Machine Learning for Risk Managers

K-means Clustering Part 1

Disclaimer: In this R Notebook, you can find the code I have used in the second video lesson of the ML4RM course (https://youtu.be/fYmOkEezR30); a basic understanding of R and R Studio is however assumed. If you need a quick introduction to R, I can suggest the following resources:

- An Introduction to R from the R-Project: https://cran.r-project.org/doc/manuals/r-release/R-intro.pdf
- Video Lessons on R: https://www.edx.org/course/introduction-to-r-for-data-science-2
- The R For Data Science Website and Book: https://r4ds.had.co.nz
- R-Studio Markdown and Notebooks: https://rmarkdown.rstudio.com/lesson-1.html

Notice that the code I am sharing is not always the most efficient one. Sometimes I prefer to use less elegant and less compact formulations for the sake of clarity. If you can write a better code, feel free to do it, it is always a good exercise.

Preliminaries

To guarantee that we all get the same results, let us set a seed for R

```
set.seed(10)
```

Data Import and Exploration

Let us import the dataset "BankCustomers.txt", which contains comma separated values (csv). It goes without saying that you will have to modify the path in your code.

We store the information in the dataset `Customers`.

Being a csv dataset, the `read.csv` function is a natural candidate for the job. However, in R studio you can also simply use the "import button".

```
Customers=read.csv("C:/Users/marco/OneDrive/Documentos/R/Kmeans/BankCustomers.txt")
```

Let us now have a look at the data, using for example the function `head`, to understand its structure. This function only shows the first lines, just to give an idea of the dataset.

```
head(Customers)
```

```
##   id   Feature1   Feature2 Default
## 1  1 -0.4881956 -0.2194427       0
## 2  2 -0.2086393  0.4331034       0
## 3  3  1.5239642 -2.8344796       1
## 4  4 -0.6351854 -1.3853834       0
## 5  5  2.1795895  1.0813874       0
## 6  6 -0.3101013  0.1244764       0
```

Tha dataset is a small and pedagogical (yet real) sample from a larger dataset, containing information about hundreds of customers of an important European bank. *I have the permission to share some of the data, but mainly in anonymized form.*

Each row in the dataset corresponds to a customer, whose name has been replaced by a number in the `id` column. We have 72 customers in total.

For each customer we know two features (columns 2 and 3): Feature 1 is a measure of the unbalance between income and consumption patters (higher values indicate "spendthrift" customers, lower values strong savers); Feature 2 is a measure of creditworthiness (the lower the worse, with a value close to 0 indicating what the bank deems acceptable). The last colum, `Default`, is a dummy variable telling if the customer has defaulted (1) or not (0). The number of defaulted customers is equal to 25 (we simply sum the ones in the column `Default`).

A nice function, which I always suggest to use, is the `summary` one, which gives us basic info about the variables we have in a dataset. Since the first column contains a simple counter, and the last one just a dummy about the default state, we only focus our attention on the two columns containing the "features". A quick look at the data suggests that these can be continous variables.

```
summary(Customers[,2:3])
```

```
##     Feature1          Feature2
##  Min.   :-1.8345   Min.   :-4.64413
##  1st Qu.:-0.4749   1st Qu.:-1.47375
##  Median : 0.2468   Median :-0.59511
##  Mean   : 0.5102   Mean   :-0.73882
##  3rd Qu.: 1.3212   3rd Qu.: 0.09909
##  Max.   : 3.4224   Max.   : 1.91972
```

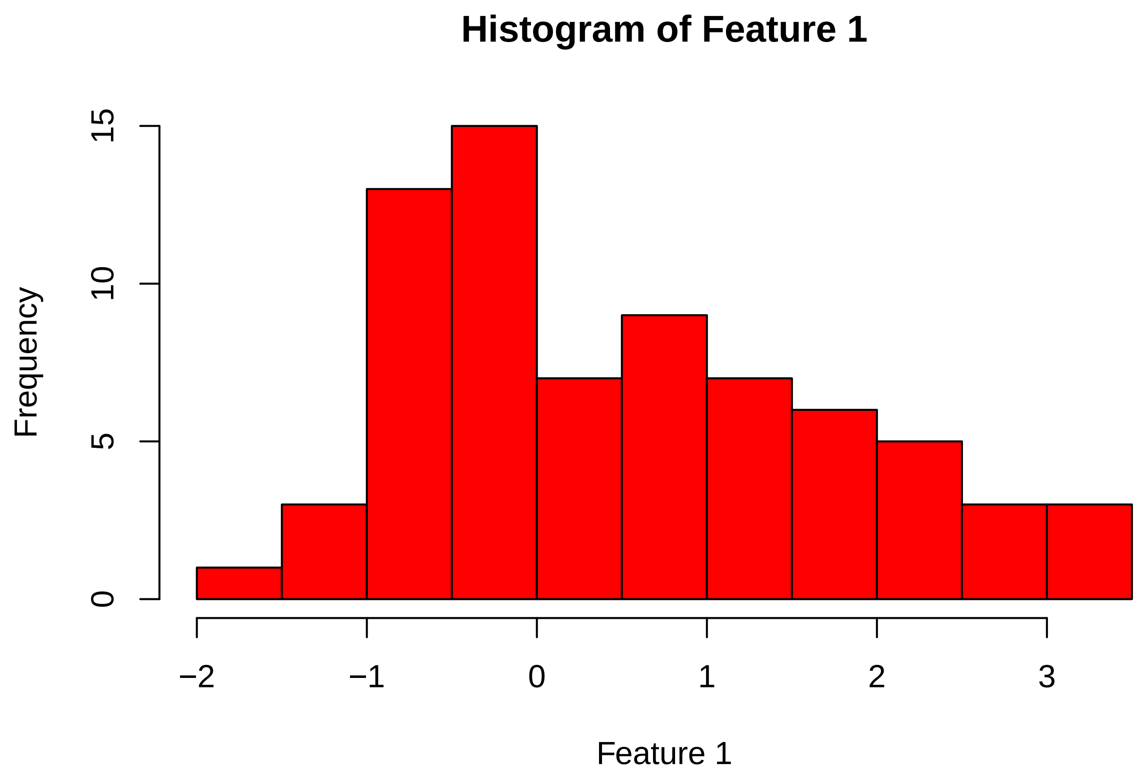
The customers of this bank tend to be, on average, moderately creditworthy (especially compared to other banks), with accetable expenses patterns.

