Getting starting with **R**

Alexandre Courtiol

Leibniz Institute of Zoo and Wildlife Research

June 2018

Who am I?

- evolutionary biologist / statistician
- studies in France (Montpellier), postdoc in the UK (Sheffield)
- researcher at Leibniz IZW / lecturer at Freie University
- experience with **R**:
  - 2003 –: study **R** (still ongoing)
  - 2008 –: use **R** most days
  - 2010 –: teach **R**
  - 2016 –: develop **R** packages

Getting started with **R**

Getting started with R

R in brief

R is a programming language and software environment for statistical computing & graphics.

Key points about **R**:

- free for all
- open source (explore: `https://github.com/wch/r-source`)

R in brief

R is a programming language and software environment for statistical computing & graphics.

Key points about **R**:

- free for all
- open source (explore: `https://github.com/wch/r-source`)
- polyvalent:
  - from laptop to most advanced supercomputers
  - local or remote
  - Windows, MacOS, linux or many other Unix-based systems

R in brief

R is a programming language and software environment for statistical computing & graphics.

Key points about **R**:

- free for all
- open source (explore: `https://github.com/wch/r-source`)
- polyvalent:
  - from laptop to most advanced supercomputers
  - local or remote
  - Windows, MacOS, linux or many other Unix-based systems
- rich (tons of **R** packages out there)
- cutting edge (check updates for today: `http://dirk.eddelbuettel.com/cranberries/cran/updated/`)

R in brief

R is a programming language and software environment for statistical computing & graphics.

Key points about **R**:

- free for all
- open source (explore: `https://github.com/wch/r-source`)
- polyvalent:
  - from laptop to most advanced supercomputers
  - local or remote
  - Windows, MacOS, linux or many other Unix-based systems
- rich (tons of **R** packages out there)
- cutting edge (check updates for today: `http://dirk.eddelbuettel.com/cranberries/cran/updated/`)
- used by millions
- **R** is the best software environment for statistical computing, but it is far from perfect!

Getting started with R

A short history of **S**/**R**

S (`http://ect.bell-labs.com/sl/S/`)

- 1976-1980: version 1: interactive statistical system, Fortran based (Becker, Chambers, & al. at Bell Labs)
- 1980-1988: version 2: portable version (thanks to Unix)
- 1988: version 3 (**S**3): "everything is an object" paradigm, C-based (very much like R)
- 1991: a large statistical modeling toolbox is added to **S**3
- 1993: **S**+ exclusive license (to StatSci, later MathSoft, later SolutionMetrics)
- 1998: version 4 (**S**4): advanced object-oriented features
- 2012: **S**+ becomes TIBCO Enterprise Runtime for R (TERR)

A short history of **S**/**R**

S (`http://ect.bell-labs.com/sl/S/`)

- 1976-1980: version 1: interactive statistical system, Fortran based (Becker, Chambers, & al. at Bell Labs)
- 1980-1988: version 2: portable version (thanks to Unix)
- 1988: version 3 (**S**3): "everything is an object" paradigm, C-based (very much like R)
- 1991: a large statistical modeling toolbox is added to **S**3
- 1993: **S**+ exclusive license (to StatSci, later MathSoft, later SolutionMetrics)
- 1998: version 4 (**S**4): advanced object-oriented features
- 2012: **S**+ becomes TIBCO Enterprise Runtime for R (TERR)

R (`https://www.r-project.org/about.html`)

- 1993: the replication of **S** as the **R** project starts (Ihaka & Gentleman at University of Auckland)
- 23/04/1997: first version of **R** archieved on The Comprehensive R Archive Network (CRAN)
- 05/12/1997: **R** version 0.6 is part of GNU project ("freedom to share, freedom to change")
- 29/02/2000: **R** version 1.0 (judged stable enough for production use by the R Development Core Team)

Getting started with R

Is **R** good for you?

Good for:

- data manipulation
- plots, including GIS
- analysing small, medium and big data
- programming around data

Is **R** good for you?

Good for:

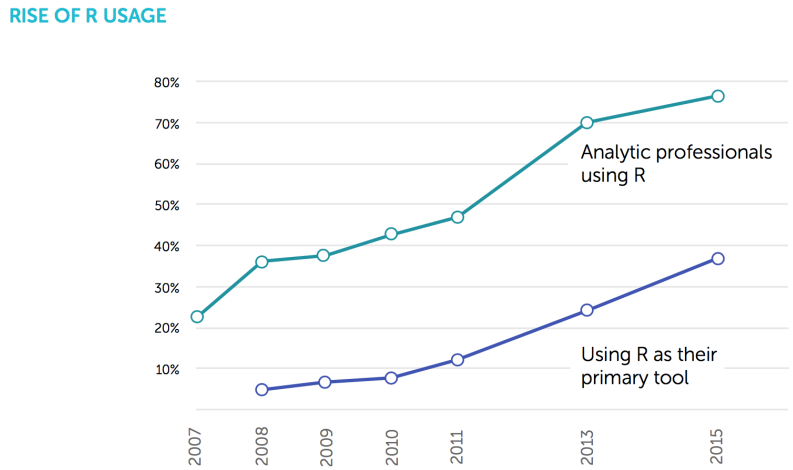
- data manipulation
- plots, including GIS
- analysing small, medium and big data
- programming around data

Not optimal for:

- beginners
- data entry
- formal algebra

Getting started with R

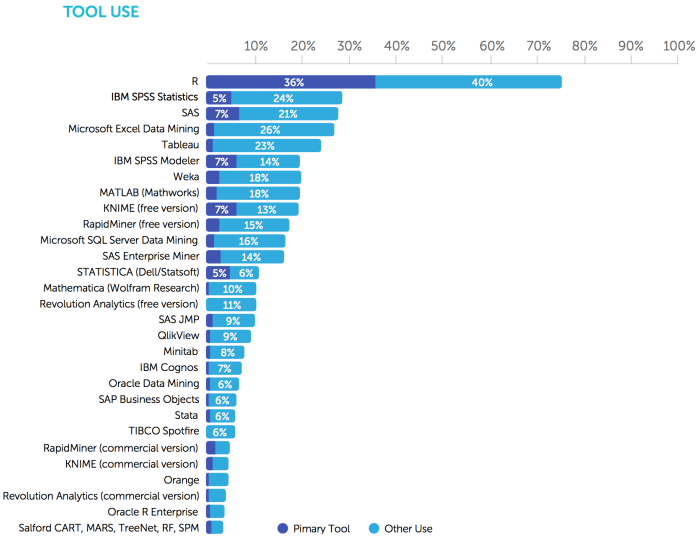
Who uses **R**?



