Clustering of quantitative data sets

Non-exhaustive descriptions and implementation turorials of different clustering algorithms. We will see hierarchical clustering, k-means algorithm and other. Pre-requisites: R programming and PCA.

Long story short

Unsupervised classification (or clustering) is a type of machine learning technique that involves clustering data points into different groups based on their similarity. Unlike supervised classification, unsupervised classification does not require labeled data or predefined classes, but rather identifies patterns and structures within the data itself.

The main conceptual idea behind unsupervised classification is to identify similarities and differences among the data points without prior knowledge of the underlying structure or classification. The process involves finding patterns in the data that can be used to group similar objects together into clusters. The goal is to find a partitioning of the data into groups, such that objects within a cluster are more similar to each other than they are to objects in other clusters.

The unsupervised classification algorithm typically uses distance metrics to measure the similarity between data points and clustering methods to group them together. Common distance metrics used in unsupervised classification include Euclidean distance, Manhattan distance, and cosine distance. Clustering methods may include hierarchical clustering, k-means clustering, and density-based clustering.

One key advantage of unsupervised classification is that it can be used to identify previously unknown patterns and structures within the data, which can be useful for exploratory data analysis, data mining, and anomaly detection. However, the output of unsupervised classification can be subjective, as the number of clusters and their boundaries may be determined by the algorithm or by the analyst's interpretation of the results. Additionally, the quality of the clustering results depends on the choice of distance metric, clustering method, and the quality of the data itself.

Is clustering the solution to your problem?

The first thing you want to ask yourself is that whether or not this method is the one you need. If your objective is to find the best predictors of a given metric, you're in the wrong place. To the other hand, if you don't have any particular objective with your data and just want to explore them, clustering might be a good idea!

It's important to keep in mind that no matter the performance and how great the method we use is, if the observations and the variables in the dataset don't have a group structure and are not correlated, we will not be able to obtain much information. But it's also important to know that the fact that some variables are not correlated is as much as important as if they were. It is therefore important not to get stuck in a method when you realize that it is not adapted and/or does not give conclusive results.

Graph based clustering

In this short section, we will do a preliminary analysis, using only graphs, in order to check if we are able to find groups in the data set. It's actually not a bad idea to start by doing this since data visualization is a very great tool (even though it requires to be careful and needs you to really understand what you're doing).

This method is quite subjective (but clustering method also are sometimes) but can gives us lots of relevant information.

If you don't know what data visualization is, check ***this example I saw on linkedin*** and ***this excellent ted talk by Hans Rosling***.

