R for datascience

Nicolas Sutton-Charani - Euromov DHM

IMT Mines Alès
École Mines-Télécom

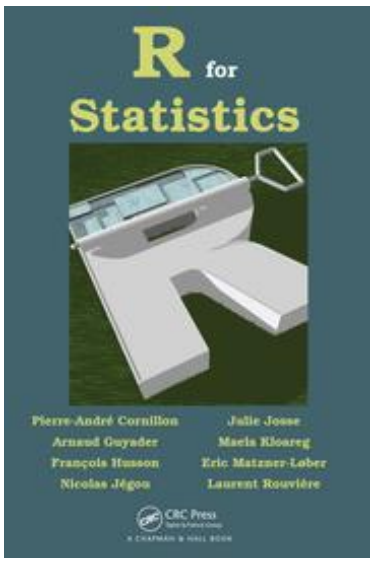
largely inspired by Laurent Rouviere*'s work*

Overview

- Materials : available at https://lrouviere.github.io/R-for-datascience-lecture/
- Prerequisites : Basics on probability, statistics and computer programming
- Objectives : be able to control classical tools for datascience
  - import and concatenate datasets, manipulate individuals and variables
  - implement some of the most important statistical algorithms on real data
- Teacher : Nicolas Sutton-Charani, nicolas.sutton-charani@mines-ales.fr
  - Research interests : machine learning, modern uncertainty theories, health and movement applications
  - Teaching : statistics and probability, machine learning, data science

Resources

- Slides and tutorials (1 tutorial=1 or 2 concepts+exercises) available at https://lrouviere.github.io/intro_R//
- The web
- Book : R for statistics, Chapman & Hall



Outline

1. Introduction
2. Some examples
3. Formats
4. Rstudio, Rmarkdown and R-packages
5. R objects (Review)
6. Reading data from files
7. Data manipulation with Dplyr

Plan

1. Introduction
2. Some examples
3. Formats
4. Rstudio, Rmarkdown and R-packages
5. R objects (Review)
6. Reading data from files
7. Data manipulation with Dplyr

Why R ?

- More and more data available in many fields (energy, health, sport, economy, etc.)
- Data science provides 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 (statisticians) 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.

Plan

1. Introduction
2. Some examples
3. Formats
4. Rstudio, Rmarkdown and R-packages
5. R objects (Review)
6. Reading data from files
7. Data manipulation with Dplyr

Few words about R

```
> 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
> summary(iris)
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width          Species
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100   setosa    :50
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300   versicolor:50
 Median :5.800   Median :3.000   Median :4.350   Median :1.300   virginica :50
 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
```