A trained risk manager (or simply someone who knows statistics decently) would probably observe that these features show a relatively strong negative correlation (equal to -0.4877655), so that some information could be redundant (this is a problem we will discuss with linear models, later in the course), but for the moment we are satisfied with what we have observed (also considering the limited number of variables).

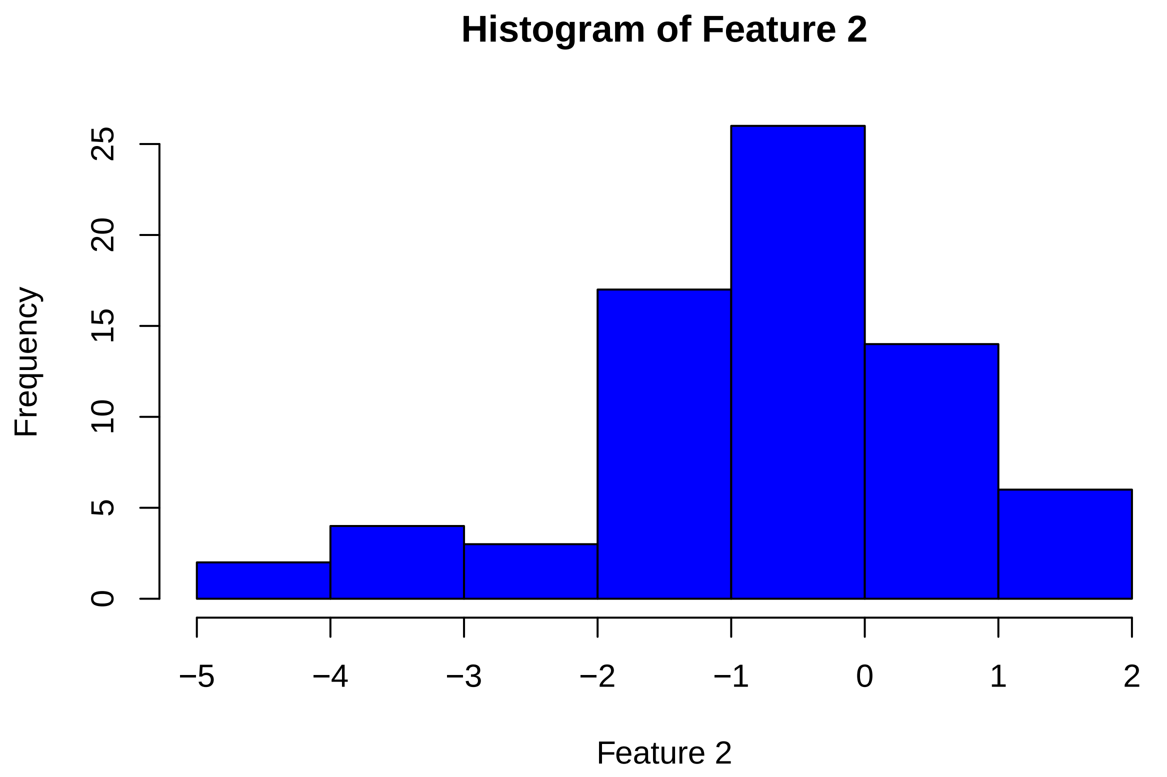
The two features seem to have ranges of variation that are compatible in terms of magnitude. Yet, looking at their quantiles above (min, max, median, quartiles), we see that Feature 1 is skewed to the right, while Feature 2 is skewed to the left. This is clearer if we have a look at the histograms (always plot your data!).