Rexer Analytics

[1220 analytic professionals from 72 countries participated in this survey]

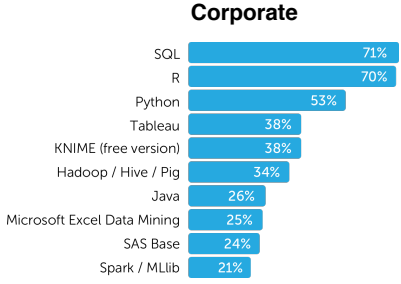
What else?

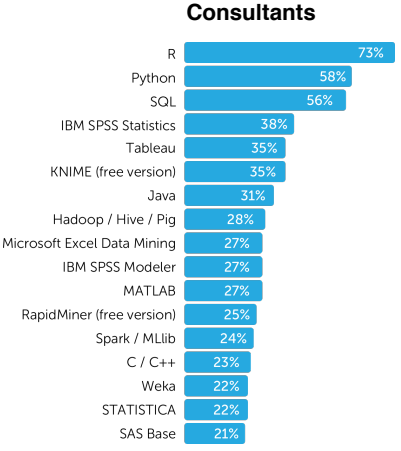


Rexer Analytics

Most Data Scientists use Multiple Tools

What data science / analytic tools, technologies, and languages did you use in the past year?

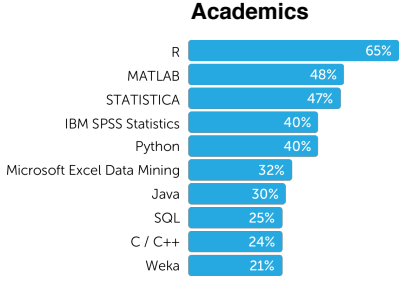


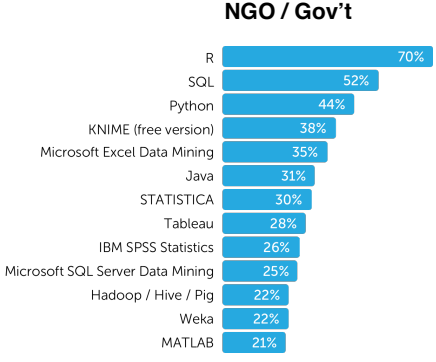


All tools used by more than 20% of a group are shown

Most Data Scientists use Multiple Tools

What data science / analytic tools, technologies, and languages did you use in the past year?





All tools used by more than 20% of a group are shown

Rich companies rely on **R** too!

Some examples from a blog post
(`http://blog.revolutionanalytics.com/2014/05/companies-using-r-in-2014.html`)

- Facebook (data analysis, big-data visualization, user behaviour analysis)
- Google (advertising effectiveness, economic forecasting, and big-data statistical modeling)
- Twitter (data visualization and semantic clustering)
- The City of Chicago (food poisoning monitoring)
- The New York Times (interactive features such as the Dialect Quiz and the Election Forecast)
- Microsoft (Xbox matchmaking)
- The Human Rights Data Analysis Group (counts of casualties in war zones)
- ANZ Bank (credit risk analysis)
- The FDA (regulatory drug approvals process)
- Monsanto (statistical analysis in plant breeding, fertility mapping and yield forecasting)
- Lloyds of London (risk analysis and catastrophe modeling)
- RealClimate.org (climate change analysis)
- NOAA (flood warnings)

Getting started with **R**

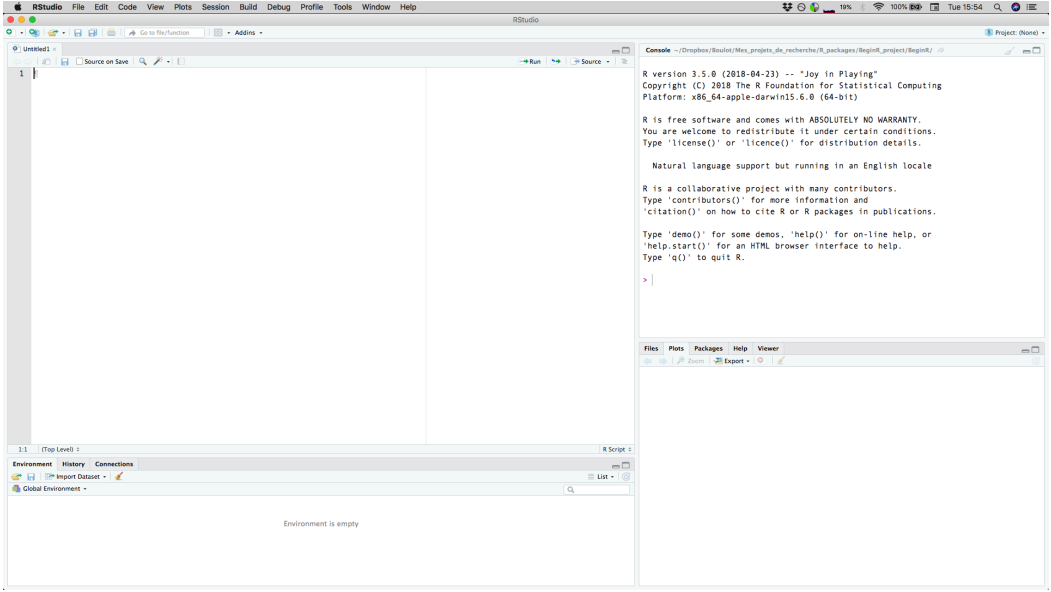
Getting started with R

Installation steps

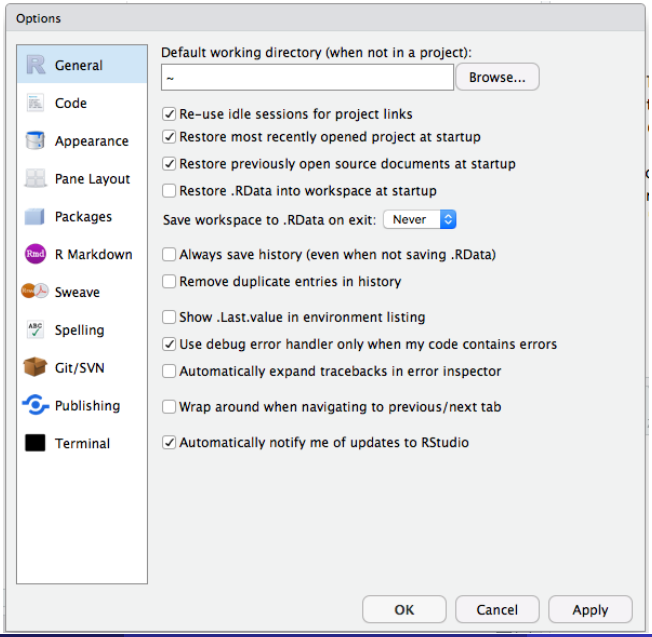
1. check that you do get internet access
2. install **R**: `https://cran.r-project.org/`
3. install RStudio: `https://www.rstudio.com/products/rstudio/download/`
4. open RStudio