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

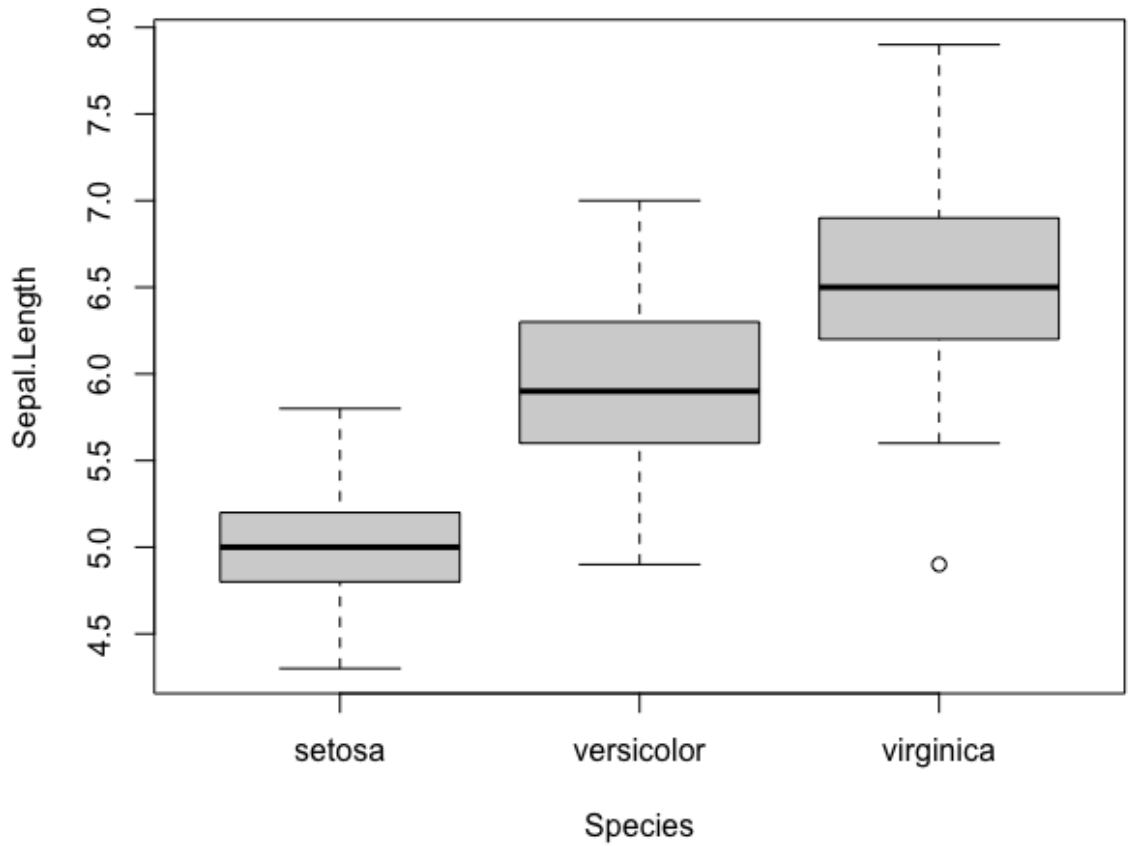
- dplyr is powerful R-package to transform and summarize tabular data with rows and columns.

```
> library(dplyr)
