Getting started with R

Alexandre Courtiol & Liam Bailey

Leibniz Institute of Zoo and Wildlife Research

February 2019

Leibniz Institute for Zoo
and Wildlife Research
IN THE FORSCHUNGSVERBUND BERLIN E.V.

What is R?

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

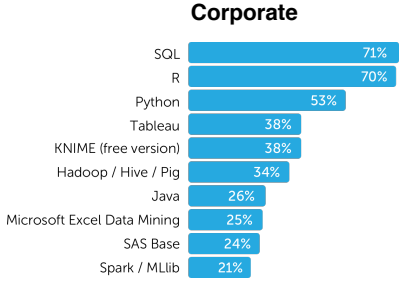
R includes **long-established parametric and non-parametric tests**, forefront methods in regression, classification & clustering, and much more.

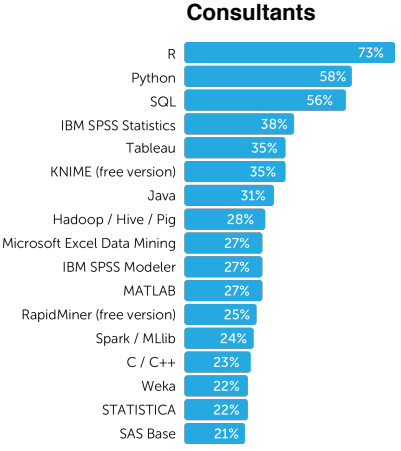
There are many thousands **R packages** out there for you to do even more.

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

Most Data Scientists use Multiple Tools

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

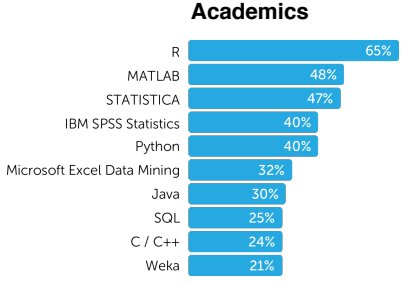


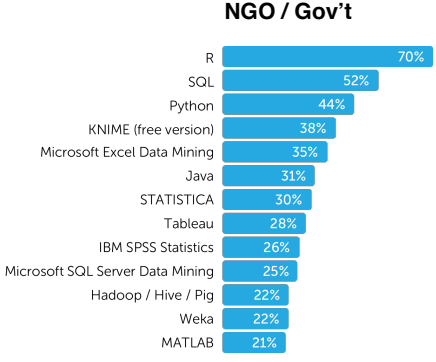


All tools used by more than 20% of a group are shown

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

Is R good for you?

Good for:

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

Is R good for you?

Good for:

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

Not optimal for:

- beginners
- data entry
- formal algebra

Getting started with R

1. Installing R
2. Basics
3. Organising data
4. Importing/exporting data
5. Plotting
6. Programming
7. Learning about R

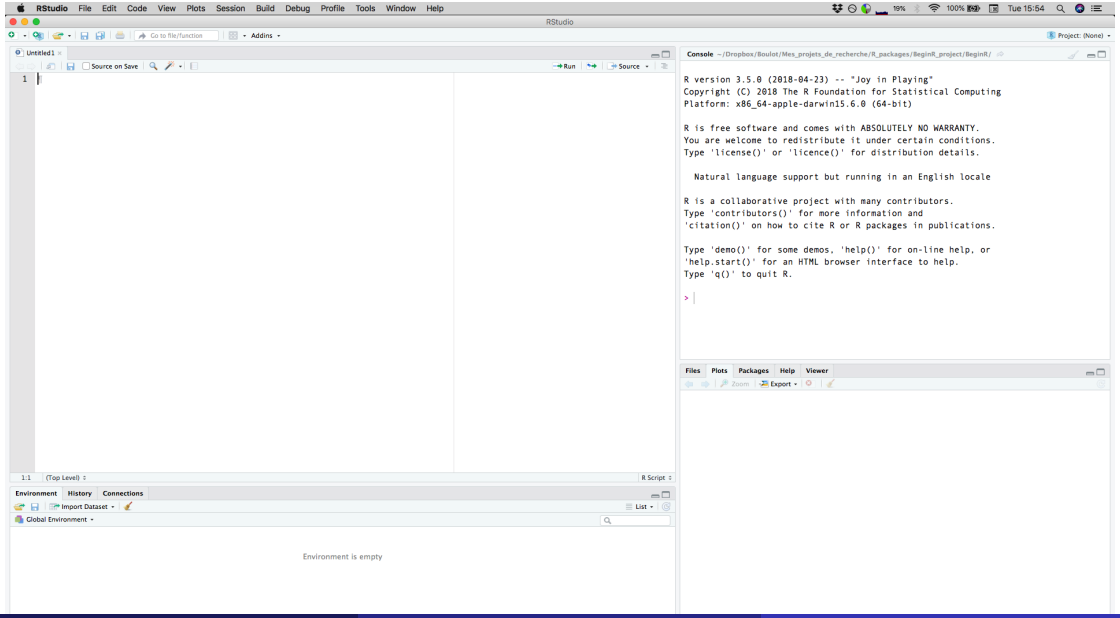
Getting started with R

Installation steps

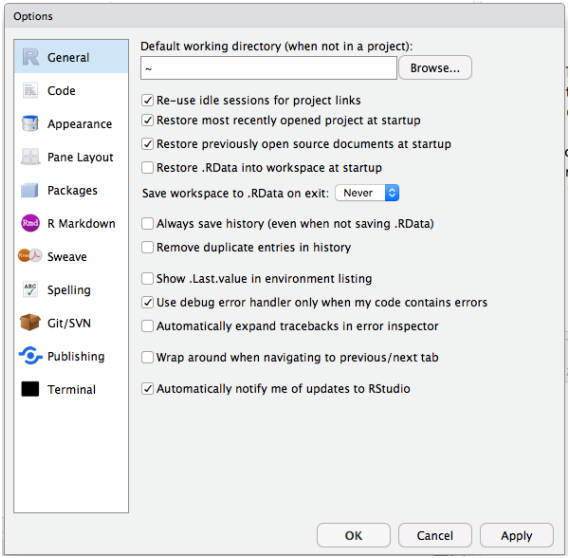
1. connect to the WIFI network `izw-gast` (check password on board)
2. set the internet proxy: 192.168.2.2:3128 (necessary at IZW, but usually not)
3. check that you do get internet access
4. install R: `https://cran.r-project.org/`
5. install RStudio: `https://www.rstudio.com/products/rstudio/download/`
6. open RStudio

Note: I will use RStudio but you don't have to (RStudio is free and open source).

RStudio


RStudio File Edit Code View Plots Session Build Debug Profile Tools Window Help
Tue 15:54
RStudio
Go to file/function
Addins
Project: (None)
Untitled1
Source on Save
Run
Source
1
1:1
(Top Level)
R Script
Environment History Connections
Import Dataset
List
Global Environment
Environment is empty
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
Zoom
Export

Better default settings for RStudio



Getting started with R

The concept of an R script

All instructions should 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 instructions must be preceded by the character #

The concept of an R script

All instructions should 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 instructions must be preceded by the character #

```
################################
## this is my first R script ##
################################

### simple arithmetic
1 + 1 ## compute 1 + 1
```

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

```
#1 + 2 ## commented lines of code won't run!
```

The concept of an R script

All instructions should 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 instructions must be preceded by the character #

```
################################
## this is my first R script ##
################################

### simple arithmetic
1 + 1 ## compute 1 + 1
```

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

```
#1 + 2 ## commented lines of code won't run!
```

Why bother writing a script?

- transparent & reproducible
- easy to share & modify

Good practice

1. only use the "Console" panel to mess around
2. write a script and comment it thoroughly
3. **make sure your script always works by re-running the whole script often!**

Good practice

1. only use the "Console" panel to mess around
2. write a script and comment it thoroughly
3. **make sure your script always works by re-running the whole script often!**

Alternatives for you to explore in the future:

- You can use R Notebooks or R Markdown files instead of a simple script:
  RStudio: File → New File (but it needs packages, so don't try before we set internet up!)

Good practice

1. only use the "Console" panel to mess around
2. write a script and comment it thoroughly
3. **make sure your script always works by re-running the whole script often!**

Alternatives for you to explore in the future:

- You can use R Notebooks or R Markdown files instead of a simple script:
  RStudio: File → New File (but it needs packages, so don't try before we set internet up!)

  Using such documents offers the possibility to easily combine formatted text, embedded R code and outputs. Such documents can be rendered into HTML, PDF, Microsoft Office Word and Powerpoints formats, and more.
  (see `https://rmarkdown.rstudio.com/` for details)

Good practice

1. only use the "Console" panel to mess around
2. write a script and comment it thoroughly
3. **make sure your script always works by re-running the whole script often!**

Alternatives for you to explore in the future:

- You can use R Notebooks or R Markdown files instead of a simple script:
  RStudio: File → New File (but it needs packages, so don't try before we set internet up!)

  Using such documents offers the possibility to easily combine formatted text, embedded R code and outputs. Such documents can be rendered into HTML, PDF, Microsoft Office Word and Powerpoints formats, and more.
  (see `https://rmarkdown.rstudio.com/` for details)
- If you know LaTeX, you can also integrate your R code within LaTeX documents.
  (I created those slides that way)

R as a calculator

Try in the following:

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

Creating objects

Information can be stored into *objects* which are being created using the "arrow" operator:

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

Creating objects

Information can be stored into *objects* which are being created using the "arrow" operator:

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

Once created, objects are used via 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
```

Creating objects

Information can be stored into *objects* which are being created using the "arrow" operator:

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

Once created, objects are used via 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
```

Note 1: avoid spaces & weird characters (but "_" and "." are OK).
Note 2: names are case sensitive.