Note: we will use RStudio but you don't have to (RStudio is free but not for all).

RStudio


RStudio File Edit Code View Plots Session Build Debug Profile Tools Window Help
RStudio
Addins
Project: (None)
Untitled1
Source on Save
Run
Source
Console ~/Dropbox/Boulot/Mes_projets_de_recherche/R_packages/BeginR_project/BeginR/
R version 3.5.0 (2018-04-23) -- "Joy in Playing"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin15.6.0 (64-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
Natural language support but running in an English locale
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
>
Files Plots Packages Help Viewer
Export
1:1 (Top Level)
R Script
Environment History Connections
Import Dataset
List
Global Environment
Environment is empty

Better default setting for RStudio



Getting started with R

Basic arithmetic

Try in the following in the "Console" pannel:

```
1 + 1
## [1] 2
1 - 1
## [1] 0
2 * pi
## [1] 6.283185
3 / 2
## [1] 1.5
10 %% 3
## [1] 1
5^2
## [1] 25
5^2 + 1
## [1] 26
5^(2+1)
## [1] 125
5^(2 + 1)
## [1] 125
```

Conclusion: you may never need a hand calculator anymore!

Getting started with R

Good practice

1. only use the "Console" pannel to mess around
2. write a script and comment it properly
3. make sure your script always work

The concept of an **R** script

All instructions must be written as a computer script!

- it is just a text file (no need for R to read it, it never gets corrupted)
- the script must be saved at a known location
- all non-**R** instruction must be preceeded by #

Why bother?

- transparent & reproducible
- easy to share & modify

Good practice

1. only use the "Console" pannel to mess around
2. write a script and comment it properly
3. make sure your script always work
4. store each result as an object with a useful name

Getting started with R

Creating objects

Objects are being assigned using the "arrow" operator:

```
one.plus.one <- 1 + 1 # storing the result
```

Objects are being used through their name (that is the whole point):

```
one.plus.one # displaying the result
```

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

```
one.plus.one.plus.one <- one.plus.one + 1
one.plus.one.plus.one
```

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

Tip:

```
(one.times.two <- 1 * 2) # storing and displaying the result
```

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

Note 1: avoid spaces & weird characters in object names to avoid troubles.
Note 2: names are case sensitive.

Common mistakes

The huge majority of beginners problems are typos:

```
one.plus.one
## [1] 2
one.plus.two
## Error in eval(expr, envir, enclos):  object 'one.plus.two' not found
one.plusone
## Error in eval(expr, envir, enclos):  object 'one.plusone' not found
1 +
one.plus.one <- 1 + 1
## Error in 1 + one.plus.one <- 1 + 1:  target of assignment expands to non-language object
```

The concept of an **R** object

What is an object?

- everything in **R** is an object (more on that later)
- objects have names
- objects allow astraction
- objects belongs to classes for which specific methods exist (and can be created)

Note for geeks who know other computer languages

R objects are (by default) not mutable (there is copy on demand):

```
a <- 1
b <- a
b <- b + 1
b
```

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

```
a
```

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

Getting started with R

Functions

```
citation()  # function showing how to cite R
```

```
##
## To cite R in publications use:
##
##   R Core Team (2018). R: A language and environment
##   for statistical computing. R Foundation for
##   Statistical Computing, Vienna, Austria. URL
##   https://www.R-project.org/.
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {R: A Language and Environment for Statistical Computing},
##     author = {{R Core Team}},
##     organization = {R Foundation for Statistical Computing},
##     address = {Vienna, Austria},
##     year = {2018},
##     url = {https://www.R-project.org/},
##   }
##
## We have invested a lot of time and effort in creating
## R, please cite it when using it for data analysis.
## See also 'citation("pkgname")' for citing R packages.
```

```
help(citation) # getting help for this function
?citation() # same but shorter (syntactic sugar)
```

Functions

```
mean()
```

```
?mean()
```

```
Usage:
     mean(x, ...)
     ## Default S3 method:
     mean(x, trim = 0, na.rm = FALSE, ...)

Arguments:
       x: An R object.  Currently there are methods for numeric/logical
          vectors and date, date-time and time interval objects, and
          for data frames all of whose columns have a method.  Complex
          vectors are allowed for 'trim = 0', only.
    trim: the fraction (0 to 0.5) of observations to be trimmed from
          each end of 'x' before the mean is computed.  Values of trim
          outside that range are taken as the nearest endpoint.
   na.rm: a logical value indicating whether 'NA' values should be
          stripped before the computation proceeds.
[...]
```

Syntax for functions

Basic syntax:

```
sign(x = -5)
## [1] -1
sign(-5) # dangerous: avoid!
## [1] -1
sign(y = -5)
## Error in sign(y = -5):  supplied argument name 'y' does not match 'x'
```

Equal signs and arrows are not the same:

```
sign(y <- -5) # dangerous: avoid!
## [1] -1
y
## [1] -5
sign(x = y <- -5) # same as above
## [1] -1
```

Syntax for functions

Not puting the parentheses shows the definition of the function:

```
sign
## function (x)  .Primitive("sign")
```

All functions need parentheses and exceptions correspond to syntactic sugar:

```
1 + 1
## [1] 2
`+`(1, 1)
## [1] 2
a <- 1
a
## [1] 1
`<-`(a, 1)
a
## [1] 1
```

Finding functions

To find the name of the function you are look for, you may try:

```
??"linear model"
```

or

```
help.search(pattern = "linear model", package = "stats")  # if you know where to look for
```

Getting started with R

The concept of an **R** package

Packages extend **R** functionalities:

- for most users; e.g. `ggplot2`
- for specific users; e.g. `IsoriX`
- for developpers; eg. `Rcpp`

Key facts about packages:

- a package is a folder (often compressed) containing **R** functions, data & documentation
- a library is an installed package
- there are tons of packages out there:
  - 12490 packages are available on `cran.r-project.org`
  - $\sim$ 1500 packages aimed at bioinformatic on `bioconductor.org`
  - many more on `github.com`
  - many more shared between users in other ways

Note: packages can be used to create research compendia!