> iris %>% group_by(Species) %>% summarise_all(mean)
# A tibble: 3 × 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.

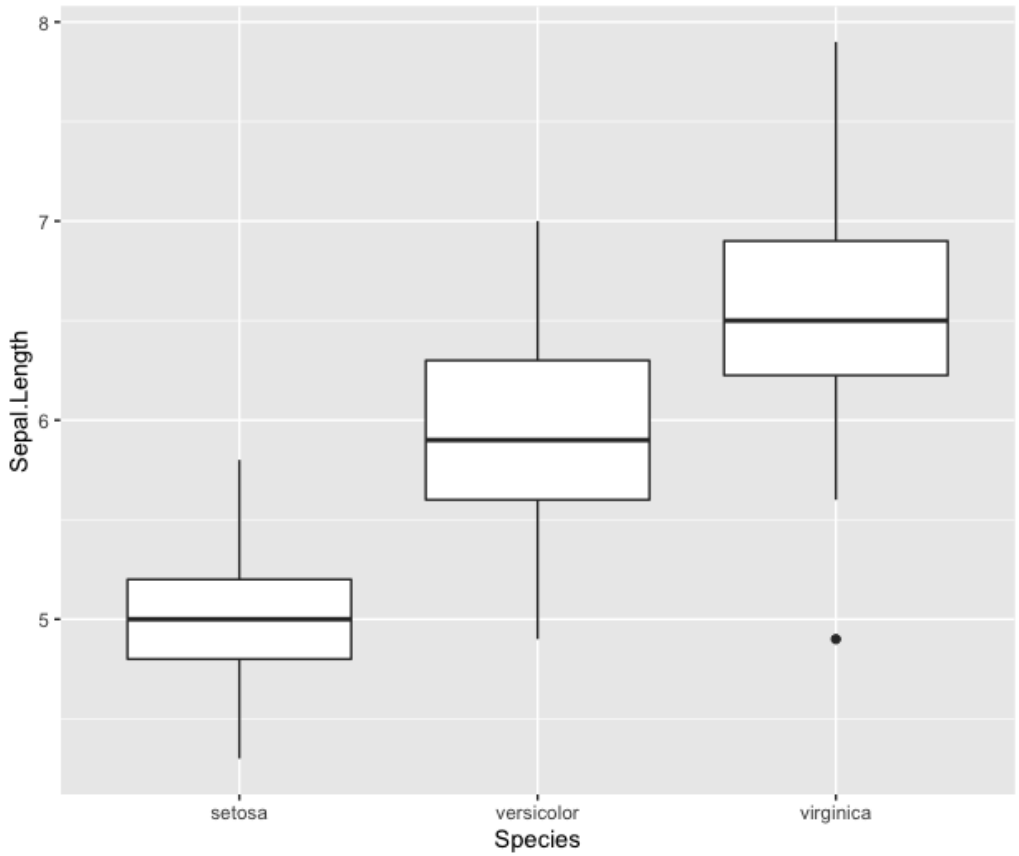
Visualization

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



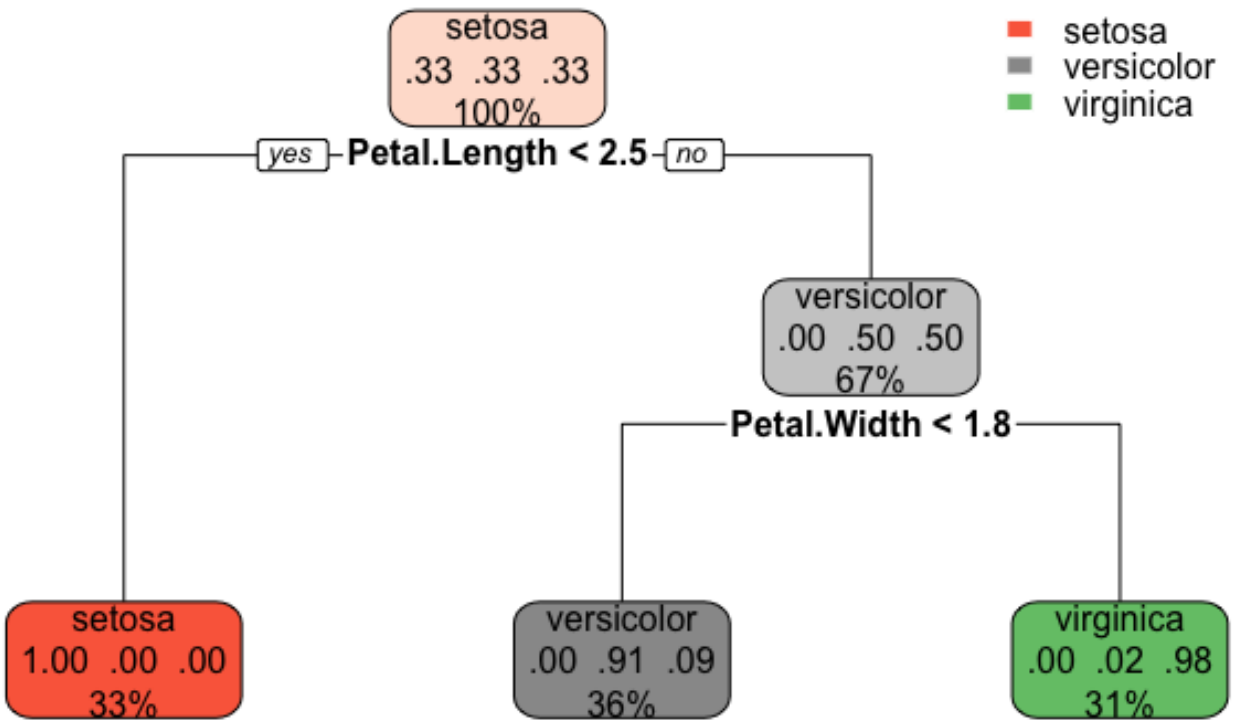
Visualization with ggplot2

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



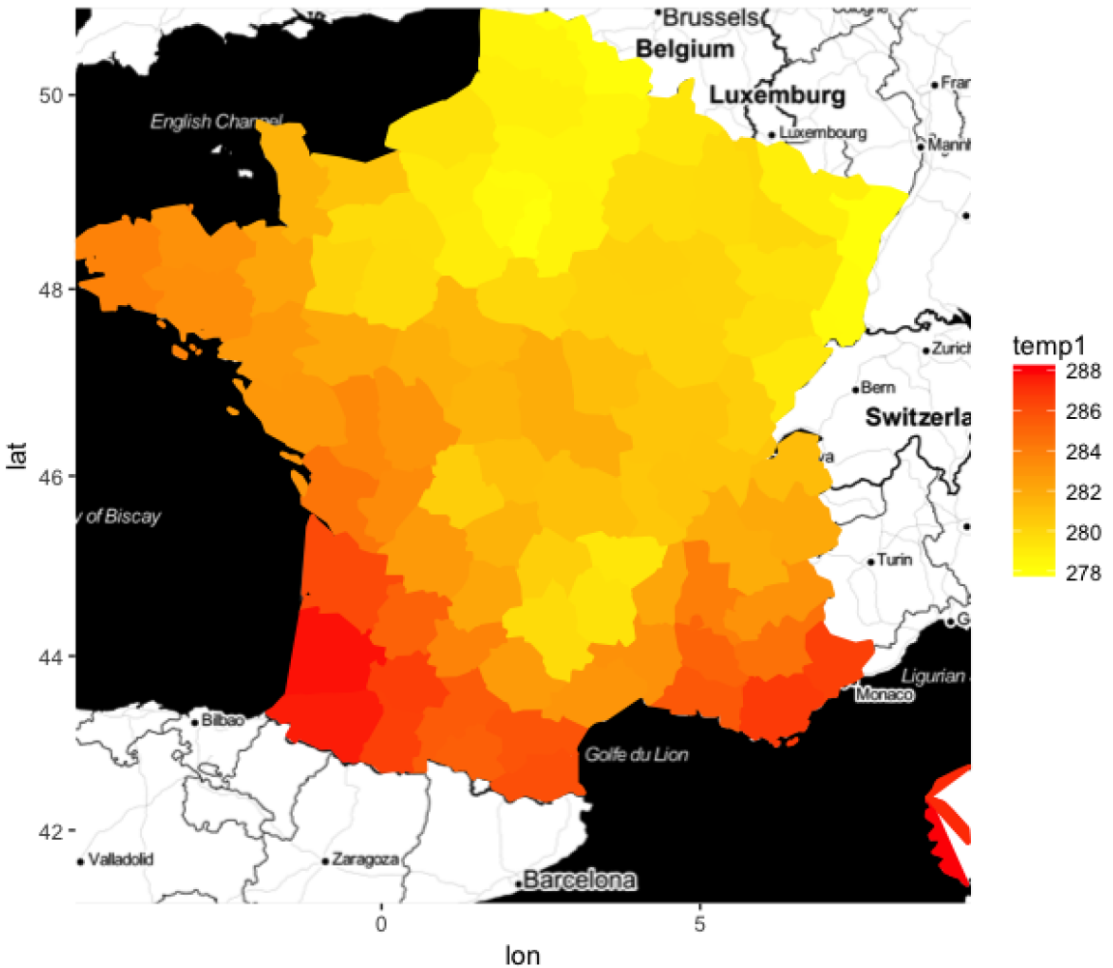
Modelling

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



