Getting started with R

Alexandre Courtiol @rdataberlin

Leibniz Institute of Zoo and Wildlife Research

March 2020

Leibniz Institute for Zoo and Wildlife Research

IN THE FORSCHUNGSVERBUND BERLIN E.V.

What is **R**?

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

What is **R**?

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

Key points (for this course):

- **R** includes **long-established parametric and non-parametric tests**, as well as cutting edge statistical methods

What is **R**?

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

Key points (general):

- interactive (you can explore each step of your workflow in details)

What is **R**?

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

Key points (general):

- interactive (you can explore each step of your workflow in details)
- flexible (you can create your own functions)

What is **R**?

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

Key points (general):

- interactive (you can explore each step of your workflow in details)
- flexible (you can create your own functions)
- transparent (you use scripts which allows for reproducible research)

What is **R**?

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

Key points (general):

- interactive (you can explore each step of your workflow in details)
- flexible (you can create your own functions)
- transparent (you use scripts which allows for reproducible research)
- free (`https://www.r-project.org/COPYING`)

What is **R**?

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

Key points (general):

- interactive (you can explore each step of your workflow in details)
- flexible (you can create your own functions)
- transparent (you use scripts which allows for reproducible research)
- free (`https://www.r-project.org/COPYING`)
- open (`https://github.com/wch/r-source`)

What is **R**?

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

Key points (general):

- interactive (you can explore each step of your workflow in details)
- flexible (you can create your own functions)
- transparent (you use scripts which allows for reproducible research)
- free (`https://www.r-project.org/COPYING`)
- open (`https://github.com/wch/r-source`)
- honest (`https://bugs.r-project.org/bugzilla`)

What is **R**?

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

Key points (general):

- interactive (you can explore each step of your workflow in details)
- flexible (you can create your own functions)
- transparent (you use scripts which allows for reproducible research)
- free (`https://www.r-project.org/COPYING`)
- open (`https://github.com/wch/r-source`)
- honest (`https://bugs.r-project.org/bugzilla`)
- scalable (laptop/supercomputer; local/remote; Windows/MacOS/Linux/Unix)

What is **R**?

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

Key points (general):

- interactive (you can explore each step of your workflow in details)
- flexible (you can create your own functions)
- transparent (you use scripts which allows for reproducible research)
- free (`https://www.r-project.org/COPYING`)
- open (`https://github.com/wch/r-source`)
- honest (`https://bugs.r-project.org/bugzilla`)
- scalable (laptop/supercomputer; local/remote; Windows/MacOS/Linux/Unix)
- rich (`https://rdrr.io`; `https://www.rdocumentation.org`)

What is **R**?

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

Key points (general):

- interactive (you can explore each step of your workflow in details)
- flexible (you can create your own functions)
- transparent (you use scripts which allows for reproducible research)
- free (`https://www.r-project.org/COPYING`)
- open (`https://github.com/wch/r-source`)
- honest (`https://bugs.r-project.org/bugzilla`)
- scalable (laptop/supercomputer; local/remote; Windows/MacOS/Linux/Unix)
- rich (`https://rdrr.io`; `https://www.rdocumentation.org`)
- up to date (`http://dirk.eddelbuettel.com/cranberries/cran/updated`)

What is **R**?

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

Key points (general):

- interactive (you can explore each step of your workflow in details)
- flexible (you can create your own functions)
- transparent (you use scripts which allows for reproducible research)
- free (`https://www.r-project.org/COPYING`)
- open (`https://github.com/wch/r-source`)
- honest (`https://bugs.r-project.org/bugzilla`)
- scalable (laptop/supercomputer; local/remote; Windows/MacOS/Linux/Unix)
- rich (`https://rdrr.io`; `https://www.rdocumentation.org`)
- up to date (`http://dirk.eddelbuettel.com/cranberries/cran/updated`)
- friendly community (more on that later)

Is **R** good for you?