Installing a package

Simple situation: the package is available as binary for your system on CRAN

```
install.packages("dpylr")
```

In general, the installation procedure depends on:

- where the package is being hosted (local, CRAN, bioconductor, GitHub, other)
- if the package contains sources in another language that have been compiled or not (yes, no)

In order to be able to install packages that require compilation (and thus have access to more or newer version of packages), you need to install:

- Rtools if you use Windows (`https://cran.r-project.org/bin/windows/Rtools/`)
- Xcode if you use macOS (`https://developer.apple.com/xcode/`)
- nothing if you use Linux or other Unix-based system

Installing the package for this course

The package is not on CRAN as I want to be able to update it instantaneously and have potentially large files.
I host the package here: `https://github.com/courtiol/BeginR`
You should install it as follows:

```
install.packages("drat")
drat::addRepo("courtiol")
install.packages("BeginR")
```

Loading a package

Loading a package makes the exported functions of the package and its data (if lazy-loaded) available to the user. Example:

```
library(BeginR)
```

Getting started with R

Updating R packages

Some things to know:

- **R** packages evovle quickly
- young **R** packages can be very buggy
- packages are not reviewed

Good practice:

- update your **R** packages daily

```
update.packages(ask = FALSE)
```

- check what is being changed if you heavily rely on a recent package
- contact the maintainer when you spot bugs (but write minimal reproductive examples!)

Updating R

Some things to know:

- **R** has many bugs (like all other software)
- **R** bugs are reported, discussed and solved in the open (unlike most other software): `https://bugs.r-project.org/bugzilla3/`
- each new version of **R** is in general more efficient, richer, and less buggy

What to do?

- check for **R** new versions on CRAN
- check for what has changed if you fancy (`https://cran.r-project.org/index.html`)
- install the new version of **R** (unless it is not a minor update that you don't need)
- re-install all your packages

Note 1: some packages can help to do this: `InstallR` on Windows and `UpdateR` on macOS
Note 2: also update Rstudio for full compatibility with **R**.

Getting started with R

Useful resources

- the help files and official documentation (boring but thorough `https://cran.r-project.org/manuals.html`)
- a nice blog: `http://www.r-bloggers.com/`
- 2 **R** local meetup groups: `https://www.meetup.com/Berlin-R-Users-Group/` & `https://www.meetup.com/rladies-berlin/`

Books:

Why is R the right choice for you?

R is a **free open source** programming language and software environment for statistical computing & graphics.

R includes **long-established parametric and non-parametric tests**, forefront methods in regression, classification & clustering, and much more.

- **numerous functions are included** within the core installation of R
- **12490 packages are available on** `cran.r-project.org`
- $\sim$ **1500 packages** aimed at bioinformatic on `bioconductor.org`

The user can run workflows stored in one or several script file(s), which allows for **reproducible research & easy communication**.

Installing a package to check proxy settings in R-Studio

```
install.packages("coin") # dialog box may be behind!
```

If it does not work follow one of the following options and try again!

- option 1:

```
options(repos = c(CRAN = "http://cran.r-project.org"))
```

- option 2: (for MAC users)

```
Sys.setenv(http_proxy = "http://192.168.2.2:3128")
```

- option 3:

```
paste(R.home(), "etc", "Renviron.site", sep = "/") # or "\" for windows
```

```
## [1] "/usr/lib64/R/etc/Renviron.site"
```

edit this file by adding:
`http_proxy=http://192.168.2.2:3128/`
`https_proxy=http://192.168.2.2:3128/`

Getting started with **R**

Getting started with R

Functions

```r
mean(x = c(1, 5, 3, 4))
```

```
## [1] 3.25
```

```r
vector.of.numbers <- c(1, 5, 3, 4)
mean(x = vector.of.numbers)
```

```
## [1] 3.25
```

```r
mean(vector.of.numbers)
```

```
## [1] 3.25
```

```r
mean(y = vector.of.numbers)
```

```
## Error in mean.default(y = vector.of.numbers):  argument "x" is missing, with no default
```

Challenge #1

```
new.vector <- c(vector.of.numbers, 6, NA, 2)
new.vector
## [1]  1  5  3  4  6 NA  2
```

Try to compute the mean of `new.vector` using `mean()`!

Key principles of the R language

- Everything that exists in R is an object
- Everything that happens in R is a function call

John M. Chambers

This is true even for things that do not look like it at first glance:

```
class(`+`)  # same for `?`...
## [1] "function"
`+`(1, 1)
## [1] 2
```

Getting started with **R**

Getting started with R

Vectors

Vectors allow the organisation of entities (e.g. numbers, characters...) along one dimension which can be indexed!

```
height.girls <- c(178, 175, 159, 164, 183, 192)
height.boys  <- c(181, 189, 174, 177)
```

```
height.girls[2]
## [1] 175
height.boys[3]
## [1] 174
```

Vectors

Vectors can be combined:

```
height <- c(height.girls, height.boys)
height
```

```
##  [1] 178 175 159 164 183 192 181 189 174 177
```

Vectors

They can be indexed logically (i.e. indexed by anything leading to a vector of booleans):

```
height > 179
```

```
##  [1] FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE FALSE
## [10] FALSE
```

```
height[height > 179]
```

```
## [1] 183 192 181 189
```

```
height[!(height == min(height))]
```

```
## [1] 178 175 164 183 192 181 189 174 177
```

```
height[height != min(height)]
```

```
## [1] 178 175 164 183 192 181 189 174 177
```

Vectors

They work with other things than numbers:

```
sex <- c("girl","girl","girl","girl","girl", "girl",
         "boy","boy","boy","boy")
sex <- factor(sex)
sex
```

```
##  [1] girl girl girl girl girl girl boy  boy  boy  boy
## Levels: boy girl
```

```
# Or
sex <- factor(c(rep("girl", times = 6),
                rep("boy",  times = 4)))
# Or
sex <- factor(c(rep("girl", times = length(height.girls)),
                rep("boy",  times = length(height.boys))))
```