Define a data set

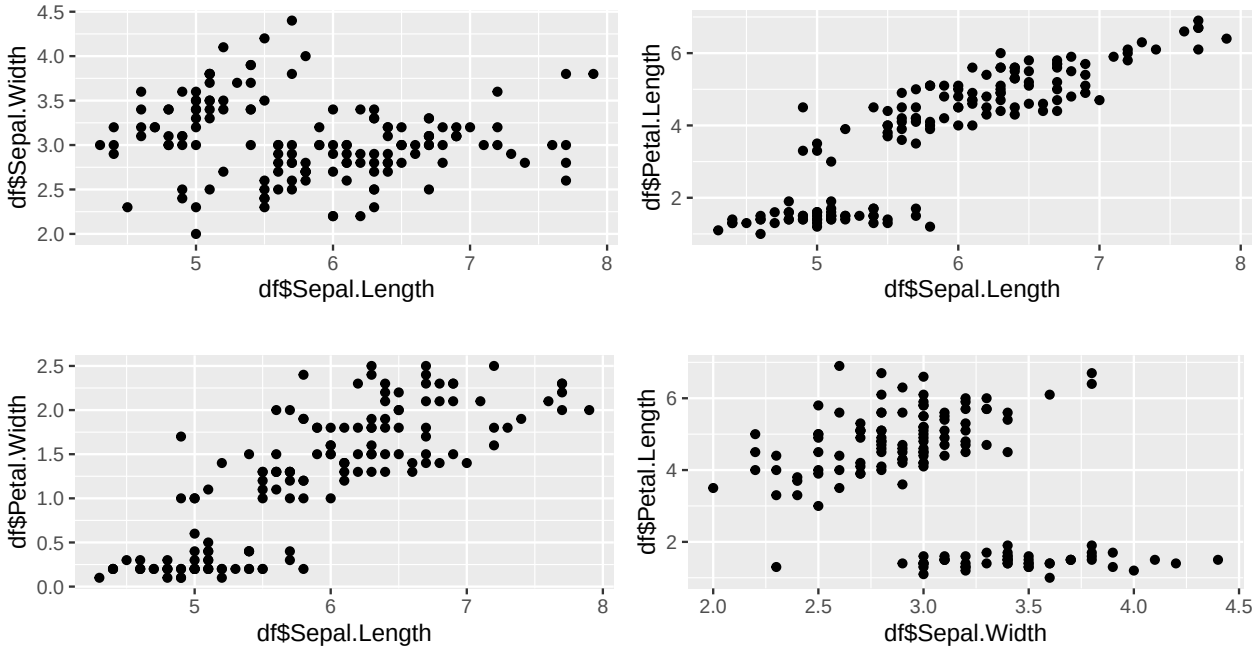
```
data("iris") #load the built-in iris dataset
df = iris[, 1:4] #remove the species feature
head(df) #print the first rows
```

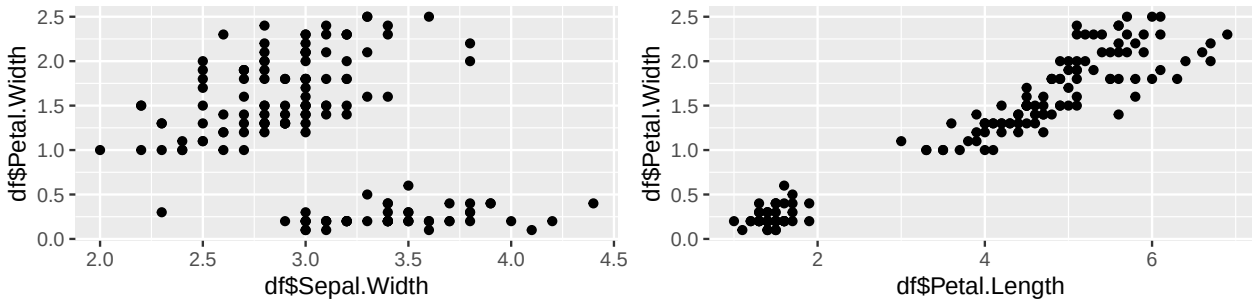
```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1          5.1         3.5          1.4         0.2
## 2          4.9         3.0          1.4         0.2
## 3          4.7         3.2          1.3         0.2
## 4          4.6         3.1          1.5         0.2
## 5          5.0         3.6          1.4         0.2
## 6          5.4         3.9          1.7         0.4
```

Between all these observations on iris flowers, we want to try to group them into groups. This dataset is actually very relevant since these flowers are indeed of a certain species (3 in total). We will try to find each flower species without explicitly using this parameter.

First let's see if we can graphically observe some groups. But since this data set contains 4 variables, we will have to use PCA and project them in the first axe. We will start by doing scatter plot (2 variables at a time) to see any useful information, and then do a PCA.