R is good for:

- data manipulation
- statistical analyses
- plotting
- programming around data

Is **R** good for you?

R is good for:

- data manipulation
- statistical analyses
- plotting
- programming around data

R is not optimal for:

- data entry
- formal algebra
- general-purpose programming
- beginners (yet, getting easier)

Getting started with **R**

1. Installation
2. RStudio
3. Basics of the language
4. Packages
5. Manipulating data
6. Importing data
7. Plotting
8. Programming
9. Housekeeping
10. Learning on your own

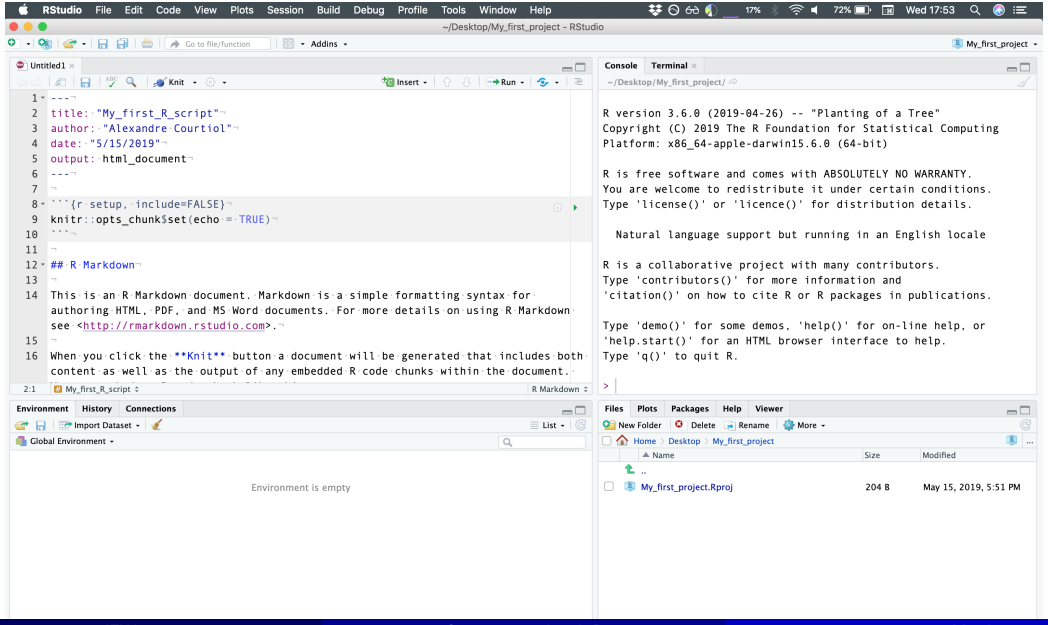
Getting started with **R**

Installation steps

1. connect to the WIFI network `izw-gast` (check password on blackboard)
2. check that you do get internet access
3. install **R**: `https://cran.r-project.org/`
4. install RStudio: `https://www.rstudio.com/products/rstudio/download/`
5. open RStudio

Getting started with **R**

The RStudio Integrated Development Environment (IDE)



Why using RStudio?

You can use **R** without RStudio, but RStudio is:

- provided with more functionalities than the official **R** IDE
- integrated with several packages developed by RStudio (e.g. `{readr}`)
- suitable for both desktop and remote computers (web server)
- free and open source

Overall structure

- Top menu
- Tool bar (small subset from the top menu)
- 4 visible panes

The top menu

File Edit Code View Plots Session Build Debug Profile Tools Window Help

10 most useful items:

- `File` → `New File`
- `File` → `New Project...`
- `File` → `Import Dataset`
- `File` → `Save`
- `Edit` → `Undo`
- `Edit` → `Clear console`
- `Code` → `Comment/Uncomment Lines`
- `Tools` → `Global Options...`
- `Help` → `Check for Updates` (for updating RStudio, not **R**, nor **R** packages)
- `Help` → `Cheatsheets`