Vectors

Many functions can take a vector as an input:

```
unique(sex)
```

```
## [1] girl boy
## Levels: boy girl
```

```
length(sex)
```

```
## [1] 10
```

```
table(sex)
```

```
## sex
##  boy girl
##    4    6
```

```
min(height)
```

```
## [1] 159
```

```
max(height)  # try range()
```

```
## [1] 192
```

```
mean(height)  # try median()
```

```
## [1] 177.2
```

```
var(height) # try sd()
```

```
## [1] 103.0667
```

```
summary(height)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   159.0   174.2   177.5   177.2   182.5   192.0
```

Data frames

Data frames allow the organisation of entities as a matrix-like structure whose columns have the same length:

```
dataframe.ht <- data.frame(Height = height, Sex = sex)
head(dataframe.ht)
```

```
##   Height  Sex
## 1    178 girl
## 2    175 girl
## 3    159 girl
## 4    164 girl
## 5    183 girl
## 6    192 girl
```

Data frames

It is good practice to always check their structure:

```
str(dataframe.ht)
```

```
## 'data.frame': 10 obs. of  2 variables:
##  $ Height: num  178 175 159 164 183 192 181 189 174 177
##  $ Sex   : Factor w/ 2 levels "boy","girl": 2 2 2 2 2 2 1 1 1 1
```

Data frames

You access the columns by means of the extractor $

```
height
```

```
##  [1] 178 175 159 164 183 192 181 189 174 177
```

```
rm(list = c("height", "sex")) # removing original vectors
height
```

```
## Error in eval(expr, envir, enclos):  object 'height' not found
```

```
dataframe.ht$Height #Or: with(data = dataframe.ht, Height)
```

```
##  [1] 178 175 159 164 183 192 181 189 174 177
```

$\Rightarrow$ What is the average height?

Data frames

Some functions can take a data frame as an input:

```
summary(dataframe.ht)
```

```
##      Height        Sex
##  Min.   :159.0   boy :4
##  1st Qu.:174.2   girl:6
##  Median :177.5
##  Mean   :177.2
##  3rd Qu.:182.5
##  Max.   :192.0
```

Note: this will be the case of a lot of functions performing statistical tests!

Data frames

How to compute the average height per sex?

- simple

```
mean(dataframe.ht$Height[dataframe.ht$Sex == "boy"])
```

```
## [1] 180.25
```

- more elegant

```
tapply(X = dataframe.ht$Height, INDEX = dataframe.ht$Sex,
       FUN = mean)
```

```
##      boy     girl
## 180.2500 175.1667
```

```
# Or: with(data = dataframe.ht, tapply(X = Height,
#   INDEX = Sex, FUN = mean))
```

Data frames

They can also be indexed:

```
dataframe.ht[1, ]
##   Height  Sex
## 1    178 girl
dataframe.ht[, 1]  # Or: dataframe.ht[, "Sex"]
##  [1] 178 175 159 164 183 192 181 189 174 177
```

Data frames

They can be edited:

```
dataframe.ht[1, 1]
```

```
## [1] 178
```

```
dataframe.ht[1, 1] <- 171.3
dataframe.ht[1, 1]
```

```
## [1] 171.3
```

```
dataframe.ht$linenumber <- 1:nrow(dataframe.ht)  # add column
ncol(dataframe.ht) # try dim()
```

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

```
dataframe.ht$linenumber <- NULL  # remove column
ncol(dataframe.ht)
```

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

Lists

Lists allow the organisation of any set of entities into a single R object:

```
list.ht <- list(girls = height.girls, boys = height.boys)
list.ht
```

```
## $girls
## [1] 178 175 159 164 183 192
##
## $boys
## [1] 181 189 174 177
```

Lists

Lists can also be indexed and their elements extracted:

```
list.ht$girls
## [1] 178 175 159 164 183 192
list.ht["boys"]  # still a list
## $boys
## [1] 181 189 174 177
list.ht[["boys"]]  # vector
## [1] 181 189 174 177
list.ht[[2]][3]
## [1] 174
```

Lists

Some functions can take a list as an input:

```
lapply(list.ht, FUN = mean)
```

```
## $girls
## [1] 175.1667
##
## $boys
## [1] 180.25
```

Summary

```
dataframe.ht
```

```
##    Height  Sex
## 1   171.3 girl
## 2   175.0 girl
## 3   159.0 girl
## 4   164.0 girl
## 5   183.0 girl
## 6   192.0 girl
## 7   181.0  boy
## 8   189.0  boy
## 9   174.0  boy
## 10  177.0  boy
```

```
list.ht
```

```
## $girls
## [1] 178 175 159 164 183 192
##
## $boys
## [1] 181 189 174 177
```

Summary

- `data.frame`
  - All columns have same length
  - Each column can have its own class (e.g. `numeric`, `factor`, `character`)
- `list`
  - Each element can have its own length
  - Each element can have its own class (e.g. `numeric`, `factor`, `character`)

Getting started with **R**

Getting started with R

Working directory

```
getwd()  # to change, use setwd()
```

```
## [1] "/home/colin/BeginR/BeginR/vignettes/introduction"
```

```
dir() # listing all files in the working directory
```

```
## [1] "figure"                    "introduction.pdf"
## [3] "introduction.Rnw"          "my_plot.pdf"
## [5] "my.first.R.dataframe.csv"
```

```
dir(pattern = "*.csv")
```

```
## [1] "my.first.R.dataframe.csv"
```

Exporting and importing data in R

```r
write.csv(dataframe.ht,
    file = "my.first.R.dataframe.csv", row.names = FALSE)

rm(list = ls())  # deleting everything in R

dataframe.ht <- read.csv("my.first.R.dataframe.csv")
```

R cannot read/write .xls files out of the box
Packages can do that but it is safer to use .csv files
Excel can read and write .csv files!

Challenge #2

Create a dataframe using your favorite spreadsheet software
and import it in R!

