Session 2: renforcement de R

Teachers: Magali Berland, Claire Vandiedonck
Helpers: Antoine Bridier-Nahmias, Yves Clément, Bruno Toupance, Jacques van Helden

Plan de la session 2: renforcement de R

1. Contrôles de flux:
    a. Boucles versus vectorisation
    b. Executions conditionnelles

2. Implémenter ses propres fonctions

3. Paquets/librairies

4. Génération de figures à façon avec R bases

5. Tidyverse: tidyr, dplyr, ggplot2 -> cf. other slides

1. Contrôle de flux

1.1. Conditional executions

Conditional executions: the basis

Aim:

- To perform a test with a logical outcome
  - comparison: ==, != (different), >, < , >=, <=
  - what is an element of a vector using : %in%
  - etc…
- And if the outcome is « TRUE » to execute the commands between { and }

Do you know what is TRUE?
Test your knowledge with the swirl tutorial on logical expressions:
1. R Programming -> 8: Logic

Syntax:

using **if(**« the condition »**)** followed by **{**«commands to be performed **if TRUE**»**}**

```
if (condition){
    instruction 1
    instruction 2
    …
}
```

```
> a <- 0
if (a == 0) {
print ("hello")
}
[1] "hello"
> if (a != 0) {
    print ("a is different from zero")
}
>                    # here R did not execute print ("a is  different from zero")
                     # since the condition was FALSE)
```

Conditional executions: alternative conditions

Syntax:

using **if(**« the condition »**)** followed by **{**«commands to be performed **if TRUE**»**}**

adding **else()** followed by **{**«alternative commands if condition was FALSE»**}**

adding **else if()** followed by **{**«alternative commands if new condition is TRUE»**}**

```
a <- 3
if (a < 3.14) {
        print ("a is < 3.14 ")
        } else {
        print ("a is > 3.14")
}
[1] "a is < 3.14 "
```

```
a <- 3.14
if (a < 3.14) {
        print ("a is < 3.14 ")
        } else if (a >3.14){
        print ("a is > 3.14")
        } else {print ("a is equal to 3.14")
 }
[1] "a is equal to 3.14 "
```

Conditional executions: multiple conditions (1)

Example with multiple conditions:

using « & » = et, « | » = or

```
a <- 11
if ( (a < 2) & (a < 10) ) {
  print ("both conditions are true")
} else if ( (a < 2) | (a < 10) ) {
  print ("one of the two conditions is true")
} else {
  print ("none of the conditions is verified")
}
[1] "none of the conditions is verified“
```

```
a <- 6
if ( (a < 2) & (a < 10) ) {
  print ("both conditions are true")
} else if ( (a < 2) | (a < 10) ) {
  print ("one of the two conditions is true")
} else {
  print ("none of the conditions is verified")
}
[1] "one of the two conditions is true"
```

Conditional executions: multiple conditions (1)

💣If your conditions are not mutually exclusive, order carefully your conditions

```
a <- 1
if ( (a < 2) & (a < 10) ) {
  print ("both conditions are true")
} else if ( (a < 2) | (a < 10) ) {
  print ("one of the two conditions is true")
} else {
  print ("none of the conditions is verified")
}
[1] “both conditions are verified“
```

```
a <- 1
if ( (a < 2) | (a < 10) ) {
 print ("one of the two conditions is true")
} else if ( (a < 2) & (a < 10) ) {
  print ("both conditions are true")
} else {
  print ("none of the conditions is verified")
}
[1] "one of the two conditions is true"
```

💣R executes the command of the 1rst condition which is TRUE

Function ifelse()

ifelse()

- returns a vector of same length as the length of the tested vector
- for each element of the tested vector, the elements of the returned vector are defined depending on whether the condition is TRUE or FALSE

syntax: **ifelse(** test, yes, no **)**

```
a <- 3
ifelse(a == 3.14, "a is equal to pi", "a is different from pi" )
[1] "a is different from pi"
a <- 3.14
ifelse(a == 3.14, "a is equal to pi", "a is different from pi" )
[1] "a is equal to pi“
```

```
norm_values <- rnorm(10, 0, 1)
below_median <- ifelse(norm_values < median(norm_values), TRUE, FALSE)
table(below_median)
below_median
FALSE  TRUE
   5     5
```

1.2. Loops

Loops/iterations

Aim: repeat a command or a set of commands several times for each value of the variable

Syntax: using **for()** followed by **{}**

for(«how to repeat on numbers/values of a variable or on vector indexes» **) {** «commands to be repeated for each value of the variable» **}**

```
for(i in 1:6){
        print(i)
}
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
```

```
counter <- 0
for (i in seq(5,8)) {
        counter <- counter + i
        cat(counter, "\n")
}
5
11
18
26
```

```
teachers <- c("Pierre","Claire")
for (t in teachers) {
        cat(t,"was one of my bioinformatics teachers \n")
}
Pierre was one of my bioinformatics teachers
Claire was one of my bioinformatics teachers
```

Loops and condition

Fonction while()

- executes the instruction as long as the condition is TRUE, stops as soon as the condition is FALSE

syntax:

```
while( "the condition to be tested“ ) {
                    “instruction”
}
```

```
i <- 0
while (i < 5) {
  i  <- i + 1
  print(i)
}
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

1.3. Vectorization

Vectorization instead of iterations

Unlike in other programming languages, in R vectorization makes some loops implicit and is computationnally more efficient

Exemple:

computing the BMI

```
weight <- c(60, 72, 57, 90, 95, 72)
size <- c(1.75, 1.8, 1.65, 1.9, 1.74, 1.91)
```

- **Using loops**

```
bmi <- numeric(length(weight))
for (i in 1:length(bmi)) {
        bmi[i] <- weight[i] / size[i]^2
}
bmi
[1] 19.59184 22.22222 20.93664 24.93075 31.37799 19.73630
```

- **Using vectorization** (to be preferred): only if vectors of same length!

```
bmi <- weight/size^2
bmi
[1] 19.59184 22.22222 20.93664 24.93075 31.37799 19.73630
```

Vectorization again!

Vectorization is more efficient to replace values in a vector than conditional executions in loops

- **Using loops and condition**

```
for (i in 1:length(bmi)){
  if (bmi[i] > 30) {
    bmi[i] <- "obesity"
  } else if ( bmi[i] < 25) {
    bmi[i] <- "normal"
  } else (bmi[i] <- "overweight")
}
bmi
[1] "normal"  "normal"  "normal"  "normal"  "obesity" "normal"
```

- **Using logical indexing of vectors** (to be preferred)

```
bmi[which(bmi > 30)] <- "obesity"
bmi[which(bmi < 25)] <- "normal"
bmi[which(bmi <= 30 & bmi >= 25 )] <- "overweight“
bmi
[1] "normal"  "normal"  "normal"  "normal"  "obesity" "normal"
```

Using apply() function

- processes matrices or dataframes
- applies a function to rows or columns of the matrix
- syntax = apply(X, margin, FUN, …) where:
  - X is a matrix or dataframe
  - margin = 1 for row, 2 for columns
  - FUN the function to apply; il can be « built-in » or « custom »
  - … additional arguments to pass to FUN (e.g. : na.rm = TRUE)

```
> m <- matrix(1:12, nr=3)
> m
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> apply(m, 1, min) # identifies the minimum of each row
[1] 1 2 3
```

➢ Easier to write and much faster than loops!

```
> min_m <- c()
> for (i in 1:nrow(m)){
      min_m <- c(min_m, min(m[i,])) }
> min_m
[1] 1 2 3
```

apply(): a family of functions

- Special cases of *apply()* when FUN=*sum* or FUN=*mean*:
  - rowSums (~ apply(X, 1, sum)
  - colSums (~ apply(X, 2, sum)
  - rowMeans (~ apply(X, 1, mean)
  - colMeans (~ apply(X, 2, mean)

- tapply(X, INDEX, FUN,…)
  - where INDEX is a factor for grouping elements of X
  - FUN is applied to groups of X defined by INDEX

- lapply(X, FUN)
  - applies FUN to elements of list X and returns a list

- sapply(X, FUN)
  - applies FUN to elements of list X but returns a vector

```
> height <-runif(20, min=155, max=180)
#simulate 20 random heights
> sex <-gl(2, 10, labels = c("Male", "Female"))
#Generate levels : factors by specifying the
pattern of their levels.
Factor w/ 2 levels "Male","Female": 1 1 1 1 1 1 1
1 1 1 ...
> tapply(height, sex, mean)
 Male   Female
166.3049 166.6483
```

2. Writing your own functions

Your own R functions?

Why?

Allows efficient, flexible and rational use of R, if you want **to redo an operation in different situations**

Properties

Similar structure as native R functions, except there is no help menu

- name
- arguments put within brackets to execute the command

```
func1 <- function ( x, y, z ) {
        …
        commands executing actions on x, y and z
        …
        return (results)
        }
```

arguments

Body of the function

results

Functions

Syntax uses two functions:

```
myFunction <- function (argument1, argument2) {
        myResult <- « what you want to do with arguments 1 and 2 »
        return(myResult)
    }
```

1. **function()** followed by **{}**
   - assign the function to the name of the function (not already implemented in R)
   - specify the names of the created function parameters as « arguments » within the (); you may specify default values with =

2. **return()** inside the **{}** of the function so that the output of the function can be saved outside of the function space; if multiple results, they must be stored in a single output in a list format

Rules in writing your own functions

1. 💣 **The name of your own function must not be a native R function,** otherwise the native R function is overwritten

exemple not to do:

```
mean <- function (x) {
return (x^2)
}
mean(c(3,4))
[1]  9 16          # it returns the squared values of 3 and 4 and not the mean of 3 and 4

rm(mean)      # if you run the above command to restore the native mean function!
```

Rules in writing your own functions

2. The function space is closed

The argument names, all the variables created inside the functions and the results exist only within the enclosed function space!

- All the required objects must be arguments of the function or they must be defined in the body of the function
- Risk to call an R object that is outside your function : by default, if the object is not defined in your function, R looks for it outside the function

```
rm(a)
func <- function (x) {
        x <- x+a
        return(x^2)
}

func(2)
Error in x + a : non-numeric argument to binary operator
```

```
a <- 2
func <- function (x) {
        x <- x+a
        return(x^2)
}

func(2)
[1] 16

#  can also be run on a vector
func(c(2,3,10))
[1]  16  25 144
```

Rules in writing your own functions

2. The function space is closed

The argument names, all the variables created inside the functions and the results exist only within the enclosed function space!

- By defaults, R uses the object defined in your function

```
a <- 2
func2 <- function (x, a) {
        x <- x+a
        return(x^2)
}

func2(2,3)        # it uses a defined as an argument in the enclosed function
[1]  25           #and not a=2 which is in your R workspace

func2(2,10)       # idem
[1] 144

func2(2)
Error in func2(2) : argument "a" is missing, with no default
# a is a required argument and is not passed to the function
# it cannot use a=2 which is in your workspace instead
```

Rules in writing your own functions

- Several arguments can be passed to your function
- They can be of different types:

numeric, logical, factors, vectors, matrices, dataframes, lists…and even functions!

- They are defined by their name or by their order

```
func3 <- function (x, a) {
        x <- x+2*a
        return(x^2)
}

func3(2,5)
[1] 144

func3(x=2, a=5)
[1] 144

func3(a=5, x=2)
[1] 144

func3(5,2)
[1] 81
```

Rules in writing your own functions

3. Do not hesitate to assign default values to the arguments

- > more robust function

```
func4 <- function (x, a=4) {
        x <- x+a
        return(x^2)
}

func4(2,5)
[1] 49

func4(2)          # since there is a default value for a, there is no need to
[1] 36            # specify it if is equal to the default value
```

Function results with return()

By default, the returned result is the last object of the function body

```
func <- function (x) {
        x ^2
}
func(2)
[1] 4
```

It is recommanded to return the result with the function return()

If more than one result has to be returned, use a list to store results and return the list

```
func <- function (x) {
        temp <- x ^2
        return(temp)
}
func(2)
[1] 4
```

```
func <- function (x) {
        temp1 <- x ^2
        temp2 <- temp1^x
        results <- list(res1=temp1, res2=temp2)
        return(results)
        }
func(2)
$res1
[1] 4

$res2
[1] 16
```

Assigning function results

- Examples

```
rm ( list=ls() )
f1 <- function( a,b ){
    Op <- a + b
}
f1(a = 6, b = 20) # the result is not shown
ls()
[1] "f1“ # the results is not saved in R
```

Add return()

```
rm ( list=ls() )
f1 <- function( a,b ){
    Op <- a + b
    return(Op)
}
f1(a = 6, b = 20)
[1] 26
ls()
[1] "f1“ # the results is not saved in R
```

Assign the result of the function

```
rm ( list=ls() )
f1 <- function( a,b ){
    Op <- a + b
    return(Op)
}
res1 <- f1(a = 6, b = 20)
ls()
[1] "f1“ “res1” # the results is saved in R
 res1
[1] 26
```

Caution: function space is closed

- Examples