Practice

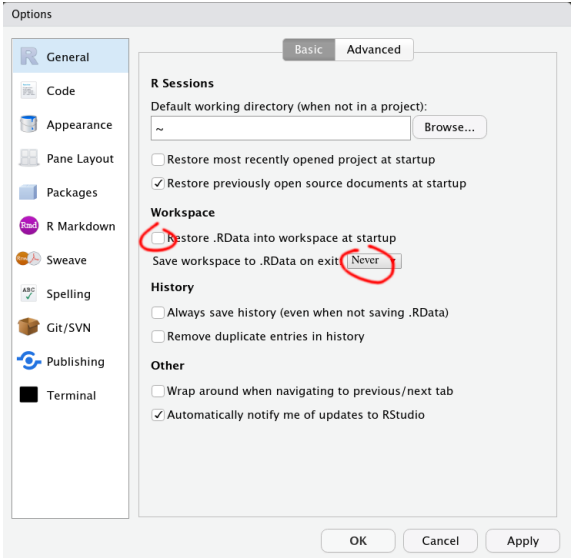
1. create a new project
2. create a new **R** Script file
3. save the created **R** Script file into the project folder directory
4. have a look at the RStudio cheatsheet

NB:

- a project is defined by a simple (text) file. Its main benefit is to open RStudio with the correct working directory set (that is, the one where the project file is)
- an **R** Script file is a (text) file where we can write **R** code