Scatter plots

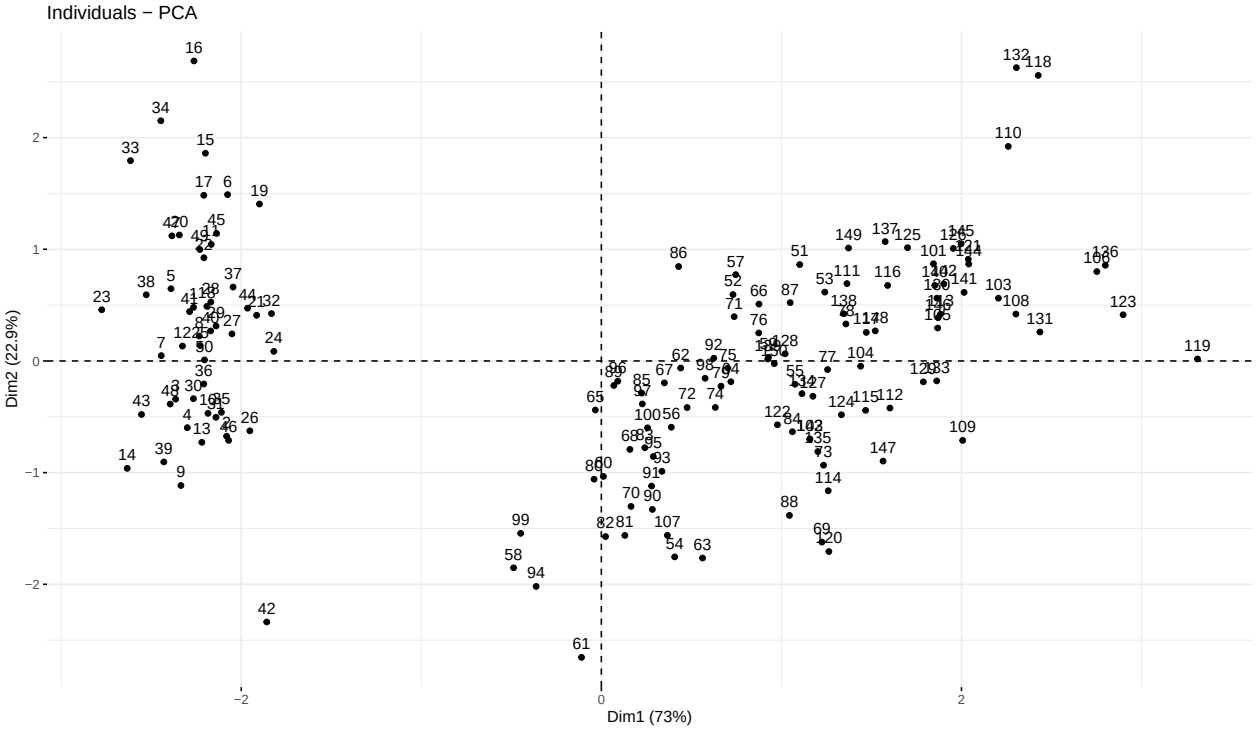




It seems that there are at least 2 groups. One of them also seems to be larger. Since it's hard to interpret lots of graphs at the same time, we will now do a PCA and plot only one graph.

```
library(FactoMineR)
pca_results = PCA(df, graph = FALSE) #apply PCA (by default the function scale the variables)

library(factoextra)
fviz_pca_ind(pca_results)
```

Principal component analysis



The projection is pretty good: more than 90% of the initial inertia is conserved in the first new axe. It still seems that there are at least 2 well defined groups even though the right one is larger than the other (maybe there is 2 groups in the right one). We still don't know if the groups that we see here are related to the original species since we did not use this information for the moment.

Hierarchical clustering analysis (HCA)

There are 2 ways of doing HCA: agglomerative (each observation starts in its own cluster, and pairs of clusters are merged as one moves up the hierarchy) and divisive (all observations start in one cluster, and splits are

performed recursively as one moves down the hierarchy).

Top-down (or divisive)

The algorithms starts by considering all the data set as 1 cluster. It means that at the first step of this algorithm, any data set starts with 1 cluster (unlike the previous algorithm which depends on the sample size). The algorithm will, at each step, divides in 2 clusters a cluster from the previous step until the number of clusters equals the sample size.