```
rm(list=ls())
a <- 27
f1 <- function(a, b){
    Op <- a + b
    return(Op)
}
f1(a = 6, b = 20) # it uses the a value you
                  # assign within the function
[1] 26
ls()
[1] "a"  "f1“
 a          # a within the R session is not
[1] 27      # modified by the function
```

```
rm(list=ls())
a <- 27
f1 <- function(a=22, b){
    Op <- a + b
    return(Op)
}
f1(a, b = 20) # it uses the a value assigned
              # within the R session
[1] 47
f1(b = 20) # while here it uses the default
[1] 42      # a value specified when
            # creating the function
```

It is thus highly recommanded to use different names for your R objects within function space and within your R session to avoid any confusion !

Exemple: writing a function to compute bmi

- **How?**

arguments?

body?

results?

Exemple: writing a function to compute bmi

- **How?**

arguments?

body?

results?

- **Solution:**

```
compute.bmi <- function(w, s)
{
  z <- w/s^2
  return(z)
}
```

Writing an R program

Rules of good practice

- ✓ **Use a text editor with R syntactic coloration: eg TinnR, notepad++**
- Pairing (), {}, " " , []…
- With a clear indentation for functions, loops,…

```
all.cp.signif = NULL
acp.eig = PCA(desc.prox.sle, graph = FALSE)$eig[,1]


for (i in 1:nbrSimul){
    #random matrix
    mat.alea = NULL
    for (i in 1:dim(desc.sle.lig)[2]){
        desc.perm = sample(desc.sle.lig[,i])
        mat.alea = cbind(mat.alea, desc.perm)
    }
    colnames(mat.alea) = colnames(mat.desc)
    #compute the CP
    acp.Alea.eig = PCA(mat.alea, graph = FALSE)$eig[,1]
    mat.cp = cbind(acp.eig , acp.Alea.eig)
    vCP.signif = NULL
    for (i in 1:dim(mat.cp)[1]){
        if( mat.cp[i,1] >= mat.cp[i,2]) {
            vCP.signif = c(vCP.signif, i)
        }
    }
    all.cp.signif = c(all.cp.signif, vCP.signif)
}
countCP = table( all.cp.signif)
signifCP.sle = names(which(countCP>= nseuil))
```

```
all.cp.signif = NULL
acp.eig = PCA(desc.prox.sle, graph = FALSE)$eig[,1]


for (i in 1:nbrSimul){
    #random matrix
    mat.alea = NULL
    for (i in 1:dim(desc.sle.lig)[2]){
        desc.perm = sample(desc.sle.lig[,i])
        mat.alea = cbind(mat.alea, desc.perm)
    }
    colnames(mat.alea) = colnames(mat.desc)
    #compute the CP
    acp.Alea.eig = PCA(mat.alea, graph = FALSE)$eig[,1]
    mat.cp = cbind(acp.eig , acp.Alea.eig)
    vCP.signif = NULL
    for (i in 1:dim(mat.cp)[1]){
        if( mat.cp[i,1] >= mat.cp[i,2]) {
            vCP.signif = c(vCP.signif, i)
        }
    }
    all.cp.signif = c(all.cp.signif, vCP.signif)
}
countCP = table( all.cp.signif)
signifCP.sle = names(which(countCP>= nseuil))
```

Rules of good practice

✓ **Explicitly name the variables**:

- avoid reserved terms = prebuilt R functions or parameters -> identified by proper text editors
- strings of characters in Camel type: starts with small letters, caps for the first letter of each new word, may be separated by « . » or « _ »
- never start with a number
- no special characters
- try to use a letter defining the type of variable: v for vector, ma for matrix, d for datafarme, l for list, f for factor…

```
> theWeightofYourTeachers <- c(60,55,89,44,132)
```

↓

```
> vTeachersWeight <- c(60,55,89,44,132)
```

Rules of good practice

✓ **An understandable script for you and others**

- one command per line, leave blank lines between blocks
- using clear indentations

```
> valCount <- 0; randomVal <- rnorm(1000,3,5) ; for (i in randomVal) {if (i >=3) { valCount <- valCount +1 }}
```

```
> valCount <- 0
randomVal <- rnorm(1000,3,5)
for (i in randomVal) {
  if (i >=3)
  { valCount <- valCount +1 }
}
```

- well-organized: group paths for inputs and outputs in the same section, group variable assignations together, structure scripts with modular functions that can be reused rather than writing a single very large function
- easy to modify
- easy to understand inclunding later by you!

Rules of good practice

✓ **An understandable script for you and others**

- annotate your script by functions…but not every single command!

```
> vMeanMarkUE <- NULL            # initialize the mean of the marks by UE
> for (i in 1:dim(mM1MEGMarks)[2]){     # loop on each column of the matrix containing the marks by UE
            valMean <- mean(mM1MEGMarks)[,i]          # compute the mean of the marks per UE
            vMeanMarkUE<- c(vMeanMarkUE,valMean) # store the mean values of the ith UE in the
                                                          # vector with the means of the other UEs
}
> names(vMeanMarkUE) <- colnames(mM1MEGMarks)# assigns the names of the columns to each value
                                                          # of the vector
```

```
#computes the mean value of each UE
> vMeanMarkUE <- NULL
> for (i in 1:dim(mM1MEGMarks)[2]){
            valMean <- mean(mM1MEGMarks)[,i]
            vMeanMarkUE<- c(vMeanMarkUE,valMean)

}
>names(vMeanMarkUE) <- colnames(mM1MEGMarks)
```

Rules of good practice

✓ **An understandable script for you and others**

- Explain the usage of your script at the beginning, including the format of the input files
- Provide a command line example

✓ **For Unix usage without opening R**:

- At the fisrt line of your script, add the shebang:

  #!/usr/bin/env Rscript -> to allow executing your script

- There are 3 ways to run an R script without opening R in Unix:

```
1. R CMD BATCH [options] /path/myscript.R [path/out.file]
2. R --vanilla < myscript.R
```

or `R --vanilla --args arg1 arg2 … < /path/myscript.R` if you want to pass arguments that you can get in the R code using the fonction commandArgs()

```
3. Rscript /path/myscript.R arg1 arg2 path/out.file
```

Some further help in:
Genolini-RBonnesPartiques.pdf link on module webpage
Google's R Style Guide: https://google.github.io/styleguide/Rguide.xml

3. R packages

Packages in R

R packages:

- set of functions and sometimes of data aiming at fulfilling specific tasks or adressing sepcific problems
- uses core R functions
- may use other packages functions

  -> these other packages are called 'dependencies'

➢ use R packages rather than rewriting a function already written by someone else !

Which R packages are installed on my computer?

R program itself is installed in a « bin » folder

R packages are installed in a « library » folder…there may be different library folders

- Getting the folders, i.e libraries, where R packages are installed using .libPaths() and corresponding packages with list.files()

```
> .libPaths()
[1] "/srv/home/cvandiedonck/R/x86_64-conda_cos6-linux-gnu-library/3.6"
[2] "/srv/conda/envs/notebook/lib/R/library"


> list.files(.libPaths()[2])
 [1] "askpass"      "assertthat"   "backports"    "base"         "base64enc"
 [6] "BH"           "BiocGenerics" "BiocManager"  "BiocVersion"  "Biostrings"
 [11] "broom"        "callr"        "cellranger"   "cli"          "clipr"
 [16] "clisymbols"   "colorspace"   "compiler"     "crayon"       "curl"
 [21] "datasets"     "DBI"          "dbplyr"       "desc"         "devtools"
 [26] "diagram"      "dichromat"    "digest"       "dplyr"        "ellipsis"
 [31] "evaluate"     "fansi"        "forcats"      "fs"           "generics"
 [36] "ggplot2"      "gh"           "git2r"        "glue"         "graphics"
 [41] "grDevices"    "grid"         "gtable"       "haven"        "highr"
etc…
```

Which R packages are installed on my computer?

- Or getting the installed packages directly with the function installed.packages()

that returns a matrix containing all packages with their version and location…

```
> colnames(installed.packages())
[1] "Package" "LibPath"  "Version" "Priority" "Depends"
[6] "Imports" "LinkingTo""Suggests""Enhances" "License"
[11] "License_is_FOSS" "License_restricts_use" "OS_type" "MD5sum" "NeedsCompilation"
[16] "Built"

> head(installed.packages()[,c(1,2,3)]) # to get the most useful columns
 Package  LibPath    Version
affyio   "affyio" "/srv/home/cvandiedonck/R/x86_64-conda_cos6-linux-gnu-library/3.6" "1.56.0"
annotate "annotate" "/srv/home/cvandiedonck/R/x86_64-conda_cos6-linux-gnu-library/3.6" "1.64.0"
AnnotationDbi "AnnotationDbi" "/srv/home/cvandiedonck/R/x86_64-conda_cos6-linux-gnu-library/3.6" "1.48.0"
            # etc…
```

Loading installed R packages

- Loading an installed R package using the function **library()** and the name of the package as an argument, either with or without ". This is the recommended function to load a package.
- You might also see the function **require()** : sometimes preferred if within a function since it returns warnings instead of errors although it might be better to know the package is missing before using the function

```
> library(MASS)          # load the MASS library dedicated to statistics
> sessionInfo() # check loaded version of all loaded packages
R version 3.5.2 Patched (2019-01-02 r75949)
Platform: x86_64-w64-mingw32/x64 (64-bit)
Running under: Windows 10 x64 (build 17134)

Matrix products: default

locale:
[1] LC_COLLATE=French_France.1252  LC_CTYPE=French_France.1252
[3] LC_MONETARY=French_France.1252 LC_NUMERIC=C
[5] LC_TIME=French_France.1252

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] MASS_7.3-51.1

loaded via a namespace (and not attached):
[1] compiler_3.5.2
```

What happens if I try to load an uninstalled package?

```
library(tutu) # it returns an error
Error in library(tutu) : aucun package nommé 'tutu' n'est trouvé
require(tutu)     # it returns a warning
Le chargement a nécessité le package : tutu
Warning message:
In library(package, lib.loc = lib.loc, character.only = TRUE,  logical.return = TRUE,  :
aucun package nommé 'tutu' n'est trouvé
```

- Check and install missing package before loading using **require()** since require returns (invisibly) a logical indicating whether the required package is available

```
require(tutu) == FALSE
Le chargement a nécessité le package : tutu
[1] TRUE                    # TRUE here means require(tutu) returns the logical value FALSE
# Etc…
```

=> solution recommended when you pass your script to others

