Data exploration and statistics with R and the tidyverse

Vivien Roussez (A1 Telekom Austria Group)

Fall 2020

Contents

Chapter 1

Foreword

Welcome to this learning week in Asigmo's program !

This week will be dedicated to data exploration, manipulation and visualization using R and the tidyverse. To that end, we will also cover the basics of statistics, that are essential for your understanding of modeling and, later on, machine learning.

We will deal with complex and pretty dirty data, but this is real-world data, and you will also see that, more than artificial intelligence, you need common sense to deal with data !

This material was powered by bookdown

To contact me : vivien.roussez@gmail.com

Chapter 2

Introduction

2.1 Topics of the week

During this week, we will cover a lot of *basic* and nonetheless indispensable tools for a data scientist. We will cover mainly the "boring side" of data science (aka data analysis :P). As a matter of fact, if machine learning is more and more automatized, everything that is related to data **exploration**, **wrangling**, **cleaning**, **understanding** and **analysis** is hardly doable by "AI".

What's on our agenda :

- Introduction to R
- Data manipulation
- Introduction to statistics
- Data exploration
- Explainable machine learning (eg regression)
- Data visualization
- Data cleaning
- Dimension reduction

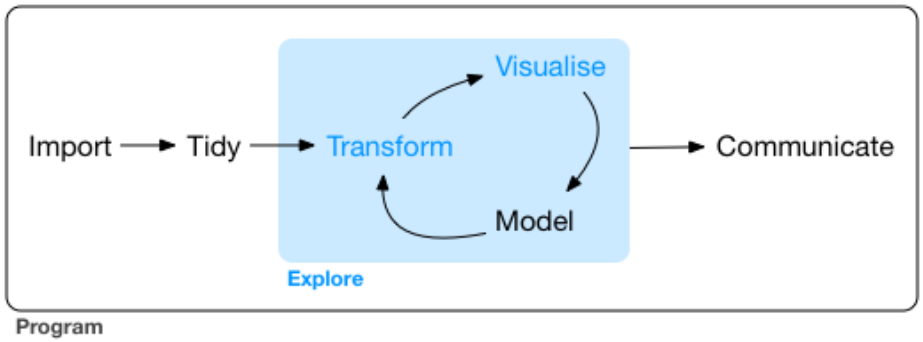
Important notes :

- Of course, it's not a linear path and the data science workflow is not a simple execution of those steps in a pre-determined order. It's all connected and you'll have to move back and forth between all of them to achieve your goal. And this goal, what is it already ?
- In data science is the word "science". And what is science ? Huge topic... But a few keywords that you should always have in mind when starting a project
  - Reproducibility
  - Hypothesis testing
  - Incremental

- Iterative
- Monitored

2.2 The data science workflow

A popular schematic overview of a the workflow



The iteration loop has to be done taking the business perspective / constraint into account !

2.3 Resources

You will find a lot of resources online. Here is a selection, mostly related to R. However, our goal for this session is to have you understand how all the methods we will cover are related to each other and when you should consider using them

- R for data science
- Statistics and econometrics
- Statistics and exploration with R
- From data to viz (featuring python :D)
- Data visualization
- More resource online books using R (text mining, machine learning…)

2.4 Data of the week

In this course, we will handle with my *own* personal data (I give my consent ! ;)) Those are my sports activity data, which I got from garmin (thanks to the GDPR). If you want to get your own data from this page. Being a triathlete, there are several sports involved and a lot of activities. The goal will be to import, clean, analyze the data with statistics and eventually build some first models (**explainable AI**).

The export results in a lot of JSON files and we will focus on the summary data. It's real world data and you'll see it's complicated, dirty and requires a lot of

preparation/manipulation !

```
require(tidyverse)
dat <- read.csv("Data/Sports/Activities.csv",header = T)
head(dat)
```

```
##   activityId       uuidMsb       uuidLsb                name activityType
## 1 5570974040  3.695224e+18 -7.098507e+18          En piscine lap_swimming
## 2 5566524321 -5.212790e+18 -8.140623e+18     Vienna Cyclisme      cycling
## 3 5561266034  1.271356e+18 -5.522844e+18 Korneuburg Cyclisme      cycling
## 4 5555881653 -2.923938e+18 -4.676832e+18       Vienna Course      running
## 5 5551811953  7.859043e+18 -7.909532e+18      Zwift - London virtual_ride
## 6 5551052200 -9.554086e+17 -6.866435e+18          En piscine lap_swimming
##   userProfileId timeZoneId beginTimestamp eventTypeId   rule sportType
## 1       1141258        124   1.600700e+12           9 public   GENERIC
## 2       1141258        124   1.600602e+12           9 public   CYCLING
## 3       1141258        124   1.600523e+12           9 public   CYCLING
## 4       1141258        124   1.600439e+12           9 public   RUNNING
## 5       1141258        124   1.600362e+12           9 public   GENERIC
## 6       1141258        124   1.600353e+12           9 public   GENERIC
##   startTimeGmt startTimeLocal duration distance elevationGain elevationLoss
## 1 1.600700e+12   1.600708e+12  3640221   300000            NA            NA
## 2 1.600602e+12   1.600609e+12 15064570 12293220        200100        196800
## 3 1.600523e+12   1.600530e+12 10382433  8360927         74500         74200
## 4 1.600439e+12   1.600447e+12  5393830  1792890         42600         42500
## 5 1.600362e+12   1.600369e+12  3626825  3492347         16000             0
## 6 1.600353e+12   1.600360e+12  3359145   285000            NA            NA
##   avgSpeed maxSpeed avgHr maxHr  calories startLongitude startLatitude
## 1   0.0986   0.1122    NA    NA  2547.532            NA            NA
## 2   0.8160   1.9315   144   176 16345.268       16.31524      48.20915
## 3   0.8053   1.9557   117   161  8744.572       16.33986      48.34994
## 4   0.3324   1.2475   155   176  4923.274       16.31587      48.21432
## 5   0.9629   1.6364   138   163  3083.855        0.00000       0.00000
## 6   0.1011   0.3524    NA    NA  2505.632            NA            NA
##   aerobicTrainingEffect avgFractionalCadence maxFractionalCadence
## 1                    NA               0.0000                    0
## 2                   3.5               0.0000                    0
## 3                   2.4               0.0000                    0
## 4                   3.0               0.1875                    0
## 5                   0.0               0.0000                    0
## 6                    NA               0.0000                    0
##   elapsedDuration movingDuration anaerobicTrainingEffect   deviceId
## 1         3955901        3099653                      NA 3968818126
## 2        16176575       15000179                     0.0 3968818126
## 3        11319866       10356078                     0.0 3968818126
## 4         5538505        5389603                     0.2 3968818126
```

```
## 5         3629000        3611000                    NA 3825981698
## 6         3585826        2858951                    NA 3968818126
##   minTemperature maxTemperature minElevation maxElevation        locationName
## 1             25             26           NA           NA                <NA>
## 2             19             29        18480        95480              Vienna
## 3             18             29        13980        32200           Korneuburg
## 4             19             27        24920        54400              Vienna
## 5             NA             NA          300         3420 City of Westminster
## 6             26             27           NA           NA                <NA>
##   maxVerticalSpeed lapCount endLongitude endLatitude activeSets totalSets
## 1               NA       34           NA          NA         NA        NA
## 2       0.43999939       25     16.25314    48.20399         NA        NA
## 3       0.34000092       17     16.38793    48.38009         NA        NA
## 4       0.08000183       18     16.31980    48.22264         NA        NA
## 5       0.16000004        1           NA          NA         NA        NA
## 6               NA       32           NA          NA         NA        NA
##   totalReps purposeful autoCalcCalories favorite    pr elevationCorrected
## 1        NA      FALSE            FALSE    FALSE FALSE              FALSE
## 2        NA      FALSE            FALSE    FALSE FALSE              FALSE
## 3        NA      FALSE            FALSE    FALSE FALSE              FALSE
## 4        NA      FALSE            FALSE    FALSE FALSE              FALSE
## 5        NA      FALSE            FALSE    FALSE FALSE              FALSE
## 6        NA      FALSE            FALSE    FALSE FALSE              FALSE
##   atpActivity parent maxRunCadence steps avgVerticalOscillation
## 1       FALSE  FALSE            NA    NA                    NA
## 2       FALSE  FALSE            NA    NA                    NA
## 3       FALSE  FALSE            NA    NA                    NA
## 4       FALSE  FALSE           104 15620                    NA
## 5       FALSE  FALSE            NA    NA                    NA
## 6       FALSE  FALSE            NA    NA                    NA
##   avgGroundContactTime avgStrideLength vO2MaxValue avgVerticalRatio
## 1                   NA              NA          NA               NA
## 2                   NA              NA          72               NA
## 3                   NA              NA          71               NA
## 4                   NA        115.6389          58               NA
## 5                   NA              NA          NA               NA
## 6                   NA              NA          NA               NA
##   avgGroundContactBalance avgDoubleCadence maxDoubleCadence avgPower
## 1                      NA               NA               NA       NA
## 2                      NA               NA               NA      260
## 3                      NA               NA               NA      202
## 4                      NA          172.375              208       NA
## 5                      NA               NA               NA      212
## 6                      NA               NA               NA       NA
##   avgBikeCadence maxBikeCadence strokes normPower avgLeftBalance
## 1             NA             NA    1198        NA             NA
```

```
## 2             83            114   17968  296.0000          49.92
## 3             82            107   12571  231.0000          49.90
## 4             NA             NA      NA        NA             NA
## 5             91            114       0  218.3049             NA
## 6             NA             NA    1152        NA             NA
##   avgRightBalance max20MinPower trainingStressScore intensityFactor
## 1              NA            NA                  NA             NA
## 2           50.08      375.1583               291.4          0.835
## 3           50.10      242.7692               122.8          0.653
## 4              NA            NA                  NA             NA
## 5              NA      221.0525                  NA             NA
## 6              NA            NA                  NA             NA
##   lactateThresholdBpm lactateThresholdSpeed avgStrokes activeLengths avgSwolf
## 1                  NA                    NA       23.0            60       74
## 2                  NA                    NA         NA            NA       NA
## 3                  NA                    NA         NA            NA       NA
## 4                  NA                    NA         NA            NA       NA
## 5                  NA                    NA         NA            NA       NA
## 6                  NA                    NA       22.6            57       72
##   poolLength avgStrokeDistance avgSwimCadence maxSwimCadence maxFtp workoutId
## 1       5000               217             27             29     NA        NA
## 2         NA                NA             NA             NA     NA        NA
## 3         NA                NA             NA             NA     NA        NA
## 4         NA                NA             NA             NA     NA        NA
## 5         NA                NA             NA             NA     NA        NA
## 6       5000               221             27             30     NA        NA
##   decoDive parentId avgVerticalSpeed maxDepth avgDepth surfaceInterval
## 1       NA       NA               NA       NA       NA              NA
## 2       NA       NA               NA       NA       NA              NA
## 3       NA       NA               NA       NA       NA              NA
## 4       NA       NA               NA       NA       NA              NA
## 5       NA       NA               NA       NA       NA              NA
## 6       NA       NA               NA       NA       NA              NA
##   floorsDescended bottomTime
## 1              NA         NA
## 2              NA         NA
## 3              NA         NA
## 4              NA         NA
## 5              NA         NA
## 6              NA         NA
```

Chapter 3

Introduction to R

R is one of the most popular data science language along with Python and Julia

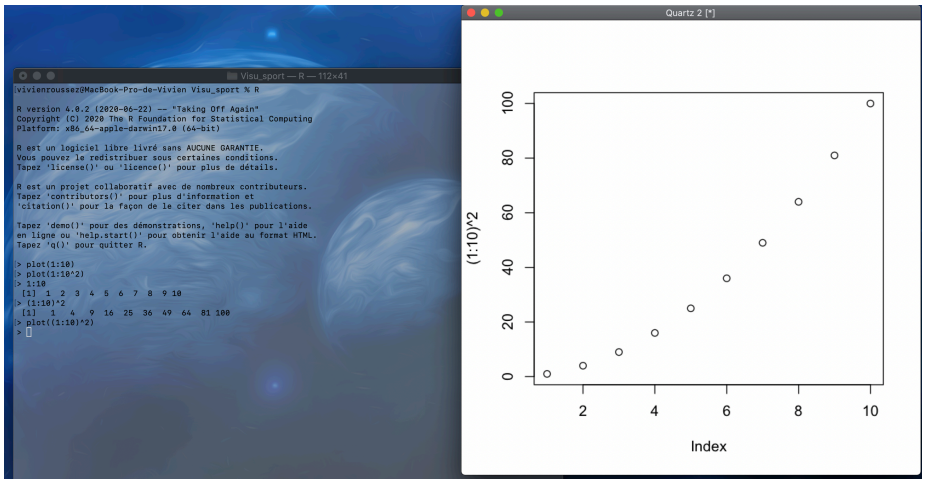
3.1 What is R

3.1.1 Description

R is an open source programming language initially dedicated to **statistics** and **data analysis**. It is the open-source version of the original S/S-plus language, developed by Bell labs a looong time ago. It was developed in the late 90's. Being open-source, the number of **packages** available is considerable, generating both completeness and confusion.

R is a functional programming language, meaning that **functions** are at the very core of its usage. It is not a object-oriented language although there are classes, but which are mainly hidden from the end-user.

THe basic R is a command line interface, pretty similar to the bash



It is an **interactive** language, meaning you can execute commands one after the other, no compilation is needed to execute a sequence of commands and you can try / adjust yourt code on the fly → very flexible (like IPython) Another very interesting characteristic of the language is that it is by design **vectorized**, meaning operations are executed at once on vectors, without explicit loops, which makes it very effective (as long as you don't loop…)

3.1.2 (Objective) comparison with Pyhton

What they have in common :

- Both language are open source and come with a wide set of capabilities and a community
- Interactivity
- Data science development environment (Jupyter)
- Several ways to achieve the same task

What differs :

- R is dedicated to data / python is a generic programming language
- Functional vs object oriented
- Analysis (R) vs final product (Py) orientation

Analysis Tool	Similar Superhero	Super Powers in Common
R 	Batman 	• Detective Work • Intelligence • Cunning • Usage of Tools • More Brain than Muscles
Python 	Superman 	• Muscle Power • Super Strength • Elegance • Wide Range • More Muscles than Brain

3.1.3 What can I do with R

R's core relies in data manipulation and statistical analysis. But the community made it grow in many directions

- Read data from multiple sources (excel, text files, databases, big data infrastructures…)
- Machine learning and deep learning
- Data visualization
- Communication
- Publications
- … Usages I probably have no idea about !!

3.1.4 Quick presentation of the ecosystem

The core functionalities are available with base R on CRAN. On top of that, you can install several IDEs, the most popular ones being Rstudio, jupyter or VSCode

For this training, we will use the R kernel of google collabs, but for many purposes, you'll have to use another IDE (shiny, markdown…). This kernel comes with basics packages AND the **tidyverse**

To add features to R, you'll have then to install **packages**. Generally, when facing a problem (eg : I have to implement a naive Bayes estimator), you google

it adding r at the beginning of the query and you'll get the name of the packages that allow you to do that. Then you can install and activate it.

```
# install.packages("e1071")
library(e1071)
```

Note : You can also call functions from an installed package without loading the whole package with `::`. You might prefer this solution in several cases :

- You use only one function from the package only once → maybe not necessary to load everything
- Function names can be common across packages (eg: `intersect`, `summarise`…) using `::` ensures you are using the function from the package you meant.
- Drawback : when not appearing at the beginning of the script, it can be unseen (for a new user) that the script requires such package to be installed

```
dplyr::summarise(iris,n_species=dplyr::n_distinct(Species))
```

```
##   n_species
## 1         3
```

To find more information about R and its functinalities / latest news :

- R bloggers
- tidyverse.org
- twitter : #rstat
- Rstudio website

Now you can use all functions of this packages !

3.2 Basic commands to know

- Where to find help :
  - Search engine to know how to do something
  - `help(lm)` or `?lm` to get help about a specific function (its inputs and output)
  - stackoverflow to debug
- List the objects in memory `ls()`
- What is the current directory `getwd()` ; change it `setwd()`
- Browse folders and files `dir()`
- Session information (loaded packages and so on) `sessionInfo()`
- Install and load packages : see above
- View the source code of a function : `lm`
- Create a new object and assign a value to it `<-`. display in the console by typing its name
- Commented lines, like in Python, start with a `#`

```
?lm
ls()
```

```
## [1] "dat"
```

```
sessionInfo()
```

```
## R version 4.0.2 (2020-06-22)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS Catalina 10.15.6
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRblas.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] e1071_1.7-3     forcats_0.5.0   stringr_1.4.0   dplyr_1.0.2
##  [5] purrr_0.3.4     readr_1.3.1     tidyr_1.1.2     tibble_3.0.3
##  [9] ggplot2_3.3.2   tidyverse_1.3.0
##
## loaded via a namespace (and not attached):
##  [1] tidyselect_1.1.0 xfun_0.17        haven_2.3.1      colorspace_1.4-1
##  [5] vctrs_0.3.4      generics_0.0.2   htmltools_0.5.0  yaml_2.2.1
##  [9] blob_1.2.1       rlang_0.4.7      pillar_1.4.6     glue_1.4.2
## [13] withr_2.2.0      DBI_1.1.0        dbplyr_1.4.4     modelr_0.1.8
## [17] readxl_1.3.1     lifecycle_0.2.0  munsell_0.5.0    gtable_0.3.0
## [21] cellranger_1.1.0 rvest_0.3.6      evaluate_0.14    knitr_1.29
## [25] class_7.3-17     fansi_0.4.1      broom_0.7.0      Rcpp_1.0.5
## [29] scales_1.1.1     backports_1.1.10 jsonlite_1.7.1   fs_1.5.0
## [33] hms_0.5.3        digest_0.6.25    stringi_1.5.3    bookdown_0.20
## [37] grid_4.0.2       cli_2.0.2        tools_4.0.2      magrittr_1.5
## [41] crayon_1.3.4     pkgconfig_2.0.3  ellipsis_0.3.1   xml2_1.3.2
## [45] reprex_0.3.0     lubridate_1.7.9  assertthat_0.2.1 rmarkdown_2.3
## [49] httr_1.4.2       rstudioapi_0.11  R6_2.4.1         compiler_4.0.2
```

```
getwd()
```

```
## [1] "/Users/vivienroussez/Documents/Datascience/Asigmo/DataExploration"
```

```
dir("/")
```

```
##  [1] "Applications" "bin"          "cores"        "dev"          "etc"
```

```
##  [6] "home"         "Library"      "opt"          "private"      "sbin"
## [11] "System"       "tmp"          "Users"        "usr"          "var"
## [16] "Volumes"
```

```
lm
```

```
## function (formula, data, subset, weights, na.action, method = "qr",
##     model = TRUE, x = FALSE, y = FALSE, qr = TRUE, singular.ok = TRUE,
##     contrasts = NULL, offset, ...)
## {
##     ret.x <- x
##     ret.y <- y
##     cl <- match.call()
##     mf <- match.call(expand.dots = FALSE)
##     m <- match(c("formula", "data", "subset", "weights", "na.action",
##         "offset"), names(mf), 0L)
##     mf <- mf[c(1L, m)]
##     mf$drop.unused.levels <- TRUE
##     mf[[1L]] <- quote(stats::model.frame)
##     mf <- eval(mf, parent.frame())
##     if (method == "model.frame")
##         return(mf)
##     else if (method != "qr")
##         warning(gettextf("method = '%s' is not supported. Using 'qr'",
##             method), domain = NA)
##     mt <- attr(mf, "terms")
##     y <- model.response(mf, "numeric")
##     w <- as.vector(model.weights(mf))
##     if (!is.null(w) && !is.numeric(w))
##         stop("'weights' must be a numeric vector")
##     offset <- model.offset(mf)
##     mlm <- is.matrix(y)
##     ny <- if (mlm)
##         nrow(y)
##     else length(y)
##     if (!is.null(offset)) {
##         if (!mlm)
##             offset <- as.vector(offset)
##         if (NROW(offset) != ny)
##             stop(gettextf("number of offsets is %d, should equal %d (number of obse
##                 NROW(offset), ny), domain = NA)
##     }
##     if (is.empty.model(mt)) {
##         x <- NULL
##         z <- list(coefficients = if (mlm) matrix(NA_real_, 0,
##             ncol(y)) else numeric(), residuals = y, fitted.values = 0 *
```

```
##             y, weights = w, rank = 0L, df.residual = if (!is.null(w)) sum(w !=
##             0) else ny)
##         if (!is.null(offset)) {
##             z$fitted.values <- offset
##             z$residuals <- y - offset
##         }
##     }
##     else {
##         x <- model.matrix(mt, mf, contrasts)
##         z <- if (is.null(w))
##             lm.fit(x, y, offset = offset, singular.ok = singular.ok,
##                 ...)
##         else lm.wfit(x, y, w, offset = offset, singular.ok = singular.ok,
##             ...)
##     }
##     class(z) <- c(if (mlm) "mlm", "lm")
##     z$na.action <- attr(mf, "na.action")
##     z$offset <- offset
##     z$contrasts <- attr(x, "contrasts")
##     z$xlevels <- .getXlevels(mt, mf)
##     z$call <- cl
##     z$terms <- mt
##     if (model)
##         z$model <- mf
##     if (ret.x)
##         z$x <- x
##     if (ret.y)
##         z$y <- y
##     if (!qr)
##         z$qr <- NULL
##     z
## }
## <bytecode: 0x7f86369af408>
## <environment: namespace:stats>
```

```
obj <- 3
obj
```

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

3.3 Data structures in R

As mentioned, R is a *functional* programming language, which means that you will always call… functions. And functions are defined by

- parameters : the inputs you have to provide the function so that it can

do what it's meant for
- the result : the output you get. Stricly speacking, the result of a function is unique (as opposed to procedures). Of course, depending on the **class** of the result, it may of course be composite

This chapter gives you some keys to understand and explore the results as they are provided by the functions.

3.3.1 Basic data structures

Before introducing the data structure, a short precision about *types*. Values are stored in data structures which partially depend on their type :

- Logical (`TRUE` or `FALSE`)
- Numerical (integer, continuous or complex)
- Character (strings or categories)

R recognizes the type of the value and modify it dynamically (no need to declare the type and it can e changed). To force R to coerce values to another type, you can use the functions `as.numeric`, `as.character`, `as.logical`.

Important note : `NA` stands for *not available* and is common to all type when a value is **missing**. You can have other missing values though for numerical variables :

- `Nan` (not a mumber) eg 0/0
- `Inf` (infinity) eg log(0)

Attention : `NULL` applies to objects (eg a matrix or a list) and not to values themselves

3.3.1.1 Vectors

Vectors are the basic data structure : it is a unidimensional collection of values *having the same type*. There are a lot of ways to generate vectors :

```
my_vect <- c(1,2,19,1) ; print(my_vect)
```

```
## [1]  1  2 19  1
```

```
my_vect <- 1:10 ; print(my_vect)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
my_vect <- seq(-15,100,.1) ; print(my_vect)
```

```
##    [1] -15.0 -14.9 -14.8 -14.7 -14.6 -14.5 -14.4 -14.3 -14.2 -14.1 -14.0 -13.9
##   [13] -13.8 -13.7 -13.6 -13.5 -13.4 -13.3 -13.2 -13.1 -13.0 -12.9 -12.8 -12.7
##   [25] -12.6 -12.5 -12.4 -12.3 -12.2 -12.1 -12.0 -11.9 -11.8 -11.7 -11.6 -11.5
##   [37] -11.4 -11.3 -11.2 -11.1 -11.0 -10.9 -10.8 -10.7 -10.6 -10.5 -10.4 -10.3
##   [49] -10.2 -10.1 -10.0  -9.9  -9.8  -9.7  -9.6  -9.5  -9.4  -9.3  -9.2  -9.1
```

```
##   [61] -9.0 -8.9 -8.8 -8.7 -8.6 -8.5 -8.4 -8.3 -8.2 -8.1 -8.0 -7.9
##   [73] -7.8 -7.7 -7.6 -7.5 -7.4 -7.3 -7.2 -7.1 -7.0 -6.9 -6.8 -6.7
##   [85] -6.6 -6.5 -6.4 -6.3 -6.2 -6.1 -6.0 -5.9 -5.8 -5.7 -5.6 -5.5
##   [97] -5.4 -5.3 -5.2 -5.1 -5.0 -4.9 -4.8 -4.7 -4.6 -4.5 -4.4 -4.3
##  [109] -4.2 -4.1 -4.0 -3.9 -3.8 -3.7 -3.6 -3.5 -3.4 -3.3 -3.2 -3.1
##  [121] -3.0 -2.9 -2.8 -2.7 -2.6 -2.5 -2.4 -2.3 -2.2 -2.1 -2.0 -1.9
##  [133] -1.8 -1.7 -1.6 -1.5 -1.4 -1.3 -1.2 -1.1 -1.0 -0.9 -0.8 -0.7
##  [145] -0.6 -0.5 -0.4 -0.3 -0.2 -0.1  0.0  0.1  0.2  0.3  0.4  0.5
##  [157]  0.6  0.7  0.8  0.9  1.0  1.1  1.2  1.3  1.4  1.5  1.6  1.7
##  [169]  1.8  1.9  2.0  2.1  2.2  2.3  2.4  2.5  2.6  2.7  2.8  2.9
##  [181]  3.0  3.1  3.2  3.3  3.4  3.5  3.6  3.7  3.8  3.9  4.0  4.1
##  [193]  4.2  4.3  4.4  4.5  4.6  4.7  4.8  4.9  5.0  5.1  5.2  5.3
##  [205]  5.4  5.5  5.6  5.7  5.8  5.9  6.0  6.1  6.2  6.3  6.4  6.5
##  [217]  6.6  6.7  6.8  6.9  7.0  7.1  7.2  7.3  7.4  7.5  7.6  7.7
##  [229]  7.8  7.9  8.0  8.1  8.2  8.3  8.4  8.5  8.6  8.7  8.8  8.9
##  [241]  9.0  9.1  9.2  9.3  9.4  9.5  9.6  9.7  9.8  9.9 10.0 10.1
##  [253] 10.2 10.3 10.4 10.5 10.6 10.7 10.8 10.9 11.0 11.1 11.2 11.3
##  [265] 11.4 11.5 11.6 11.7 11.8 11.9 12.0 12.1 12.2 12.3 12.4 12.5
##  [277] 12.6 12.7 12.8 12.9 13.0 13.1 13.2 13.3 13.4 13.5 13.6 13.7
##  [289] 13.8 13.9 14.0 14.1 14.2 14.3 14.4 14.5 14.6 14.7 14.8 14.9
##  [301] 15.0 15.1 15.2 15.3 15.4 15.5 15.6 15.7 15.8 15.9 16.0 16.1
##  [313] 16.2 16.3 16.4 16.5 16.6 16.7 16.8 16.9 17.0 17.1 17.2 17.3
##  [325] 17.4 17.5 17.6 17.7 17.8 17.9 18.0 18.1 18.2 18.3 18.4 18.5
##  [337] 18.6 18.7 18.8 18.9 19.0 19.1 19.2 19.3 19.4 19.5 19.6 19.7
##  [349] 19.8 19.9 20.0 20.1 20.2 20.3 20.4 20.5 20.6 20.7 20.8 20.9
##  [361] 21.0 21.1 21.2 21.3 21.4 21.5 21.6 21.7 21.8 21.9 22.0 22.1
##  [373] 22.2 22.3 22.4 22.5 22.6 22.7 22.8 22.9 23.0 23.1 23.2 23.3
##  [385] 23.4 23.5 23.6 23.7 23.8 23.9 24.0 24.1 24.2 24.3 24.4 24.5
##  [397] 24.6 24.7 24.8 24.9 25.0 25.1 25.2 25.3 25.4 25.5 25.6 25.7
##  [409] 25.8 25.9 26.0 26.1 26.2 26.3 26.4 26.5 26.6 26.7 26.8 26.9
##  [421] 27.0 27.1 27.2 27.3 27.4 27.5 27.6 27.7 27.8 27.9 28.0 28.1
##  [433] 28.2 28.3 28.4 28.5 28.6 28.7 28.8 28.9 29.0 29.1 29.2 29.3
##  [445] 29.4 29.5 29.6 29.7 29.8 29.9 30.0 30.1 30.2 30.3 30.4 30.5
##  [457] 30.6 30.7 30.8 30.9 31.0 31.1 31.2 31.3 31.4 31.5 31.6 31.7
##  [469] 31.8 31.9 32.0 32.1 32.2 32.3 32.4 32.5 32.6 32.7 32.8 32.9
##  [481] 33.0 33.1 33.2 33.3 33.4 33.5 33.6 33.7 33.8 33.9 34.0 34.1
##  [493] 34.2 34.3 34.4 34.5 34.6 34.7 34.8 34.9 35.0 35.1 35.2 35.3
##  [505] 35.4 35.5 35.6 35.7 35.8 35.9 36.0 36.1 36.2 36.3 36.4 36.5
##  [517] 36.6 36.7 36.8 36.9 37.0 37.1 37.2 37.3 37.4 37.5 37.6 37.7
##  [529] 37.8 37.9 38.0 38.1 38.2 38.3 38.4 38.5 38.6 38.7 38.8 38.9
##  [541] 39.0 39.1 39.2 39.3 39.4 39.5 39.6 39.7 39.8 39.9 40.0 40.1
##  [553] 40.2 40.3 40.4 40.5 40.6 40.7 40.8 40.9 41.0 41.1 41.2 41.3
##  [565] 41.4 41.5 41.6 41.7 41.8 41.9 42.0 42.1 42.2 42.3 42.4 42.5
##  [577] 42.6 42.7 42.8 42.9 43.0 43.1 43.2 43.3 43.4 43.5 43.6 43.7
##  [589] 43.8 43.9 44.0 44.1 44.2 44.3 44.4 44.5 44.6 44.7 44.8 44.9
##  [601] 45.0 45.1 45.2 45.3 45.4 45.5 45.6 45.7 45.8 45.9 46.0 46.1
```

```
##  [613] 46.2 46.3 46.4 46.5 46.6 46.7 46.8 46.9 47.0 47.1 47.2 47.3
##  [625] 47.4 47.5 47.6 47.7 47.8 47.9 48.0 48.1 48.2 48.3 48.4 48.5
##  [637] 48.6 48.7 48.8 48.9 49.0 49.1 49.2 49.3 49.4 49.5 49.6 49.7
##  [649] 49.8 49.9 50.0 50.1 50.2 50.3 50.4 50.5 50.6 50.7 50.8 50.9
##  [661] 51.0 51.1 51.2 51.3 51.4 51.5 51.6 51.7 51.8 51.9 52.0 52.1
##  [673] 52.2 52.3 52.4 52.5 52.6 52.7 52.8 52.9 53.0 53.1 53.2 53.3
##  [685] 53.4 53.5 53.6 53.7 53.8 53.9 54.0 54.1 54.2 54.3 54.4 54.5
##  [697] 54.6 54.7 54.8 54.9 55.0 55.1 55.2 55.3 55.4 55.5 55.6 55.7
##  [709] 55.8 55.9 56.0 56.1 56.2 56.3 56.4 56.5 56.6 56.7 56.8 56.9
##  [721] 57.0 57.1 57.2 57.3 57.4 57.5 57.6 57.7 57.8 57.9 58.0 58.1
##  [733] 58.2 58.3 58.4 58.5 58.6 58.7 58.8 58.9 59.0 59.1 59.2 59.3
##  [745] 59.4 59.5 59.6 59.7 59.8 59.9 60.0 60.1 60.2 60.3 60.4 60.5
##  [757] 60.6 60.7 60.8 60.9 61.0 61.1 61.2 61.3 61.4 61.5 61.6 61.7
##  [769] 61.8 61.9 62.0 62.1 62.2 62.3 62.4 62.5 62.6 62.7 62.8 62.9
##  [781] 63.0 63.1 63.2 63.3 63.4 63.5 63.6 63.7 63.8 63.9 64.0 64.1
##  [793] 64.2 64.3 64.4 64.5 64.6 64.7 64.8 64.9 65.0 65.1 65.2 65.3
##  [805] 65.4 65.5 65.6 65.7 65.8 65.9 66.0 66.1 66.2 66.3 66.4 66.5
##  [817] 66.6 66.7 66.8 66.9 67.0 67.1 67.2 67.3 67.4 67.5 67.6 67.7
##  [829] 67.8 67.9 68.0 68.1 68.2 68.3 68.4 68.5 68.6 68.7 68.8 68.9
##  [841] 69.0 69.1 69.2 69.3 69.4 69.5 69.6 69.7 69.8 69.9 70.0 70.1
##  [853] 70.2 70.3 70.4 70.5 70.6 70.7 70.8 70.9 71.0 71.1 71.2 71.3
##  [865] 71.4 71.5 71.6 71.7 71.8 71.9 72.0 72.1 72.2 72.3 72.4 72.5
##  [877] 72.6 72.7 72.8 72.9 73.0 73.1 73.2 73.3 73.4 73.5 73.6 73.7
##  [889] 73.8 73.9 74.0 74.1 74.2 74.3 74.4 74.5 74.6 74.7 74.8 74.9
##  [901] 75.0 75.1 75.2 75.3 75.4 75.5 75.6 75.7 75.8 75.9 76.0 76.1
##  [913] 76.2 76.3 76.4 76.5 76.6 76.7 76.8 76.9 77.0 77.1 77.2 77.3
##  [925] 77.4 77.5 77.6 77.7 77.8 77.9 78.0 78.1 78.2 78.3 78.4 78.5
##  [937] 78.6 78.7 78.8 78.9 79.0 79.1 79.2 79.3 79.4 79.5 79.6 79.7
##  [949] 79.8 79.9 80.0 80.1 80.2 80.3 80.4 80.5 80.6 80.7 80.8 80.9
##  [961] 81.0 81.1 81.2 81.3 81.4 81.5 81.6 81.7 81.8 81.9 82.0 82.1
##  [973] 82.2 82.3 82.4 82.5 82.6 82.7 82.8 82.9 83.0 83.1 83.2 83.3
##  [985] 83.4 83.5 83.6 83.7 83.8 83.9 84.0 84.1 84.2 84.3 84.4 84.5
##  [997] 84.6 84.7 84.8 84.9 85.0 85.1 85.2 85.3 85.4 85.5 85.6 85.7
## [1009] 85.8 85.9 86.0 86.1 86.2 86.3 86.4 86.5 86.6 86.7 86.8 86.9
## [1021] 87.0 87.1 87.2 87.3 87.4 87.5 87.6 87.7 87.8 87.9 88.0 88.1
## [1033] 88.2 88.3 88.4 88.5 88.6 88.7 88.8 88.9 89.0 89.1 89.2 89.3
## [1045] 89.4 89.5 89.6 89.7 89.8 89.9 90.0 90.1 90.2 90.3 90.4 90.5
## [1057] 90.6 90.7 90.8 90.9 91.0 91.1 91.2 91.3 91.4 91.5 91.6 91.7
## [1069] 91.8 91.9 92.0 92.1 92.2 92.3 92.4 92.5 92.6 92.7 92.8 92.9
## [1081] 93.0 93.1 93.2 93.3 93.4 93.5 93.6 93.7 93.8 93.9 94.0 94.1
## [1093] 94.2 94.3 94.4 94.5 94.6 94.7 94.8 94.9 95.0 95.1 95.2 95.3
## [1105] 95.4 95.5 95.6 95.7 95.8 95.9 96.0 96.1 96.2 96.3 96.4 96.5
## [1117] 96.6 96.7 96.8 96.9 97.0 97.1 97.2 97.3 97.4 97.5 97.6 97.7
## [1129] 97.8 97.9 98.0 98.1 98.2 98.3 98.4 98.5 98.6 98.7 98.8 98.9
## [1141] 99.0 99.1 99.2 99.3 99.4 99.5 99.6 99.7 99.8 99.9 100.0
```

```
my_vect <- rnorm(100) ; print(my_vect)
```

```
##   [1] -1.21004842 -1.06121248  0.71634673  1.47637602  0.48728132 -1.40233045
##   [7] -0.01020396 -1.00491601 -1.13058465 -0.51105332 -3.04697783 -0.41697855
##  [13]  1.14776250 -0.06190428 -0.61406102  1.08143552  2.37724254  2.27646272
##  [19]  0.91724004 -0.44129808  0.03188855 -0.89633876  0.18671548 -0.23189549
##  [25]  1.67617991 -0.27757116  0.26018862  2.00163360 -1.41111960 -0.55119739
##  [31]  1.42648442  0.73438772  0.86664661 -0.89074131  0.32450045  0.86402894
##  [37] -1.85744581  0.82117312  0.20926134 -1.02316778 -0.69767865 -0.77886117
##  [43] -0.42240714  0.41656734  0.51914181 -0.20057167  0.19494615  0.01393820
##  [49] -0.99481747  0.54007011  0.49309055 -0.79606589 -2.36025571  0.48564480
##  [55]  0.29072507 -0.38146633  0.56337763 -0.48490022  1.06052814  0.24571570
##  [61]  0.46890216 -0.24452292 -0.92088724 -0.86546828 -0.18666685  0.78926829
##  [67] -0.78086945  0.46444770 -0.76284691  0.61650581  0.59103079 -0.17137042
##  [73]  0.12108668 -1.63122288 -0.80778850  1.06061406 -0.95744478  0.36101337
##  [79]  0.05042920 -0.97059999 -1.80390869  0.15294386  0.32179016  1.11523338
##  [85]  0.33365395 -1.17738912  1.46300597 -0.29402040 -0.06691599 -0.45963294
##  [91]  0.46536078  1.06914660 -1.00272811 -1.26934158 -0.55375181 -0.09793410
##  [97] -2.65666912 -0.76039332  0.33721873 -0.39925402
```

```
my_vect <- sample(letters,100,replace = T) ; print(my_vect)
```

```
##   [1] "u" "x" "a" "g" "m" "f" "a" "y" "p" "i" "n" "r" "i" "m" "d" "u" "c" "v"
##  [19] "p" "k" "e" "d" "f" "n" "i" "h" "d" "h" "z" "f" "t" "s" "g" "t" "b" "m"
##  [37] "a" "f" "a" "s" "z" "c" "u" "g" "s" "t" "t" "l" "g" "j" "g" "i" "v" "o"
##  [55] "k" "n" "l" "l" "k" "d" "t" "m" "v" "p" "t" "z" "q" "t" "o" "d" "v" "j"
##  [73] "l" "t" "h" "u" "m" "f" "q" "b" "c" "q" "s" "w" "j" "j" "s" "m" "o" "z"
##  [91] "h" "n" "y" "d" "w" "f" "r" "z" "h" "e"
```