Getting started with **R**

Getting started with R

Foreplay

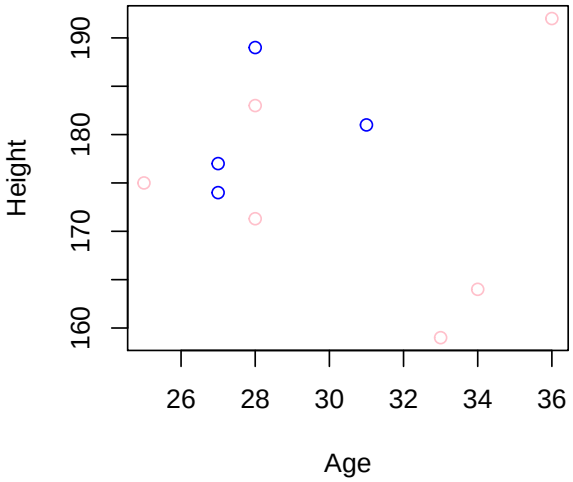
First, let's create a new column `Age` in our set `dataframe.ht`:

```
dataframe.ht$Age <- c(28, 25, 33, 34, 28, 36, 31, 28, 27, 27)
head(dataframe.ht)
```

```
##   Height  Sex Age
## 1  171.3 girl  28
## 2  175.0 girl  25
## 3  159.0 girl  33
## 4  164.0 girl  34
## 5  183.0 girl  28
## 6  192.0 girl  36
```

Scatter plot

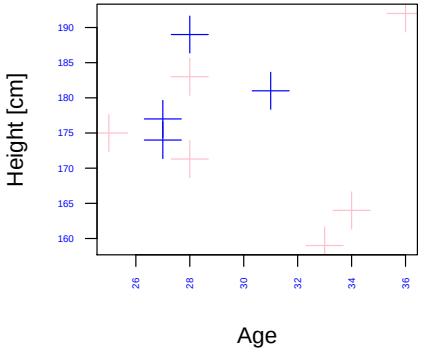
```
color.sex <- ifelse(test = dataframe.ht$Sex == "girl",
                    yes = "pink", no = "blue")

plot(Height ~ Age, data = dataframe.ht, col = color.sex)
```



Scatter plot

```
?plot.default
?par # to set graphical parameters
```

```
plot(Height ~ Age, data = dataframe.ht, col = color.sex,
    pch = 3, xlab = "Age", ylab = "Height [cm]",
    cex = 3, cex.lab = 1.2, cex.axis = 0.5,
    las = 2, col.axis = "blue")
```

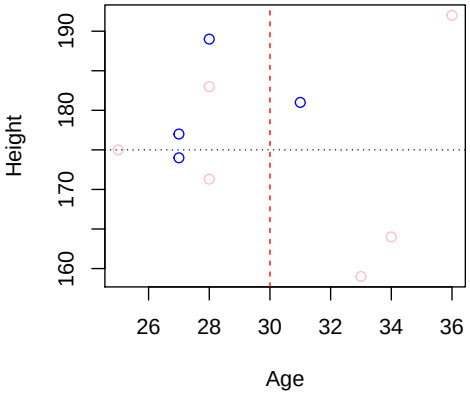


Decorating plots

Drawing lines (h = horizontal, v = vertical, or "intercept, slope"):

```
?abline
?text
```

```
plot(Height ~ Age, data = dataframe.ht, col=color.sex)
abline(v = 30, lty = "dashed", col = "red") #lty = line type
abline(h = 175, lty = "dotted")
```

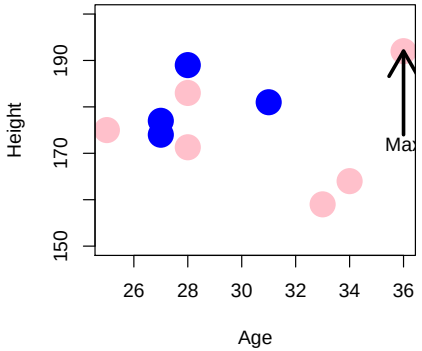


Decorating plots

Drawing arrows and adding text:

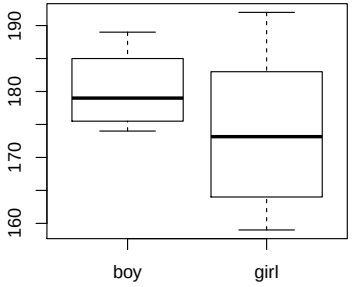
```
?arrows
?text
```

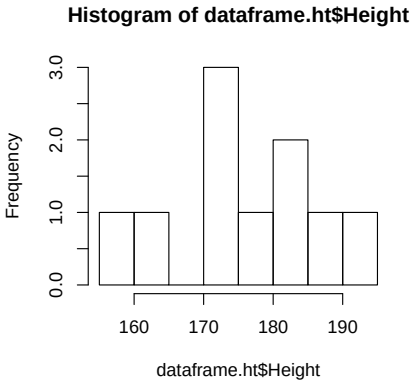
```
plot(Height ~ Age, data = dataframe.ht, col = color.sex,
     pch = 16, cex = 3, ylim = c(150, 200))
max.value <- max(x = dataframe.ht$Height)
where.max <- dataframe.ht$Age[which.max(x = dataframe.ht$Height)]
arrows(x0 = where.max, y0 = max.value - 18,
       x1 = where.max,  y1 = max.value, col = "black", lwd = 3) #lwd = line width
text(x = where.max, y = max.value - 20, labels = "Max")
```



Box plots and histograms

```
par(mfrow = c(1, 2))
boxplot(Height ~ Sex, data = dataframe.ht)
hist(dataframe.ht$Height)
```

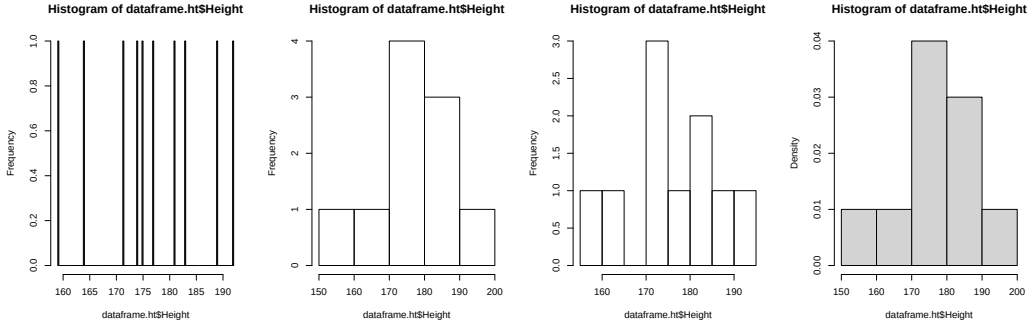




Histograms

```
?hist
```

```
par(mfrow = c(1, 4))
hist(dataframe.ht$Height, breaks = max)
hist(dataframe.ht$Height, breaks = 3)
hist(dataframe.ht$Height, breaks = 5)
hist(dataframe.ht$Height, breaks = seq(from = 150, to = 200, by = 10),
     freq = FALSE, col = "lightgrey")
```



