

Introduction to R software

Céline Keime
keime@igbmc.fr

Introduction to R software

- R software fundamentals
- Graphics using R
- Basic statistics using R

Introduction to R software

- R software fundamentals
- Graphics using R
- Basic statistics using R

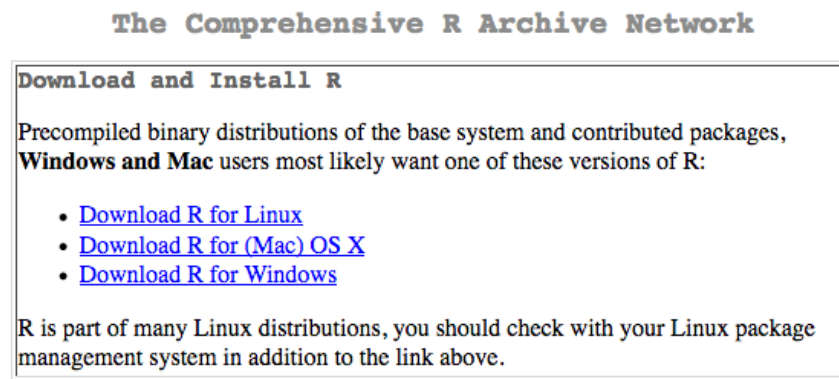
What is ?

- Created by Ross Ihaka and Robert Gentleman*
 - <https://www.r-project.org/>
- R is
 - An environment allowing to perform
 - statistical analyses
 - graphics
 - A programming language
 - Inspired by S
- Freely distributed under GNU General Public License

* Ihaka R. & Gentleman R. 1996. R: a language for data analysis and graphics. *Journal of Computational and Graphical Statistics*;5:299–314.

Where to download R ?

- The comprehensive R archive network (CRAN)
 - <http://cran.r-project.org>
 - Available for Linux, Mac and Windows



- RStudio
 - <https://www.rstudio.com/>
 - Integrated development environment for R
 - Available for Linux, Mac and Windows
(<https://www.rstudio.com/products/rstudio/download/>)



RStudio

The screenshot displays the RStudio interface with three main panes:

- R script:** Contains R code for reading data, plotting, correlation analysis, and prediction.
- Environment:** Shows the loaded data frame 'myo' with 8 observations and 2 variables: 'dose' and 'rep'.
- Plots / Help:** Displays a scatter plot of 'Level of noradrenalin' vs 'Ouabain dose' with a regression line.

R Console: Shows the execution output of the R script, including the Pearson's product-moment correlation coefficient and the results of a correlation test.

```
1 # read data
2 myo = read.table("myocarde.txt",h=T, sep="\t")
3
4 # graphical representation
5 plot(myo$dose,myo$rep,
6      xlab="Ouabain dose",
7      ylab="Level of noradrenalin")
8
9 # correlation
10 cor(myo$dose, myo$rep)
11
12 # correlation test
13 cor.test(myo$dose, myo$rep)
14
15 # Linear regression
16 lm1 = lm(rep ~ dose, data=myo)
17
18 # Diagnostic graphs
19 par(mfrow=c(2,2))
20 plot(lm1, which=c(1,2,4))
21
22 # graphical representation
23 plot(myo$dose,myo$rep,
24      xlab="Ouabain dose",
25      ylab="Level of noradrenalin")
26 abline(lm1)
27
28 # prediction
29 predict(lm1,data.frame(dose=0.75))
```

Environment:

Variable	Type	Value
lm1	List of 12	
myo	8 obs. of 2 variables	
dose	num	0 0.25 0.5 1 1.25 1.5 1.75 2
rep	num	0.66 0.63 0.51 0.48 0.3 0.26 0.26 0.14

Plots / Help:

The plot shows a negative correlation between Ouabain dose and Level of noradrenalin. The x-axis is labeled 'Ouabain dose' and ranges from 0.0 to 2.0. The y-axis is labeled 'Level of noradrenalin' and ranges from 0.2 to 0.6. A regression line is fitted to the data points.

R Console:

```
> # read data
> myo = read.table("myocarde.txt",h=T, sep="\t")
>
> # graphical representation
> plot(myo$dose,myo$rep,
+      xlab="Ouabain dose",
+      ylab="Level of noradrenalin")
>
> # correlation
> cor(myo$dose, myo$rep)
[1] -0.9782861
>
> # correlation test
> cor.test(myo$dose, myo$rep)

Pearson's product-moment correlation

data: myo$dose and myo$rep
t = -11.562, df = 6, p-value = 2.518e-05
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
-0.9962041 -0.8808381
```

Start an R session

- Launch R or RStudio
 - Choose a working directory
 - Why ?
 - Store data files : by default R will search files in this directory
 - Save results : graphs, files, R working sessions
 - Continue a previous analysis
 - How ?
 - `> setwd("path/to/directory")`
 - Rstudio : Session / Set Working Directory / Choose Directory
 - Drag and drop your folder on RStudio icon
 - Verification
 - `> getwd()` : get current working directory
 - RStudio : Console title
- ➔ Create a working directory for this training and choose to work in this directory
- ➔ Verify that you are working in the correct directory

Interactive usage of R (1/2)

- Principle

- Enter a command and press *return*
- R execute the command (and eventually display the corresponding result)
- Another command can then been entered

- Examples

```
> 1+1
```

```
[1] 2
```

```
> x=1
```

```
> x
```

```
[1] 1
```

Comment (not taken into account by R)

affectation

- The arrow  allow to recall a previous command

Interactive usage of R (2/2)

- Examples

```
> rnorm(10) # rnorm() : function, c.f. brackets
```

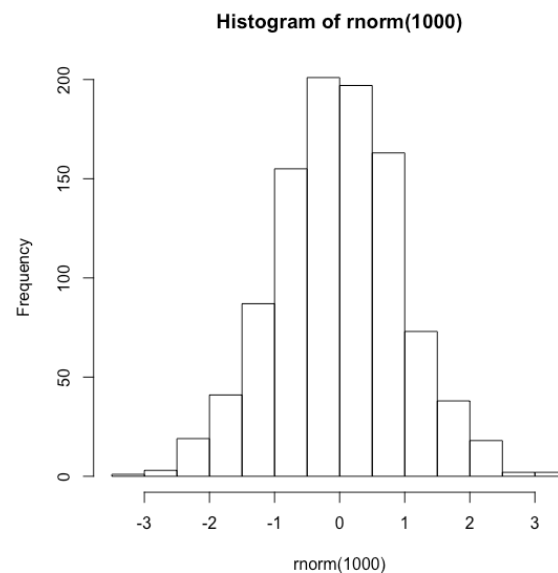
```
[1] -1.087784299 1.157426928 0.364218560
```

```
[4] 2.065149368 -0.893725503 -0.251256812
```


```
[7] 0.005916529 0.072525550 -0.446474935
```

```
[10] -0.399408315
```

```
> hist(rnorm(1000)) #rnorm() : graphical function
```

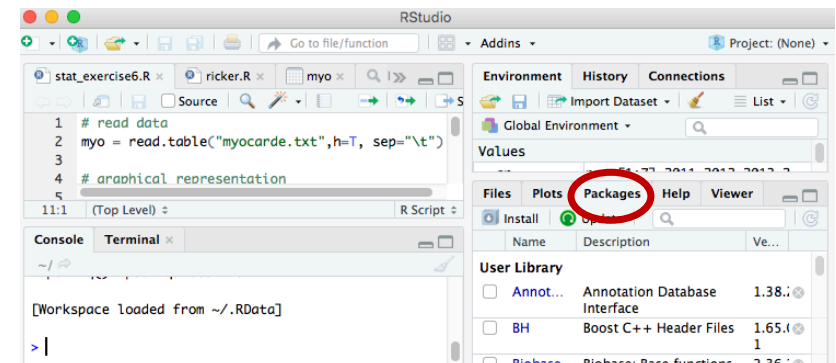


Backups

- Backups during an interactive session
 - > `save.image()` # save R objects created during a session
default file that will be created : `.RData`
 - > `savehistory()` # save the history of commands
default file that will be created : `.Rhistory`
 - Prefer to save commands in an R script (text file with R commands AND comments)
- Save when exiting R
 - > `q()` # to exit R (or click on )
Save workspace image? [y/n/c]: `y`
- Continue an analysis started with R
 - > `load("path/to/.RData")`
 - or open R in the directory chosen for this analysis
(e.g. drag and drop this directory to RStudio icon)

R libraries (1/2)

- A lot of functions available by default in R (*base* library)
- More specific functions (multivariate analyses, clustering, non linear regression, phylogeny, ...) → libraries
 - A library is a set of tools on a specific subject
 - RStudio : packages tab
 - Several libraries are installed within R :
 - > `installed.packages()` # list of all installed packages and their version
 - > `library()` #list of installed libraries and brief description
 - To install a new library
 - > `install.packages("library_name")`
 - Or click on install on RStudio *Packages* tab
 - To load a library in the working session
 - > `library(library_name)`
 - Required to be able to use a function from the package *library_name*, in each R session



R libraries (2/2)

- CRAN libraries :
 - [https://cran.r-project.org/ Packages](https://cran.r-project.org/Packages) link
 - 12,308 libraries currently available
- Bioconductor project :
 - <http://www.bioconductor.org/>
 - Provides tools for the analysis of high-throughput genomic data
 - Last version : 3.6
 - 1,473 packages to analyse and represent genomic data
 - 911 annotations packages
 - 326 packages containing experiment data
 - Installation
 - `source("http://bioconductor.org/biocLite.R")`
 - `biocLite() #install core packages`
 - `biocLite("library_name") #install a specific package called library_name`



CRAN
[Mirrors](#)
[What's new?](#)
[Task Views](#)
[Search](#)

About R
[R Homepage](#)
[The R Journal](#)

Software
[R Sources](#)
[R Binaries](#)
[Packages](#)
[Other](#)

The Comprehensive R Archive Network

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- [Download R for Linux](#)
- [Download R for \(Mac\) OS X](#)
- [Download R for Windows](#)

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

Source Code for all Platforms

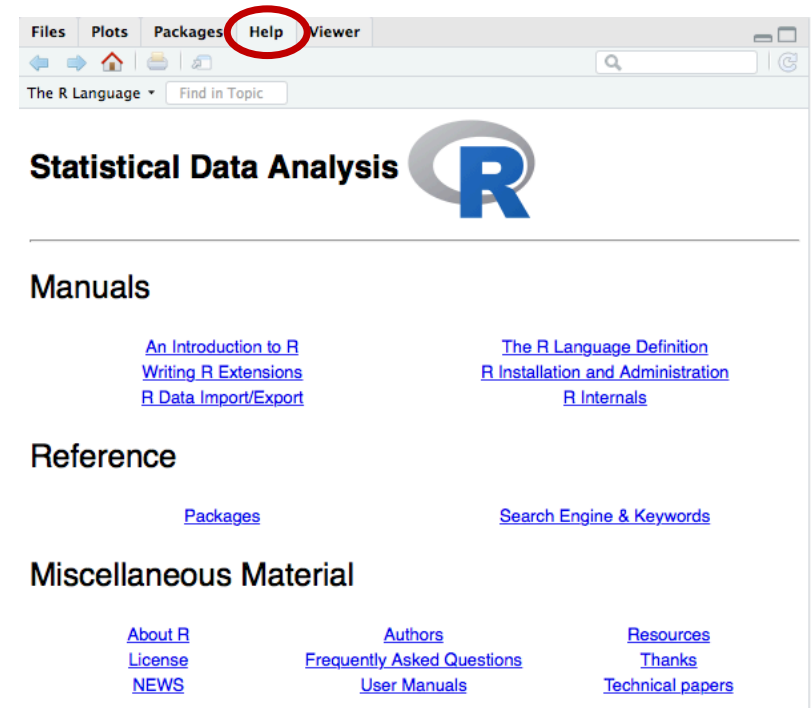
Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

- The latest release (2018-03-15, Someone to Lean On) [R-3.4.4.tar.gz](#).



Sources of documentation on R (1/2)

- R homepage : <http://www.r-project.org>
 - Manuals (to start : *An introduction to R*) + *contributed documentation* link
 - FAQ
- Rseek : <http://www.rseek.org>
- R Site Search : <http://finzi.psych.upenn.edu/search.html>
- Bioconductor : <http://bioconductor.org/help/>
 - Very helpful : package vignettes and workflows
- Within R :
 - `help.start()` # html help / Help tab on RStudio
 - `?y` # help on y
 - Very useful when you know the name of the object/function of interest
 - Example : `?plot`
 - `??subject`
 - # help pages containing the word “subject”
 - # Search Engine & Keywords in RStudio
 - Helpful when you don’t know the name of the corresponding object/function
 - Example : `??Student`



The screenshot shows the RStudio Help window. The 'Help' tab is selected in the menu bar. The window displays the R documentation website with the following sections:

- Manuals**
 - [An Introduction to R](#)
 - [Writing R Extensions](#)
 - [R Data Import/Export](#)
 - [The R Language Definition](#)
 - [R Installation and Administration](#)
 - [R Internals](#)
- Reference**
 - [Packages](#)
 - [Search Engine & Keywords](#)
- Miscellaneous Material**
 - [About R](#)
 - [License](#)
 - [NEWS](#)
 - [Authors](#)
 - [Frequently Asked Questions](#)
 - [User Manuals](#)
 - [Resources](#)
 - [Thanks](#)
 - [Technical papers](#)

Sources of documentation on R (2/2)

- Structure of an R help page (c.f. for example `?plot`)
 - Top
 - Name of the help page and corresponding library
 - Description
 - Brief description of the utility of the function/operator
 - Usage
 - Syntax to use in order to call a function (with default parameters values)
 - Typical use of an operator
 - Arguments
 - For each argument of the function : detailed description and type
 - Details
 - Precisions on the function/operator (possibly references)
 - Value
 - If applicable, the type of the object returned by the function
 - See also
 - Other help pages related to this page
 - Examples
 - Examples of use of the function
 - Can be executed with the `example()` function : c.f. `> example(plot)`

Objects

- Variables, data, results are stored in memory in **objects**
- We act on these objects using **operators** and **functions**
- Object name
 - Must start with a letter and may contain : a-z / A-Z / 0-9 / . / _
 - Case sensitive (a ≠ A)
- An object can be created with the affectation operator : = or <-
> a = 1
- To view the content of an object : type its name
> a
[1] 1
- List all objects in memory : ls()
> ls()
[1] "a"
- Remove an object from memory : rm()
> rm(a) # remove from memory the object a
> ls()
character(0)

Operators

- Main arithmetic operators

+ - * / ^(power)

- Relational operators

< > <= >= == (equality) != (difference)

- Logical operators

- No : !
- And : &
- Or : |

Functions

- Call a function :
 `function_name()`
- Function arguments
 - Into brackets
 - Argument not given → default value if defined, c.f. `?function`
 - Arguments identification
 - By name. Can be abbreviated if the abbreviation is not ambiguous
 - By position

Example :

```
> ?seq
> seq(from=1, to=10, by=1) # identification by name ← to prefer
[1] 1 2 3 4 5 6 7 8 9 10
> seq(fr=1, to=10, by=1) # identification by abbreviated name
[1] 1 2 3 4 5 6 7 8 9 10
> seq(1, 10, 1) # identification by position
[1] 1 2 3 4 5 6 7 8 9 10
> seq(to=10, by=1) # from=1 by default
[1] 1 2 3 4 5 6 7 8 9 10
```

Some useful functions in R

- To obtain information about an object
 - `class` : class of the object
 - `length` resp. `dim` : dimension of an object with 1 resp. 2 dimensions
 - `head` : the beginning of an object (useful for the visualisation of large objects)
 - `summary` : summary of the content of an object
- Mathematical functions
 - `log`, `log10`, `log2`, `exp`
 - `min`, `max` : minimal and maximal values of an object / `range` : `c(min,max)`
 - `sum`, `diff`, `prod` : sum, difference and product of the elements of a vector
 - `round` : round of a value
- Objects manipulation
 - `sort` : sort a vector or factor
 - `order` : give the permutation which allows to sort a vector

```
> obs = c(3,9,7,4,1)
> obs
[1] 3 9 7 4 1
> sort(obs)
[1] 1 3 4 7 9
> order(obs)
[1] 5 1 4 3 2
```

Some useful functions in R : examples

iris is a dataset available in R

It contains the measurements (in cm) of sepal length and width and petal length and width, for 50 flowers from 3 species of iris (*Iris setosa*, *versicolor* and *virginica*).

```
> data(iris) # load the iris dataset
```

```
> class(iris) #class of the iris object
```

```
[1] "data.frame"
```

```
> dim(iris) # dimensions of the dataset (number of lines and columns)
```

```
[1] 150 5
```

```
> head(iris) # visualize the beginning of this dataset
```

	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



from https://fr.wikipedia.org/wiki/Iris_de_Fisher

Exercise 1

Use the *iris dataset* available in R :

- load iris dataset with `> load(iris)`
- How many flowers from *setosa* species are available in this dataset ?
- Print sepal length of all flowers in increasing order
- What is the mean sepal length of all flowers ?
- How many flowers have a sepal length larger than 5 cm ?
- How many *setosa* flowers have a sepal length larger than 5 cm ?