```
if (!require("RColoBrewer" , quietly = T)) {      # if the package "RColoBrewer" is already installed,
        install.packages(" RColoBrewer ")  # it will not be installed again, whereas if it was not installed, it will be
}
library(RColorBrewer)
```

Three main sources of R packages

Packages are stored in several possible repositories:

1. CRAN -> the general R repository

2. GitHub -> geeks' repository…includes tools in many programming languages

   You may use git with gitHub or gitLab also for your own scripts. It is possible with Rstudio to push and pull documentsto or from Git -> excellent for versioning control

3. Bioconductor -> a repository for bioinformatics tools = the Bioconductor project

etc…

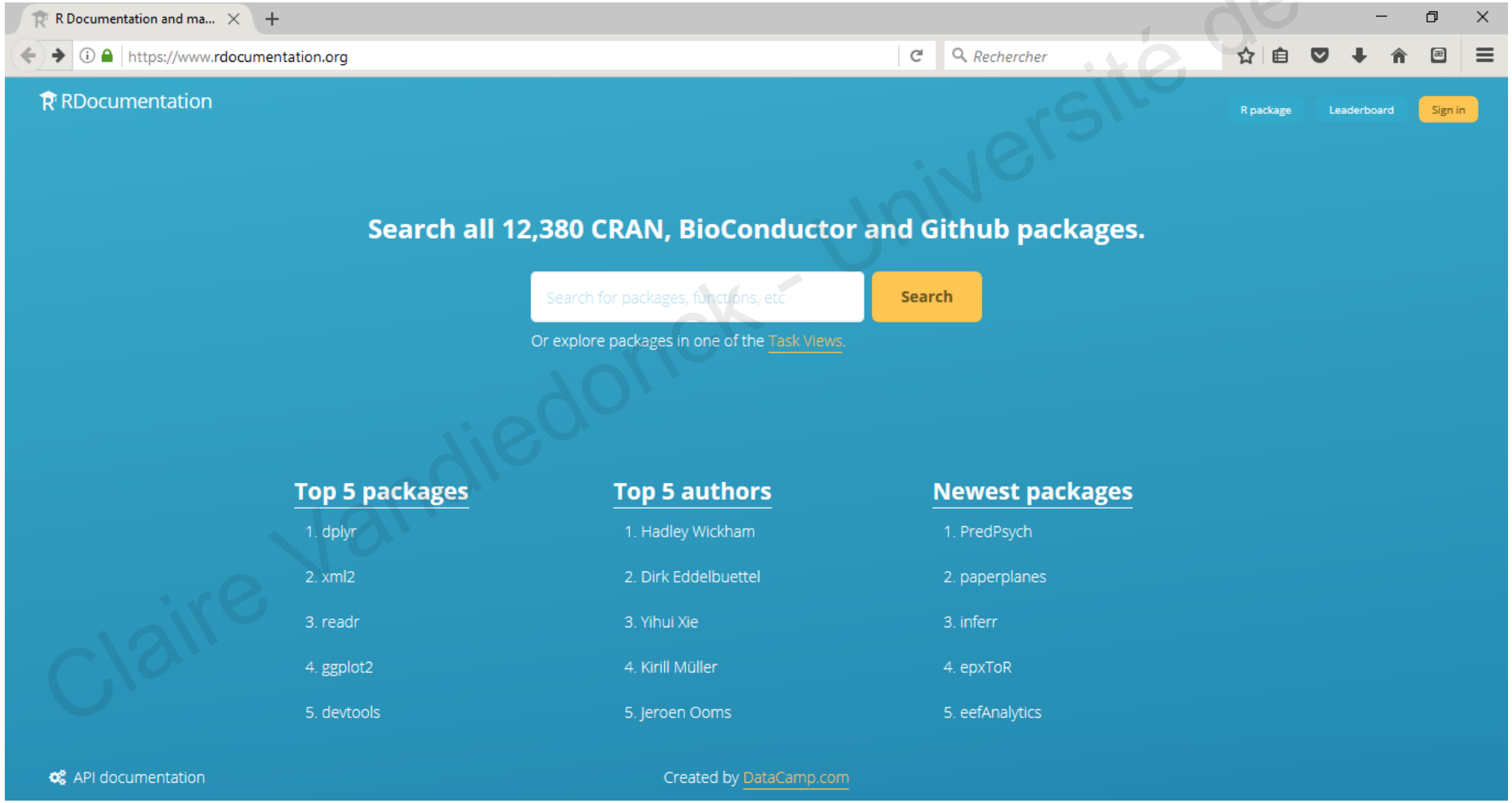
Packages are written for a specific minimal R version

Packages may require dependent packages

Functions and their corresponding packages in R

Finding the package corresponding to a given function using

https://www.rdocumentation.org/



Packages / Functions in R

12,383
indexed packages

1,767,103
indexed functions

Most downloaded packages

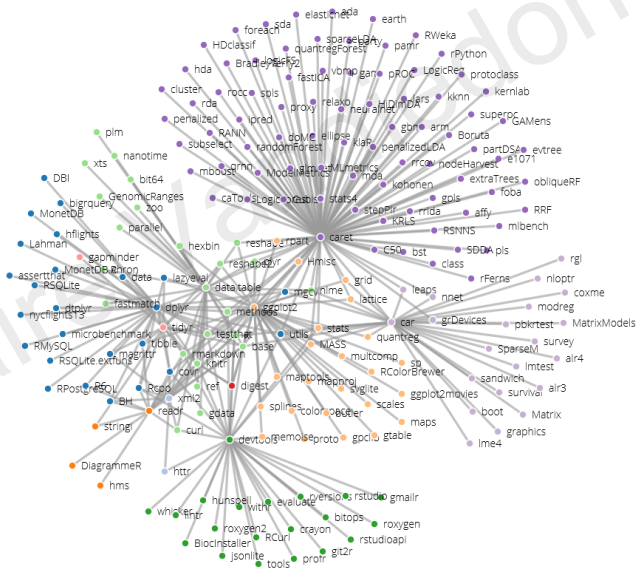
Name	Direct downloads	Indirect downloads	Total
1. dplyr	175,372	112,196	287,568
2. xml2	142,452	72,434	214,886
3. readr	131,156	55,550	186,706
4. ggplot2	112,900	326,458	439,358
5. devtools	98,949	28,320	127,269
6. data.table	96,431	138,603	235,034
7. digest	69,842	355,208	425,050
8. tidyr			,763
9. caret			397
10. car			,691

Next ›

Most active maintainers

Name	Direct downloads	Indirect downloads	Total
1. Hadley Wickham	1,027,446	2,893,015	3,920,461
2. Dirk Eddelbuettel	236,584	1,383,989	1,620,573
3. Yihui Xie	127,911	1,091,664	1,219,575
4. Kirill Müller	127,219	1,012,993	1,140,212
5. Jeroen Ooms	133,420	791,788	925,208
6. Achim Zeileis	89,741	622,394	712,135
7. Winston Chang	130,126	451,078	581,204
8. Simon Urbanek	124,815	426,403	551,218
9. JJ Allaire	85,663	416,177	501,840
10. tests Martin Maechler	79,546	317,111	396,657

Next ›



https://www.rdocumentation.org/

Installing new R packages

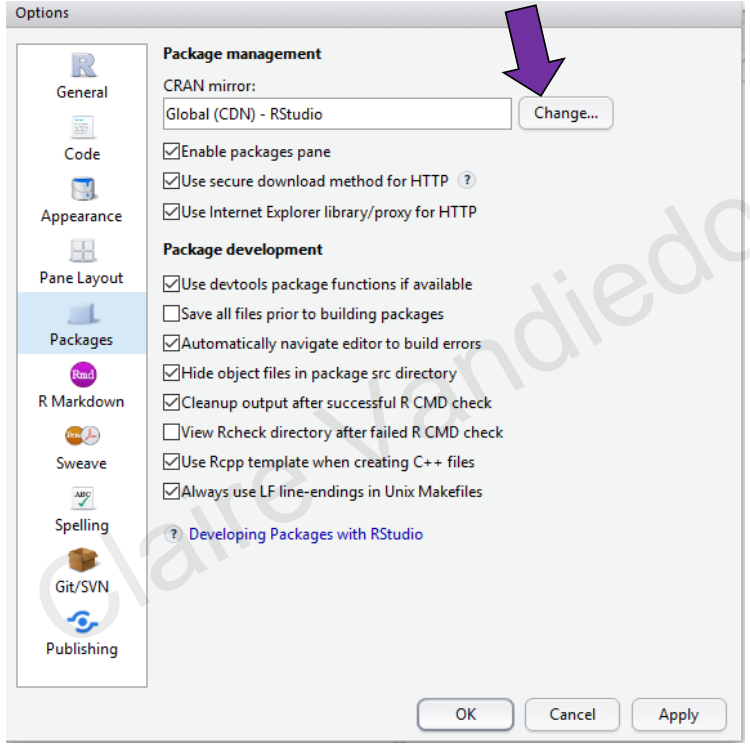
➢ Installing a package with the function install.packages()

💣 with the name of the package between “quotes”

- by default from the CRAN miror repository of your choice. Historically, France(Lyon1) or France(Lyon2) were more exhaustive than France(Paris)

  If working with Rstudio, by default Global (CDN) –Rstudio which is fine

Occasionally, you may change it by clicking in the Menu on Tools/Global options




```
> install.packages("qqman")
                # to install the qqman package
```

You may install several packages at once:

```
> install.packages(c("qqman", "MASS"))
# to install both qqman and MASS packages
```

↳ getting all possible packages from CRAN using available.packages()

```
> dim(available.packages())[1]
# currently 15159 in Lyon1 and in Rstudio
```

Installing new R packages

➢ Installing a package via the **devtools** package

If you have to regularly install packages from different sources, the devtools package simplifies this process.

It includes specific functions for each repository including:

install_local()	from a local file
install_cran()	from CRAN
install_github()	from GitHub
install_url()	from a URL
install_bioc()	from BioConductor
…	

You may also use it to install a specific older version from CRAN:

```
install_version(package, version=NULL) # by default NULL installs the last version
```

And devtools is also a package to help packages developments!

Possible issues when installing package…and solutions!

1. Packages are not available for your current R version

You will have an error message when installing the library.

To overcome this issue, download either the source **tar.gz** if you are working on Unix, or the binaries for Windows or Mac if working on these OS.

Then rerun the installation by specifying the argument « repos=NULL » and providing the path of the downloaded file

You may also specify the library folder where to install it with the argument « lib » : see next issue

```
> install.packages("/mypath/qqman/qqman_0.1.2.tar.gz" , repos=NULL, lib="mylibrarypath")
```

Possible issues when installing package…and solutions!

2. You are not allowed to install the library in the user library folder

You have not the rights to write within the folder. By default it starts with the first element returned by .libPaths(), then the second, etc…

In that case, by default R will offer you the possibility to install the library in a local user folder that it will create giving you the rights to write in

-> a question is asked to you: answer y for yes to allow this installation in your local/file/library folder

You may also want to install the package in a folder that already exists for which you have the rights to write in by specifying the argument « lib »

3. Errors occur when dependencies are not installed

The installation stops.

It often happens if the dependent packages are not available in your current R version. An error message will include the names of the packages that could not be loaded. Install them one by one as described in issues 1 and 2.

Managing R packages and their functions

To update packages to their latest version: update.packages()

To remove obsolete or useless packages: remove.packages()

Further considerations:

- If needed, you may have several R versions -> there will be several « bin » folders and their corresponding « library » folders
- If needed, you may have several versions of the same library:

Each version must be saved in a different folder. Then load the desired one with library() using its argument « lib.loc » to specify the folder of the library version

- If a function from a library does not perform exactly as wanted:

try to write your own function with its own name -> you may borrow most of the library function code: look at it by typing it without the () and adjust the function as needed *(example: treatment of NA values not always implemented…)*

Data from packages

➢ Using data from an R package:

Loading data with the function data() with the argument « package »

```
> install.packages("rpart")

> library(rpart)

> try(data(package="rpart" )) #list the data available from the package « rpart »
```

R data sets

```
Data sets in package 'rpart':

car.test.frame          Automobile Data from 'Consumer Reports' 1990
car90                   Automobile Data from 'Consumer Reports' 1990
cu.summary              Automobile Data from 'Consumer Reports' 1990
kyphosis                Data on Children who have had Corrective Spinal
                        Surgery
solder                  Soldering of Components on Printed-Circuit
                        Boards
stagec                  Stage C Prostate Cancer
```

```
> data(stagec, package="rpart") # load the dataset « stagec » corresponding to Stage C Prostate Cancer in R

> ls()

[1] "stagec"

> help(stagec, package="rpart") # to get help on the stagecdata
```

Help on data from an R package

indicates to which package the data or functions belong

R: Stage C Prostate Cancer

127.0.0.1:25859/library/rpart/html/stagec.html

stagec {rpart}

R Documentation

Stage C Prostate Cancer

Description

A set of 146 patients with stage C prostate cancer, from a study exploring the prognostic value of flow cytometry.

Usage

```
data(stagec)
```

Format

A data frame with 146 observations on the following 8 variables.

pgtime

Time to progression or last follow-up (years)

pgstat

1 = progression observed, 0 = censored

Using installed R packages

- Using a function of an installed R package without loading the package using the notation packagename::functionname()
  can be used if sporadic use of a few functions from the package instead of loading the full package

```
> gwasResults <- qqman::gwasResults # load preloaded simulated GWAS results in qqman
> qqman::qq(gwasResults) # calls the function qq from the package qqman to plot a qqplot of gwasResults
```

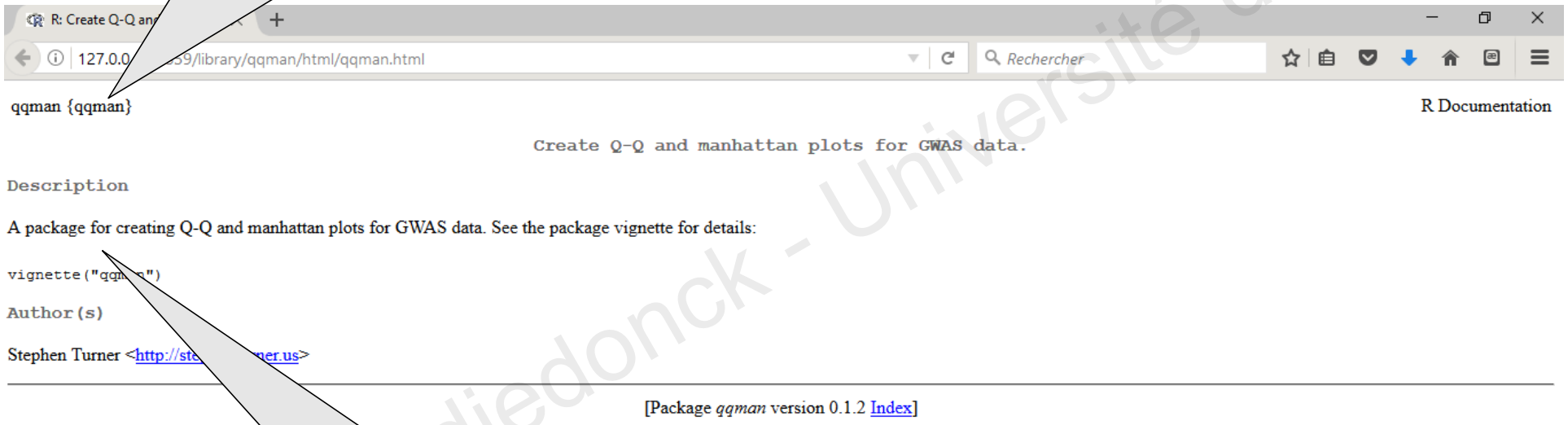
But to access to the documentation, you need to use library ()

```
> library(qqman)
> ?qqman            # only works for some packages
> qq(gwasResults$P) # same plot as previously, once the library is loaded
> manhattan(gwasResults) # manhattan plot of the results
```

Help on R packages

information on ‘qqman’ function

followed by {the package}



all packages have one or several vignettes describing their usage

= vignettes are the « user guides » ➡

vignette("package_name") directs to a webpage with package usage description

or to get all available vignettes of a package:

browseVignettes("package_name")

Help on R packages

➢ Access to help for any CRAN package:
using the function help.start ()
then select « packages »

Statistical Data Analysis

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

then choose the package :

Package Index

- Contents of C:/Users/claire/Documents/R/win-library/3.3
- Contents of C:/Program Files/R/R-3.3.3/library

Packages in C:\Users\claire\Documents\R\win-library\3.3

AnnotationDbi	Annotation Database Interface
backports	Reimplementations of Functions Introduced Since R-3.0.0
base64enc	Tools for base64 encoding
BH	Boost C++ Header Files
Biobase	Biobase: Base functions for Bioconductor
BiocGenerics	S4 generic functions for Bioconductor

Q-Q and Manhattan Plots for GWAS Data

Documentation for package 'qqman' version 0.1.4

- DESCRIPTION file.
- User guides, package vignettes and other documentation.

Help Pages

qqman-package	Create Q-Q and manhattan plots for GWAS data.
gwasResults	Simulated GWAS results
manhattan	Creates a manhattan plot
qq	Creates a Q-Q plot
qqman	Create Q-Q and manhattan plots for GWAS data.
snpsOfInterest	snpsOfInterest

Vignettes and other documentations

R packages on CRAN

Example with qqman:

CRAN - Package qqman

https://cran.r-project.org/web/packages/qqman/index.html

qqman: Q-Q and manhattan plots for GWAS data

Q-Q and manhattan plots for GWAS data

Version:	0.1.2
Depends:	R (≥ 3.0.0)
Suggests:	knitr
Published:	2014-09-25
Author:	Stephen Turner
Maintainer:	Stephen Turner <vustephen at gmail.com>
License:	GPL-3
NeedsCompilation:	no
Materials:	README
CRAN checks:	qqman results

Downloads:

Reference manual:	qqman.pdf
Vignettes:	Intro to the qqman package
Package source:	qqman_0.1.2.tar.gz
Windows binaries:	r-devel: qqman_0.1.2.zip, r-release: qqman_0.1.2.zip, r-oldrel: qqman_0.1.2.zip
OS X Mavericks binaries:	r-release: qqman_0.1.2.tgz, r-oldrel: qqman_0.1.2.tgz
Old sources:	qqman archive

Reverse dependencies:

Reverse imports:	mrMLM, pweight
Reverse suggests:	solarius

Linking:

the minimal R version

the manual describing each function within the package as when using help() or ?

the vignette describing the usage of the functions with some examples

other packages depending on this one

the package source = that may be useful for custom installation

R packages on CRAN

Example with ggplot2:

ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics

A system for 'declaratively' creating graphics, based on "The Grammar of Graphics". You provide the data, tell 'ggplot2' how to map variables to aesthetics, what graphical primitives to use, and it takes care of the details.

Version:	2.2.1
Depends:	R (≥ 3.1)
Imports:	digest, grid, gtable (≥ 0.1.1), MASS, plyr (≥ 1.7.1), reshape2, scales (≥ 0.4.1), stats, tibble, lazyeval
Suggests:	covr, ggplot2movies, hexbin, Hmisc, lattice, mapproj, maps, maptools, mgcv, multcomp, nlme, testthat (≥ 0.11.0), quantreg, knitr, rpart, rmarkdown, svglite
Enhances:	sp
Published:	2016-12-30
Author:	Hadley Wickham [aut, cre], Winston Chang [aut], RStudio [cph]
Maintainer:	Hadley Wickham <hadley at rstudio.com>
BugReports:	https://github.com/tidyverse/ggplot2/issues
License:	GPL-2 | file LICENSE
URL:	http://ggplot2.tidyverse.org, https://github.com/tidyverse/ggplot2
NeedsCompilation:	no
Citation:	ggplot2 citation info
Materials:	README NEWS
In views:	Graphics, Phylogenetics
CRAN checks:	ggplot2 results

dependencies if any

Downloads:

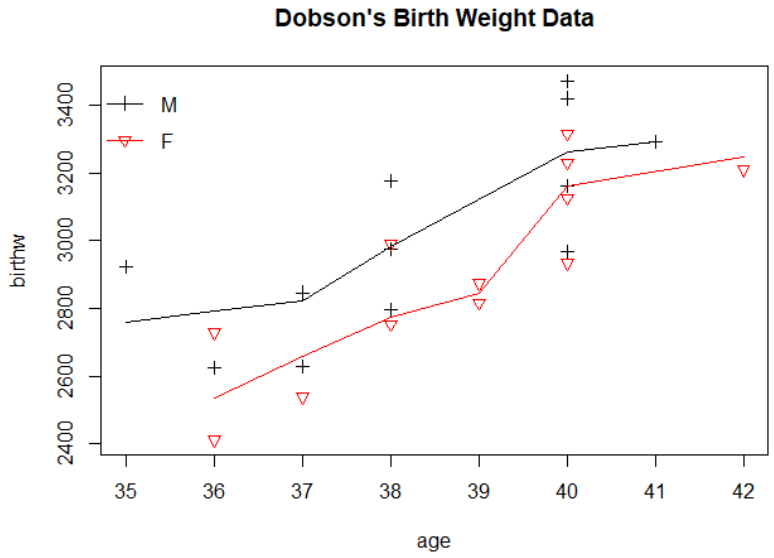
Reference manual:	ggplot2.pdf
Vignettes:	Extending ggplot2 Aesthetic specifications
Package source:	ggplot2_2.2.1.tar.gz
Windows binaries:	r-devel: ggplot2_2.2.1.zip, r-release: ggplot2_2.2.1.zip, r-oldrel: ggplot2_2.2.1.zip

Demo of R packages

- Some packages have a demo accessible with demo()

```
> demo(lm.glm, package="stats", ask=TRUE)


        demo(lm.glm)
        ---- ~~~~~~

Type  <Return>   to start :
```

```
> ### Examples from: "An Introduction to Statistical Modelling"
> ###                   By Annette Dobson
> ###
> ### == with some additions ==
>
> #  Copyright (C) 1997-2015 The R Core Team
>
> require(stats); require(graphics)

> ## Plant Weight Data (Page 9)
> ctl <- c(4.17,5.58,5.18,6.11,4.50,4.61,5.17,4.53,5.33,5.14)

> trt <- c(4.81,4.17,4.41,3.59,5.87,3.83,6.03,4.89,4.32,4.69)

> group <- gl(2,10, labels=c("Ctl","Trt"))

> weight <- c(ctl,trt)

...

> plot(age, birthw, col=as.numeric(sex), pch=3*as.numeric(sex),
+      main="Dobson's Birth Weight Data")
Hit <Return> to see next plot:
```



What is Bioconductor?

Bioconductor
OPEN SOURCE SOFTWARE FOR BIOINFORMATICS

Search:

Home | Install | Help | Developers | About

Home » BiocViews

All Packages

Bioconductor version 3.4 (Release)

Autocomplete biocViews search:

- Software (1294)
  - AssayDomain (486)
  - BiologicalQuestion (462)
  - Infrastructure (277)
  - ResearchField (341)
  - StatisticalMethod (404)
  - Technology (815)
  - WorkflowStep (678)
- AnnotationData (939)
- ExperimentData (308)

Packages found under Software:

Show All entries

Search table:

Package	Maintainer	Title
a4	Tobias Verbeke, Willem Ligtenberg	Automated Affymetrix Array Analysis Umbrella Package
a4Base	Tobias Verbeke, Willem Ligtenberg	Automated Affymetrix Array Analysis Base Package
a4Classif	Tobias Verbeke, Willem Ligtenberg	Automated Affymetrix Array Analysis Classification Package
a4Core	Tobias Verbeke, Willem Ligtenberg	Automated Affymetrix Array Analysis Core Package
a4Preproc	Tobias Verbeke, Willem Ligtenberg	Automated Affymetrix Array Analysis Preprocessing Package
a4Reporting	Tobias Verbeke, Willem Ligtenberg	Automated Affymetrix Array Analysis Reporting Package
ABAEnrichment	Steffi Grote	Gene expression enrichment in human brain regions
ABarray	Yongming Andrew Sun	Microarray QA and statistical data analysis for Applied Biosystems Genome Survey Microrarray (AB1700) gene

Many packages in version 3.8

4 main Components

Software (1649)

- AssayDomains (661)
- BiologicalQuestion (668)
- Infrastructure (360)
- ResearchFiled (728)
- StatisticalMethod (572)
- Technology (1049)
- WorkflowSetp (884)

Annotation Data (942)

- ChipManufacturer (387)
- ChipName (195)
- CustomArray (2)
- CustomDBSchema (4)
- FunctionalAnnotation (29)
- Organism (610)
- SequenceAnnotation (1)

Experiment Data (360)

- AssayDomainDara (61)
  - including CNV, CpG, expression, SNPData…
- DiseaseModel (86)
  - including CancerData (83)
- OrganismData (123)
  - including A thaliana, E Coli, D Melanogaster, S Cerevisae, H Sapien, M musculus…
- PackageTypeData (2)
- RepositoryData(85)
  - including ArrayExpres, ENCODE, GEO, 1KG…
- ReproductibleResearch (16)
- SpecimenSource (94)
  - including CelleCulture, StemCell…
- TechnologyData (230)
  - including arrays, massspec, FACS, sequencing

Workflow (23)

- AnnotationWorkflow (2)
- BasicWorkflow (4)
- EpigeneticsWorkflow (3)
- GeneExpressionWorkflow (13)
- GenomicVariantsWorkflow (13)
- ImmunoOncology Workflow (2)
- ResourceQueryingWorkflow (2)
- SingleCellWorkflow (2)

A semi-annual release

Two coexisting versions both designed to work with a specific R version

a released version

a development version

Current: Bioconductor 3.12 October 28, 2020 working with with R4.0

Previous versions archived for use with Bioconductor (R)

Release	Date	Software packages	R
3.10	October 30, 2019	1823	3.6
3.9	May 3, 2019	1741	3.6
3.8	October 31, 2018	1649	3.5
3.7	May 1, 2018	1560	3.5
3.6	October 31, 2017	1473	3.4
3.5	April 25, 2017	1383	3.4
3.4	October 18, 2016	1296	3.3
3.3	May 4, 2016	1211	3.3
3.2	October 14, 2015	1104	3.2
3.1	April 17, 2015	1024	3.2
3.0	October 14, 2014	934	3.1
2.14	April 14, 2014	824	3.1
2.13	October 15, 2013	749	3.0
2.12	April 4, 2013	671	3.0
2.11	October 3, 2012	610	2.15
2.10	April 2, 2012	554	2.15
2.9	November 1, 2011	517	2.14
2.8	April 14, 2011	466	2.13
2.7	October 18, 2010	418	2.12
2.6	April 23, 2010	389	2.11
2.5	October 28, 2009	352	2.10
2.4	April 21, 2009	320	2.9
2.3	October 22, 2008	294	2.8
2.2	May 1, 2008	260	2.7
2.1	October 8, 2007	233	2.6
2.0	April 26, 2007	214	2.5
1.9	October 4, 2006	188	2.4
1.8	April 27, 2006	172	2.3

Etc…

https://www.bioconductor.org/about/release-announcements/#release-announcements

Installing a bioconductor package

Obsolete: R versions <3.5

Installing the package -> it automatically adapts to your R version

```
# first install the Bioconductor installer package called "biocLite"
source("http://bioconductor.org/biocLite.R")
biocLite()# to install the minimum set of packages
biocLite("IRanges")# to install a specific package like "Iranges"
```

For R versions >= 3.5

Installing the package -> it automatically adapts to your R version

```
if (!requireNamespace("BiocManager"))
     install.packages("BiocManager") # to install the installer
BiocManager::install() # to install the minimum set of packages
BiocManager::install("IRanges")# to install a specific package like "Iranges"
```

Loading the package

```
library(IRanges)# load the package
library(IRanges,lib.loc=.libPaths()[1])#load the package from specific path
```

R data types in bioconductor

The main R objects:

- Vectors of logical, integer, numeric, complex, character, raw types
- Statistical concepts such as factors
- More complicated data structure: matrix, data.frame, list

In Bioconductor:

The classes are structured around an object-oriented programming system of formal classes and methods S4 proposed by John Chambers

Why?

Lists are contrived and have limited functionalities

Real object-oriented standards are better, especially with large complex biological data objects

For easier coding, to secure reliable package inter-operability

Methods are defined both generically to specify the basic contract and behaviour and specifically to cater for objects of particular classes

The S4 class system

A class provides a software abstraction of a real world object. It reflects how we think about certain objects and what information they should contain

Classes are defined to have specified structures in terms of slots. These are like the components in a list. They contain the relevant data.

An object is an instance of a class

A class defines the structure and inheritance relationships of objects

Implemented in the *methods* R package