Histograms

```
my.h <- hist(dataframe.ht$Height)
my.h
```

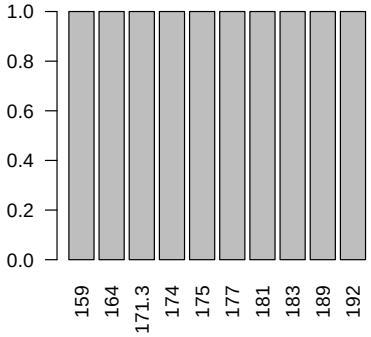
```
## $breaks
## [1] 155 160 165 170 175 180 185 190 195
##
## $counts
## [1] 1 1 0 3 1 2 1 1
##
## $density
## [1] 0.02 0.02 0.00 0.06 0.02 0.04 0.02 0.02
##
## $mids
## [1] 157.5 162.5 167.5 172.5 177.5 182.5 187.5 192.5
##
## $xname
## [1] "dataframe.ht$Height"
##
## $equidist
## [1] TRUE
##
## attr(,"class")
## [1] "histogram"
```

Bar charts

```
mybp <- table(dataframe.ht$Height)
mybp
```

```
##
##   159   164 171.3   174   175   177   181   183   189   192
##     1     1     1     1     1     1     1     1     1     1
```

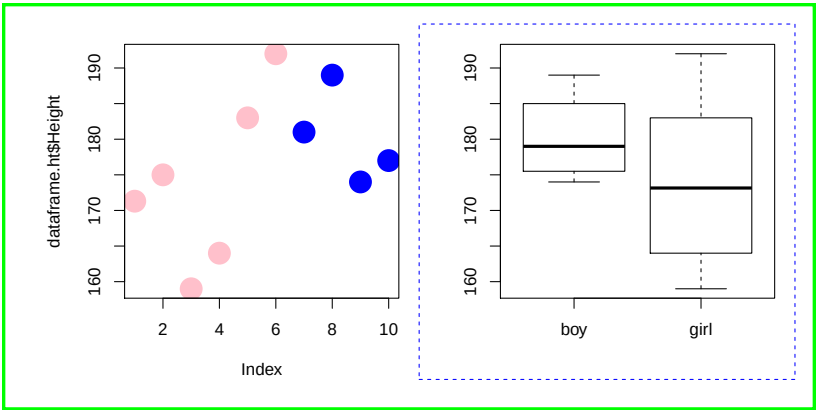
```
par(mfrow = c(1, 1), las = 2)
barplot(mybp)
```



Margins and multiple figures

Dividing the graphics device: `mfrow` = multiframe rowwise, `mar` = margins (dashed blue), `oma` = outer margins (green)

```
par(mfrow = c(1, 2), mar = c(4, 4, 1, 1),
    oma = c(1.5, 2, 1, 1))
plot(dataframe.ht$Height, col = color.sex, pch = 16, cex = 3)
boxplot(Height ~ Sex, data = dataframe.ht)
```



Exporting figures

```
?pdf ?postscript
?bmp ?jpeg ?png ?tiff
```

```
pdf(file = "my_plot.pdf", width = 14, height = 7)
par(mfrow = c(1, 2), mar = c(4, 4, 1, 1),
    oma = c(1.5, 2, 1, 1))
plot(dataframe.ht$Height, col = color.sex, pch = 16, cex = 3)
boxplot(Height ~ Sex, data = dataframe.ht)
dev.off() # close the pdf
```

```
## pdf
##   2
```

```
getwd() # to check where the file is
```

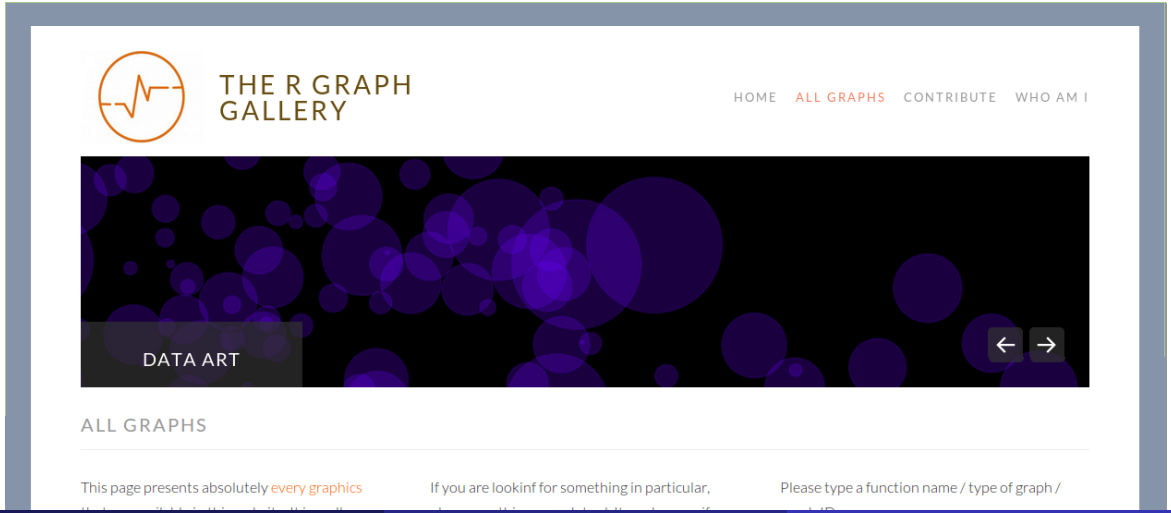
```
## [1] "/home/colin/BeginR/BeginR/vignettes/introduction"
```

R code for graphs and examples

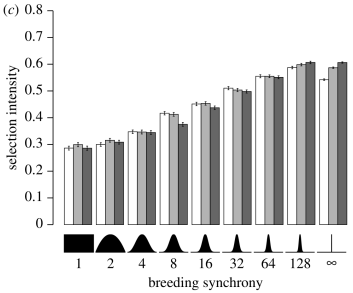
1. Check graph example codes in the help function, e.g.