Bottom-up (or agglomerative)

The algorithm starts by considering all observations as a cluster. For example, the iris contains 150 observations and would have 150 clusters at the first step of this algorithm. The algorithm will, at each step, merges 2 clusters into 1 until there is nothing more that one big cluster with all observations. At the end of the algorithm, we are able to represent clusters into a tree that summarizes all the past step. It gives us all the partition possible for our data set. We will focus on this algorithm.

The way used in order to know which observations are most similar (have the same characteristics) is to calculate the distance between them. There are lots of different way to do this, but we will only use Euclidean distance since it works well and is easy to understand. The Euclidean distance, between to points, on a data set with 4 variables (like the iris data set) is defined as follow:

$$d(x_1, x_2) = \sqrt{\sum_{i=1}^{4}(x_1 - x_2)^2}$$

Let's break down a concrete example (the distance between 2 points using only 2 of the 4 variables)

```
df$Sepal.Length[1] #position of the 1st observation of the iris data set on the Sepal.Length variable
```

```
## [1] 5.1
```

```
df$Sepal.Width[1] #position of the 1st observation of the iris data set on the Sepal.Width variable
```

```
## [1] 3.5
```

```
df$Sepal.Length[2] #position of the 2nd observation of the iris data set on the Sepal.Length variable
```

```
## [1] 4.9
```

```
df$Sepal.Width[2] #position of the 2nd observation of the iris data set on the Sepal.Width variable
```

```
## [1] 3
```

Now, let's do calculate the Euclidean distance between those observations:

$$d(i_1, i_2) = \sqrt{(5.1 - 4.9)^2 + (3.5 - 3)^2} \approx 0.539$$

Now we verify this result:

```
temp_df = df[c(1:2), c(1:2)] #subset the first two individuals and the first two variables
dist(temp_df, method = "euclidean") #compute euclidean distance between i1 and i2
```

```
##           1
## 2 0.5385165
```

From now on, you are supposed to know what the (Euclidean) distance between two observations is. The algorithm we use will first calculate all the distances between the observations and make a matrix of them. Fortunately, there is a function that does this in R.

```r
distances_matrix = dist(df, method = "euclidean") #save all the distances
```

From this matrix, the algorithm will group the 2 closest observations into a single cluster. As the algorithm only stops as long as there is no more than one global cluster, we need to define a way to compute the distance between clusters. Intuitively, **we might want to use the distance between centroids** ("center" of a cluster). However, this method has a problem: inversions can be observed in the tree. There are also many other methods, which optimize (more or less) different criteria.

We will only talk here about one of the most useful ones: the **Ward's distance**. This metric has good properties: there can be no inversions in the tree, it avoids the chain effect (however useful in some data sets) that one can have with some methods, and the partition constructed at each step maximizes the inter-cluster inertia between the partitions resulting from the aggregation of two classes of the previous partition. In other words, this algorithm makes sure that the inter-cluster inertia is as high as possible (compared to the previous step), and this is exactly what we want (**remember: maximizing the inter-cluster inertia is equivalent to minimizing the intra-cluster inertia**).

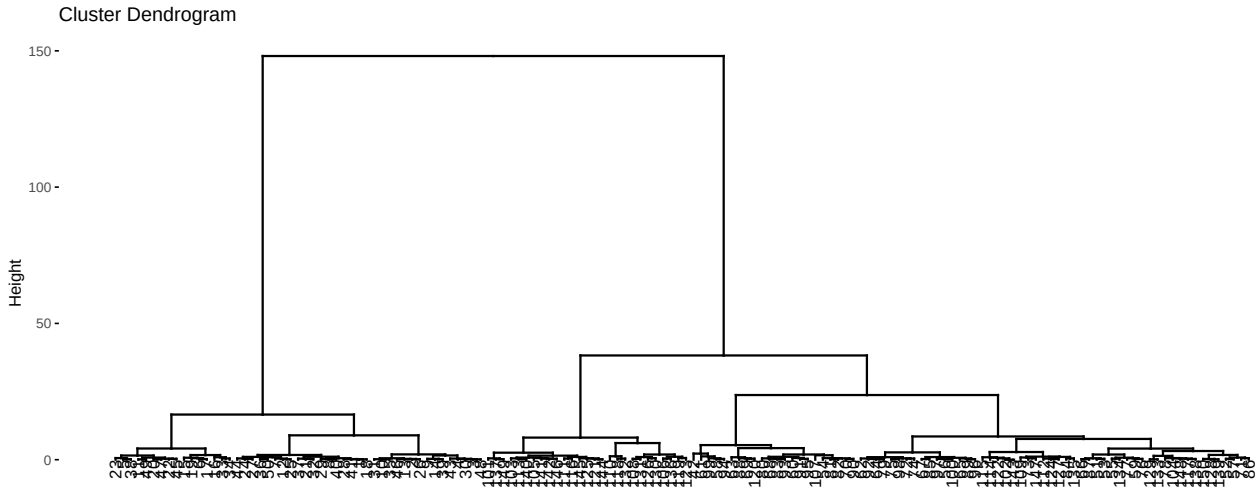
The Ward's distance is define as follow:

$$d_{Ward}(c_1, c_2) = \frac{\mu_1 \mu_2}{\mu_1 + \mu_2} d^2(g_1, g_2)$$

With:
- $\mu_k = \sum w_i$ the weight of the k_{th} cluster. It means that we can add ponderation to the metric (however, we will generally use $1/n$, with n the sample size)
- d^2 the squared euclidean distance
- g_p the gravity center of the p_{th} cluster.

And now again, there a R function that do all the calculations. The following function directly create the final tree. We will comment it right after. Before applying this algorithm, we will standardize our data set (transform in z-score) in order to not give any variable different weight from others.

```r
scaled_df = scale(df)
scaled_df = as.data.frame(scaled_df)
scaled_distance_matrix = dist(scaled_df, method = "euclidean")
hca = hclust(scaled_distance_matrix, method = "ward.D")
fviz_dend(hca)
```

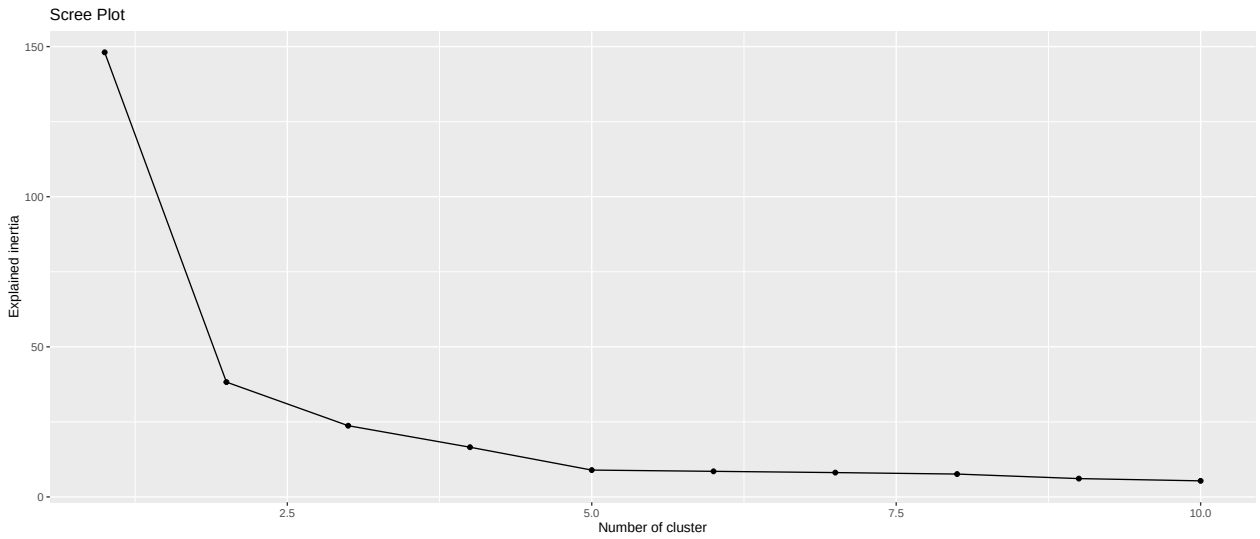


In order to represent when the algorithm aggregates 2 clusters (or observations, during the first steps), it creates a line between the points. This way, we can easily see the reunification in the clusters.

It is now time to determine how many clusters we want to have. The more we have, the best the partition explain the data. However it's not parsimonious to do is. The parsimony is an important concept and is

important when doing statistics. Generally, the goal of clustering algorithm is to explain the more variance possible. Put another way, we are looking for the number of cluster k that is parsimonious AND that explains a sufficient amount of inertia. As said before, clustering might imply some subjective decisions. In order to facilitate those, we decide to plot the amount of explained inertia depending on the number of clusters. There is a great function for that. One famous method to determine the number of cluster is the elbow method: in the following graph (think of it as an arm), where is the elbow?

```
qplot(c(1:10), sort(hca$height, decreasing = TRUE)[1:10]) +
  geom_line() +
  xlab("Number of cluster") +
  ylab("Explained inertia") +
  ggtitle("Scree Plot")
```



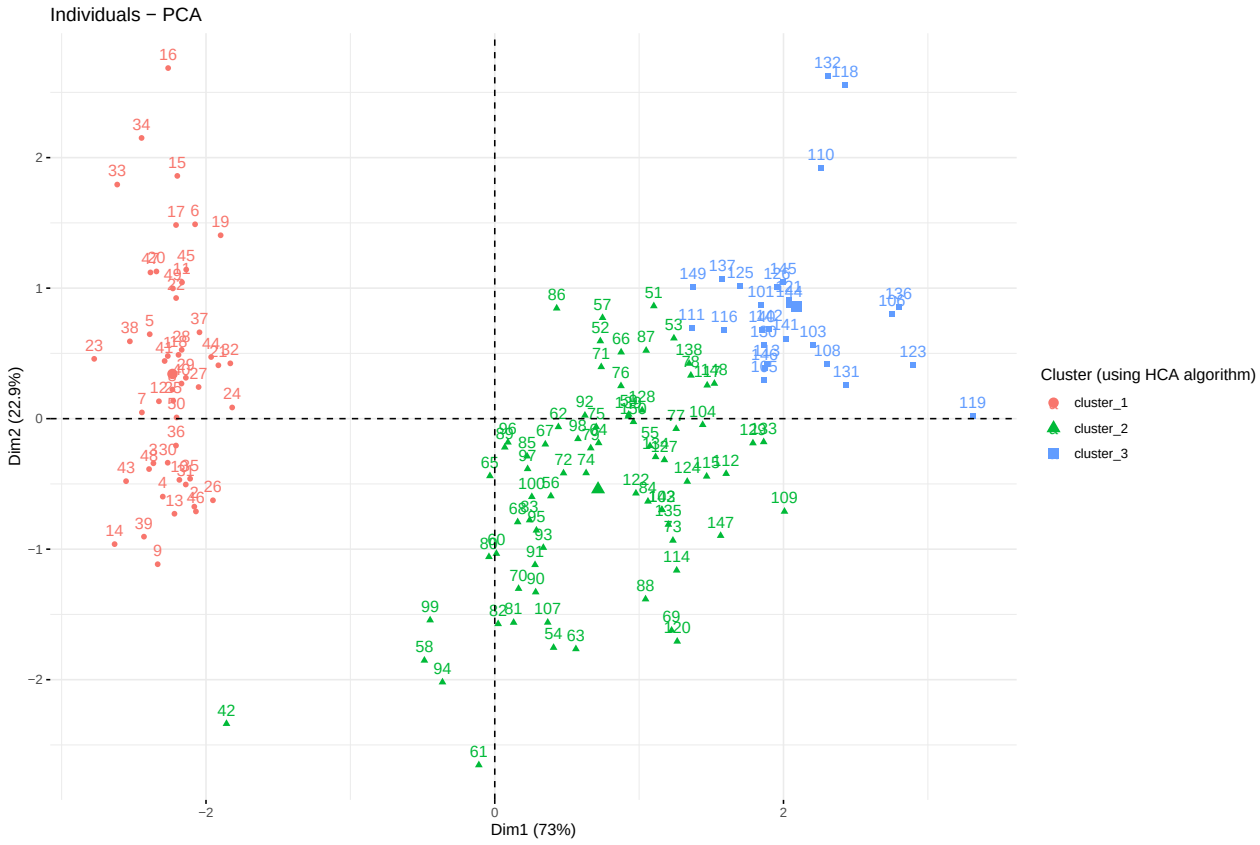
We see that the explained inertia does not much rise after 5 clusters. The "right" (if it exists) number of cluster seems to be between 2 and 5. We said before that there actually is a qualitative variable that goes with this the data set, the species. There is 3 of them. Knowing this, we now want to chose a cluster partition of 3. But in real life, we don't know this. Let's still pick 3 and see what we can do about this information.

