```
0.4048341363918299
513.3038433517568
```

The Davies Bouldin score (0.4048) estimate the similarity between clusters. So, the values close to 0 indicate distinctive clusters, thus a good-quality model. 0.4 is a decent value for a good one.

The Calinski Harabasz score is the ratio of the sum of between-clusters dispersion and of inter-cluster dispersion for all clusters (where dispersion is defined as the sum of distances squared). Therefore, the higher the value, the more dispersed the two clusters. 513.3038 is an adequate indication of the good quality for our model.

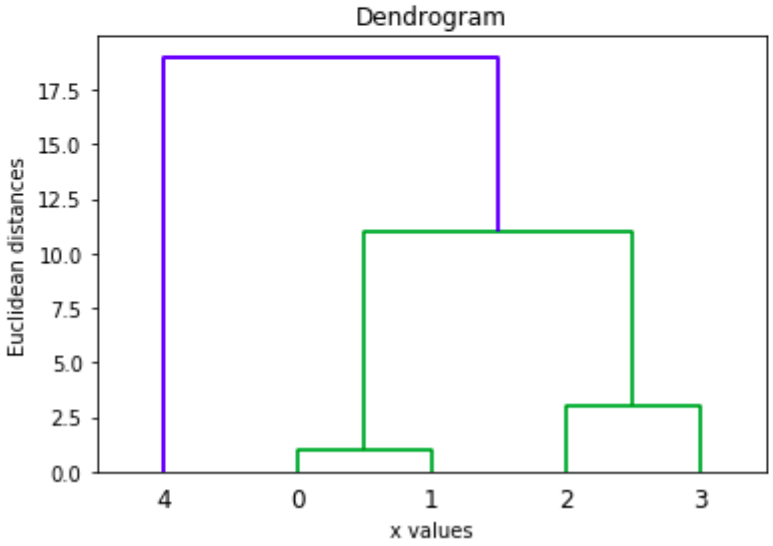
II) Agglomerative Hierarchical Clustering (AHC)

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.cluster.hierarchy as sch

X = np.array([[1, 0], [2, 0], [9, 0], [12, 0], [20, 0]])

dendrogram = sch.dendrogram(sch.linkage(X, method  = "single"))

plt.title('Dendrogram')
plt.xlabel('x values')
plt.ylabel('Euclidean distances')
plt.show()
```



With the data X = {1, 2, 9, 12, 20}, we apply the AHC clustering using Single Linkage / Complete Linkage, then plot it via dendrogram using package scipy.

```
[5]  dendrogram = sch.dendrogram(sch.linkage(X, method  = "complete"))

     plt.title('Dendrogram')
     plt.xlabel('x values')
     plt.ylabel('Euclidean distances')
     plt.show()
```



Single Linkage divides the data into 3 distinct clusters (1-2-2) while Complete method returns 2 clusters (1-4). So, with this data, Single method is a better solution.

2.1 Breast Cancer Dataset

For AHC, we continue with breast cancer dataset. First, we use pandas to read the data. Remember, this data has 8 attribute (ranging from 1 to 10) and 2 classes : 2 for benign, 4 for malignant.

```
[20] # Breast cancer dataset
     import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt

     column = ["Sample code number", "Clump Thickness", "Uniformity of Cell Size", "Uniformity of Cell Shape", "Marginal Adhesion", "Single Epithelial Cell Size",
               "Bare Nuclei", "Bland Chromatin", "Normal Nucleoli", "Mitoses", "Class"]
     df = pd.read_csv("/content/gdrive/My Drive/breast-cancer-wisconsin.data", sep=',', header=None, skipinitialspace=True)
     df.columns = column