You can access vector values with integer indexes (that are vector themselves).
Note : unlike Python, the indexes start with the value 1, not 0 !

```
my_vect[4]
```

```
## [1] "g"
```

```
my_vect[1:4]
```

```
## [1] "u" "x" "a" "g"
```

A vector can be *named* meaning that each element has a name through which it can be accessed.

```
my_vect <- 1:10
names(my_vect) <- letters[1:10] ; print(my_vect)
```

```
##  a  b  c  d  e  f  g  h  i  j
##  1  2  3  4  5  6  7  8  9 10
```

```
my_vect["b"]
```

```
## b
## 2
```

Did you notice you can assign values to a vector's attribute ? :D

3.3.1.2 Matrices and arrays

Matrices are a 2-dimensional collection of values *having the same type*. An array is an extension of matrices for more than 2 dimensions.

```
mat <- matrix(1,ncol=10,nrow = 15) ; print(mat)
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
##  [1,]    1    1    1    1    1    1    1    1    1     1
##  [2,]    1    1    1    1    1    1    1    1    1     1
##  [3,]    1    1    1    1    1    1    1    1    1     1
##  [4,]    1    1    1    1    1    1    1    1    1     1
##  [5,]    1    1    1    1    1    1    1    1    1     1
##  [6,]    1    1    1    1    1    1    1    1    1     1
##  [7,]    1    1    1    1    1    1    1    1    1     1
##  [8,]    1    1    1    1    1    1    1    1    1     1
##  [9,]    1    1    1    1    1    1    1    1    1     1
## [10,]    1    1    1    1    1    1    1    1    1     1
## [11,]    1    1    1    1    1    1    1    1    1     1
## [12,]    1    1    1    1    1    1    1    1    1     1
## [13,]    1    1    1    1    1    1    1    1    1     1
## [14,]    1    1    1    1    1    1    1    1    1     1
## [15,]    1    1    1    1    1    1    1    1    1     1
```

```
mat <- matrix(1:5,ncol=5,nrow=7) ; print(mat)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    3    5    2    4
## [2,]    2    4    1    3    5
## [3,]    3    5    2    4    1
## [4,]    4    1    3    5    2
## [5,]    5    2    4    1    3
## [6,]    1    3    5    2    4
## [7,]    2    4    1    3    5
```

```
arr <- array(1:10,dim = c(10,2,3)) ; print(arr)
```

```
## , , 1
##
##       [,1] [,2]
##  [1,]    1    1
```

```
##  [2,]    2    2
##  [3,]    3    3
##  [4,]    4    4
##  [5,]    5    5
##  [6,]    6    6
##  [7,]    7    7
##  [8,]    8    8
##  [9,]    9    9
## [10,]   10   10
##
## , , 2
##
##       [,1] [,2]
##  [1,]    1    1
##  [2,]    2    2
##  [3,]    3    3
##  [4,]    4    4
##  [5,]    5    5
##  [6,]    6    6
##  [7,]    7    7
##  [8,]    8    8
##  [9,]    9    9
## [10,]   10   10
##
## , , 3
##
##       [,1] [,2]
##  [1,]    1    1
##  [2,]    2    2
##  [3,]    3    3
##  [4,]    4    4
##  [5,]    5    5
##  [6,]    6    6
##  [7,]    7    7
##  [8,]    8    8
##  [9,]    9    9
## [10,]   10   10
```

```
matrix(rnorm(9),3,3)
m1 <- matrix(1,2,3)
m2 <- matrix(1,3,2)
m1*m2
```

3.3.1.3 Lists

Lists are a very versatile and convenient class that allows you to store heterogeneous values and data structures

```
my_list <- list("A",1,LETTERS[1:10],matrix(1,3,3)) ; my_list
```

```
## [[1]]
## [1] "A"
##
## [[2]]
## [1] 1
##
## [[3]]
##  [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J"
##
## [[4]]
##      [,1] [,2] [,3]
## [1,]    1    1    1
## [2,]    1    1    1
## [3,]    1    1    1
```

Like with vectors, list elements can be accessed via their index or their name. If a list has been named, you have something very similar to python dictionaries. In case the list is named, you can also access its elements via the `$` operator.

```
names(my_list) <- paste0("thing",1:length(my_list))
my_list[1]
```

```
## $thing1
## [1] "A"
```

```
my_list["thing1"]
```

```
## $thing1
## [1] "A"
```

```
my_list$thing3
```

```
##  [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J"
```

```
# my_list[2]*10
```

3.3.1.4 Dataframes

A Dataframe is the most common data representation (think of an excel spreadsheet): it is made out of columns and rows like a matrix, but the columns can have different types. In R, Dataframes are natives (no need to install another package). They are basically a list of vectors that have the same length.

Let's have a look at Fisher's iris dataframe (included in base R for demonstration purposes)

```
head(iris)
```

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

to explore the content of a dataframe, you can of course print it, but if you want amore detailed overview of it, you can use the `str` or the `glimpse` functions

```
str(iris)
```

```
## 'data.frame': 150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
glimpse(iris)
```

```
## Rows: 150
## Columns: 5
## $ Sepal.Length <dbl> 5.1, 4.9, 4.7, 4.6, 5.0, 5.4, 4.6, 5.0, 4.4, 4.9, 5.4,...
## $ Sepal.Width  <dbl> 3.5, 3.0, 3.2, 3.1, 3.6, 3.9, 3.4, 3.4, 2.9, 3.1, 3.7,...
## $ Petal.Length <dbl> 1.4, 1.4, 1.3, 1.5, 1.4, 1.7, 1.4, 1.5, 1.4, 1.5, 1.5,...
## $ Petal.Width  <dbl> 0.2, 0.2, 0.2, 0.2, 0.2, 0.4, 0.3, 0.2, 0.2, 0.1, 0.2,...
## $ Species      <fct> setosa, setosa, setosa, setosa, setosa, setosa, setosa...
```

In general, `str` (for structure) is a very powerful function to explore the content of a data structure (see next part). To explore it further, you can use the following functions

```
names(iris)
```

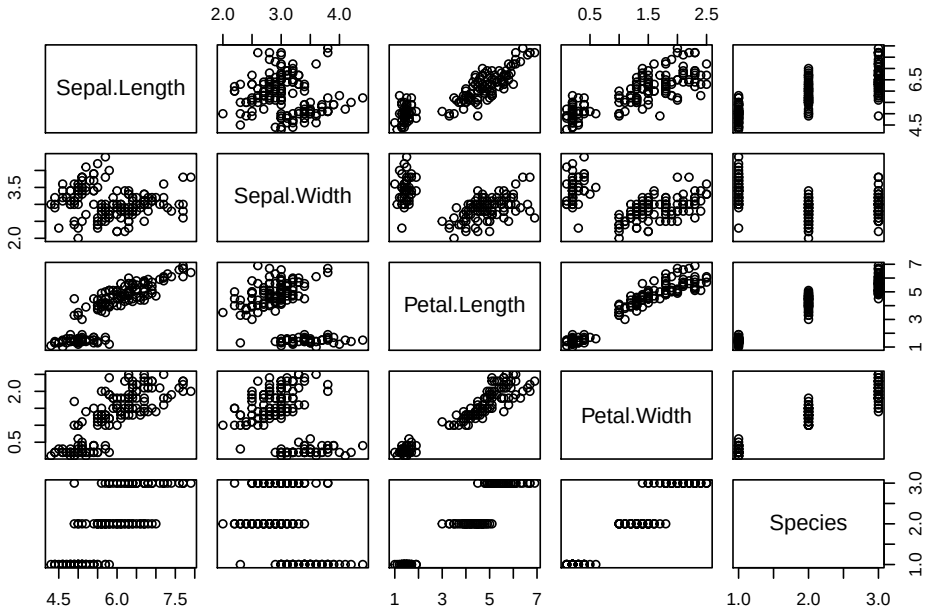
```
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"
```

```
summary(iris)
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
```

```
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
##        Species
##  setosa    :50
##  versicolor:50
##  virginica :50
##
##
##
```

```
plot(iris) # don't do that with too big data of course !
```



3.3.1.5 Functions

As mentioned before, R is a functional programming langueage and you can of course create your own functions (which can be afterwards integrated in a package). To cfeate a function, the syntax is such :

```
square <- function(xx=2) # 2 is the default value (not mandatory)
{
  res <- xx^2
  return(res)
}
# Shorthand
# square <- function(xx) xx^2
# Use it
square()
```

```
## [1] 4
```

```
square(5)
```

```
## [1] 25
```

3.3.1.6 Exercices

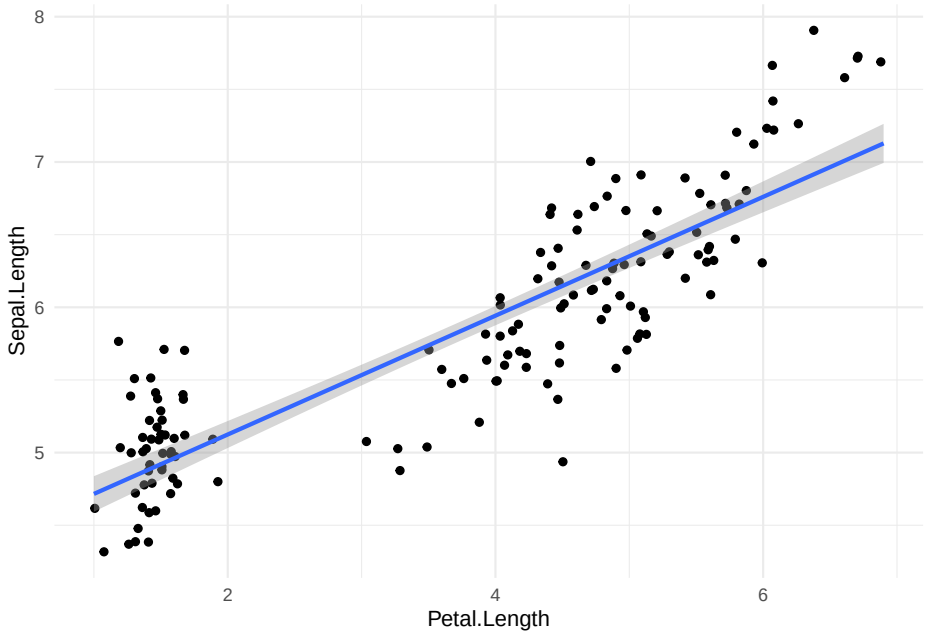
- Create a vector mixing both numbers and strings : what happens ?
- Create a vector containing the values "fellow 1" to fellow 15". Hint : be lazy and use the `paste()` function
- Replace the value "fellow 5" with "best fellow"
- Create a matrix (3,3) of random numbers drawn from a gaussian distribution.
- Create two numerical matrices of size resp (2,3) and (3,2) filled with 1s and compute their product
- From the previous list, the second element is a number ; multiply this number by 10 accessing it via its index
- Create a new list containing the previous list and some other random elements

3.3.2 Explore a new data structure (or object)

You will often face new data structures resulting from new functions, and they will be more complicated than the ones we've just covered.

Let us take the example of the linear regression (which we will cover in section 7)

```
library(ggplot2)
ggplot(iris,aes(Petal.Length,Sepal.Length)) + geom_jitter() +
  geom_smooth(method="lm") +
  theme_minimal()
```



Spoiler alert : the regression aims to find α and β such that an explained variable y can be expressed as $y = \alpha \cdot x + \beta$ where x is an explanatory variable. In R, to find the values of α and β, you will use the `lm`function. So let's fit this model and print the result

```
reg <- lm(Petal.Length ~ Sepal.Length,data=iris)
reg
```

```
##
## Call:
## lm(formula = Petal.Length ~ Sepal.Length, data = iris)
##
## Coefficients:
##  (Intercept)  Sepal.Length
##       -7.101         1.858
```

Ok, that's really minimal information... Let's try to dig into this `reg` object to find more.

```
names(reg)
```

```
##  [1] "coefficients"  "residuals"     "effects"       "rank"
##  [5] "fitted.values" "assign"        "qr"            "df.residual"
##  [9] "xlevels"       "call"          "terms"         "model"
```

```
str(reg)
```

```
## List of 12
```

```
##  $ coefficients : Named num [1:2] -7.1 1.86
##   ..- attr(*, "names")= chr [1:2] "(Intercept)" "Sepal.Length"
##  $ residuals    : Named num [1:150] -0.9766 -0.6049 -0.3332 0.0527 -0.7907 ...
##   ..- attr(*, "names")= chr [1:150] "1" "2" "3" "4" ...
##  $ effects      : Named num [1:150] -46.026 18.785 -0.207 0.184 -0.679 ...
##   ..- attr(*, "names")= chr [1:150] "(Intercept)" "Sepal.Length" "" "" ...
##  $ rank         : int 2
##  $ fitted.values: Named num [1:150] 2.38 2 1.63 1.45 2.19 ...
##   ..- attr(*, "names")= chr [1:150] "1" "2" "3" "4" ...
##  $ assign       : int [1:2] 0 1
##  $ qr           :List of 5
##   ..$ qr   : num [1:150, 1:2] -12.2474 0.0816 0.0816 0.0816 0.0816 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:150] "1" "2" "3" "4" ...
##   .. .. ..$ : chr [1:2] "(Intercept)" "Sepal.Length"
##   .. ..- attr(*, "assign")= int [1:2] 0 1
##   ..$ qraux: num [1:2] 1.08 1.09
##   ..$ pivot: int [1:2] 1 2
##   ..$ tol  : num 1e-07
##   ..$ rank : int 2
##   ..- attr(*, "class")= chr "qr"
##  $ df.residual  : int 148
##  $ xlevels      : Named list()
##  $ call         : language lm(formula = Petal.Length ~ Sepal.Length, data = iris)
##  $ terms        :Classes 'terms', 'formula'  language Petal.Length ~ Sepal.Length
##   .. ..- attr(*, "variables")= language list(Petal.Length, Sepal.Length)
##   .. ..- attr(*, "factors")= int [1:2, 1] 0 1
##   .. .. ..- attr(*, "dimnames")=List of 2
##   .. .. .. ..$ : chr [1:2] "Petal.Length" "Sepal.Length"
##   .. .. .. ..$ : chr "Sepal.Length"
##   .. ..- attr(*, "term.labels")= chr "Sepal.Length"
##   .. ..- attr(*, "order")= int 1
##   .. ..- attr(*, "intercept")= int 1
##   .. ..- attr(*, "response")= int 1
##   .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
##   .. ..- attr(*, "predvars")= language list(Petal.Length, Sepal.Length)
##   .. ..- attr(*, "dataClasses")= Named chr [1:2] "numeric" "numeric"
##   .. .. ..- attr(*, "names")= chr [1:2] "Petal.Length" "Sepal.Length"
##  $ model        :'data.frame': 150 obs. of  2 variables:
##   ..$ Petal.Length: num [1:150] 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##   ..$ Sepal.Length: num [1:150] 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##   ..- attr(*, "terms")=Classes 'terms', 'formula'  language Petal.Length ~ Sepal.Length
##   .. .. ..- attr(*, "variables")= language list(Petal.Length, Sepal.Length)
##   .. .. ..- attr(*, "factors")= int [1:2, 1] 0 1
##   .. .. .. ..- attr(*, "dimnames")=List of 2
##   .. .. .. .. ..$ : chr [1:2] "Petal.Length" "Sepal.Length"
```

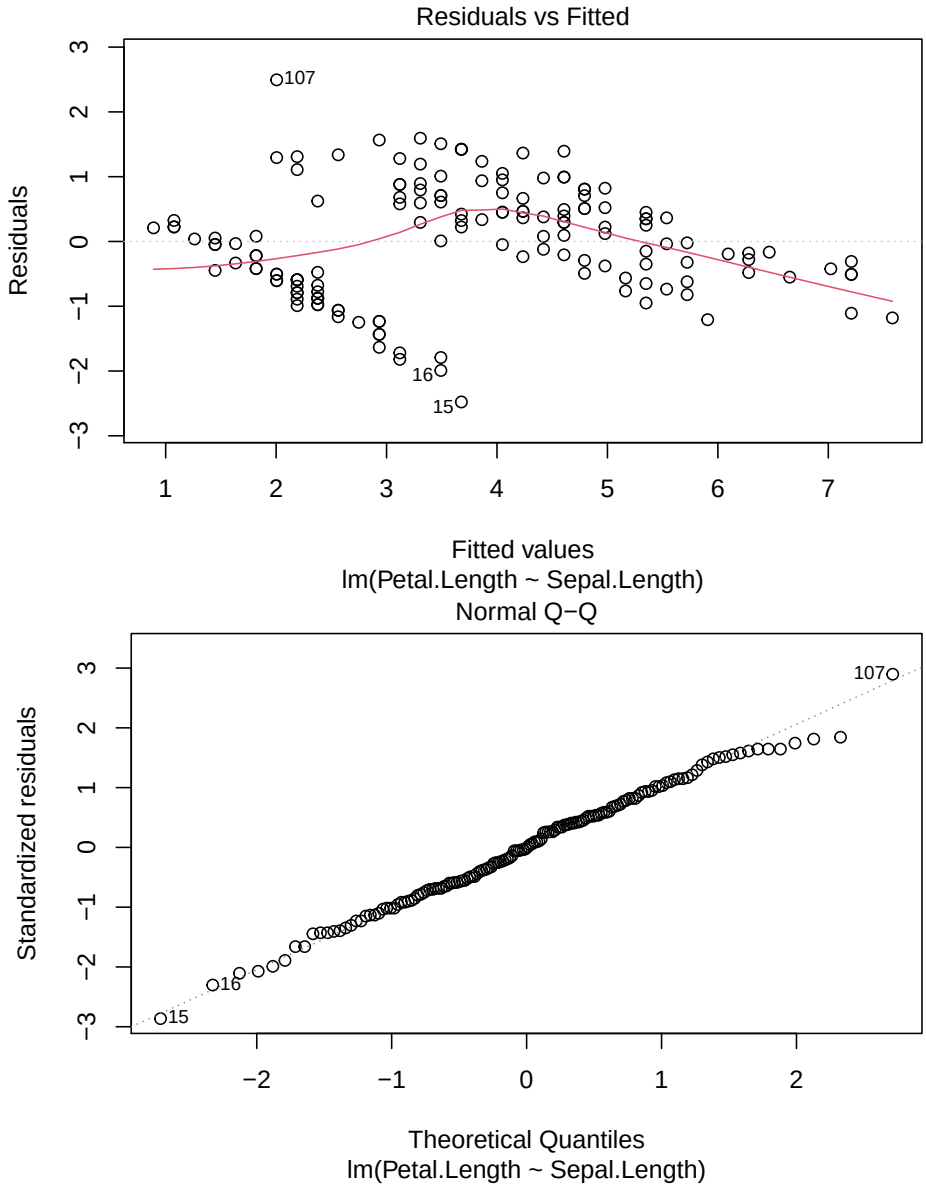
```
##   .. .. .. .. ..$ : chr "Sepal.Length"
##   .. .. ..- attr(*, "term.labels")= chr "Sepal.Length"
##   .. .. ..- attr(*, "order")= int 1
##   .. .. ..- attr(*, "intercept")= int 1
##   .. .. ..- attr(*, "response")= int 1
##   .. .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
##   .. .. ..- attr(*, "predvars")= language list(Petal.Length, Sepal.Length)
##   .. .. ..- attr(*, "dataClasses")= Named chr [1:2] "numeric" "numeric"
##   .. .. .. ..- attr(*, "names")= chr [1:2] "Petal.Length" "Sepal.Length"
##  - attr(*, "class")= chr "lm"
```

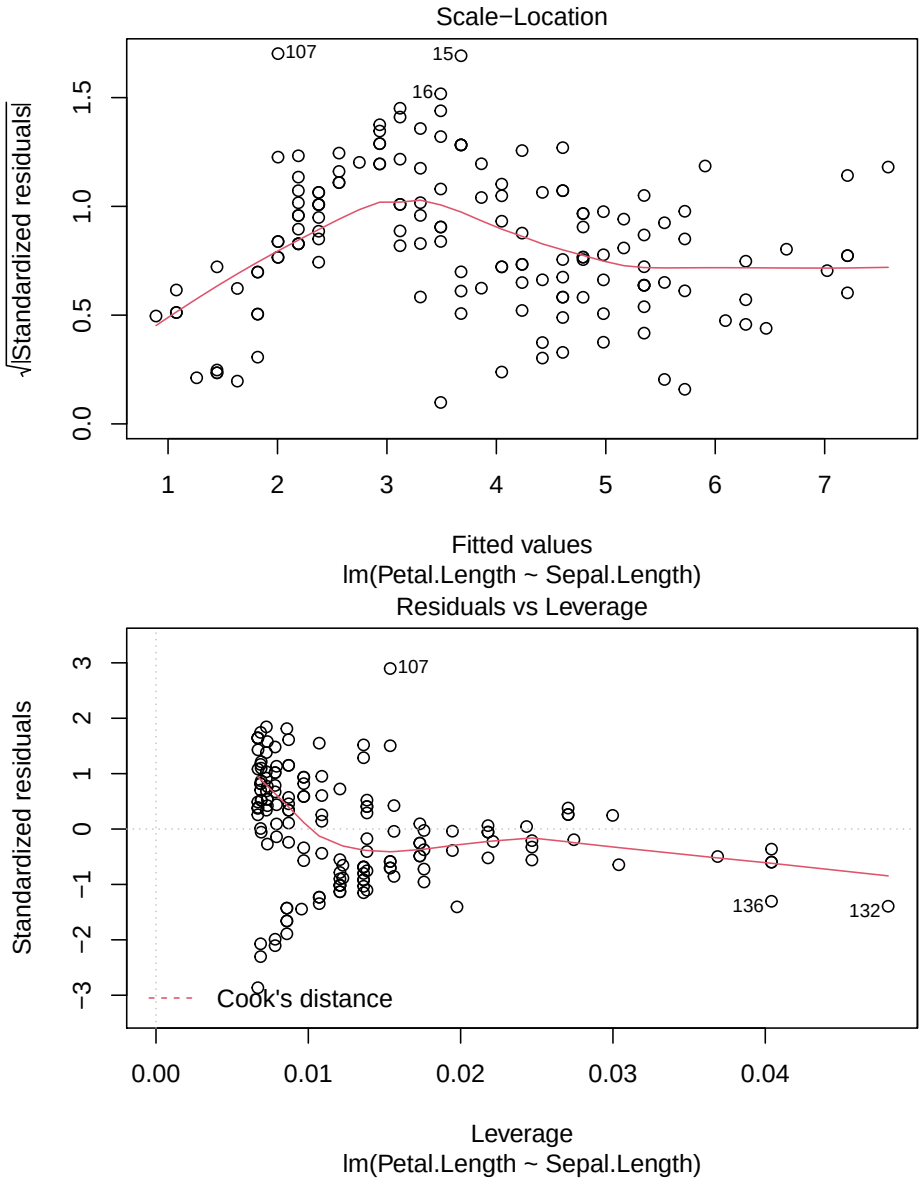
That's more interesting ! It seems that I can get more, including raw data, residuals, coefficients, degrees of freedom... And in general, you can apply standard functions on it as well

```
summary(reg)
```

```
##
## Call:
## lm(formula = Petal.Length ~ Sepal.Length, data = iris)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.47747 -0.59072 -0.00668  0.60484  2.49512
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -7.10144    0.50666  -14.02   <2e-16 ***
## Sepal.Length  1.85843    0.08586   21.65   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8678 on 148 degrees of freedom
## Multiple R-squared:   0.76,Adjusted R-squared:  0.7583
## F-statistic: 468.6 on 1 and 148 DF,  p-value: < 2.2e-16
```

```
plot(reg)
```


Residuals vs Fitted
Residuals
Fitted values
lm(Petal.Length ~ Sepal.Length)
Normal Q−Q
Standardized residuals
Theoretical Quantiles
lm(Petal.Length ~ Sepal.Length)
107
16
15



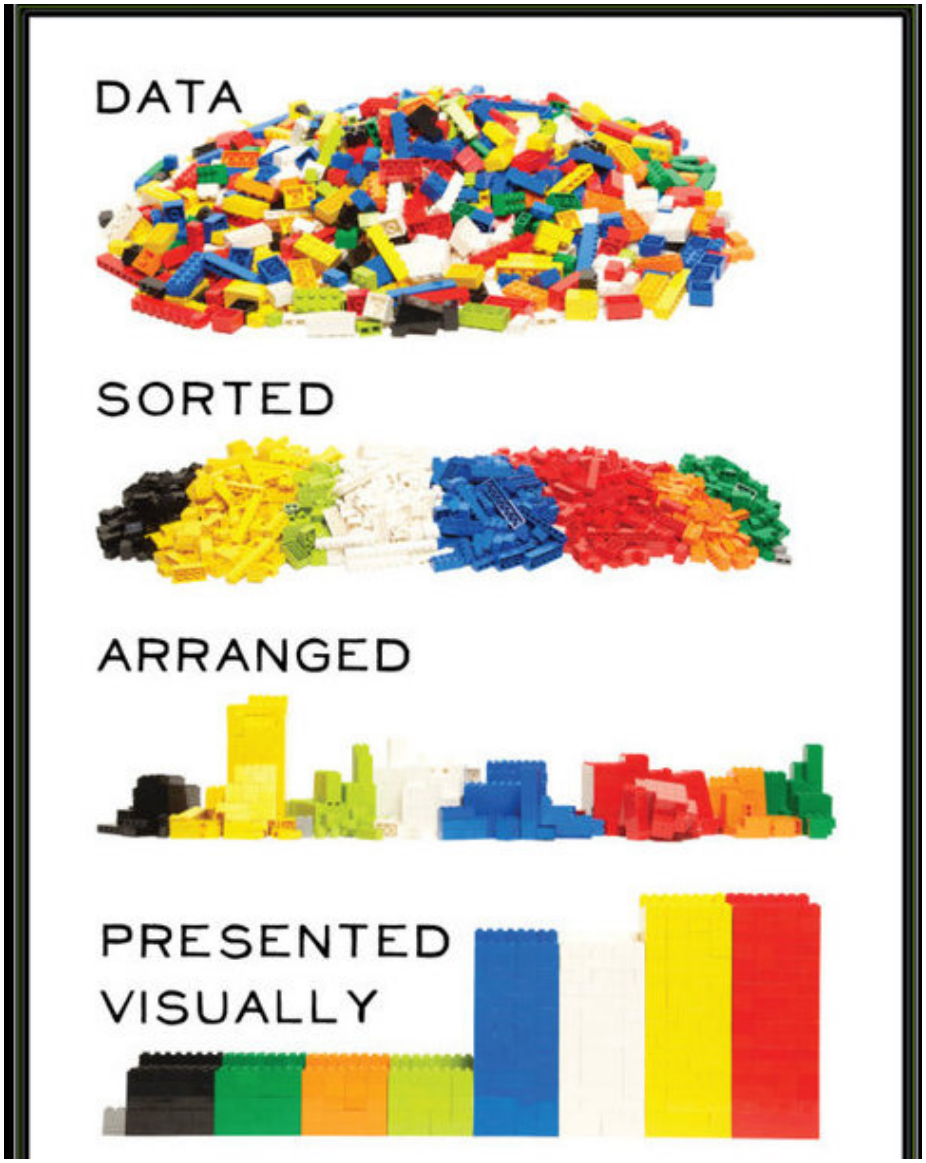
Ok, that's it, I have almost all that I wanted ! We'll cover the rest at the end of the week !

3.3.2.1 Exercise

On the `iris` dataset, use the `kmeans` function to cluster the flowers with respect to Sepal.Length and Petal.Length and try to find your way in the resulting object.

Chapter 4

Data manipulation



In order to manipulate and wrangle data, there are (at least) 3 frameworks available :

- Base R (not covered) : similar to Pandas
- Tidyverse/dplyr : high level interface
- `data.table` : less friendly user interface but amazingly optimized

We'll cover the tidyverse approach as it provides a very nice and coherent framework for more than data manipulation. The "tidy" comes an original paper from Hadley Wickham which sets those common sense principles

- Each variable must have its own column.
- Each observation must have its own row.
- Each value must have its own cell.

You can check further details and visualization on the R for data science online book

Those principles have been largely adopted by the community (powered by Rstudio) and created a full parallel dialect in R for almost all data science tasks : the **tidyverse** which offers a coherent set of features that are, in addition, often nicely optimized (written in C++)

4.1 Import

4.1.1 Text files

You can either use the basic `read.table` and related functions (eg `read.csv`) which work pretty fine. If the file is large, you might consider tools from other package such as read_csv and others from the `readr` package (part of the tidyverse)

```
dat <- read.csv("Data/Sports/Activities.csv",header = T)
head(dat)
```

```
##   activityId       uuidMsb       uuidLsb                name activityType
## 1 5570974040  3.695224e+18 -7.098507e+18          En piscine lap_swimming
## 2 5566524321 -5.212790e+18 -8.140623e+18     Vienna Cyclisme      cycling
## 3 5561266034  1.271356e+18 -5.522844e+18 Korneuburg Cyclisme      cycling
## 4 5555881653 -2.923938e+18 -4.676832e+18       Vienna Course      running
## 5 5551811953  7.859043e+18 -7.909532e+18      Zwift - London virtual_ride
## 6 5551052200 -9.554086e+17 -6.866435e+18          En piscine lap_swimming
##   userProfileId timeZoneId beginTimestamp eventTypeId   rule sportType
## 1       1141258        124   1.600700e+12           9 public   GENERIC
## 2       1141258        124   1.600602e+12           9 public   CYCLING
## 3       1141258        124   1.600523e+12           9 public   CYCLING
## 4       1141258        124   1.600439e+12           9 public   RUNNING
## 5       1141258        124   1.600362e+12           9 public   GENERIC
## 6       1141258        124   1.600353e+12           9 public   GENERIC
```

```
##   startTimeGmt startTimeLocal duration distance elevationGain elevationLoss
## 1 1.600700e+12   1.600708e+12  3640221   300000            NA            NA
## 2 1.600602e+12   1.600609e+12 15064570 12293220        200100        196800
## 3 1.600523e+12   1.600530e+12 10382433  8360927         74500         74200
## 4 1.600439e+12   1.600447e+12  5393830  1792890         42600         42500
## 5 1.600362e+12   1.600369e+12  3626825  3492347         16000             0
## 6 1.600353e+12   1.600360e+12  3359145   285000            NA            NA
##   avgSpeed maxSpeed avgHr maxHr  calories startLongitude startLatitude
## 1   0.0986   0.1122    NA    NA  2547.532            NA            NA
## 2   0.8160   1.9315   144   176 16345.268       16.31524      48.20915
## 3   0.8053   1.9557   117   161  8744.572       16.33986      48.34994
## 4   0.3324   1.2475   155   176  4923.274       16.31587      48.21432
## 5   0.9629   1.6364   138   163  3083.855        0.00000       0.00000
## 6   0.1011   0.3524    NA    NA  2505.632            NA            NA
##   aerobicTrainingEffect avgFractionalCadence maxFractionalCadence
## 1                    NA               0.0000                    0
## 2                   3.5               0.0000                    0
## 3                   2.4               0.0000                    0
## 4                   3.0               0.1875                    0
## 5                   0.0               0.0000                    0
## 6                    NA               0.0000                    0
##   elapsedDuration movingDuration anaerobicTrainingEffect   deviceId
## 1         3955901        3099653                      NA 3968818126
## 2        16176575       15000179                     0.0 3968818126
## 3        11319866       10356078                     0.0 3968818126
## 4         5538505        5389603                     0.2 3968818126
## 5         3629000        3611000                      NA 3825981698
## 6         3585826        2858951                      NA 3968818126
##   minTemperature maxTemperature minElevation maxElevation        locationName
## 1             25             26           NA           NA                <NA>
## 2             19             29        18480        95480              Vienna
## 3             18             29        13980        32200           Korneuburg
## 4             19             27        24920        54400              Vienna
## 5             NA             NA          300         3420 City of Westminster
## 6             26             27           NA           NA                <NA>
##   maxVerticalSpeed lapCount endLongitude endLatitude activeSets totalSets
## 1               NA       34           NA          NA         NA        NA
## 2       0.43999939       25     16.25314    48.20399         NA        NA
## 3       0.34000092       17     16.38793    48.38009         NA        NA
## 4       0.08000183       18     16.31980    48.22264         NA        NA
## 5       0.16000004        1           NA          NA         NA        NA
## 6               NA       32           NA          NA         NA        NA
##   totalReps purposeful autoCalcCalories favorite    pr elevationCorrected
## 1        NA      FALSE            FALSE    FALSE FALSE             FALSE
## 2        NA      FALSE            FALSE    FALSE FALSE             FALSE
## 3        NA      FALSE            FALSE    FALSE FALSE             FALSE
```

```
## 4        NA      FALSE            FALSE    FALSE FALSE             FALSE
## 5        NA      FALSE            FALSE    FALSE FALSE             FALSE
## 6        NA      FALSE            FALSE    FALSE FALSE             FALSE
##   atpActivity parent maxRunCadence steps avgVerticalOscillation
## 1       FALSE  FALSE            NA    NA                    NA
## 2       FALSE  FALSE            NA    NA                    NA
## 3       FALSE  FALSE            NA    NA                    NA
## 4       FALSE  FALSE           104 15620                    NA
## 5       FALSE  FALSE            NA    NA                    NA
## 6       FALSE  FALSE            NA    NA                    NA
##   avgGroundContactTime avgStrideLength vO2MaxValue avgVerticalRatio
## 1                   NA              NA          NA               NA
## 2                   NA              NA          72               NA
## 3                   NA              NA          71               NA
## 4                   NA        115.6389          58               NA
## 5                   NA              NA          NA               NA
## 6                   NA              NA          NA               NA
##   avgGroundContactBalance avgDoubleCadence maxDoubleCadence avgPower
## 1                      NA               NA               NA       NA
## 2                      NA               NA               NA      260
## 3                      NA               NA               NA      202
## 4                      NA          172.375              208       NA
## 5                      NA               NA               NA      212
## 6                      NA               NA               NA       NA
##   avgBikeCadence maxBikeCadence strokes normPower avgLeftBalance
## 1             NA             NA    1198        NA             NA
## 2             83            114   17968  296.0000          49.92
## 3             82            107   12571  231.0000          49.90
## 4             NA             NA      NA        NA             NA
## 5             91            114       0  218.3049             NA
## 6             NA             NA    1152        NA             NA
##   avgRightBalance max20MinPower trainingStressScore intensityFactor
## 1              NA            NA                   NA              NA
## 2           50.08      375.1583                291.4           0.835
## 3           50.10      242.7692                122.8           0.653
## 4              NA            NA                   NA              NA
## 5              NA      221.0525                   NA              NA
## 6              NA            NA                   NA              NA
##   lactateThresholdBpm lactateThresholdSpeed avgStrokes activeLengths avgSwolf
## 1                  NA                    NA       23.0            60       74
## 2                  NA                    NA         NA            NA       NA
## 3                  NA                    NA         NA            NA       NA
## 4                  NA                    NA         NA            NA       NA
## 5                  NA                    NA         NA            NA       NA
## 6                  NA                    NA       22.6            57       72
##   poolLength avgStrokeDistance avgSwimCadence maxSwimCadence maxFtp workoutId
```

```
## 1       5000               217             27             29     NA        NA
## 2         NA                NA             NA             NA     NA        NA
## 3         NA                NA             NA             NA     NA        NA
## 4         NA                NA             NA             NA     NA        NA
## 5         NA                NA             NA             NA     NA        NA
## 6       5000               221             27             30     NA        NA
##   decoDive parentId avgVerticalSpeed maxDepth avgDepth surfaceInterval
## 1       NA       NA               NA       NA       NA              NA
## 2       NA       NA               NA       NA       NA              NA
## 3       NA       NA               NA       NA       NA              NA
## 4       NA       NA               NA       NA       NA              NA
## 5       NA       NA               NA       NA       NA              NA
## 6       NA       NA               NA       NA       NA              NA
##   floorsDescended bottomTime
## 1              NA         NA
## 2              NA         NA
## 3              NA         NA
## 4              NA         NA
## 5              NA         NA
## 6              NA         NA
```

You have many options to deal with issues :

- `sep` to specify the separator (`\t` for tabulation, `';'` for semicolon…)
- `dec` the decimal separator
- `encoding` the file encoding (special characters from windows/unix systems can be misdetected)
- `colClasses` to force one column to be imported in another type than what is detected
- `help(read.table)` for more options !