Maps with ggmap

- Goal : draw a map of the temperatures in france.

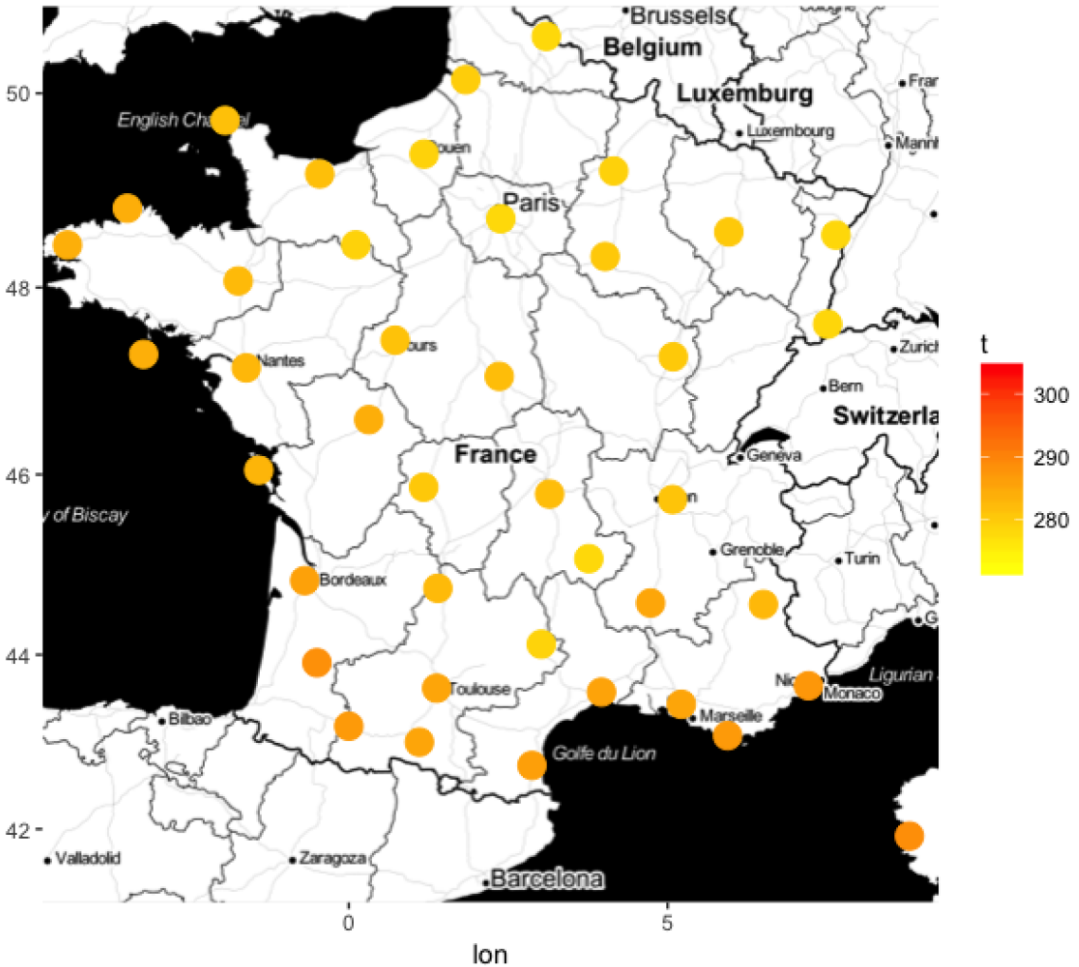


Load the data + background map

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

```
> temp <- fread("https://donneespubliques.meteofrance.fr/
+                   donnees_libres/Txt/Synop/synop.2019082815.csv")
> station <- fread("https://donneespubliques.meteofrance.fr/
+                   donnees_libres/Txt/Synop/postesSynop.csv")
> back.fr <- get_map("France",maptype="toner",zoom=6)
> ggmap(back.fr)+geom_point(data=D,
+         aes(y=Latitude,x=Longitude,color=t),size=5)+
+         scale_color_continuous(low="yellow",high="red")
```

A first map


Brussels
Belgium
Luxemburg
Luxembourg
Paris
Nantes
Bern
Switzerla
France
Geneva
Grenoble
Turin
Bordeaux
Toulouse
Monaco
Marseille
Ligurian
Bilbao
Golfe du Lion
Valladolid
Zaragoza
Barcelona
English Ch
of Biscay
50
48
46
44
42
0
5
lon
t
300
290
280

Model

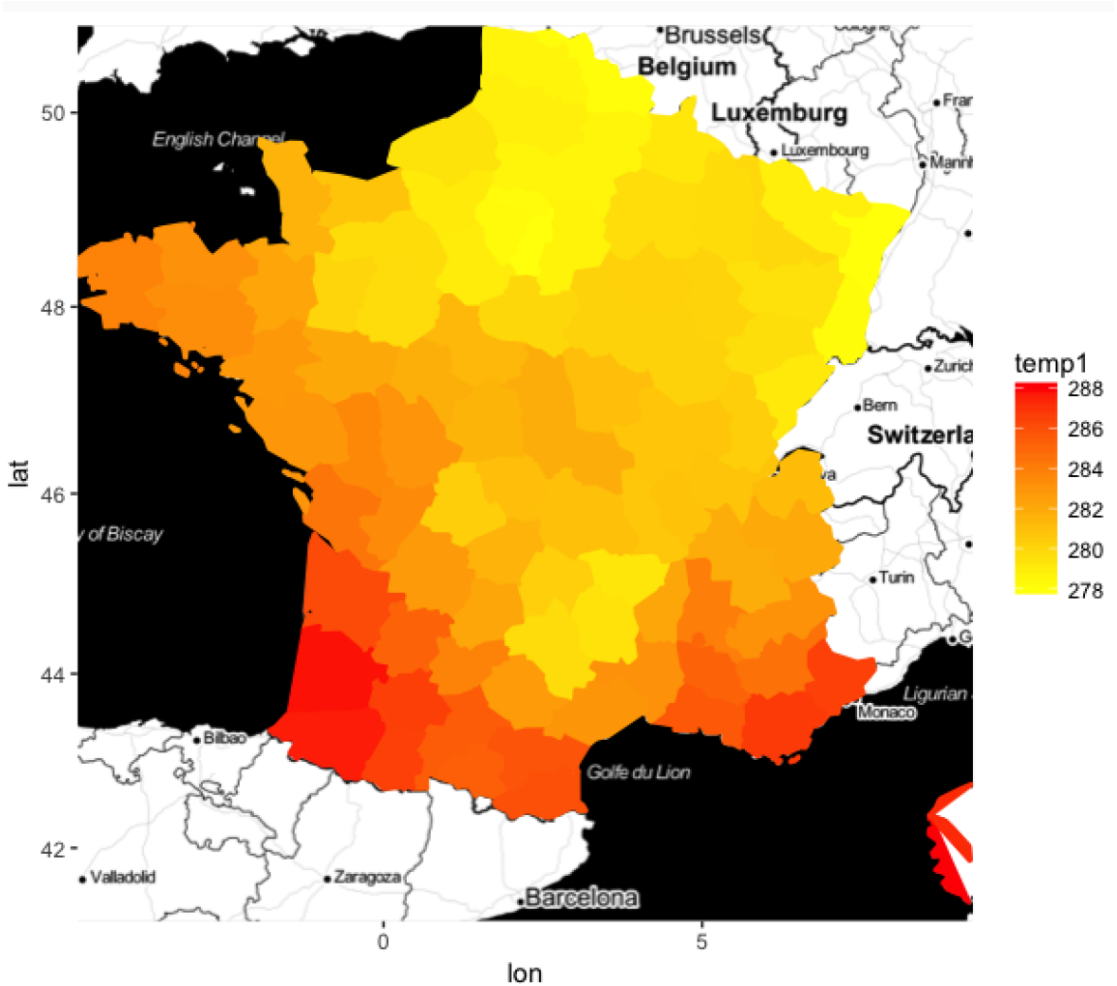
- model of nearest neighbors to estimate temperatures for all longitudes and latitudes.

```
> 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(back.fr)+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')
```

Plan

1. Introduction
2. Some examples
3. Formats
4. Rstudio, Rmarkdown and R-packages
5. R objects (Review)
6. Reading data from files
7. Data manipulation with Dplyr

Formats

- **.R :** script
- **.Rmd :** notebook (R markdown)
- **.RData :** data

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.

Plan

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 objects
- 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

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...).

- 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 R functions.
- Generally reserved for specific methods or fields of applications
- More than 15 000 packages available at https://cran.r-project.org
- 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.
→ work on Tuto 1

Tuto 1

- Download the .Rmd file Tuto1.Rmd in https://lrouviere.github.io/R-for-datascience-lecture/
- 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.

Plan

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)
> x1
[1] 1 3 4
> x2 <- 1:5
> x2
[1] 1 2 3 4 5
> x3 <- seq(0,10,by=2)
> x3
[1]  0  2  4  6  8 10
> x4 <- rep(x1,3)
> x4
[1] 1 3 4 1 3 4 1 3 4
> x5 <- rep(x1,3,each=3)
> x5
 [1] 1 1 1 3 3 3 4 4 4 1 1 1 3 3 3 4 4 4 1 1 1 3 3 3 4 4 4
```

- 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
```

Logical

```
> size <- runif(5,150,190) # 5 sizes randomly generated between 150 and 190
> size
[1] 154.0772 154.7104 189.1931 160.4938 182.3981
```

Problem

Select size smore than 174.

```
> size>174
[1] FALSE FALSE  TRUE FALSE  TRUE
> size[size>174]
[1] 189.1931 182.3981
```

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 → it could generate problems for some statistical studies.
- 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]
[1] 2
> m[1,] #First row
[1] 1 2
> m[,2] #Second column
[1] 2 4
```

List

- Allow to manage objects of different types

```
> 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]]
[1] 1 2 3 4 5
> mylist$vector
[1] 1 2 3 4 5
> mylist[["vector"]]
[1] 1 2 3 4 5
```

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
```

Dataframe

```
> summary(data)
     name                sex              size
 Length:5           Min.   :0.0   Min.   :165.0
 Class :character   1st Qu.:0.0   1st Qu.:168.0
 Mode  :character   Median :0.0   Median :170.0
                    Mean   :0.4   Mean   :171.6
                    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.

Dataframe

```
> data$sex <- as.factor(data$sex)
> levels(data$sex) <- c("man","woman")
> summary(data)
     name                sex          size
 Length:5           man  :3   Min.   :165.0
 Class :character   woman:2   1st Qu.:168.0
 Mode  :character             Median :170.0
                              Mean   :171.6
                              3rd Qu.:175.0
                              Max.   :180.0
```

Problem

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

Dataframe

```
> 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
 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
```

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.

→ work on tuto 2

Plan

- 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 specify 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.
- ...

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)
- RODBC : SQL database

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
```

→ 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
```

→ we keep all the individuals (NAs 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
```

→ 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
```

→ 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.

→ work on tuto 3 - Part 1

Plan

1. Introduction
2. Some examples
3. Formats
4. Rstudio, Rmarkdown and R-packages
5. R objects (Review)
6. Reading data from files
7. 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 <- 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
```

Arrange

Goal

To order individuals.

```
> 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(df)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species diff_petal
1          5.1         3.5          1.4         0.2  setosa        1.2
2          4.9         3.0          1.4         0.2  setosa        1.2
3          4.7         3.2          1.3         0.2  setosa        1.1
4          4.6         3.1          1.5         0.2  setosa        1.3
5          5.0         3.6          1.4         0.2  setosa        1.2
6          5.4         3.9          1.7         0.4  setosa        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 × 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)
```

More generally

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

```
> X <- c(1:10, NA, 20:25)
> X
 [1]  1  2  3  4  5  6  7  8  9 10 NA 20 21 22 23 24 25
> mean(X, na.rm = TRUE)
[1] 11.875
```

or equivalently

```
> X %>% mean(na.rm = TRUE)
[1] 11.875
```

Reshaping data

- Some statistical analysis require 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 × 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 × 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))
> df
# A tibble: 3 × 2
  date        temp
  <date>     <dbl>
1 2015-01-03    18
2 2017-05-18    21
3 2018-09-14    15
```

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

→ work on tuto 3 - Part 2