First, we can add a new variable that will contains the cluster number of the observation.

```
#we split the tree in 3, according to the number of clusters wanted
group_labels = cutree(hca, k = 3)

#add this variable to our dataset
scaled_df$group = group_labels

#convert it to a factor
scaled_df$group = as.factor(scaled_df$group)

#make the variable more lisible (facultative)
scaled_df$group = paste("cluster", scaled_df$group, sep="_")

#indicates that we want the cluster variable as a supplementary variable (the 5th)
results = PCA(scaled_df, quali.sup=5, graph=FALSE)
```

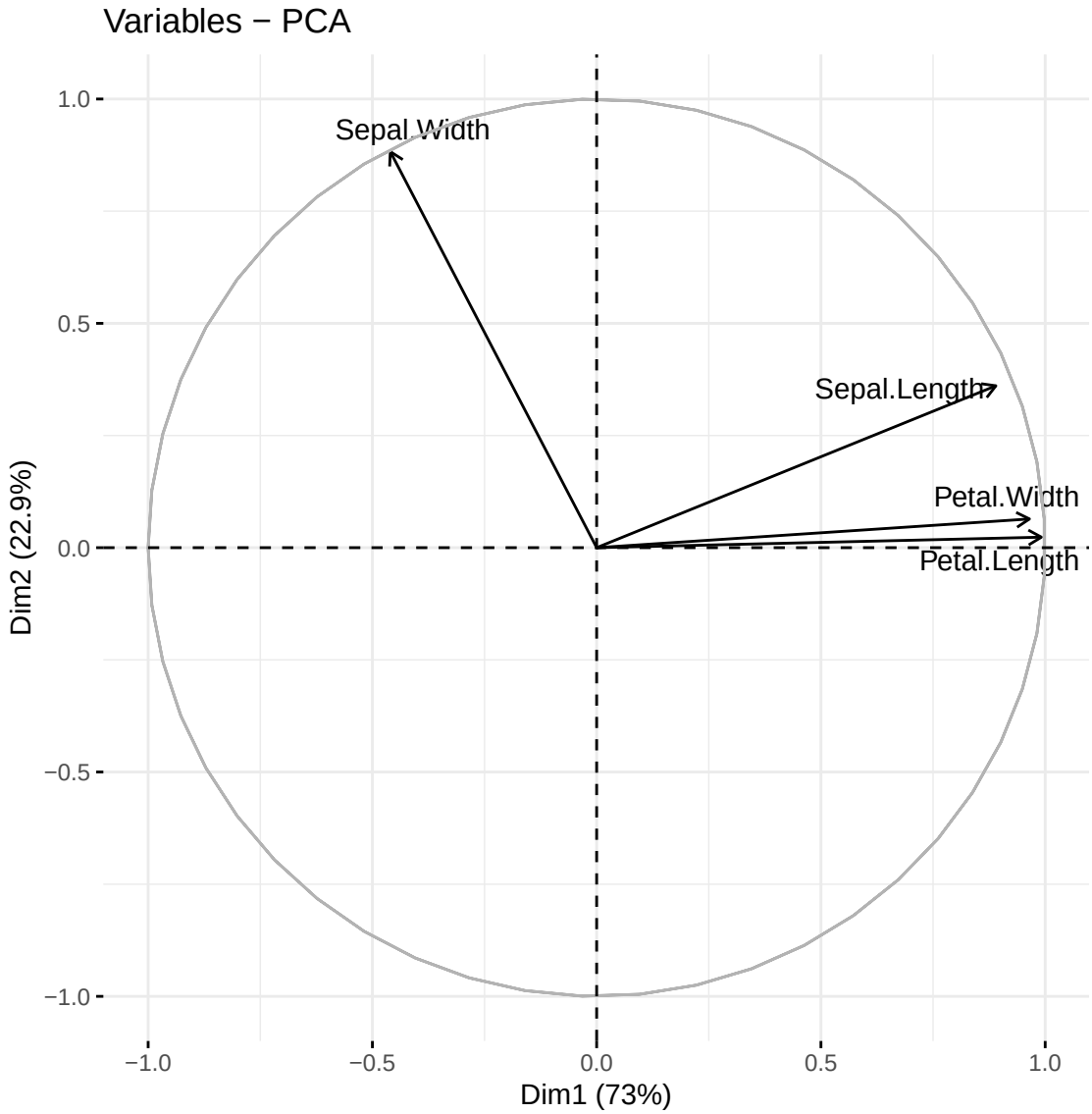
Now let's see what the individuals projection looks like when we color the point according to the cluster number.

```
#ind projection
fviz_pca_ind(results, axes = c(1,2),
             col.ind = scaled_df$group,
             legend.title = "Cluster (using HCA algorithm)")
```



The partition seems to make sense! However the 2nd and the 3rd group are a bit confused (we'll talk about that later). Now we want to interpret our clusters. But since we did a PCA on the original data set, the axes are not basic variables. In order to know what they represent, we first need to check the projection of the variables.

```
fviz_pca_var(results, axes = c(1,2), repel = TRUE)
```



The first thing that we see here is the fact that the principal component is highly correlated to the petal width and the petal length. It also seems that the sepal length and the sepal width are orthogonals (uncorrelated). They both contributes to the second axe, even though the sepal width probably has a higher contribution to it. This interpretation could be way longer, but this article is not about principal component analysis.

Here's one way to interpret the clustering:

- Cluster 1 represents the flowers with the lowest petal width and length (probably not the most beautiful flower since they have small petal). However, some of them might have a relatively high sepal width and others a high sepal length
- Cluster 2 represents the flowers with a relative important petal width and length. They probably not have the highest sepal width and a moderate sepal length.
- Cluster 3 represents the flowers with the highest petal width/length and sepal length.

Now that we have an idea of how our clusters represent, we want to compare these results when use the original of the species of the flowers instead of the clustering number for the projection.

```
results = PCA(iris, quali.sup=5, graph=FALSE)
```

```
#ind projection
fviz_pca_ind(results, axes = c(1,2),
             col.ind = iris$Species,
             legend.title = "Cluster (by species)")
```

These are very similar! There are some differences but the species are very similar to the clusters. What does it mean (try to answer this question by yourself first)? It means that there is a way to find the species of iris flowers only using its dimension: these things are correlated. It might be a good idea to look into the flower's genes in order to maybe find more information.

You now how to implement and interpret a hierarchical clustering algorithm!

K-means