     # Save the class information
     myclass = df["Class"]
     # class: 2 for benign, 4 for malignant

     # Set the ID number as index
     df.set_index("Sample code number", inplace=True)
     df.head()
```

Sample code number	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Class
1000025	5	1	1	1	2	1	3	1	1	2
1002945	5	4	4	5	7	10	3	2	1	2
1015425	3	1	1	1	2	2	3	1	1	2
1016277	6	8	8	1	3	4	3	7	1	2
1017023	4	1	1	3	2	1	3	1	1	2

We apply the AHC with Complete Linkage and number of clusters is 2, then evaluate the model with 2 metrics Accuracy and Rand Index.

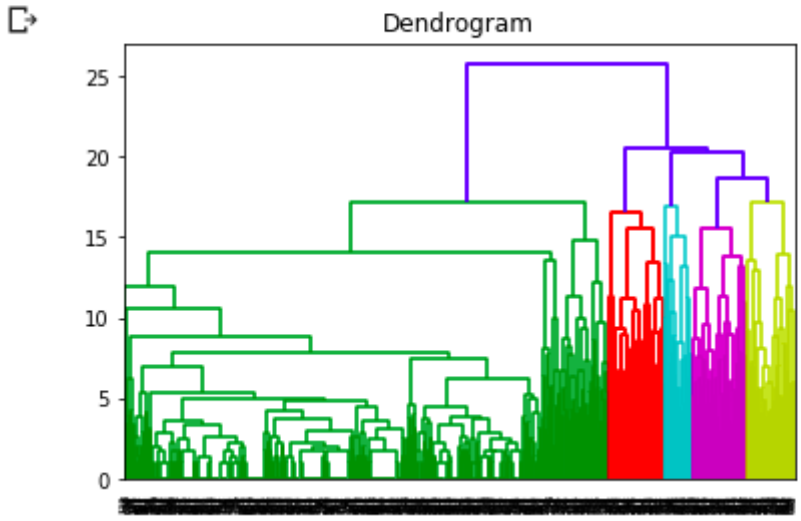
```
[22] from sklearn.cluster import AgglomerativeClustering
     hc = AgglomerativeClustering(n_clusters = 2, affinity = 'euclidean', linkage ='complete')

     pred_y = []
     for i in y_hc:
       if(i==0):
         pred_y.append(4)
       else:
         pred_y.append(2)

     from sklearn import metrics
     # Accuracy reflects the proportion of objects that were correctly assigned
     print("Accuracy: ", metrics.accuracy_score(myclass.values, pred_y))

     # Rand index computes how similar the obtained clusters are to the benchmark classifications
     print("Rand index: ", metrics.adjusted_rand_score(myclass.values, pred_y))
```

```
Accuracy:  0.9155937052932761
Rand index:  0.6845263606757068
```

It is clearly seen that accuracy score and rand index score is close to 1, which indicates that our model is accurate with ground truth.

Afterwards, we draw the dendrogram using scipy. Obviously, the data is separated into 2 major clusters.

```
[21] import scipy.cluster.hierarchy as sch
     import matplotlib.pyplot as plt
     X = df.drop(['Class'],axis=1).replace("?", 1).values
     dendrogram = sch.dendrogram(sch.linkage(X, method  = "complete"))
     plt.title("Dendrogram")
     plt.show()
```



2.2 Iris dataset

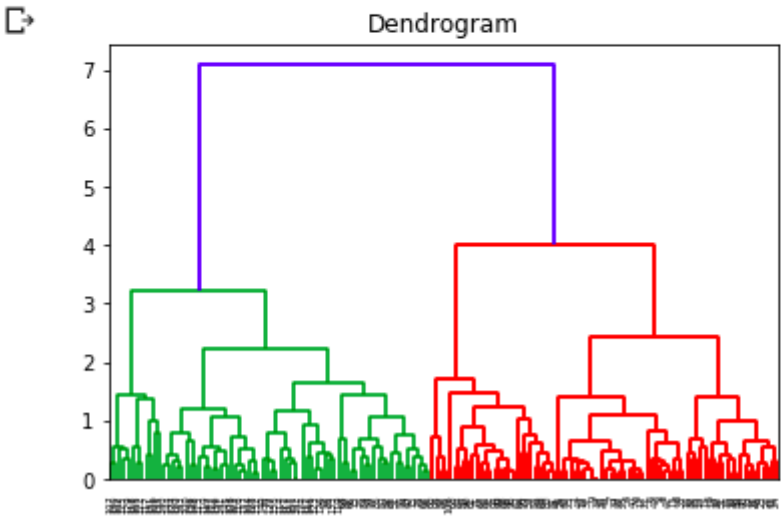
We again explore the traditional Iris dataset with our new method (AHC). This data has 4 attributes and 3 categories for all the samples.

```
import pandas as pd
column = ["sepal length", "sepal width", "petal length", "petal width", "class"]
df = pd.read_csv("/content/gdrive/My Drive/iris.data", sep=',', header=None)
df.columns = column
myclass = df["class"]
df.drop(columns=['class'], inplace=True)
df.head()
```

	sepal length	sepal width	petal length	petal width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

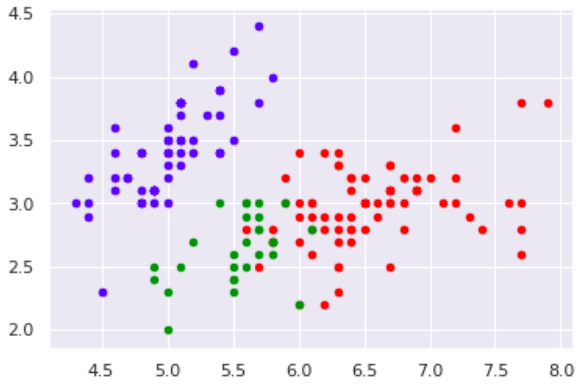
First, we draw the dendrogram to see how many clusters to choose. Linkage method is complete. It is evident that the number of clusters should is 2 in this case. However, for study purpose, we decided to choose 3 clusters instead of 2.

```
[25] X = df.values
     dendrogram = sch.dendrogram(sch.linkage(X, method  = "complete"))

     plt.title("Dendrogram")
     plt.show()
```



```
[27] hc = AgglomerativeClustering(n_clusters = 3, affinity = 'euclidean', linkage ='complete')

    # y_hc: Which cluster the points belong to?
    y_hc=hc.fit_predict(X)
    print(y_hc)

    plt.scatter(X[y_hc==0, 0], X[y_hc==0, 1], s=20, c='red', label ='Cluster 1')
    plt.scatter(X[y_hc==1, 0], X[y_hc==1, 1], s=20, c='blue', label ='Cluster 2')
    plt.scatter(X[y_hc==2, 0], X[y_hc==2, 1], s=20, c='green', label ='Cluster 3')
```

```
[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 0 0 0 2 0 2 0 2 0 2 2 2 2 0 2 0 2 2 0 2 0 2 0 0
 0 0 0 0 0 2 2 2 2 0 2 0 0 0 2 2 2 0 2 2 2 2 2 0 2 2 0 0 0 0 0 2 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0]
<matplotlib.collections.PathCollection at 0x7fccc5dc8630>
```



We apply AHC with n_clusters = 3 and linkage = ”complete”, then plot the results with 2 attributes (sepal length and sepal width). We can see our 3 clusters look distinctive.

Finally, we evaluate the model using 2 familiar metrics: accuracy and rand index.

```
myclass.replace("Iris-setosa", 1, inplace=True)
myclass.replace("Iris-versicolor", 2, inplace=True)
myclass.replace("Iris-virginica", 0, inplace=True)

from sklearn import metrics
# Accuracy reflects the proportion of objects that were correctly assigned
print("Accuracy: ", metrics.accuracy_score(myclass.values, y_hc))

# Rand index computes how similar the obtained clusters are to the benchmark classifications
print("Rand index: ", metrics.adjusted_rand_score(myclass.values, y_hc))
```

```
Accuracy:  0.84
Rand index:  0.64225125183629
```

Our model reaches 84% of accuracy, it is a really good score. And also 0.64 is a decent score for rand index.

Advantages:

- AHC does not require any input parameters in advance.

- Simple visualization with dendrogram which is a hierarchy structure, thus more information and more easy comparison in similarity between objects.

- The mathematics behind the algorithm is not complicated, therefore easy to implement.

Drawbacks:

- High complexity O(n2) or O(n3) → not suitable for large dataset.

- Sensible to outliers.

- Initial seed and the order of data have strong impact on the final results.