```
> sessionInfo()
R version 3.0.0 (2013-04-03)
…
attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

> ls("package:methods")
  [1] "addNextMethod" "allGenerics" "allNames"
  [4] "Arith" "as" "as<-"
  [7] "asMethodDefinition" "assignClassDef" "assignMethodsMetaData"
Etc…
```

Accessing slots

The slots in an object that can be accessed in several ways

- Example:

The class for microarray expression data is ExpressionSet

The slot in an Expression Set object containing the matrix of expression values is named exprs

If upp1Eset is an ExpressionSet object, the exprs slot can be accessed by any one of the following:

```
upp1Eset@exprs
exprs(upp1Eset)
slot(upp1Eset, "exprs")
```

slotNames(upp1eset) lists all the slots in this object

Exemple of an S4 object

```
library(IRanges)
mydata <- IRanges(start=c(101, 25), end=c(110, 80))
mydata
IRanges of length 2
   start end width
[1]   101 110    10
[2]    25  80    56

str(mydata)
Formal class 'IRanges' [package "IRanges"] with 6 slots
  ..@ start          : int [1:2] 101 25
  ..@ width          : int [1:2] 10 56
  ..@ NAMES          : NULL
  ..@ elementType    : chr "integer"
  ..@ elementMetadata: NULL
  ..@ metadata       : list()
```

The object-oriented method

A method is a function that performs an action on an object

Methods define how a particular function should behave depending on the class of its arguments

Methods allow computations to be adapted to particular data types, i.e. classes

Associated to any object is a list of the methods that can be applied to it

The classes and methods implemented in BioC packages can be hard to document, especially when the class hierarchy is complicated

For the end-user: it's mostly transparent. But when something goes wrong, error messages issued by the S4 class system can be hard to understand. Also it can be hard to find the documentation for a specific method

For going further on S4 objects and related methods:
http://www.duclert.org/r-divers/classes-S4-R.php

Some widely-used R functions and packages in genomics

For genomic intervals and annotations

- the rle() function: groups of consecutive values and counts their numbers
- IRanges: to store, manipulate and aggregate intervals on sequences
- GenomicRanges: serves as the foundation for representing genomic locations within the Bioconductor project
- biomaRt: to get genomic annotations tables and cross them
- Rctracklayer: to export/import/manipulate genome browser tracks in different formats

For genetic association studies:

- qqman: to perform QCs on GWAS data (manhattan and qqplots)

For microarray analyses:

- affy: to read affymetrix array data, to perform microarray normalisations
- limma: to perform differential expression analysis on microarrays (the gold-standard method) and now on RNASeq data

For NGS data:

- Rsamtools: as samtools in Unix to handel sam/bam files
- edgeR: normalization and differential expression of RNASeq data
- DESeq2: normalization and differential expression of RNASeq data

4. Figures with R base

Three-level graph functions

1. Primary graph functions = high-level graphical functions

to plot the most principal graphs in R

2. Secondary graph functions = low-level plotting commands

to complement an existing plot

3. Graphical parameters

to modify the presentation of the plots

- either as options within the above two kind of graphic functions
- or permanently with the par() function before plotting the graph

The primary graph functions

Examples of the most frequently used graphs in R

plot()	to plot points at given coordinates (x) or (x,y) ordered on the axes
pie()	to plot a circular pie chart of a qualitative variable
barplot()	to plot occurrences/frequencies of a qualitative variable
hist()	to plot the distribution of a quantitative variable as an histogram
boxplot()	to plot the distribution of a quantitative variable as a boxplot
stripchart()	to plot the values of a quantitative variable along an axis
pairs()	to draw pair-wise plots between the columns of a matrix

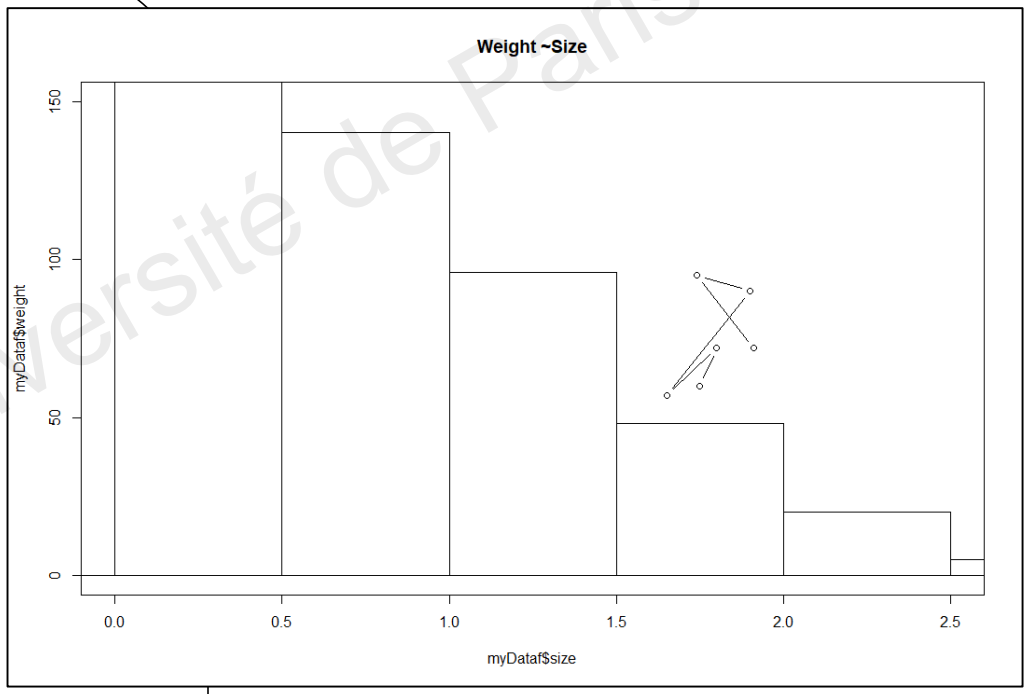
…

Some arguments/options are identical for several graph functions

eg. "main"	to specify the title
"xlim", "ylim"	to specify the limits of axes
"type"	to specify the type of plots ("p" for points, "l" for lines, "n" for none…)
"add"	to supperpose to the previous plot if TRUE

Example of primary graph functions

```
> plot(myDataf$weight~myDataf$size)
> plot(myDataf$weight~myDataf$size,
main="Weight ~Size")        # to add a title

> plot(myDataf$weight~myDataf$size,
    main="Weight ~Size", type="l")  # to draw a line

> plot(myDataf$weight~myDataf$size,
    main="Weight ~Size", type="b")  # to connect a line between points

> plot(myDataf$weight~myDataf$size,
    main="Weight ~Size", type="b",
    xlim= c(0,2.5), ylim=c(0,150))     # to specify axis limits

> hist(a,breaks=20, add=T)
        # the add argument allows to draw the new plot
        # above the previously called plot
        # note: add does not work for plot, use points(), cf. secondary functions)
```



The secondary graph functions

Examples of the most frequently used low-level plotting functions in R

- complement an existing plot

eg.	points()	to add new points
	lines()	to add points connected to a line
	abline()	to add a new line of given slop and interecpt
	mtext()	to add text in a margin
	axis()	to add axis with a given layout
	legend()	to add a legend
	title()	to add a global title

…

Graphical parameters

Examples of important parameters

« mar » size of margins

« mfrow and mfcol » to specify the display of plots (number of lines and columns) within the graph window

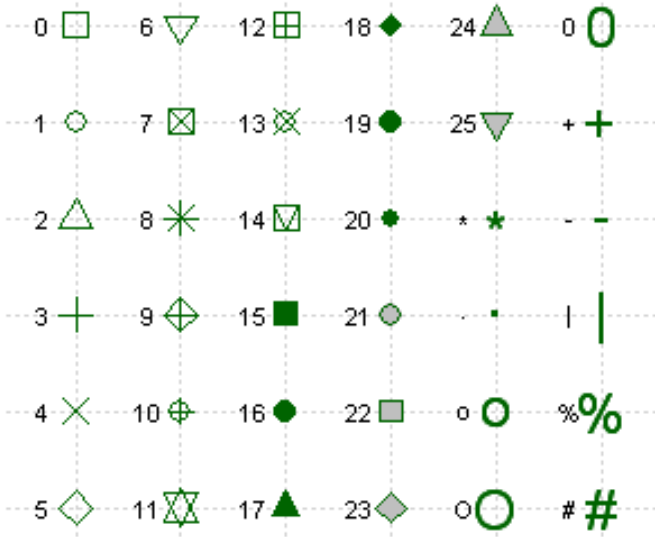
« cex » size of texts and symbols

similarly, specific cex parameters for axis: cex.axis, for labels: cex.lab…

« pch » type of symbols

E. Paradis

QUICK R



« bg » background color (by default = "tranparent", or "white" in Rstudio)

« col » color of symbols, texts…

similarly specific col parameters for axis: col.axis, for labels: col.lab

Example of secondary graph functions and parameters

```
> plot(myDataf$weight~myDataf$size, main="Weight ~Size",
   xlim= c(-3,3), ylim=c(0,200), type="n", xlab="size", ylab="weight")
        # draw the frame of the plot but not the data with type="n«

> points(myDataf$weight[1:2]~myDataf$size[1:2], pch=6, col="blue")
        # points() allows to add the data to the existing plot
        # it is usefull to filter data to display points on different manners

> points(myDataf[3:6,"weight"]~myDataf$size[3:6], type="b", pch=23, col="magenta",
        bg="cyan", cex=2)
        # here for the last 4 points, I change the type and its color and background

> points(seq(0,2.5, 0.5), c(1, 10, 25, 50, 125, 150), type="l")
        # using type="l", I can aslo draw a line through the points
> lines(-seq(0,2.5, 0.5), c(1, 10, 25, 50, 125, 150), lty= "dotdash", col="blue", lwd=3)
        # lines() also draws a line. You can specify its type with lty and width with lwd
> abline(0, -50, lty=3, col="red")
> abline(v=0, lty=2, col="green")
        # abline is a further function to draw lines with a given slope, vertical or horizontal
```

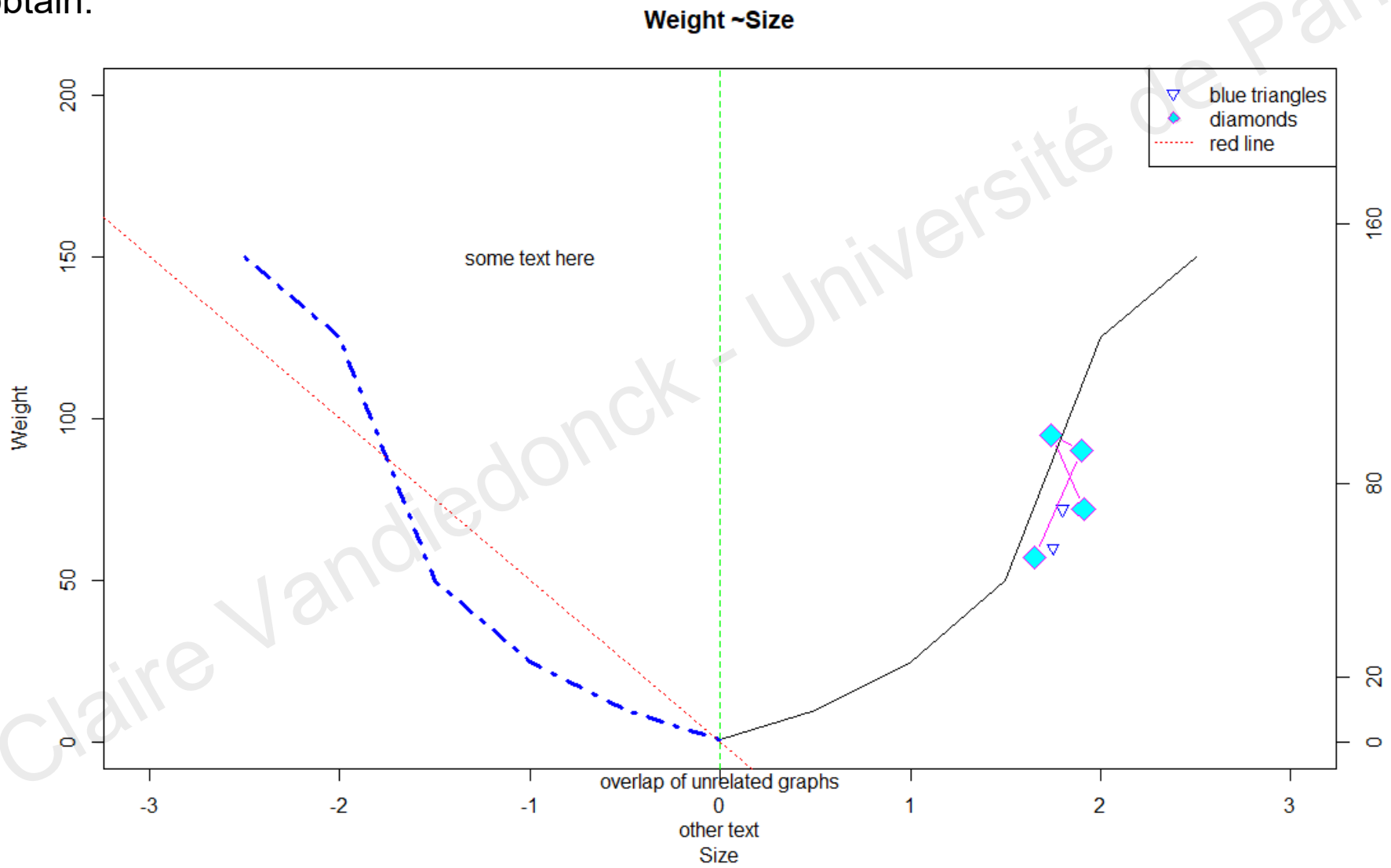
Example of secondary graph functions and parameters

```
> mtext("overlap of unrelated graphs", side=1)
> mtext("other text", side=1, line=2)
            #mtext() is used to write text in the margins of the plot
> text(-1, 150, "some text here")
            # while the function text adds text at the given coordinates
> axis(side=4, labels=c(0, 20, 80, 160),  at=c(0, 20, 80, 160), tick=T)
             # axis is another way to drawe x, y axis but also an additional axis
            # on the right side
```

```
> legend("topright", c("blue triangles", "black  dots", "redline"),
             col=c("blue", "black", "red"),  pch=c(6, 1, NA), lty=c(0,0,3))
#  you specify within vectors the text of the different elements, their color, etc…
```

Example of secondary graph functions and parameters

- What you obtain:



Colors in R

Display current colors with palette()

Specify colors by their index, "name", "hexadecimal" or "rgb" values

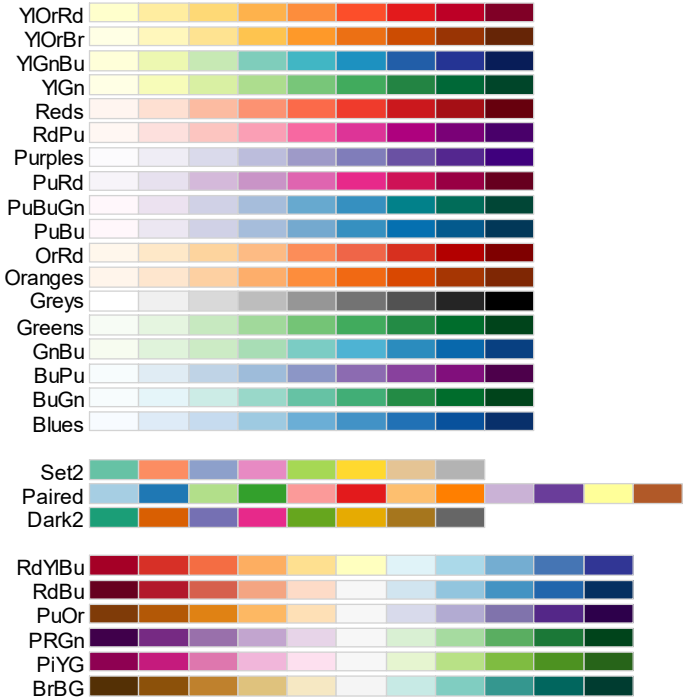
R Chart color at https://web.archive.org/web/20121202022815/http://research.stowers-institute.org/efg/R/Color/Chart/ColorChart.pdf and on the module webpage

white	#FFFFFF	255	255	255
aliceblue	#F0F8FF	240	248	255
antiquewhite	#FAEBD7	250	235	215
antiquewhite1	#FFEFDB	255	239	219
antiquewhite2	#EEDFCC	238	223	204
antiquewhite3	#CDC0B0	205	192	176
antiquewhite4	#8B8378	139	131	120
aquamarine	#7FFFD4	127	255	212
aquamarine1	#7FFFD4	127	255	212
aquamarine2	#76EEC6	118	238	198
aquamarine3	#66CDAA	102	205	170
aquamarine4	#458B74	69	139	116
azure	#F0FFFF	240	255	255
azure1	#F0FFFF	240	255	255
azure2	#E0EEEE	224	238	238
azure3	#C1CDCD	193	205	205
azure4	#838B8B	131	139	139
beige	#F5F5DC	245	245	220
bisque	#FFE4C4	255	228	196

Etc…

```
#install.packages("RColorBrewer")
library(RColorBrewer)
display.brewer.all(colorblindFriendly=TRUE)
```

↳Very useful package « **RColorBrewer** » with associated palettes including for colorblind



Tip: to find the color reference from an electronic document, use InstantEye Dropper

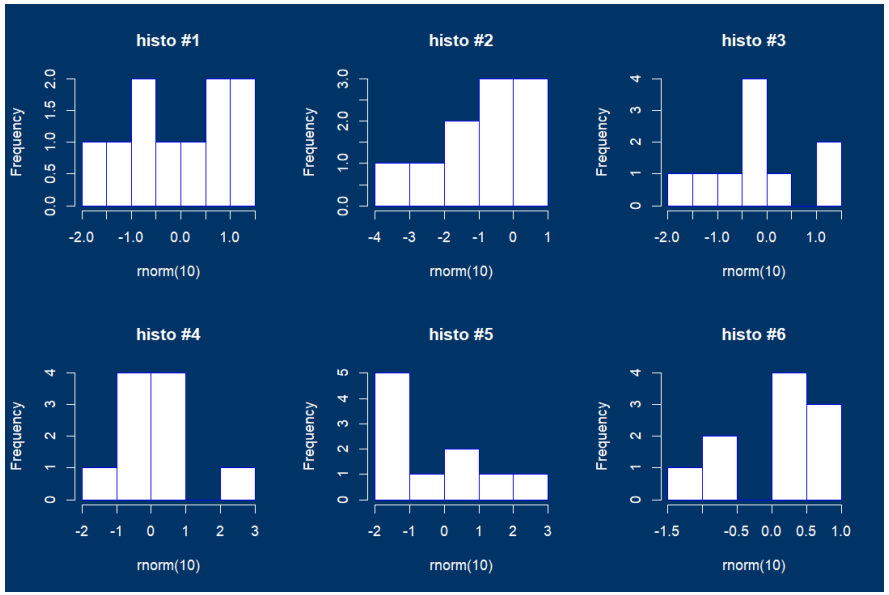
Graphical parameters with par()

```
> par()                    # displays the current parameters in a list!
> par()$cex                # displays the current cex parameter

> opar <- par()            # to save the current parameters VERY IMPORTANT
> par(bg=rgb(0, 51, 102, max=255), col="white", mfrow=c(2,3), cex=1.1)
                 # new graphs will have a background of the same color as my slide titles
                 # and 6 plots will be plotted on the same graph window (2 rows, 3 columns)
                 # and the size of the text will be 10% larger than by default
```

Then do your plots…

```
> hist(rnorm(10), col="white", border="blue“,
    col.axis="white", fg="white”, col.lab="white",
    col.main="white")
```

…and 5 other plots



and finally restore the initial parameters

```
> par(opar)     # to restore default parameters
```

Saving your figures

Save figures in different formats with the appropriate function (by default in the working directory)

bmp()	.bmp
jpeg()	.jpeg
tiff()	.tiff
png()	.png
postscript()	.eps

```
> png("MyPlot.png")
> hist(rnorm(10000, 0, 1), freq=F)
> dev.off()
```

Three steps

1. Type the function with the name of the saved file as an argument with the correct extension
   Other arguments like « width » and « height » to specify dimensions
2. Do your plot -> it is directed to the file and not displayed in the graphical window within R
3. Close the graph by typing the following function

   dev.off()

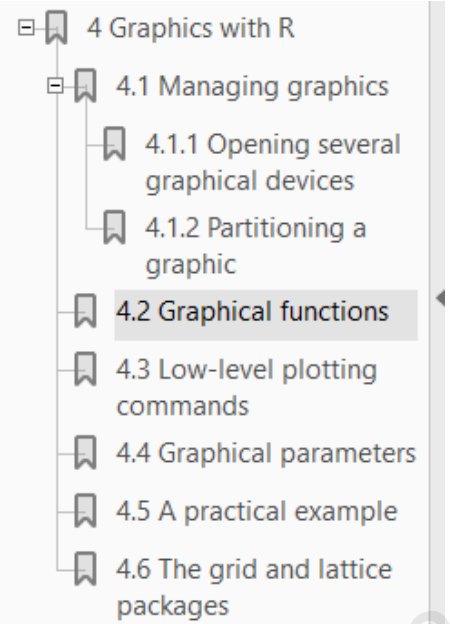
Specific case for pdf() to save graphs in a .pdf

- you may save each figure at a time
- or all several (all) figures generated with all the command lines entered between pdf() and dev.off()

Getting help of figures

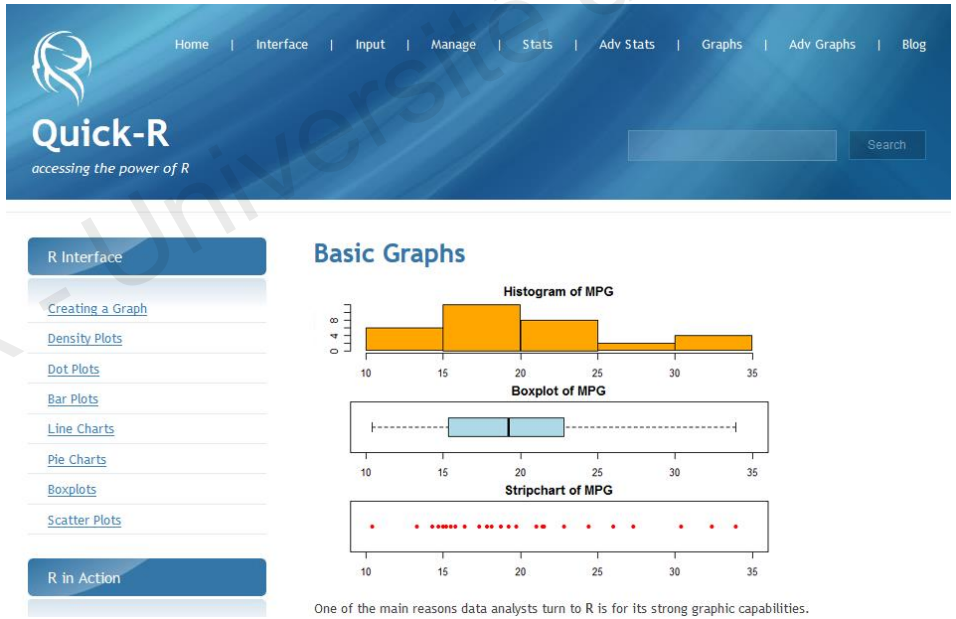
R for beginners E. Paradis

Chapter 4 for graphs quite exhaustive
in moodle in French and English



QUICK R:

http://www.statmethods.net/
basic and avdanced graphs
with main parameters



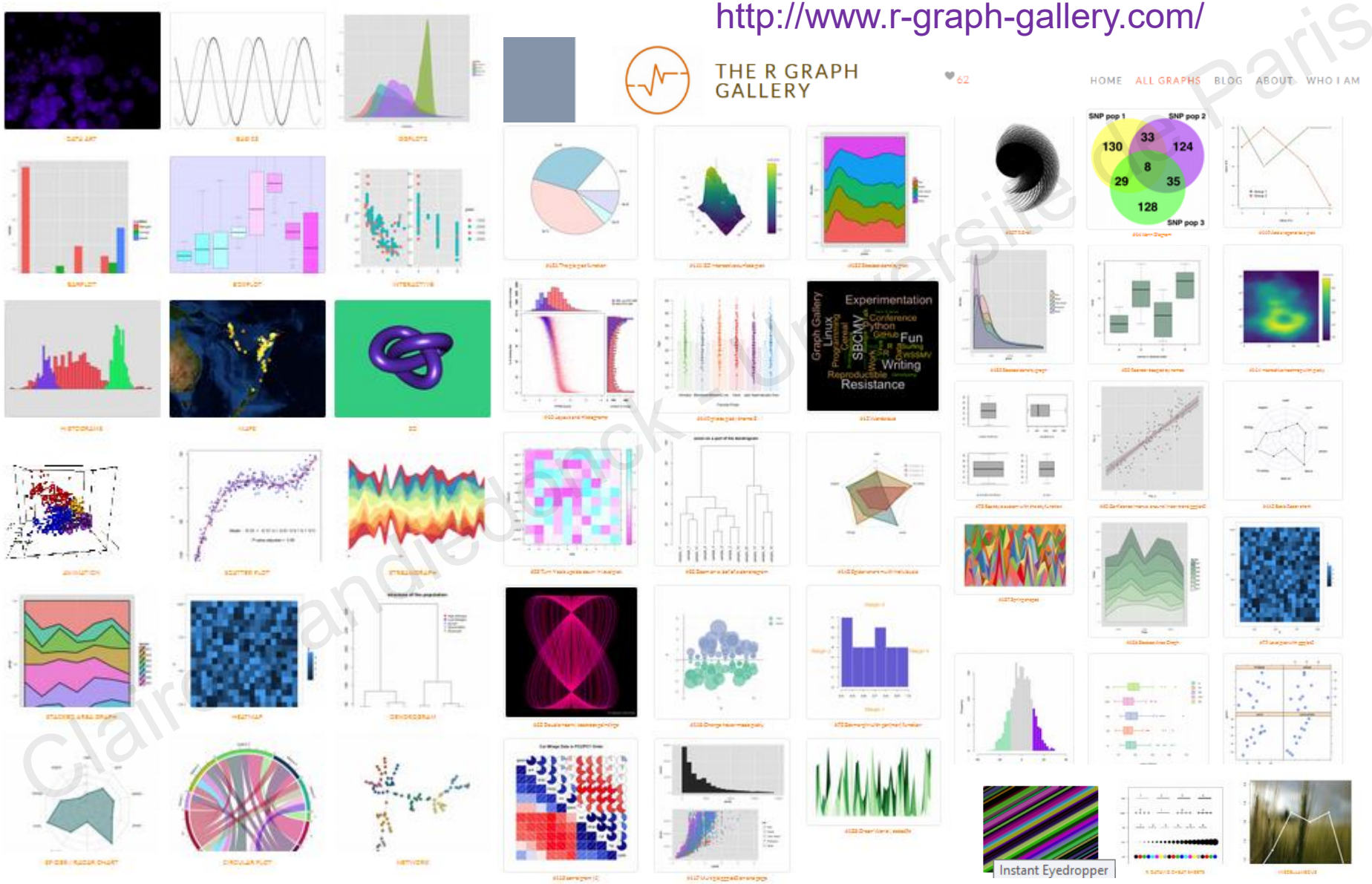
R gallery: http://www.r-graph-gallery.com/all-graphs/ for specific kinds of graphs

STHDA: http://www.sthda.com/english/

And some blogs for specific questions
https://www.stat.ubc.ca/~jenny/STAT545A/block14_colors.html#using-colors-in-r
https://danieljhocking.wordpress.com/2013/03/12/high-resolution-figures-in-r/

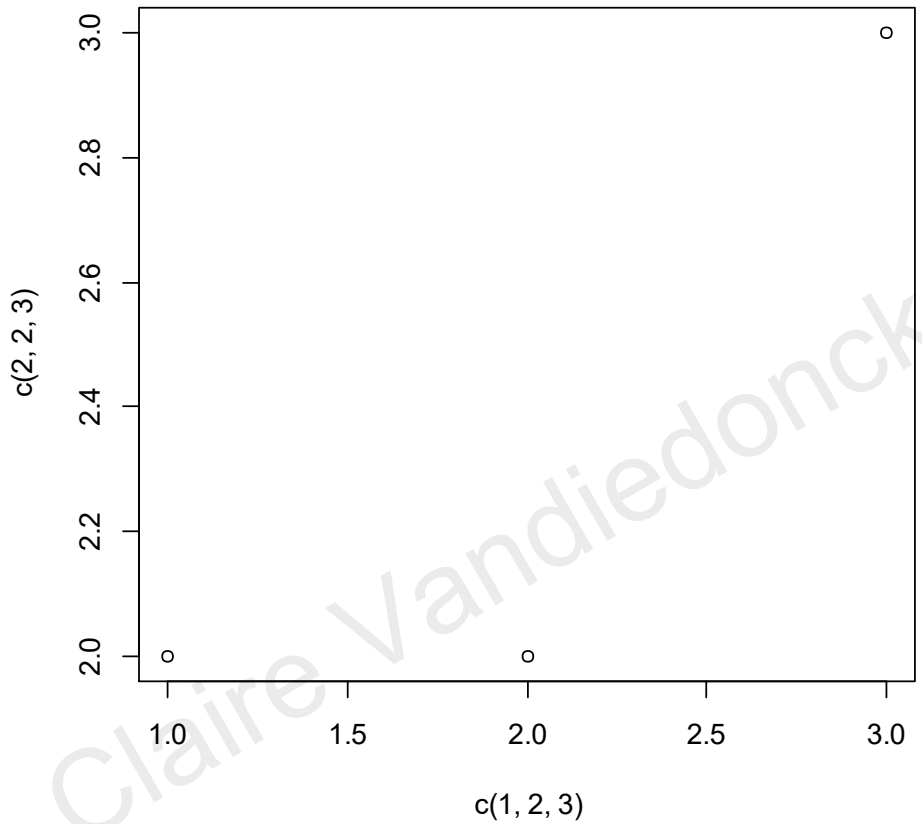
Endless kinds of graphs with R

http://www.r-graph-gallery.com/

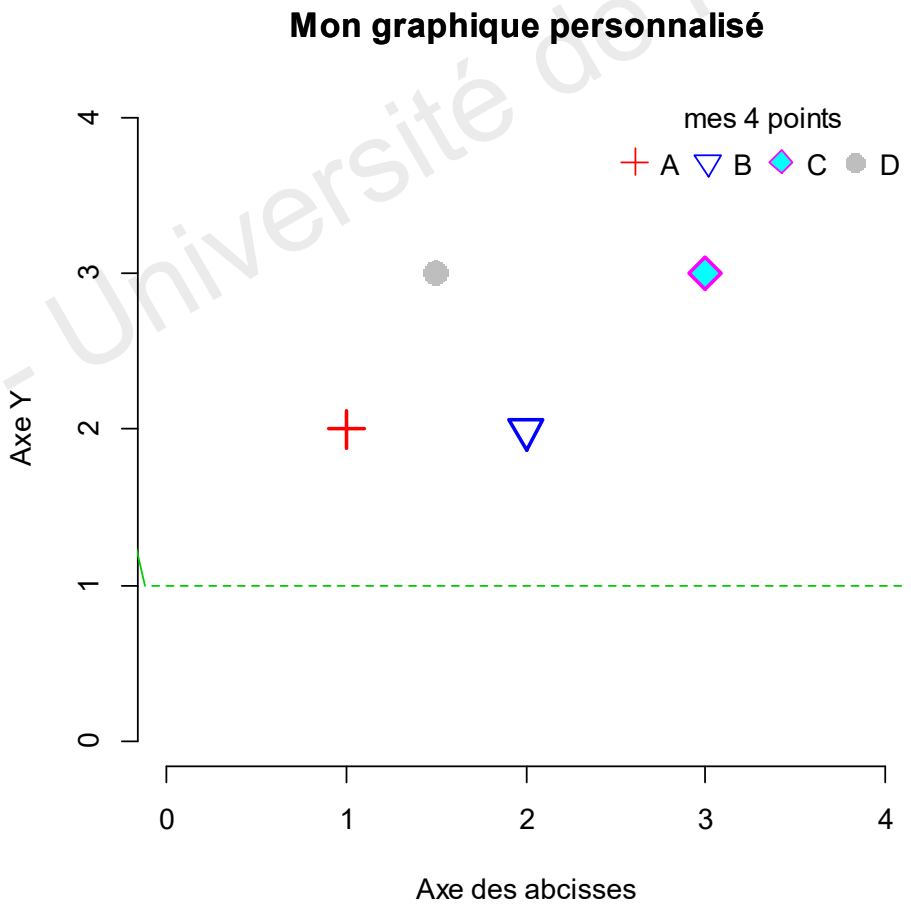


Practical: try to generate this custom graph

before



after



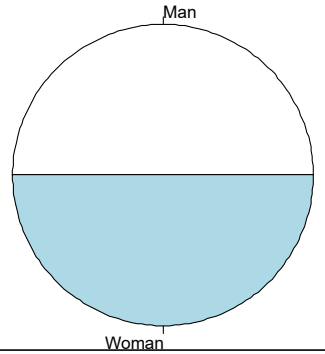
Some graph examples for qualitative variables

Cross-tabulations of occurrences using table()

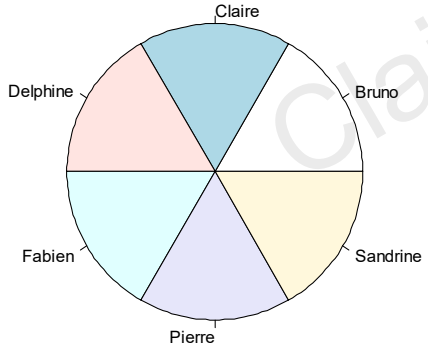
```
> table(myDataf$sex)
Man  Woman
  3      3
> table(myDataf$sex, row.names(myDataf)) # can be done on two or more variables
       Bruno Claire Delphine Fabien Pierre Sandrine
Man        1      0        0      1      1        0
Woman      0      1        1      0      0        1
```

Display proportions using pie() or barplot()

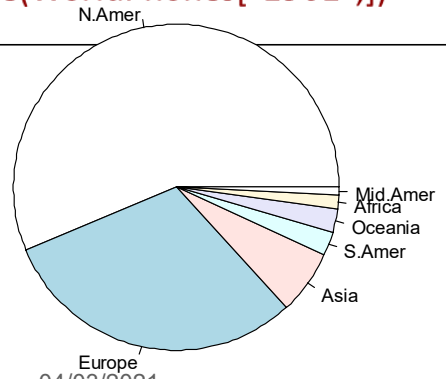
```
> pie(table(myDataf$sex))
```




```
> pie(table(row.names(myDataf)))
```

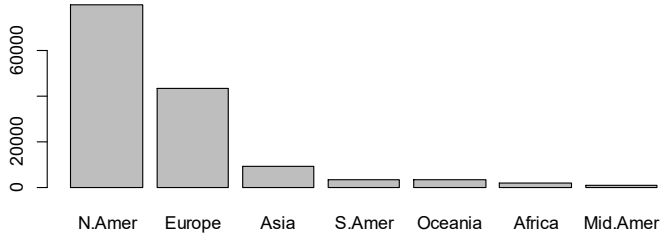



```
> data(WorldPhones)
> tail(WorldPhones)
     N.Amer Europe Asia S.Amer Oceania Africa Mid.Amer
1956  60423  29990 4708   2568    2366   1411      733
1957  64721  32510 5230   2695    2526   1546      773
1958  68484  35218 6662   2845    2691   1663      836
1959  71799  37598 6856   3000    2868   1769      911
1960  76036  40341 8220   3145    3054   1905     1008
196179831  43173 9053   3338    3224   2005     1076
> pie(WorldPhones["1961",])
```

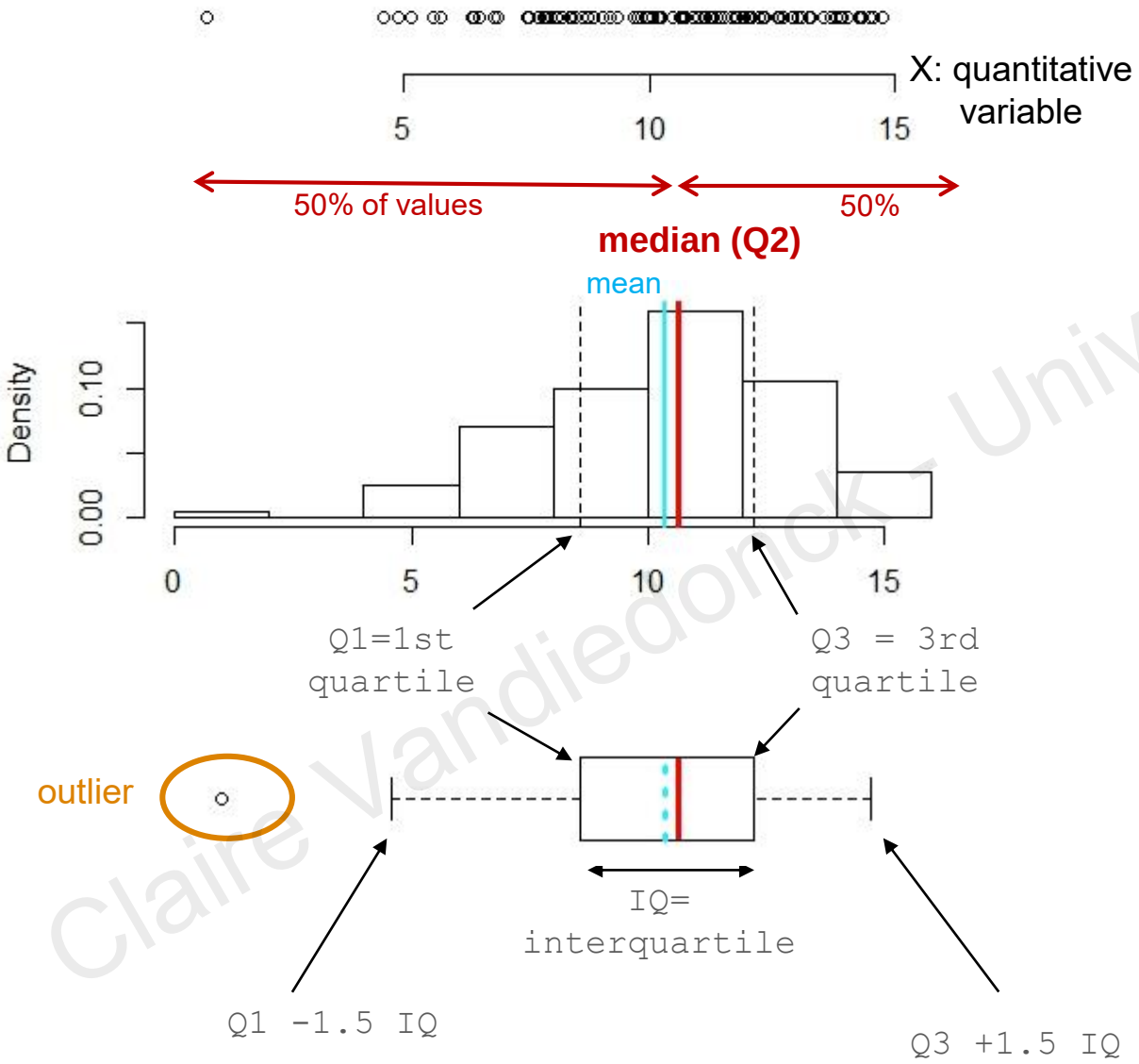


AVOID PIE CHART
USE BARPLOTS

```
> barplot(WorldPhones["1961",])
```



Plotting distribution for continuous variables



stripchart()

« vertical » =F by default

hist()

« freq » = T by default to display counts while = F to display density

boxplot()

does not display the mean but the median

« range »=1.5 by default = k*IQ distance of whisker edges

if 0: up to min and max, no outliers

« outlines »=T to display outliers by default, F to hide outliers

Some graph examples for continuous variables

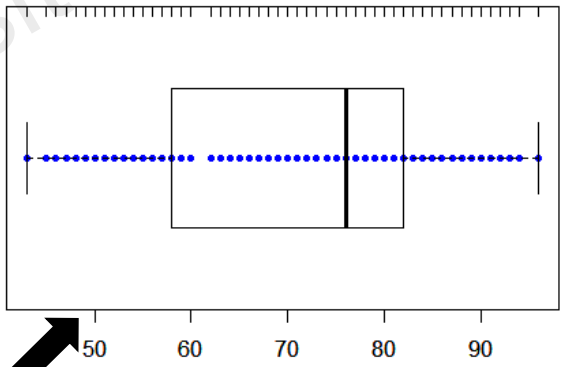
Example: the old faithful geyser
in Yellowstone National Park, USA

```
> data(faithful)
> str(faithful)
'data.frame':   272 obs. of  2 variables:
 $ eruptions: num  3.6 1.8 3.33 2.28 4.53 ...
 $ waiting  : num  79 54 74 62 85 55 88 85 51 85 ...
> ?faithful
```

Format

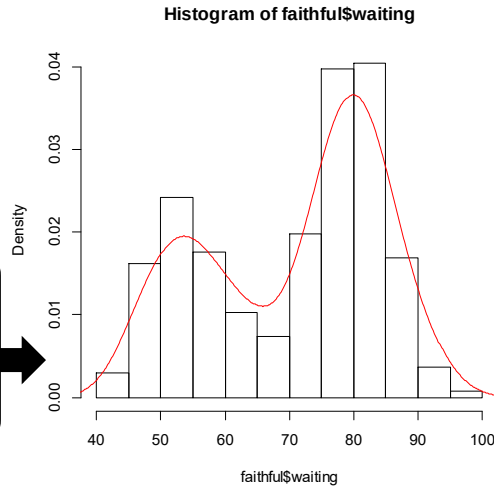
A data frame with 272 observations on 2 variables.

[,1] eruptions numeric Eruption time in mins
[,2] waiting numeric Waiting time to next eruption (in mins)




```
> stripchart(faithful$waiting, col="blue", pch=20) # col is a parameter used inside
> boxplot(faithful$waiting, horizontal=T, add=T) # add=T to superpose graphs
> rug(faithful$waiting, side=3) # example of secondary function
```

```
> hist(faithful$waiting, freq=F)      # freq=T to display counts,
                                      # freq =F to display frequencies!
> lines(density(faithful$waiting),col="red")
```



Why not using barplots for quantitative data?

A MUST READ THREAD:

https://twitter.com/T_Weissgerber/status/1040576802979233793

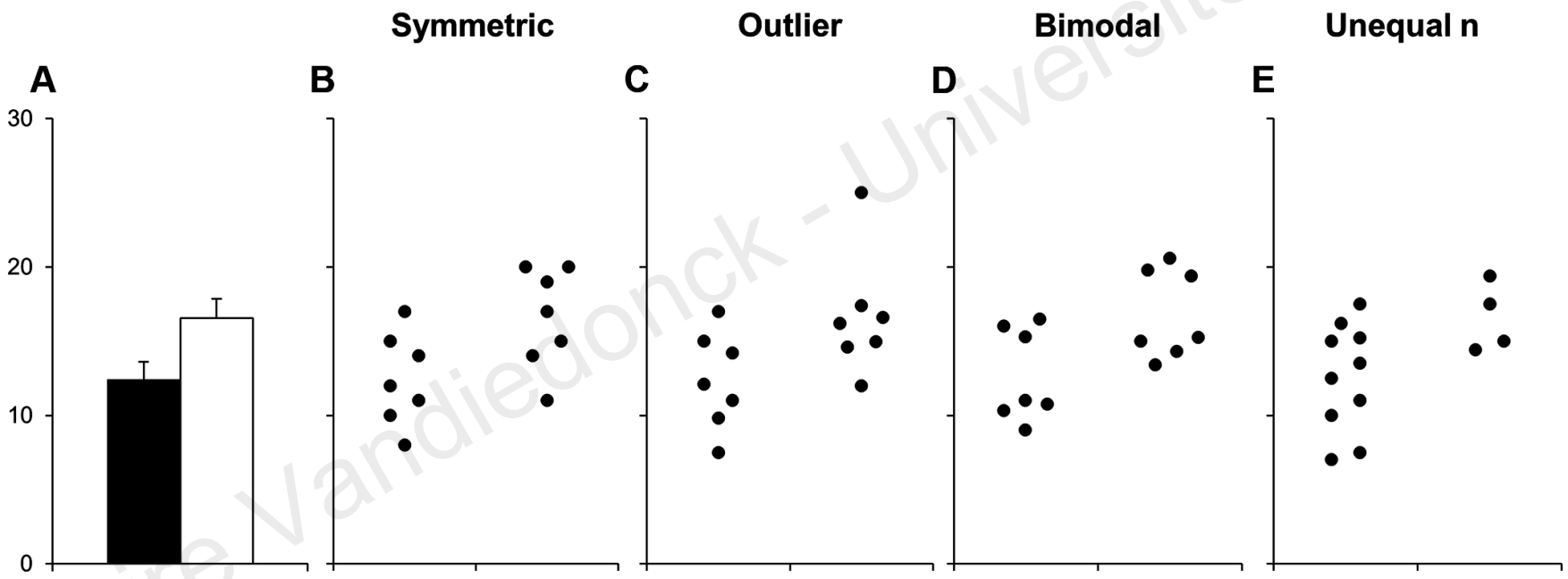
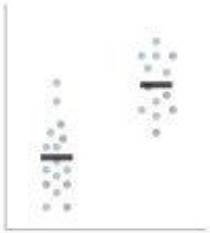
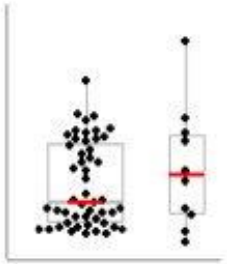
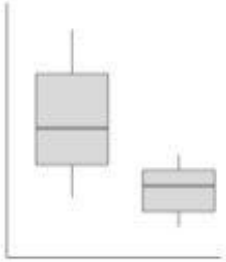
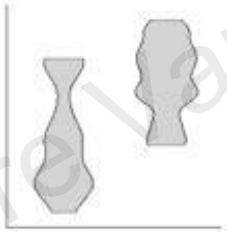
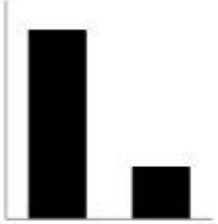


Figure Types	Example	Type of Variable	What the Plot Shows	Sample Size	Data Distribution	Best Practices
Dot plot		Continuous	Individual data points & mean or median line Other summary statistics (i.e. error bars) can be added for larger samples	Very small OR small; can also be useful with medium samples	Sample size is too small to determine data distribution OR Any data distribution	• Make all data points visible - use symmetric jittering • Many groups: Increase white space between groups, emphasize summary statistics & de-emphasize points • Only add error bars if the sample size is large enough to avoid creating a false sense of certainty • Avoid "histograms with dots"
Dot plot with box plot or violin plot		Continuous	Combination of dot plot & box plot or violin plot (see descriptions above and below)	Medium	Any	• Make all data points visible (symmetric jittering) • Smaller n: Emphasize data points and de-emphasize box plot, delete box plot and show only median line for groups with very small n • Larger n: Emphasize box plot and de-emphasize points
Box plot		Continuous	Horizontal lines on box: 75th, 50th (median) and 25th percentile Whiskers: varies; often most extreme data points that are not outliers Dots above or below whiskers: outliers	Large	Do not use for bimodal data	• List sample size below group name on x-axis • Specify what whiskers represent in legend
Violin plot		Continuous	Gives an estimated outline of the data distribution. The precision of the outline increases with increasing sample size.	Large	Any	• List sample size below group name on x-axis • The violin plot should not include biologically impossible values
Bar graph		Counts or proportions	Bar height shows the value of the count or proportion	Any	Any	• **Do not use for continuous data**

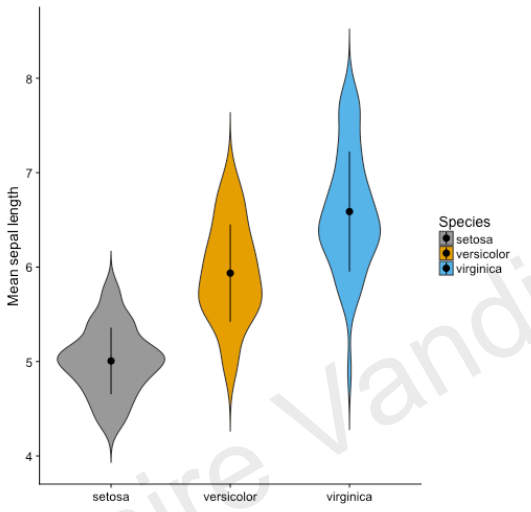
Source: Tracey Weissgerber
Twitter: @T_Weissgerber

Alternative to barplots

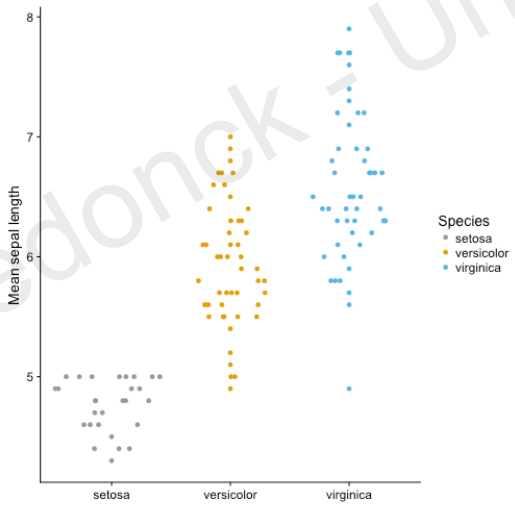
https://audhalbritter.com/alternatives-to-barplots/

https://cran.r-project.org/web/packages/sinaplot/vignettes/SinaPlot.html

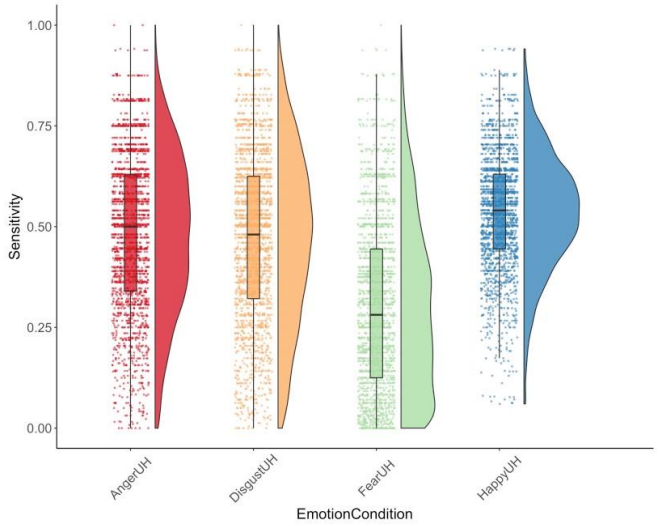
https://micahallen.org/2018/03/15/introducing-raincloud-plots/



violon plots



sina plots



raincloud plots