4.1.2 Excel files

You can import excel files (.xls and .xlsx) with the `readxl` package

```
readxl::read_excel("Data/Sports/Activities.xlsx") %>%
  head()
```

```
## # A tibble: 6 x 89
##   activityId uuidMsb uuidLsb name  activityType userProfileId timeZoneId
##        <dbl> <chr>   <chr>   <chr> <chr>                <dbl>      <dbl>
## 1 5570974040 3.6952~ -7.098~ En p~ lap_swimming       1141258        124
## 2 5566524321 -5.212~ -8.140~ Vien~ cycling            1141258        124
## 3 5561266034 1.2713~ -5.522~ Korn~ cycling            1141258        124
## 4 5555881653 -2.923~ -4.676~ Vien~ running            1141258        124
## 5 5551811953 7.8590~ -7.909~ Zwif~ virtual_ride       1141258        124
## 6 5551052200 -9.554~ -6.866~ En p~ lap_swimming       1141258        124
## # ... with 82 more variables: beginTimestamp <chr>, eventTypeId <dbl>,
```

```
## #   rule <chr>, sportType <chr>, startTimeGmt <dbl>, startTimeLocal <chr>,
## #   duration <chr>, distance <chr>, elevationGain <chr>, elevationLoss <chr>,
## #   avgSpeed <chr>, maxSpeed <chr>, avgHr <chr>, maxHr <chr>, calories <chr>,
## #   startLongitude <chr>, startLatitude <chr>, aerobicTrainingEffect <chr>,
## #   avgFractionalCadence <chr>, maxFractionalCadence <chr>,
## #   elapsedDuration <chr>, movingDuration <chr>, anaerobicTrainingEffect <chr>,
## #   deviceId <dbl>, minTemperature <chr>, maxTemperature <chr>,
## #   minElevation <chr>, maxElevation <chr>, locationName <chr>,
## #   maxVerticalSpeed <chr>, lapCount <dbl>, endLongitude <chr>,
## #   endLatitude <chr>, activeSets <chr>, totalSets <chr>, totalReps <chr>,
## #   purposeful <chr>, autoCalcCalories <chr>, favorite <chr>, pr <chr>,
## #   elevationCorrected <chr>, atpActivity <chr>, parent <chr>,
## #   maxRunCadence <chr>, steps <chr>, avgVerticalOscillation <chr>,
## #   avgGroundContactTime <chr>, avgStrideLength <chr>, vO2MaxValue <chr>,
## #   avgVerticalRatio <chr>, avgGroundContactBalance <chr>,
## #   avgDoubleCadence <chr>, maxDoubleCadence <chr>, avgPower <chr>,
## #   avgBikeCadence <chr>, maxBikeCadence <chr>, strokes <chr>, normPower <chr>,
## #   avgLeftBalance <chr>, avgRightBalance <chr>, max20MinPower <chr>,
## #   trainingStressScore <chr>, intensityFactor <chr>,
## #   lactateThresholdBpm <chr>, lactateThresholdSpeed <chr>, avgStrokes <chr>,
## #   activeLengths <chr>, avgSwolf <chr>, poolLength <chr>,
## #   avgStrokeDistance <chr>, avgSwimCadence <chr>, maxSwimCadence <chr>,
## #   maxFtp <chr>, workoutId <chr>, decoDive <chr>, parentId <chr>,
## #   avgVerticalSpeed <chr>, maxDepth <chr>, avgDepth <chr>,
## #   surfaceInterval <chr>, floorsDescended <chr>, bottomTime <chr>
```

Options :

- `sheet` to select whioch you want to import
- `range` : the "zone" of the sheet you want to import (beginning and ending row/column to be provided)
- `col_types` to specify the types of the column if misdetected
- `?readxl::read_excel` for more information

4.1.3 More formats

The `readr` package provides other convenient functions to read the most common (open) formats. With `haven`, you can also read data from proprietary formats (SPSS, SAS, Stat,...).

JSON files can be read with for example `rjsonlite` and we will use it in an applicaiton example.

XML and HTML files can be parsed with the `xml2`package.

4.1.4 Read from databases / big data

This is a huge topic that we will only mention here, but for (almost) each database engine, there is a package available in order to be able to read data from databases

- General purpose : `odbc`, `RODBC`, `DBI` → you will need to install the DB's drivers
- Dedicated : `RSQLite`, `RPostgres`, `RMariaDB` (can be used for mySQL too)… → drivers included

With the 3 first package, you can connect to "monolith" databases, as well as to distributed databases. You can find more information on the Rstudio website. Another interesting resources is the `dbplyr` vignette, that describes how to connect to a database and query it using `dplyr`'s verbs.

In addition, the `sparlyr` package allows you to interact with a spark cluster (using `dplyr syntax`)

4.2 The grammar of data manipulation

Alert : After this section, pandas will appear much less appealing….

Following the tidy data principles, `dplyr` implements an actual grammar of data manipulation with verbs and human-readable syntax.

4.2.1 The pipe

The first operator to know is the *pipe* operator, `%>%` which allows you to redirect the output of a command "to the right" and hence create readable chains of commands. Let's extract the last 3 characters of "hello world"

First solution : create useless objects

```
char <- "hello world"
rev_char <- stringi::stri_reverse(char)
sub3 <- substr(rev_char,1,3)
stringi::stri_reverse(sub3)
```

```
## [1] "rld"
```

Second solution : where's the beginning ????

```
stringi::stri_reverse(substr(stringi::stri_reverse("Hello World"),1,3))
```

```
## [1] "rld"
```

Third solution : using the pipe

```
"Hello world" %>%
  stringi::stri_reverse() %>%
  substr(1,3) %>%
  stringi::stri_reverse()
```

```
## [1] "rld"
```

Under the hood : the dot represents the result of the previous step and can be placed somewhere else in the next function (rather than the first argument)

```
"Hello world" %>%
  stringi::stri_reverse(.) %>%
  substr(.,1,3) %>%
  stringi::stri_reverse(.)
```

```
## [1] "rld"
```

4.2.2 The verbs of manipulation

What do you do with data ?

- Select columns → `select()`
- Filter rows → `filter()`
- Create / modify columns → `mutate()`
- Compute summaries of the columns → `summarise()`
- Do group-wise operations → `group_by()`
- Join with other tables → `left_join()`,`right_join()`, `inner_join()`, `anti_join()`, `full_join()`

I have the verbs, now I can associate them to make a sentence ! All those functions take as first argument a dataframe, which makes it very easy when chaining them with the pipe.

```
read.csv("Data/Sports/Activities.csv") %>%
  select(activityType,avgSpeed,distance,startLongitude,startLatitude,sportType) %>% #s
  mutate(distance=distance/100) %>% # distances are in decameter (?)
  filter(activityType!="other") %>% # remove activities "other
  group_by(activityType) %>%
  summarise(total_dist=mean(distance))
```

```
## # A tibble: 22 x 2
##    activityType            total_dist
##    <chr>                        <dbl>
##  1 cross_country_skiing_ws     16717.
##  2 cycling                     34124.
##  3 cyclocross                  41210.
##  4 hiking                      10678.
##  5 indoor_cardio                  NA
##  6 indoor_cycling                 NA
##  7 indoor_running              2311.
##  8 lap_swimming                3014.
##  9 multi_sport                63873.
## 10 open_water_swimming         2359.
## # ... with 12 more rows
```

4.2.3 Filter : conditions

This is the way you write conditions in R :

Syntax	Condition
==	Equality test
!=	Different than
%in% c(...)	Is in this list of values
>, >= <, <=	Greater/less than
! (x %in% c(...))	Not in the list

4.2.4 Mutate

Most of the data manipulation will be done in a mutate statement. This is where you can create additional columns, modify the ones existing. You can do any kind of transformation you want with this one. Depending on the type of the data, here are some additional packages that will help you :

- `lubridate` to easily handle date variables
- `forcats` to handle factors (categorical variables)
- `stringr` (and `stringi`) to handle strings variables and work with regular expressions
- `ifelse()` and `case_when()` to handle conditional operations

```
require(lubridate)
require(stringr)

dat <- dat %>%
  mutate(start_time=as_datetime(startTimeLocal/1000), # create a timestamp
         date = floor_date(start_time,"day"), # round to the day
         is_bike=ifelse(activityType %in%
                          c("cycling","virtual_ride","indoor_cycling","road_biking","cyclocross")
         # is it bike or not ?
         is_run = str_detect(activityType,"running|hicking"),
         activity_recoded = case_when(is_bike ~ "Bike",
                                      is_run ~ "Run",
                                      str_detect(activityType,"swim") ~"Swim",
                                      TRUE ~ "Other"))
```

4.2.5 Summarize

This operation consists in summarizing several rows ofinto one or more synthetic value(s). We will cover the topic more in detail 5 but the most common summary function that you can use are :

- For continuous variables : average, sum, median, standard deviation, interquartile range (IQR), concentration indexes,...
- For categorical variables : count, count distinct, concentration indexes,...

Simple summary statistics over one numerical variable :

```
dat %>% summarise(total_distance=sum(distance)) # Oups
```

```
##   total_distance
## 1             NA
```

```
dat %>% summarise(total_distance=sum(distance,na.rm = T))
```

```
##   total_distance
## 1    10693637146
```

```
dat %>% summarise(avg_distance=mean(distance,na.rm = T))
```

```
##   avg_distance
## 1      1941826
```

```
dat %>% summarise(median_distance=median(distance,na.rm = T))
```

```
##   median_distance
## 1         1186419
```

4.2.6 Manipulate several data in the same time

With all previous verbs above, you can use the `across` function to apply the same operation over a bunch of columns that you can select depending a simple enumeration or a condition (on their type or their name). This is a really powerful tool !

Example : we will convert all columns that are identifiers as character variables because the numbers have no meaning

```
dat <- dat %>%
  mutate(across(c(contains("Id"),contains("uuid")),
                as.character))
# Other stupid examples
dat %>%  summarise(across(where(is.numeric),
                          function(xx) sum(xx,na.rm=T)))
```

```
##   beginTimestamp startTimeGmt startTimeLocal    duration    distance
## 1   8.021486e+15 8.036783e+15   8.036814e+15 22850917079 10693637146
##   elevationGain elevationLoss avgSpeed maxSpeed  avgHr  maxHr calories
## 1     163374842     149923907 2186.754 113881.5 538175 637627 20569669
##   startLongitude startLatitude aerobicTrainingEffect avgFractionalCadence
## 1        18707.3      190193.1                7579.3             383.8125
##   maxFractionalCadence elapsedDuration movingDuration anaerobicTrainingEffect
## 1                170.5     11995810258     6623490405                   276.1
##   minTemperature maxTemperature minElevation maxElevation maxVerticalSpeed
## 1          46890          65603     35504232     66534189         322.7602
##   lapCount endLongitude endLatitude activeSets totalSets totalReps
## 1    26369     10138.19    39058.89          0         0         0
##   maxRunCadence    steps avgVerticalOscillation avgGroundContactTime
## 1         81893 27827620                5202.02             149787.7
##   vO2MaxValue avgVerticalRatio avgGroundContactBalance avgDoubleCadence
## 1       50052          4590.83                29360.86         124685.3
##   maxDoubleCadence avgPower avgBikeCadence maxBikeCadence  strokes normPower
## 1           163305   159062         120499         160192 11984172  172253.5
##   avgLeftBalance avgRightBalance max20MinPower trainingStressScore
## 1       29746.24        29853.76      178191.6            146746.6
##   intensityFactor lactateThresholdBpm lactateThresholdSpeed avgStrokes
```

```
## 1         640.338               12296               28.8716   36334.99
##   activeLengths avgSwolf poolLength avgStrokeDistance avgSwimCadence
## 1         82178    38881    2028199            113919          21590
##   maxSwimCadence maxFtp avgVerticalSpeed maxDepth avgDepth surfaceInterval
## 1          28979  28273                0        0        0               0
##   floorsDescended bottomTime
## 1               0          0
```

4.3 Let's import and wrangle some data !

4.3.1 The data

We will work on the summary data of all past activities, which come in JSON files. So basically, the data is contained in (nested) lists. This is real word data, it's super messy and dirty ! You will have to :

- Import the data
  - Import one of the files using `jsonlite`
  - Inspect and understand the structure of the list
  - Get all the metrics that are included
  - Figure out how to extract one specific metric for one activity
  - Design a function to extract one metric for all activities contained in the JSON
  - Design a function that will extract all metrics for all activities in the JSON
- Have a first cleaning of the data :
  - Check the distance/elevation variables ; what do you think ?
  - Check the speed related variables : what do you think ?
  - Check the calories variable and adjust it
  - Check the duration related variable and adjust them to have minutes
  - To help you figuring out, you can check an activity on garmin's site using this url and change the activity number for the one you are inspecting

A lot of reverse engineering ahead

4.3.2 One tool you will need : `lapply()`

JSON are lists, and to iterate over list elements, you can either use `for` loops, which is highly *not recommended* (R is no good with loops), or use `lapply()`. This function applies an operation over all elements of a list (or vector) and returns a list containing the result. You can also use `sapply()` which tries to coerce the result to a vector (if possible) if the expected output is not a list.

```
random_list <- lapply(1:100,function(xx) rnorm(100,xx,xx/5))
str(random_list[1:5])
```

```
## List of 5
##  $ : num [1:100] 0.653 0.839 0.952 0.781 0.758 ...
##  $ : num [1:100] 1.305 2.453 0.972 2.419 1.907 ...
##  $ : num [1:100] 2.38 3.31 2.65 3.27 3.2 ...
##  $ : num [1:100] 4.95 3.46 4.23 5.08 5.37 ...
##  $ : num [1:100] 4.53 6.16 5.43 5.09 3.46 ...
```

Attention : be careful with the squared brackets

```
random_list[1] # is a list
```

```
## [[1]]
##   [1] 0.6534956 0.8394734 0.9520231 0.7810726 0.7577881 1.1833391 1.2953283
##   [8] 1.1170205 0.7598093 0.9069834 1.0612687 0.4860967 0.9583442 1.2883162
##  [15] 0.7559182 0.9321351 1.1605419 0.8560776 1.2296965 1.1961155 1.1452899
##  [22] 0.8937648 1.1180562 1.1291864 1.1588141 0.7867302 1.2711314 0.8855429
##  [29] 1.2287864 0.7456788 0.9652640 0.7649677 0.9147540 0.8360354 1.3852953
##  [36] 1.0987498 0.9648034 0.6702452 0.8118263 0.8490696 1.1890466 1.0536951
##  [43] 0.9466750 1.1516804 1.2437038 1.3202114 1.0162013 0.3732095 0.8947668
##  [50] 1.1670457 1.0740586 0.9312860 1.1014046 1.2574607 0.8831380 1.2873086
##  [57] 1.3694036 1.2518263 1.1654449 0.5834937 0.6536882 0.9924216 1.1499586
##  [64] 1.1995749 1.4525548 0.9177919 1.0160229 0.7752616 1.0055312 0.9059316
##  [71] 0.8091390 0.6507289 0.9291632 0.9774625 0.9732182 0.9829048 1.1303813
##  [78] 1.0197155 0.8936495 0.7905860 1.1401793 1.1591278 1.2085776 1.3789252
##  [85] 1.1331738 0.7921537 1.3730403 0.8314237 0.9845227 0.7082945 0.9509746
##  [92] 1.1636481 1.0714989 0.9785293 1.0279219 1.0541376 1.1354783 0.8092468
##  [99] 0.8652608 0.7311741
```

```
random_list[[1]] # is a vector
```

```
##   [1] 0.6534956 0.8394734 0.9520231 0.7810726 0.7577881 1.1833391 1.2953283
##   [8] 1.1170205 0.7598093 0.9069834 1.0612687 0.4860967 0.9583442 1.2883162
##  [15] 0.7559182 0.9321351 1.1605419 0.8560776 1.2296965 1.1961155 1.1452899
##  [22] 0.8937648 1.1180562 1.1291864 1.1588141 0.7867302 1.2711314 0.8855429
##  [29] 1.2287864 0.7456788 0.9652640 0.7649677 0.9147540 0.8360354 1.3852953
##  [36] 1.0987498 0.9648034 0.6702452 0.8118263 0.8490696 1.1890466 1.0536951
##  [43] 0.9466750 1.1516804 1.2437038 1.3202114 1.0162013 0.3732095 0.8947668
##  [50] 1.1670457 1.0740586 0.9312860 1.1014046 1.2574607 0.8831380 1.2873086
##  [57] 1.3694036 1.2518263 1.1654449 0.5834937 0.6536882 0.9924216 1.1499586
##  [64] 1.1995749 1.4525548 0.9177919 1.0160229 0.7752616 1.0055312 0.9059316
##  [71] 0.8091390 0.6507289 0.9291632 0.9774625 0.9732182 0.9829048 1.1303813
##  [78] 1.0197155 0.8936495 0.7905860 1.1401793 1.1591278 1.2085776 1.3789252
##  [85] 1.1331738 0.7921537 1.3730403 0.8314237 0.9845227 0.7082945 0.9509746
##  [92] 1.1636481 1.0714989 0.9785293 1.0279219 1.0541376 1.1354783 0.8092468
##  [99] 0.8652608 0.7311741
```

4.4 Tidy your data

The data will almost never come in a ready-to-use format. Wrangling the data, beyond *cleaning* it also sometimes imply to *reshape* it so that it conforms to the tidy principles. For that you have 2 functions :

- `pivot_longer()` which will convert columns into rows
- `pivot_wider()`, the reciprocate operation, which will convert rows into columns

For example, we can chose that an observation is the combination of an activity and a metric. This representation can be useful in some cases (see @ref(adv_viz)).

```
dat_long <- dat %>%
  select(activityId,where(is.numeric)) %>%
  pivot_longer(-activityId,names_to="metric",values_to="value")
dat_long
```

```
## # A tibble: 363,924 x 3
##    activityId metric            value
##    <chr>      <chr>             <dbl>
##  1 5570974040 beginTimestamp  1.60e+12
##  2 5570974040 startTimeGmt    1.60e+12
##  3 5570974040 startTimeLocal  1.60e+12
##  4 5570974040 duration        3.64e+ 6
##  5 5570974040 distance        3.00e+ 5
##  6 5570974040 elevationGain  NA
##  7 5570974040 elevationLoss  NA
##  8 5570974040 avgSpeed        9.86e- 2
##  9 5570974040 maxSpeed        1.12e- 1
## 10 5570974040 avgHr          NA
## # ... with 363,914 more rows
```

With this format you can get summary statistics for all metrics also easily :

```
group_by(dat_long,metric) %>%
  summarise(mean_val=mean(value,na.rm=T))
```

```
## # A tibble: 66 x 2
##    metric                   mean_val
##    <chr>                       <dbl>
##  1 activeLengths             61.8
##  2 activeSets                 0
##  3 aerobicTrainingEffect      2.88
##  4 anaerobicTrainingEffect    0.298
##  5 avgBikeCadence            91.2
##  6 avgDepth                   0
##  7 avgDoubleCadence         162.
```

```
##  8 avgFractionalCadence      0.0696
##  9 avgGroundContactBalance  49.4
## 10 avgGroundContactTime    252.
## # ... with 56 more rows
```

And you you can go back to the original format if you want :

```
group_by(dat_long,metric) %>%
  summarise(mean_val=mean(value,na.rm=T)) %>%
  pivot_wider(names_from = metric,values_from=mean_val)
```

```
## # A tibble: 1 x 66
##   activeLengths activeSets aerobicTraining~ anaerobicTraini~ avgBikeCadence
##           <dbl>      <dbl>            <dbl>            <dbl>          <dbl>
## 1          61.8          0             2.88            0.298           91.2
## # ... with 61 more variables: avgDepth <dbl>, avgDoubleCadence <dbl>,
## #   avgFractionalCadence <dbl>, avgGroundContactBalance <dbl>,
## #   avgGroundContactTime <dbl>, avgHr <dbl>, avgLeftBalance <dbl>,
## #   avgPower <dbl>, avgRightBalance <dbl>, avgSpeed <dbl>,
## #   avgStrokeDistance <dbl>, avgStrokes <dbl>, avgSwimCadence <dbl>,
## #   avgSwolf <dbl>, avgVerticalOscillation <dbl>, avgVerticalRatio <dbl>,
## #   avgVerticalSpeed <dbl>, beginTimestamp <dbl>, bottomTime <dbl>,
## #   calories <dbl>, distance <dbl>, duration <dbl>, elapsedDuration <dbl>,
## #   elevationGain <dbl>, elevationLoss <dbl>, endLatitude <dbl>,
## #   endLongitude <dbl>, floorsDescended <dbl>, intensityFactor <dbl>,
## #   lactateThresholdBpm <dbl>, lactateThresholdSpeed <dbl>, lapCount <dbl>,
## #   max20MinPower <dbl>, maxBikeCadence <dbl>, maxDepth <dbl>,
## #   maxDoubleCadence <dbl>, maxElevation <dbl>, maxFractionalCadence <dbl>,
## #   maxFtp <dbl>, maxHr <dbl>, maxRunCadence <dbl>, maxSpeed <dbl>,
## #   maxSwimCadence <dbl>, maxTemperature <dbl>, maxVerticalSpeed <dbl>,
## #   minElevation <dbl>, minTemperature <dbl>, movingDuration <dbl>,
## #   normPower <dbl>, poolLength <dbl>, startLatitude <dbl>,
## #   startLongitude <dbl>, startTimeGmt <dbl>, startTimeLocal <dbl>,
## #   steps <dbl>, strokes <dbl>, surfaceInterval <dbl>, totalReps <dbl>,
## #   totalSets <dbl>, trainingStressScore <dbl>, vO2MaxValue <dbl>
```

The result looks very much like the one we had with `across` but the intermediate manipulationsncan be very useful in some cases. For instance, you could join the `dat_long` dataframe or its summary with an external data that has values by metrics (eg the average metric values for pro athletes).

Chapter 5

Statistics

Let's load and clean the data (which you have done during the exercises)

This session aims to give a practical guide to explore a dataset you've never seen before and to understand some of the key statistical concepts. You will learn to describe each *variable* of a dataset and assess the strength of the relationship between two variables whatever their types may be. For that, we'll see how to visually explore a dataset and to quantify what the graphics show. We will also give an overview of what statistical inference is and what it can be used for.

This section covers the following topics :

- Definitions
- Descriptive statistics
- Univariate statistics
- Bivariate statistics
- Statistical inference :
- The statistical model
- Main theorems to be aware of
- Introduction to statistical tests

5.1 Definitions

5.1.1 Terminology

A data set can be viewed in two different manners :

- A set of rows, or **statistical individuals**, aka observations (or instances in the galaxy of machine learning). This can be anything

- A set of columns, or **variables** that describe the individuals

It is crucial to have a good understanding of **what the statistical individual is**, and that can be challenging !

Some examples :

```
head(dat)
```

```
##   activityId              uuidMsb              uuidLsb                name
## 1 5570974040  3695223521635878400 -7098506714510231552          En piscine
## 2 5566524321 -5212790351453402112 -8140622953684101120     Vienna Cyclisme
## 3 5561266034  1271355725617447936 -5522843681386392576 Korneuburg Cyclisme
## 4 5555881653 -2923937867469140992 -4676832321872288768       Vienna Course
## 5 5551811953  7859042982901073920 -7909531516612413440      Zwift - London
## 6 5551052200  -955408594416025216 -6866434522826685440          En piscine
##   activityType userProfileId timeZoneId beginTimestamp eventTypeId   rule
## 1 lap_swimming       1141258        124   1.600700e+12           9 public
## 2      cycling       1141258        124   1.600602e+12           9 public
## 3      cycling       1141258        124   1.600523e+12           9 public
## 4      running       1141258        124   1.600439e+12           9 public
## 5 virtual_ride       1141258        124   1.600362e+12           9 public
## 6 lap_swimming       1141258        124   1.600353e+12           9 public
##   sportType startTimeGmt startTimeLocal  duration  distance elevationGain
## 1   GENERIC 1.600700e+12   1.600708e+12  60.67035   3.00000            NA
## 2   CYCLING 1.600602e+12   1.600609e+12 251.07617 122.93220          2001
## 3   CYCLING 1.600523e+12   1.600530e+12 173.04054  83.60927           745
## 4   RUNNING 1.600439e+12   1.600447e+12  89.89717  17.92890           426
## 5   GENERIC 1.600362e+12   1.600369e+12  60.44708  34.92347           160
## 6   GENERIC 1.600353e+12   1.600360e+12  55.98575   2.85000            NA
##   elevationLoss avgSpeed maxSpeed avgHr maxHr  calories startLongitude
## 1            NA   3.5496   4.0392    NA    NA  608.8748            NA
## 2          1968  29.3760  69.5340   144   176 3906.6128       16.31524
## 3           742  28.9908  70.4052   117   161 2090.0028       16.33986
## 4           425  11.9664  44.9100   155   176 1176.6906       16.31587
## 5             0  34.6644  58.9104   138   163  737.0590        0.00000
## 6            NA   3.6396  12.6864    NA    NA  598.8604            NA
##   startLatitude aerobicTrainingEffect avgFractionalCadence maxFractionalCadence
## 1            NA                    NA               0.0000                    0
## 2      48.20915                   3.5               0.0000                    0
## 3      48.34994                   2.4               0.0000                    0
## 4      48.21432                   3.0               0.1875                    0
## 5       0.00000                   0.0               0.0000                    0
## 6            NA                    NA               0.0000                    0
##   elapsedDuration movingDuration anaerobicTrainingEffect   deviceId
## 1        65.93168       51.66088                      NA 3968818126
## 2       269.60959      250.00298                     0.0 3968818126
```

```
## 3       188.66444      172.60130                    0.0 3968818126
## 4        92.30841       89.82672                    0.2 3968818126
## 5        60.48333       60.18333                     NA 3825981698
## 6        59.76377       47.64918                     NA 3968818126
##   minTemperature maxTemperature minElevation maxElevation        locationName
## 1             25             26           NA           NA                <NA>
## 2             19             29        184.8        954.8              Vienna
## 3             18             29        139.8        322.0           Korneuburg
## 4             19             27        249.2        544.0              Vienna
## 5             NA             NA          3.0         34.2 City of Westminster
## 6             26             27           NA           NA                <NA>
##   maxVerticalSpeed lapCount endLongitude endLatitude activeSets totalSets
## 1               NA       34           NA          NA         NA        NA
## 2        15.839978       25     16.25314    48.20399         NA        NA
## 3        12.240033       17     16.38793    48.38009         NA        NA
## 4         2.880066       18     16.31980    48.22264         NA        NA
## 5         5.760001        1           NA          NA         NA        NA
## 6               NA       32           NA          NA         NA        NA
##   totalReps purposeful autoCalcCalories favorite    pr elevationCorrected
## 1        NA      FALSE            FALSE    FALSE FALSE                 0
## 2        NA      FALSE            FALSE    FALSE FALSE                 0
## 3        NA      FALSE            FALSE    FALSE FALSE                 0
## 4        NA      FALSE            FALSE    FALSE FALSE                 0
## 5        NA      FALSE            FALSE    FALSE FALSE                 0
## 6        NA      FALSE            FALSE    FALSE FALSE                 0
##   atpActivity parent maxRunCadence steps avgVerticalOscillation
## 1       FALSE  FALSE            NA    NA                    NA
## 2       FALSE  FALSE            NA    NA                    NA
## 3       FALSE  FALSE            NA    NA                    NA
## 4       FALSE  FALSE           104 15620                    NA
## 5       FALSE  FALSE            NA    NA                    NA
## 6       FALSE  FALSE            NA    NA                    NA
##   avgGroundContactTime  avgStrideLength vO2MaxValue avgVerticalRatio
## 1                   NA             <NA>          NA               NA
## 2                   NA             <NA>          72               NA
## 3                   NA             <NA>          71               NA
## 4                   NA 115.638917703599          58               NA
## 5                   NA             <NA>          NA               NA
## 6                   NA             <NA>          NA               NA
##   avgGroundContactBalance avgDoubleCadence maxDoubleCadence avgPower
## 1                      NA               NA               NA       NA
## 2                      NA               NA               NA      260
## 3                      NA               NA               NA      202
## 4                      NA          172.375              208       NA
## 5                      NA               NA               NA      212
## 6                      NA               NA               NA       NA
```

```
##   avgBikeCadence maxBikeCadence strokes normPower avgLeftBalance
## 1             NA             NA    1198        NA             NA
## 2             83            114   17968  296.0000          49.92
## 3             82            107   12571  231.0000          49.90
## 4             NA             NA      NA        NA             NA
## 5             91            114       0  218.3049             NA
## 6             NA             NA    1152        NA             NA
##   avgRightBalance max20MinPower trainingStressScore intensityFactor
## 1              NA            NA                  NA              NA
## 2           50.08      375.1583               291.4           0.835
## 3           50.10      242.7692               122.8           0.653
## 4              NA            NA                  NA              NA
## 5              NA      221.0525                  NA              NA
## 6              NA            NA                  NA              NA
##   lactateThresholdBpm lactateThresholdSpeed avgStrokes activeLengths avgSwolf
## 1                  NA                    NA       23.0            60       74
## 2                  NA                    NA         NA            NA       NA
## 3                  NA                    NA         NA            NA       NA
## 4                  NA                    NA         NA            NA       NA
## 5                  NA                    NA         NA            NA       NA
## 6                  NA                    NA       22.6            57       72
##   poolLength avgStrokeDistance avgSwimCadence maxSwimCadence maxFtp workoutId
## 1       5000               217             27             29     NA      <NA>
## 2         NA                NA             NA             NA     NA      <NA>
## 3         NA                NA             NA             NA     NA      <NA>
## 4         NA                NA             NA             NA     NA      <NA>
## 5         NA                NA             NA             NA     NA      <NA>
## 6       5000               221             27             30     NA      <NA>
##   decoDive parentId avgVerticalSpeed maxDepth avgDepth surfaceInterval
## 1       NA     <NA>               NA       NA       NA              NA
## 2       NA     <NA>               NA       NA       NA              NA
## 3       NA     <NA>               NA       NA       NA              NA
## 4       NA     <NA>               NA       NA       NA              NA
## 5       NA     <NA>               NA       NA       NA              NA
## 6       NA     <NA>               NA       NA       NA              NA
##   floorsDescended bottomTime          start_time       date is_bike is_run
## 1              NA         NA 2020-09-21 17:00:51 2020-09-21   FALSE  FALSE
## 2              NA         NA 2020-09-20 13:37:03 2020-09-20    TRUE  FALSE
## 3              NA         NA 2020-09-19 15:38:41 2020-09-19    TRUE  FALSE
## 4              NA         NA 2020-09-18 16:28:37 2020-09-18   FALSE   TRUE
## 5              NA         NA 2020-09-17 18:55:06 2020-09-17    TRUE  FALSE
## 6              NA         NA 2020-09-17 16:33:09 2020-09-17   FALSE  FALSE
##   activity_recoded qual_distance        qual_avgHr
## 1             Swim         Short              <NA>
## 2             Bike     Very long    High intensity
## 3             Bike     Very long     Low intensity
```

```
## 4              Run          Long              <NA>
## 5             Bike     Very long Average intensity
## 6             Swim         Short              <NA>
```

```
group_by(dat,activityType) %>%
  summarise(total_dist=sum(distance,na.rm=T),avg_speed=mean(avgSpeed,na.rm=T),avg_power=mean(avgP
            .groups="keep") %>%
  head()
```

```
## # A tibble: 6 x 4
## # Groups:   activityType [6]
##   activityType            total_dist avg_speed avg_power
##   <chr>                        <dbl>     <dbl>     <dbl>
## 1 cross_country_skiing_ws      635.      12.6       NaN
## 2 cycling                    74185.      22.8       253.
## 3 cyclocross                    41.2     18.4       NaN
## 4 hiking                        74.7      3.95      NaN
## 5 indoor_cardio                  0        0         NaN
## 6 indoor_cycling               592.       1.55      225.
```

```
group_by(dat,date) %>%
  summarise(total_dist=sum(distance,na.rm=T),avg_speed=mean(avgSpeed,na.rm=T),avg_power=mean(avgP
            .groups="keep") %>%
  head()
```

```
## # A tibble: 6 x 4
## # Groups:   date [6]
##   date                total_dist avg_speed avg_power
##   <dttm>                   <dbl>     <dbl>     <dbl>
## 1 2008-05-27 00:00:00      9.43      21.0        NaN
## 2 2008-11-25 00:00:00      9.25      23.3        NaN
## 3 2008-11-26 00:00:00     19.9       12.7        NaN
## 4 2008-11-27 00:00:00     21.3       22.1        NaN
## 5 2008-11-28 00:00:00     10.6       13.2        NaN
## 6 2008-11-29 00:00:00      0.208      6.90       NaN
```

5.1.2 Types of variables

The way we analyse variables depends on their *type* :

- Numerical variables :
- Continuous : income, revenue $\in \mathbb{R}, \mathbb{R}^+$
- Discrete : number of person per household $\in \mathbb{Z}, \mathbb{N}$
- Categorical variables :
- Ordered : small, medium, large
- Unordered : male, female

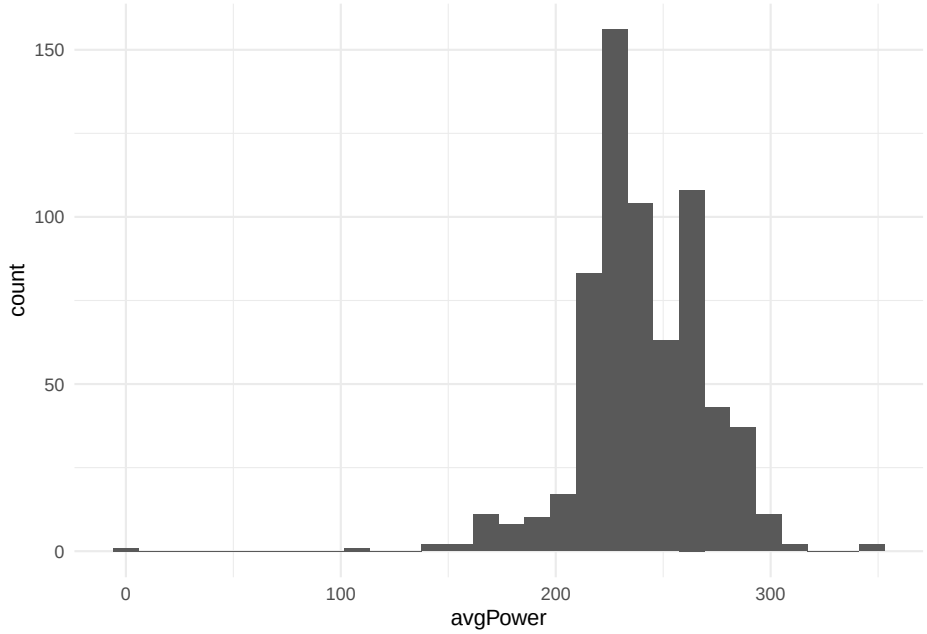
5.2 Univariate statistics

5.2.1 Numerical variables

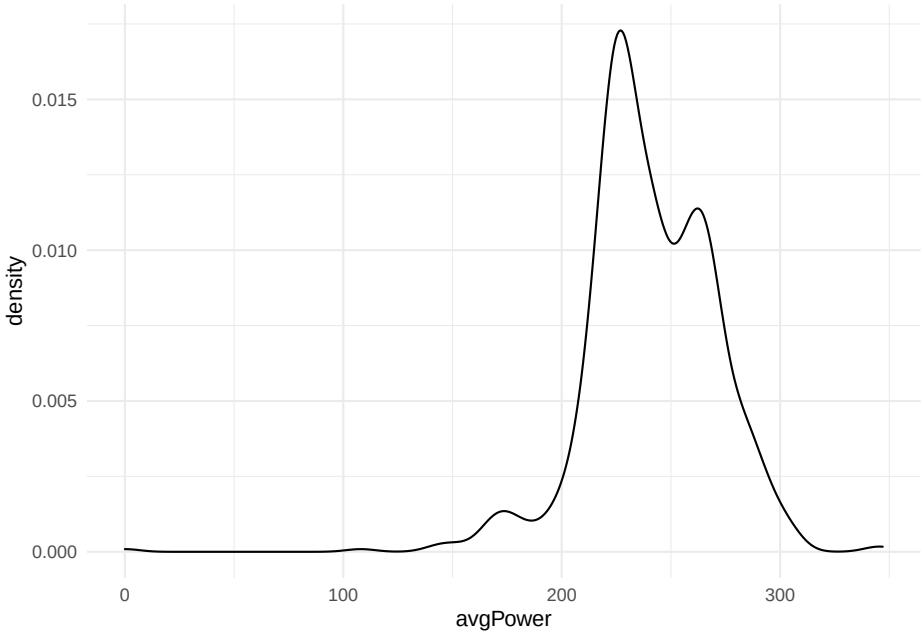
5.2.1.1 Distribution

The distribution of a variable quantifies the number of individuals how have a certain value of the variable. We can visualize the distribution either with histograms or density plot, which are the "empirical counterparts" of the probability density function.

```
ggplot(dat,aes(avgPower)) + geom_histogram() + theme_minimal()
```



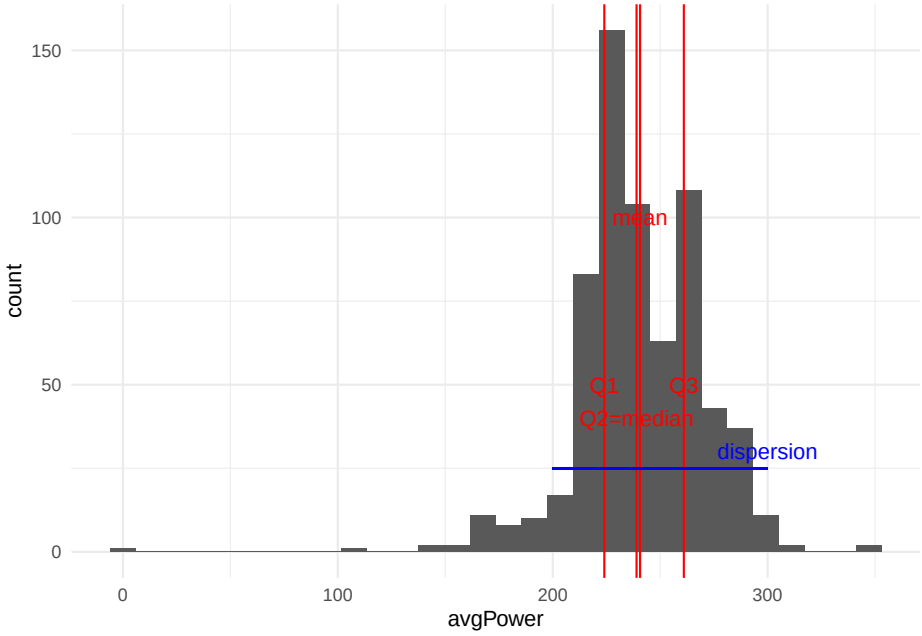

```
ggplot(dat,aes(avgPower)) + geom_density() + theme_minimal()
```



5.2.1.2 Descriptive statistics

We typically want to measure what the "average" value is, along with "how diverse is my population". For that, we can use either sum-based statistics (mean, standard deviation) or quantiles. Quantile-based statistics are said to be **robust** because much less sensitive to outliers. But they are more computationally expensive.

```
stats <- c(quantile(dat$avgPower,1:3/4,na.rm = T),mean(dat$avgPower,na.rm = T))
ggplot(dat,aes(avgPower)) + geom_histogram() +
  geom_vline(xintercept = stats,color="red") +
  annotate(geom = "text",x=stats,y=c(50,40,50,100),label=c("Q1","Q2=median","Q3","mean"),color="r
  geom_segment(aes(x=200,y=25,xend=300,yend=25),color="blue") +
  annotate(geom="text",x=300,y=30,label="dispersion",color="blue") + theme_minimal()
```



5.2.1.2.1 Central tendency Central tendency statistics allow you to have an idea of the order of magnitude of the attribute you are interested in, over the population.

```
summary(dat$avgPower)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##     0.0   224.0   239.0   240.6   261.0   347.0    4853
```

```
quantile(dat$avgPower,probs=0:10/10,na.rm = T)
```

```
##   0%  10%  20%  30%  40%  50%  60%  70%  80%  90% 100%
##    0  212  221  226  230  239  246  258  265  276  347
```

5.2.1.2.2 Dispersion Dispersion describes how heterogenous our population is. It can be measured with various measurements (not exhaustive here)

```
sd(dat$avgPower,na.rm = T) # standard deviation
```

```
## [1] 29.83959
```

```
IQR(dat$avgPower,na.rm = T) # interquartile range
```

```
## [1] 37
```

```
sd(dat$avgPower,na.rm = T)/mean(dat$avgPower,na.rm = T) # coefficient of variation
```

```
## [1] 0.1240017
```

How to read it :

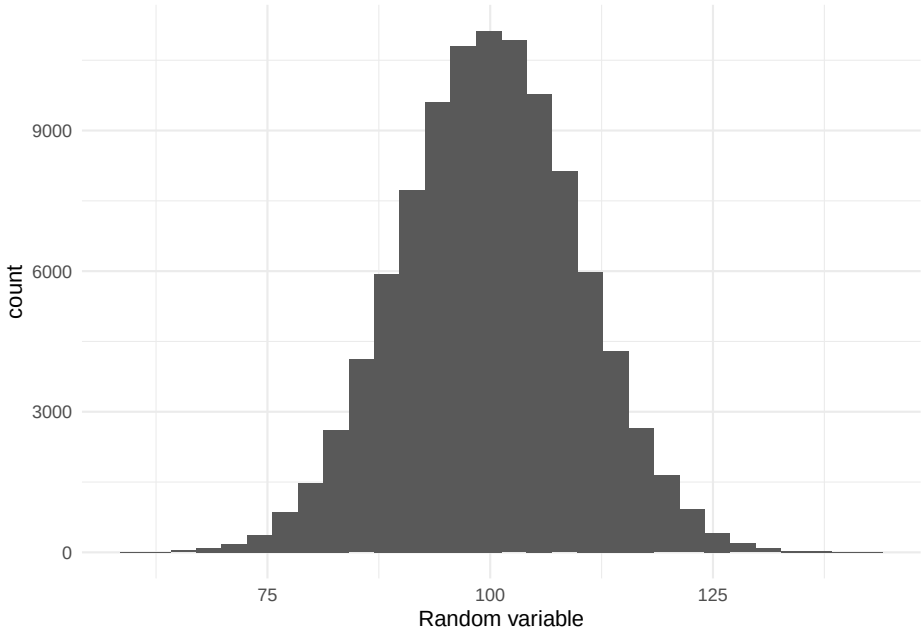
- The average deviation to the average power is 29 watts
- The age difference between the rides in the 25% "less powerful" rides and the 25% "most powerful" rides is 37 watts
- The average deviation to the average power is 12% of the average power

The latter allows to compare dispersion between variables that have different units

5.2.1.3 Dealing with various shapes

The traditional example of a distribution is the gaussian distribution

```
fake <- data.frame(xx=rnorm(100000,100,10))
ggplot(fake,aes(xx)) + geom_histogram() + labs(x="Random variable")+ theme_minimal()
```



In this case, we have a very interesting property : symmetry, which makes mean and median very close. If the coefficient of variation is not too high, the **tail** is pretty short.

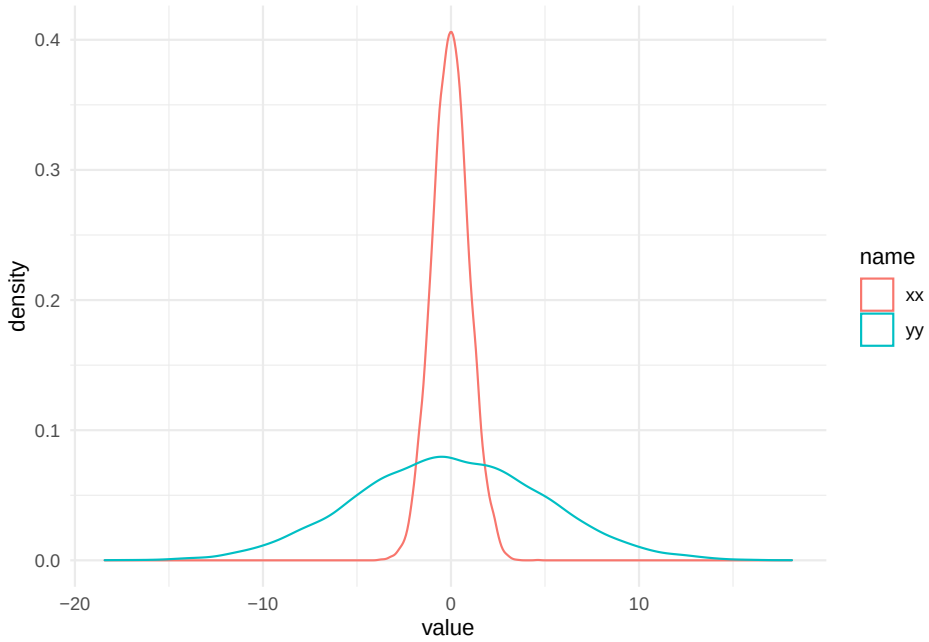
In real life,it (almost) never happens. Therefore, to understand what happens, you can check :

- How different are mean and median
- Does a log transformation make the distribution "look better"
- Is it symmetric → skewness
- Is flat no not → kurtosis

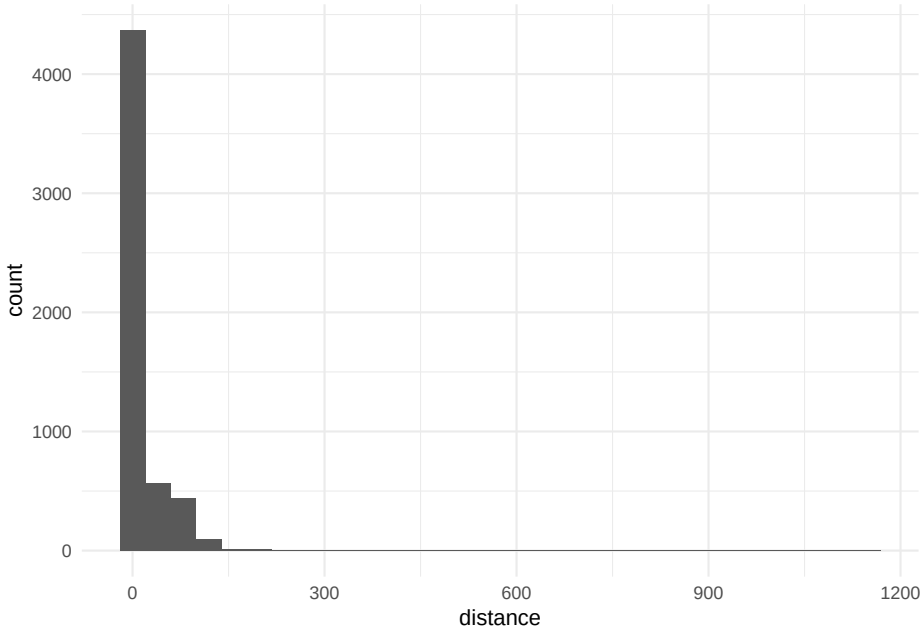
- Is the distribution highly concentrated (few individuals get almost the whole cake) → concentration indexes (Gini, enthropy, Herfindahl…). You can check the package `ineq`
- Are there outliers (which generates a long tail) → outlier detection (vast field…). You can start with the previous

Flat or not flat ?

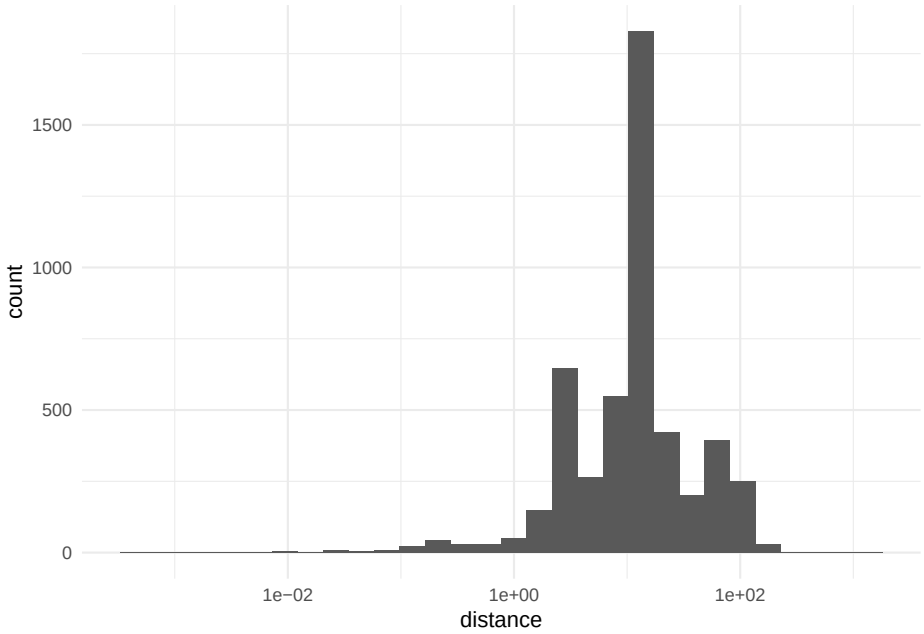
```
data.frame(xx=rnorm(10000),yy=rnorm(10000,0,5))  %>%
  pivot_longer(everything()) %>%
  ggplot(aes(value,color=name)) + geom_density()+ theme_minimal()
```




```
ggplot(dat,aes(distance)) + geom_histogram() + theme_minimal()
```




```
ggplot(dat,aes(distance)) + geom_histogram() + scale_x_log10() + theme_minimal()
```



5.2.1.4 Exercises :

What can you tell about the distance variable ? - Draw the distribution of this variable. How much is the maximum distance of the 20% shortest activities ; the minimum distance of the 5% longest activities ? - What unit do you think it is ? Did you check the maximum value ? - Is there more dispersion in the distance or the average power ? using the `facet_wrap` function of `ggplot2`, compare the distributions of distance and avgPower. - I want to group activities in 5 categories based on the distance. This operation is called discretization (very useful for choropleth maps). Search for available methods, and apply some of them. Which one is best suited to this variable ? Which one should you avoid ?

5.2.2 Categorical variables

5.2.2.1 Working with factors

Factors are an optimized way to store categorical variables (encoded in integers). The distinct categories are stored in the `level` attribute which you can interact with.

```
as.factor(dat$activityType) %>%
  levels()
```

```
##  [1] "cross_country_skiing_ws" "cycling"
##  [3] "cyclocross"              "hiking"
##  [5] "indoor_cardio"           "indoor_cycling"
##  [7] "indoor_running"          "lap_swimming"
##  [9] "multi_sport"             "open_water_swimming"
## [11] "other"                   "road_biking"
## [13] "running"                 "street_running"
## [15] "strength_training"       "swimming"
## [17] "swimToBikeTransition"    "trail_running"
## [19] "transition"              "treadmill_running"
## [21] "uncategorized"           "virtual_ride"
## [23] "walking"
```

```
as.factor(dat$activityType) %>%
  str()
```

```
##  Factor w/ 23 levels "cross_country_skiing_ws",..: 8 2 2 13 22 8 2 13 2 8 ...
```

For more functionalities you can use the forcats package which provides convenient tools (eg to recode the variable)

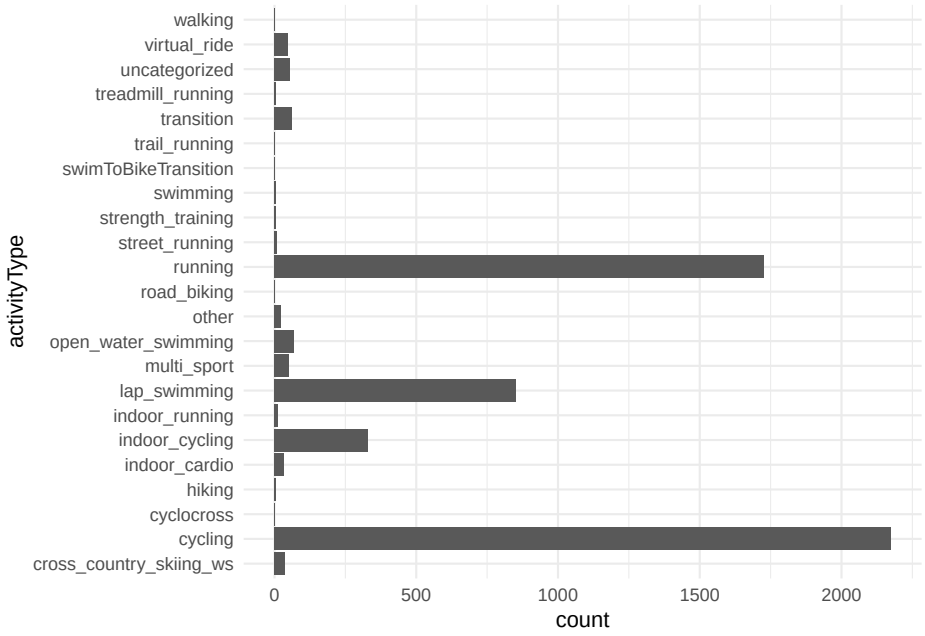
5.2.2.2 Barcharts

The barchart (which IS NOT a histogram) is the most common representation for categorical variables. You can also use the pie chart (but it requires to hack

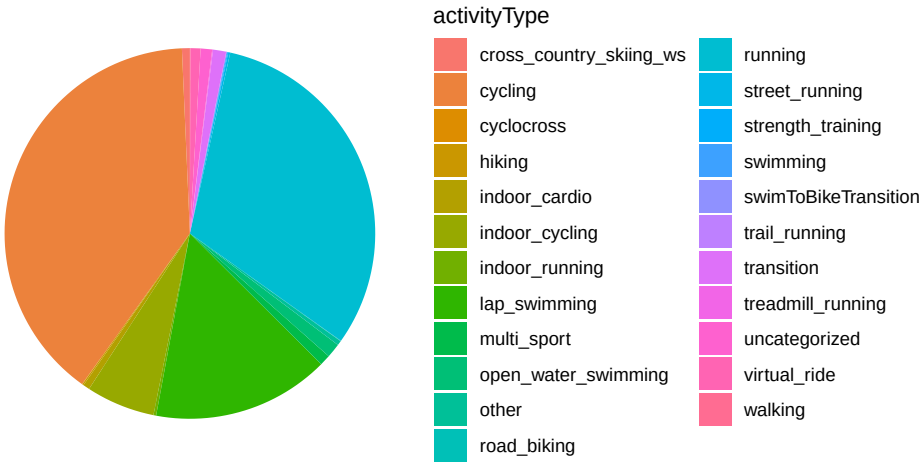
a little ggplot). Pie charts are despised by the majority of statisticians but it can be adapted if the sizes really differ. Some material to make your own opinion :

- Why it's bad
- Defense

```
ggplot(dat,aes(activityType)) + geom_bar() + coord_flip() + theme_minimal()
```




```
ggplot(dat,aes(x="",fill=activityType)) +
  geom_bar(width=1) +
  coord_polar("y",start=0) +
  theme_void()
```



5.2.2.3 Contingency tables

After visualizing, how can we measure the number of cases and the percent in each category ?

```
table(dat$activityType)
```

```
##
## cross_country_skiing_ws                  cycling                cyclocross
##                      38                     2174                        1
##                  hiking             indoor_cardio          indoor_cycling
##                       7                       32                      331
##          indoor_running             lap_swimming             multi_sport
##                      13                      852                       52
##     open_water_swimming                    other             road_biking
##                      69                       22                        2
##                 running            street_running       strength_training
##                    1728                       10                        4
##                swimming      swimToBikeTransition           trail_running
##                       4                        1                        1
##              transition         treadmill_running           uncategorized
##                      63                        4                       55
##            virtual_ride                   walking
##                      49                        2
```

```
table(dat$activityType) %>%
  prop.table()*100
```

```
##
## cross_country_skiing_ws                  cycling                cyclocross
##              0.68915488             39.42691331              0.01813565
##                  hiking             indoor_cardio          indoor_cycling
```

```
##              0.12694958              0.58034095              6.00290170
##          indoor_running            lap_swimming             multi_sport
##              0.23576351             15.45157780              0.94305404
##     open_water_swimming                   other             road_biking
##              1.25136017              0.39898440              0.03627131
##                 running          street_running       strength_training
##             31.33841132              0.18135655              0.07254262
##                swimming    swimToBikeTransition          trail_running
##              0.07254262              0.01813565              0.01813565
##              transition       treadmill_running          uncategorized
##              1.14254625              0.07254262              0.99746101
##            virtual_ride                 walking
##              0.88864708              0.03627131
```

Another solution is to use what you've learned in the previous section (@ref()) : aggregation !

```
group_by(dat,activityType) %>%
  summarise(number=n(),proportion=n()/nrow(dat))
```

```
## # A tibble: 23 x 3
##    activityType            number proportion
##    <chr>                    <int>      <dbl>
##  1 cross_country_skiing_ws     38   0.00689
##  2 cycling                   2174   0.394
##  3 cyclocross                   1   0.000181
##  4 hiking                       7   0.00127
##  5 indoor_cardio               32   0.00580
##  6 indoor_cycling             331   0.0600
##  7 indoor_running              13   0.00236
##  8 lap_swimming               852   0.155
##  9 multi_sport                 52   0.00943
## 10 open_water_swimming         69   0.0125
## # ... with 13 more rows
```

5.3 Bivariate statistics

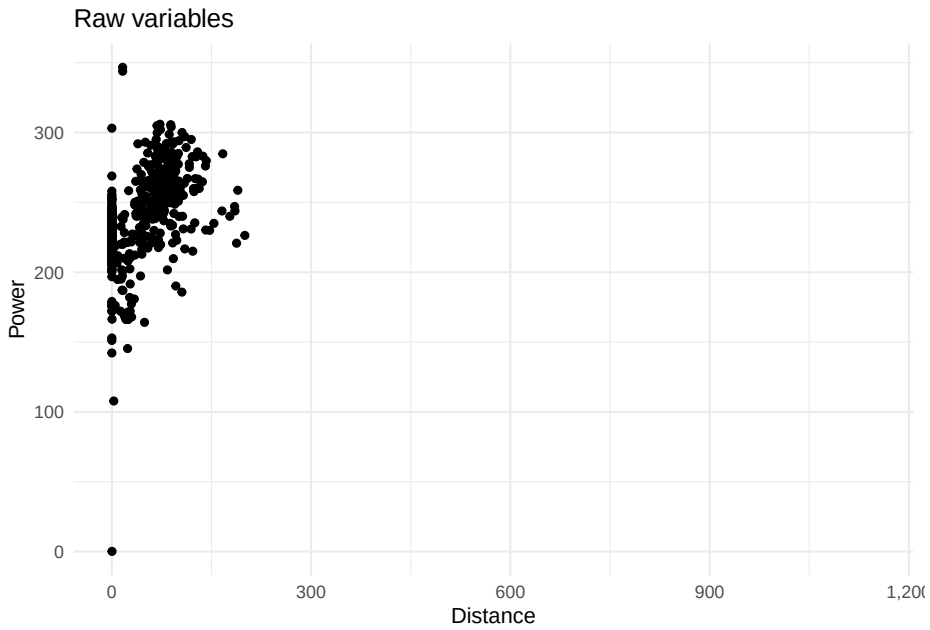
In this section, we see how to represent the relationship between two variables and measure it

5.3.1 2 continuous variables

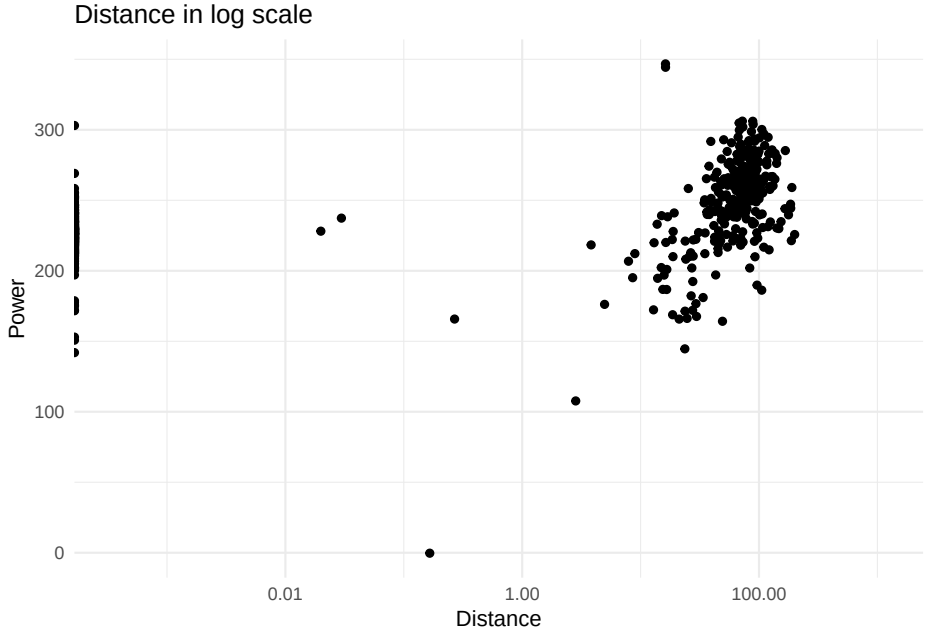
5.3.1.1 Graphical exploration

To visualize the relationship between two numerical variables, we can use the scatter plot. Don't forget that the log function can help you identify non linear relationships since $log(a \cdot x^b) = log(a) + b \cdot log(x)$

```
ggplot(dat,aes(distance,avgPower)) + geom_jitter() +
  labs(x="Distance",y="Power",title="Raw variables") + scale_x_continuous(labels = scal
```

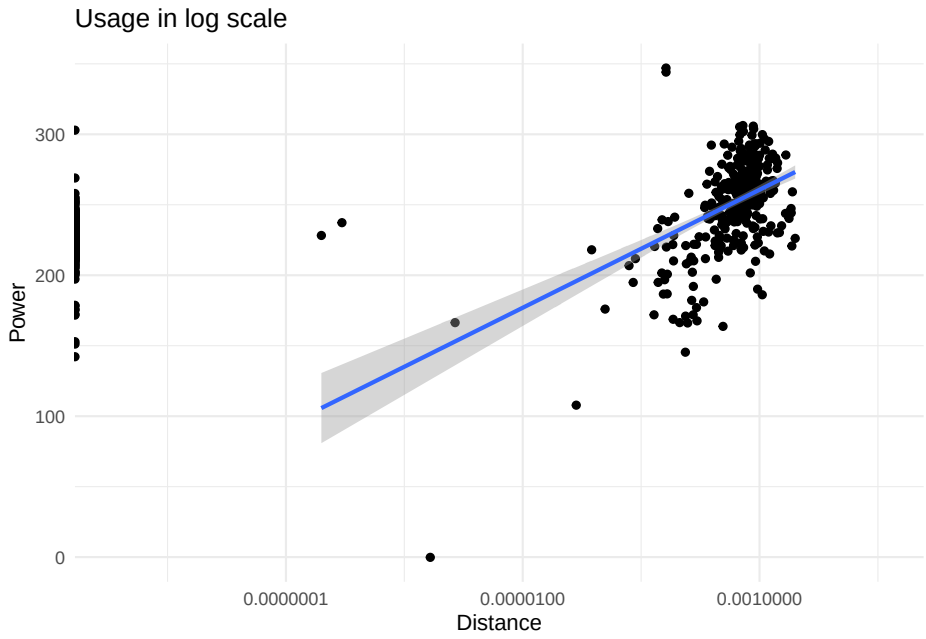



```
ggplot(dat,aes(distance,avgPower)) + geom_jitter() + scale_x_log10(labels = scales::co
  labs(x="Distance",y="Power",title="Distance in log scale") + theme_minimal()
```



We can pimp up the graphics a bit to visualize the correlation

```
ggplot(dat,aes(distance/1E5,avgPower)) + geom_jitter() + scale_x_log10(labels = scales::comma) +
  labs(x="Distance",y="Power",title="Usage in log scale")+  geom_smooth(method="lm") + theme_mini
```



We can see here that there is a positive relationship between distance and data usage and that this relationship has an exponential shape, meaning that the usage increases A LOT when the age drops.

5.3.1.2 Quantifying the relationship : correlations

To quantify this relationship, you can use the coefficients of correlation. There are 3 main coefficients : Pearson (the most famous and used), Kendall and Spearman. The latter can handle non-linear functional dependencies (ranks correlation) ; this is (roughly) equivalent to computing the coefficients on the log-transformed variables.

```
# Pearson coeff
cor(dat$distance,dat$avgPower,method="pearson")
```

```
## [1] NA
```

```
# Pearson coeff, NAs removed
cor(dat$distance,dat$avgPower,method="pearson",use = "complete.obs")
```

```
## [1] 0.5437859
```

```
# Spearman coeff
cor(dat$distance,dat$avgPower,method="spearman",use = "complete.obs")
```

```
## [1] 0.6316721
```

```
# Why is R a beautiful language ? Do it at once, without loops (loops are evil)
print("all coeffs")
```

```
## [1] "all coeffs"
```

```
sapply(c("pearson","spearman","kendall"),function(xx) cor(dat$distance,dat$avgPower,me
```

```
##   pearson  spearman   kendall
## 0.5437859 0.6316721 0.4452121
```

More info about correlation coefficients

Should there be a complex relationship (eg sine), the graphical exploration is mandatory !

5.3.2 2 categorical variables

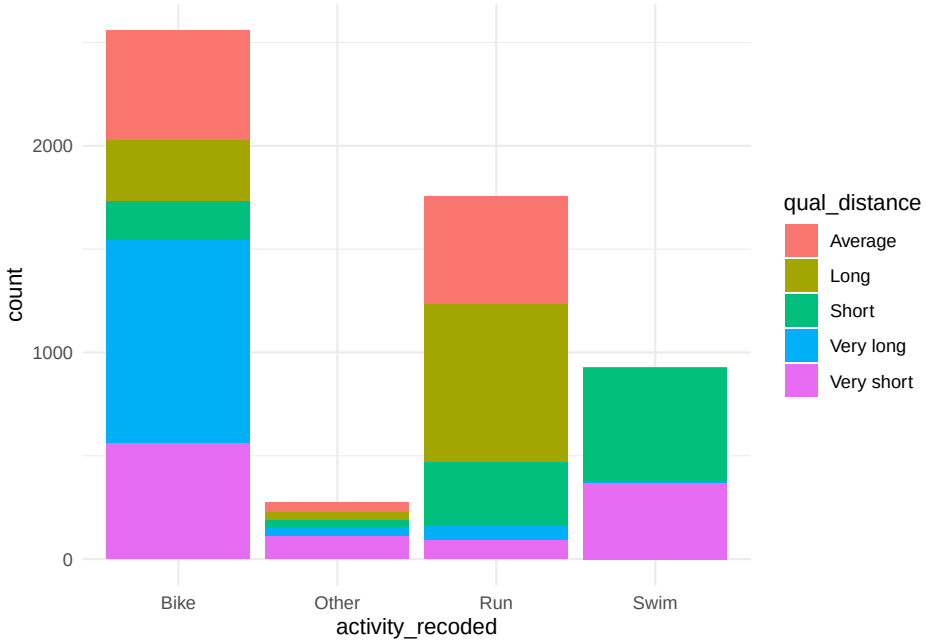
For this part, I will create a discrete variable out of the distance variable (see previous exercises) to use it as second qualitative variable (the other ones are not really meaningful)

```
dat <- mutate(dat,qual_distance=as.character(cut(distance,
                                                 quantile(distance,probs = 0:5/5,na.rm=
                                                 include.lowest = T,
                                                 labels=c("Very short","Short",
                                                          "Average","Long","Very long"
              qual_avgHr=as.character(cut(avgHr,quantile(avgHr,0:3/5,na.rm = T),
                                          include.lowest = T,
                                          labels=c("Low intensity","Average intensity"
                                                   "High intensity"))),
              qual_distance=ifelse(is.na(qual_distance),"Very short",qual_distance))
```

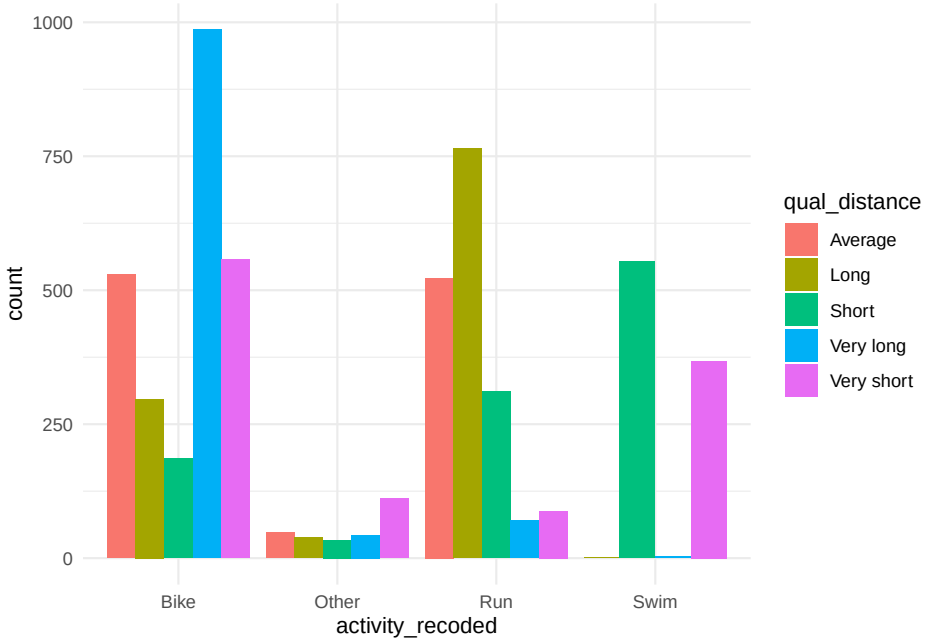
Try different layouts with you barcharts !

5.3.2.1 Barcharts

```
ggplot(dat,aes(activity_recoded,fill=qual_distance)) +
  geom_bar(position = "stack") + theme_minimal()
```

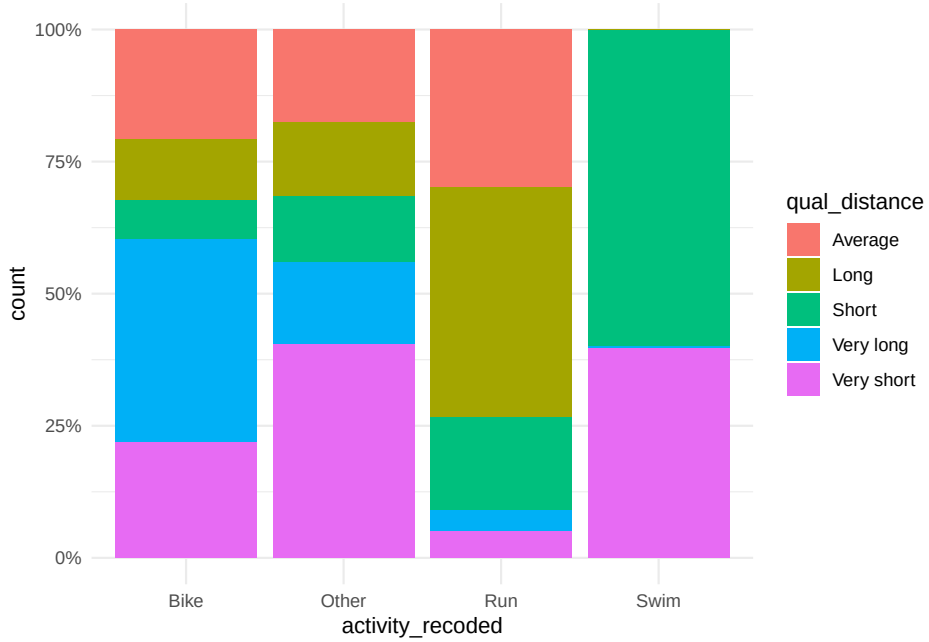



```
ggplot(dat,aes(activity_recoded,fill=qual_distance)) +
  geom_bar(position = "dodge") + theme_minimal()
```




```
ggplot(dat,aes(activity_recoded,fill=qual_distance)) +
```

```
  geom_bar(position = "fill") +
  scale_y_continuous(labels = scales::percent) + theme_minimal()
```



You get really different insights depending on the representation you chose !

5.3.2.2 Contingency tables

```
table(dat$activity_recoded,dat$qual_distance)
```

```
##
##         Average Long Short Very long Very short
##   Bike      530  297   186       986        558
##   Other      48   39    34        43        111
##   Run       523  764   311        70         88
##   Swim        0    1   554         3        368
```

```
table(dat$activity_recoded,dat$qual_distance) %>%
  prop.table()*100
```

```
##
##             Average        Long       Short   Very long  Very short
##   Bike   9.61189699  5.38628945  3.37323177 17.88175553 10.11969532
##   Other  0.87051143  0.70729053  0.61661226  0.77983315  2.01305767
##   Run    9.48494741 13.85564019  5.64018861  1.26949583  1.59593761
##   Swim   0.00000000  0.01813565 10.04715270  0.05440696  6.67392093
```

```
table(dat$activity_recoded,dat$qual_distance) %>%
  prop.table(margin = 1)*100
```

```
##
##            Average       Long      Short  Very long Very short
##   Bike  20.7274149 11.6151740  7.2741494 38.5608135 21.8224482
##   Other 17.4545455 14.1818182 12.3636364 15.6363636 40.3636364
##   Run   29.7835991 43.5079727 17.7107062  3.9863326  5.0113895
##   Swim   0.0000000  0.1079914 59.8272138  0.3239741 39.7408207
```

```
table(dat$activity_recoded,dat$qual_distance) %>%
  prop.table(margin = 2)*100
```

```
##
##             Average        Long       Short   Very long  Very short
##   Bike  48.13805631 26.97547684 17.14285714 89.47368421 49.60000000
##   Other  4.35967302  3.54223433  3.13364055  3.90199637  9.86666667
##   Run   47.50227066 69.39146231 28.66359447  6.35208711  7.82222222
##   Swim   0.00000000  0.09082652 51.05990783  0.27223230 32.71111111
```

5.3.2.3 Quantifying relationships : χ^2, Cramer's V

The Chi-square (χ^2) statistics is used to measure the distance between the actual distribution of cases among categories of both variables and the distribution if the variables were independant. The higher the `X-squared`, the higher the divergence with independance, meaning that the variables are likely *linked* (correlation does not apply to categorical variables). The `p-value` indicates whether this relationship is **statistically significant** or not. We will see this in more details in the last chapter (Inference).

More info and detailed way to compute the value : this website

5.3.2.4 Extreme examples :

Let's assume we want to assess the relationship between gender and churn. Further assumption, we have 100 customers, 50 males, 50 females on the one hand, 50 churners, 50 non churners on the other hand.

- If the variables are independent, the cross table would look that way :

```
##
##  Pearson's Chi-squared test
##
## data:  .
## X-squared = 0, df = 12, p-value = 1
```

- In the opposite situation (full dependency), the contingency table would look like that :

```
##       Short Very long Long Average
## Swim     25         0    0       0
## Bike      0        25    0       0
## Run       0         0   25       0
## Other     0         0    0      25
```

The χ^2 statistic measures the "distance" between reality and the first case (independence)

Note : if some cells of the contingency table have less than 5 cases, the statistic is not reliable (you'll get a message in this case)

The chi-square suffers 2 main drawbacks : its value depends on the number of observations and the total number of categories $\Rightarrow$ one cannot compare the χ^2 values for 2 different tables that have different numbers of underlying observations and number of categories. To deal with that, you can use Cramer's V, which is a (kind of) *normalized* χ^2. You can use the function built in the `lsr` package. Cramer's V $\in [0, 1]$ and the higher it is, the more intense the link between both variables.

```
# install.packages("lsr") # If not installed
table(dat$activity_recoded,dat$qual_distance) %>%
  lsr::cramersV()
```

```
## [1] 0.4454809
```

Let's check with our 2 extreme examples :

```
lsr::cramersV(ex_dep[,-5])
```

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

```
lsr::cramersV(ex_indep)
```

```
## [1] 0
```

In practice, it is very rare to get high values ; a rule of thumb is that a value around 0.2-0.3 is already "decent". The χ^2 p-value (if under 0.05) shows that there is a relationship ; Cramer's V allows to compare between two tables.

5.3.3 1 continuous, 1 categorical variable

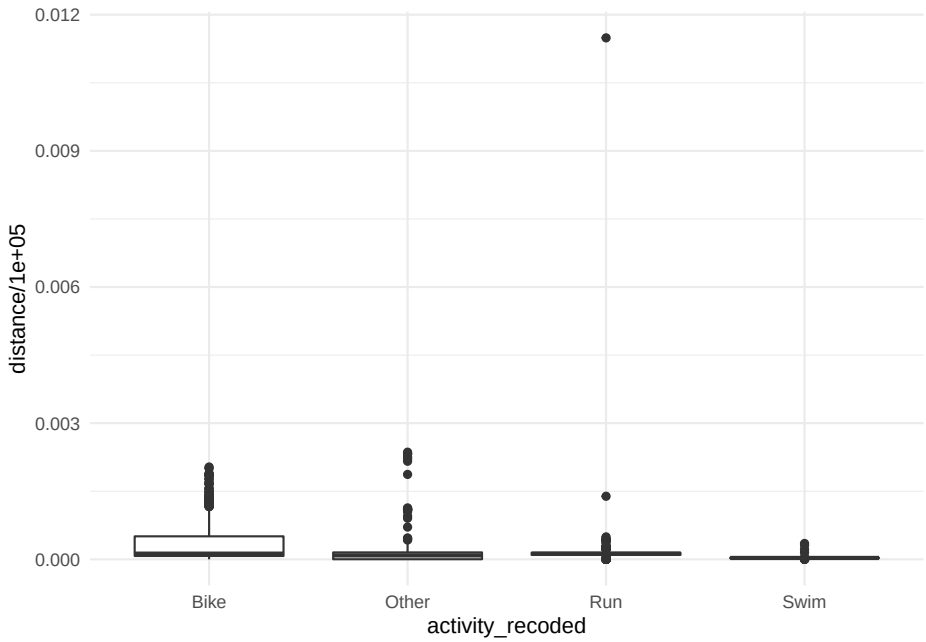
In this part, we see how to deal with 2 variables that have different types. The goal remains the same : getting insights about the relationship between those 2 variables and quantify the strength of the link. We will try to assess if there is a connection between the distance and the discipline.

5.3.3.1 Boxplots, violin plots

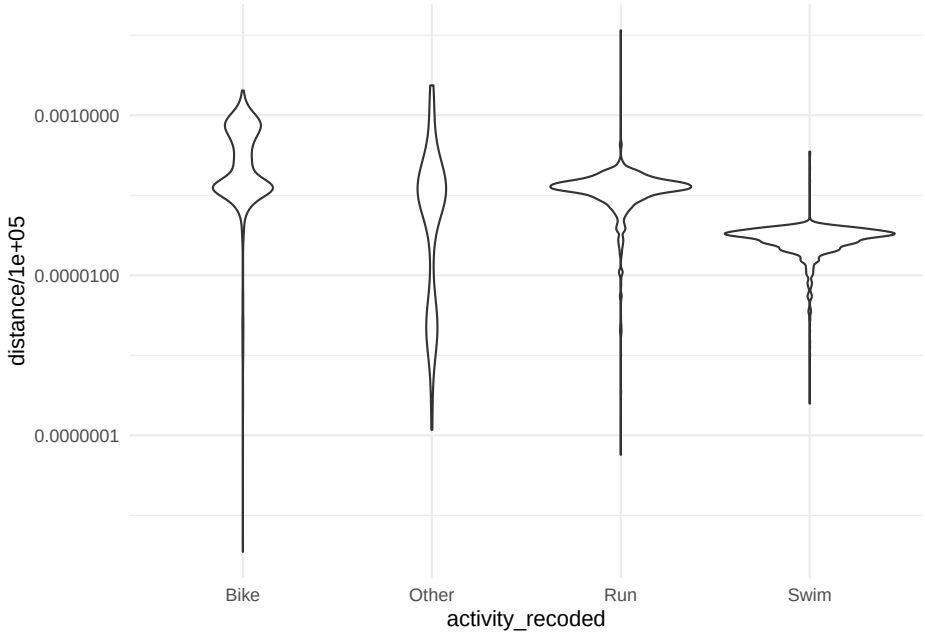
Boxplots are a simple, effective and compact representation of a variable's distribution. It relies on quantiles. Vilin plots allows to see the ful distribution of

both variables

```
# Compute bounds of the boxplot
# bounds <- group_by(dat,activity_recoded) %>%
#   summarise(q1=quantile(distance,.25,na.rm = T),
#             q2=quantile(distance,.50,na.rm = T),
#             q3=quantile(distance,.75,na.rm = T),
#             lower_bound=q1-1.5*IQR(distance,na.rm = T),
#             upper_bound=q3+1.5*IQR(distance,na.rm = T)) %>%
#   pivot_longer(-activity_recoded)
ggplot(dat,aes(activity_recoded,distance/1E5)) + geom_boxplot() + theme_minimal()
```




```
  # geom_point(data = bounds,aes(activity_recoded,value,color=name))
ggplot(dat,aes(activity_recoded,distance/1E5)) + geom_violin()+ theme_minimal() + scale_y_log10(l
```



It looks like bike activities are longer than the others ! Big surprise !

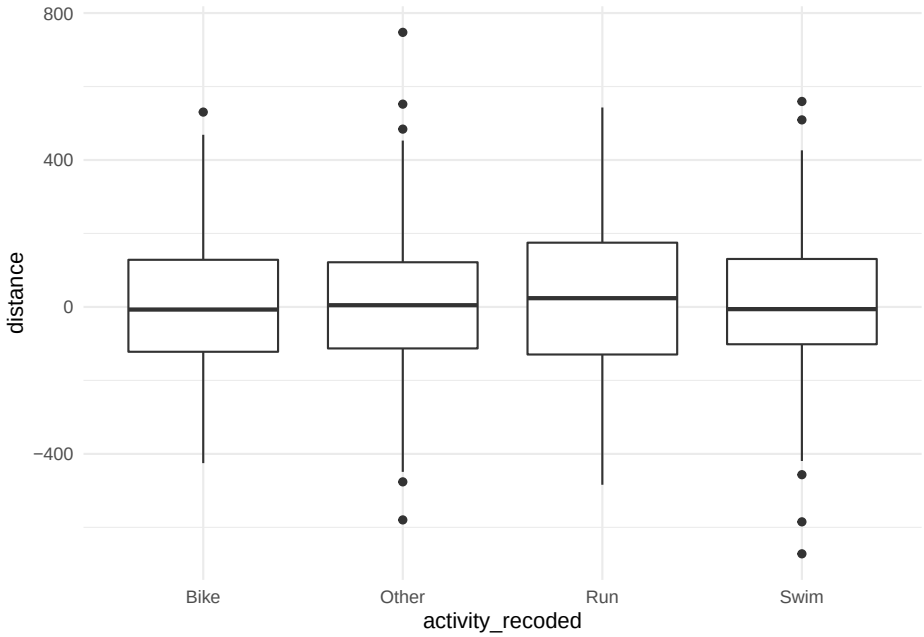
5.3.3.2 Quantifying relationship intensity : η^2

The graphics indicate that there is a relationship between activity type and distance (if not, boxplots would have the same shape for all groups). We can assess the strength of the connection with the η^2 statistics.

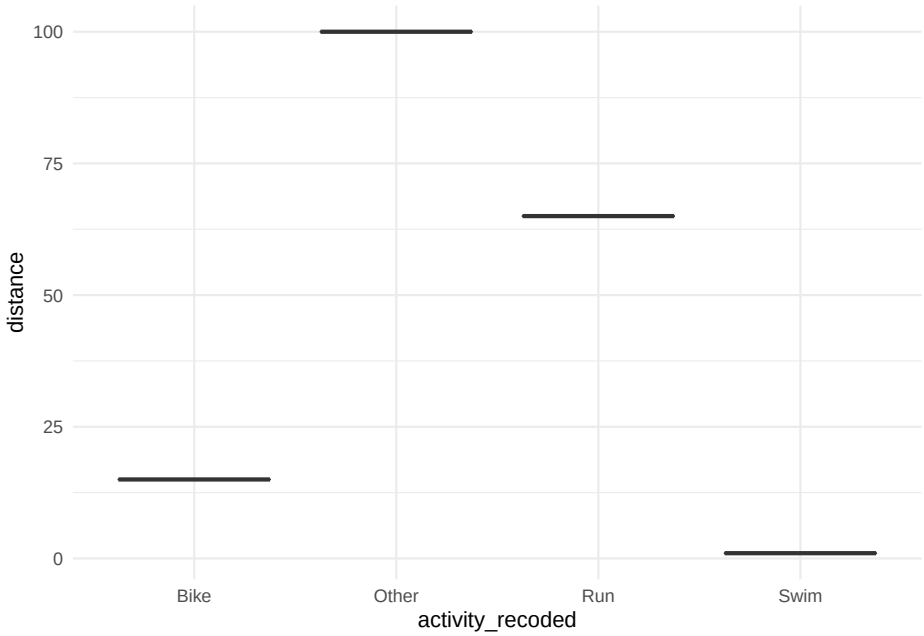
Using the decomposition of the variance formula $SS_{total} = SS_{between} + SS_{within}$, η^2 is defined as $\eta^2 = \frac{SS_{between}}{SS_{total}} \in [0, 1]$

5.3.3.3 Extreme examples :

- If the variables are independent, the boxplots should look like this (almost no difference in the distributions) :



- If they are fully "correlated", the activity variable would explain all variance in the data set :



In this case, we see that all the variance lies between the subgroups : there is no dispersion within the groups.

Note : In practice, the previous situation will of course never happen, and a categorical variable can't carry by itself a lot of variance (since the number of possible values are de facto limited).

This is also the R^2 of the 1-factor ANOVA regression of distance explained by activty type,

```
anova <- aov(distance~activity_recoded,data=dat)
# Variance decomposition
summary(anova)
```

```
##                    Df  Sum Sq Mean Sq F value Pr(>F)
## activity_recoded    3  610577  203526   230.6 <2e-16 ***
## Residuals        5503 4857888     883
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 7 observations deleted due to missingness
```

```
print("eta squared")
```

```
## [1] "eta squared"
```

```
lsr::etaSquared(anova)
```

```
##                      eta.sq eta.sq.part
## activity_recoded 0.1116542   0.1116542
```

```
# Alternatively
lm(distance~activity_recoded,data=dat) %>% summary()
```

```
##
## Call:
## lm(formula = distance ~ activity_recoded, data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
##  -29.98  -17.47   -1.23    1.37 1136.08
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)            29.9816     0.5877  51.017  < 2e-16 ***
## activity_recodedOther -11.9153     1.8950  -6.288 3.47e-10 ***
## activity_recodedRun   -17.0613     0.9214 -18.517  < 2e-16 ***
## activity_recodedSwim  -27.0224     1.1396 -23.712  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 29.71 on 5503 degrees of freedom
##   (7 observations deleted due to missingness)
```

```
## Multiple R-squared:  0.1117,Adjusted R-squared:  0.1112
## F-statistic: 230.6 on 3 and 5503 DF,  p-value: < 2.2e-16
```

In this case, 11.2% of the age variance is explained by the difference in activity types ; it is very high.

5.3.4 Exercises

- Explore the distribution of the average speed. What can you say about it ?
- Explore the correlation between average speed and average power
- For all the the categorical variables, get the frequent category (with table AND dplyr/tidyr)

Very important : For the next parts, we will remove the extreme observation that is clearly an error

```
dat_clean <- filter(dat,!(activityId %in% c(407226313,2321338)) & year(date)>=2012)
```

5.4 Statistical inference

5.4.1 The statistical model

We want to measure a characteristic in the *general population*, let's say the average distance of all potential activites, and let's denote it by D.

The fundamental assumption of the statistical model is that there is an underlying *data-genereting process*, which means that D is distributed with a certain probability distribution. The goal of the statistician is to find which distribution it is, and estimate its parameters.

The big problem is that it is impossible to observe D on the whole population, and any dataset is only a **sample of the general population** (which does not really exists). The question is then : how can we estimate the parameters of the *true* distribution ?

$\Rightarrow$ There is a difference between the **sample mean** and the **population mean** (noted μ). As a matter of fact the sample mean is an estimator of the population mean. The value of an estimator (often noted $\hat{\theta}$) is a random variable (it depends on the sample), meaning this is not a single deterministic value, but has a probability distribution. Therefore it has an **expectation** and a **variance** An estimator is said to be *biased* if $\mathbb{E}(\hat{\theta}) \neq \mu$; it is said to be *efficient* if its variance is minimal.

One fundamental hypothesis of the model is that all observations are independent and identically distributed (*iid*). This is typically not the case for *time series*, but this hypothesis is, in general, reasonable.

5.4.2 Two fundamental theorems

Eventough we cannot observe the true parameter(s), and that the sample mean is an estimator (hence a random variable), 2 theorems save the game :

5.4.2.1 The law of large numbers

$$\bar{D} = \frac{1}{n}\sum_{i=1}^{n} D_i \xrightarrow[n\to+\infty]{a.s.} \mu$$

In other words, when the sample size n is big enough, the sample mean converges to the population mean → We can estimate this parameter with a simple mean without **bias**.

5.4.2.2 The central limit theorem (CLT)

Probably the most important theorem in statistics, valid whatever the true distribution is

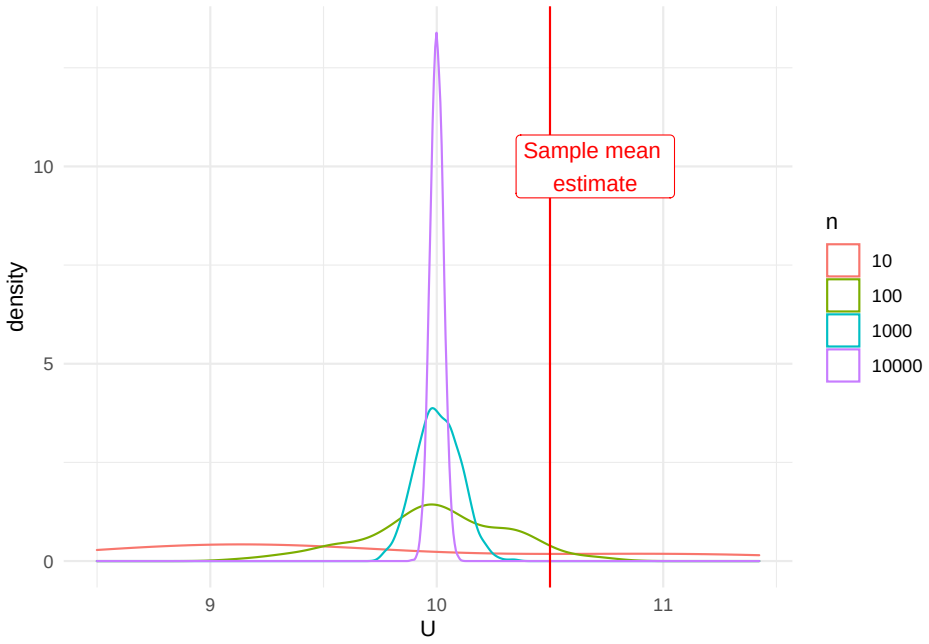
$$\sqrt{n} \cdot \bar{D} \xrightarrow[n\to+\infty]{p} \mathcal{N}(\mu, \sigma^2)$$

Meaning that the sample mean converges in probability to a normal distribution with population parameters at "speed" $\sqrt{n}$. This is equivalent to :

$$\bar{D} - \mu \xrightarrow[n\to+\infty]{p} \mathcal{N}(0, \frac{\sigma^2}{n})$$

Meaning that :

- I can quantify "how far" my sample mean is from the true value
- The larger the sample size, the smaller the average deviation to the true value → the variance of my estimator reduces when the sample size increases.



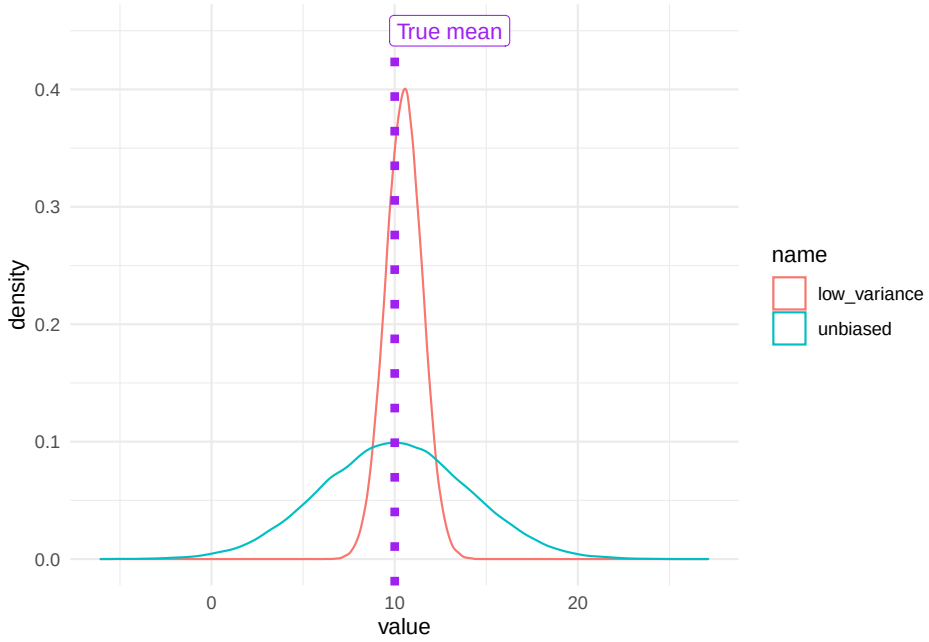
Take away :

- The **sample mean** is an **estimator** of the true value of an underlying "true" mean. The estimator's value depends on the sample I have
- This estimator (any estimator) has a **variance** that I could measure if I had several samples to compute several sample means
- Probality theory gives us tools to estimate the **bias** and **the variance** of an estimator
- **Bias-variance trade-off** :$\mathbb{E}((D - \bar{D})^2) = \mathbb{E}^2(D - \bar{D}) + \mathbb{V}(\bar{D})$, in other words : $MSE_{\bar{D}} = bias^2 + \mathbb{V}(\bar{D}) \rightarrow$ see you during ML course ;-)

Illustration of the biais-variance trade-off.

Let's assume the **true** average of the usage is 10, what do you prefer over the following scenarios ? Let's simulate two distributions (let's say it is the distribution of 2 different estimators) :

- One with mean 10 and variance 4 $\rightarrow$ unbiased
- The second with mean 10.5 and variance 1 $\rightarrow$ biased but with low variance

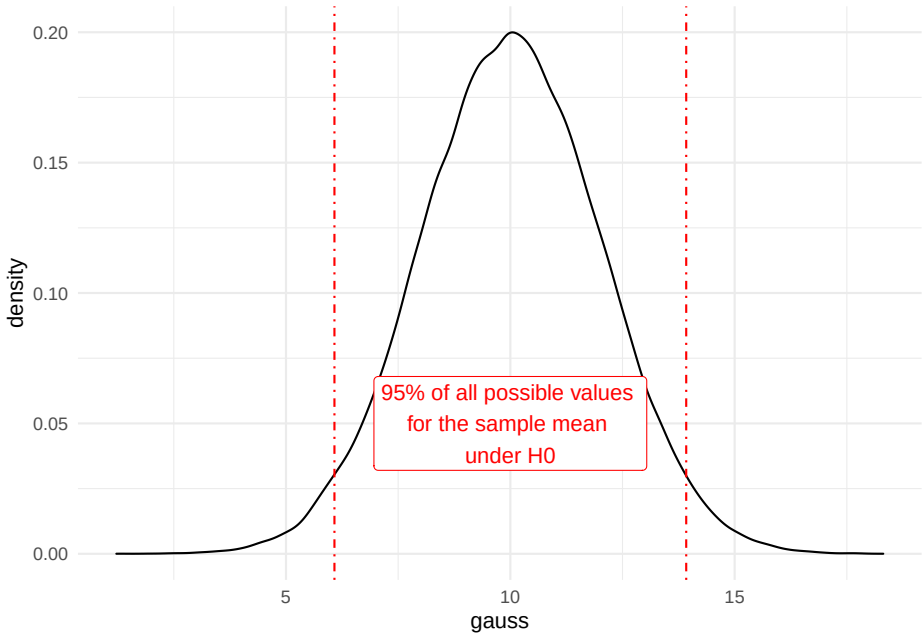


In the first case, the estimator is unbiased, but with a higher variance than the second : if we go for it, we take the chance to have an estimate (depending on our sample) of eg 15 or 5, which is almost unlikely to happen with the second estimator, although this second is not centered on the true value. It is up to you to decide, but you generally can't have both an unbiased and very precise estimator…

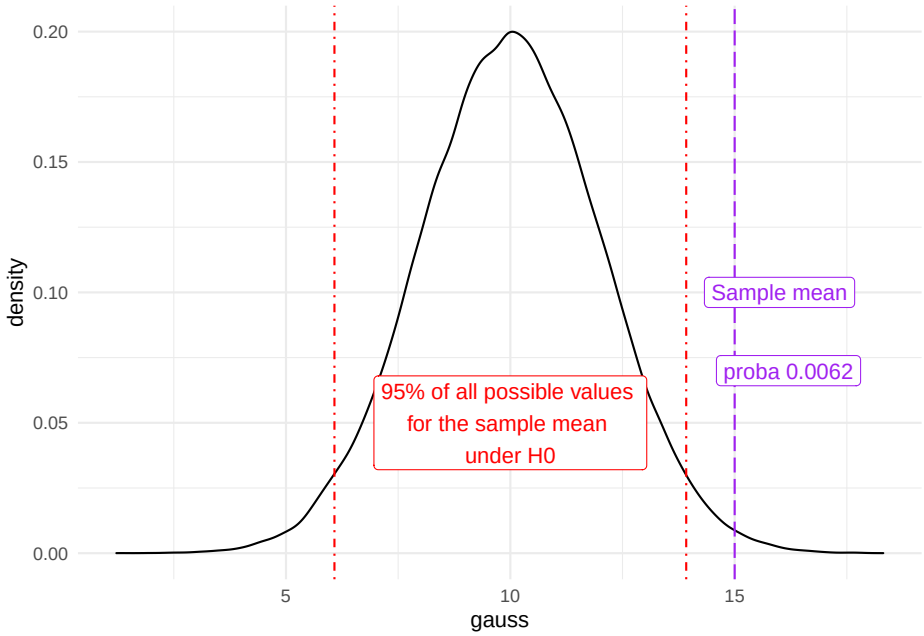
5.4.3 Statistical tests

5.4.3.1 Introductory example

Knowing the theoretical probability distribution of our estimator, we can assess the likelihood of an hypothesis. For the example, let's make the hypothesis ($\mathcal{H}_0$) that the true mean is 10 and standard deviation is $2\sqrt{n}$. If this hypothesis is true, thanks to the CTL, the distribution of $\bar{D}$ would be the following :



Now, I can compute my sample mean and check its value against the hypothetical distribution :



Obviously my actual value doesn't fit with my hypothesis : the probability of getting such a sample mean under $\mathcal{H}_{\mathcal{O}}$ is very small $\rightarrow$ my hypothesis is very

unlikely to be valid → the true mean is probably not 10.

5.4.3.2 Student test

A test is defined by its *null hypothesis* $\mathcal{H}_0$, the contrapositive (alternative) being $\mathcal{H}_1$. The general procedure is to set $\mathcal{H}_0$ such that we can build a *test statistic* of which we can derive the distribution.

In general, statisticians chose a null hypothesis such that they can build a statistic for which they know the distribution → they can compute the probability of a specific value to occur.

For the example, we'll focus on the **Student test**. The test is meant to check whether the true value of the mean is equal to a specific value. Example, I want to test whether the average distance for all activities is 20km/h. Our test is the following :

- $\mathcal{H}_0$: The average distance of activities is 20
- $\mathcal{H}_1$: The average distance of activities is not 20

The next important parameter of a test if α, the risk level, meaning the probability we are willing to take to be wrong while accepting $\mathcal{H}_0$. 5% is a value that is often chosen.

Thanks to the CLT, we can build the following test statistic :

$$T = \sqrt{n}\frac{\bar{D} - 20}{\sigma} \hookrightarrow \mathcal{N}(0,1)$$

Problem : we don't know the value of the true standard deviation. The final test statistic is distributed with a Student distribution :

$$T = \sqrt{n}\frac{\bar{D} - 20}{\hat{s}} \hookrightarrow \mathcal{S}t_{n-1}$$

where $\hat{s}^2 = \frac{1}{n-1}\sum_{i=1}^{n}(D - \bar{D})^2$ is the unbiased estimator of the variance.

```
t.test(dat$distance,mu=20)
```

5.4.3.2.1 Implementation in R :

```
##
##  One Sample t-test
##
## data:  dat$distance
## t = -1.3698, df = 5506, p-value = 0.1708
## alternative hypothesis: true mean is not equal to 20
```

```
## 95 percent confidence interval:
##  18.58573 20.25079
## sample estimates:
## mean of x
##  19.41826
```

5.4.3.2.2 Interpretation : If you have one thing to remember : **the p-value is the probability to be wrong while rejecting $\mathcal{H}_0$.** In our case, this probability is very small, meaning that we should not consider that the average distance of all activities is 20km/h.

In short : all you have to know is what the null hypothesis is, and make your decision depending on the p-value. In general, we reject the null is p-value $<$ 5% (the risk level), but you can decide to be more demanding and chose a lower threshold if you don't want to make a mistake.

Note : If you want to dig deeper, you can test whether the mean is strictly greater than a specific value using the parameter `alternative` of the `t-test` function.

5.4.3.3 Student test to compare group means

You can also use the Student test to check whether the means of two sub-populations are equal or not. In this case, the tests checks whether the difference in means differs significantly from 0. We'll check in this example if average distance differs between rides and other activities. For that we have to extract on the one hand the bike activities' distance and the other activites' distance on the other hand. We can run the test on those 2 vectors :

```
bike <- filter(dat,is_bike) %>% pull(distance)
non_bike <- filter(dat,!is_bike) %>% pull(distance)
t.test(bike,non_bike)
```

```
##
##  Welch Two Sample t-test
##
## data:  bike and non_bike
## t = 23.825, df = 4586.7, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  18.09061 21.33476
## sample estimates:
## mean of x mean of y
##  29.98157  10.26888
```

In this case, the null hypothesis is "the difference in means is 0" and the p-value is very very small (almost 0) $\rightarrow$ we can reject the null without second

thoughts, which means there is a **significant** difference between the 2 subgroups regarding average distance.

5.4.3.4 Back to our χ^2

Remember the χ^2 test is used to assess if 2 categorical variables are independent or not. The null hypothesis in this test is "both variable are independant". To check that, a test statistic, D^2 (X-squared in R output) is built, and under $\mathcal{H}_0$, it is distributed with a χ^2 probability distribution. We can can then test the validity of the null depending on the test value.

Let's check if there is a relationship between activity and distance in bins :

```
tab <- table(dat$qual_distance,dat$activity_recoded)
tab
```

```
##
##              Bike Other Run Swim
##   Average     530    48 523    0
##   Long        297    39 764    1
##   Short       186    34 311  554
##   Very long   986    43  70    3
##   Very short  558   111  88  368
```

```
tab %>% chisq.test()
```

```
##
##  Pearson's Chi-squared test
##
## data:  .
## X-squared = 3282.8, df = 12, p-value < 2.2e-16
```

In this case, the p-value is again much smaller than 5% $\Rightarrow$ the probability to be wrong by rejecting the null is again very small... It is hence reasonable to reject the null and we can consider that the two variables are independent, meaning there is a connection between the activity type and the distance.

5.4.4 Other estimators : maximum of likelihood

We saw that the sample mean is a good estimator for the population mean. If you assume that the underlying probability distribution of the variable of interest is something else than a normal distribution, you can be interested in estimating another parameter than the mean.

Let's say we want to estimate the s parameter of the Zipf's law.

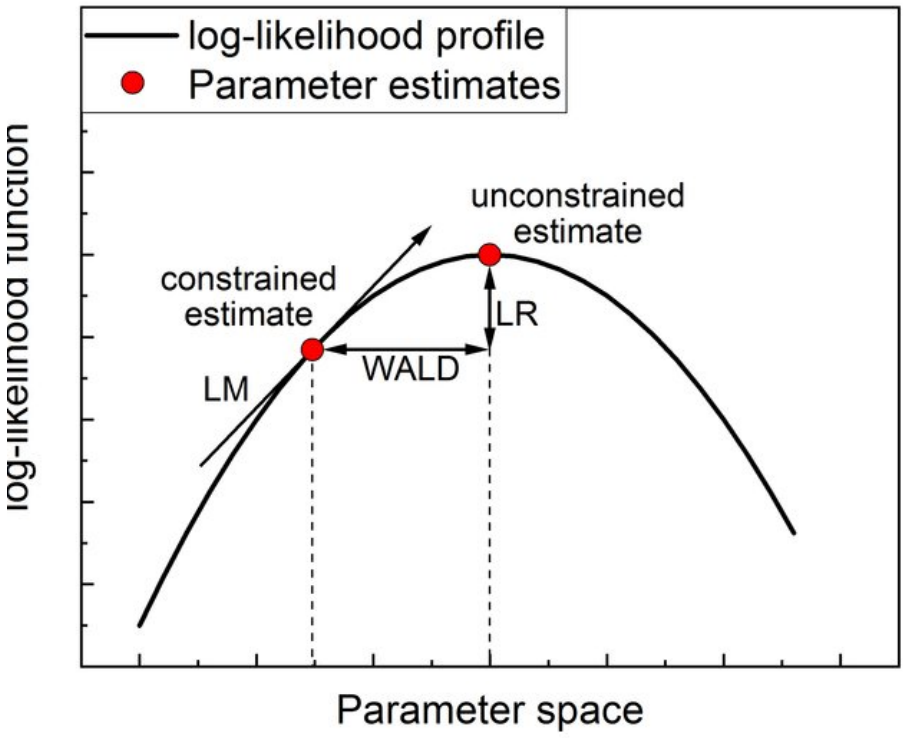
In this case, you can use the **maximum likelihood estimator** (MLE). The likelihood is the joint probability of observing the sample I got (which is the product of individual probabilities under the iid hypothesis).

$$L_X(s) = \prod_{i=1}^{n} f(k; s, N)$$

Since we state that the underlying probability distribution is Zipf's law, we can express the likelihood as a function of the s parameter. The idea of the MLE is since this sample happened, it was the *most likely* to happen, hence the value of s is such that it maximizes the likelihood for my sample. And finding the maximum value is something for which we have a few algorithms !

For this estimator, we also know some asymptotic properties that allows to build tests, which you can interpret in the same manner. Namely, you have three tests :

- Wald test
- Likelihood ratio test
- LM test (Lagrange multiplier)



All those test statistics are distributed as a χ^2 distribution

MLE is also used for machine learning/econometrics, especially when relationships are non linear $\Rightarrow$ logistic regression.

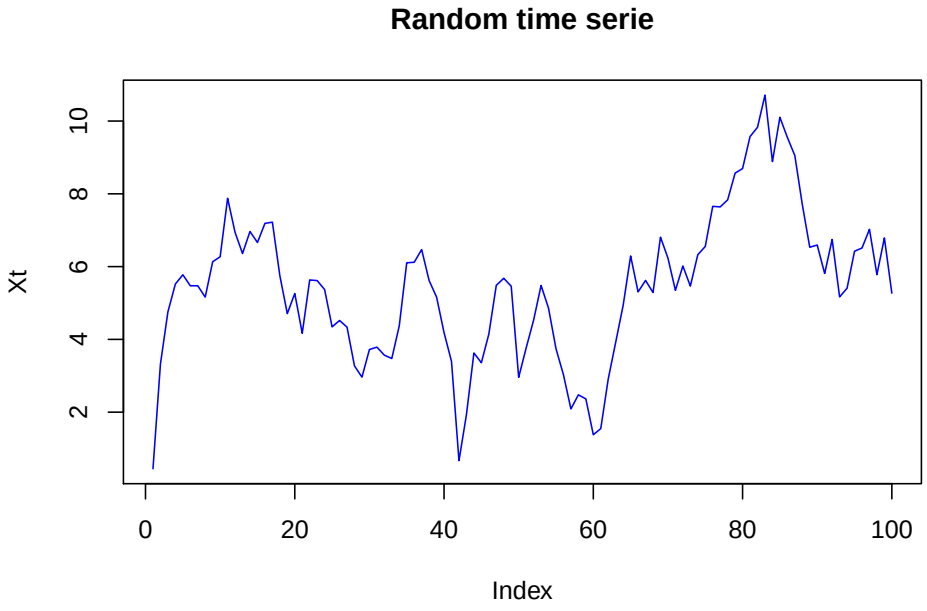
Notes :

- Sample mean is the MLE for the exponential family (eg gaussian distribution)
- MLE is a subset of M-estimators

5.4.5 Exercises : interprete a test you don't know

- Run a Kolmogorov–Smirnov test on the distance variable, and the logarithm of this variable. What can you conclude ?
- In time series analysis, it is crucial to check if a time series is stationary or not. Stationarity means that both mean and variance are constant over time (no drift). If it's the case, it's easier to model it. To check that, there are several tests called "unit-root" tests (if there is a unit root, the serie is not stationary). In this example, I simulate a time series and run the Phillips-Perron test. The alternative hypothesis is specified in the output ; please interpret the result.

```
library(tseries)
# generation of a random walk
Xt <- cumsum(rnorm(100))
plot(Xt,type="l",main="Random time serie",col="blue")
```



```
pp.test(Xt)
```

```
##
##  Phillips-Perron Unit Root Test
##
## data:  Xt
## Dickey-Fuller Z(alpha) = -17.901, Truncation lag parameter = 3, p-value
## = 0.09351
## alternative hypothesis: stationary
```

Chapter 6

Multivariate analysis and dimension reduction

6.1 Multivariate analysis

6.1.1 Advanced visualization

6.1.1.1 The grammar of graphics

The grammar of graphics was introduced in 2005 by Wilkinson and Leland as a general framework for graphical representation of data. It was adapted by Hadley Wickham in the R package `ggplot2`. *It presents a unique foundation for producing almost every quantitative graphic found in scientific journals, newspapers, statistical packages, and data visualization systems.*

You can check the paper from Wickham

When ploting data, one has to define :

- What are the aesthetics → **the dimensions you want to represent**
- What is the geometry you want to use → **the kind of plot you want**
- Ploting options (that come with default values) :
  - scales of the axis
  - fonts and colors for text
  - labels (title, axis titles…)

In short, what you have to find is the right combination of aesthetics and geometry that best represent the data. To find the recommended combination, don't forget to use From data to viz

6.1.1.2 Playing with aesthetics

You must define at least 1 dimension for the plot, either continuous or categorical for the x axis. Then you can increase the number of dimensions (ie columns of the data frame) you want to represent :

- x and y for the axis
- size : optional (integer) dimension that will represent an additional number
- color/fill : optional (categorical) dimension reprensented by a color. Color is for line/point geoms, fill for bars/heatmap geoms
- linetype : optional (categorical) : different type of lines (solid, dotted, dashed…). Only for geometries using lines
- shape : optional (categorical) : variable that will make the shape of the dot vary. Only for point geometries
- alpha : optional (continuous) : the transparency of the dots (the lower the value, the more transparent the dot)
- …

More about the aesthetics here

You can also use the `facet_wrap()` or `facet_grid()` functions to add up to 2 more dimensions with categorical variables (see demo)

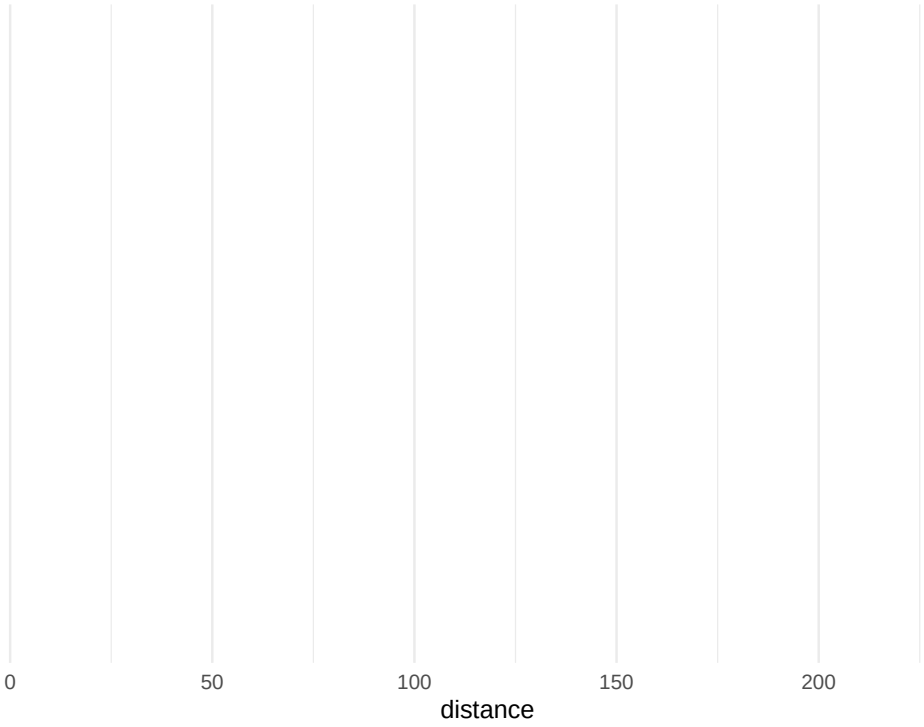
6.1.1.3 Geometries

Once you chose the variables (dimensions) you want to plot, you have to chose the geometry, that highly depends on the type of variable (numerical or character/factor). Don't forget to reffer to the website *from data to viz* if you need some inspiration.

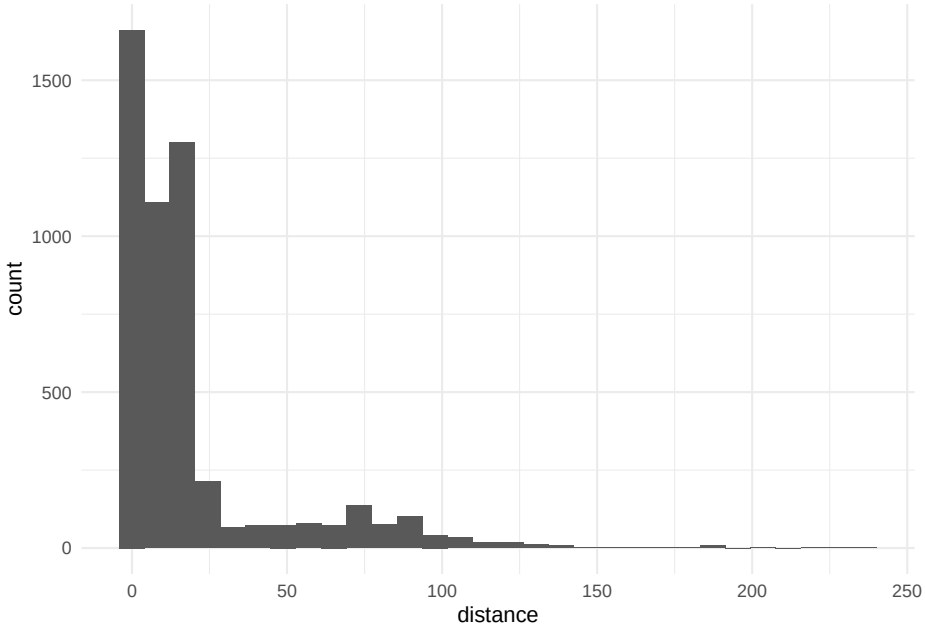
A short list of most common geometries :

- geom_bar for barplots
- geom_histogram for histograms
- geom_jitter for scatter plots
- geom_boxplot & geom_violin for compared density plots
- geom_tile for heatmaps
- geom_line for time series
- geom_text or geom_label for text (annotations)
- geom_hline and geom_vline for horizontal and vertiacal lines
- ……

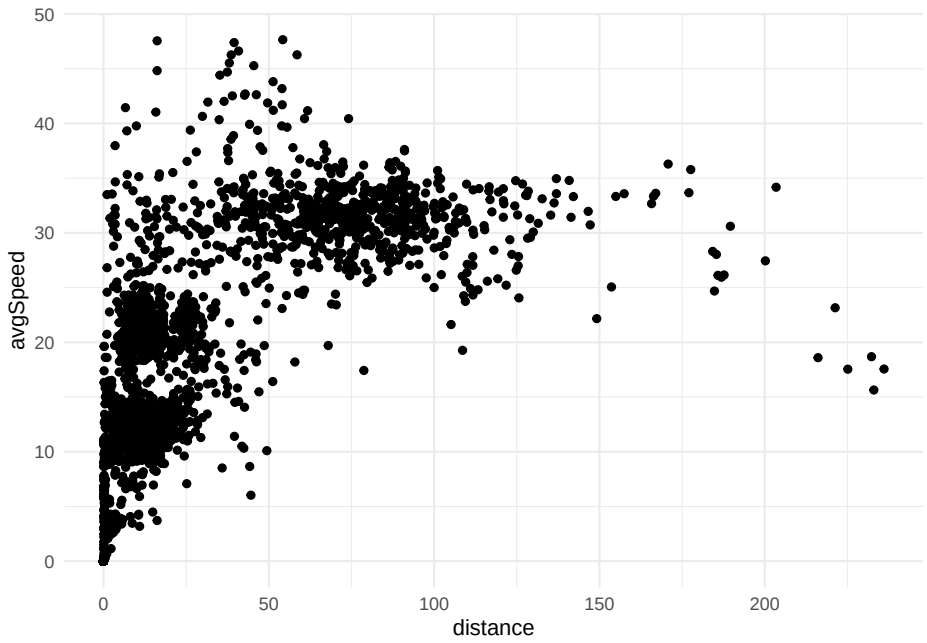
```
# From
ggplot(dat_clean,aes(x=distance)) +
  theme_minimal()
```



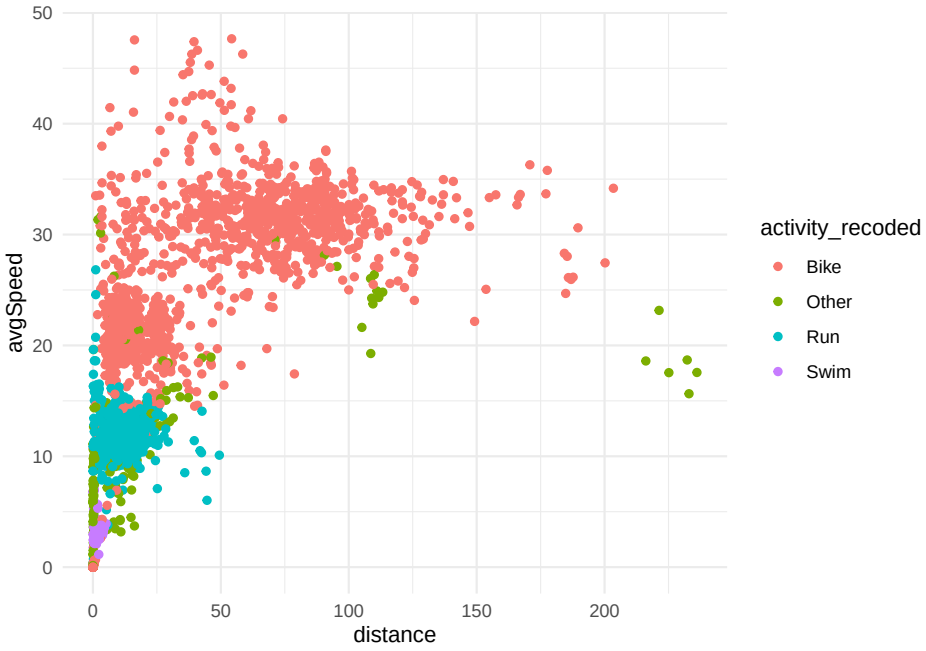

```
ggplot(dat_clean,aes(x=distance)) + geom_histogram()+
  theme_minimal()
```



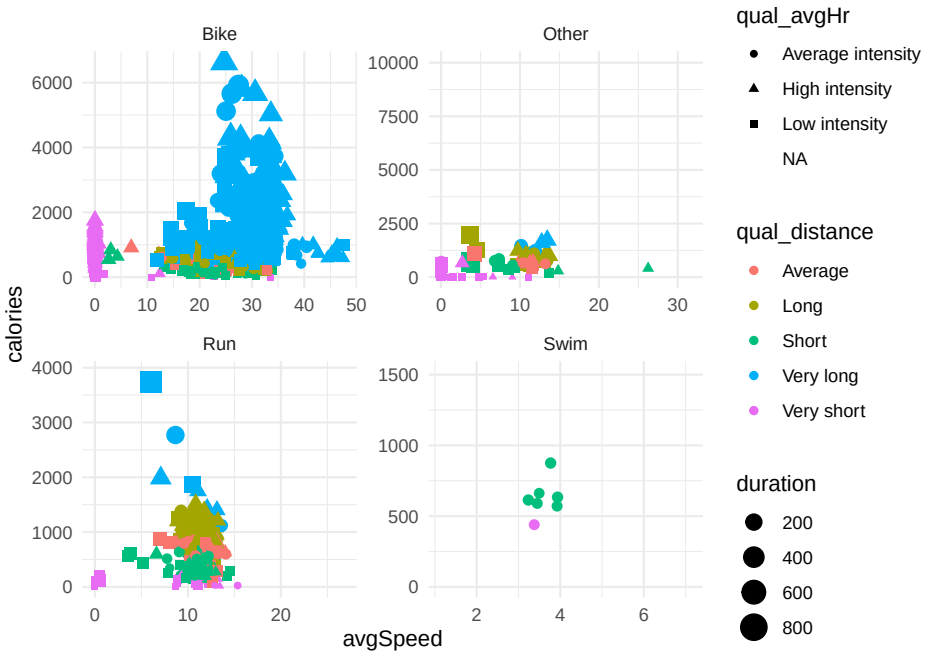
```
ggplot(dat_clean,aes(distance,avgSpeed)) + geom_jitter()+
  theme_minimal()
```



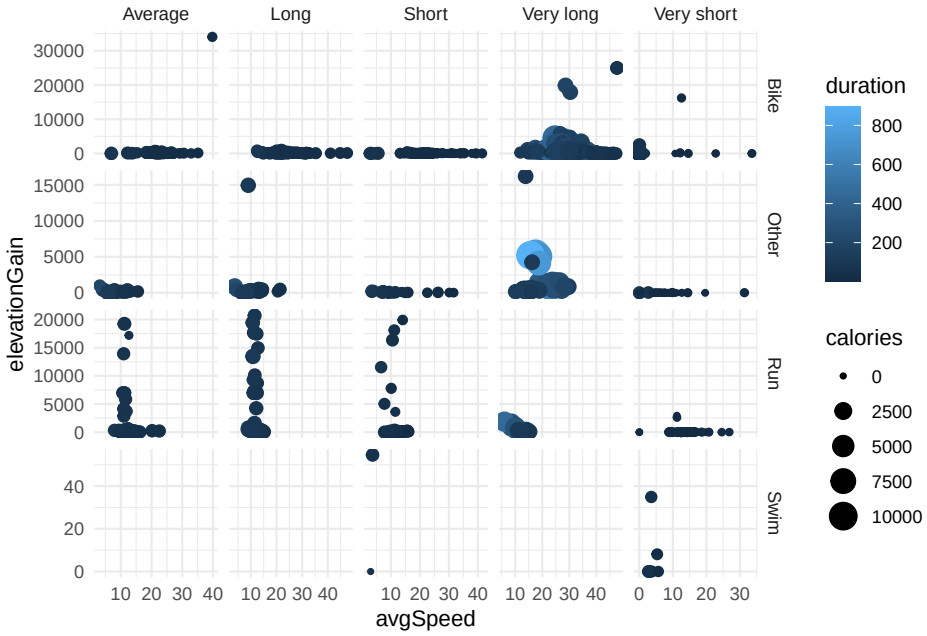

```
ggplot(dat_clean,aes(distance,avgSpeed,color=activity_recoded))  + geom_jitter()+
  theme_minimal()
```




```
# To
ggplot(dat_clean,aes(x=avgSpeed,y=calories,size=duration,color=qual_distance,shape=qual_avgHr)) +
  geom_jitter() +
  facet_wrap(.~ activity_recoded,scales = "free")+
  theme_minimal()
```




```
# Or maybe
ggplot(dat_clean,aes(x=avgSpeed,y=elevationGain,size=calories,color=duration)) +
  geom_jitter() +
  facet_grid(activity_recoded~qual_distance,scales = "free" )+
  theme_minimal()
```



6.1.1.4 Important options

With ggplot, one can make publishable graphics that don't need to be modified in another software. For that, the most useful functions are :

- scale_xx_yy : functions that allow you to tweak the axis' scale, the colors used by either `color` or `fill` aesthetics (eg : `scale_color_manual()`) and other options.
- labs : allows you to proper label title, axis' titles, legend titles…
- theme : allows you to tweak general parameters for the plot (font family, font size, margins, background colors…). You have several `theme_xx()` functions already defined with different default values for those parameters (eg `theme_minimal()` or `theme_void()`)
- guides : allows you to modify the legend entries

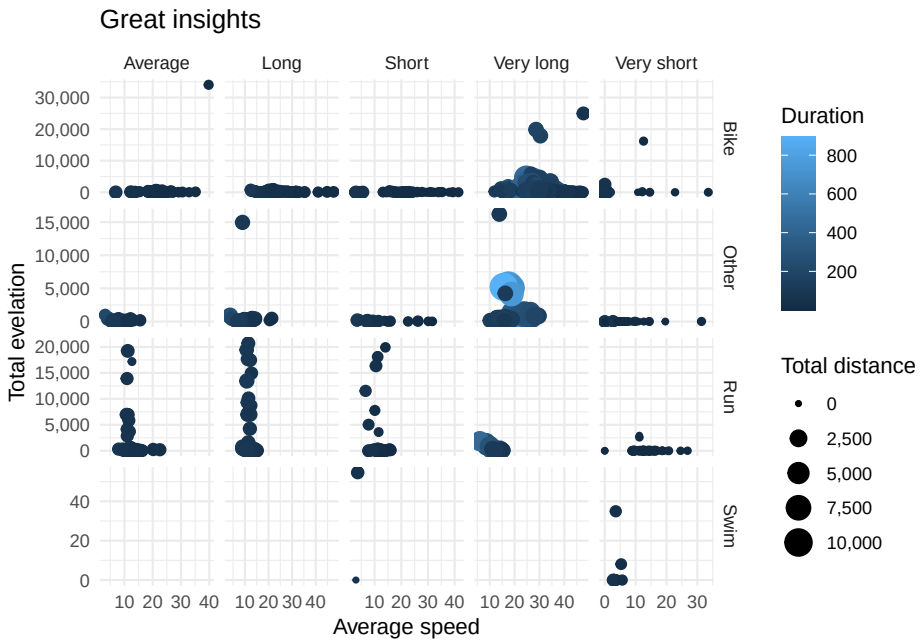
Whats you can also do is recode the levels of the factor variables to make them more understanble or reorder them if you want them to be displayed in a specific order. See `forcats::fct_recode()` and `forcats::fct_reorder()`

Hint : when working with strings, you can force a string to be split in 2 rows with `\n`

Here is an example :

```
dat_clean %>%
  ggplot(aes(x=avgSpeed,y=elevationGain,size=calories,color=duration,size=distance)) +
  geom_jitter() +
```

```
  facet_grid(activity_recoded~qual_distance,scales = "free" ) +
  theme_minimal() +
  labs(title="Great insights",y="Total evelation",x="Average speed",
       color="Duration",size="Total distance") +
  # scale_color_manual(values = c("magenta","orange")) +
  scale_size_continuous(labels=scales::comma) +
  scale_y_continuous(labels=scales::comma) +
  scale_x_continuous(labels=scales::comma)+
  theme_minimal()
```



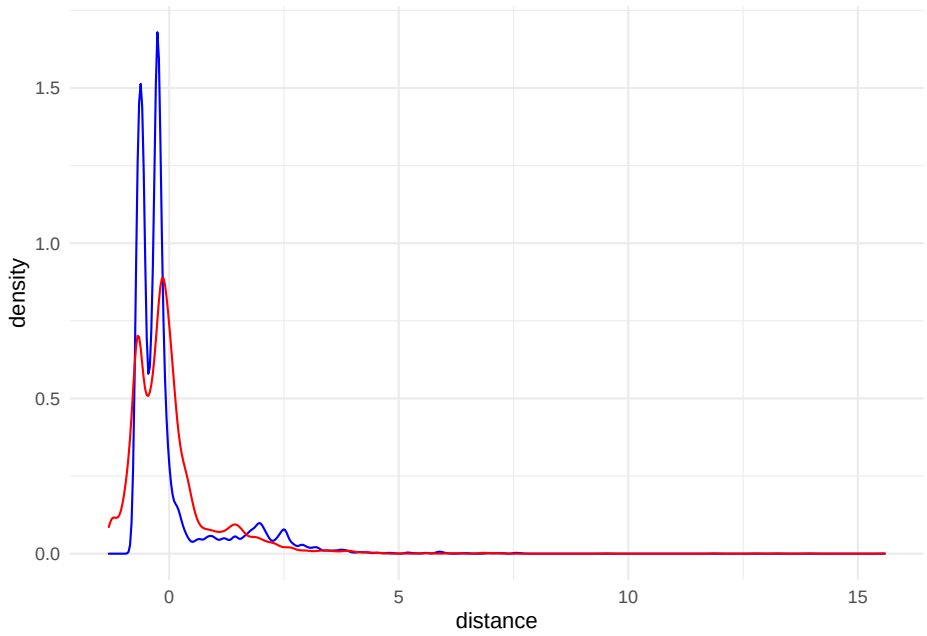
6.1.1.5 Some tricks with ggplot

This approach (grammar of graphics) is very coherent but makes it sometimes difficult. For example, how can I represent the distribution of several variables (and not the distribution of one variable according to different sub-groups - meaning that there is a second dimension -) ?

6.1.1.5.1 The cheater way You can brute-force the graph by superposing different geometries. But first, you'll have to standardize the variables (they don't have the same scale). Remember the usage of `across` to apply a function to a selection of variables.

```
mutate(dat_clean,across(where(is.numeric),
                        function(xx) (xx-mean(xx,na.rm=T))/sd(xx,na.rm=T))) %>%
  ggplot() + geom_density(aes(distance),color="blue") +
```

```
  geom_density(aes(duration),color="red")+
  theme_minimal()
```



This solution can be also used if you want to superpose different geometries (bars and lines, bars and hlines,...), and is in this case legal :D

6.1.1.5.2 Do it with the tidy philosophy You can reformulate this task as : I want to represent the distribution of one unique variable for 2 subgroups : distance and duration. I have to reshape the data first to create such a variable. For that I will use `tidyr::pivot_longer()` which helps me to transform columns into rows.

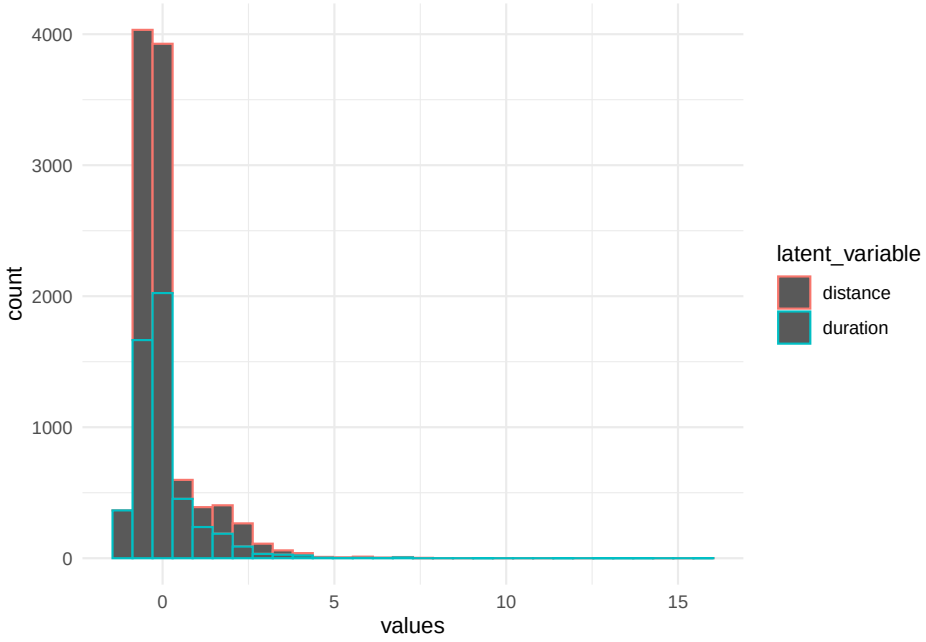
```
reshaped <- select(dat_clean,duration,distance) %>%
  pivot_longer(cols = everything(),names_to="latent_variable",values_to="values")
reshaped
```

```
## # A tibble: 10,250 x 2
##    latent_variable values
##    <chr>            <dbl>
##  1 duration          60.7
##  2 distance           3
##  3 duration         251.
##  4 distance         123.
##  5 duration         173.
##  6 distance          83.6
##  7 duration          89.9
```

```
##  8 distance          17.9
##  9 duration          60.4
## 10 distance          34.9
## # ... with 10,240 more rows
```

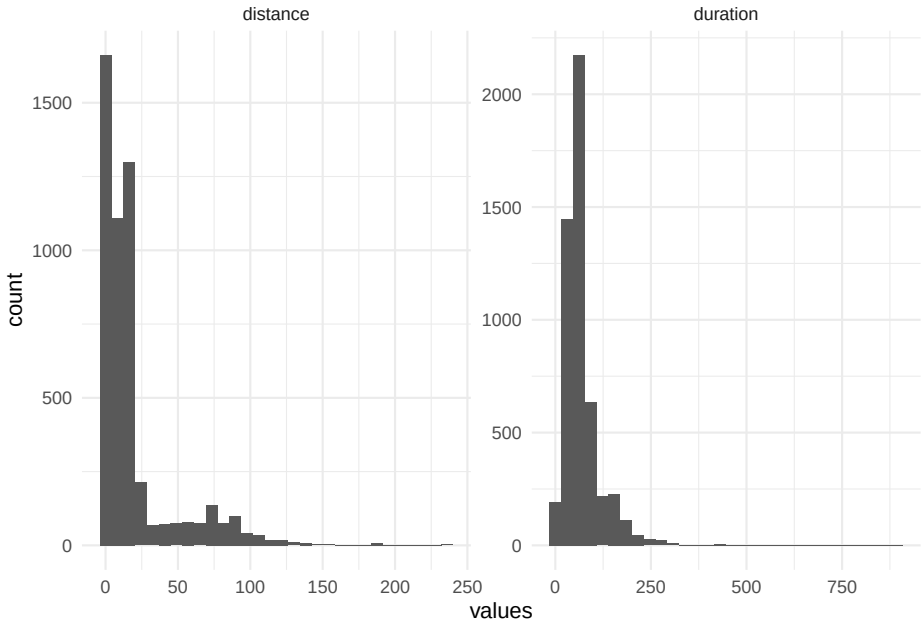
Now I can use the latent_variable variable as a dimension in a classic ggplot statement, with a prior standardization

```
group_by(reshaped,latent_variable) %>%
  mutate(values=(values-mean(values,na.rm = T))/sd(values,na.rm = T)) %>%
  ggplot(aes(values,color=latent_variable)) + geom_histogram()+
  theme_minimal()
```



Or you can even skip the standardization step thanks to facets :

```
ggplot(reshaped,aes(values)) + geom_histogram() +
  facet_wrap(.~latent_variable,scales = "free")+
  theme_minimal()
```

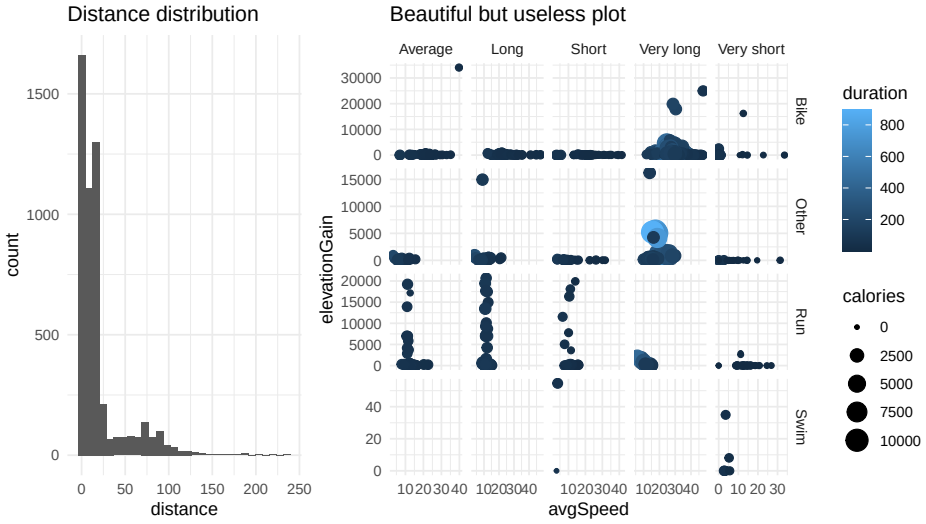


6.1.1.5.3 Combine different plots with ggpubr

ggpubr makes your life much easier to make publication-ready graphics. It allows you for example to combine several ggplot graphics in a grid, regardless of any latent dimension that facet_grid would require. For that you have to store the graphics and "replay" them in a defined grid generated by ggarrange()

```
# install.packages("ggpubr")
require(ggpubr)
gg1 <- ggplot(dat_clean,aes(distance)) + geom_histogram() + labs(title = "Distance distribution")
  theme_minimal()
gg2 <- ggplot(dat_clean,aes(x=avgSpeed,y=elevationGain,size=calories,color=duration,size=distance
  geom_jitter() +
  facet_grid(activity_recoded~qual_distance,scales = "free" ) +
  theme_minimal() + labs(title = "Beautiful but useless plot")+
  theme_minimal()

ggarrange(gg1,gg2,ncol = 2,widths = c(1,2))
```



There are a lot of options in this function (common legend, height and width of each plot,...). You can check the full documentation of this package.

Note that you can mix tables and graphics with this function. Tables can be rendered as ggplot object with `ggpubr::ggtexttable()`

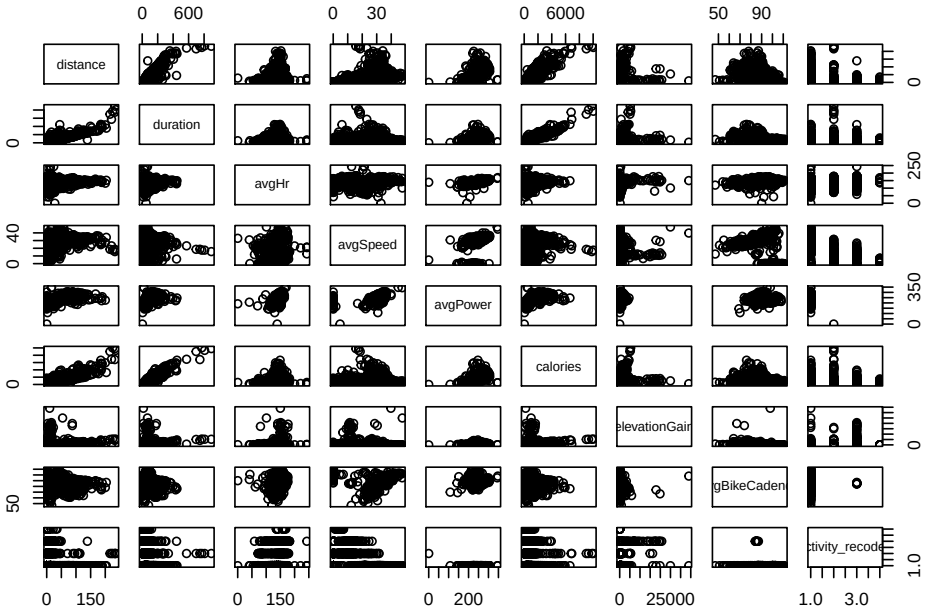
6.1.2 Easily explore an entire dataset

Now, you know how to produce one graph including several dimensions. To explore a new dataset and identify the correlations between them, you can visualize at a glance all variables in a datasets and their correlations with `GGally`

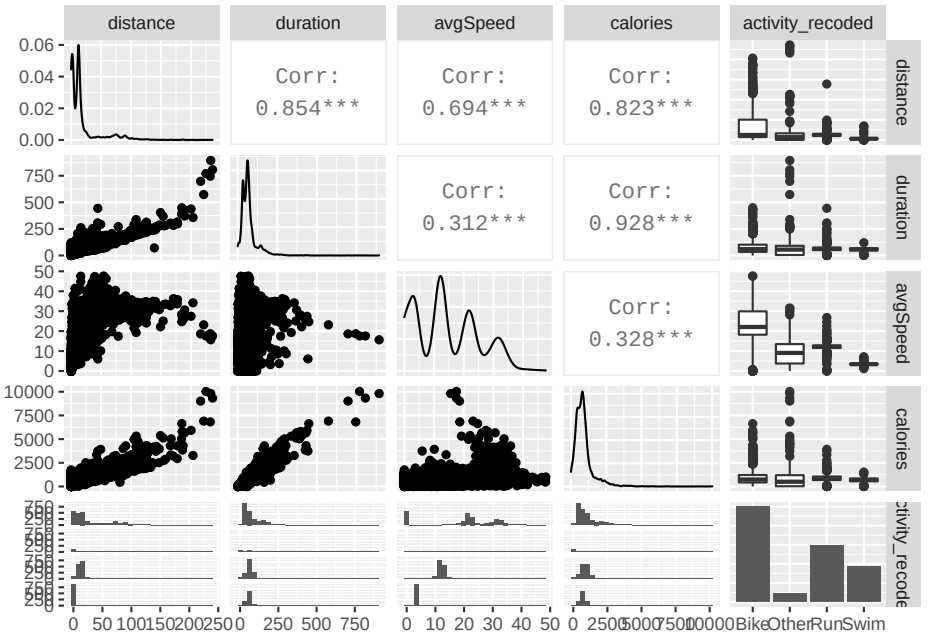
6.1.2.1 Scatter plot matrix

The scatter plot matrix shows the correlations between all variables and helps you to spot dependencies between them. We can use either the basic `plot()` function on the dataframe or the GGally package which provides nice extensions to ggplot. Side note : the `ggpairs()` function does a lot of computation and can take a lot of time ! → if your dataset is large, you should run it only on a sample of the observations with `dplyr::sample_n()` or `dplyr::sample_frac()` or only on a selection of columns.

```
select(dat_clean,distance,duration,avgHr,avgSpeed,avgPower,
       calories,elevationGain,avgBikeCadence,activity_recoded) %>%
  plot()
```

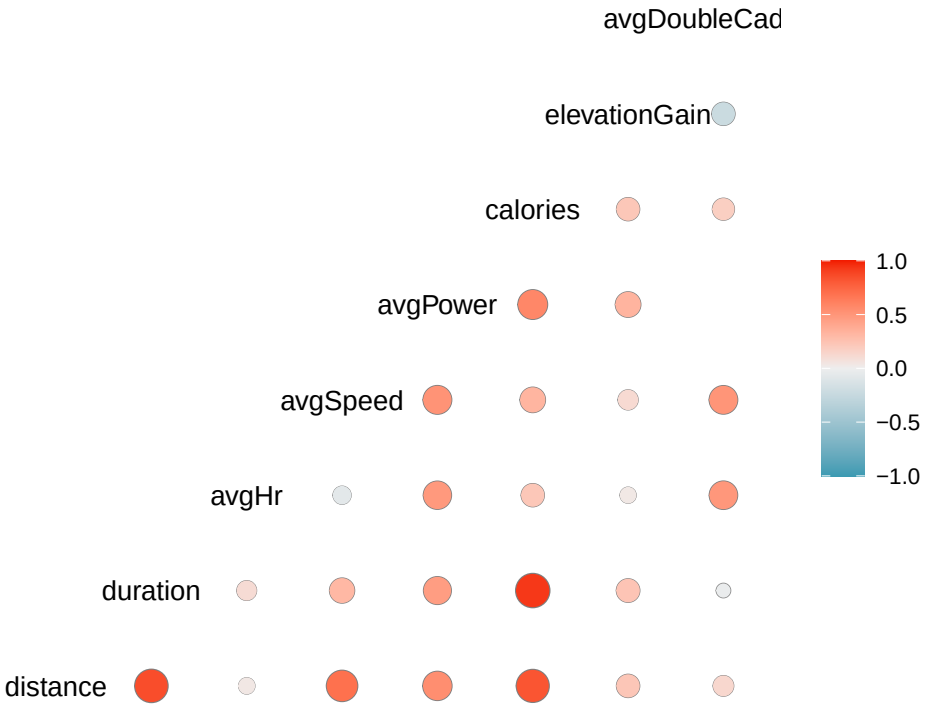



```
# install.packages("GGally")
require(GGally)
select(dat_clean,distance,duration,avgSpeed,
       calories,activity_recoded) %>%
  GGally::ggpairs()
```



6.1.2.2 Correlation plots

```
select(dat_clean,distance,duration,avgHr,avgSpeed,avgPower,
       calories,elevationGain,avgDoubleCadence,activity_recoded)  %>%
  GGally::ggcorr(geom = "circle")
```



6.2 Multivariate analysis and dimension reduction

In this section, we will focus on the bike activities, which have the highest number of metrics. However, before we can go further, we have to deal with missing data and scale them to avoid that the column with a large order of magnitude are over-weighted.

6.2.1 Imputation

So far, we have ignored the missing values because we were computing summary statistics on one or 2 variables. The problem when taking into account more columns is that the probability of having one missing value on one of these features is higher, and therefore the risk that the whole observation is ignored increases. Every observation should still contain some original information that an analysis (or a model) should reflect. To avoid to ignore to many observations

because of missing data we perform **imputation**, meaning that we replace the missing value with a true value. Many methods can be used to that end :

For numeric variables

- Imputation with a random value from the sample
- Imputation with the mean/median
- Imputation with the nearest neighbours (k-nn)
- Hotdeck
- Imputation with a model (regression)

For categorical variables :

- Imputation with random category selection
- Imputation with the most frequent category (total or of the neighbours)
- Hotdeck
- Model-based imputation

The challenge here is to chose between "reflecting the instance's originality" or "not creating noise"

The following code counts the number of missing values per column and does a simple mean or median imputation. We will practice hte imputation via regression in the final chapter's exercises.

```
dat_bike <- filter(dat_clean,is_bike) # Bike activities
sapply(dat_bike,function(xx) sum(is.na(xx)))
```

```
##              activityId                 uuidMsb                 uuidLsb
##                       0                     932                     932
##                    name            activityType          userProfileId
##                     229                       0                       0
##              timeZoneId          beginTimestamp            eventTypeId
##                       0                       0                       0
##                    rule               sportType           startTimeGmt
##                       0                    1262                       0
##          startTimeLocal                duration                distance
##                       0                       0                       1
##           elevationGain         elevationLoss                avgSpeed
##                     175                     176                       2
##                maxSpeed                   avgHr                   maxHr
##                     495                     665                     664
##                calories          startLongitude           startLatitude
##                       2                     583                     583
##   aerobicTrainingEffect  avgFractionalCadence    maxFractionalCadence
##                    1376                       0                       0
##         elapsedDuration         movingDuration anaerobicTrainingEffect
##                    1189                    1528                    2021
##                deviceId          minTemperature          maxTemperature
```

```
##                     852                    1000                    1000
##            minElevation            maxElevation            locationName
##                    1018                    1018                    1781
##        maxVerticalSpeed               lapCount            endLongitude
##                    1708                    1642                    2186
##             endLatitude             activeSets               totalSets
##                    2186                    1855                    1855
##               totalReps              purposeful        autoCalcCalories
##                    1855                       0                    1246
##                favorite                      pr      elevationCorrected
##                       0                       0                     871
##             atpActivity                  parent          maxRunCadence
##                    1828                     474                    2491
##                   steps  avgVerticalOscillation    avgGroundContactTime
##                    2491                    2492                    2492
##         avgStrideLength             vO2MaxValue        avgVerticalRatio
##                    2491                    2289                    2492
## avgGroundContactBalance        avgDoubleCadence        maxDoubleCadence
##                    2492                    2491                    2491
##                avgPower          avgBikeCadence          maxBikeCadence
##                    1832                    1188                    1188
##                 strokes              normPower          avgLeftBalance
##                    1197                    1832                    1896
##         avgRightBalance           max20MinPower     trainingStressScore
##                    1896                    1838                    1881
##         intensityFactor     lactateThresholdBpm   lactateThresholdSpeed
##                    1881                    2492                    2492
##              avgStrokes           activeLengths                avgSwolf
##                    2492                    2311                    2492
##              poolLength       avgStrokeDistance          avgSwimCadence
##                    2492                    2492                    2492
##          maxSwimCadence                   maxFtp               workoutId
##                    2492                    2435                    2381
##                decoDive                parentId        avgVerticalSpeed
##                    2492                    2479                    2492
##                maxDepth                avgDepth         surfaceInterval
##                    2492                    2492                    2492
##         floorsDescended              bottomTime              start_time
##                    2492                    2492                       0
##                    date                 is_bike                  is_run
##                       0                       0                       0
##        activity_recoded           qual_distance              qual_avgHr
##                       0                       0                     914
```

For this use case, we will use a median imputation, because for some of the variables, the mean would not make sense (eg longitude and latitude).

```
dat_bike_imp <- mutate(dat_bike,across(where(is.numeric),
                                       function(xx) ifelse(is.na(xx),median(xx,na.rm=T),xx)))
sapply(dat_bike_imp,function(xx) sum(is.na(xx)))
```

```
##              activityId                 uuidMsb                 uuidLsb
##                       0                     932                     932
##                    name            activityType          userProfileId
##                     229                       0                       0
##              timeZoneId          beginTimestamp             eventTypeId
##                       0                       0                       0
##                    rule               sportType           startTimeGmt
##                       0                    1262                       0
##          startTimeLocal                duration                distance
##                       0                       0                       0
##           elevationGain           elevationLoss                avgSpeed
##                       0                       0                       0
##                maxSpeed                   avgHr                   maxHr
##                       0                       0                       0
##                calories          startLongitude           startLatitude
##                       0                       0                       0
##   aerobicTrainingEffect    avgFractionalCadence    maxFractionalCadence
##                       0                       0                       0
##         elapsedDuration         movingDuration anaerobicTrainingEffect
##                       0                       0                       0
##                deviceId          minTemperature          maxTemperature
##                     852                       0                       0
##            minElevation            maxElevation            locationName
##                       0                       0                    1781
##        maxVerticalSpeed                lapCount           endLongitude
##                       0                       0                       0
##             endLatitude              activeSets               totalSets
##                       0                       0                       0
##               totalReps              purposeful        autoCalcCalories
##                       0                       0                    1246
##                favorite                      pr      elevationCorrected
##                       0                       0                       0
##             atpActivity                  parent           maxRunCadence
##                    1828                     474                       0
##                   steps  avgVerticalOscillation    avgGroundContactTime
##                       0                    2492                    2492
##         avgStrideLength             vO2MaxValue        avgVerticalRatio
##                    2491                       0                    2492
## avgGroundContactBalance         avgDoubleCadence        maxDoubleCadence
##                    2492                       0                       0
##                 avgPower          avgBikeCadence          maxBikeCadence
```

```
##                        0                        0                        0
##                  strokes               normPower          avgLeftBalance
##                        0                        0                        0
##         avgRightBalance           max20MinPower     trainingStressScore
##                        0                        0                        0
##         intensityFactor     lactateThresholdBpm   lactateThresholdSpeed
##                        0                     2492                     2492
##              avgStrokes           activeLengths                avgSwolf
##                     2492                        0                     2492
##              poolLength       avgStrokeDistance          avgSwimCadence
##                     2492                     2492                     2492
##          maxSwimCadence                   maxFtp               workoutId
##                     2492                        0                     2381
##                decoDive                 parentId        avgVerticalSpeed
##                     2492                     2479                     2492
##                maxDepth                 avgDepth         surfaceInterval
##                     2492                     2492                     2492
##         floorsDescended               bottomTime               start_time
##                     2492                     2492                        0
##                    date                  is_bike                   is_run
##                        0                        0                        0
##        activity_recoded           qual_distance               qual_avgHr
##                        0                        0                      914
```

Let's do the same for categorical (maximum frequency), even though it is not mandatory for PCA. We first have to create a function that will return the most frequent category of a vector.

```
most_freq_cat <- function(xx)
{
  tab <- table(xx)
  return(names(tab[which.max(tab)]))
}

dat_bike_imp <- mutate(dat_bike_imp,across(where(is.character),
                                           function(xx) coalesce(xx,most_freq_cat(xx))))

sapply(dat_bike_imp,function(xx) sum(is.na(xx)))
```

```
##               activityId                  uuidMsb                  uuidLsb
##                        0                        0                        0
##                     name             activityType            userProfileId
##                        0                        0                        0
##               timeZoneId           beginTimestamp              eventTypeId
##                        0                        0                        0
##                     rule                sportType             startTimeGmt
##                        0                        0                        0
```

```
##          startTimeLocal               duration               distance
##                       0                      0                      0
##           elevationGain          elevationLoss               avgSpeed
##                       0                      0                      0
##                maxSpeed                 avgHr                  maxHr
##                       0                      0                      0
##                calories         startLongitude          startLatitude
##                       0                      0                      0
##   aerobicTrainingEffect   avgFractionalCadence   maxFractionalCadence
##                       0                      0                      0
##         elapsedDuration        movingDuration anaerobicTrainingEffect
##                       0                      0                      0
##                deviceId         minTemperature         maxTemperature
##                       0                      0                      0
##            minElevation           maxElevation           locationName
##                       0                      0                      0
##        maxVerticalSpeed              lapCount           endLongitude
##                       0                      0                      0
##             endLatitude            activeSets              totalSets
##                       0                      0                      0
##               totalReps             purposeful       autoCalcCalories
##                       0                      0                   1246
##                favorite                     pr     elevationCorrected
##                       0                      0                      0
##             atpActivity                 parent          maxRunCadence
##                    1828                    474                      0
##                   steps avgVerticalOscillation   avgGroundContactTime
##                       0                   2492                   2492
##         avgStrideLength            vO2MaxValue        avgVerticalRatio
##                       0                      0                   2492
## avgGroundContactBalance        avgDoubleCadence        maxDoubleCadence
##                    2492                      0                      0
##                avgPower          avgBikeCadence          maxBikeCadence
##                       0                      0                      0
##                 strokes              normPower          avgLeftBalance
##                       0                      0                      0
##         avgRightBalance           max20MinPower     trainingStressScore
##                       0                      0                      0
##         intensityFactor     lactateThresholdBpm   lactateThresholdSpeed
##                       0                   2492                   2492
##              avgStrokes          activeLengths               avgSwolf
##                    2492                      0                   2492
##              poolLength       avgStrokeDistance          avgSwimCadence
##                    2492                   2492                   2492
##          maxSwimCadence                  maxFtp              workoutId
##                    2492                      0                      0
```

```
##                decoDive                parentId        avgVerticalSpeed
##                    2492                       0                    2492
##               maxDepth                avgDepth         surfaceInterval
##                    2492                    2492                    2492
##         floorsDescended              bottomTime              start_time
##                    2492                    2492                       0
##                    date                 is_bike                  is_run
##                       0                       0                       0
##        activity_recoded          qual_distance              qual_avgHr
##                       0                       0                       0
```

```
# replacements <-   select(dat_bike_imp,activityId,where(is.character)) %>%
#   pivot_longer(-activityId,names_to="name",values_to="val") %>%
#   filter(!is.na(val)) %>%
#   group_by(name,val) %>%
#   summarise(cat_nb=n()) %>%
#   arrange(name,-cat_nb) %>%
#   group_by(name) %>%
#   summarise(most_freq=first(val)) %>%
#   pivot_wider(names_from = name,values_from=most_freq) %>%
#   rename_with(function(xx) paste0(xx,"_imp"))
```

6.2.2 Normalization

Normalization is the operation consisting in scaling the columns so that their unit do not matter in the end. For example, the distance in meter is much larger than the cadence or the power, which have totally different units. To normalize the columns and make them unit-less, there are several methods among which the most common are the following :

- Standardization : $X_i^{std} = \frac{X_i - \bar{X}}{\sigma_X} \rightarrow$ mean 0 and standard deviation 1
- Min-Max scaling : $X_i^{std} = \frac{X_i - min(X)}{max(X) - min(X)} \rightarrow$ between 0 and 1
- Robust standardization $X_i^{std} = \frac{X_i - Q2(X)}{Q3(X) - Q1(X)} \rightarrow$ similar to the first option but robust to outliers

You can check scikit-learn's documentation to see what other options you have (and then search for their R implementation).

Normalization is a mandatory step before fitting a model, in order to avoid that only one feature bears the majority of the variance and makes the model biased.

6.2.3 PCA

Principal Components Analysis (PCA) is often seen by machine learning engineers "only" as a dimension reduction technique, but it is also a very powerful

tool to explore your data. PCA applies on numerical variables only, and aims to **create new synthetic and uncorrelated variables : principal components** (as a linear combination of the original variables) such that the *inertia* (ie variance) is highly concentrated on a small number of variables.

The mathematical problem is to find eigenvalues and eigenvectors of the correlation matrix. The eigenvectors represent the linear combination of the original variables needed to design those new variables, and the eigenvalues the variance that each of these new value bears. Once those new variables have been found, graphical representations within a few dimensions are possible.

To apply PCA, we will use the `FactoMineR` package, very easy to use for multivariate analysis.

Notes :

- You can have supplementary continuous or categorical variables. They won't be used in the construction of the eigenvectors, but you will be able to place them in the newly defined vector space.
- By default, almost all implementations of PCA standardizes the data, so you do not have to do it by yourself, but check in the documentation of the function you use

```
# install.packages("FactoMineR")
require(FactoMineR)
acp_dat <- select(dat_bike_imp,deviceId,duration,distance,elevationGain,elevationLoss,avgSpeed,av
              minTemperature,maxTemperature,lapCount,avgBikeCadence,avgPower,vO2MaxValue,
              max20MinPower)
acp <-   PCA(acp_dat,graph = F,quali.sup = c(1))
```

What's in this new object ?

```
str(acp)
```

```
## List of 6
##  $ eig      : num [1:14, 1:3] 4.73 1.95 1.76 1.29 1.26 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:14] "comp 1" "comp 2" "comp 3" "comp 4" ...
##   .. ..$ : chr [1:3] "eigenvalue" "percentage of variance" "cumulative percentage of variance"
##  $ var      :List of 4
##   ..$ coord  : num [1:14, 1:5] 0.799 0.902 0.495 0.507 0.658 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:14] "duration" "distance" "elevationGain" "elevationLoss" ...
##   .. .. ..$ : chr [1:5] "Dim.1" "Dim.2" "Dim.3" "Dim.4" ...
##   ..$ cor    : num [1:14, 1:5] 0.799 0.902 0.495 0.507 0.658 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:14] "duration" "distance" "elevationGain" "elevationLoss" ...
##   .. .. ..$ : chr [1:5] "Dim.1" "Dim.2" "Dim.3" "Dim.4" ...
##   ..$ cos2   : num [1:14, 1:5] 0.639 0.814 0.245 0.258 0.433 ...
```

```
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:14] "duration" "distance" "elevationGain" "elevationLoss" ...
##   .. .. ..$ : chr [1:5] "Dim.1" "Dim.2" "Dim.3" "Dim.4" ...
##   ..$ contrib: num [1:14, 1:5] 13.49 17.21 5.18 5.44 9.15 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:14] "duration" "distance" "elevationGain" "elevationLoss" ...
##   .. .. ..$ : chr [1:5] "Dim.1" "Dim.2" "Dim.3" "Dim.4" ...
##  $ ind      :List of 4
##   ..$ coord  : num [1:2492, 1:5] 7.59 1.88 -1.2 3.41 3.63 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:2492] "1" "2" "3" "4" ...
##   .. .. ..$ : chr [1:5] "Dim.1" "Dim.2" "Dim.3" "Dim.4" ...
##   ..$ cos2   : num [1:2492, 1:5] 0.6916 0.0999 0.1419 0.4143 0.4213 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:2492] "1" "2" "3" "4" ...
##   .. .. ..$ : chr [1:5] "Dim.1" "Dim.2" "Dim.3" "Dim.4" ...
##   ..$ contrib: num [1:2492, 1:5] 0.4883 0.0301 0.0123 0.0987 0.1116 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:2492] "1" "2" "3" "4" ...
##   .. .. ..$ : chr [1:5] "Dim.1" "Dim.2" "Dim.3" "Dim.4" ...
##   ..$ dist   : Named num [1:2492] 9.13 5.96 3.2 5.3 5.59 ...
##   .. ..- attr(*, "names")= chr [1:2492] "1" "2" "3" "4" ...
##  $ svd      :List of 3
##   ..$ vs: num [1:14] 2.18 1.4 1.33 1.14 1.12 ...
##   ..$ U : num [1:2492, 1:5] 3.488 0.866 -0.553 1.569 1.668 ...
##   ..$ V : num [1:14, 1:5] 0.367 0.415 0.228 0.233 0.302 ...
##  $ quali.sup:List of 5
##   ..$ coord : num [1:9, 1:5] 0.6548 -0.3543 0.1883 0.0828 -0.3446 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:9] "3454054575" "3825981698" "3842681113" "3858417560" ...
##   .. .. ..$ : chr [1:5] "Dim.1" "Dim.2" "Dim.3" "Dim.4" ...
##   ..$ cos2  : num [1:9, 1:5] 0.15832 0.05206 0.0159 0.00389 0.21496 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:9] "3454054575" "3825981698" "3842681113" "3858417560" ...
##   .. .. ..$ : chr [1:5] "Dim.1" "Dim.2" "Dim.3" "Dim.4" ...
##   ..$ v.test: num [1:9, 1:5] 0.4257 -1.1513 0.1732 0.0659 -1.5304 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:9] "3454054575" "3825981698" "3842681113" "3858417560" ...
##   .. .. ..$ : chr [1:5] "Dim.1" "Dim.2" "Dim.3" "Dim.4" ...
##   ..$ dist  : Named num [1:9] 1.646 1.553 1.494 1.327 0.743 ...
##   .. ..- attr(*, "names")= chr [1:9] "3454054575" "3825981698" "3842681113" "385841
##   ..$ eta2  : num [1, 1:5] 0.1543 0.0882 0.1056 0.1529 0.0213
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr "deviceId"
##   .. .. ..$ : chr [1:5] "Dim.1" "Dim.2" "Dim.3" "Dim.4" ...
##  $ call     :List of 10
```

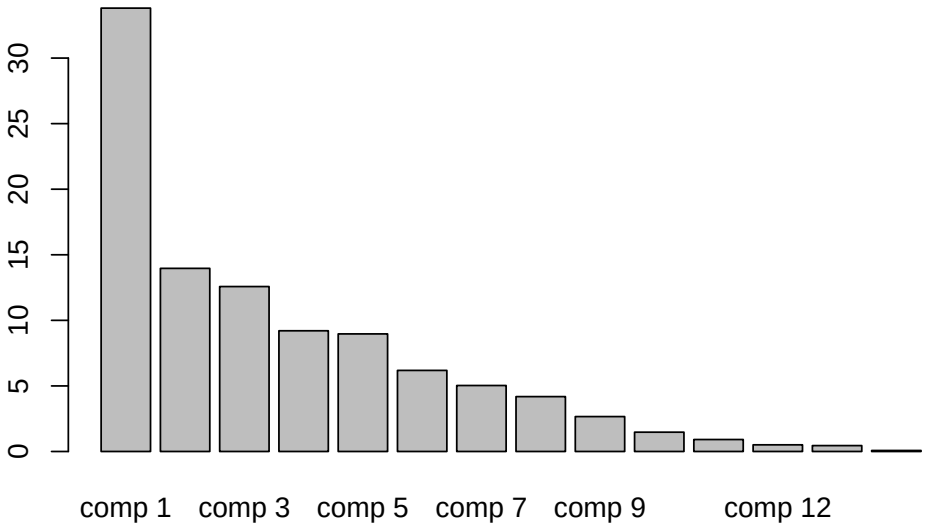
```
##   ..$ row.w     : num [1:2492] 0.000401 0.000401 0.000401 0.000401 0.000401 ...
##   ..$ col.w     : num [1:14] 1 1 1 1 1 1 1 1 1 1 ...
##   ..$ scale.unit: logi TRUE
##   ..$ ncp       : num 5
##   ..$ centre    : num [1:14] 77.8 29.8 368.5 356.2 20.1 ...
##   ..$ ecart.type: num [1:14] 59 35 1144.6 1121.7 11.9 ...
##   ..$ X         :'data.frame': 2492 obs. of  15 variables:
##   .. ..$ deviceId      : Factor w/ 9 levels "3454054575","3825981698",..: 9 9 2 9 9 9 9 2 2 9
##   .. ..$ duration      : num [1:2492] 251.1 173 60.4 141.8 140.5 ...
##   .. ..$ distance      : num [1:2492] 122.9 83.6 34.9 79.9 74.2 ...
##   .. ..$ elevationGain : num [1:2492] 2001 745 160 619 1117 ...
##   .. ..$ elevationLoss : num [1:2492] 1968 742 0 611 1114 ...
##   .. ..$ avgSpeed      : num [1:2492] 29.4 29 34.7 33.8 31.7 ...
##   .. ..$ avgHr         : int [1:2492] 144 117 138 140 140 126 138 128 120 151 ...
##   .. ..$ calories      : num [1:2492] 3907 2090 737 2229 2181 ...
##   .. ..$ minTemperature: num [1:2492] 19 18 18 21 22 23 19 18 18 17 ...
##   .. ..$ maxTemperature: num [1:2492] 29 29 27 30 32 32 30 27 27 29 ...
##   .. ..$ lapCount      : num [1:2492] 25 17 1 16 15 19 17 1 1 36 ...
##   .. ..$ avgBikeCadence: num [1:2492] 83 82 91 88 87 82 86 87 86 88 ...
##   .. ..$ avgPower      : num [1:2492] 260 202 212 263 259 210 263 213 197 240 ...
##   .. ..$ vO2MaxValue   : int [1:2492] 72 71 64 71 71 71 64 64 64 64 ...
##   .. ..$ max20MinPower : num [1:2492] 375 243 221 287 311 ...
##   ..$ row.w.init: num [1:2492] 1 1 1 1 1 1 1 1 1 1 ...
##   ..$ call      : language PCA(X = acp_dat, quali.sup = c(1), graph = F)
##   ..$ quali.sup :List of 5
##   .. ..$ quali.sup :'data.frame': 2492 obs. of  1 variable:
##   .. .. ..$ deviceId: Factor w/ 9 levels "3454054575","3825981698",..: 9 9 2 9 9 9 9 2 2 9 ...
##   .. ..$ modalite  : int 9
##   .. ..$ nombre    : num [1:9] 2 49 4 3 90 ...
##   .. ..$ barycentre:'data.frame': 9 obs. of  14 variables:
##   .. .. ..$ duration      : num [1:9] 96.9 62.4 72.9 110.1 52.8 ...
##   .. .. ..$ distance      : num [1:9] 42.4 34 38.9 30.5 20.5 ...
##   .. .. ..$ elevationGain : num [1:9] 281 367 287 490 419 ...
##   .. .. ..$ elevationLoss : num [1:9] 275 0 302 468 416 ...
##   .. .. ..$ avgSpeed      : num [1:9] 25.22 32.54 29.22 9.52 22.39 ...
##   .. .. ..$ avgHr         : num [1:9] 145 132 125 134 129 ...
##   .. .. ..$ calories      : num [1:9] 1894 844 655 1018 842 ...
##   .. .. ..$ minTemperature: num [1:9] 18 18 18 18 18 ...
##   .. .. ..$ maxTemperature: num [1:9] 27 27 27 27 27 ...
##   .. .. ..$ lapCount      : num [1:9] 3 1.29 3 3 3 ...
##   .. .. ..$ avgBikeCadence: num [1:9] 95.5 90.8 83.8 85.3 89.9 ...
##   .. .. ..$ avgPower      : num [1:9] 239 228 239 239 239 ...
##   .. .. ..$ vO2MaxValue   : num [1:9] 64 64 64 64 64 ...
##   .. .. ..$ max20MinPower : num [1:9] 269 258 269 269 269 ...
##   .. ..$ numero    : num 1
##  - attr(*, "class")= chr [1:2] "PCA" "list "
```

This is a list with several elements :

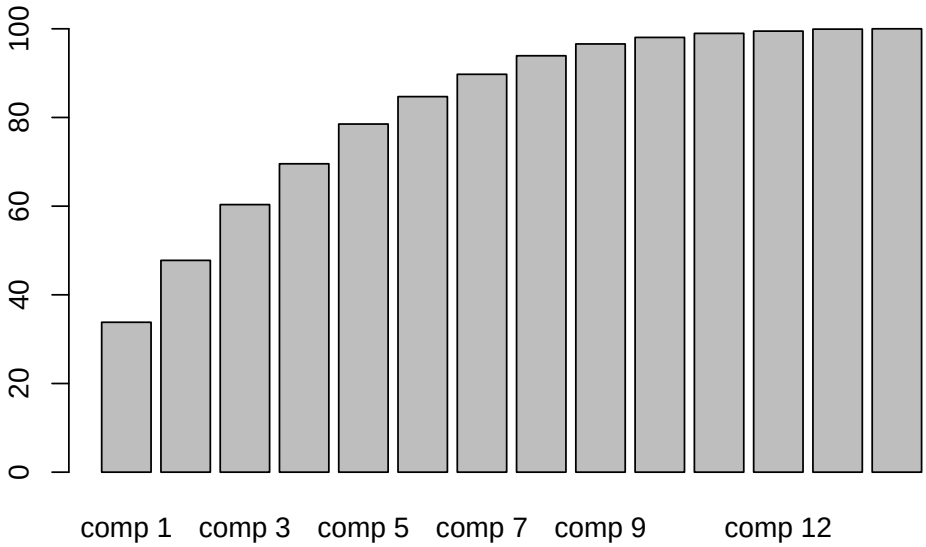
- `eig` contains the eigenvalues, and their share in the total inertia (PCA is performed by default on scaled variables → the trace of the diagonal matrix equals to the number of columns)
- `ind` and `var` refer to rows and columns. They both have the same elements : "coord" for the coordinates on each principal component, "contr" the contributions to the inertia of this row (resp column) to the inertia of the component, "cos2" the squared cosine (ie quality of projection) for the row/column
- `quali.sup` have the same elements than the previous ones (coordinates are the barycenter of each category of the qualitative variable) plus the η^2 statistic.

The first thing is to have a look at the eignevalues to see how well the PCA could summarize the information

```
barplot(acp$eig[,2])
```

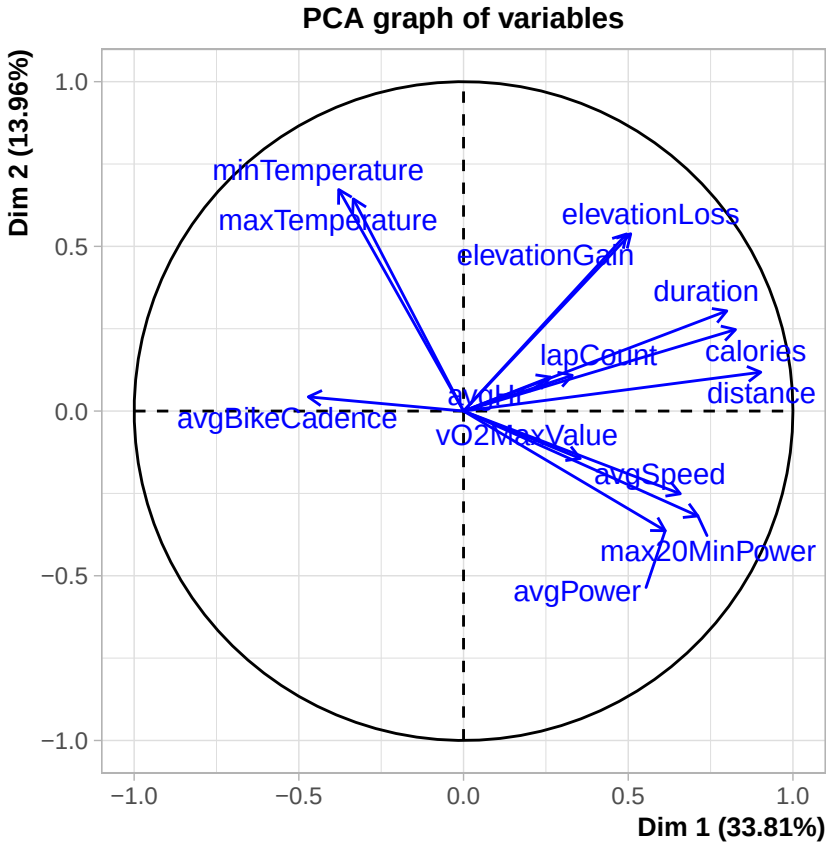



```
barplot(acp$eig[,3])
```



In our case, the first component bears 35% of the total inertia and the second 15%. Hence, the first factorial plane (the first 2 components) : let's look at how the variable correlate with them :

```
plot.PCA(acp,choix="var",col.var = "blue")
```



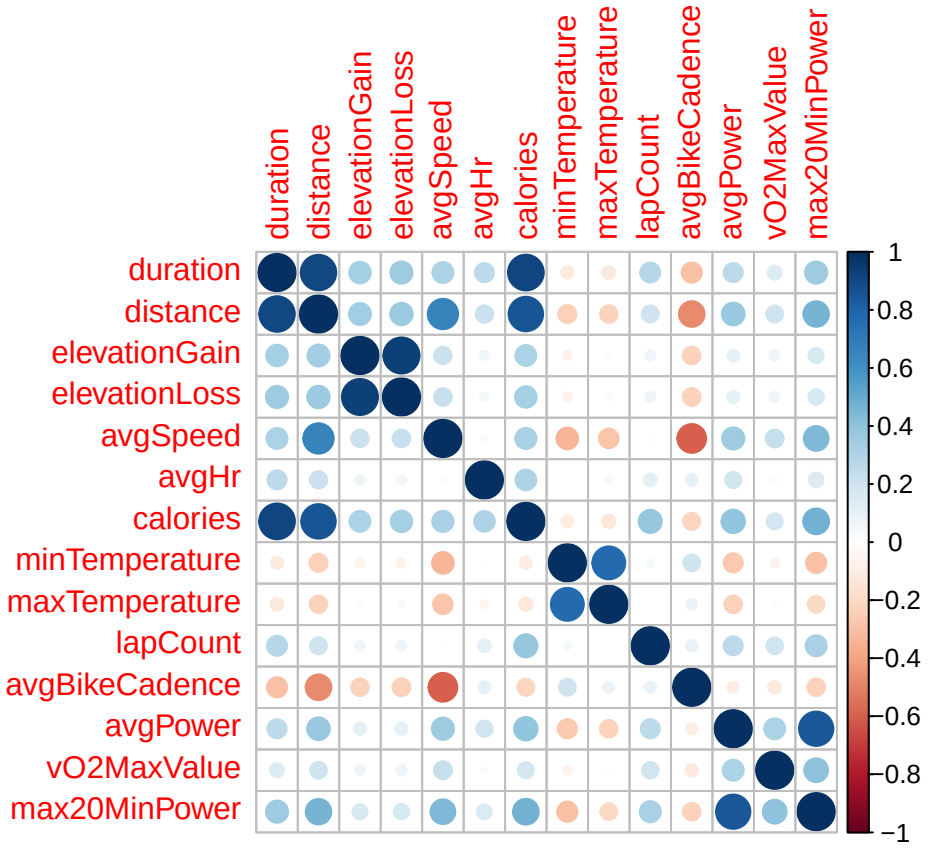
What you can read at once on this plot is how important each variable is to construct the new variables. As a matter of fact, contribution, cos2 and coordinates are almost the same in PCA, meaning that the closer to the unit disc the coordinate is, the higher both contribution and quality of representation.

What we deduct from this graph :

- Distance, calories and duration have the largest values on the first component → they contribute largely to the first component, which bears the 35% of total variance → those variables are the most discriminant
- Those three variables are very close to each other, which means they are highly correlated
- The average cadence is on the other side of the first component → it is negatively correlated with those variables
- There is a right angle between elevationGain and avgPower → the correlation is very small

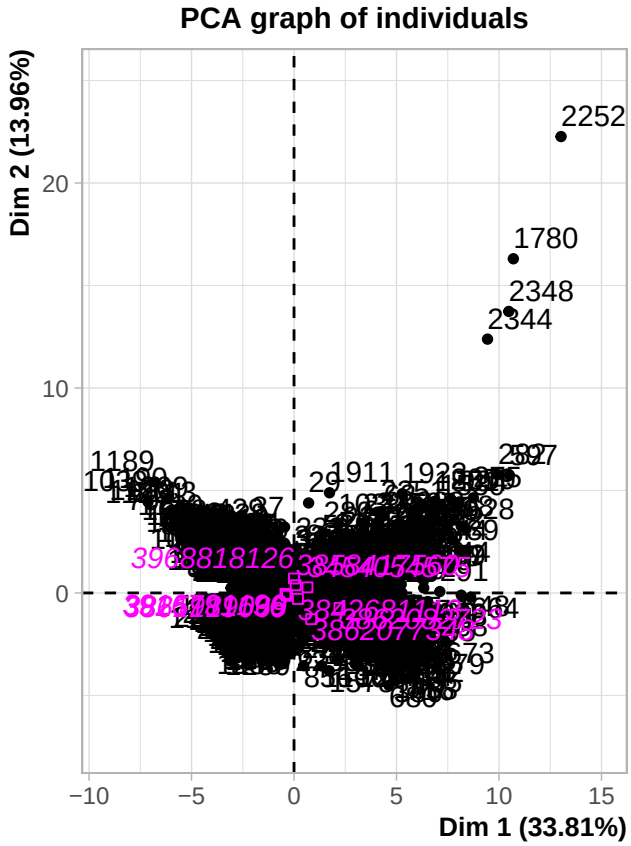
Let's check the correlation matrix to be sure about what we are reading here

```
select(acp_dat,-deviceId) %>%
  cor() %>%
  corrplot::corrplot()
```



How about the individuals ?

```
plot.PCA(acp,choix="ind")
```



This graph is harder to read, but we still can identify outliers and understand what they are. Anyway in the top-right corner, those are long rides with a lot of ups and downs ! The most extreme is 380343030 which is another measurement error (indoor cycling with 34000 elevation meters !)

6.2.4 Exercises

- Remove the measurement errors and re-run the PCA ; what are the changes ?
- Let's focus the running activities : select relevant features, impute the missing values using k-nn, run a PCA and analyze the results

6.3 Dimension reduction

Dimension reduction aims to reduce the size of the data ; most of the time the goal is to decrease the number of columns but it can also apply to rows (depending on your use case). Why reduce the dimension :

- Avoid the curse of dimensionality

- Remove undesired noise
- Avoid multicollinearity in the features

6.3.1 Use the results of the PCA

After performing a PCA, you can select a limited number of principal components that you will use further in the modeling task. You can chose the number of components to keep with several criteria :

- average inertia (eigenvalue > 1)
- minimal total inertia (eg 80%)
- elbow criteria : keep components before there is a "drop" in the eigenvalues barplot

6.3.2 Other inertia-based methods

PCA is designed only for continuous variables, but you can use other methods for other types of data :

- MCA (Multiple Correspondence Analysis) for categorical variables (if you have a mixture of continuous and categorical, you can discretize numerical variables and perform MCA)
- FDA (Functional Data Analysis for functions/curves)

6.3.3 t-SNE

t-SNE has a completely different approach to dimension reduction : it does not aim to preserve the total variance but rather the distances between the observations.

6.3.4 A word about clustering

6.4 Visualization bonus : dashboards and reports

6.4.1 Shiny and rmarkdown

R offers 2 main packages that allow you to create dashboards and reports in HTML format and the latest feature of web technologies without knowing anything about HTML, CSS or other web-specific languages :

- Rmarkdown is markdown adapted to R, allows you to create *standalone* documents in either PDF or HTML format, and mixes regular word-processor features and R code. The flexdashboard package is an extension to Rmarkdown that helps to generate dashboards in HTML format.

Those tools are essentially used to design reproducible documents (scientific reports, regularly updated reports/dashboards...). You can also create presentation with it.
- Shiny is a package to develop interactive web-apps. A shiny app requires an R engine to be rendered on the client side.

In the course material, you'll find one example for each package. You'll easily find online material to develop your skills further with these tools

- For rmarkdown
- For shiny

6.4.2 Web-based graphics with `plotly`

plotly is a graphical library that generate "interactive" web-based graphics. It is available for R, but also python and other programming languages. You can learn the syntax of this package too, but there is a very useful function, `plotly::ggplotly()` that translates your ggplot object into a `plotly` graphic. It works for most of the cases.

```
# install.packages("plotly")
require(plotly)
ggplotly(gg1)
```

This is very useful to embed in a rmarkdown document or a shiny app !

6.4.3 Other packages & widgets

You can use a lot of other html widgets to incorporate in your report/shiny app

Example of leaflet :

```
filter(dat,is_bike) %>%
  leaflet::leaflet(data=.) %>%
  leaflet::addMarkers(lng = ~startLongitude,lat = ~startLatitude,
                      popup = ~activityId) %>%
  leaflet::addTiles()
```

Chapter 7

Linear and logistic regression

Regression is the first machine learning algorithm. It allows you to model a *target variable* y depending on a set of *explanatory variables* or *features* X such that $y = f(X) + \epsilon$ where f is a linear function (for linear regression).

7.1 Linear regression

We will jump directly to the *multiple regression model*, which is the generalization of the simple linear model, which you can check here

7.1.1 General presentation

The basic equation of the linear regression is

$$y_i = x_i \cdot b + \epsilon_i \Leftrightarrow y_i = \sum_{j=1}^{p} x_{ij} b_j + \epsilon_i$$

Where :

- x_i is a row-vector of size p (number of explanatory variables), containng the values of each feature of observation i. It is the row i of the matrix $X = (x_{ij})$
- b is a column-vector of coefficients, one per explanatory variable
- ϵ_i is the error term for observation i

This regression is said to be linear because it is *linear in the parameters*, you can however transform the original variables at will with non-linear functions (see feature engineering).

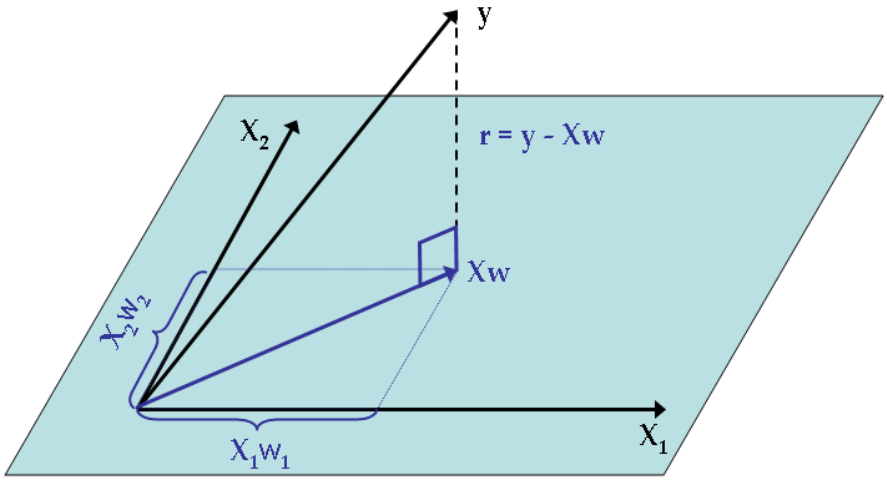
The biggest assumptions of this model are :

- Observations are iid
- There is no perfect multi-collinearity among features
- ϵ_i has a zero conditional mean $\mathbb{E}(\epsilon|X) = 0$

This last condition helps us to derive an estimator for b (which can be derived in several ways) which is called the OLS estimator (Ordinary Least Squares), which is the solution of the optimization program :

$$\hat{b} = argmin_b \sum_{i=1}^{n} \epsilon_i^2 = argmin_b \sum_{i=1}^{n} (x_i - x_i \cdot b)^2$$

The solution is $\hat{b} = (X'X)^{-1}X'y$ where $X' = t(X)$. $(X'X)^{-1}X$ is the projection matrix over the hyperplane defined by the features.



Implementation and diagnostics

To implement a linear regression with R, we use the lm function :

```
reg <- lm(avgSpeed~avgPower + avgBikeCadence + distance + avgHr + max20MinPower, data=d
summary(reg)
```

```
##
## Call:
## lm(formula = avgSpeed ~ avgPower + avgBikeCadence + distance +
##     avgHr + max20MinPower, data = dat_bike)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -29.460  -3.588  -1.053   3.133  22.718
##
```

```
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)    109.32567    6.69319  16.334  < 2e-16 ***
## avgPower         0.25349    0.02257  11.231  < 2e-16 ***
## avgBikeCadence  -1.41346    0.07388 -19.131  < 2e-16 ***
## distance         0.09537    0.01205   7.917 1.14e-14 ***
## avgHr           -0.04565    0.03239  -1.410    0.159
## max20MinPower   -0.06764    0.01523  -4.441 1.06e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.566 on 616 degrees of freedom
##   (1870 observations deleted due to missingness)
## Multiple R-squared:  0.8309,Adjusted R-squared:  0.8296
## F-statistic: 605.6 on 5 and 616 DF,  p-value: < 2.2e-16
```

The goodness of fit is measured through 2 main statistics :

- Adjusted R-squared, $1 - \frac{n-1}{n-k-1}\frac{SSR}{TSS} \in [0,1]$, which takes the number of regressors into account. The closer to 1, the better the fit
- RMSE (root mean squared error), or residual standard error which has to be compared to the average value of y. the smaller the value, the better the fit.

7.1.2 Coefficients interpretation and inference

Back to original equation, we can understand how much each feature influences in average the output.

$$\frac{\partial y}{\partial x_1} = b_1$$

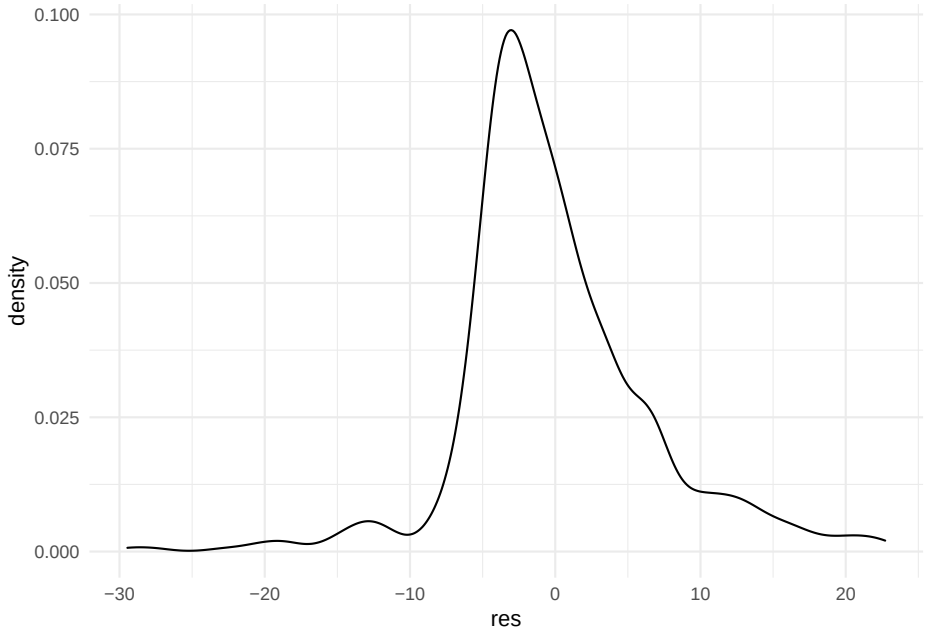
Meaning that the increase of x_1 by one unit causes the output to increase in average by b_1 (which can of course be negative). In our example, one additional watt will result in an increase of the average speed by 0.25 km/h

The fundamental hypothesis being fulfilled and the sample being large enough, the distribution of the OLS estimate $(b_1, ..., b_p)$ are jointly normally distributed, meaning that each $\hat{b_j} \hookrightarrow \mathcal{N}(b_j, \sigma^2_{b_j})$ We can therefore perform statistical tests following the previous methodology (see @ref(stat_inf).

The most common test is the Student test which tests the null hypothesis $b_i = 0$. This allows to check whether a regressor has a significant effect on the target variable or not.

But you have to check your residuals !

```
residuals(reg) %>%
  data.frame(res=.) %>%
  ggplot(aes(res)) + geom_density() + theme_minimal()
```



Those are pretty long tailed, which might reflect some outliers or a wrong functional specification !

7.1.3 The Frisch–Waugh Theorem and the omitted variable bias

The Frisch-Waugh theorem tells us that adding a variable as regressor ensures that our estimates controls for the effect of this variable. In other words, you can interpret the coefficients' values *ceteris paribus* (other things equal).

This also means that if you omit a variable, the coefficient of the other variables are likely to be biased, because you did not take an important variable into account. Back to our example, we can add the elevationGain variable and check what happens :

```
reg <- lm(avgSpeed~avgPower + avgBikeCadence + distance + avgHr + max20MinPower + eleva
summary(reg)
```

```
##
## Call:
## lm(formula = avgSpeed ~ avgPower + avgBikeCadence + distance +
##     avgHr + max20MinPower + elevationGain, data = dat_bike)
##
```

```
## Residuals:
##      Min       1Q   Median       3Q      Max
## -17.6200  -3.8680  -0.5302   2.6892  19.5814
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)     1.265e+02  6.630e+00  19.076  < 2e-16 ***
## avgPower        1.374e-01  2.354e-02   5.837 1.02e-08 ***
## avgBikeCadence -1.521e+00  7.511e-02 -20.253  < 2e-16 ***
## distance        1.987e-01  1.431e-02  13.884  < 2e-16 ***
## avgHr           7.929e-03  3.149e-02   0.252    0.801
## max20MinPower  -7.367e-03  1.547e-02  -0.476    0.634
## elevationGain  -1.281e-02  9.543e-04 -13.424  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.938 on 450 degrees of freedom
##   (2035 observations deleted due to missingness)
## Multiple R-squared:  0.8086,Adjusted R-squared:  0.8061
## F-statistic: 316.9 on 6 and 450 DF,  p-value: < 2.2e-16
```

See how the coefficients changed. This is understandable because when climbing mountains :

- More power will not increase the speed, just maintain it (… or not)
- The cadence is harder to maintain unless you have unlimited gears !
- The surprising negative effect of the max20MinPower is no more

Notice though that the RMSE and the adjusted R^2 degraded… See the variable selection to see how to mitigate that problem.

7.1.4 Feature engineering and functional specification

The omitted variable bias makes it very important to include as much variables as possible if you want to be able to estimate the coefficient as accurately as possible. What you can do is add :

- Exponents to the regressors
- Interactions between regressors

Example with 2 variables $y = b_1x_1 + b_2x_2 + b_3x_1^2 + b_4x_1x_2 + \epsilon$

In this case : $\frac{\partial y}{\partial x1} = b_1 + 2b_3x_1 + b_4x_2$

```
reg <- lm(avgSpeed~ avgPower + I(avgPower^2) + avgBikeCadence +
             distance + I(avgPower*distance)+  avgHr + max20MinPower , data=dat_bike)
summary(reg)
```

```
##
```

```
## Call:
## lm(formula = avgSpeed ~ avgPower + I(avgPower^2) + avgBikeCadence +
##     distance + I(avgPower * distance) + avgHr + max20MinPower,
##     data = dat_bike)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -33.917  -3.359  -0.905   2.912  21.795
##
## Coefficients:
##                            Estimate Std. Error t value Pr(>|t|)
## (Intercept)              1.357e+02  1.401e+01   9.686  < 2e-16 ***
## avgPower                -1.091e-01  1.258e-01  -0.867 0.386163
## I(avgPower^2)            8.780e-04  2.536e-04   3.462 0.000574 ***
## avgBikeCadence          -1.292e+00  7.750e-02 -16.675  < 2e-16 ***
## distance                 4.553e-01  8.268e-02   5.506 5.40e-08 ***
## I(avgPower * distance) -1.417e-03  3.253e-04  -4.356 1.55e-05 ***
## avgHr                   -5.727e-02  3.197e-02  -1.791 0.073726 .
## max20MinPower           -6.822e-02  1.535e-02  -4.444 1.05e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.454 on 614 degrees of freedom
##   (1870 observations deleted due to missingness)
## Multiple R-squared:  0.8372,Adjusted R-squared:  0.8354
## F-statistic: 451.1 on 7 and 614 DF,  p-value: < 2.2e-16
```

```
reg_full <- lm(avgSpeed~(avgPower + avgBikeCadence + distance + avgHr + max20MinPower +
summary(reg_full)
```

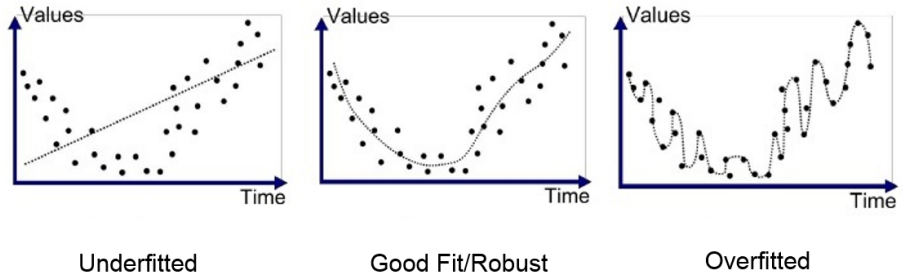
```
##
## Call:
## lm(formula = avgSpeed ~ (avgPower + avgBikeCadence + distance +
##     avgHr + max20MinPower + elevationGain)^2, data = dat_bike)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -16.1520  -1.6039  -0.2125   1.1575  17.2608
##
## Coefficients:
##                                Estimate Std. Error t value Pr(>|t|)
## (Intercept)                   1.275e+02  4.291e+01   2.971 0.003136 **
## avgPower                      2.369e+00  2.726e-01   8.691  < 2e-16 ***
## avgBikeCadence               -8.207e-01  5.625e-01  -1.459 0.145275
## distance                     -2.150e+00  2.698e-01  -7.971 1.40e-14 ***
## avgHr                        -1.461e+00  4.516e-01  -3.235 0.001311 **
```

```
## max20MinPower                -1.421e+00  2.394e-01  -5.935 6.02e-09 ***
## elevationGain                 4.233e-02  1.954e-02   2.167 0.030793 *
## avgPower:avgBikeCadence      -3.128e-02  3.010e-03 -10.392  < 2e-16 ***
## avgPower:distance            -2.345e-03  7.476e-04  -3.137 0.001821 **
## avgPower:avgHr                4.632e-03  1.614e-03   2.869 0.004313 **
## avgPower:max20MinPower        4.628e-04  2.141e-04   2.162 0.031177 *
## avgPower:elevationGain       -2.371e-04  6.007e-05  -3.948 9.20e-05 ***
## avgBikeCadence:distance       3.529e-02  2.862e-03  12.331  < 2e-16 ***
## avgBikeCadence:avgHr          1.174e-02  5.294e-03   2.217 0.027109 *
## avgBikeCadence:max20MinPower  1.757e-02  2.456e-03   7.152 3.64e-12 ***
## avgBikeCadence:elevationGain -5.565e-04  1.794e-04  -3.102 0.002046 **
## distance:avgHr               -2.098e-03  1.182e-03  -1.775 0.076613 .
## distance:max20MinPower       -1.296e-04  5.249e-04  -0.247 0.805185
## distance:elevationGain        4.559e-06  1.288e-05   0.354 0.723472
## avgHr:max20MinPower          -2.513e-03  1.417e-03  -1.773 0.076914 .
## avgHr:elevationGain           1.972e-04  1.008e-04   1.957 0.050990 .
## max20MinPower:elevationGain   1.276e-04  3.286e-05   3.883 0.000119 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.707 on 435 degrees of freedom
##   (2035 observations deleted due to missingness)
## Multiple R-squared:  0.9279,Adjusted R-squared:  0.9244
## F-statistic: 266.5 on 21 and 435 DF,  p-value: < 2.2e-16
```

7.1.5 Variable selection

So far we focused on getting the best coefficient estimates to be able to interpret how features impact our target variable ("explainable AI"), but following the previous logic, adding the more feature the better ! However, when focusing on the best prediction, you are more interested in finding the most general model which will perform well *out of sample* adding more and more variables can lead, as a matter of fact, to an overfitted model, which will hardly generalize.

This is illustration of the bias-variance trade-off which you will see more in depth during the machine learning session.



Regarding regression, avoiding overfitting can be done with variable selection : starting from an extensive model, the procedure will try every feature combination that leads to the best prediction. There are 3 ways of constructing the models :

- backward selection : remove the less useful feature at a time
- forward selection : introduce the most useful feature at a time
- stepwise selection : a mixture of the previous methods

The quality of each model is determined by the AIC or BIC which are a function of the opposite of the log-likelihood (because OLS can also be estimated with MLE) and the number of parameters. The lower this number, the better the model.

We can implement this method easily

```
selection <- step(reg_full)
```

```
## Start:  AIC=1219.11
## avgSpeed ~ (avgPower + avgBikeCadence + distance + avgHr + max20MinPower +
##     elevationGain)^2
##
##                                  Df Sum of Sq    RSS    AIC
## - distance:max20MinPower          1      0.84 5980.1 1217.2
## - distance:elevationGain          1      1.72 5981.0 1217.2
## <none>                                       5979.2 1219.1
## - avgHr:max20MinPower             1     43.21 6022.5 1220.4
## - distance:avgHr                  1     43.30 6022.5 1220.4
## - avgHr:elevationGain             1     52.64 6031.9 1221.1
## - avgPower:max20MinPower          1     64.24 6043.5 1222.0
## - avgBikeCadence:avgHr            1     67.59 6046.8 1222.2
## - avgPower:avgHr                  1    113.17 6092.4 1225.7
## - avgBikeCadence:elevationGain    1    132.29 6111.5 1227.1
## - avgPower:distance               1    135.29 6114.5 1227.3
## - max20MinPower:elevationGain     1    207.20 6186.4 1232.7
## - avgPower:elevationGain          1    214.21 6193.5 1233.2
## - avgBikeCadence:max20MinPower    1    703.12 6682.4 1267.9
## - avgPower:avgBikeCadence         1   1484.41 7463.6 1318.5
```

```
## - avgBikeCadence:distance       1   2090.01 8069.3 1354.1
##
## Step:  AIC=1217.18
## avgSpeed ~ avgPower + avgBikeCadence + distance + avgHr + max20MinPower +
##     elevationGain + avgPower:avgBikeCadence + avgPower:distance +
##     avgPower:avgHr + avgPower:max20MinPower + avgPower:elevationGain +
##     avgBikeCadence:distance + avgBikeCadence:avgHr + avgBikeCadence:max20MinPower +
##     avgBikeCadence:elevationGain + distance:avgHr + distance:elevationGain +
##     avgHr:max20MinPower + avgHr:elevationGain + max20MinPower:elevationGain
##
##                                 Df Sum of Sq    RSS    AIC
## - distance:elevationGain         1      1.07 5981.1 1215.3
## <none>                                       5980.1 1217.2
## - distance:avgHr                 1     42.48 6022.6 1218.4
## - avgHr:max20MinPower            1     46.57 6026.6 1218.7
## - avgHr:elevationGain            1     51.93 6032.0 1219.1
## - avgPower:max20MinPower         1     63.74 6043.8 1220.0
## - avgBikeCadence:avgHr           1     69.39 6049.5 1220.5
## - avgPower:avgHr                 1    120.80 6100.9 1224.3
## - avgBikeCadence:elevationGain   1    196.06 6176.1 1229.9
## - avgPower:elevationGain         1    256.61 6236.7 1234.4
## - max20MinPower:elevationGain    1    356.88 6337.0 1241.7
## - avgPower:distance              1    392.97 6373.0 1244.3
## - avgBikeCadence:max20MinPower   1    763.84 6743.9 1270.1
## - avgPower:avgBikeCadence        1   1555.90 7536.0 1320.9
## - avgBikeCadence:distance        1   2289.03 8269.1 1363.3
##
## Step:  AIC=1215.26
## avgSpeed ~ avgPower + avgBikeCadence + distance + avgHr + max20MinPower +
##     elevationGain + avgPower:avgBikeCadence + avgPower:distance +
##     avgPower:avgHr + avgPower:max20MinPower + avgPower:elevationGain +
##     avgBikeCadence:distance + avgBikeCadence:avgHr + avgBikeCadence:max20MinPower +
##     avgBikeCadence:elevationGain + distance:avgHr + avgHr:max20MinPower +
##     avgHr:elevationGain + max20MinPower:elevationGain
##
##                                 Df Sum of Sq    RSS    AIC
## <none>                                       5981.1 1215.3
## - distance:avgHr                 1     42.29 6023.4 1216.5
## - avgHr:max20MinPower            1     51.09 6032.2 1217.2
## - avgHr:elevationGain            1     60.15 6041.3 1217.8
## - avgBikeCadence:avgHr           1     69.05 6050.2 1218.5
## - avgPower:max20MinPower         1     69.57 6050.7 1218.5
## - avgPower:avgHr                 1    123.62 6104.8 1222.6
## - avgBikeCadence:elevationGain   1    197.63 6178.8 1228.1
## - avgPower:elevationGain         1    326.10 6307.2 1237.5
## - avgPower:distance              1    398.40 6379.5 1242.7
```

```
## - max20MinPower:elevationGain   1    491.53 6472.7 1249.3
## - avgBikeCadence:max20MinPower  1   1102.37 7083.5 1290.6
## - avgPower:avgBikeCadence       1   1747.29 7728.4 1330.4
## - avgBikeCadence:distance       1   2673.93 8655.1 1382.1
```

```
summary(selection)
```

```
##
## Call:
## lm(formula = avgSpeed ~ avgPower + avgBikeCadence + distance +
##     avgHr + max20MinPower + elevationGain + avgPower:avgBikeCadence +
##     avgPower:distance + avgPower:avgHr + avgPower:max20MinPower +
##     avgPower:elevationGain + avgBikeCadence:distance + avgBikeCadence:avgHr +
##     avgBikeCadence:max20MinPower + avgBikeCadence:elevationGain +
##     distance:avgHr + avgHr:max20MinPower + avgHr:elevationGain +
##     max20MinPower:elevationGain, data = dat_bike)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -16.1360  -1.6132  -0.2158   1.1313  17.2782
##
## Coefficients:
##                                Estimate Std. Error t value Pr(>|t|)
## (Intercept)                   1.299e+02  4.223e+01   3.077 0.002223 **
## avgPower                      2.396e+00  2.619e-01   9.149  < 2e-16 ***
## avgBikeCadence               -8.535e-01  5.535e-01  -1.542 0.123771
## distance                     -2.149e+00  2.393e-01  -8.981  < 2e-16 ***
## avgHr                        -1.464e+00  4.466e-01  -3.278 0.001129 **
## max20MinPower                -1.455e+00  2.165e-01  -6.719 5.71e-11 ***
## elevationGain                 4.465e-02  1.713e-02   2.607 0.009441 **
## avgPower:avgBikeCadence      -3.167e-02  2.803e-03 -11.299  < 2e-16 ***
## avgPower:distance            -2.467e-03  4.572e-04  -5.395 1.12e-07 ***
## avgPower:avgHr                4.742e-03  1.578e-03   3.005 0.002806 **
## avgPower:max20MinPower        4.643e-04  2.059e-04   2.255 0.024653 *
## avgPower:elevationGain       -2.364e-04  4.844e-05  -4.881 1.48e-06 ***
## avgBikeCadence:distance       3.518e-02  2.517e-03  13.977  < 2e-16 ***
## avgBikeCadence:avgHr          1.182e-02  5.263e-03   2.246 0.025194 *
## avgBikeCadence:max20MinPower  1.807e-02  2.014e-03   8.975  < 2e-16 ***
## avgBikeCadence:elevationGain -5.812e-04  1.530e-04  -3.800 0.000165 ***
## distance:avgHr               -2.057e-03  1.170e-03  -1.758 0.079484 .
## avgHr:max20MinPower          -2.645e-03  1.369e-03  -1.932 0.053994 .
## avgHr:elevationGain           2.026e-04  9.667e-05   2.096 0.036623 *
## max20MinPower:elevationGain   1.253e-04  2.090e-05   5.993 4.32e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 3.7 on 437 degrees of freedom
##   (2035 observations deleted due to missingness)
## Multiple R-squared:  0.9279,Adjusted R-squared:  0.9247
## F-statistic: 295.8 on 19 and 437 DF,  p-value: < 2.2e-16
```

7.1.6 Exercises

- From the last functional form used, design a graphic that shows the final impact of an increase in power to the average speed, taking the distance into account.
- Design a regression model that will predict best the theoretical average speed for indoor bike activities (that have no speed, no coordinates…)
- Develop a model that will help you identify the measurement errors (thanks to residuals)

7.2 Logitic regression

7.2.1 Mathematical formulation

Logistic regression aims to model a *binary* output. In this case, $y \in \{0, 1\}$ and the previous specification can't apply. We still have a linear relationship, but which applies to the log-odd ratio :

$$log\frac{\mathbb{P}(y=1|x)}{1-\mathbb{P}(y=1|x)} = log\frac{^{p}}{1-p} = x_i b + \epsilon_i$$

This is called the **link function** and working the expression further we find that : $p(x_i; b) = \mathbb{P}(y_i = 1|x_i) = \frac{1}{1+e^{-x_i b}}$

This allows us to derive the likelihood :

$$\mathcal{L}(b) = \prod_{i=1}^{n} p(x_i; b)^{y_i} \cdot (1 - p(x_i; b))^{1-y_i}$$

This expression can be simplified, but there is no exact expression as for the OLS → the optimal solution has to be found via numerical optimization (eg Newton-Raphson).

7.2.2 Implementation in R and interpretation

In R, we use the `glm` function while specifying the family. We model the probability of an activity to be bike or something else, which is a binary variable.

```
logit <- glm(is_bike~distance+duration+elevationGain+avgSpeed+avgHr,data=dat_clean,family = "binc
summary(logit)
```

```
##
## Call:
## glm(formula = is_bike ~ distance + duration + elevationGain +
##     avgSpeed + avgHr, family = "binomial", data = dat_clean)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.1893  -0.7940   0.1199   0.6260   3.7661
##
## Coefficients:
##                 Estimate Std. Error z value Pr(>|z|)
## (Intercept)    8.451e+00  5.147e-01  16.419  < 2e-16 ***
## distance       3.862e-02  7.815e-03   4.942 7.72e-07 ***
## duration       1.955e-03  2.946e-03   0.663   0.5071
## elevationGain -9.304e-05  4.615e-05  -2.016   0.0438 *
## avgSpeed       2.718e-02  1.080e-02   2.517   0.0118 *
## avgHr         -6.839e-02  3.412e-03 -20.041  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 4439.8  on 3204  degrees of freedom
## Residual deviance: 2972.5  on 3199  degrees of freedom
##   (1920 observations deleted due to missingness)
## AIC: 2984.5
##
## Number of Fisher Scoring iterations: 5
```

The sign of the coefficient indicates whether the feature increases the probability for an activity to be a ride ride or not. However, the values cannot be interpreted as directly as in the case of the linear regression. But you can use the exponent of the value of the coefficient and interpret it in terms of odd-ratios. For instance, adding one more kilometer to the average distance multiplies the probability for an activity to be a ride *rather than anything else* by 1.0393797, meaning 4% more chances. In the contrary, an activity that has 1 bpm more than the average HR has 6.6100901 6% less chances to be a ride. This makes sense because, as observed earlier, rides are longer and the heart rate is a bit smaller than for other activities.

7.2.3 Goodness of fit

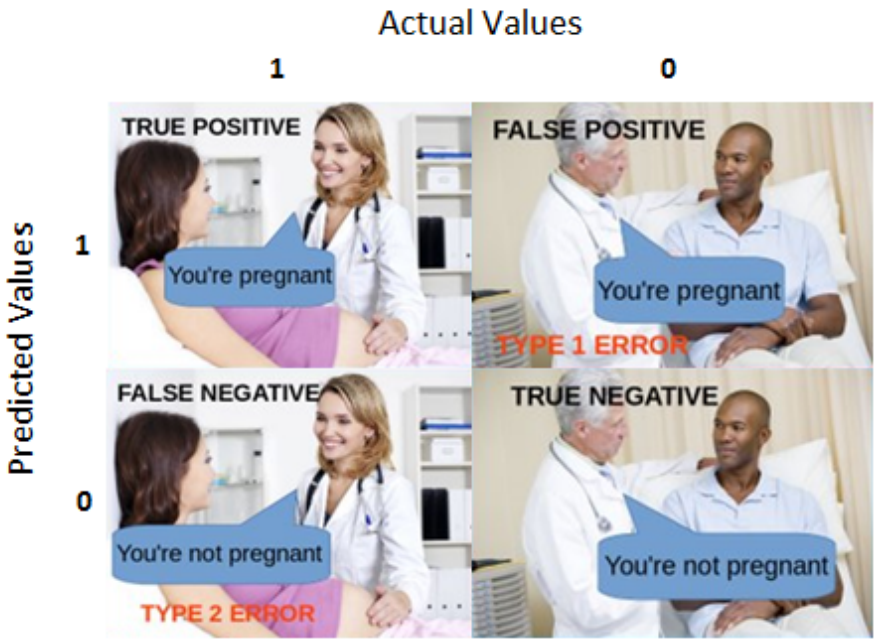
As you might have noticed, there is no R^2 or RMSE in our case, just the AIC (which only allows you to compare different models, not know how good the model is). What we can do is check the fitted values of the model and the actual values

```
pred <- predict(logit,dat_clean,type="response")
pred_bin <- as.numeric(pred>.5)
table(pred_bin,dat_clean$is_bike)
```

```
##
## pred_bin FALSE TRUE
##        0  1418  413
##        1   133 1241
```

And we can compute the accuracy as the sum of correct predictions divided by total number of activities : 0.8296412 You can derive other goodness of fit metrics from the previous **confusion matrix** :

- Sensitivity (recall) : $\frac{TP}{TP+FN}$
- Specificity : $\frac{TN}{TN+FP}$
- Precision : $\frac{TP}{TP+FP}$



Depending on your business use case, you will focus more on one or the other metric. You will cover this in more detail durinng the machine learning week :)