Common mistakes

```
one.plus.one
## [1] 2
one.plus.two
## Error in eval(expr, envir, enclos):  object 'one.plus.two' not found
One.plus.one
## Error in eval(expr, envir, enclos):  object 'One.plus.one' 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
```

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.
```

```
?citation() # getting help for this function
```

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.
[...]
```

Functions

```
mean(x = c(1, 5, 3, 4))
## [1] 3.25
vector.of.numbers <- c(1, 5, 3, 4)
mean(x = vector.of.numbers)
## [1] 3.25
mean(vector.of.numbers)
## [1] 3.25
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()`!

R packages

Packages extend R functionalities:

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

R packages

Packages extend R functionalities:

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

Key facts about packages:

- a package is just a folder (often compressed) containing R functions, data & documentation
- a library is the installed version of the package (also a folder)
- there are tons of packages out there:
  - 13662 packages are available on `cran.r-project.org`
  - $\sim$ 1500 packages aimed at bioinformatics on `bioconductor.org`
  - many more on `github.com`
  - many more shared between users in other ways

R packages

Packages extend R functionalities:

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

Key facts about packages:

- a package is just a folder (often compressed) containing R functions, data & documentation
- a library is the installed version of the package (also a folder)
- there are tons of packages out there:
  - 13662 packages are available on `cran.r-project.org`
  - $\sim$ 1500 packages aimed at bioinformatics 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 (outside IZW): the package is available as a binary file prepared for your system on CRAN

```
install.packages("coin")  ## install coin
```

Installing a package

Simple situation (outside IZW): the package is available as a binary file prepared for your system on CRAN

```
install.packages("coin")  ## install coin
```

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

Installing a package

Simple situation (outside IZW): the package is available as a binary file prepared for your system on CRAN

```
install.packages("coin")  ## install coin
```

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

In order to be able to install packages that require compilation (and thus have access to more packages or more rencent versions), 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 a package to check IZW proxy settings in RStudio

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

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

- create or edit the file `.Renviron`:

  ```
  file.edit("~/.Renviron")
  ```

- add to this file the following lines:
  `http_proxy=http://192.168.2.2:3128/`
  `https_proxy=http://192.168.2.2:3128/`

Note: at home, just delete the file, rename it or comment the lines inside it!

Loading a library

That is always simple:

```
library(coin)
```

```
## Loading required package:  survival
```

Contrary to the installation that is only needed once per R installation, loading the libraries you need must be done each time you open an R session!

Updating packages

Some things to know:

- R packages evovle quickly
- young R packages can be very buggy
- packages are not reviewed
  (CRAN tests that they can install and that the examples run without generating error or warning messages)

Updating packages

Some things to know:

- R packages evovle quickly
- young R packages can be very buggy
- packages are not reviewed
  (CRAN tests that they can install and that the examples run without generating error or warning messages)

Good practice:

- update your R packages frequently (I do it daily)

```
update.packages(ask = FALSE, checkBuilt = TRUE) ## or use the RStudio menu
```

Updating R itself

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 more efficient and less buggy

Updating R itself

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 more efficient and less buggy

What to do?

- check and install new versions of R using CRAN
- 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 regularly for the same reasons.

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, 172)
height.boys  <- c(181, 175, 174, 174)
```

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

Vectors

Vectors can be combined:

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

```
##  [1] 178 175 159 164 183 172 181 175 174 174
```

Vectors

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

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

```
## [1] 183 181
```

```
height > 179
```

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

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

```
## [1] 178 175 164 183 172 181 175 174 174
```

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

```
## [1] 178 175 164 183 172 181 175 174 174
```

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] 183
```

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

```
## [1] 173.5
```

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

```
## [1] 52.72222
```

```
summary(height)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   159.0   172.5   174.5   173.5   177.2   183.0
```

Data frames

Data frames allow the organisation of entities as a matrix-like structure whose columns are vectors of 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    172 girl
```

Data frames

It is good practice to always check the structure of data frames:

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

```
## 'data.frame': 10 obs. of  2 variables:
##  $ Height: num  178 175 159 164 183 172 181 175 174 174
##  $ 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] 178 175 159 164 183 172 181 175 174 174
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] 178 175 159 164 183 172 181 175 174 174
```

$\Rightarrow$ 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.:172.5   girl:6
##  Median :174.5
##  Mean   :173.5
##  3rd Qu.:177.2
##  Max.   :183.0
```

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] 176
```

- more elegant

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

```
##      boy     girl
## 176.0000 171.8333
```

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

Data frames

The package `dplyr` can be used to manipulate data frames (& `tibbles`; their kissing cousins) very effectively.

```
library(dplyr)

dataframe.ht %>% # the weird operator is a forward pipe, see ?magrittr::`%>%`
  group_by(Sex) %>%
  summarise(Mean_height = mean(Height), N = n()) %>%
  data.frame()
```

```
##    Sex Mean_height N
## 1  boy    176.0000 4
## 2 girl    171.8333 6
```

Data frames

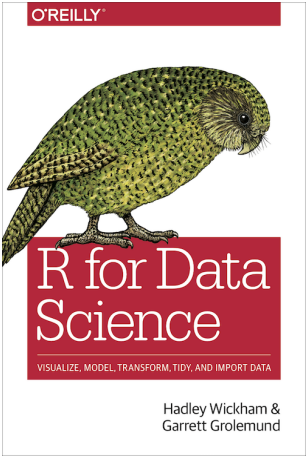
The package `dplyr` can be used to manipulate data frames (& `tibbles`; their kissing cousins) very effectively.

```
library(dplyr)

dataframe.ht %>% # the weird operator is a forward pipe, see ?magrittr::`%>%`
  group_by(Sex) %>%
  summarise(Mean_height = mean(Height), N = n()) %>%
  data.frame()
##    Sex Mean_height N
## 1  boy    176.0000 4
## 2 girl    171.8333 6
```

It works similarly on data stored in relational database management system (e.g. MySQL, SQLite, MariaDB, Postgres...) using `dbplyr`.

To learn more about the tidyverse (i.e. dplyr and related packages)



free at `https://r4ds.had.co.nz` or $\sim$ 28€ for the printed book

Data frames

Data frames can also be indexed:

```
dataframe.ht[1, ]
##   Height  Sex
## 1    178 girl
dataframe.ht[, 1]  # Or: dataframe.ht[, "Sex"]
##  [1] 178 175 159 164 183 172 181 175 174 174
```

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] 178 175 159 164 183 172
##
## $boys
## [1] 181 175 174 174
```

Lists

Lists can also be indexed and their elements extracted:

```
list.ht$girls
## [1] 178 175 159 164 183 172
list.ht["boys"]  # still a list
## $boys
## [1] 181 175 174 174
list.ht[["boys"]]  # vector
## [1] 181 175 174 174
list.ht[[2]][3]
## [1] 174
```

Lists

Some functions can take a list as an input:

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

```
## $girls
## [1] 171.8333
##
## $boys
## [1] 176
```

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   172.0 girl
## 7   181.0  boy
## 8   175.0  boy
## 9   174.0  boy
## 10  174.0  boy
```

```
list.ht
```

```
## $girls
## [1] 178 175 159 164 183 172
##
## $boys
## [1] 181 175 174 174
```

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

Working directory

Working with the working directory in R:

```
getwd()                  # to change, use setwd()
## [1] "/home/alex/Dropbox/Boulot/My_teaching/stat_course_izw/stat_course_beg
head(dir())              # listing first 6 files in the working directory
## [1] "figure"
## [2] "figures"
## [3] "Getting_started_with_R.nav"
## [4] "Getting_started_with_R.pdf"
## [5] "Getting_started_with_R.Rnw"
## [6] "Getting_started_with_R.snm"
dir(pattern = "*.csv")   # list only csv files
## character(0)
```

You can also use RStudio for that and setting up a "Project" may help!
File → New Project...

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 (e.g. `readxl`),
but it is easy (and 100% reliable) to use .csv files.
Excel can read and write .csv files!

Note: if you do use `readxl`, it creates a `tibble` and not a `data.frame`,
so factors won't be created which is problematic for some analyses.
So, don't forget to convert characters variables to factors using `factor()`.

Challenge #2

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

Note: RStudio can help you: File → Import Dataset → From Text (base)

Getting started with R

Graphics paradigms in R

They are three main graphics paradigms in R:

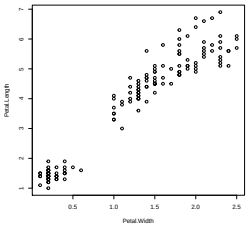
- traditional graphics (based on `graphics`), which we will briefly study
- `lattice` (based on `grid`), which is powerful but complex
- `ggplot2` (based on `grid`, from RStudio people), which is now very popular

Graphics paradigms in R

They are three main graphics paradigms in R:

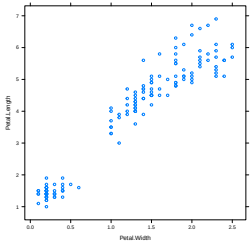
- traditional graphics (based on `graphics`), which we will briefly study
- `lattice` (based on `grid`), which is powerful but complex
- `ggplot2` (based on `grid`, from RStudio people), which is now very popular

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



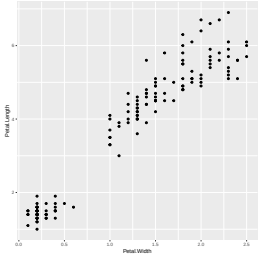

```
library(lattice)

xyplot(Petal.Length ~ Petal.Width,
       data = iris)
```




```
library(ggplot2)

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



Foreplay

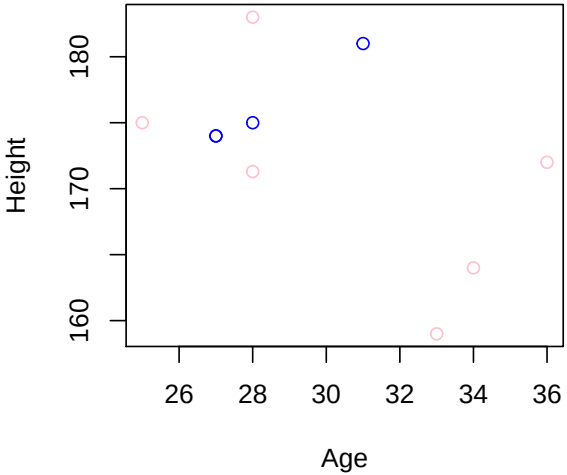
First, let's create a new column Age in our set dataframe.ht:

```
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  172.0 girl  36
```

Scatter plot

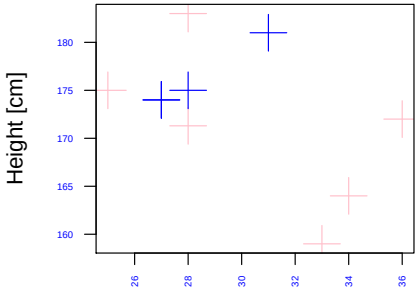
```
palette(c("blue", "pink")) ## same order as in levels(dataframe.ht$Sex)
plot(Height ~ Age, data = dataframe.ht, col = Sex)
```



Scatter plot

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

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

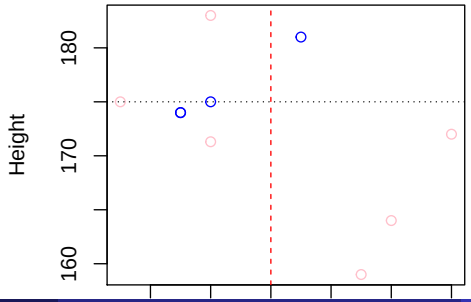


Decorating plots

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

```
?abline
?text
```

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

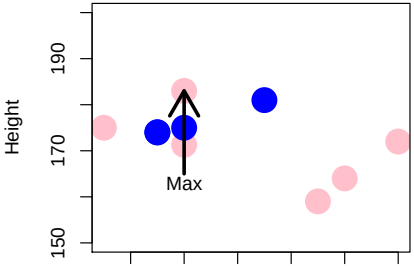


Decorating plots

Drawing arrows and adding text:

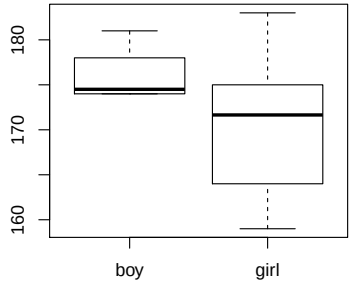
```
?arrows
?text
```

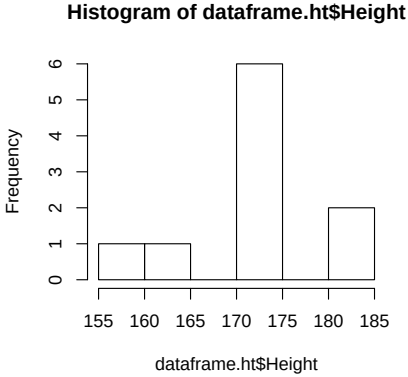
```
plot(Height ~ Age, data = dataframe.ht, col = Sex, pch = 16, cex = 3, ylim = c(150, 200))
max.value <- max(x = dataframe.ht$Height)
where.max <- dataframe.ht$Age[which.max(x = dataframe.ht$Height)]
arrows(x0 = where.max, y0 = max.value - 18,
       x1 = where.max,  y1 = max.value, col = "black", 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)
```

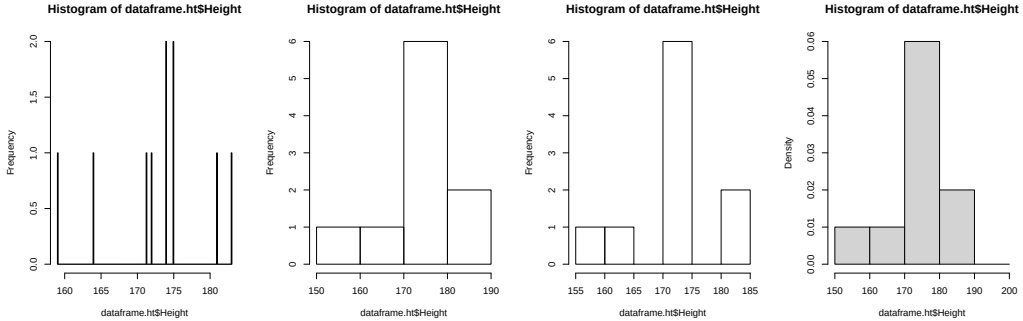




Histograms

```
?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
```

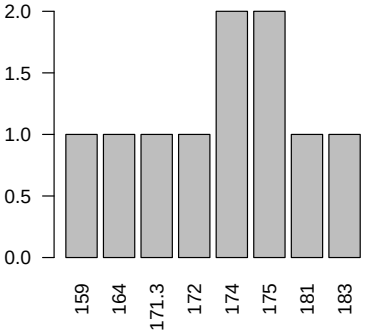
```
## $breaks
## [1] 155 160 165 170 175 180 185
##
## $counts
## [1] 1 1 0 6 0 2
##
## $density
## [1] 0.02 0.02 0.00 0.12 0.00 0.04
##
## $mids
## [1] 157.5 162.5 167.5 172.5 177.5 182.5
##
## $xname
## [1] "dataframe.ht$Height"
##
## $equidist
## [1] TRUE
##
## attr(,"class")
## [1] "histogram"
```

Bar charts

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

```
##
##   159   164 171.3   172   174   175   181   183
##     1     1     1     1     2     2     1     1
```

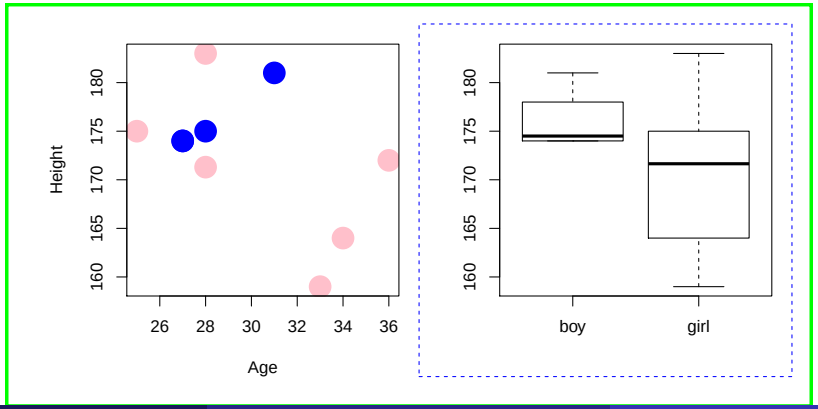
```
par(mfrow = c(1, 1), las = 2)
barplot(mybp)
```



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(Height ~ Age, data = dataframe.ht, col = Sex, pch = 16, cex = 3)
boxplot(Height ~ Sex, data = dataframe.ht)
```



Exporting figures

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

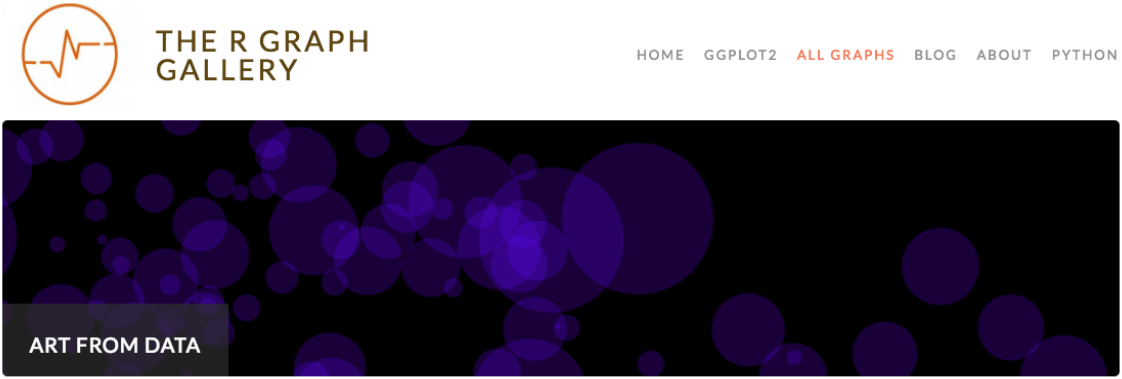
```
pdf(file = "my_plot.pdf", width = 14, height = 7)
par(mfrow = c(1, 2), mar = c(4, 4, 1, 1), oma = c(1.5, 2, 1, 1))
plot(Height ~ Age, data = dataframe.ht, col = Sex, pch = 16, cex = 3)
boxplot(Height ~ Sex, data = dataframe.ht)
dev.off() # close the pdf
getwd() # to check where the file is
```

Learning how to plot

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

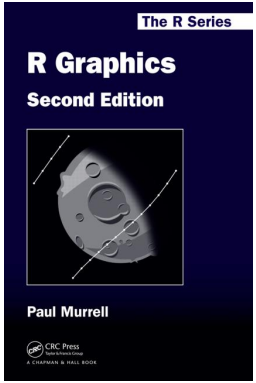
```
example(hist)
```

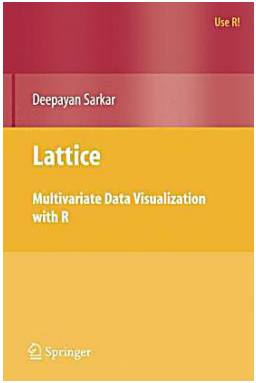
2. Scroll the web
(e.g. `http://www.r-graph-gallery.com/all-graphs/`)

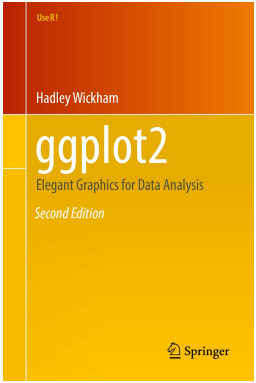


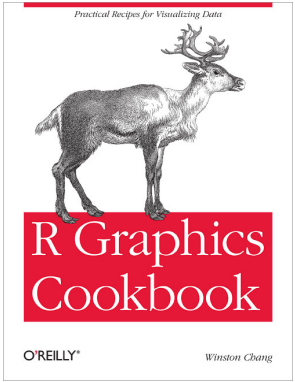
Learning how to plot

3. Read books:

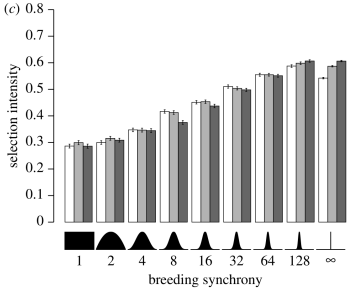


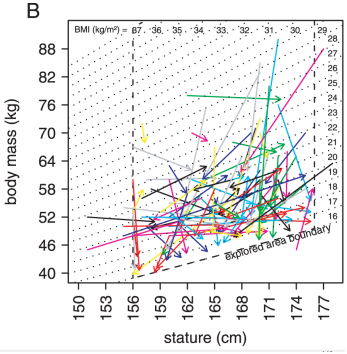


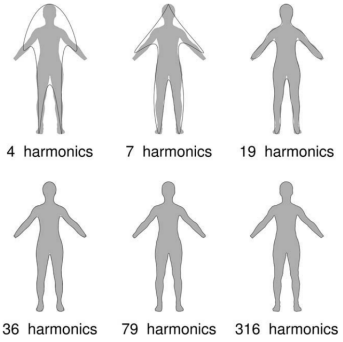




Some of my home-made R graphics







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))
```

How to get the code behind a function?

R methods (S3):

```
residuals
```

```
## function (object, ...)
## UseMethod("residuals")
## <bytecode: 0x55ed6b0068e0>
## <environment: namespace:stats>
```

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

```
methods(residuals)
```

```
##  [1] residuals.coxph*         residuals.coxph.null*    residuals.coxph.penal*
##  [4] residuals.default*       residuals.glm            residuals.HoltWinters*
##  [7] residuals.isoreg*        residuals.lm             residuals.loglm*
## [10] residuals.nls*           residuals.smooth.spline* residuals.survreg*
## [13] residuals.survreg.penal* 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: 0x55ed72c751e8>
## <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") or stats:::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"
##     }
```

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: 0x55ed747eef80>
## <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)

For more info, check: `https://stackoverflow.com/questions/19226816/how-can-i-view-the-source-code-for-a-function`.

Numerical issues common to most programming languages

```
0.7 - 0.4 - 0.3 == 0
```

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

Numerical issues common to most programming languages

```
0.7 - 0.4 - 0.3 == 0
```

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

Why?

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

```
##  [1] 0.00000000000000000000 0.10000000000000000055511 0.20000000000000000111022
##  [4] 0.30000000000000000444089 0.40000000000000000222045 0.50000000000000000000000
##  [7] 0.60000000000000000888178 0.70000000000000000666134 0.80000000000000000444089
## [10] 0.90000000000000000222045 1.00000000000000000000000
```

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

```
## [1] -5.5511151231257827021181e-17
```

NB: similar issues can happen in Excel too (`https://support.office.com/en-us/article/Set-rounding-precision-E5D707E3-07A8-4DF2-810C-218C531EB06A`)

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 = 0.7 - 0.4 - 0.3)
```

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

Getting started with R

Learning about R

Official documentation:

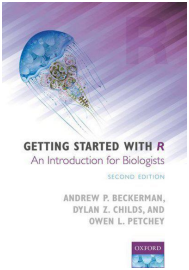
- the help files: every single (exported) function has a help file associated with it!
- official manuals (boring but thorough: `https://cran.r-project.org/manuals.html`)

Learning about R

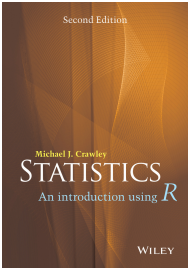
Official documentation:

- the help files: every single (exported) function has a help file associated with it!
- official manuals (boring but thorough: `https://cran.r-project.org/manuals.html`)

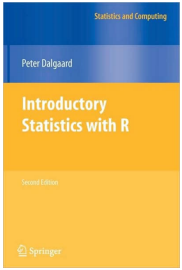
Books (roughly sorted by amount of conceptual content):



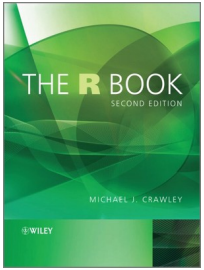
$\sim$ 30 €



$\sim$ 35 €



$\sim$ 40 €



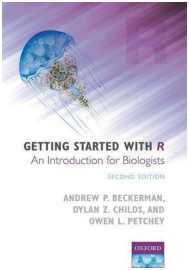
$\sim$ 60 €

Learning about R

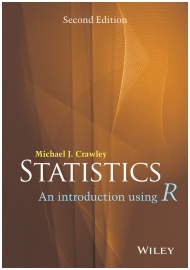
Official documentation:

- the help files: every single (exported) function has a help file associated with it!
- official manuals (boring but thorough: `https://cran.r-project.org/manuals.html`)

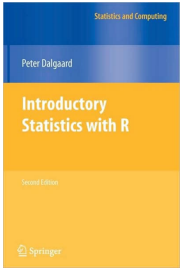
Books (roughly sorted by amount of conceptual content):



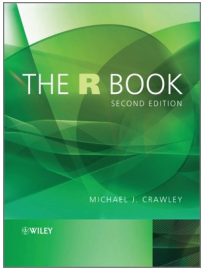
$\sim$ 30 €



$\sim$ 35 €



$\sim$ 40 €



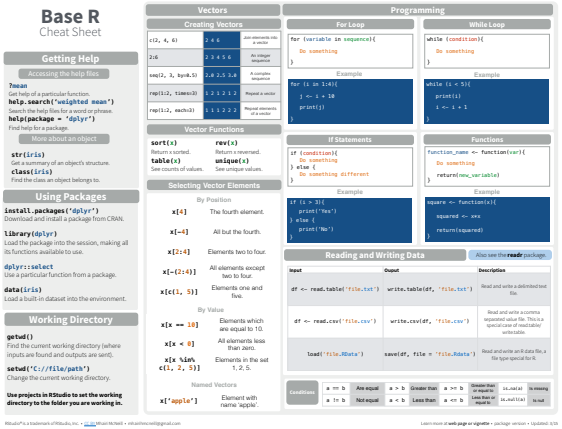
$\sim$ 60 €

Journals:

- Journal of Statistical Software (`https://www.jstatsoft.org/index`)
- The R Journal (`https://journal.r-project.org`)

Learning about R

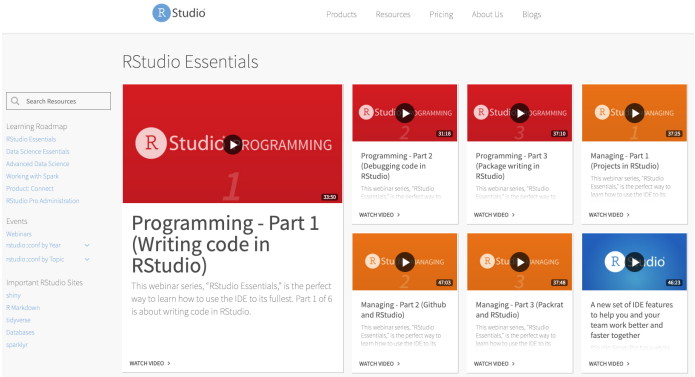
RStudio cheatsheets (`https://www.rstudio.com/resources/cheatsheets/`):



Note: there are many cheatsheets covering many aspects of R and several packages developed by RStudio!

Learning about R

RStudio webinars (`https://resources.rstudio.com/`):



Note: there are many videos covering many aspects of R and several packages developed by RStudio!

Learning about R

Blogs:

- http://www.r-bloggers.com
- https://rweekly.org
- http://blog.revolutionanalytics.com

Learning about R

Blogs:

- http://www.r-bloggers.com
- https://rweekly.org
- http://blog.revolutionanalytics.com

Forum:

- https://stackoverflow.com/questions/tagged/r
- https://stats.stackexchange.com/questions/tagged/r

Learning about R

Blogs:

- http://www.r-bloggers.com
- https://rweekly.org
- http://blog.revolutionanalytics.com

Forum:

- https://stackoverflow.com/questions/tagged/r
- https://stats.stackexchange.com/questions/tagged/r

Mailing lists:

- https://www.r-project.org/mail.html

Learning about R

Blogs:

- http://www.r-bloggers.com
- https://rweekly.org
- http://blog.revolutionanalytics.com

Forum:

- https://stackoverflow.com/questions/tagged/r
- https://stats.stackexchange.com/questions/tagged/r

Mailing lists:

- https://www.r-project.org/mail.html

Twitter:

- #rstats

Learning about R

Meetup groups:

- https://www.meetup.com/Berlin-R-Users-Group/
- https://www.meetup.com/rladies-berlin/

Learning about R

Meetup groups:

- https://www.meetup.com/Berlin-R-Users-Group/
- https://www.meetup.com/rladies-berlin/

Courses & Workshop:

- Physalia (https://www.physalia-courses.org)
- DataCamp (online: https://www.datacamp.com/courses/tech:r)

Learning about R

Meetup groups:

- https://www.meetup.com/Berlin-R-Users-Group/
- https://www.meetup.com/rladies-berlin/

Courses & Workshop:

- Physalia (https://www.physalia-courses.org)
- DataCamp (online: https://www.datacamp.com/courses/tech:r)

Conferences:

- useR (https://user2018.r-project.org)
- European R User meetings (https://erum.io)

The best person who can teach you R is YOU!

After having learned some basics, just open the console and test your understanding by performing experiments!

Do not copy and paste stuff from internet without trying to understand!!!