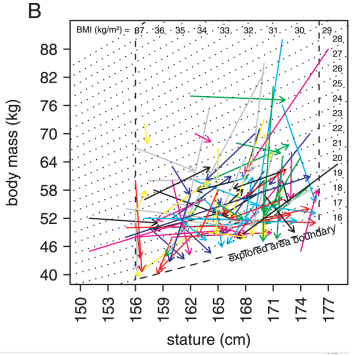
```
example(hist)
```

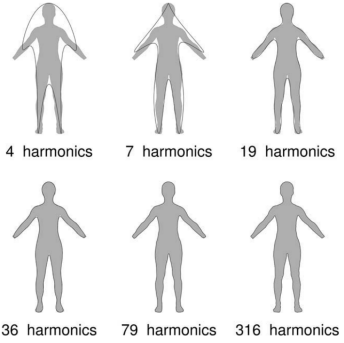
2. Scroll the web
(e.g. `http://www.r-graph-gallery.com/all-graphs/`)



Some of my home-made R graphics







Getting started with **R**

Getting started with R

Usual programming commands exist in R

```
for (i in 1:4) {
  print(x = i)
  if (i == 2) print(x = "found 2!")
}
```

```
## [1] 1
## [1] 2
## [1] "found 2!"
## [1] 3
## [1] 4
```

```
?"for"
```

You can write your own functions!

```
OddRatio <- function(a, b) {
  odd.a <- a/(1 - a)
  odd.b <- b/(1 - b)
  return(odd.a/odd.b)
}
```

```
OddRatio(0.1, 0.01)
```

```
## [1] 11
```

How to get the code behind a function?

Usually, by simply typing its name (without brackets). But that is not always sufficient...

```
lm
```

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

How to get the code behind a function?

R methods (S3):

```
residuals
```

```
## function (object, ...)
## UseMethod("residuals")
## <bytecode: 0x5601684ac3a8>
## <environment: namespace:stats>
```

`residuals()` is a *generic* function which rely on class specific methods:

```
methods(residuals)
```

```
## [1] residuals.default*       residuals.glm           residuals.HoltWinters*
## [4] residuals.isoreg*        residuals.lm            residuals.nls*
## [7] residuals.smooth.spline* residuals.tukeyline*
## see '?methods' for accessing help and source code
```

The methods with a $*$ are not exported from their package namespace.

How to get the code behind a function?

Getting the code for exported R methods (S3):

```
residuals.lm
```

```
## function (object, type = c("working", "response", "deviance",
##     "pearson", "partial"), ...)
## {
##     type <- match.arg(type)
##     r <- object$residuals
##     res <- switch(type, working = , response = r, deviance = ,
##         pearson = if (is.null(object$weights)) r else r * sqrt(object$weights),
##         partial = r)
##     res <- naresid(object$na.action, res)
##     if (type == "partial")
##         res <- res + predict(object, type = "terms")
##     res
## }
## <bytecode: 0x560169f99cc8>
## <environment: namespace:stats>
```

How to get the code behind a function?

Getting the code for non-exported R methods (S3):

```
residuals.nls
```

```
## Error in eval(expr, envir, enclos):  object 'residuals.nls' not found
```

```
getAnywhere("residuals.nls") # or getS3method("residuals", "nls")
```

```
## A single object matching 'residuals.nls' was found
## It was found in the following places
##   registered S3 method for residuals from namespace stats
##   namespace:stats
## with value
##
## function (object, type = c("response", "pearson"), ...)
## {
##     type <- match.arg(type)
##     if (type == "pearson") {
##         val <- as.vector(object$m$resid())
##         std <- sqrt(sum(val^2)/(length(val) - length(coef(object))))
##         val <- val/std
##         if (!is.null(object$na.action))
##             val <- naresid(object$na.action, val)
##         attr(val, "label") <- "Standardized residuals"
##     }
##     else {
##         val <- as.vector(object$m$lhs() - object$m$fitted())
##         if (!is.null(object$na.action))
##             val <- naresid(object$na.action, val)
##         lab <- "Residuals"
##         if (!is.null(aux <- attr(object, "units")$y))
##             lab <- paste(lab, aux)
##         attr(val, "label") <- lab
##     }
##     val
```

Challenge #3

What is the code behind *t.test()*?

How to get the code behind a function?

Some functions – the interfaces – call functions that are written in other languages. The source code of these latter functions is not directly visible (spotted as `.C()`, `.Fortran()`, `.Call()`, `.Primitive()`, `.Internal()`, `.External()`).

```
dnorm
```

```
## function (x, mean = 0, sd = 1, log = FALSE)
## .Call(C_dnorm, x, mean, sd, log)
## <bytecode: 0x5601694626b8>
## <environment: namespace:stats>
```

In these cases, the easiest is to use the read-only mirror for R (`https://github.com/wch/r-source`) or the relevant package on Github! (here, the answer lies in r-source/src/nmath/dnorm.c)

Numerical issues common to most programming languages

```
print(seq(0, 1, 0.1), digits = 22)
```

```
##  [1] 0.0000000000000000000000 0.1000000000000000055511 0.2000000000000000111022
##  [4] 0.3000000000000000444089 0.4000000000000000222045 0.5000000000000000000000
##  [7] 0.6000000000000000888178 0.7000000000000000666134 0.8000000000000000444089
## [10] 0.9000000000000000222045 1.0000000000000000000000
```

```
x <- 0.7 - 0.4 - 0.3
print(x, digits = 22)
```

```
## [1] -5.551115123125782702118e-17
```

```
x == 0
```

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

NB: same kind of thing can happen in Excel too (`https://support.microsoft.com/en-us/kb/214118`)

Numerical issues common to most programming languages

```
??"equality"
```

```
Help files with alias or concept or title matching 'equality' using
fuzzy matching:


FactoMineR::prefpls     Scatter plot and additional variables with
                        quality of representation contour lines
base::all.equal         Test if Two Objects are (Nearly) Equal
base::identical         Test Objects for Exact Equality
datasets::airquality    New York Air Quality Measurements
```

```
?all.equal
all.equal(target = 0, current = x)
```

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

Getting started with **R**

Getting started with R

How to learn R on your own?

R:

- `RSiteSearch("keyword(s)")`

Internet:

- `http://www.r-project.org/` (Manuals in many languages for free)
- `http://www.r-bloggers.com/` (Blog!)

Books: