Workshop: R for datascience

Laurent Rouvière

2019, september

CONTENTS

Overview

— *Prerequisites*: Basics on **R**, probability, statistics and computer programming

— *Objectives*: be able to control classical tools for datascience

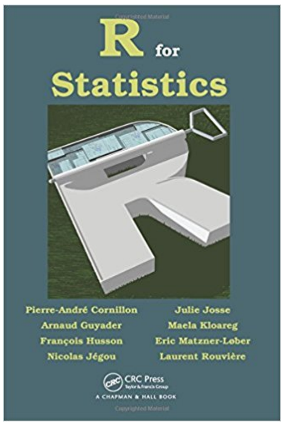
 — import and concatenate datasets, manipulate individuals and variables
 — visualize data
 — implement some of the most important statistical algorithms on real data (IML lecture)

— *Teacher*: Laurent Rouvière, `laurent.rouviere@univ-rennes2.fr`

 — Research interests: nonparametric statistics, statistical learning
 — Teaching: statistics and probability (University and engineer school)
 — Consulting: energy (ERDF), banks, marketing

Resources

- *Slides* and *sheets* (1 sheet=1 or 2 concepts+exercises) available on `https://lrouviere.github.io/R-for-datascience-lecture/`
- The *web*
- *Book*: R for statistics, Chapman & Hall



INTRODUCTION

Why R?

- More and more *data* available in many fields (energy, health, sport, economy. . . .)
- *Data science* collects all the tools which allow to extract informations from data. It includes:
  - to import (merge) datasets
  - to manipulate data (Data Mining)
  - to visualize data (Data Mining+Visualization)
  - to choose and fit models (Data Mining+statistical learning)
  - to visualize models (models are more and more complex. . .)
  - to return and visualize results (web applications)

Important remark

- All these topics can be addressed with R.
- Today, R (data scientits) and Python (computer scientists) are the most important softwares to make data science.

Few words about R

- **R** is a *free software* for statistical computing and graphics.
- It is freely distributed by CRAN (Comprehensive R Archive Network) at the following address: `https://www.r-project.org`.
- Each statistician *contributes* (everybody can create functions and distribute these functions for the community).

Consequence

- The software is always up to date.
- Clearly one of the reasons of the R success.

SOME EXAMPLES

Example: Fisher's iris

```
> data(iris)
> 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  
##                 
##                 
## 
```

Objectives

Goal

Explain *species* by the other variables.

— Species is a *categorical variable.*

— We are faced with a *supervised classification* problem.

Manipulate the data

```
> apply(iris[,1:4],2,mean)
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width 
##     5.843333     3.057333     3.758000     1.199333
> apply(iris[,1:4],2,var)
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width 
##    0.6856935    0.1899794    3.1162779    0.5810063
```

Remark

Non-informative for the problem (highlight differences between species).

Data manipulation with dplyr

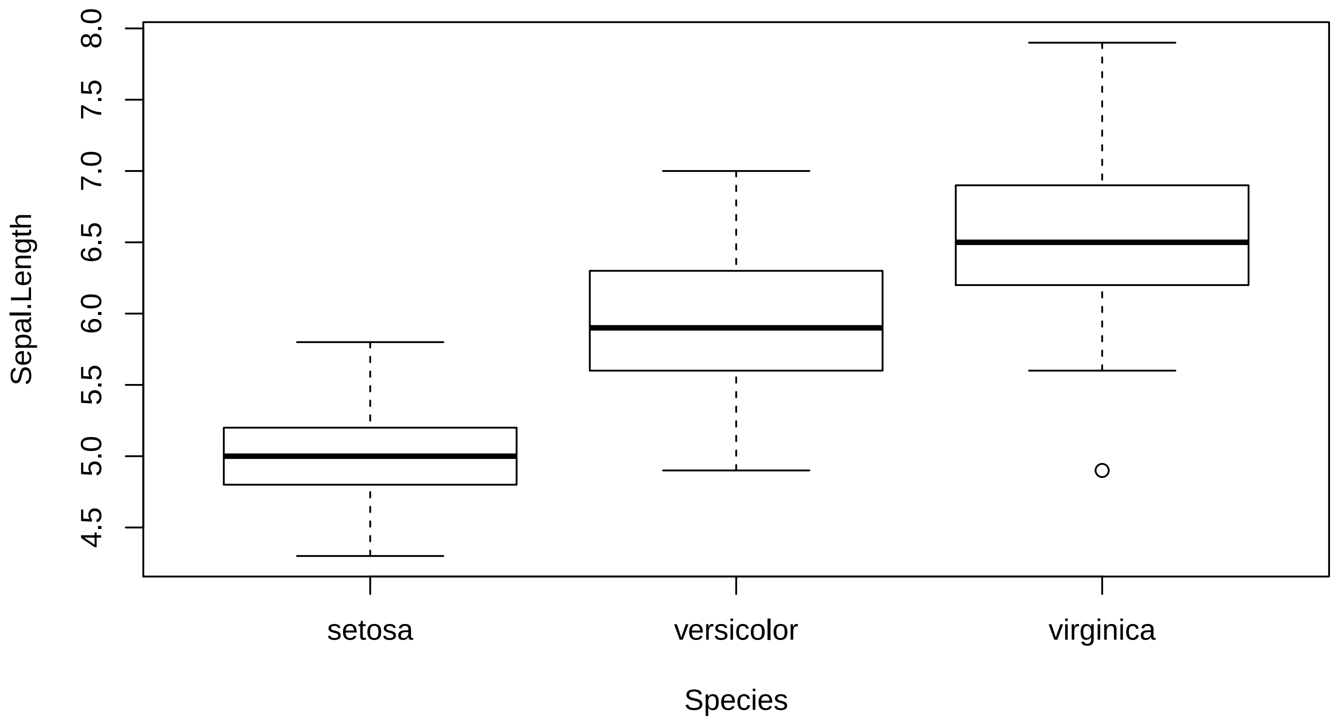
— *dplyr* is powerful R-package to transform and summarize tabular data with rows and columns.

```
> library(dplyr)
> iris %>% group_by(Species) %>% summarise_all(mean)
## # A tibble: 3 x 5
##   Species    Sepal.Length Sepal.Width Petal.Length Petal.Width
##   <fct>             <dbl>       <dbl>        <dbl>       <dbl>
## 1 setosa             5.01        3.43         1.46       0.246
## 2 versicolor         5.94        2.77         4.26       1.33 
## 3 virginica          6.59        2.97         5.55       2.03
```

— *More informative*: we obtain means for each species.

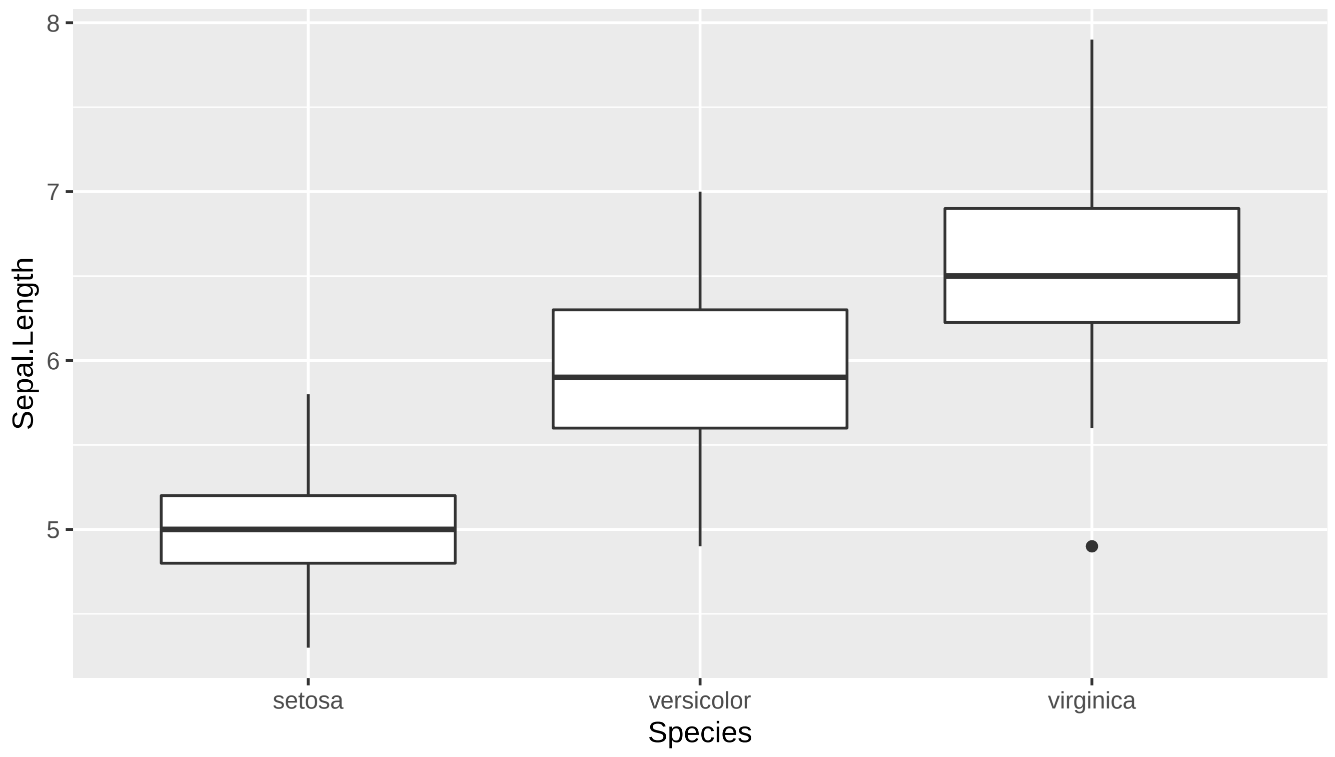
Visualization

```
> boxplot(Sepal.Length~Species,data=iris)
```



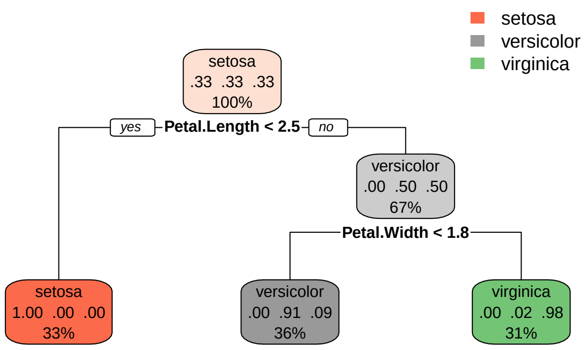
Visualization with ggplot2

```
> library(ggplot2)
> ggplot(iris)+aes(x=Species,y=Sepal.Length)+geom_boxplot()
```



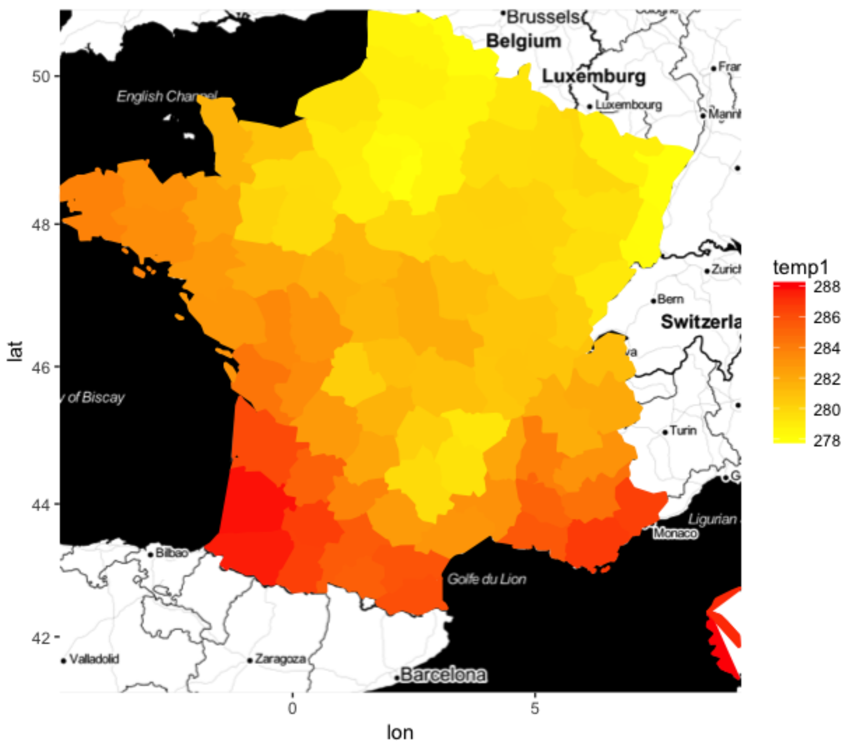
Modelling

```
> library(rpart)
> tree <- rpart(Species~.,data=iris)
> library(rpart.plot)
> rpart.plot(tree)
```



Maps with ggmap

— *Goal*: draw a map of the temperatures for france.

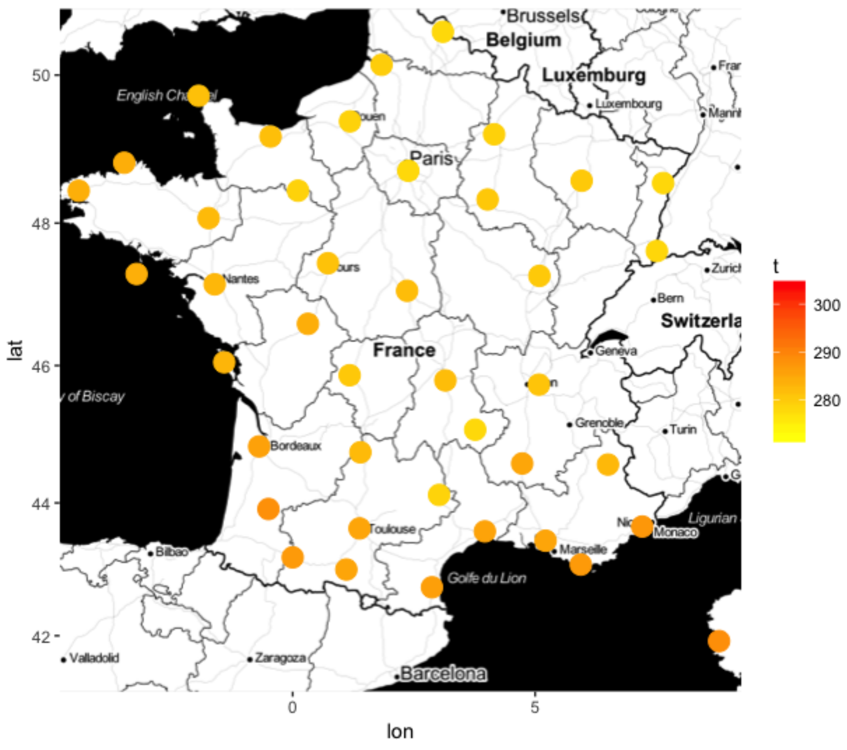


Load the data + background map

— Data are downloaded from meteofrance (temperatures for about 60 stations).

```
> donnees <- fread("https://donneespubliques.meteofrance.fr/
+                 donnees_libres/Txt/Synop/synop.2017082815.csv")
> station <- fread("https://donneespubliques.meteofrance.fr/
+                 donnees_libres/Txt/Synop/postesSynop.csv")
> fond <- get_map("France",maptype="toner",zoom=6)
> ggmap(fond)+geom_point(data=D,
+        aes(y=Latitude,x=Longitude,color=t),size=5)+
+        scale_color_continuous(low="yellow",high="red")
```

A first map



Model

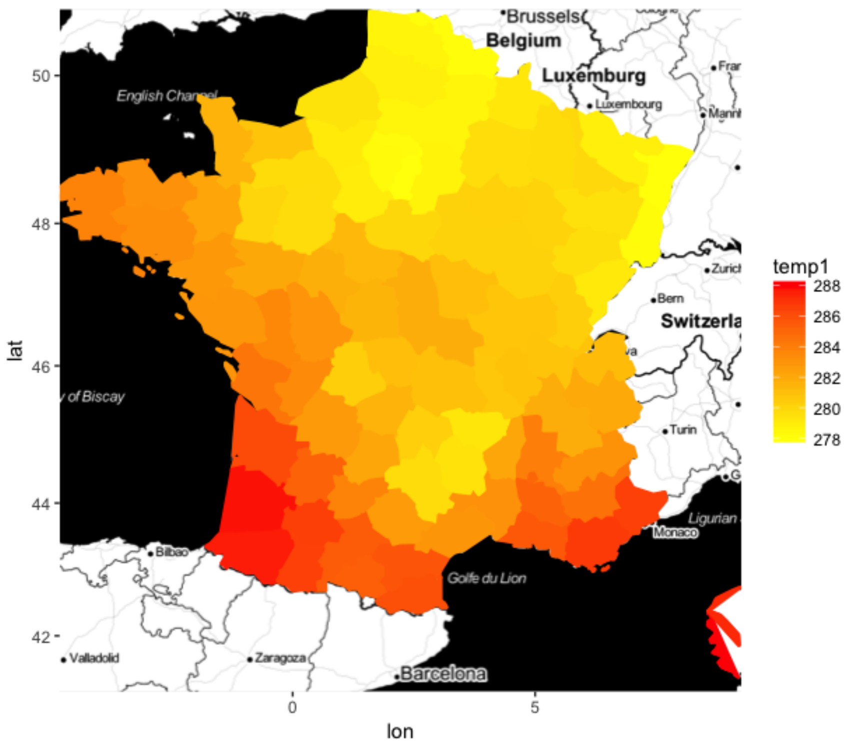
— model of *nearest neighbors* to estimate temperatures for all longitudes and latitudes.

```
> library(FNN)
> mod <- knn.reg(train=D[,.(Latitude,Longitude)],y=D[,t],
+                 test=Test1[,.(Latitude,Longitude)],k=1)$pred
```

— Visualisation with *ggmap*.

```
> library(ggmap)
> ggmap(fond)+geom_polygon(data=Test5,
+   aes(y=Latitude,x=Longitude,
+       fill=temp1,color=temp1,group=dept),size=1)+
+   scale_fill_continuous(low="yellow",high="red")+
+   scale_color_continuous(low="yellow",high="red")
```

The temperature map



Interactive web apps with shiny

- *Shiny* is a R package that makes it easy to build interactive web apps straight from R.
- Example: basic graphics for a dataset.

```
> library(shiny)
> runApp('desc_app.R')
```

OUTLINE

In this workshop

- 15 hours for 5 (or 6) topics
- 1 topic = slides + sheet (notebook) to complete (add comments and do exercises)

R Notebook

- document which combines R code and comments.
- code can be executed independently and interactively, with output visible immediately beneath the input.
- very nice to make high quality reports.

Schedule

- *Introduction to R* lecture: basics of R (objects, apply, matrices, date, control flow statements)

R for datascience

- Tuto 1: Rstudio (notebook and presentations) (1 hour)
- Tuto 2: R objects (review, 1 or 2 hours)
- Tuto 3: data manipulation with dplyr (4 hours)
- Tuto 4: data visualization with ggplot2 (4 hours)
- Tuto 5: mapping with leaflet (2 hours)
- Tuto 6: modeling with R (transition with the ISL lecture, 2 hours).

Assessment

- When ???
- combined with the machine learning lecture
  - Multiple choice test (50%)
  - Data science project (50%)

Working

- Require personal efforts.
- *To Practice*, to make mistakes and to correct these mistakes: *only way* to learn a sofware.
- You need to *work alone* between the sessions.
- Everyone can develop at its own pace (the goal is to progress, not to become a specialist of R in 15 hours), and *ask questions* during the sessions.
- I'm here to (try) to answer.

RSTUDIO, RMARKDOWN AND R-PACKAGES

Rstudio

- **RStudio** is an *integrated development environment* for R.
- It makes **R** easier to practice.
- It includes a console, syntax-highlighting editor that supports direct code execution, tools for plotting, history, debugging and workspace management.
- It is also *freely distributed* at the address `https://www.rstudio.com`.

The screen is divided into 4 windows:

- Console: where you enter command and see output
- Workspace and History: show the active object
- Files Plots...: show all files anf folders in the workspace, see output graph, install packages…
- R script: where you keep a record of your work. Don't forget to *regularly save this files*!

Rmarkdown

What is Rmarkdown

- An Rmarkdown (.Rmd) file is a record of your work.
- It contains code, output and comments of your work.
- It produces high quality report in many format (text documents, slides, etc...).
- These slides have been made with *Rmarkdwon*.
- *Reproducible Research*: at the click of a button, you can rerun the code in an R Markdown file to reproduce your work and export the results as a finished report.
- *Dynamic Documents*: you can choose to export the finished report in a wide range of outputs, including html, pdf, MS Word, or RTF documents; html or pdf based slides, Notebooks, and more.

Packages

- *Set of R programs* which supplements and enhances the functions of **R**
- Generally reserved for specific methods or fields of applications
- More than *15 000* packages
- Clearly one of the reasons of the success of R.

2 steps

- Installation: install.packages(package.name) (just one time)
- Loading: library(package.name) (each time)
- You can also use the package icon in Rstudio.

$\Longrightarrow$ work on *Tuto 1*.

Tuto 1

- Download the *.Rmd* file *Tuto1.Rmd* in `https://lrouviere.github.io/stat_grand_dim/`
- Open the file in *Rstudio*.
- Click on *File* + *Reopen with encoding* and select utf8
- Add in the begining of the file

```
---
title: 'Tuto 1: RStudio environment'
output: html_notebook
---
```

- Save the file in the repository of your choice and click on *Preview*.
- *Read* the tutorial and *do exercices*.

R OBJECTS (REVIEW)

Numeric and characters

- Numeric (easy)

```
> x <- pi
> x
## [1] 3.141593
> is.numeric(x)
## [1] TRUE
```

- Characters

```
> b <- "X"
> paste(b,1:5,sep="")
## [1] "X1" "X2" "X3" "X4" "X5"
```

Vectors

- *Creation*: **c**, **seq**, **rep**

```
> x1 <- c(1,3,4)
> x2 <- 1:5
> x3 <- seq(0,10,by=2)
> x4 <- rep(x1,3)
> x5 <- rep(x1,3,each=3)
```

- *Extraction*:

```
> x3[c(1,3,4)] # same as x3[x1]
## [1] 0 4 6
```

Logical

```
> 1<2
## [1] TRUE
> 1==2
## [1] FALSE
> 1!=2
## [1] TRUE
```

```
> x <- 1:3
> test <- c(TRUE,FALSE,TRUE)
> x[test]
## [1] 1 3
```

```
> size <- runif(5,150,190) #5 sizes randomly generated between
>                  #150 and 190
> size
## [1] 178.8362 185.0309 180.4393 185.4450 168.2592
```

Problem

Select size more than 174.

```
> size>174
## [1]  TRUE  TRUE  TRUE  TRUE FALSE
> size[size>174]
## [1] 178.8362 185.0309 180.4393 185.4450
```

Factors

— For *categorical variables* in datasets:

```
> x1 <- factor(c("a","b","b","a","a"))
> x1
## [1] a b b a a
## Levels: a b
> levels(x1)
## [1] "a" "b"
```

Data not properly collected

— Assume that data are collected: 0=man, 1=woman

```
> X <- c(1,1,0,0,1)
> summary(X)
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     0.0     0.0     1.0     0.6     1.0     1.0
```

— *Problem*: **R** reads X as a continuous vector $\Longrightarrow$ it could generate problem for satistical study.

— Solution:

```
> X <- as.factor(X)
> levels(X) <- c("man","woman")
> X
## [1] woman woman man   man   woman
## Levels: man woman
> summary(X)
##   man woman
##     2     3
```

Matrix

— Creation

```
> m <- matrix(1:4,nrow=2,byrow=TRUE)
> m
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
```

— Extraction

```
> m[1,2]
> m[1,] #First row
> m[,2] #Second column
```

List

— Allow to manage different objects

```
> mylist <- list(vector=1:5,mat=matrix(1:8,nrow=2))
> mylist
## $vector
## [1] 1 2 3 4 5
##
## $mat
##      [,1] [,2] [,3] [,4]
## [1,]    1    3    5    7
## [2,]    2    4    6    8
```

— Extraction:

```
> mylist[[1]]
> mylist$vector
> mylist[["vector"]]
```

Dataframe

— Objects for representing *data* in **R**

```
> name <- c("Paul","Mary","Steven","Charlotte","Peter")
> sex <- c(0,1,0,1,0)
> size <- c(180,165,168,170,175)
> data <- data.frame(name,sex,size)
> data
##        name sex size
## 1      Paul   0  180
## 2      Mary   1  165
## 3    Steven   0  168
## 4 Charlotte   1  170
## 5     Peter   0  175
```

```
> summary(data)
##          name        sex           size
##  Charlotte:1   Min.   :0.0   Min.   :165.0
##  Mary     :1   1st Qu.:0.0   1st Qu.:168.0
##  Paul     :1   Median :0.0   Median :170.0
##  Peter    :1   Mean   :0.4   Mean   :171.6
##  Steven   :1   3rd Qu.:1.0   3rd Qu.:175.0
##                Max.   :1.0   Max.   :180.0
```

Problem

Here **sex** is considered as a *numeric variable*. It is a categorical variable.

```
> data$sex <- as.factor(data$sex)
> levels(data$sex) <- c("man","woman")
> summary(data)
##         name      sex         size      
##  Charlotte:1   man  :3   Min.   :165.0  
##  Mary     :1   woman:2   1st Qu.:168.0  
##  Paul     :1             Median :170.0  
##  Peter    :1             Mean   :171.6  
##  Steven   :1             3rd Qu.:175.0  
##                          Max.   :180.0
```

Problem

Here **name** is considered as a *variable*. It is the individual names (the ID of individuals)!

```
> row.names(data) <- data$name
> data <- data[,-1] #delete column name
> data
##             sex size
## Paul        man  180
## Mary      woman  165
## Steven      man  168
## Charlotte woman  170
## Peter       man  175
```

Conclusion

We always have to check that data are correctly interpreted by **R** (with **summary** for instance).

Tibbles

— A *tibble* is a modern reimagining of the *data.frame*, keeping what time has proven to be effective, and throwing out what is not.

— We need to load the package *tidyverse* to use tibble.

Example: data frame

```
> name <- c("Paul","Mary","Steven","Charlotte","Peter")
> sex <- c(0,1,0,1,0)
> size <- c(180,165,168,170,175)
> age <- c("old","young","young","old","old")
> data <- data.frame(name,sex,size,age)
> summary(data)
##         name        sex            size          age    
##  Charlotte:1   Min.   :0.0   Min.   :165.0   old  :3  
##  Mary     :1   1st Qu.:0.0   1st Qu.:168.0   young:2  
##  Paul     :1   Median :0.0   Median :170.0            
##  Peter    :1   Mean   :0.4   Mean   :171.6            
##  Steven   :1   3rd Qu.:1.0   3rd Qu.:175.0            
##                Max.   :1.0   Max.   :180.0
```

Example: tibble

```
> library(tidyverse)
> data1 <- tibble(name,sex,size,age)
> summary(data1)
##      name                sex            size            age
##  Length:5           Min.   :0.0   Min.   :165.0   Length:5
##  Class :character   1st Qu.:0.0   1st Qu.:168.0   Class :character
##  Mode  :character   Median :0.0   Median :170.0   Mode  :character
##                     Mean   :0.4   Mean   :171.6
##                     3rd Qu.:1.0   3rd Qu.:175.0
##                     Max.   :1.0   Max.   :180.0
```

dataframe vs tibbles

Main difference: no factor in tibbles.

$\Longrightarrow$ work on *tuto 2*.

READING DATA FROM FILES

- Data is generally contained within a *file* in which individuals are presented in rows and variables in columns.
- Functions **read.table** and **read.csv** allow to import data from *.txt* or *.csv* files.
- **.xls** files need to be *converted* into **.csv** files.

```
> data <- read.table("file",...)
> data <- read.csv("file",...)
```

- ... corresponds to many options. Options are *very important* since the date file always contains *specificities* (missing data, names of the variables...)

Indicating the path

- The **data file** needs to be located in the working directory. Otherwise, we have to indicate the *path* in **read.table**.
- *Example*: Read the file *data.csv* located in /lectureR/Part1 :
  - Change the working directory

```
> setwd("~/lectureR/Part1")
> df <- read.csv("data.csv",...)
```

- Specify the directory in **read.csv**

```
> df <- read.csv("~/lecture_R/Part1/data.csv",...)
```

- Use the **file.path** function

```
> path <- file.path("~/lecture_R/Part1/", "data.csv")
> df <- read.csv(path,...)
```

Some important options

The are many important *options* in **read.table** and **read.csv**:

- **sep**: the field separation character (space, comma...)
- **dec**: the character used for decimal points (comma, points...)
- **header**: a logical value indicating whether the file contains the names of the variables as its first line
- **row.names**: a vector of row names (to identify indivuals if needed)
- **na.strings**: a character vector of strings which are to be interpreted as NA values.
- ...

Example

— File *data_imp.txt*

name;size;age
John;174;32
Peter;?;28
Mary;165.5;NA

Characteristics

— 3 variables

— First line=name of the variables

— Missing values: NA, ?

First try

```
> path <- file.path("~COURS/EDHEC/R/SLIDES/", "data_imp.txt")
```

```
> df <- read.table(path)
> summary(df)
##              V1
##  John;174;32  :1
##  Mary;165.5;NA:1
##  name;size;age:1
##  Peter;?;28   :1
```

Problem

R considers four line with one column!

Solution

```
> df <- read.table(path,header=TRUE,sep=";",dec=".",
+                   na.strings = c("NA","?"),row.names = 1)
> df
##        size age
## John  174.0  32
## Peter    NA  28
## Mary  165.5  NA
> summary(df)
##       size             age
##  Min.   :165.5   Min.   :28
##  1st Qu.:167.6   1st Qu.:29
##  Median :169.8   Median :30
##  Mean   :169.8   Mean   :30
##  3rd Qu.:171.9   3rd Qu.:31
##  Max.   :174.0   Max.   :32
##  NA's   :1       NA's   :1
```

readr package

— This package makes *data importation easier.*

— It includes **read_table** and **read_csv** functions instead of **read.table** and **read.csv** (underscores instead of points).

— In *Rstudio*, we can read data with readr by clicking on the **Import Dataset** icon (it does not work when things are too complicated).

Other tools to import data

— *readxl*: for xls files

— *sas7bdat*: for sas dataset

— *foreign*: for SPSS or STATA datasets

— *jsonlite*: for json files

— *rvest*: webscrapping (to import data from website)

Combine tables

— Information comes (always) from *several data tables*.
— We need to *correctly merge these tables* before a statistical analysis.
— *Standard R functions*: rbind, cbind, cbind.data.frame, merge...
— *Tidyverse functions*: bind_rows, bind_cols, left_join, inner_join (from *dplyr* or *tidyverse* package).

An example with 2 tables

```
> df1
## # A tibble: 4 x 2
##   name  nation
##   <chr> <chr>
## 1 Peter USA
## 2 Mary  GB
## 3 John  Aus
## 4 Linda USA
> df2
## # A tibble: 3 x 2
##   name    age
##   <chr> <dbl>
## 1 John     35
## 2 Mary     41
## 3 Fred     28
```

Goal

One dataset with three columns: name, nation and age.

bind_rows

```
> bind_rows(df1,df2)
## # A tibble: 7 x 3
##   name  nation   age
##   <chr> <chr>  <dbl>
## 1 Peter USA       NA
## 2 Mary  GB        NA
## 3 John  Aus       NA
## 4 Linda USA       NA
## 5 John  <NA>      35
## 6 Mary  <NA>      41
## 7 Fred  <NA>      28
```

$\Longrightarrow$ *not a safe choice* here (two lines for some individuals).

full_join

```
> full_join(df1,df2)
## # A tibble: 5 x 3
##   name  nation   age
##   <chr> <chr>  <dbl>
## 1 Peter USA       NA
## 2 Mary  GB        41
## 3 John  Aus       35
## 4 Linda USA       NA
## 5 Fred  <NA>      28
```

$\Longrightarrow$ we keep all the individuals (NA are added for missing data)

left_join

```
> left_join(df1,df2)
## # A tibble: 4 x 3
##   name  nation   age
##   <chr> <chr>  <dbl>
## 1 Peter USA       NA
## 2 Mary  GB        41
## 3 John  Aus       35
## 4 Linda USA       NA
```

$\Longrightarrow$ we keep only individuals of the first (left) dataset.

inner_join

```
> inner_join(df1,df2)
## # A tibble: 2 x 3
##   name  nation   age
##   <chr> <chr>  <dbl>
## 1 Mary  GB        41
## 2 John  Aus       35
```

$\Longrightarrow$ we keep only individuals for which *both* nation and age are observed.

Conclusion

— Many options to merge datasets.
— Find the good function according to our problem.

$\Longrightarrow$ work on *tuto 3 - Part 1*

DATA MANIPULATION WITH DPLYR

— *dplyr* is a powerful R-package to *transform and summarize* tabular data with rows and columns.
— It offers a clear syntax (based on a grammar) to manipulate data.
— For instance, to compute the mean of *Sepal.Length* for *setosa*, we usually use

```
> mean(iris[iris$Species=="setosa",]$Sepal.Length)
## [1] 5.006
```

— We can do the same with dplyr

```
> library(dplyr)
> iris %>% filter(Species=="setosa") %>%
+   summarise(mean(Sepal.Length))
##   mean(Sepal.Length)
## 1              5.006
```

Grammar

dplyr contains a *grammar* with the following verbs:

— select() select columns (variables)
— filter() filter rows (individuals)
— arrange() re-order or arrange rows
— mutate() create new columns (new variables)
— summarise() summarise values (compute statistics summaries)
— group_by() allows for group operations in the "split-apply-combine" concept

Dont't forget to look at the **cheat sheet**

Select

Goal

To select variables.

```
> df <- select(iris,Sepal.Length,Petal.Length)
> head(df)
##   Sepal.Length Petal.Length
## 1          5.1          1.4
## 2          4.9          1.4
## 3          4.7          1.3
## 4          4.6          1.5
## 5          5.0          1.4
## 6          5.4          1.7
```

Filter

Goal

To filter individuals.

```
> df <- filter(iris,Species=="versicolor")
> head(df)
##   Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
## 1          7.0         3.2          4.7         1.4 versicolor
## 2          6.4         3.2          4.5         1.5 versicolor
## 3          6.9         3.1          4.9         1.5 versicolor
## 4          5.5         2.3          4.0         1.3 versicolor
## 5          6.5         2.8          4.6         1.5 versicolor
## 6          5.7         2.8          4.5         1.3 versicolor
```

Arrange

Goal

To order individuals according to a variable.

```
> df <- arrange(iris,Sepal.Length)
> head(df)
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          4.3         3.0          1.1         0.1  setosa
## 2          4.4         2.9          1.4         0.2  setosa
## 3          4.4         3.0          1.3         0.2  setosa
## 4          4.4         3.2          1.3         0.2  setosa
## 5          4.5         2.3          1.3         0.3  setosa
## 6          4.6         3.1          1.5         0.2  setosa
```

Mutate

Goal

To define new variables in the dataset.

```
> df <- mutate(iris,diff_petal=Petal.Length-Petal.Width)
> head(select(df,Petal.Length,Petal.Width,diff_petal))
##   Petal.Length Petal.Width diff_petal
## 1          1.4         0.2        1.2
## 2          1.4         0.2        1.2
## 3          1.3         0.2        1.1
## 4          1.5         0.2        1.3
## 5          1.4         0.2        1.2
## 6          1.7         0.4        1.3
```

Summarise

Goal

To compute statistical summaries.

```
> summarise(iris,mean=mean(Petal.Length),var=var(Petal.Length))
##    mean      var
## 1 3.758 3.116278
```

group_by

Goal

To apply operations for group of data.

```
> summarise(group_by(iris,Species),mean(Petal.Length))
## # A tibble: 3 x 2
##   Species    `mean(Petal.Length)`
##   <fct>                     <dbl>
## 1 setosa                     1.46
## 2 versicolor                 4.26
## 3 virginica                  5.55
```

The pipe operator

- The pipe operator %>% allows to organize commands *step by step*.
- For instance, to calculate the **mean** of variable **Sepal.Length** for **setosa**, we can do

```
> mean(iris[iris$Species=="setosa","Sepal.Length"])
## [1] 5.006
```

or (more readable)

```
> df1 <- iris[iris$Species=="setosa",]
> df2 <- df1$Sepal.Length
> mean(df2)
## [1] 5.006
```

or (more readable with **dplyr**)

```
> df1 <- filter(iris,Species=="setosa")
> df2 <- select(df1,Sepal.Length)
> summarize(df2,mean(Sepal.Length))
##   mean(Sepal.Length)
## 1              5.006
```

— With the *pipe operator*, we expand the operations:

1. the data

```
> iris
```

2. Filter individuals according to setosa specie

```
> iris %>% filter(Species=="setosa")
```

3. Select the variable of interest

```
> iris %>% filter(Species=="setosa") %>% select(Sepal.Length)
```

4. Compute the mean

```
> iris %>% filter(Species=="setosa") %>%
+   select(Sepal.Length)%>% summarize_all(mean)
##   Sepal.Length
## 1        5.006
```

More generally

— The pipe opartor %>% *merge* the left object with the first component of the right object.

```
> X <- as.numeric(c(1:10,"NA"))
> mean(X,na.rm = TRUE)
## [1] 5.5
```

or equivalently

```
> X %>% mean(na.rm=TRUE)
## [1] 5.5
```

Reshaping data

— Some statistical analysis requires a *particular shape* for the data
— A toy example

```
> df <- iris %>% group_by(Species) %>%
+   summarize_all(funs(mean))
> head(df)
## # A tibble: 3 x 5
##   Species    Sepal.Length Sepal.Width Petal.Length Petal.Width
##   <fct>             <dbl>       <dbl>        <dbl>       <dbl>
## 1 setosa             5.01        3.43         1.46       0.246
## 2 versicolor         5.94        2.77         4.26       1.33
## 3 virginica          6.59        2.97         5.55       2.03
```

gather

— Gather columns into rows with *gather*:

```
> df1 <- df %>% gather(key=variable,value=value,-Species)
> head(df1)
## # A tibble: 6 x 3
##   Species    variable     value
##   <fct>      <chr>        <dbl>
## 1 setosa     Sepal.Length  5.01
## 2 versicolor Sepal.Length  5.94
## 3 virginica  Sepal.Length  6.59
## 4 setosa     Sepal.Width   3.43
## 5 versicolor Sepal.Width   2.77
## 6 virginica  Sepal.Width   2.97
```

Remark

Same information with a different shape.

Spread

— Spread rows into columns with *spread*:

```
> df1 %>% spread(variable,value)
## # A tibble: 3 x 5
##   Species    Petal.Length Petal.Width Sepal.Length Sepal.Width
##   <fct>             <dbl>       <dbl>        <dbl>       <dbl>
## 1 setosa             1.46       0.246         5.01        3.43
## 2 versicolor         4.26       1.33          5.94        2.77
## 3 virginica          5.55       2.03          6.59        2.97
```

Separate

— *Separate* one column into several

```
> df <- tibble(date=as.Date(c("01/03/2015","05/18/2017",
+           "09/14/2018"),"%m/%d/%Y"),temp=c(18,21,15))
```

```
> df1 <- df %>% separate(date,into = c("year","month","day"))
> df1
## # A tibble: 3 x 4
##   year  month day    temp
##   <chr> <chr> <chr> <dbl>
## 1 2015  01    03       18
## 2 2017  05    18       21
## 3 2018  09    14       15
```

Unite

— *Unite* several columns into one

```
> df1 %>% unite(date,year,month,day,sep="/")
## # A tibble: 3 x 2
##   date        temp
##   <chr>      <dbl>
## 1 2015/01/03    18
## 2 2017/05/18    21
## 3 2018/09/14    15
```

$\Longrightarrow$ work on *tuto 3, part 2.*

VISUALIZE DATA

— *Graphs* are often the starting point for statistical analysis.

— One of the main advantages of **R** is how *easy* it is for the user to create many different kinds of graphs.

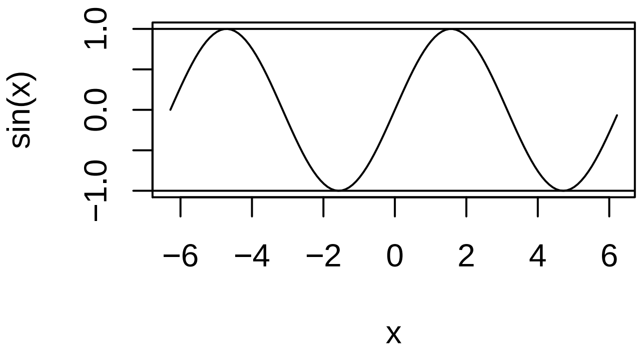
— We begin by a (short) review on *conventional graphs*,

— followed by an examination of some more complex representations, especially with *ggplot2 package*.

The plot function

— It is a *generic* function to represent all kind of data.
— For a *scatter plot*, we have to specify a vector for the x-axis and a vector for the y axis.

```
> x <- seq(-2*pi,2*pi,by=0.1)
> plot(x,sin(x),type="l",xlab="x",ylab="sin(x)")
> abline(h=c(-1,1))
```



Graphs for datasets

— *Many kind of representations* are needed according to the variables we want to visualize.

— Histogram for continuous variables, barplot for categorical variables.

— scatterplot for 2 continous variables.

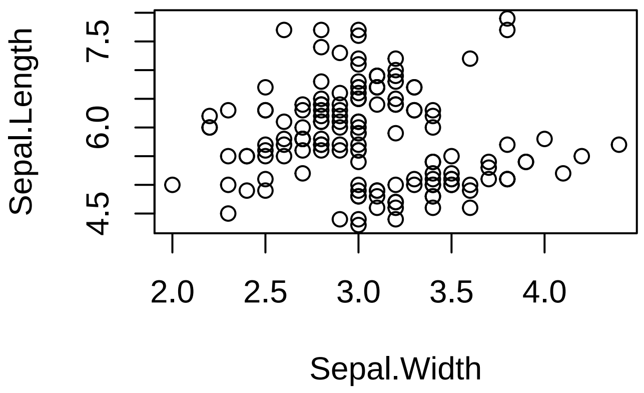
— boxplot to visualize distributions.

Fortunately

There is a R function for all the representations.

Scatter plot for dataset

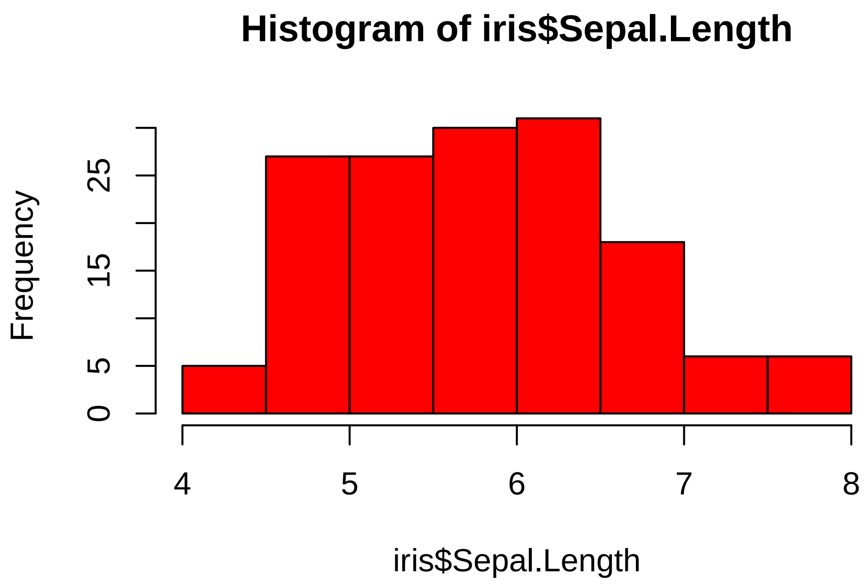
```
> plot(Sepal.Length~Sepal.Width,data=iris)
```



```
> #same as
> plot(iris$Sepal.Width,iris$Sepal.Length)
```

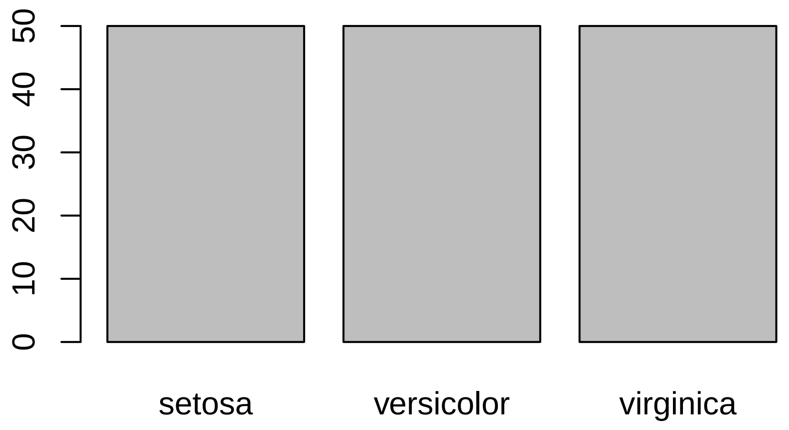
Histogram for continous variable

```
> hist(iris$Sepal.Length,col="red")
```



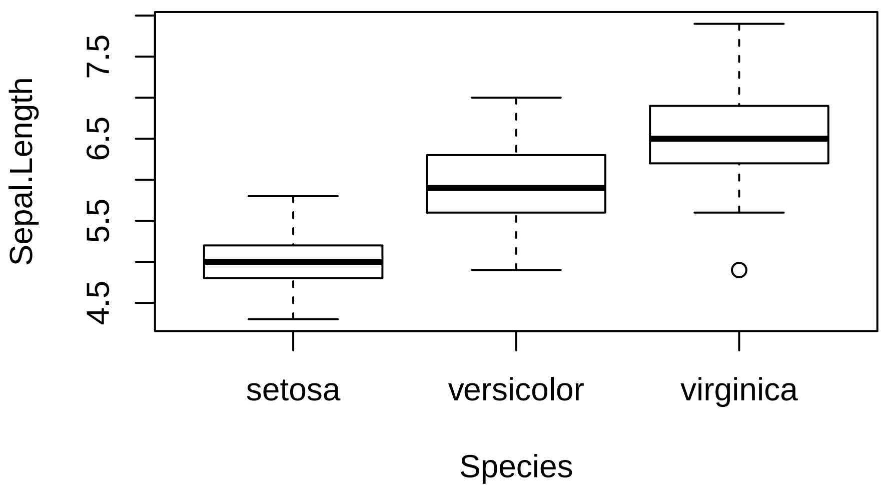
Barplot for categorical variables

```
> barplot(table(iris$Species))
```



Boxplot

```
> boxplot(Sepal.Length~Species,data=iris)
```



VISUALIZATION WITH GGPLOT2

— *ggplot2* is a plotting system for R based on the grammar of graphics (as **dplyr** to manipulate data).

— Graphs ggplot are clearly nice looking (conventionnal R graphs are not always very nice).

For a given dataset, a graph is defined from many **layers**. We have to specify:

— the *data*
— the *variables* we want to plot
— the *type of representation* (scatterplot, boxplot...).

Ggplot graphs are defined from these layers. We indicate

— the data with **ggplot**
— the variables with **aes** (aesthetics)
— the type of representation with **geom_**

The grammar

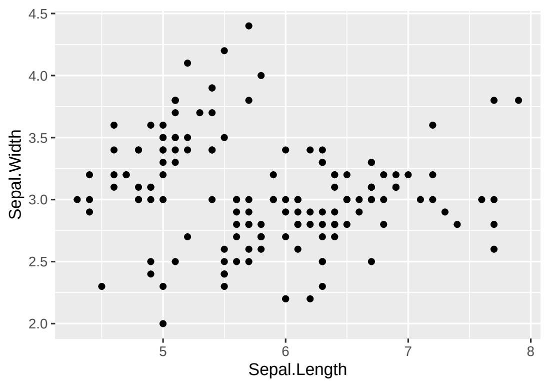
Main elements of the grammar are:

— Data (ggplot): the *dataset*, it should be a dataframe or a tibble
— Aesthetics (aes): to describe the way that *variables* in the data are mapped. All the variables used in the graph should be specified in **aes**
— Geometrics (geom_...): to control the *type* of plot
— Statistics (stat_...): to describe *transformation* of the data
— Scales (scale_...): to *control the mapping* from data to aesthetic attributes (change colors, size...)

All these elements are combined with a +.

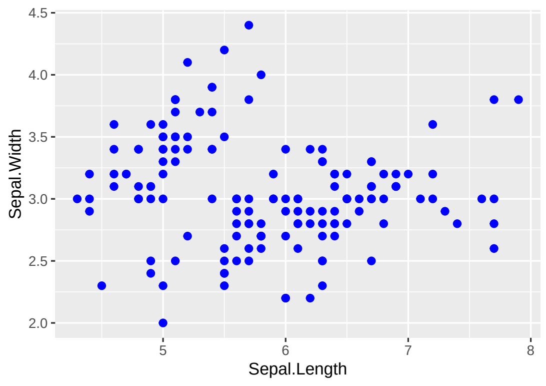
An example

```
> ggplot(iris)+aes(x=Sepal.Length,y=Sepal.Width)+geom_point()
```



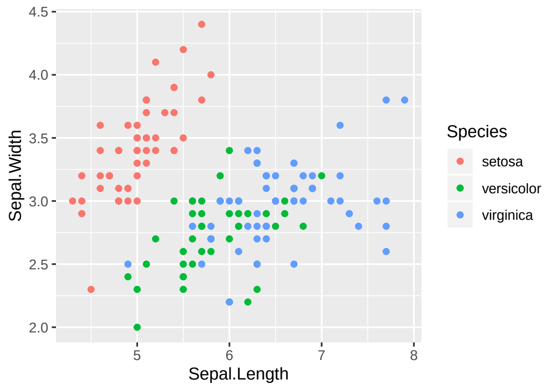
Color and size

```
> ggplot(iris)+aes(x=Sepal.Length,y=Sepal.Width)+
+   geom_point(color="blue",size=2)
```



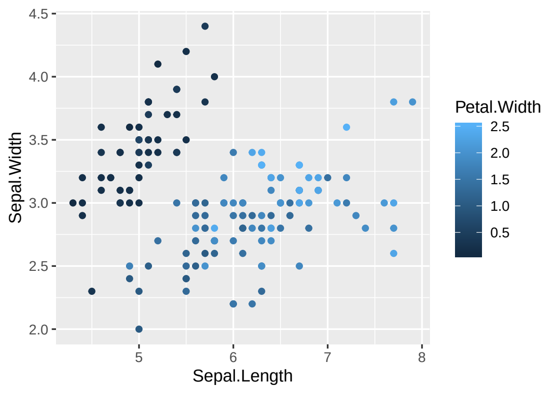
Color by (categorical) variable

```
> ggplot(iris)+aes(x=Sepal.Length,y=Sepal.Width,
+                  color=Species)+geom_point()
```



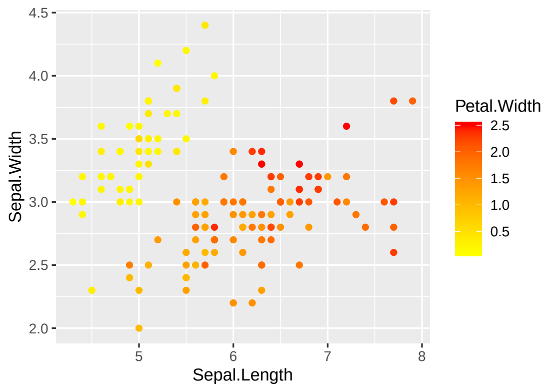
Color by (continous) variable

```
> ggplot(iris)+aes(x=Sepal.Length,y=Sepal.Width,
+                  color=Petal.Width)+geom_point()
```



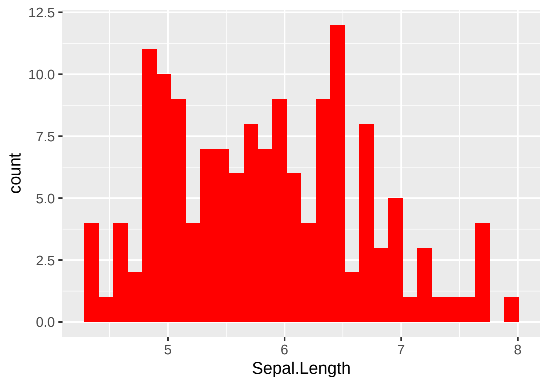
Color by (continous) variable

```
> ggplot(iris)+aes(x=Sepal.Length,y=Sepal.Width,
+                  color=Petal.Width)+geom_point()+
+             scale_color_continuous(low="yellow",high="red")
```



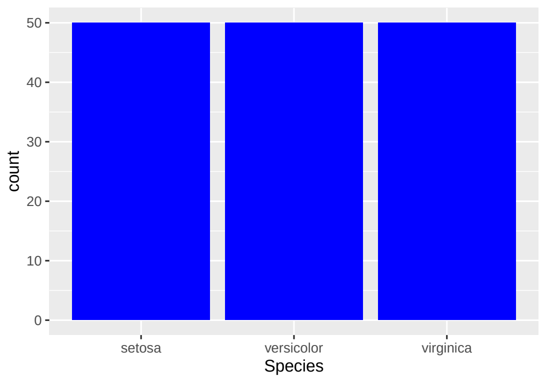
Histogram

```
> ggplot(iris)+aes(x=Sepal.Length)+geom_histogram(fill="red")
```



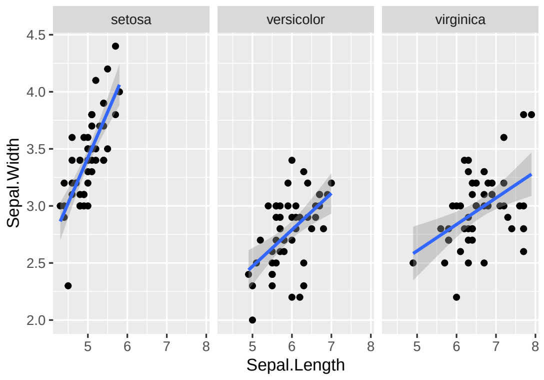
Barplot

```
> ggplot(iris)+aes(x=Species)+geom_bar(fill="blue")
```



Facetting (more complex)

```
> ggplot(iris)+aes(x=Sepal.Length,y=Sepal.Width)+geom_point()+
+         geom_smooth(method="lm")+facet_wrap(~Species)
```

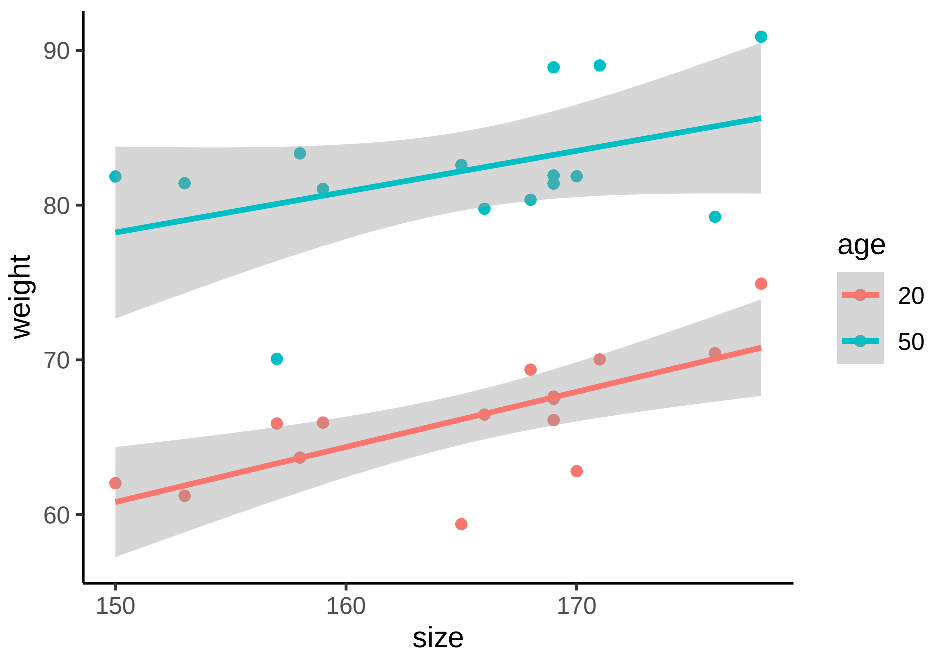


Combining ggplot with dplyr

- One has to build a *good dataframe (or tibble)* to obtain an efficient graph.
- For instance

```
> head(df)
## # A tibble: 6 x 3
##    size weight.20 weight.50
##   <dbl>     <dbl>     <dbl>
## 1   153      61.2      81.4
## 2   169      67.5      81.4
## 3   168      69.4      80.3
## 4   169      66.1      81.9
## 5   176      70.4      79.2
## 6   169      67.6      88.9
```

Goal



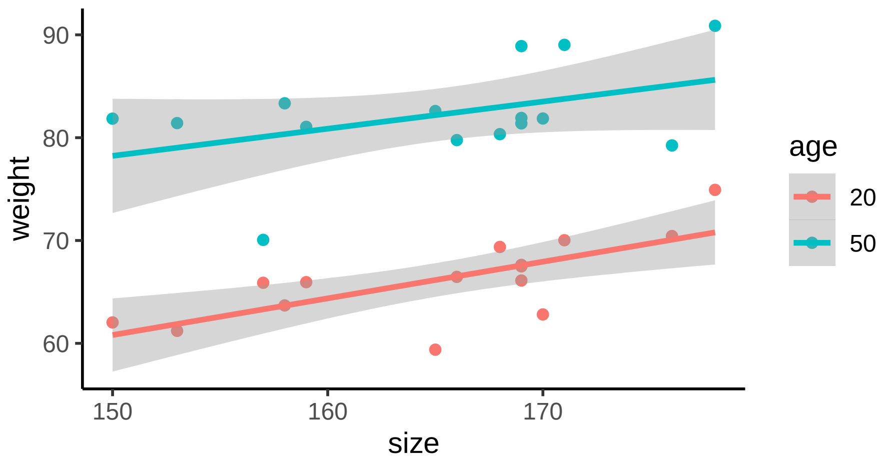
dplyr step

— Gather column weight.M and weight.W into one column *weight*:

```
> df1 <- df %>% gather(key=age,value=weight,-size)
> df1 %>% head()
## # A tibble: 6 x 3
##     size age         weight
##    <dbl> <chr>        <dbl>
## 1   153 weight.20    61.2
## 2   169 weight.20    67.5
## 3   168 weight.20    69.4
## 4   169 weight.20    66.1
## 5   176 weight.20    70.4
## 6   169 weight.20    67.6
> df1 <- df1 %>% mutate(age=recode(age,
+    "weight.20"="20","weight.50"="50"))
```

ggplot step

```
> ggplot(df1)+aes(x=size,y=weight,color=age)+
+   geom_point()+geom_smooth(method="lm")+theme_classic()
```



Complement: some demos

```
> demo(image)
> example(contour)
> demo(persp)
> library("lattice");demo(lattice)
> example(wireframe)
> library("rgl");demo(rgl)
> example(persp3d)
> demo(plotmath);demo(Hershey)
```

$\Longrightarrow$ Work on *tuto 4*.

MAPPING WITH LEAFLET

Introduction

— In many applications, it could be interesting to make *mapping* to *visualize* a dataset or the result of a model.

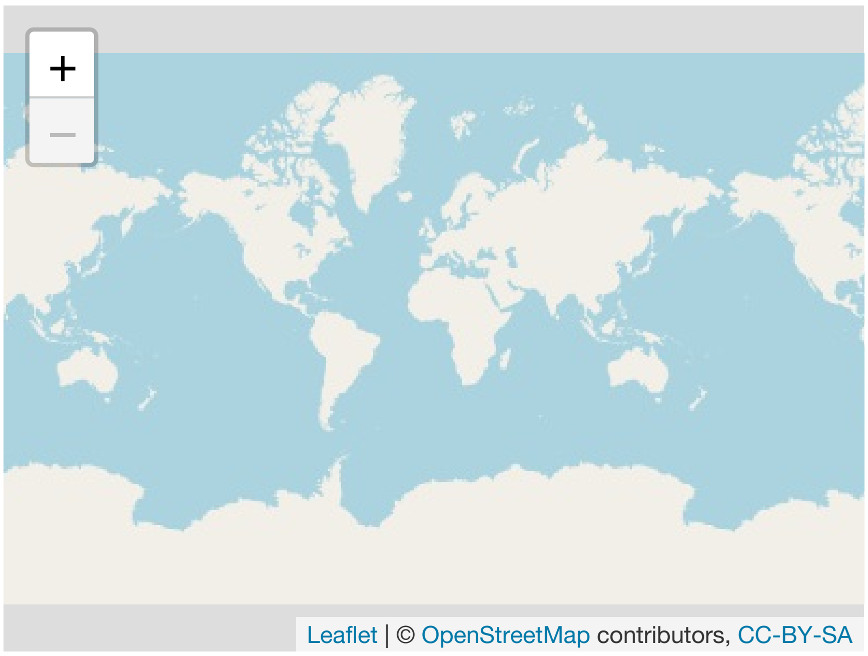
— A *lot of R packages*: ggmap, RgoogleMaps, maps. . .

— In this part: *leaflet*.

Background map

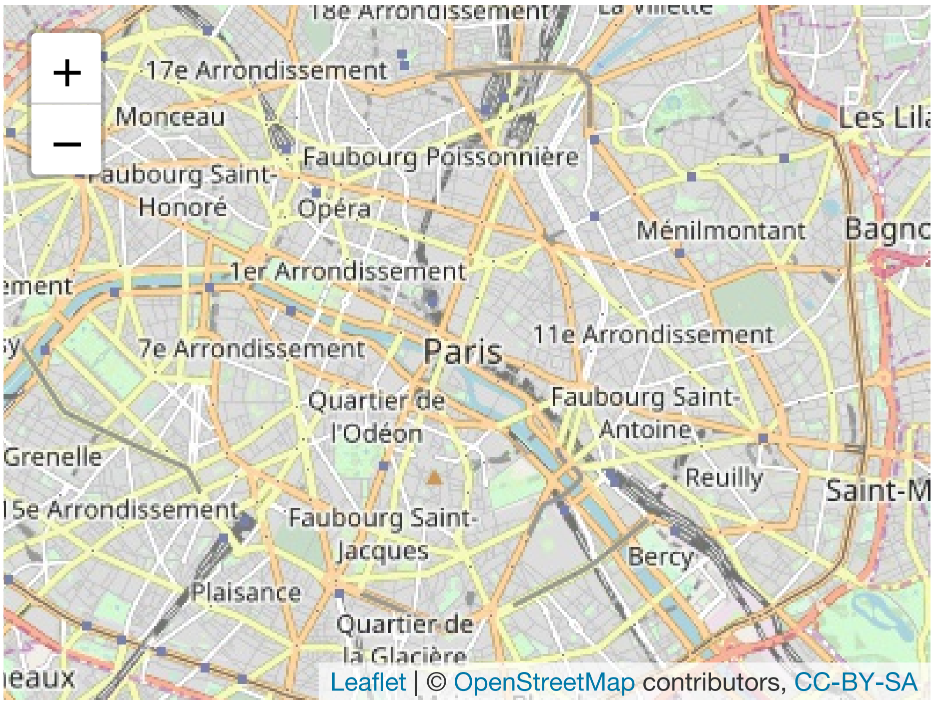
— *Leaflet* is one of the most popular open-source JavaScript libraries for interactive maps.
— *Documentation*: here

```
> library(leaflet)
> leaflet() %>% addTiles()
```

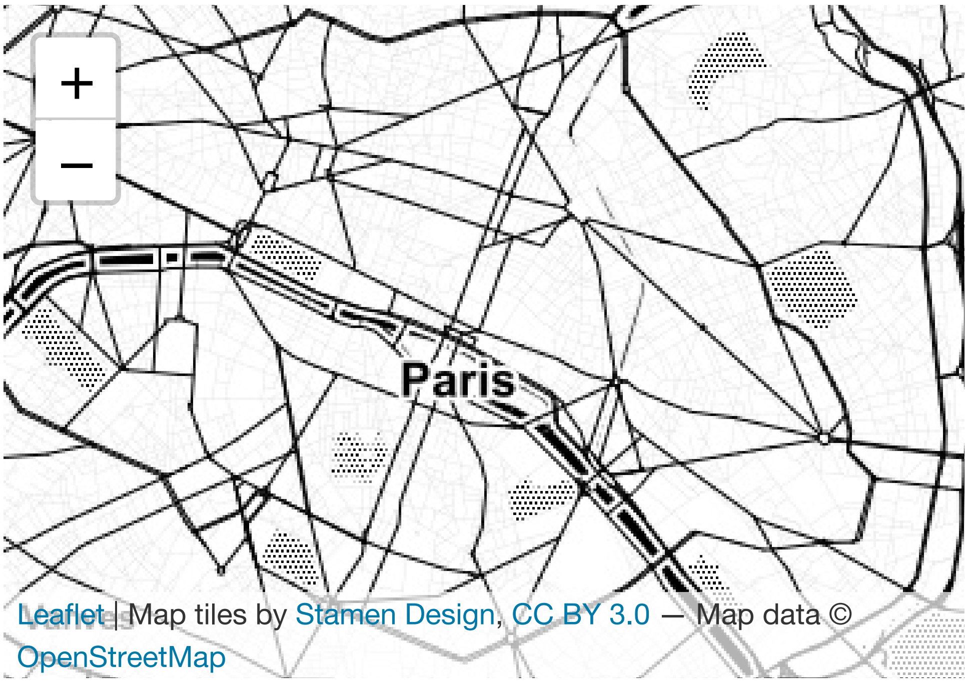


Many background style

```
> Paris <- c(2.35222,48.856614)
> leaflet() %>% addTiles() %>%
+   setView(lng = Paris[1], lat = Paris[2],zoom=12)
```



```
> leaflet() %>% addProviderTiles("Stamen.Toner") %>%
+   setView(lng = Paris[1], lat = Paris[2], zoom = 12)
```


Paris
Leaflet | Map tiles by Stamen Design, CC BY 3.0 — Map data © OpenStreetMap

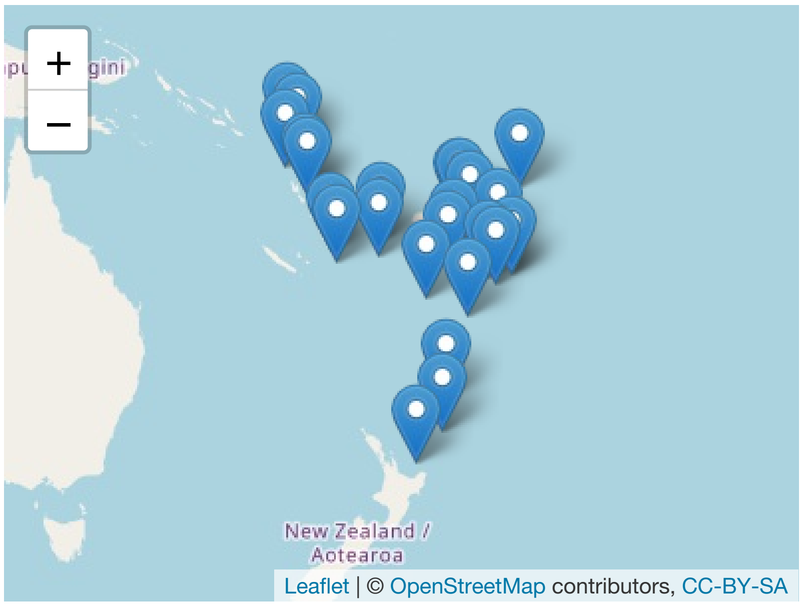
Leaflet with dataset

— Location of 1000 seismics event near Fiji

```
> data(quakes)
> head(quakes)
##      lat   long depth mag stations
## 1 -20.42 181.62   562 4.8       41
## 2 -20.62 181.03   650 4.2       15
## 3 -26.00 184.10    42 5.4       43
## 4 -17.97 181.66   626 4.1       19
## 5 -20.42 181.96   649 4.0       11
## 6 -19.68 184.31   195 4.0       12
```

Visualize seismics with magnitude more then 5.5

```
> quakes1 <- quakes %>% filter(mag>5.5)
> leaflet(data = quakes1) %>% addTiles() %>%
+   addMarkers(~long, ~lat, popup = ~as.character(mag))
```

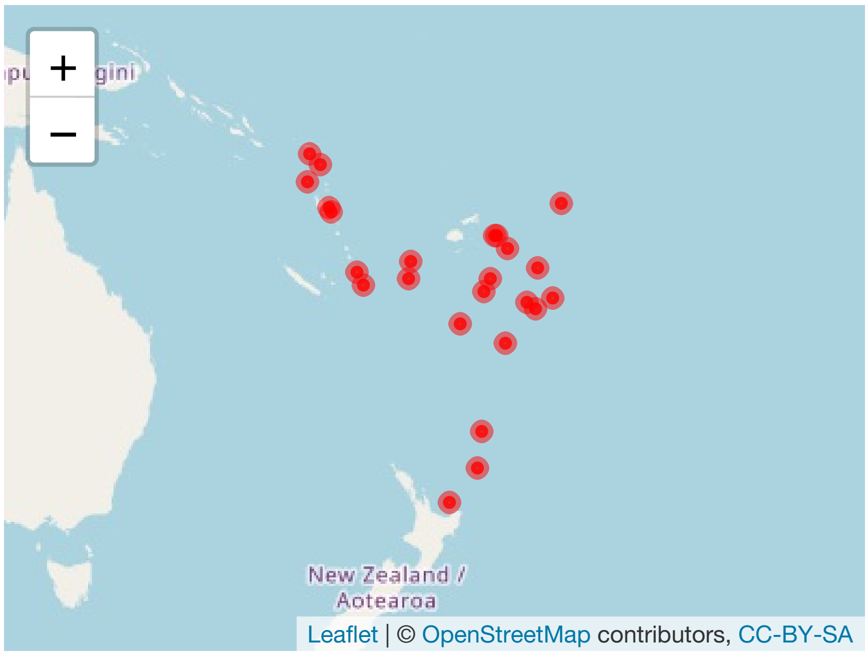


Remark

When you click on a marker, you visualize the magnitude.

addCircleMarkers

```
> leaflet(data = quakes1) %>% addTiles() %>%
+   addCircleMarkers(~long, ~lat, popup=~as.character(mag),
+                    radius=3,fillOpacity = 0.8,color="red")
```



$\Longrightarrow$ work on *tuto 5*.

REGRESSION MODELS WITH R

— *Goal*: present classical functions to make regression with *R*.

— *Transition* with the Machine Learning lecture.

— Focus on *R tools*, mathematical tools will be (or have been) presented in other lectures (statistical model, data mining, machine learning).

Data

$$\begin{array}{ccccc} Y & X_1 & X_2 & \dots & X_p \\ \hline y_1 & x_{1,1} & x_{1,2} & \dots & x_{1,p} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ y_n & x_{n,1} & x_{n,2} & \dots & x_{n,p} \end{array}$$

Goal

Explain or predict output Y by inputs $X_1, \dots, X_p$.

Example: ozone

```
> ozone <- read.table("../DATA/ozone.txt")
> head(ozone %>% select(1:5))
##          maxO3   T9  T12  T15 Ne9
## 20010601    87 15.6 18.5 18.4   4
## 20010602    82 17.0 18.4 17.7   5
## 20010603    92 15.3 17.6 19.5   2
## 20010604   114 16.2 19.7 22.5   1
## 20010605    94 17.4 20.5 20.4   8
## 20010606    80 17.7 19.8 18.3   6
```

Goal

Explain or predict the *daily maximum one-hour-average ozone* (maxO3 column) by the other variables.

Statistical model

- There exists an *unknown* function $m:\mathbb{R}^p\to\mathbb{R}$ such that

$$Y=m(X_1,\dots,X_p)+\varepsilon.$$

- ε: error terms (as small as possible).
- *Statistician's job*: find a good estimate $\widehat{m}$ of m from the data $(x_1,y_1),\dots,(x_n,y_n)$ where $x_i\in\mathbb{R}^p$ and $y_i\in\mathbb{R}$.

Statistical models

Allow to find such estimates.

An example: the linear model

- *Assumption*: the unknwon function is linear

$$Y=\beta_0+\beta_1X_1+\ldots+\beta_pX_p+\varepsilon,$$

$\beta=(\beta_0,\beta_1,\dots,\beta_p)$ are the unknown parameters.

- *Least square estimates*:

$$\widehat{\beta}=(X^tX)^{-1}X^tY.$$

- *Estimate* of m:

$$\widehat{m}(x)=\widehat{\beta}_0+\widehat{\beta}_1x_1+\ldots\widehat{\beta}_px_p.$$

Models with R

- Models on **R** are always fitted in the same way:

```
> method(formula,data=...,options)
```

where

- *method* refers to the name of the model
- *formula* specifies the input Y and the outputs X_j
- *data* is the name of the dataset
- *options* refers to many options depending on the method.

Methods

Remark

Each model corresponds to a R function.

R function	algorithm	Package	Problem
lm	linear model		Reg
glm	logistic model		Class
lda	linear discriminant analysis	MASS	Class
svm	Support Vector Machine	e1071	Class
knn.reg	nearest neighbor	FNN	Reg
knn	nearest neighbor	class	Class
rpart	tree	rpart	Reg and Class
glmnet	ridge and lasso	glmnet	Reg and Class

Formula

Remark

To specify input and outputs.

```
> lm(Y~X1+X3,data=df)
```

$\Longrightarrow Y = \beta_0 + \beta_1 X_1 + \beta_3 X_3 + \varepsilon$

```
> lm(Y~X1+I(X3^2),data=df)
```

$\Longrightarrow Y = \beta_0 + \beta_1 X_1 + \beta_3 X_3^2 + \varepsilon$

```
> lm(Y~.,data=df)
```

$\Longrightarrow Y = \beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p + \varepsilon$

Example

```
> mod.lin <- lm(maxO3~T12+Ne9,data=ozone)
> mod.lin
##
## Call:
## lm(formula = maxO3 ~ T12 + Ne9, data = ozone)
##
## Coefficients:
## (Intercept)          T12          Ne9
##       7.638        4.457       -2.696
```

— Model: $maxO3 = \beta_0 + \beta_1 T12 + \beta_2 Ne9 + \varepsilon$.
— Estimates: $\widehat{\beta}_0 = 7.638, \widehat{\beta}_1 = 4.457, \widehat{\beta}_2 = -2.696$.

Estimate of m

$$\widehat{m}(x) = 7.638 + 4.457\,T12 - 2.696\,Ne9.$$

Making predictions

— Once the model has been fitted, we can use it to make *predictions.*

Example

— Meteofrance predicts for tomorrow: T12=20 and Ne9=4.9.
— What does our model predict for the ozone concentration?
— *Answer*:
$$\widehat{m}(T12 = 20, Ne9 = 4.9) = 7.638 + 4.457 * 20 - 2.696 * 4.9 = 83.5676$$
.

Predict function

— *predict* is a generic function: we can use it to make predictions for all models (linear, logistic, tree. . .)

```
> predict(model.name,newdata=newdataset,...)
```

— *Example*

```
> new.df <- data.frame(T12=20,Ne9=4.9)
> predict(mod.lin,newdata=new.df)
##        1
## 83.57509
```

Very important

Use the same structure for both dataframes.

Estimating the mean square error (ISL lecture)

— The performance of an estimate $\widehat{m}$ can be measured by its *mean square error*:

$$MSE(\widehat{m}) = E[(Y - \widehat{m}(X))^2].$$

— This (*unknown*) error is generally estimated by *validation hold out*:

  — Split the data into a train set and a test set
  — Fit the model on the train set $\Longrightarrow \widehat{m}$
  — Estimate the MSE by

$$\frac{1}{n_{test}} \sum_{i \in test} (y_i - \widehat{m}(x_i))^2.$$

An example

— Data splitting

```
> library(caret)
> set.seed(12345)
> index.train <- createDataPartition(1:nrow(ozone),p=2/3)
> train <- ozone %>% slice(index.train$Resample1)
> test <- ozone %>% slice(-index.train$Resample1)
```

— Model fitting

```
> mod <- lm(maxO3~.,data=train)
```

— Estimated MSE

```
> pred <- predict(mod,newdata=test)
> df <- data.frame(pred=pred,obs=test$maxO3)
> df %>% summarize(MSE=mean((pred-obs)^2))
##        MSE
## 1 387.5472
```

In practice

— Very useful to choose one model.
— *Example*: many models (linear, tree, random forest. . .)

Method

1. Estimate MSE for all algorithms;
2. Choose the algorithm with the smallest MSE.

$\Longrightarrow$ Work on *tuto 6*.

CONCLUSION

Project

- Group of 3 or 4
- Find a dataset for a *supervised learning problem* (explain one variable by other variables). This dataset should contain at least 800 individuals and 30 variables (continuous or categorical)
- There are many datasets on the web, you can look at the following websites for instance:
    - UCI machine learning repository
    - kaggle datasets (you have to register but it's free)
    - other websites of your choice

- You will address the following topics in the study
    - identify the practical problem
    - translate the practical problem into a mathematical problem
    - *describe the dataset* according to the problem (with dplyr)
    - *visualize* the dataset according to the problem (with ggplot)
    - develop machine learning methods (nearest neighbor, linear/logistic models, penalized linear/logistic models, trees, random forest). You should provide a brief description of each algorithm in the context of your problem.
    - make a comparison of the different models (quadratic error, misclassification error, ROC curves, AUC. . .)

- From now on, you can:
    - choose the dataset
    - make the description of the dataset (**dplyr**) and the visualization of the dataset (**ggplot**).

Be careful

- The goal is not to provide a list of statistical summaries or graphs.
- Find *relevant* summaries and you should explain the output (with text!).

- Each group should provide a *notebook* (.rmd file) and send by email (`laurent.rouviere@univ-rennes2.fr`):
    - the notebook (only the .rmd file, not the html file)
    - the dataset (txt or csv file)
- I will run all the chunks of the notebook (the notebook should be complete!), if there is a problem with one chunk, I will not be able to see the output.

Balance sheet

- Many (modern) tools to manipulate data.
- Sufficient to *perform a wide range* of statistical analysis.
- Many lectures where you will use R.
- Try to force yourself to *use these tools* (when you want to make a graph, try to do it in ggplot).

Thank you