Note :

- `> iris$Sepal.Length` # column Sepal.Length from the *iris* dataset
- `> sum(Y)` where Y is a logical vector gives the number of TRUE elements in this vector

Class of an object

- Different object classes depending on what and how we want to store values
 - Vectors : numeric, character, logical
 - Factors : factor
 - Arrays : data.frame or matrix
 - Lists : list
 - Functions : function
- To know the class of an object use the `class()` function

Vectors

- A vector contains one or several values of the same type
 - numeric : numbers
 - character : character strings
 - logical : Boolean values (TRUE or FALSE)
 - ➔ Even the smallest objects (e.g. only 1 element) are vectors in R
- Creation of a vector
 - With several numbers, characters or boolean values : `c()`
> `obs = c(3,9,7,4,1)`
> `obs`
[1] 3 9 7 4 1
 - Sequence of integers with an increment of 1 or -1 : :
> `v = 1:5`
> `v`
[1] 1 2 3 4 5

Numeric vectors content

- Integers
- Real numbers
 - With a decimal point : 0.2
 - Scientific notation : 2e-1 # corresponds to 2×10^{-1}
- Infinite : Inf et -Inf
 - > 1/0 > -1/0
 - [1] Inf [1] -Inf
- Not a number : NaN
 - > 0/0
 - [1] NaN
 - > log(-1)
 - [1] NaN
- Missing values : NA

Logical vectors

- A logical vector may contain 3 different values
 - TRUE or T
 - FALSE or F
 - NA (missing value)
- Often created from an operation
 - > `obs = c(3,9,7,4,1)`
 - > `obs > 3` #observations strictly higher than 3
 - [1] FALSE TRUE TRUE TRUE FALSE

Character strings vectors

- A character strings vector may contain
 - One or more strings delimited by " "
- Several useful functions to manipulate character strings
 - `paste`
 - To concatenate several strings
 - `grep`
 - Search for an expression in character strings
 - `substr`
 - Extract or replace a substring in a vector of character strings
 - `match` (or `%in%`)
 - `x %in% y` return TRUE for each element of x present in y
 - `unique(x)`
 - Return one time each distinct element from x

Indexation of a vector

- Operator [] allows to access elements from a vector

```
> v[2]
```

```
[1] 2
```

```
> v[6] # Return NA if we exceed the length of the object
```

```
[1] NA
```

```
> v = 1:5
```

```
> v
```

```
[1] 1 2 3 4 5
```

- Several possible indexations

- With a vector of **positive integers** : indices of the elements to keep

```
> v[2:4]
```

```
[1] 2 3 4
```

- With a vector of **negative integers** : indices of the elements to exclude

```
> v[-c(1,5)]
```

```
[1] 2 3 4
```

➔ It is impossible to specify both positive and negative indexes

- With a **logical vector** : TRUE correspond to the values to keep, FALSE to the values to exclude

```
> v[ c(FALSE, TRUE, TRUE, TRUE, FALSE) ]
```

```
[1] 2 3 4
```

Very useful to keep only the elements from a vector satisfying a given criterion :

```
> iris$Sepal.Length[ iris$Species=="setosa" ] #Sepal Length from iris setosa
```

Manipulation of vectors

- To know the number of elements in a vector : `length()`

```
> length(v)
[1] 5
```

- Operators and usual mathematical functions operate element by element

Examples :

```
> v
```

```
[1] 1 2 3 4 5
```

```
> v2 = 5:1
```

```
> v2
```

```
[1] 5 4 3 2 1
```

```
> v + v2
```

```
[1] 6 6 6 6 6
```

```
> v+1
```

```
[1] 2 3 4 5 6
```

```
> 2 * v
```

```
[1] 2 4 6 8 10
```

```
> v==5
```

```
[1] FALSE FALSE FALSE FALSE TRUE
```

Exercise 2

Use the *iris* dataset available in R

```
> sl = iris$Sepal.Length
```

- What is the class of sl object ?
- What is the minimal and maximal sepal length of these flowers ?
- What is the length of the 10 largest sepals ?

Factors

- Factors allow to store qualitative variables in R
- A factor in R is made of :
 - The values of a qualitative variable
 - The levels of this variable, i.e. the different possible values of this variable, eventually ordered
- Use `factor()` function to create factors in R

Creation of factors

- Non-ordered qualitative variables

e.g. the winter behaviour of bird species : migratory (M), partial migratory (PM) or sedentary (S)

```
> wb = factor(c("M","M","S", "PM","S")) # winter behaviour of 5 species
> wb
[1] M M S MP S
Levels: M MP S
> class(wb)
[1] "factor"
```

- Ordered qualitative variables

- specify : ordered = TRUE in the factor creation instruction
- If the argument levels are not specified, the alphabetical order is used

e.g. the diversity of bird species observed in several stations : low, medium or high

```
> diversity = factor(c("low","high","medium","medium","low"), levels = c("low","medium","high"),
ordered = T) # diversity of bird species in 5 stations.
```

the alphabetical order is not appropriate → we need to precise the levels

```
> names(diversity) = paste("s",1:5,sep="") # names of the corresponding stations
```

```
> diversity
s1    s2    s3      s4      s5
low  high medium medium low
Levels: low < medium < high
```

Manipulation of factors

- Levels of a factor : levels()
 > levels(wb)
 [1] "M" "PM" "S"
 > levels(diversity)
 [1] "low" "medium" "high"
- Contingency table (counts for each factor level) : table()
 > table(wb)
 wb
 M PM S
 2 1 2
 > table(diversity)
 diversity
 low medium high
 2 2 1
- Extract element(s) from a factor : []
 - Same principle as for a vector

Exercise 3

Use the *iris* dataset available in R

- What is the class of the “Species” object ?
- What are the different iris species in this dataset ?
- How many flowers from each species are present in this dataset ?
- What is the species of the iris with the smallest sepal length ?

Lists (1/2)

- Very flexible data structure :
 - List = several elements of any type
 - An element from a list can be any object : vector, factor, list, data.frame, matrix, function
- Creation of a list : `list()`
`mylist = list(elt1=value1, elt2=value2, ...)`
 - `elt1, elt2` : name of each element in the list (not required)
 - `value1` : value of the first element (any R object)
- Extract element(s) from a list :
 - `mylist$elt1` only if a name has been set to the element in the list
 - `elt1` : name of the element in the list that we want to extract
 - Use `names(mylist)` to know the names of the elements from a list
 - `mylist[[i]]`
 - To extract the element `i` from the list `mylist`

Lists (2/2)

- R functions often return lists

→ It allows to easily search for information among all the results provided by R

Example :

```
> s1 = rnorm(100)
```

```
> s2 = rnorm(100)
```

```
# Student's t-test
```

```
> ttest = t.test(s1,s2,var.equal=T)
```

```
> ttest
```

```
Two Sample t-test
```

```
data: ech1 and ech2
```

```
t = 0.8984, df = 198, p-value = 0.3701
```

```
alternative hypothesis:
```

```
true difference in means is not equal to 0
```

```
95 percent confidence interval:
```

```
-0.1445087 0.3863550
```

```
sample estimates:
```

```
mean of x mean of y
```

```
0.116775078 -0.004148088
```

```
# t.test function return many information
```

```
# ttest is a list
```

```
> is.list(ttest)
```

```
[1] TRUE
```

```
# names of the elements in the list ttest
```

```
> names(ttest)
```

```
[1] "statistic" "parameter" "p.value"
```

```
[4] "conf.int" "estimate" "null.value"
```

```
[7] "alternative" "method" "data.name"
```

```
# extract the p-value of the test
```

```
> ttest$p.value
```

```
[1] 0.3700663
```

```
# extract the confident interval
```

```
> ttest$conf.int
```

```
[1] -0.1445087 0.3863550
```

```
attr(,"conf.level")
```

```
[1] 0.95
```

Matrices

- Table (2 dimensions) that contain elements of the same type

→ We can create a matrix of mode numeric, character or logical

```
> m = matrix(1:6, nrow=2, ncol=3)
```

```
> m
```

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

```
> dim(m) # number of rows and columns
```

```
[1] 2 3
```

- To extract elements from a matrix : [

– To extract the value in line i and column j : `matrix[i,j]` > `m[2,1]`
[1] 2

– To extract all rows : `matrix[,j]` > `m[,3]`
[1] 5 6

– To extract all columns : `matrix[i,]` > `m[2,]`
[1] 2 4 6

- Mathematical operators and functions work element by element

```
> m-m
```

```
      [,1] [,2] [,3]
[1,]  0   0   0
[2,]  0   0   0
```

data.frame

- data.frame : a table
 - Made of several numeric, logical and/or character vectors
 - All these vectors must have the same length
 - Difference from matrix : vectors could be of different types
- Often the most appropriate class to store datasets

```
> class(iris)
[1] "data.frame"
> dim(iris) # dimensions of iris data.frame
[1] 150  5
# iris data.frame contains vectors of different types :
> class(iris$Sepal.Length)
[1] "numeric"
> class(iris$Species)
[1] "factor"
```
- Usually data are available in a file
 - Use the `read.table` function to create a data.frame by reading data from this file

Manipulation of data.frame

- Indexation

- As for matrices

- `df[i,j]` to extract the value in line i and column j

- Example :

```
> 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

```
> iris[3,2]
```

```
[1] 3.2
```

- You can also use the operator `$` to extract a column

```
> iris$Sepal.Length #same as iris[,1]
```

- Dimensions

- `dim()` function

Data input and output

- File format for data input : tabulated text
 - Data in excel
 - Save as text
 - Choose a field separator that is never used in your data, e.g. tabulation
- Most useful function for data input : `read.table()`
 - Read a text file and create a data.frame with the data from this file
 - Rows and columns from the created data.frame correspond to rows and columns from the file

Data input

- Main options of the `read.table()` function (*c.f. ?read.table*)
 - file
 - Name of the file (between quotes, or a character string object)
 - By default R search for this file in the working directory (otherwise specify the corresponding path)
 - header
 - TRUE or FALSE depending on whether the first row contains header information
 - sep
 - Field separator (e.g. "\t" for tabulations)
 - quote
 - Characters used to delimit the character variables
 - dec
 - Character used for decimal points ("." by default)
 - row.names
 - Vector with the names of rows
 - Or number of the column that contains row names in the file (e.g. 1)

Data input example

- The ecrin.txt file is available on the following webpage :
<http://genomeast.igbmc.fr/wiki/doku.php?id=training:introduction2r>
- This file contains the number of bird species (RIC column) observed in Ecrins national park in 1991 (1315 observations)*
- Each observation was done
 - In a station indicated in the STA column
 - During the week indicated in the SEM column
 - Either during the morning or the afternoon
 - HEU column : 1:morning, 2:afternoon

Data input example


```
# read ecrin.txt file and put the corresponding data.frame in the ecrin object
# in the ecrin.txt file the first line correspond to the column names
> ecrin=read.table("ecrin.txt", header=TRUE)
```

```
> class(ecrin)
[1] "data.frame"
```

```
> head(ecrin)
  STA SEM HEU RIC
1  3   2   1   5
2  3   2   2   3
3  3   3   1   5
4  3   3   2   3
5  3   4   1   4
6  3   4   2   1
```

```
# You can then manipulate the corresponding data using R
# e.g. select only observations performed during the morning
> ecrin.morning = ecrin[ ecrin$HEU == 1, ]
```

Save data into a file (1/2)

- `write.table()` function
 - Write an object (vector, data.frame or matrix) in a file
- Main options of the function `write.table()`
 - 1st argument
 - Name of the object to write
 - file
 - Name of the file in which you want to write (default output to the R console)
 - append
 - If TRUE append to the file
 -  default=FALSE : any existing file with this name is destroyed
 - quote
 - If TRUE (default), character and factor columns, row and column names will be surrounded by “ “
 - sep
 - The field separator string

Save data into a file (2/2)

- Main options of the function `write.table()`
 - `na`
 - The string to use for missing data
 - `dec`
 - The string to use for decimal points
 - `row.names / col.names`
 - whether the row/column names of the data.frame are to be written (TRUE or FALSE)
 - or a character vector of row/column names to be written
- Example

```
# Creation of a file ecrin_morning.txt containing only observations performed during the morning
> write.table(ecrin.morning, file="ecrin_morning.txt", quote=F, sep="\t", row.names=FALSE)

# options :
– Do not use quote for character and factor columns
– Use tabulation as the field separator string
– Do not write line numbers
```

Exercise 4

For this exercise use the dataset available in the humanGenomeSummary.txt file. This file contains for each human chromosome its length in bp, the number of coding genes, pseudogenes and SNP.

This file is available on the following webpage :

<http://genomeast.igbmc.fr/wiki/doku.php?id=training:introduction2r>

- Import this file into R
- What is the total number of protein coding genes ?
- Retrieve the chromosomes with more than 1,000 annotated protein coding genes annotated.
- Create a file containing chromosome number and number of protein coding genes, ordered by decreasing protein coding gene number.

Conditional execution : make choices

- Allow to execute R instructions only under certain conditions

- Syntax