A simple histogram can be obtained with the function `hist`.

```
hist(Customers$Feature1,col='red', main='Histogram of Feature 1', xlab='Feature 1')
```



```
hist(Customers$Feature2,col='blue',main='Histogram of Feature 2', xlab='Feature 2')
```

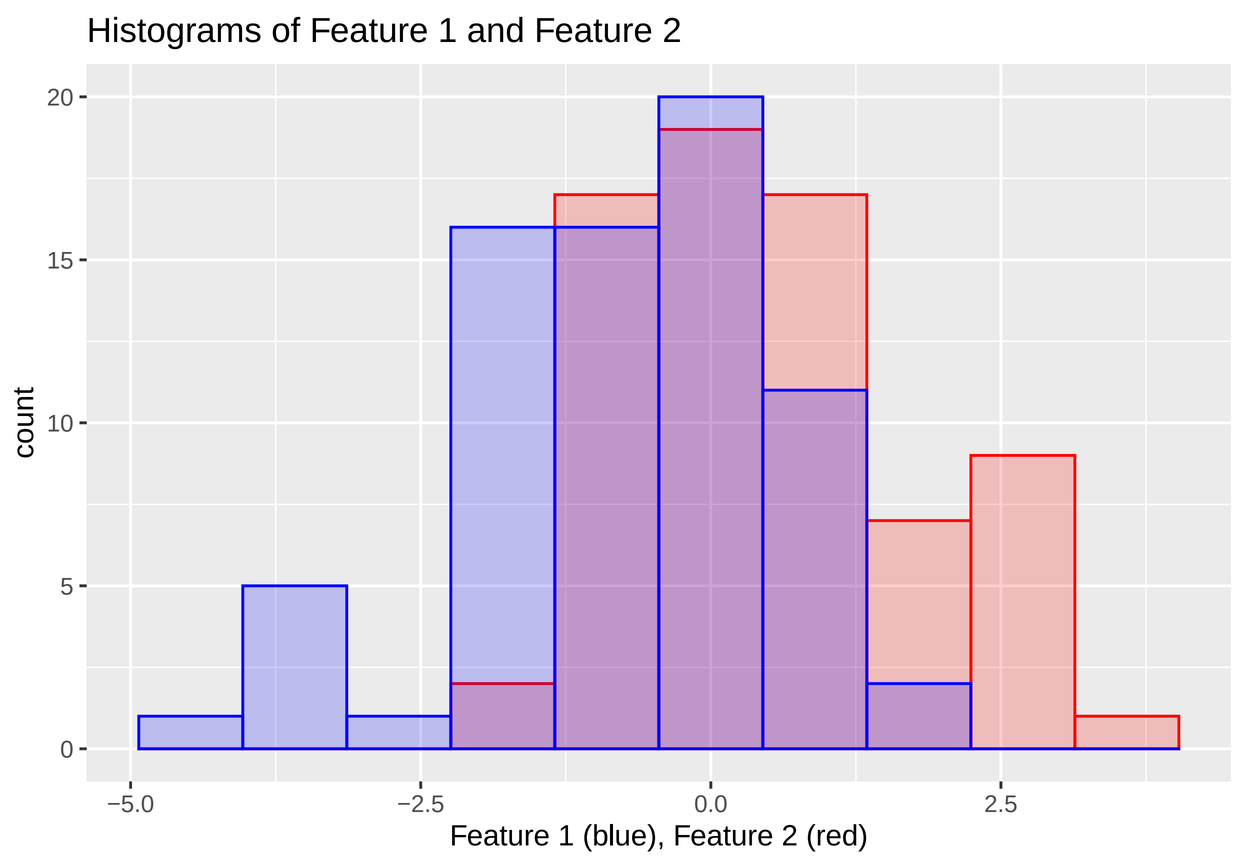


Notice that the number of bins in the two figures is different, because R tries to provide the best picture (according to it). If you want to force the number of bins, you can use the `breaks` option (type `?hist` in your console).

If you aim to produce better and more appealing graphics, a very useuful package is `ggplot2`, for which I refer to the ggplot2 website. Since we will often use the graphical grammar of `ggplot2`, I suggest having a look.

The two histograms above can be (better) re-obtained as follows, in one simple picture, confirming the (opposite) skewed nature of Feature 1 and Feature 2.

```
library(ggplot2)
ggplot(Customers, aes(Feature1))+geom_histogram(color="red", fill="red",bins=10,alpha=.2)+
   geom_histogram(aes(Feature2), color="blue", fill="blue",bins=10,alpha=.2)+
   labs(title='Histograms of Feature 1 and Feature 2', x='Feature 1 (blue), Feature 2 (red)')
```



Playing with `ggplot2` is an exercise that takes time, but you can learn something really useful and beautiful.

Scaling

As we have said in the video lesson, a clever thing to do when performing data analysis, and in particular with clustering techniques like the *k-means* algorithm, is to standardize the data. Here I only consider the common approach via mean and standard deviation (normalization), but feel free to experiment a little bit with the min-max scaling

In R you can normalize observations in many different ways. A naive one is to work with the functions `mean` and `sd` directly. Another more elegant solution is to use functions like `scale`.

In our dataset, only two variables need to be normalized: Feature 1 and Feature 2. To be honest these features are not dramatically different in terms of magnitude, and one could avoid standardization, yet it is a very good habit to standardize, hence we will do it anyway. Recall: if you standardize you are rarely (or never) wrong, if you don't you are rarely right.

My suggestion is to create a new `Customers` dataset (we call it `New`), in which Feature 1 and 2 are standardized, while the other columns (`id` and `Default`) are left untouched. The creation of a new dataset allows us to easily come back to the original data, if we make some mistake. *Better safe than sorry.*

```
New=Customers
New[,2:3]=scale(New[,2:3]) # we scale the columns of interest only
head(New)
```

```
##   id   Feature1   Feature2 Default
## 1  1 -0.8032619  0.3934122       0
## 2  2 -0.5783427  0.8876914       0
## 3  3  0.8156362 -1.5873803       1
## 4  4 -0.9215235 -0.4897442       0
## 5  5  1.3431244  1.3787423       0
## 6  6 -0.6599748  0.6539180       0
```

You can easily verify that now Feature 1 and Feature 2 have mean 0 (or very close) and standard deviation 1.

The K-Means Algorithm

Let us perform a first clustering of our data using the k-means algorithm.

Let us say that, initially, just looking at the information provided by the variables Feature 1 and Feature 2, we are interested in just two groups, and we want to verify if the k-means algo is able to re-obtain the information provided in the `Default` column, which we use as a benchmark/test. In other words, just looking at the feature variables (columns 2 and 3), and pretending we do not know which customers have defaulted (column 4), can we find out which customers are likely to have defaulted? Such an information can be vital later on to study the PD (probability of default) of our counterparties, for example (and this dataset actually comes from a PD study I supervised).

In R, the k-means algorithm is easily used via the function `kmeans` (type `?kmeans`). Such a function has several options, but the most important for us are:

- `x`, i.e. the data we are interested to use. Since we plan to use our standardized Features 1 and 2, we will use `New[,2:3]`.
- `centers`, that is to say the number of groups we want R to obtain. Recall that in the k-means algorithm, the number of clusters needs to be known in advance. If we look for two groups, we set `centers=2`.
- `nstart`, which indicates the number of guesses that should be used. I suggest a number larger than/equal to 30 (others say 20 is enough). Such a number will tell R to start with at least 30 different initial guesses for the starting centroids expressed by `centers` (two for us), and to provide us with the best result. For medium-large datasets, and in particular if your data are not as good as the ones I have collected for you, using only one or a few guesses can give unreliable results.

How "best" is defined will be discussed in the next lessons, when we also discuss how to find the optimal number of groups, in case we do not have a strong opinion/requirement.

So, let us use the algorithm and store the information in `km.out`, obtaining a list variable with many interesting features (some will be clearer in the next lessons).

```r
km.out=kmeans(New[,2:3],centers=2,nstart=30)
```

For example, by typing `km.out$centers` we can have the coordinates of the final centroids, i.e. those that will no longer change, thus stopping the algorithm. These are the average points of the two groups R has identified for us.

```r
km.out$centers
```

```
##     Feature1   Feature2
## 1 -0.4772826  0.4831026
## 2  1.0847333 -1.0979604
```

And with `km.out$size` we discover how many observations fall in one group or in the other.

```
km.out$size
```

```
## [1] 50 22
```

Hence one group contains 50 customers, while the other only 22.

The information about the clusters can be found in `km.out$cluster`.

```
km.out$cluster
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
##  1  1  2  1  1  1  2  1  1  2  1  1  2  1  1  1  2  1  1  2  2  1  1  1  1  1
## 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52
##  1  1  2  1  1  1  1  1  1  1  2  2  1  2  1  2  2  2  1  1  1  1  1  1  2  2
## 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72
##  1  1  1  2  1  1  2  1  1  1  2  1  2  1  1  2  2  1  1  1
```

According to our k-means, for instance, customers 1 and 4 belong to the same group, while customer 21 and 68 belong to the other one.

We can create a brand new column in our `New` dataset, to store this important piece of information.

```
New$Cluster=km.out$cluster
head(New)
```

```
##   id   Feature1   Feature2 Default Cluster
## 1  1 -0.8032619  0.3934122       0       1
## 2  2 -0.5783427  0.8876914       0       1
## 3  3  0.8156362 -1.5873803       1       2
## 4  4 -0.9215235 -0.4897442       0       1
## 5  5  1.3431244  1.3787423       0       1
## 6  6 -0.6599748  0.6539180       0       1
```

Comparing this new column with the `Default` one, we can see that the k-means algorithm appears to be able to separate defaulted and non-defaulted customers, even if this information was not explicitly given to it. This is a simple and clear example of machine learning: our computer has learnt from the data, modifying its results (recall how the algorithm works) until it finds a meaningful pattern, in its opinion.

But computers, for the moment, are stupid, hence it is our job to verify whether the results above make sense or not to us. The next exercise asks you to find the answer.

Exercise 1

Is the k-means algorithm really able to assign all customers to the appropriate group? Do all defaulted customers (those with 1 in `Default`) belong to the same group in `Cluster`? What about the non-defaulted ones? There are several ways to check this, feel free to use the method you prefer.

You should discover that the algorithm actually works very well (in this small sample, and with these nice variables), but that there are a few customers that are wrongly classified. This is totally understandable, if we consider the small number of observations and the fact that we are only using two features to classify each customer. Modeling defaults is a difficult task! In the next lessons we will say more about this.

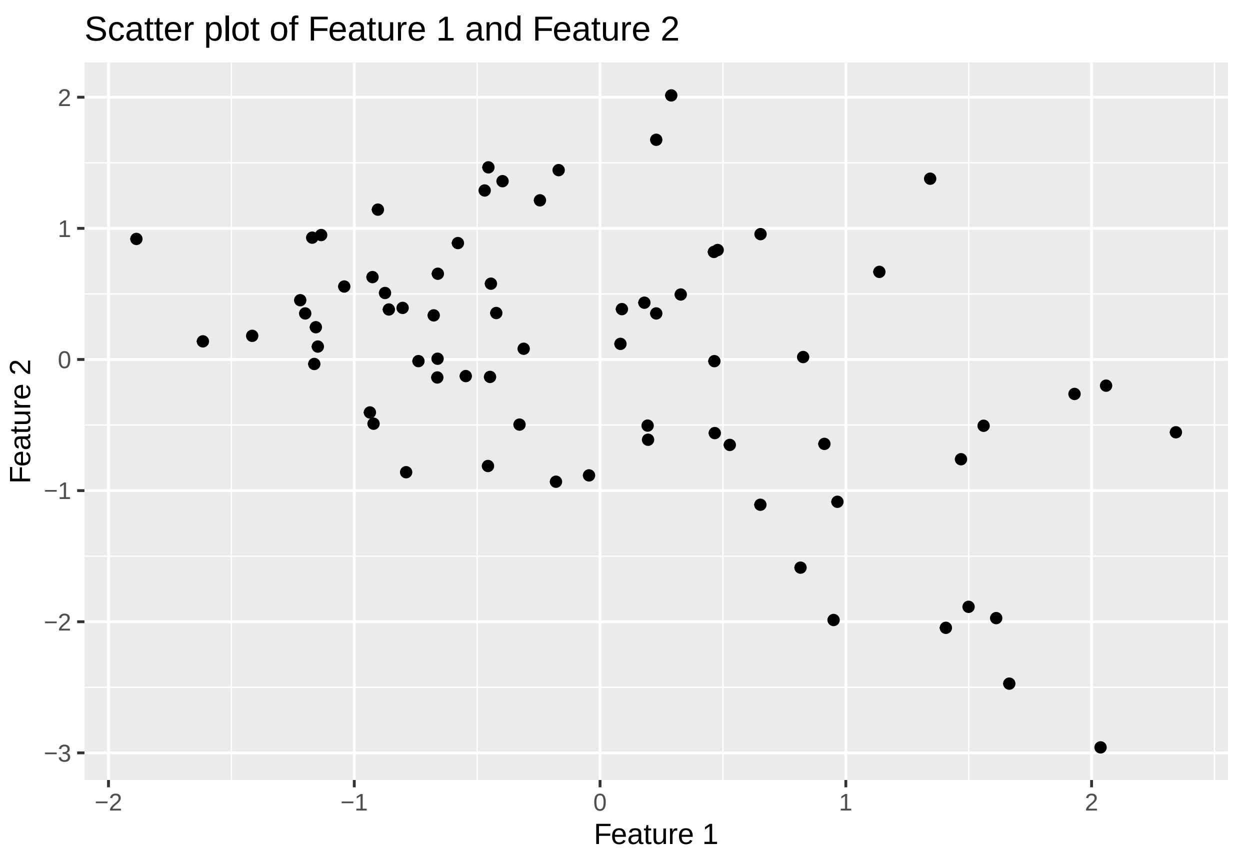
Remark: in real life, the misclassification of just a few customers may have serious consequences, that is why we have to be careful. Regulators are very picky about these things.

Plotting

The best way to profit from the k-means algorithm is graphically.

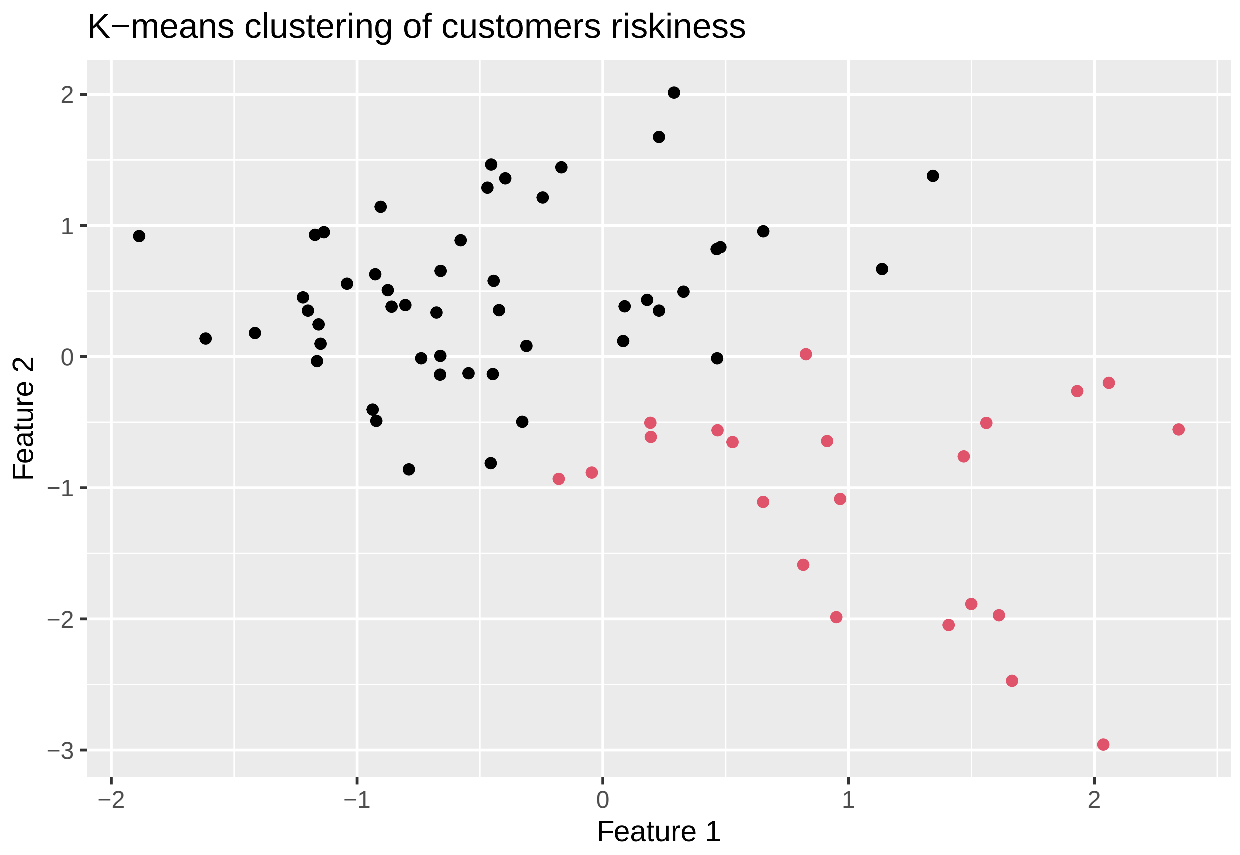
Let us start by plotting the information about Feature 1 and Feature 2, without any group distinction.

```
ggplot(New, aes(Feature1, Feature2))+geom_point()+
   labs(title='Scatter plot of Feature 1 and Feature 2',x='Feature 1', y='Feature 2')
```



Using the information stored in `km.out$cluster`, we can now color the different dots, to see whether we can observe clear homogeneous groups.

```
ggplot(New, aes(Feature1, Feature2))+geom_point(color=km.out$cluster)+
   labs(title='K-means clustering of customers riskiness',x='Feature 1', y='Feature 2')
```



It is relevant to notice that, using the k-means clustering information, we obtain two clearly separated groups (red and black). This means that Features 1 and 2 are definitely good in disciminating between defaulted and not defaulted customers (even if, we know, not perfectly, that's life!).

In principle, we could be able to find a diagonal line (or a simple curve) separating the red dots from the black dots. This is for instance what we will do with discriminant analysis later in the course.

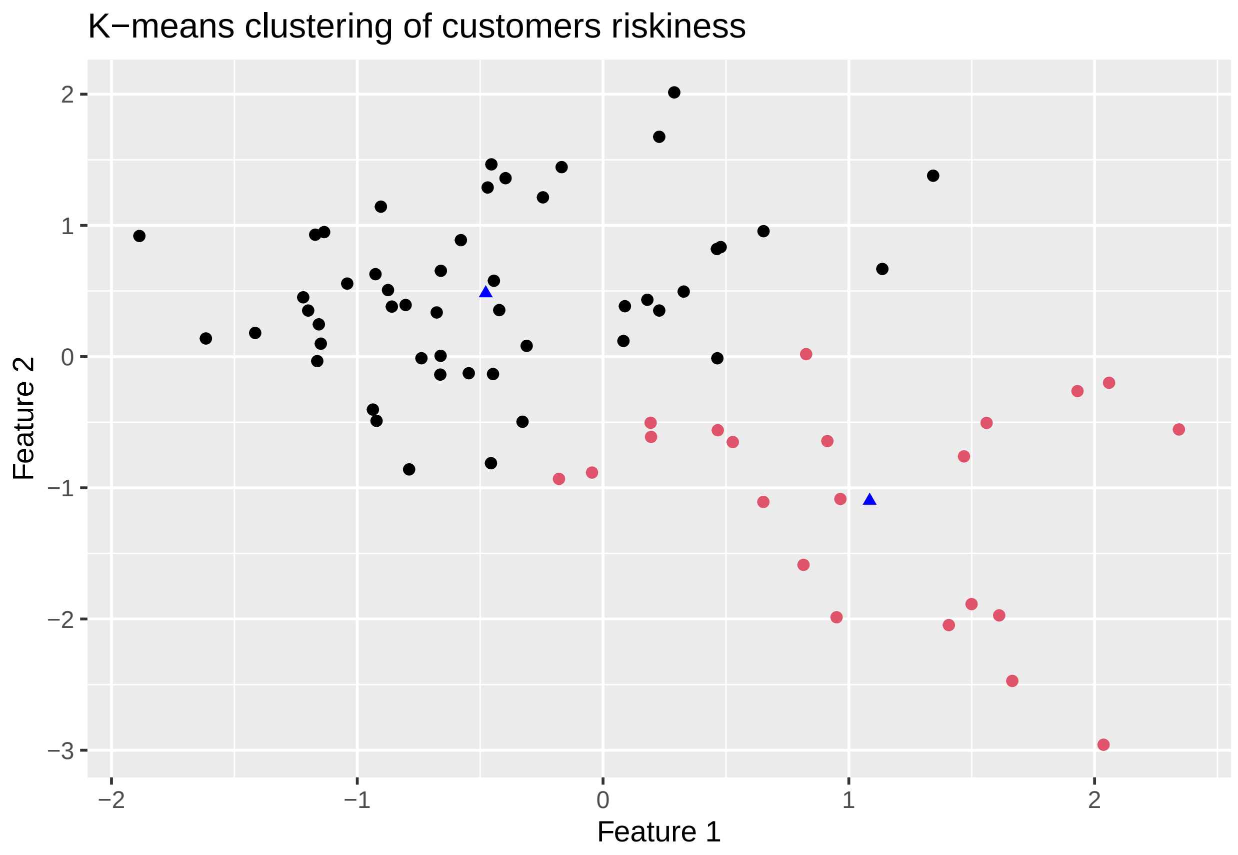
The plot above can be improved in many different directions. For example we could plot the centroids of our two clusters.

Let us start by creating a simple data frame containing the centroids.

```r
centroids = data.frame(Feature1 =km.out$centers[,1], Feature2 =km.out$centers[,2])
```

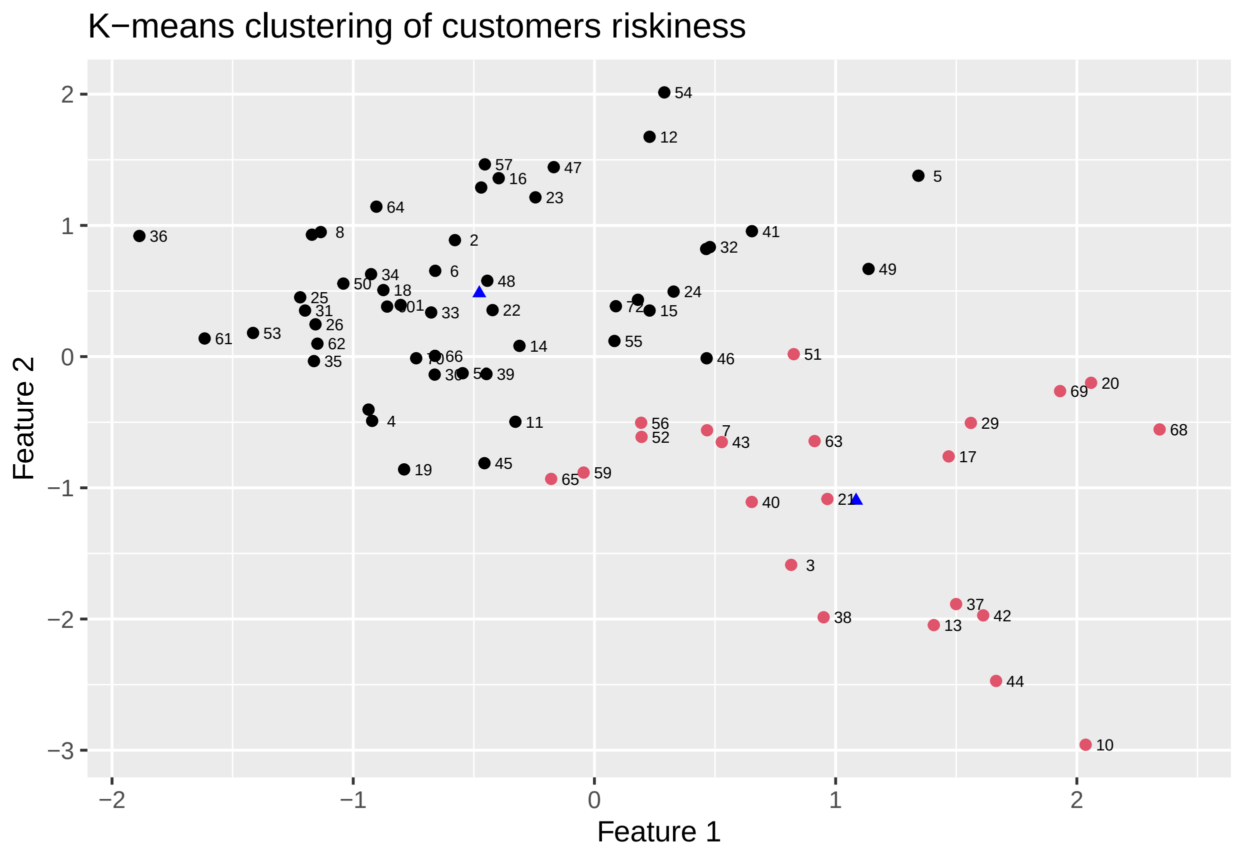
We can then add the two centroids as–for instance–little blue triangles. Not bad.

```r
ggplot(New, aes(Feature1, Feature2))+
   geom_point(color=km.out$cluster)+ geom_point(data=centroids, color ='blue',shape=17)+
   labs(title='K-means clustering of customers riskiness',x='Feature 1', y='Feature 2',
   color='Group')
```



Finally we can add the `id` info, to better understand the clusters.

```
ggplot(New, aes(Feature1, Feature2))+
   geom_point(color=km.out$cluster)+ geom_point(data=centroids, color ='blue',shape=17)+
   geom_text(label=New$id,nudge_x = 0.08,size=2,check_overlap = T)+
   labs(title='K-means clustering of customers riskiness',x='Feature 1', y='Feature 2',
   color='Group')
```



This plot tells us something interesting (but also not surprising): defaulted customers (like number 10 or 29) tend (or we should say tended, given that we know they have defaulted) to have unbalanced expenses (spendthrifts) and a worse rating. The others, who have not defaulted, conversely tend to save more money and have higher ratings. This information could be used, for example, to have a better idea of our customers, so that we could contact the not-yet-defaulted ones, who have a large Feature 1 value, trying to understand what is going on.

Other interesting objects like legends or further labels can be easily added to our plots. I leave this sort of exercise to your own curiosity. For example you could add the `Default` info in some way, or look for the misclassified observations, see Exercise 2.

Exercise 2

Try to answer again to the questions in Exercise 1, but this time graphically (if you have not yet). Can you give a third color (or a different symbol) to the misclassified customers? Where are they? What is your interpretation?

Exercise 3

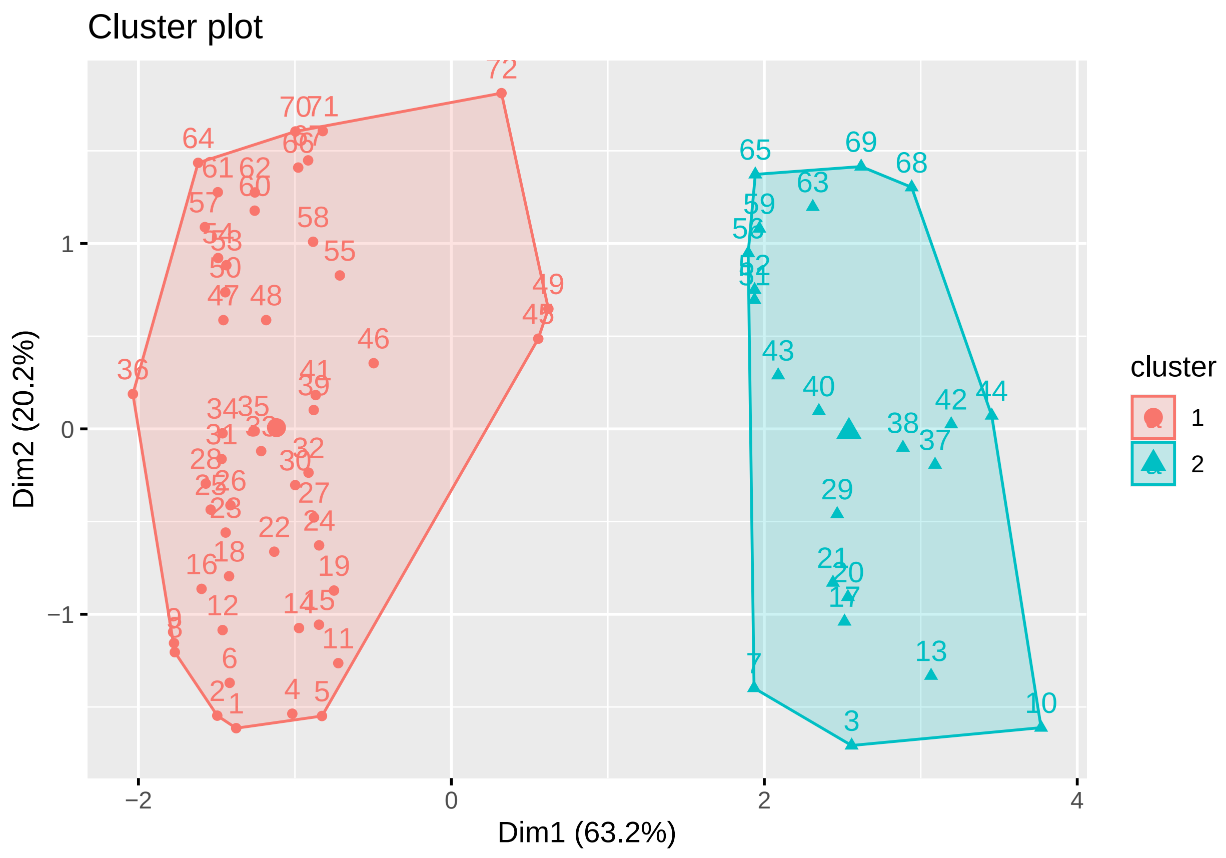
What happens if we look for 3 or 4 groups, instead of just 2? How do your plots vary? Is the separation among groups still as nice as before? Can you find (only heuristically for the moment) a simple separating line/curve? Yes or no? How could we use this new information, that no longer corresponds to `Default`*?*

The `factoextra` library

The `factoextra`library is extremely useful when playing with clustering techniques, as it contains several functions to quickly produce several nice plots and meaningful statistics. In the next lesson we will use it, hence you can already install it.

Just as a preliminary example, using the function `fviz_cluster` we can have an alternative plot of the clusters we have obtained via the k-means algorithm.

```
library(factoextra)
fviz_cluster(km.out, data = New)
```



As you can easily observe, this plot is different from our previous ones. The main difference lies in the axes, which are no longer Feature 1 and Feature 2, but new objects obtained via PCA (principal component analyisis), a noise and data reduction technique we will discuss later on. For the moment, just be patient. See this as a little spoiler.