```
if (condition){  
  true.block  
}  
else{  
  false.block  
}
```

➔ If condition is true, *true.block* is executed, otherwise *false.block* is executed

Loops : repeat an action (1/2)

- Allow to iteratively perform several analyses
- *For* loop
 - Syntax

```
for (variable in suite) {  
    expression  
}
```
 - ➔ Execute *expression* iteratively for each value of *variable* contained in *suite*
- *While* loop
 - Syntax

```
while (condition){  
    expression  
}
```
 - ➔ Execute *expression* while *condition* is true

Introduction to R software

- R software fundamentals
- Graphics using R
- Basic statistics using R

Introduction to graphics with R

- Data visualisation
 - An essential step in the exploratory data analysis
- R
 - A flexible and powerful way to perform graphics
e.g. `demo(graphics)` or <http://rgraphgallery.blogspot.fr>
- Principles
 - Graphics functions
 - To perform graphics
 - Options of graphics functions and graphics parameters
 - To modify graphics presentation
 - Graphics devices
 - In which all graphics operations occur : window or file

Dataset for example plots

```
library(MASS)
```

```
data(survey)
```

```
?survey #contains the responses of 237 students
```

```
#at the University of Adelaide to a number of questions
```

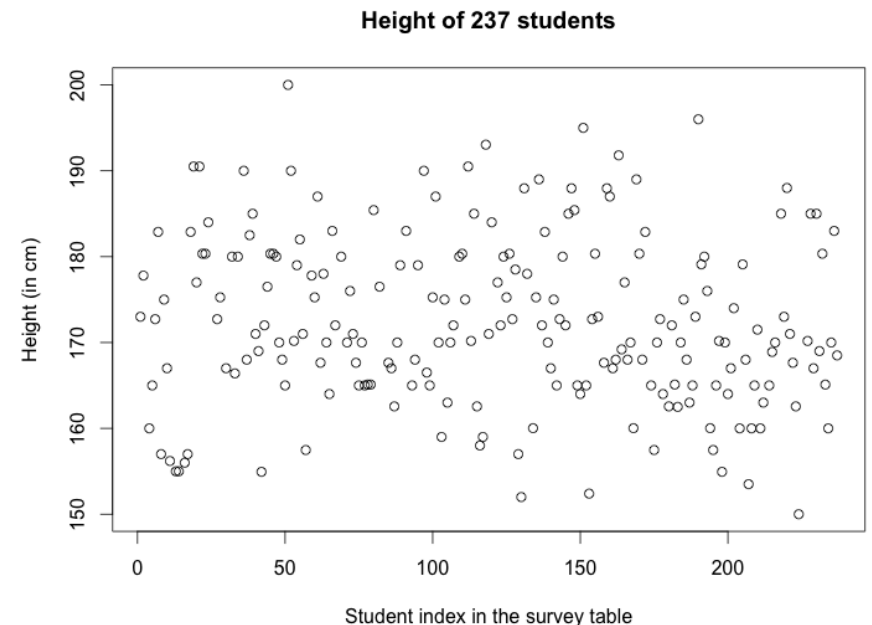
```
head(survey)
```

	Sex	Wr.Hnd	NW.Hnd	W.Hnd	Fold	Pulse	Clap	Exer	Smoke	Height	M.I	Age
1	Female	18.5	18.0	Right	R on L	92	Left	Some	Never	173.00	Metric	18.250
2	Male	19.5	20.5	Left	R on L	104	Left	None	Regul	177.80	Imperial	17.583
3	Male	18.0	13.3	Right	L on R	87	Neither	None	Occas	NA	<NA>	16.917
4	Male	18.8	18.9	Right	R on L	NA	Neither	None	Never	160.00	Metric	20.333
5	Male	20.0	20.0	Right	Neither	35	Right	Some	Never	165.00	Metric	23.667
6	Female	18.0	17.7	Right	L on R	64	Right	Some	Never	172.72	Imperial	21.000

Quantitative variables

Quantitative variable in R : numeric vector

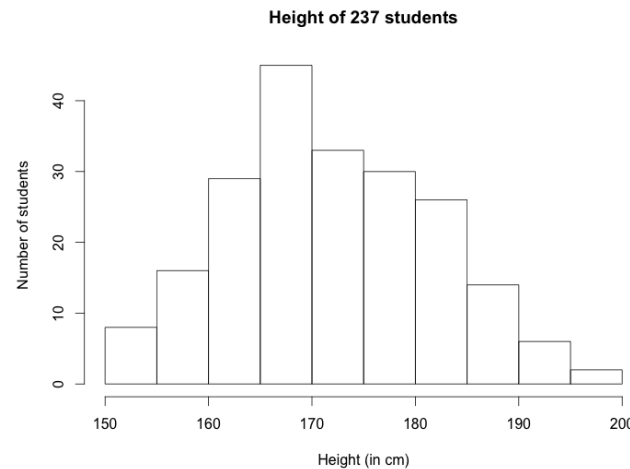
- Scatterplot : `plot()`
`plot(survey$Height,`
`main="Height of 237 students",`
`xlab="Student index in the survey table",`
`ylab="Height (in cm)"`
`)`
... not very informative...



Quantitative variables

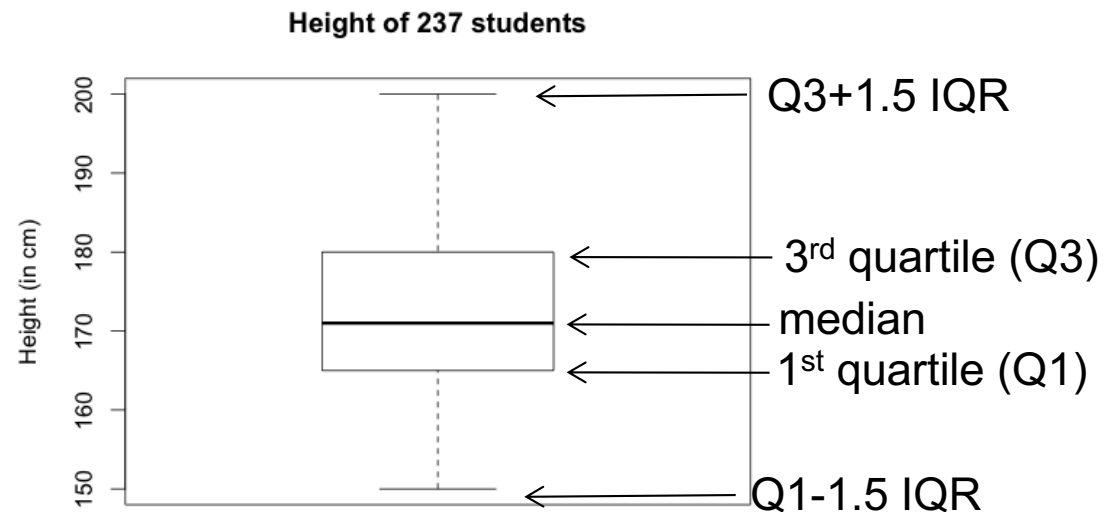
- Histogram : `hist()`

```
hist(survey$Height,  
main="Height of 237 students",  
xlab="Height (in cm)",  
ylab="Number of students")
```



- Boxplot : `boxplot()`

```
boxplot(survey$Height,  
main="Height of 237 students",  
ylab="Height (in cm)")
```

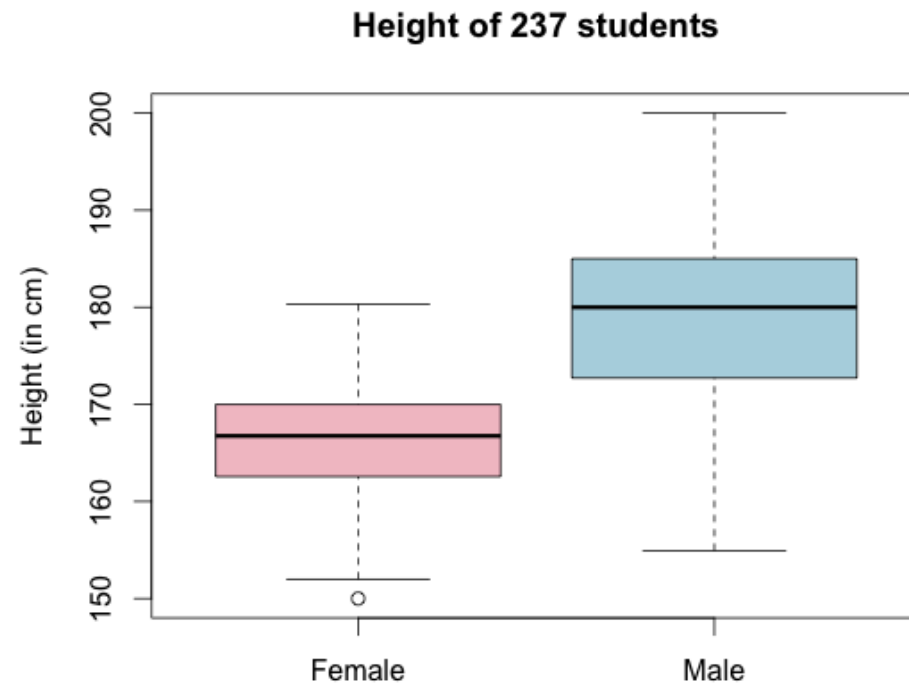


IQR=interquartile range (Q3-Q1) → contains 50% of the individuals

Quantitative variables

- Boxplots allow to easily compare distributions

```
boxplot(survey$Height ~ survey$Sex,  
        main="Height of 237 students",  
        ylab="Height (in cm)",  
        col=c("pink","lightblue")  
)
```



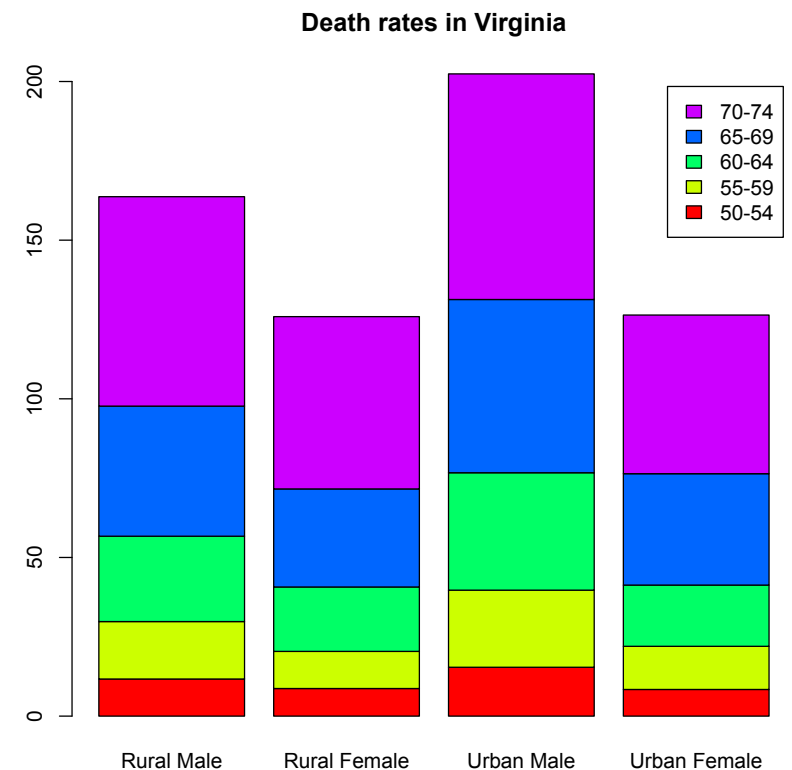
Quantitative variables

– `barplot()`

VADeaths # Death rates per 1000 in Virginia in 1940

	Rural Male	Rural Female	Urban Male	Urban Female
50-54	11.7	8.7	15.4	8.4
55-59	18.1	11.7	24.3	13.6
60-64	26.9	20.3	37.0	19.3
65-69	41.0	30.9	54.6	35.1
70-74	66.0	54.3	71.1	50.0

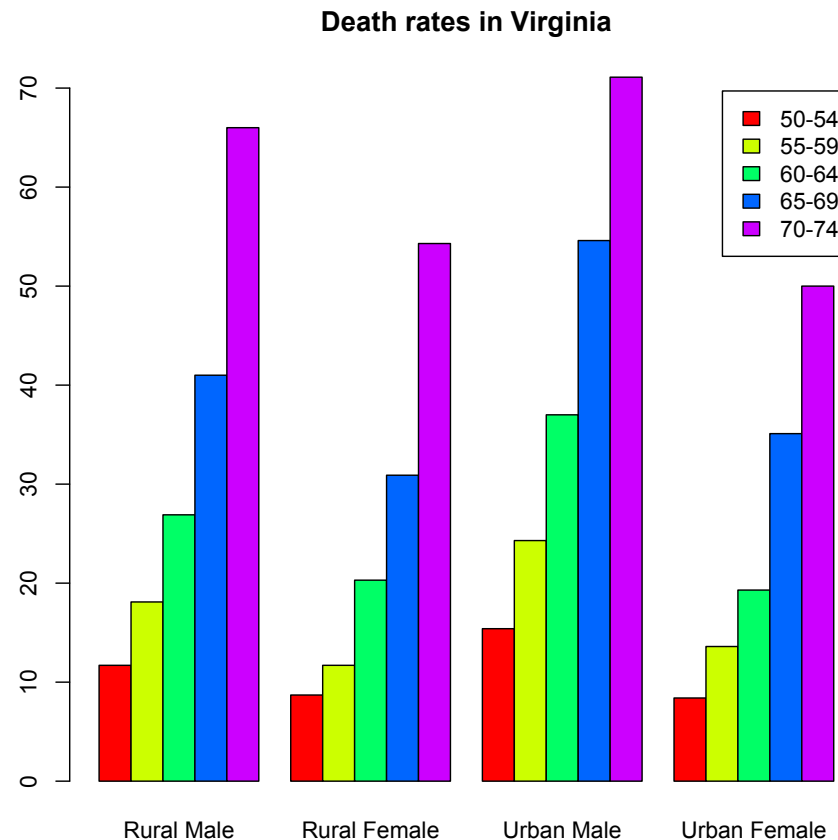
```
barplot(VADeaths, legend=rownames(VADeaths),  
col=rainbow(5), main="Death rates in Virginia")
```



Quantitative variables

– `barplot()`

```
barplot(VADeaths, legend=rownames(VADeaths),  
col=rainbow(5), main="Death rates in Virginia",  
beside=TRUE)
```



Quantitative variables

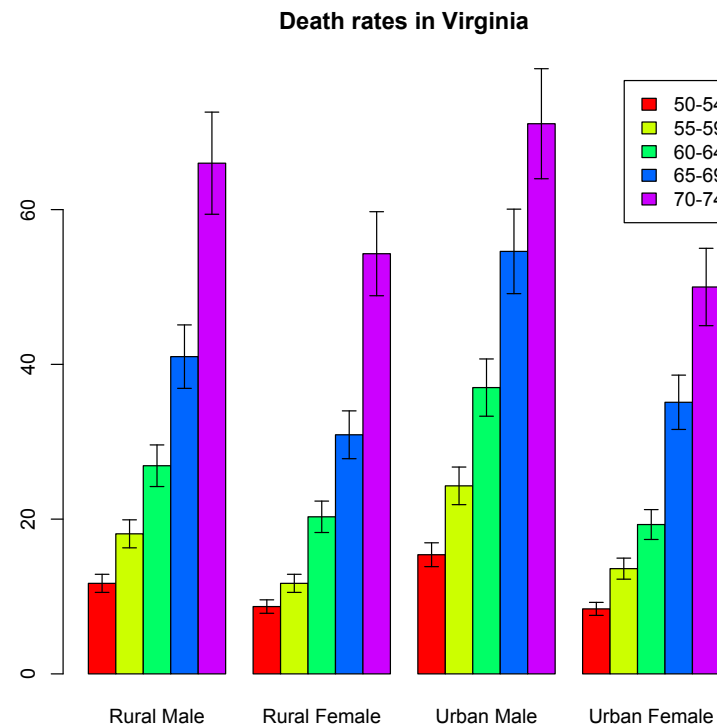
- `barplot2()` : in order to represent confidence intervals

```
myci.l = VADeaths*0.9 # just for the example...
```

```
myci.u = VADeaths*1.1
```

```
library(gplots)
```

```
barplot2(VADeaths, legend=rownames(VADeaths),  
col=rainbow(5), main="Death rates in Virginia", beside=TRUE,  
ci.l=myci.l, ci.u=myci.u, plot.ci=TRUE)
```

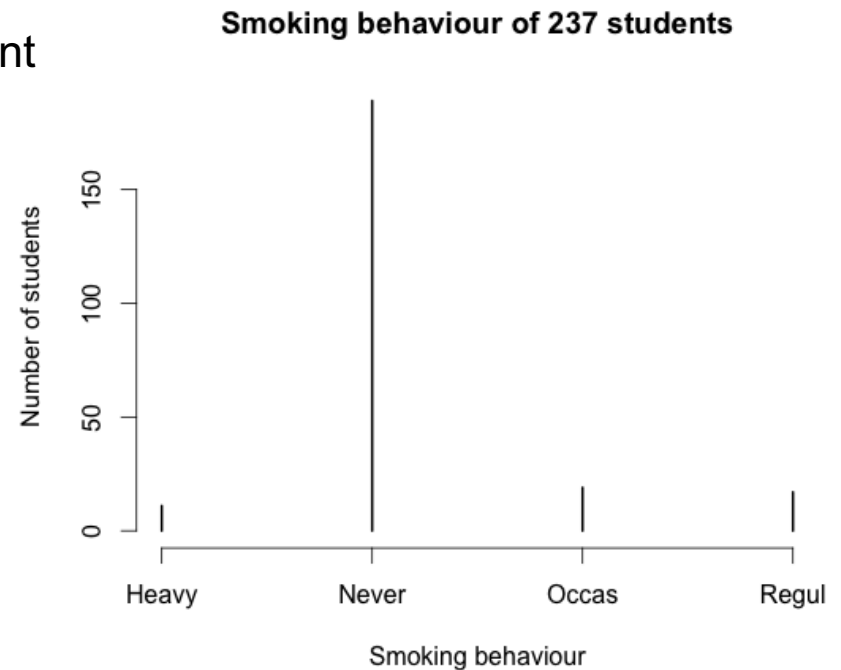


Quantitative variables

- Bar-chart (vertical lines) : `plot(...,type="h")`

```
plot(table(survey$Smoke), type="h",  
main="Smoking behaviour of 237 students",  
ylab= "Number of students",  
xlab= "Smoking behaviour")
```

N.B. : `type="h"` is the default type to represent contingency tables (`plot.table`)

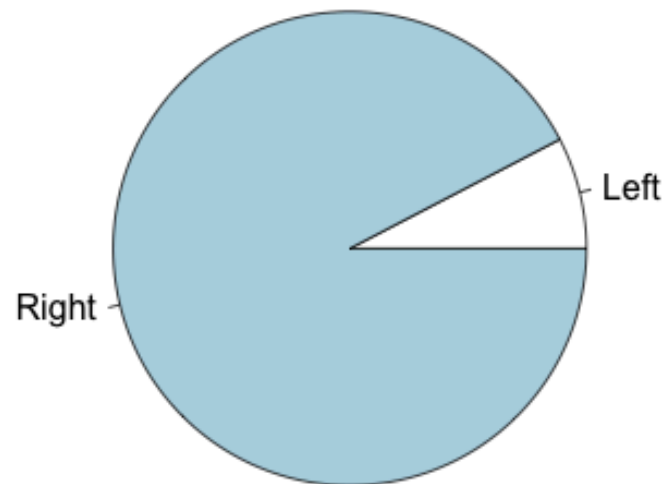


Qualitative variables

- Pie-chart

```
pie(table(survey$W.Hnd),  
main= "Writing hand of 237 students")
```

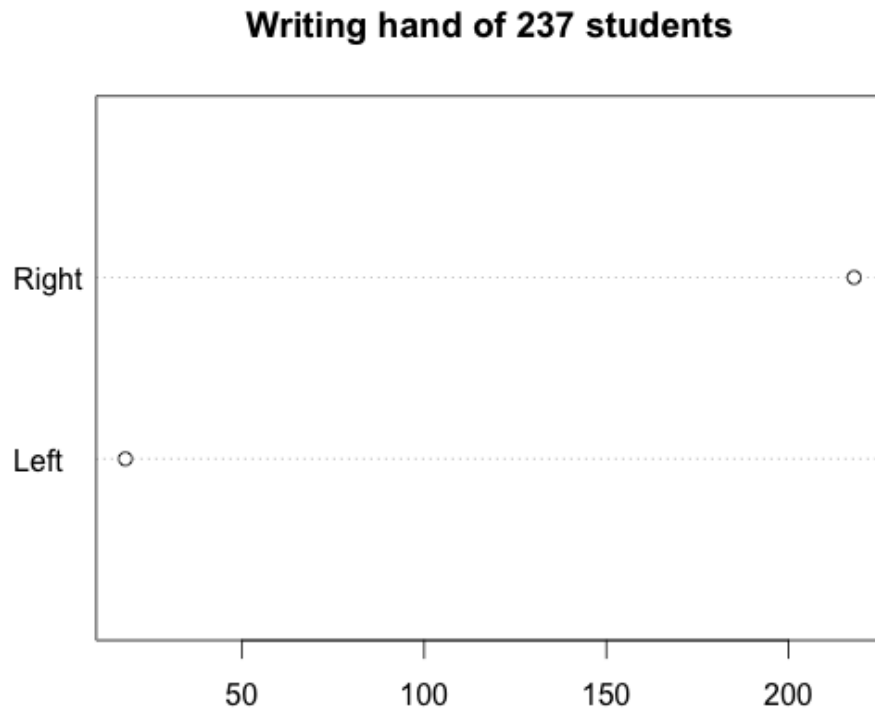
Writing hand of 237 students



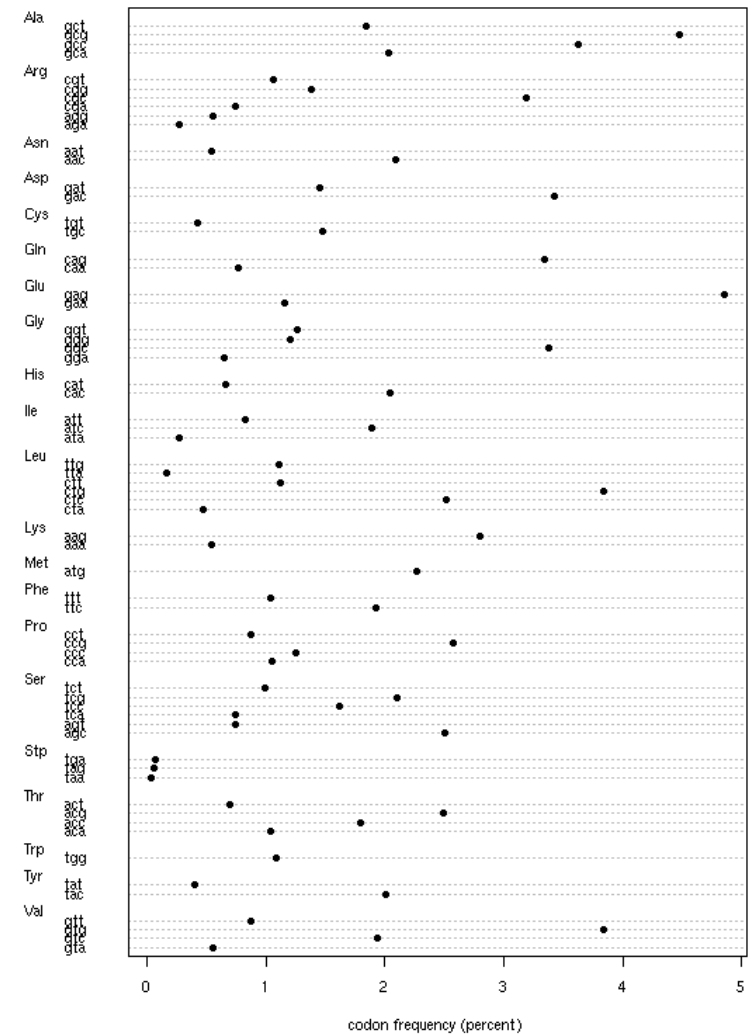
Qualitative variables

- Cleveland's dot plot

```
dotchart(table(survey$W.Hnd),  
main= "Writing hand of 237 students")
```



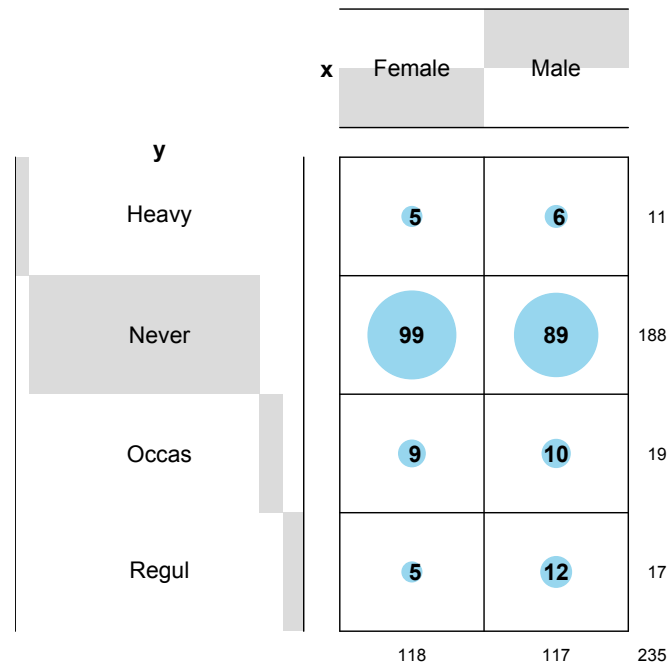
Very useful for variables with a lot of levels, e.g. codon usage :



Cross variables

- Qualitative – qualitative
 - contingency table : `balloonplot()`
- ```
library(gplots)
balloonplot(table(survey[,c("Sex","Smoke")]))
```

Balloon Plot for x by y.  
Area is proportional to Freq.

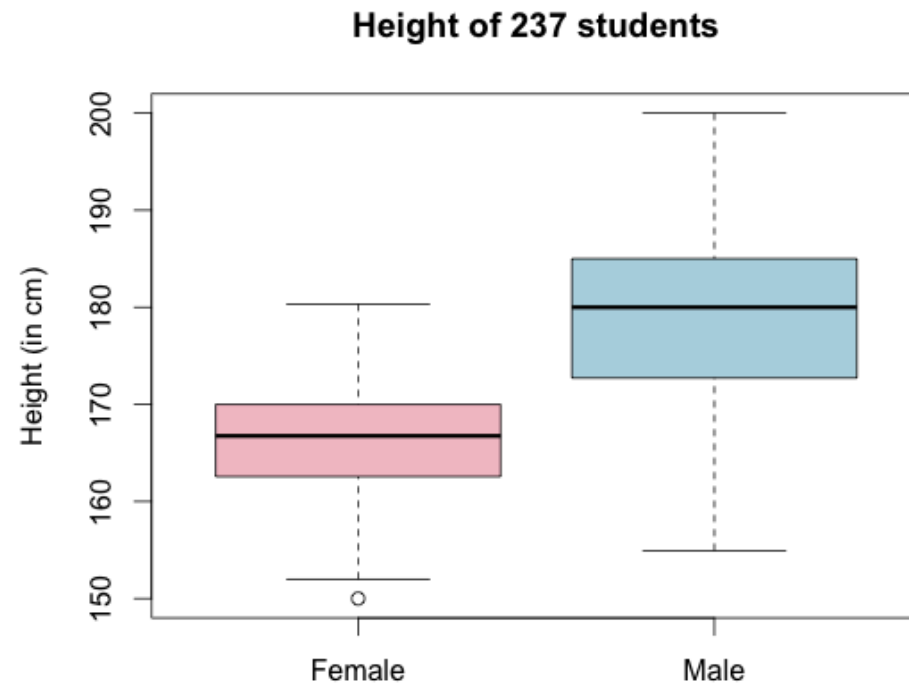


# Cross variables

- Quantitative – qualitative

- `boxplot()`

```
boxplot(survey$Height ~ survey$Sex,
 main="Height of 237 students",
 ylab="Height (in cm)",
 col=c("pink","lightblue")
)
```



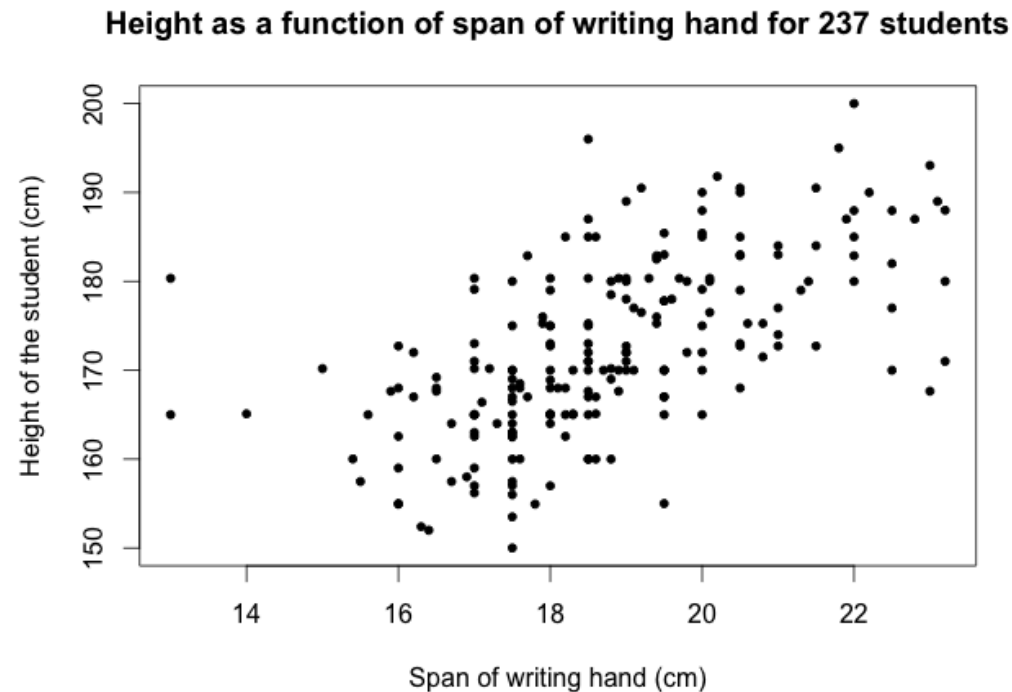
# Cross variables

---

- Quantitative – quantitative

- scatter plot : `plot()`

```
plot(survey$Wr.Hnd, survey$Height, pch=20,
xlab= "Span of writing hand (cm)",
ylab= "Height of the student (cm)",
main= "Height as a function of span of writing hand for 237 students")
```



# Cross variables

- Quantitative – quantitative

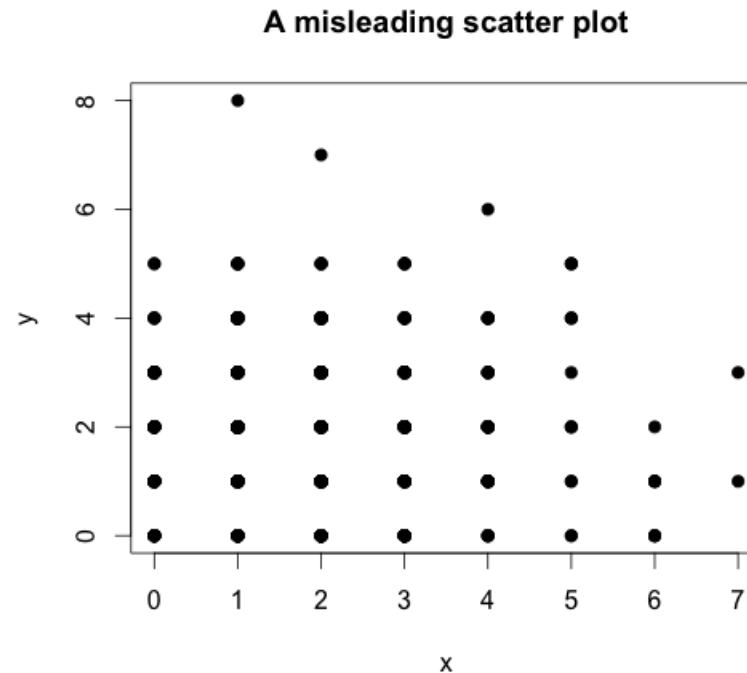
- Overlapping points problem

$n = 500$

$x = \text{rpois}(n, \text{lambda} = 2)$

$y = \text{rpois}(n, \text{lambda} = 2)$

`plot(x, y, pch = 19, main = "A misleading scatter plot")`

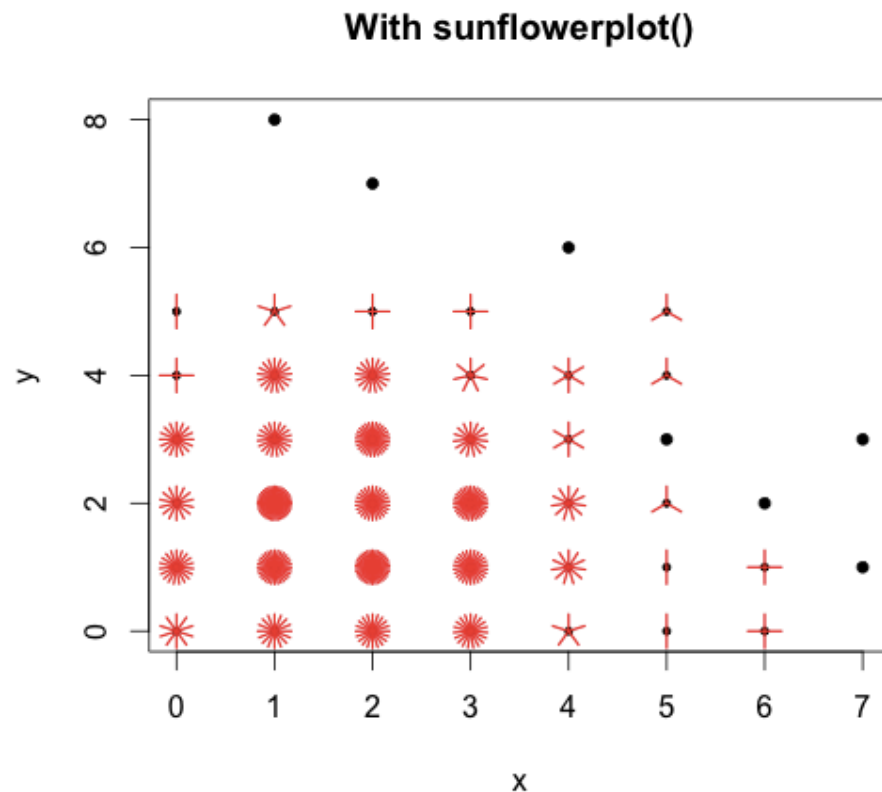


# Cross variables

- Quantitative – quantitative

- Overlapping points problem

`sunflowerplot(x, y, pch = 19, main = "With sunflowerplot()")`



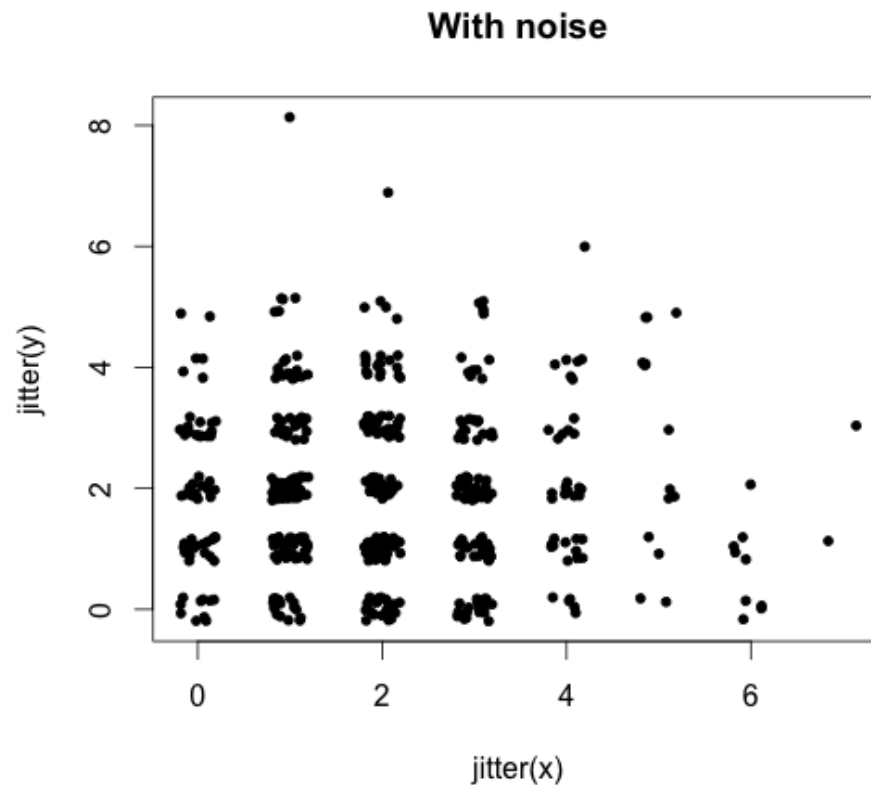
Multiple points are plotted as 'sunflowers' with multiple leaves

# Cross variables

- Quantitative – quantitative

- Overlapping points problem

```
plot(jitter(x), jitter(y), main = "With noise", pch=20)
```



A small amount of noise is added to the data



# Curves

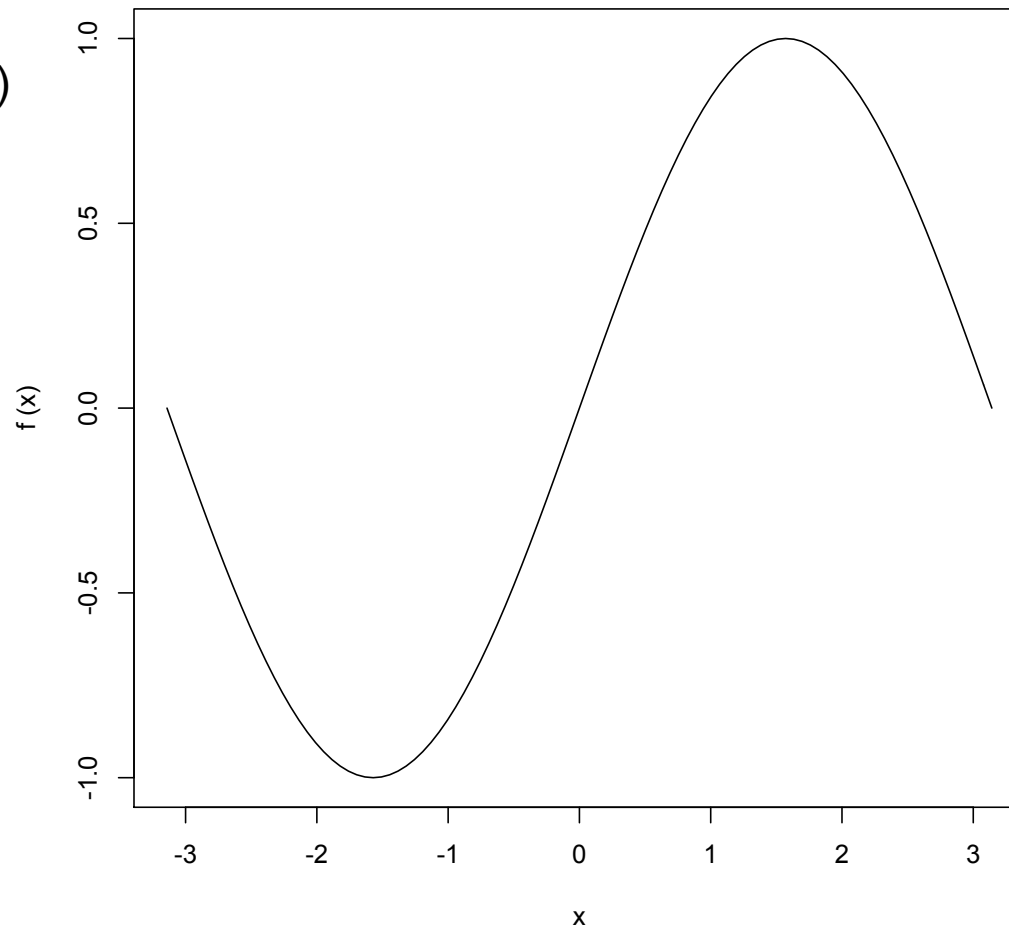
---

- Draw functions

- `curve()`

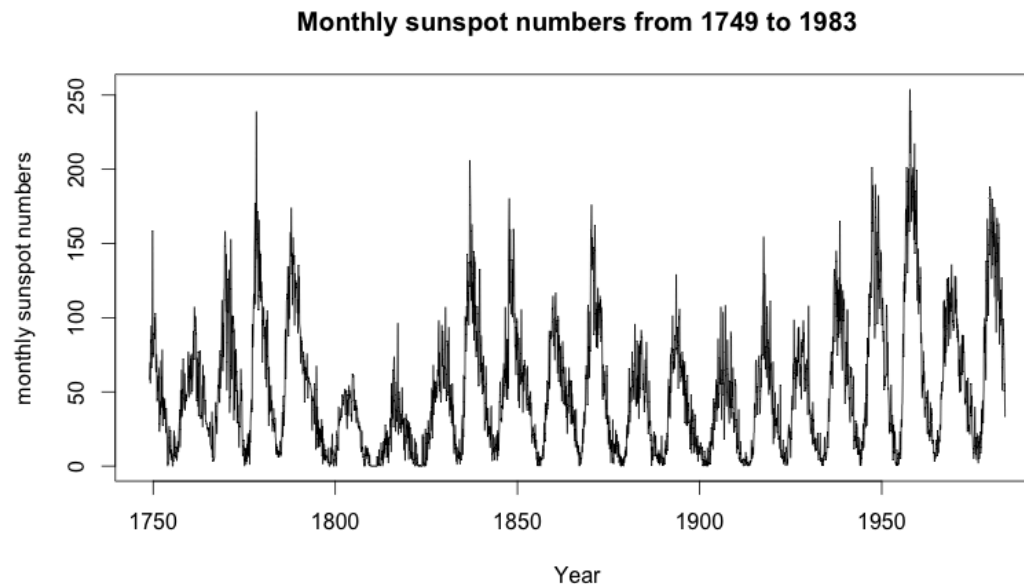
- `f = function(x){ sin(x) }`

- `curve(f, from = -pi, to = pi)`



# Curves

- Evolution of a variable as a function of a second variable
  - `plot(..., type="l")`  
e.g. time course data  
`data(sunspots)`  
`plot(time(sunspots), sunspots, type="l",`  
`main = "Monthly sunspot numbers from 1749 to 1983",`  
`xlab= "Year", ylab="monthly sunspot numbers")`



# Functions to modify plots

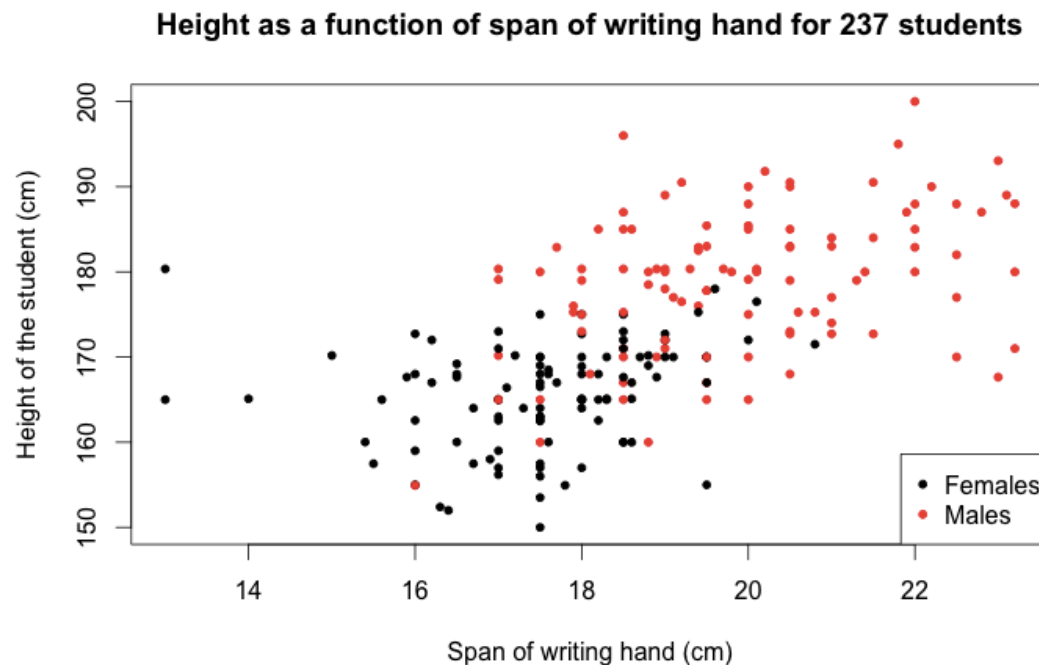
---

- Add points to a plot : `points()`
- Add lines to a plot
  - `lines()`
  - `abline()`
    - `a,b` : to specify intercept and slope
    - `h` : to specify y-value(s) for horizontal line(s)
    - `v` : to specify x-value(s) for vertical line(s)
- Add a legend : `legend()`
- Add a title : `title()`
- Add axes : `axis()`
- Add text to a plot : `text(x, y, labels)`
  - `x,y` : the coordinates where text labels should be written
- Draw segment(s) or arrow(s) : `segments()`, `arrows()`

# Functions to modify plots

Example : add a legend to a graph

```
plot(survey$Wr.Hnd,survey$Height, pch=20,
xlab= "Span of writing hand (cm)",
ylab= "Height of the student (cm)",
main= "Height as a function of span of writing hand for 237 students",col=survey$Sex)
legend("bottomright", legend=c("Females", "Males"), col=c(1,2), pch=20)
```



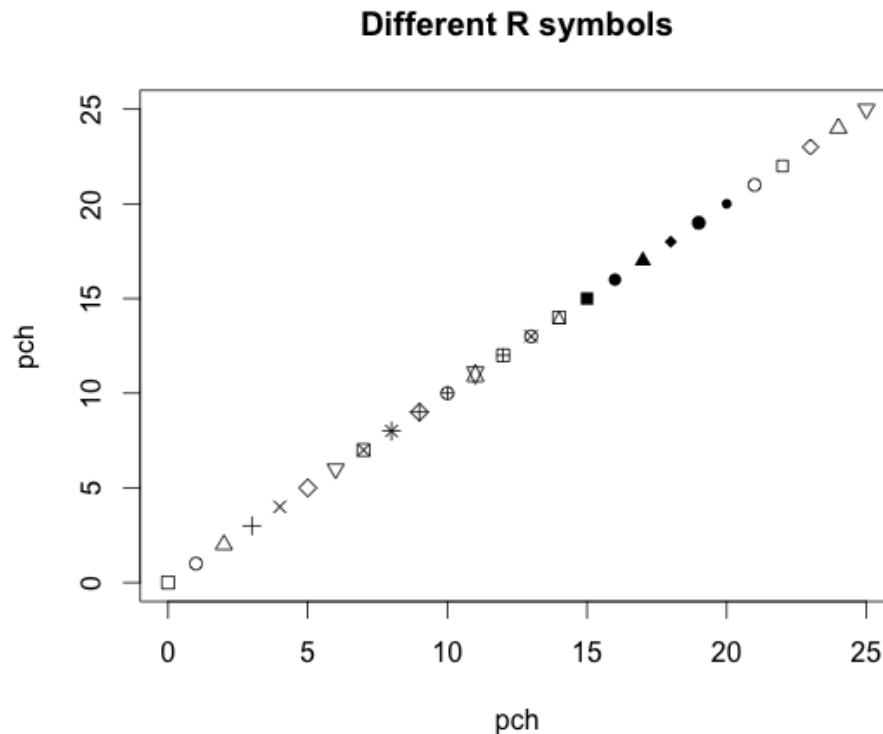
# Graphics options

---

- c.f. help page of each graphics function
  - A lot of different arguments
- Common arguments for several graphics functions
  - main : title
  - sub : subtitle
  - xlab, ylab : axes annotation
  - xlim, ylim : minimal and maximal values for the axes
  - type : “p” : points, “l” : lines, “b” : points+lines, “n” : no plotting, “h” : vertical lines
  - pch : plotting character(s) or symbol(s)
  - lty : line type(s), lwd : line width(s)
  - col : color(s), colors() list all available colours

# Graphics options

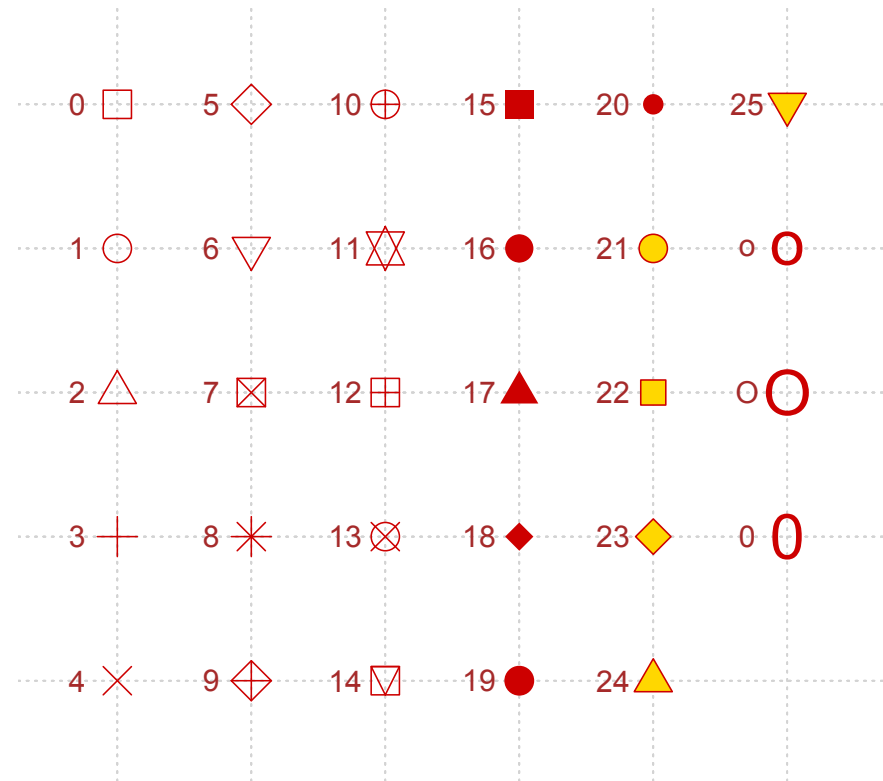
- Different plotting characters and symbols
  - 0:25 : R symbols  
`plot(0:25,0:25,pch=0:25, xlab="pch",ylab="pch",main="Different R symbols")`
  - 32:127 : ASCII characters



# Graphics options

?points

```
##----- Showing all the extra & some char graphics symbols -----
pchShow <-
function(extras = c("x", ".", "o", "O", "0", "+", "-", "|", "%", "#"),
 cex = 3, ## good for both .Device=="postscript" and "x11"
 col = "red3", bg = "gold", coltext = "brown", cextext = 1.2,
 main = paste("plot symbols : points (... pch = *, cex =",
 cex, ")"))
{
 nex <- length(extras)
 np <- 26 + nex
 ipch <- 0:(np-1)
 k <- floor(sqrt(np))
 dd <- c(-1,1)/2
 rx <- dd + range(ix <- ipch %/% k)
 ry <- dd + range(iy <- 3 + (k-1)- ipch %% k)
 pch <- as.list(ipch) # list with integers & strings
 if(nex > 0) pch[26+ 1:nex] <- as.list(extras)
 plot(rx, ry, type="n", axes = FALSE, xlab = "", ylab = "",
 main = main)
 abline(v = ix, h = iy, col = "lightgray", lty = "dotted")
 for(i in 1:np) {
 pc <- pch[[i]]
 ## 'col' symbols with a 'bg'-colored interior (where available) :
 points(ix[i], iy[i], pch = pc, col = col, bg = bg, cex = cex)
 if(cextext > 0)
 text(ix[i] - 0.3, iy[i], pc, col = coltext, cex = cextext)
 }
}
pchShow()
pchShow(c("o", "O", "0"), cex = 2.5)
```

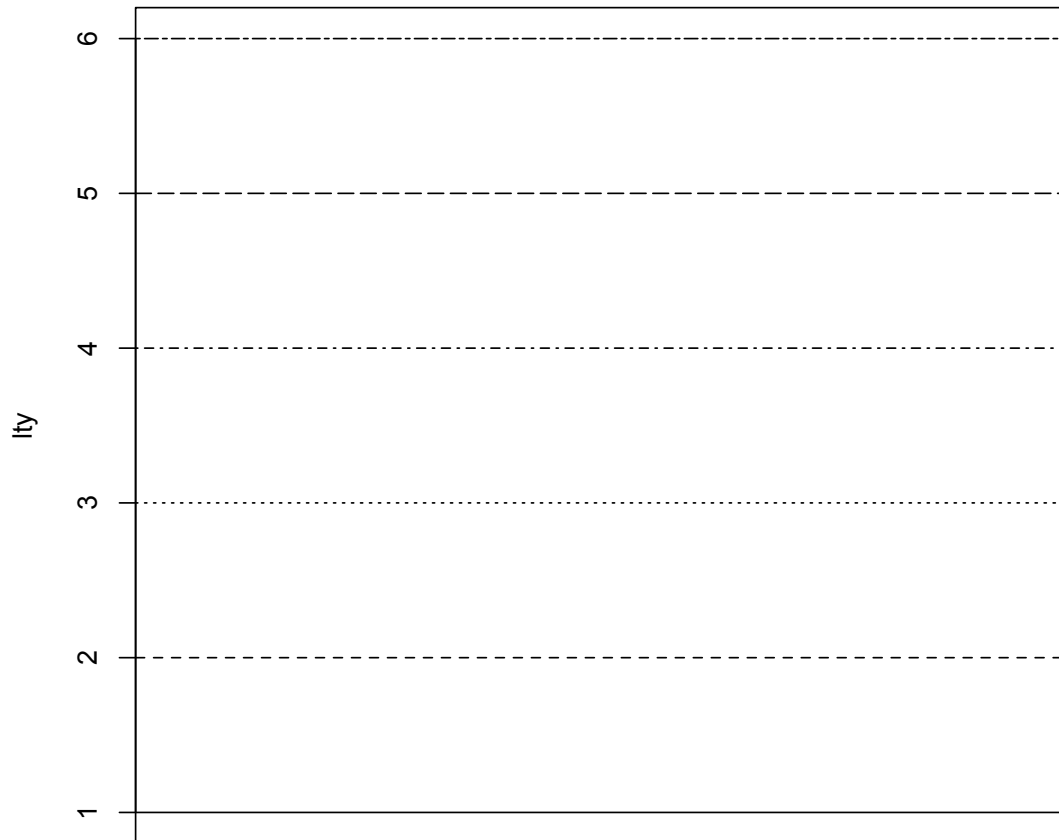


# Graphics options

---

- Different lines

```
plot(1:6,1:6,type="n",xaxt="n",ylab="lty",xlab="")
for (i in 1:6){abline(h=i,lty=i)}
```

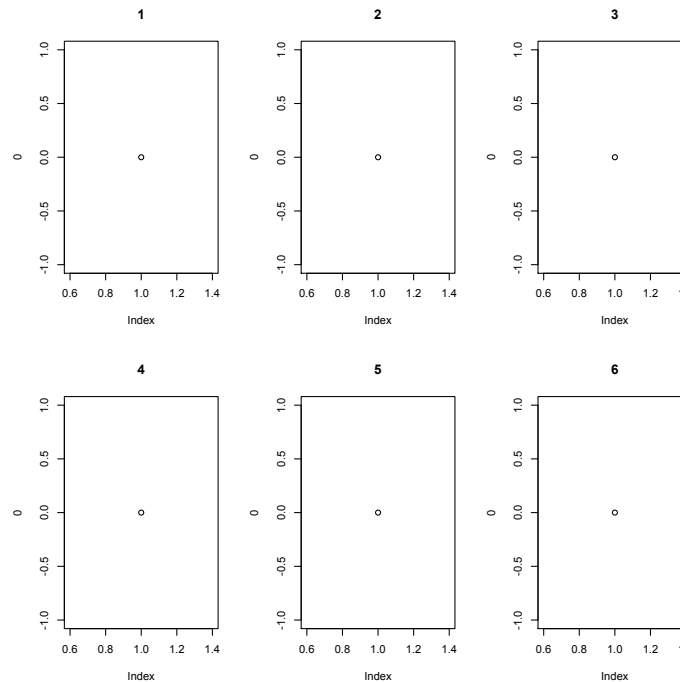




# Graphical parameters

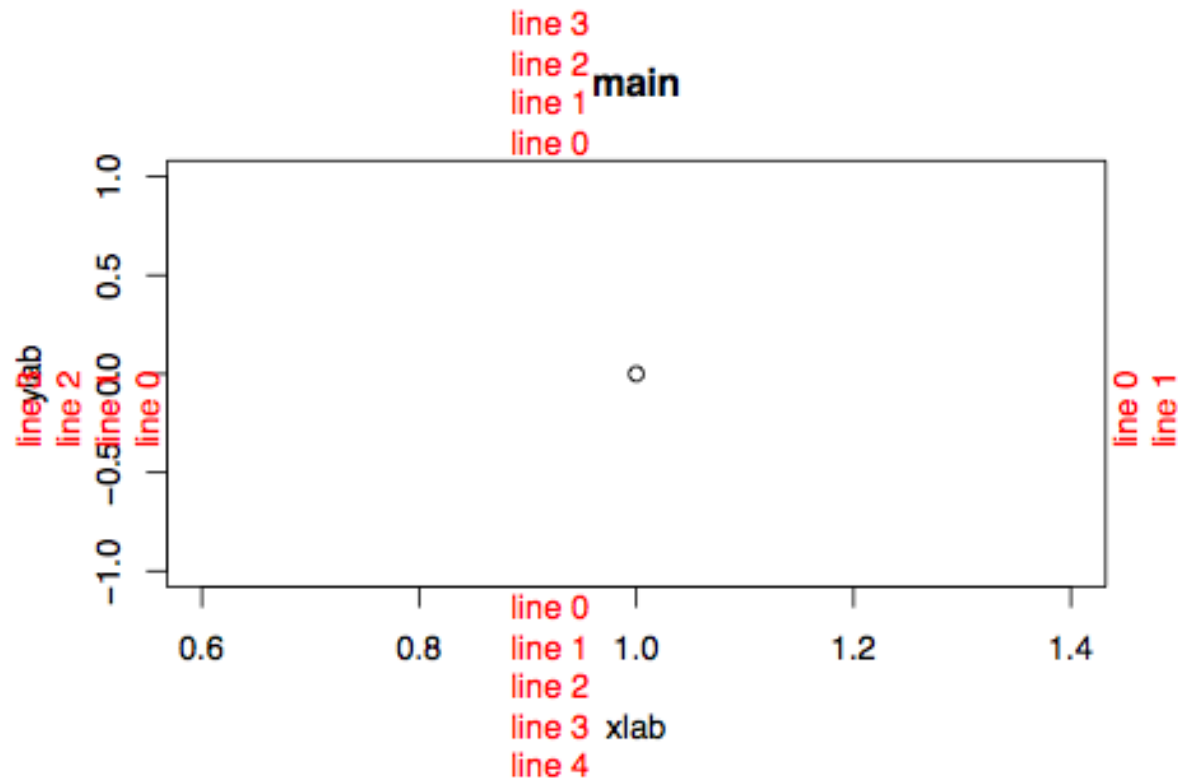
- All graphical parameters
  - ?par # how to set or query graphical parameters
  - par() # list of the 71 graphical parameters and their values
- Main graphical parameters
  - Several graphics in the same window : `mfrow = c(nrows, ncolumns)`

```
par(mfrow = c(2, 3))
for (i in 1:6) plot(0, main = i)
```



# Graphical parameters

- Margins : `mar = c(bottom, left, top, right)`
  - In text lines
  - By default, `mar = c(5,4,4,2) + 0.1`



# Graphical parameters

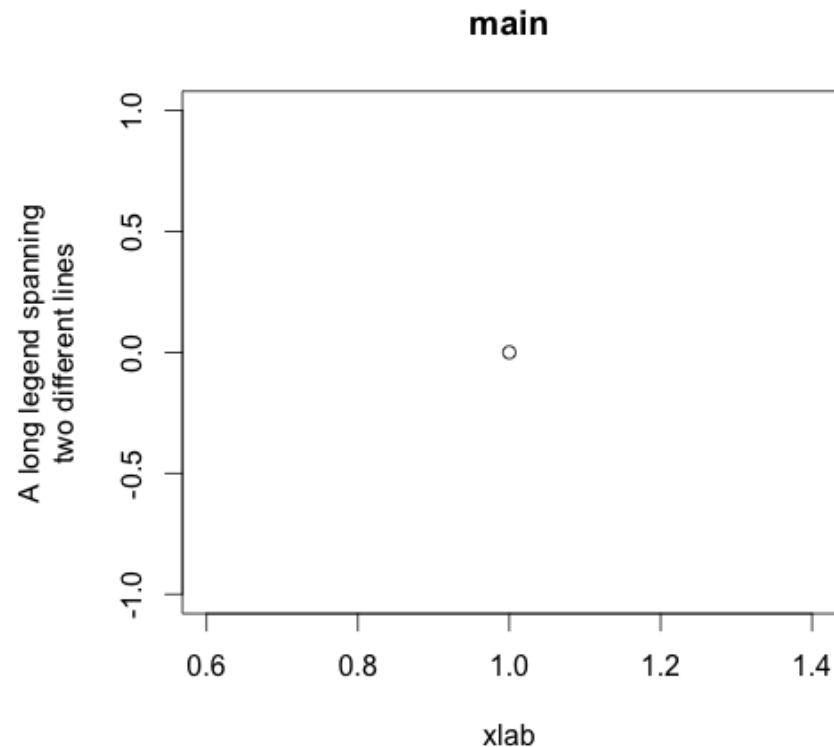
---

- Margins : `mar = c(bottom, left, top, right)`

Example : add a margin line to the left margin :

```
par(mar = par("mar") + c(0, 1, 0, 0))
```

```
plot(0, xlab = "xlab", ylab = "A long legend spanning\n two different lines", main = "main")
```



# Graphical parameters

- Colours of the elements in a graphics : `col*`

```
par(mfrow = c(2, 3))
```

```
par(col = "red")
```

```
plot(0, main = "col = \"red\"", sub = "sub title")
```

```
par(col = "black", col.axis = "red")
```

```
plot(0, main = "col.axis = \"red\"", sub = "sub title")
```

```
par(col.axis = "black", col.lab = "red")
```

```
plot(0, main = "col.lab = \"red\"", sub = "sub title")
```

```
par(col.lab = "black", col.main = "red")
```

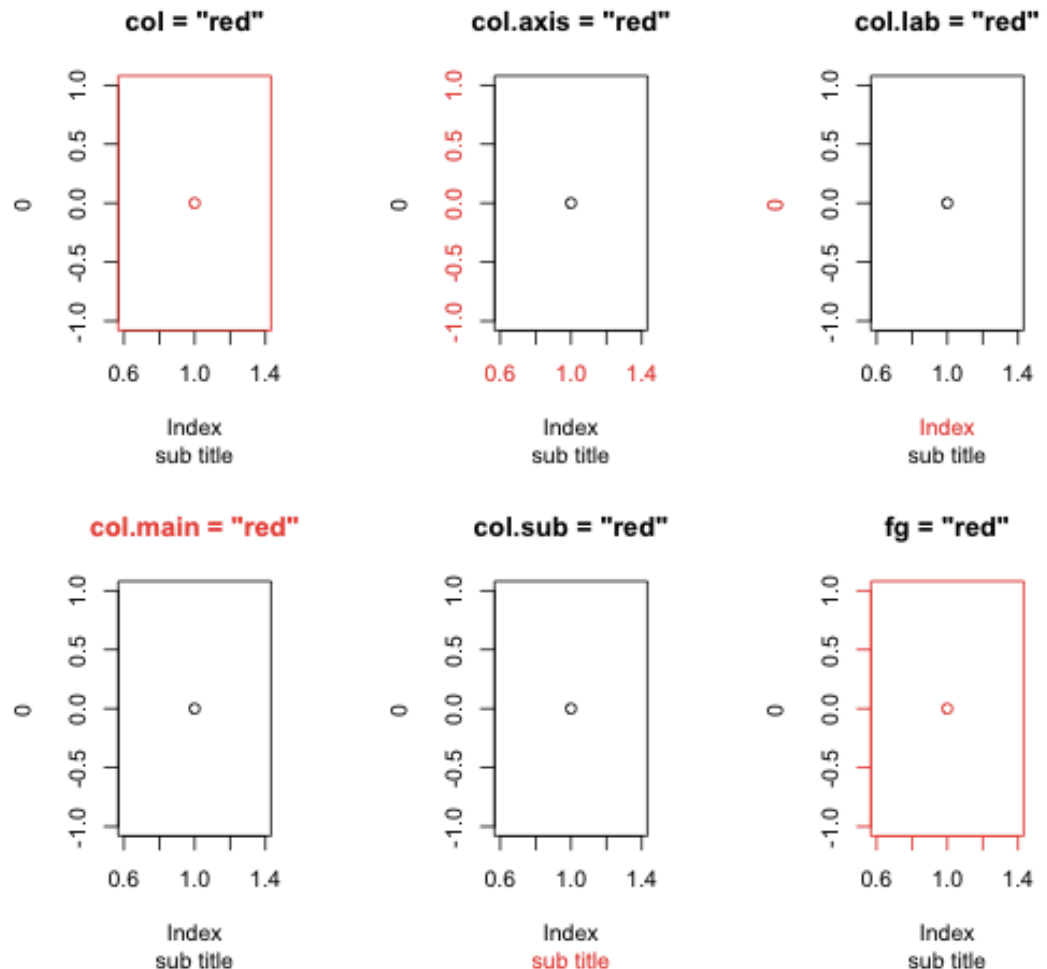
```
plot(0, main = "col.main = \"red\"", sub = "sub title")
```

```
par(col.main = "black", col.sub = "red")
```

```
plot(0, main = "col.sub = \"red\"", sub = "sub title")
```

```
par(col.sub = "black", fg = "red")
```

```
plot(0, main = "fg = \"red\"", sub = "sub title")
```

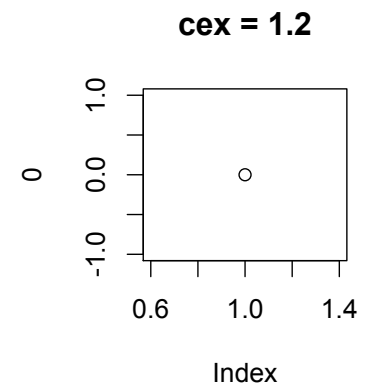
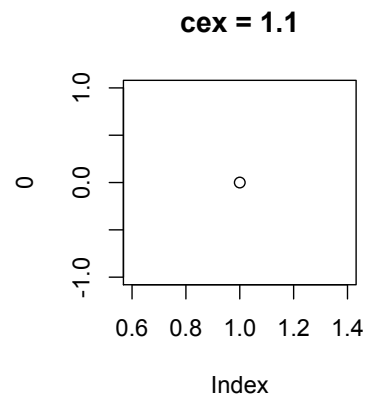
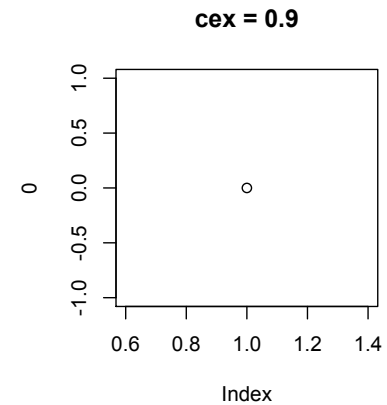
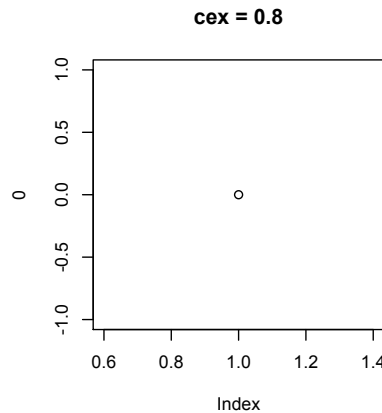


# Graphical parameters

- Relative size of characters : **cex\***

```
par(mfrow = c(2, 2))
for (cex in seq(from = 0.8, to = 1.2, length = 4)) {
 par(cex = cex)
 plot(0, main = paste("cex =", round(cex, 1)))
}
```

As for col, there are cex.main, cex.sub...



# Graphical parameters

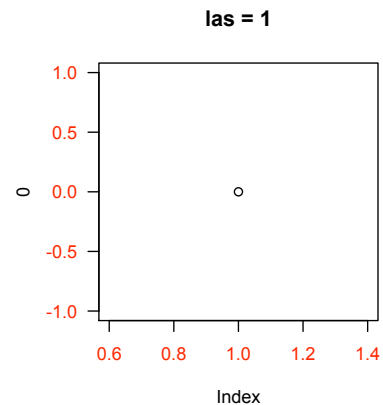
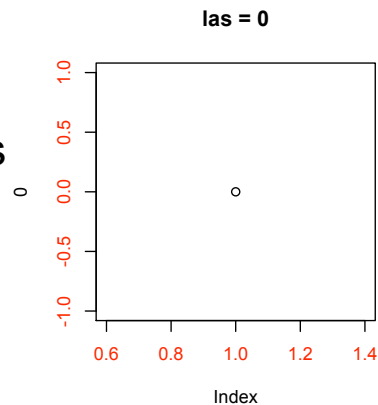
– Position of axis labels : **las**

```
par(mfrow = c(2, 2))
```

```
for (i in 0:3)
```

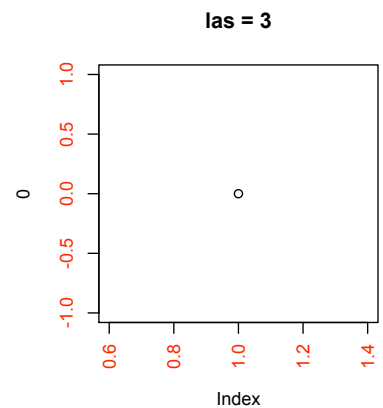
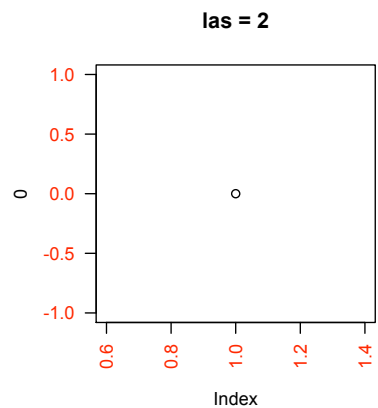
```
plot(0, las = i, main = paste("las =", i), col.axis = "red")
```

Parallel to the axis  
(default)



Horizontal

Perpendicular  
to the axis



Vertical

# Functions to interact with graphics

---

- Reads the position of the graphics cursor when the mouse button is pressed

- `locator()`

- Gives the coordinates of the graphics cursor

```
plot(survey$Wr.Hnd, survey$Height, pch=20)
```

```
locator(2) # 2 points to locate
```

- `identify()`

- Gives the label of the point on the graphics that is closest to the cursor

```
plot(survey$Wr.Hnd, survey$Height, pch=20)
```

```
in order to get the number of the corresponding students :
```

```
identify(survey$Wr.Hnd, survey$Height, rownames(survey))
```

```
type ESC to quit identify function
```

```
and get the number of the students corresponding to the points closest to the pointer
```

# Functions linked to graphics file formats

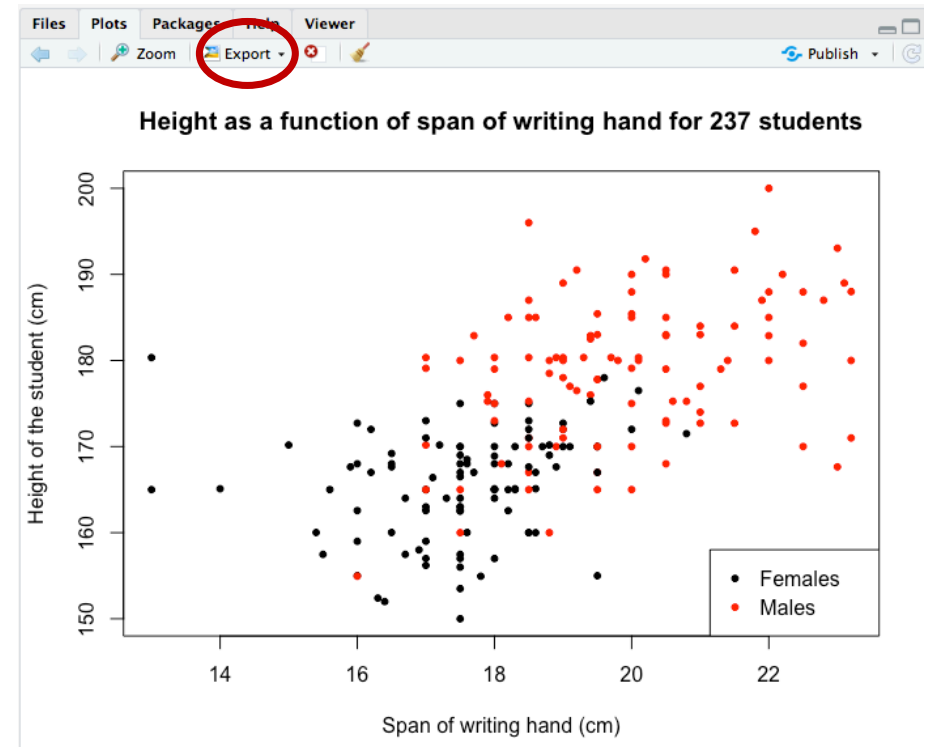
- List of available functions
  - ?Devices
- Main functions
  - pdf(), png(), jpeg(), tiff(), bmp(), postscript(), ...

- Example

- Save a graphics in a pdf file

```
pdf("myfile.pdf")
#plot instructions
dev.off() # shut down the current device
```

- Rstudio Export





# Exercise 5 (1/2)

---

Use the iris dataset available in R

> `data(iris)`

- Choose a graphics file format and save all the graphs you will obtain in this format
- Represent the distribution of the sepal length of all iris with an histogram
  - Choose an appropriate name for the title and the two axes
  - Add above each bar the number of flowers in each class
- Compare sepal length between the different species with a boxplot
  - Choose an appropriate name for the title and the two axes
  - Choose a different colour for each box

# Exercise 5 (2/2)

---

- In the same graphics windows, represent a boxplot for each quantitative variable from the iris dataset allowing to compare the distribution of this variable between the different species

For this purpose :

- Partition the graphics windows
  - Write a loop in order to represent one graph for each quantitative variable
- 
- Compare petal and sepal length for all flowers with a scatterplot
    - Choose an appropriate title for each axis
    - Choose a type of points different from the default one
- 
- Represent the same comparison as on previous graphs, with different colours for the different species
    - Add a legend to the graph

# Introduction to R software

---

- R software fundamentals
- Graphics using R
- Basic statistics using R

# One dimension descriptive statistics

---

- Mean : **mean**
- Median : **median**
- Variance : **var**
  - Variance estimation :
- Standard deviation : **sd**
- Quantiles : **quantile**
- Minimum, maximum : **min, max, range**

$$\hat{\sigma}^2 = \frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X})^2$$

- **summary**

```
> x=rnorm(1000)
```

```
> summary(x)
```

```
 Min. 1st Qu. Median Mean 3rd Qu. Max.
-2.89600 -0.64130 0.03306 0.05083 0.71320 3.19600
```

# Exercise 6

---

For this exercise use the dataset available in humanGenomeSummary.txt file.

- What is the mean and maximum number of protein coding genes per chromosome ?
- Which chromosome contains the highest / smallest number of protein coding genes ?
- Represent the distribution of the number of protein coding genes per chromosome using a bar-chart

# Two dimensions descriptive statistics

---

- Covariance : `cov`
  - Definition :  $\text{cov}(X,Y) = E(XY) - E(X)E(Y)$
  - If X and Y are independent then  $\text{cov}(X,Y) = 0$
  
- Correlation : `cor`
  - Definition :  $R(X,Y) = \frac{\text{cov}(X,Y)}{\sigma(X)\sigma(Y)}$
  - Pearson : calculation using observed values of X and Y  
→ `method = "pearson"` (default)
  - Spearman : calculation using the ranks of observed values of X and Y  
→ `method = "spearman"`
  - `cor(x,y)` : correlation between x and y if x and y are vectors
  - `cor(m)` : correlation between the columns of m if m is a matrix or data.frame

# Exercise 7

---

For this exercise use the dataset available in humanGenomeSummary.txt file.

- Calculate the Pearson correlation coefficient between chromosome size and the number of protein coding genes.
- Calculate the Pearson correlation coefficients between all pairs of quantitative variables. Represent all variables as a function of each other variable from the dataset in order to visualize these correlations (for this purpose use the pairs function).
- Calculate the number of genes, pseudogenes and SNP per kb for each chromosome.
- Are genes and pseudogenes numbers per kb correlated ? And SNP and genes numbers per kb ? To answer these questions calculate the Pearson correlation coefficient and perform a graphical representation.

# Hypothesis testing

---

- Definition
  - Method of statistical inference that allows to accept or reject the validity of hypotheses relative to one or several populations, based on the study of one or several random samples
- Example
  - Using the sample of iris available in R, can we say that sepal length is significantly different between *versicolor* and *virginica* species ?



# Hypothesis testing principle

## 1. Choice of the hypothesis to test

---

- **H0 : null hypothesis**
  - Formulated in order to be rejected
  - Example
    - H0 = “There is no significant sepal length difference between *versicolor* and *virginica* species”
- **H1 : alternative hypothesis**
  - Negation of H0
  - Example
    - H1 : “There is a significant sepal length difference between *versicolor* and *virginica* species”
- **Hypothesis test**
  - Decision rule which allows to decide between H0 and H1

# Hypothesis testing principle

## 1. Choice of the hypothesis to test

---

- Bilateral test
  - Example
    - Sepal length from *versicolor* species could be either higher or lower than from *virginica* species
  - In R : **alternative** = “two.sided” (default)
- Unilateral test
  - The theory predicts the direction of variation
  - Example
    - Is the frequency of lung cancers higher among smokers than among non-smokers ?
      - H0 : The frequency of lung cancers is not significantly higher among smokers than among non-smokers
      - H1 : There is significantly more lung cancers among smokers than among non-smokers
  - In R : **alternative** = “less” or “greater” (to specify)

# Hypothesis testing principle

## 2. Choice of a statistical test

---

- This choice depend on
  - The type of data
  - The type of hypothesis to be tested
  - The hypotheses that can be done on the characteristics of the studied population (e.g. normality, equality of variances)
- Parametric and non-parametric tests
  - Parametric
    - Assume data come from a type of probability distribution
    - Most often a normal distribution → can be tested using Shapiro-Wilk test : `shapiro.test`
  - Non-parametric
    - No assumptions about the probability distributions of the variables being assessed
- Paired data
  - Paired e.g. Two treatments have been tested on the same individuals
    - In R : `paired=TRUE`
  - Not paired e.g. Two treatments have been tested on different individuals
    - In R : `paired=FALSE` by default

# Hypothesis testing principle

## 2. Choice of a statistical test

---

- A lot of tests implemented in R :

`apropos("test")`

```
[1] ".valueClassTest" "Box.test" "PP.test" "ansari.test" "bartlett.test"
[6] "binom.test" "chisq.test" "cor.test" "file_test" "fisher.test"
[11] "fligner.test" "friedman.test" "kruskal.test" "ks.test" "mantelhaen.test"
[16] "mauchley.test" "mauchly.test" "mcnemar.test" "mood.test" "oneway.test"
[21] "pairwise.prop.test" "pairwise.t.test" "pairwise.wilcox.test" "power.anova.test" "power.prop.test"
[26] "power.t.test" "prop.test" "prop.trend.test" "quade.test" "shapiro.test"
[31] "t.test" "testPlatformEquivalence" "testVirtual" "var.test" "wilcox.test"
```

# Hypothesis testing principle

## 2. Choice of a statistical test

---

- 2 samples comparison
  - Parametric
    - Dispersion
      - Fisher-Snedecor test : `var.test`
    - Central tendency
      - Student test : `t.test( ... , var.equal=T)`
      - Welch test : `t.test`
  - Non-parametric
    - Dispersion
      - Ansari test : `ansari.test`
    - Central tendency
      - Wilcoxon (Mann-Whitney) test : `wilcox.test`
  - Comparison of distributions
    - Kolmogorov-Smirnov test : `ks.test`

# Hypothesis testing principle

## 2. Choice of a statistical test

---

- Comparison of k samples ( $k > 2$ )
  - Parametric
    - Dispersion
      - Bartlett test : `bartlett.test`
    - Central tendency
      - Anova : `anova`
  - Non-parametric
    - Dispersion
      - Fligner test : `fligner.test`
    - Central tendency
      - Kruskal-Wallis test (independent samples) : `kruskal.test`
      - Friedman test (dependant samples) : `friedman.test`

# Hypothesis testing principle

## 3. Choice of a type I error rate

---

- Different types of errors / risks

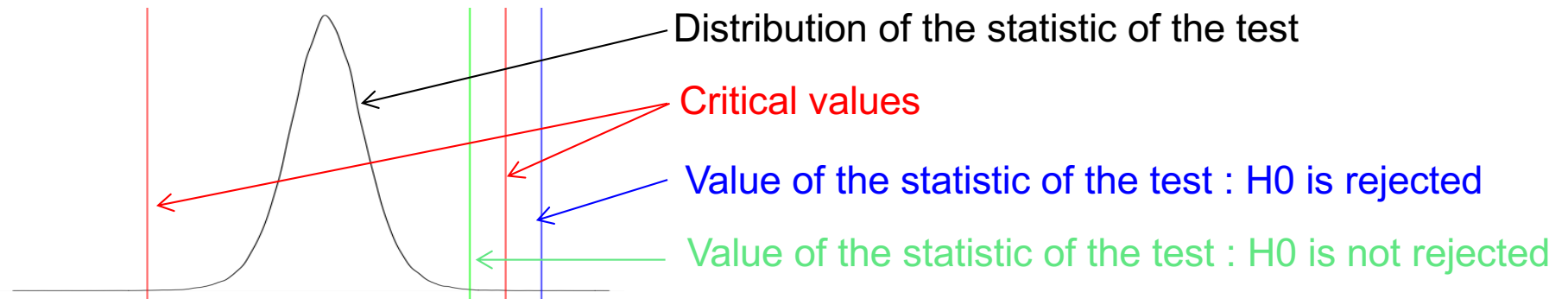
|          |    | Reality (unknown) |           |
|----------|----|-------------------|-----------|
|          |    | H0                | H1        |
| Decision | H0 | $1-\alpha$        | $\beta$   |
|          | H1 | $\alpha$          | $1-\beta$ |

- Type I error :  $\alpha = \text{prob}(\text{reject } H_0 \mid H_0 \text{ is true})$ 
  - $\alpha$  should be chosen by the experimenter before the experiment  
... not according to the data !

# Hypothesis testing principle

## 4. Decision

---



- P-value
  - Probability of obtaining a statistic at least as extreme as the observed value, assuming that the null hypothesis is true
- Decision
  - If  $p\text{-value} < \alpha$ , we reject  $H_0$  with a type I error  $\alpha$
  - If  $p\text{-value} > \alpha$ , we can not reject  $H_0$



# Example

- Using the sample of iris available in R, can we say that sepal length is significantly different between *versicolor* and *virginica* species ?

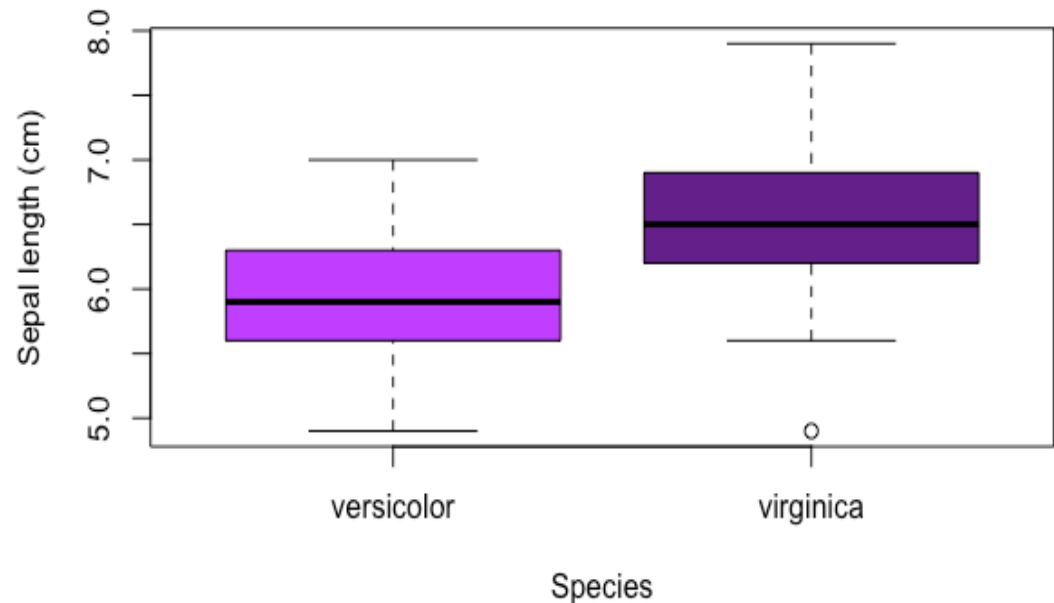
H0 = "There is no significant sepal length difference between *versicolor* and *virginica* species"

H1 : "There is a significant sepal length difference between *versicolor* and *virginica* species"

```
data(iris)
```

```
selectspecies = (iris$Species=="versicolor" | iris$Species=="virginica")
```

```
boxplot(iris$Sepal.Length[selectspecies] ~ factor(iris$Species[selectspecies]) ,
xlab="Species", ylab="Sepal length (cm)", col=c("darkorchid1","darkorchid4"))
```



# Example

---

```
Normality of the data ? Shapiro-Wilk test
```

```
H0 : we assume this is a sample from a normal distribution
```

```
H1 : we do not assume this is a sample from a normal distribution
```

```
Let $\alpha=0.05$
```

```
shapiro.test(iris$Sepal.Length[iris$Species=="versicolor"])
```

```
Shapiro-Wilk normality test
```

```
data: iris$Sepal.Length[iris$Species == "versicolor"]
```

```
W = 0.97784, p-value = 0.4647
```

```
shapiro.test(iris$Sepal.Length[iris$Species=="virginica"])
```

```
Shapiro-Wilk normality test
```

```
data: iris$Sepal.Length[iris$Species == "virginica"]
```

```
W = 0.97118, p-value = 0.2583
```

```
Conclusion : we can not reject the hypothesis of a normal distribution
```

```
We will therefore use parametric tests
```

# Example

---

```
Equality of variances ? : Fisher-Snedecor test
```

```
H0 : variance of sepal length is not different between versicolor and virginica species
```

```
H1 : this variance is significantly different between versicolor and virginica species
```

```
Let $\alpha=0.05$
```

```
var.test(iris$Sepal.Length[iris$Species=="versicolor"], iris$Sepal.Length[iris$Species=="virginica"])
```

```
#or var.test(iris$Sepal.Length[selectspecies] ~ factor(iris$Species[selectspecies]))
```

```
F test to compare two variances
```

```
data: iris$Sepal.Length[iris$Species == "versicolor"] and iris$Sepal.Length[iris$Species == "virginica"]
```

```
F = 0.65893, num df = 49, denom df = 49, p-value = 0.1478
```

```
alternative hypothesis: true ratio of variances is not equal to 1
```

```
95 percent confidence interval:
```

```
0.3739257 1.1611546
```

```
sample estimates:
```

```
ratio of variances
```

```
0.6589276
```

```
Conclusion : We can not reject the hypothesis of variance equality
```

```
We will therefore use a Student's t test to compare means
```

# Example

---

# Equality of means ? Student's t test

# H0 : mean sepal length is not different between *versicolor* and *virginica* species

# H1 : this mean is significantly different between *versicolor* and *virginica* species

# Let  $\alpha=0.05$

```
t.test(iris$Sepal.Length[iris$Species=="versicolor"], iris$Sepal.Length[iris$Species=="virginica"],
var.equal=TRUE)
```

```
#or t.test(iris$Sepal.Length[selectspecies] ~ factor(iris$Species[selectspecies]) , var.equal=TRUE)
```

Two Sample t-test

```
data: iris$Sepal.Length[iris$Species == "versicolor"] and iris$Sepal.Length[iris$Species == "virginica"]
```

```
t = -5.6292, df = 98, p-value = 1.725e-07
```

```
alternative hypothesis: true difference in means is not equal to 0
```

```
95 percent confidence interval:
```

```
-0.8818516 -0.4221484
```

```
sample estimates:
```

```
mean of x mean of y
```

```
5.936 6.588
```

# Conclusion : we reject H0, mean sepal length is significantly different between *versicolor* and *virginica* species

# Exercise 8

---

We measured on patients the blood level of a molecule before and after a treatment. These measures are available in molecule.txt file.  
Does this treatment allow to significantly increase the level of this molecule ?