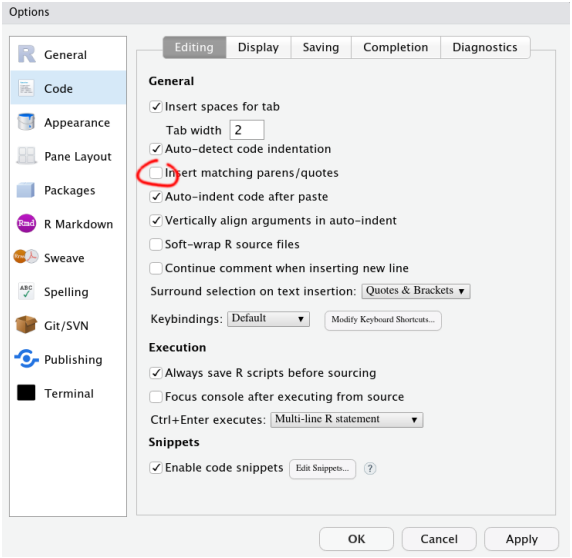
My tips about the global options

1. I never save or restore the workspace and so should you (default settings are DANGEROUS)



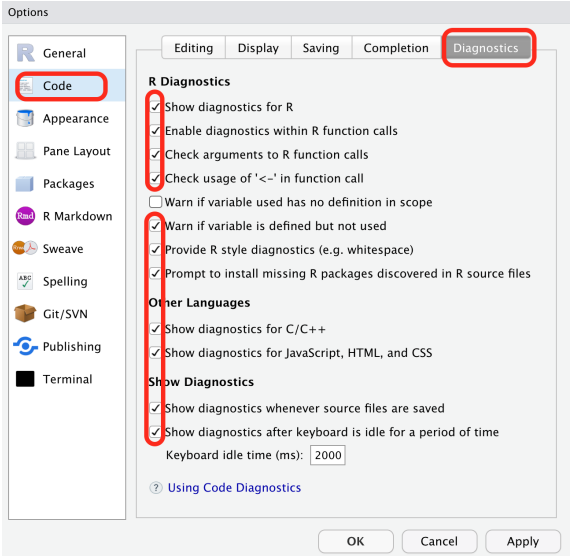
My tips about the global options

2. I don't like parentheses and quotes auto-completion, but you may



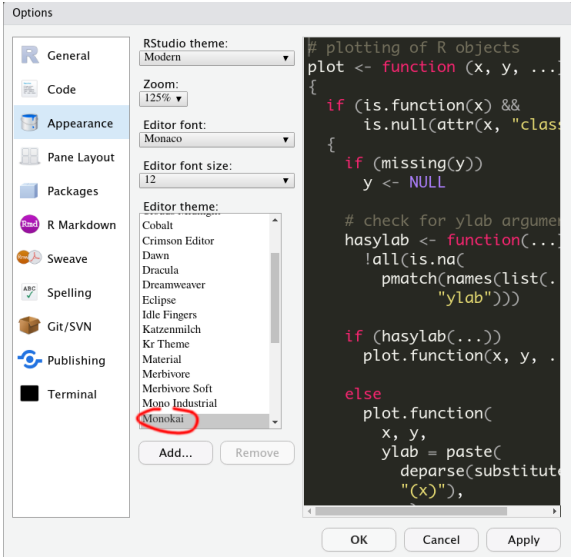
My tips about the global options

3. I activate all the code diagnostics offered by RStudio!



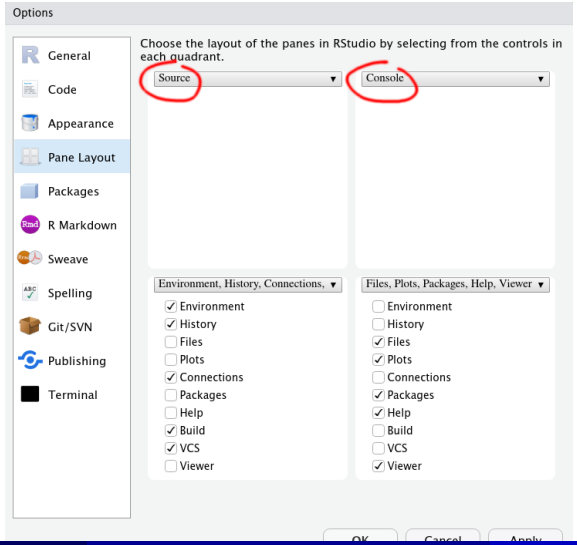
My tips about the global options

4. I use a black theme except when using a video projector (do as you please)



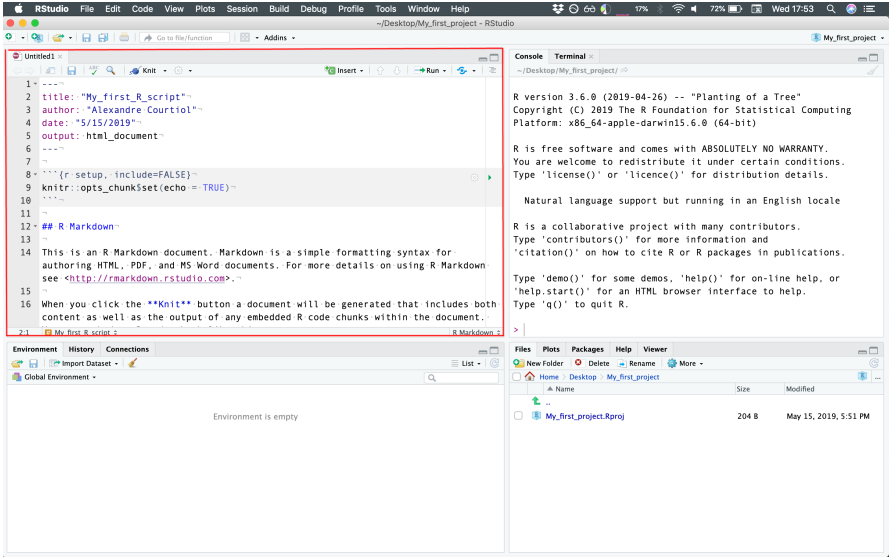
My tips about the global options

5. I like to put the `source` and the `console` panes side-by-side to give them more vertical space on the screen



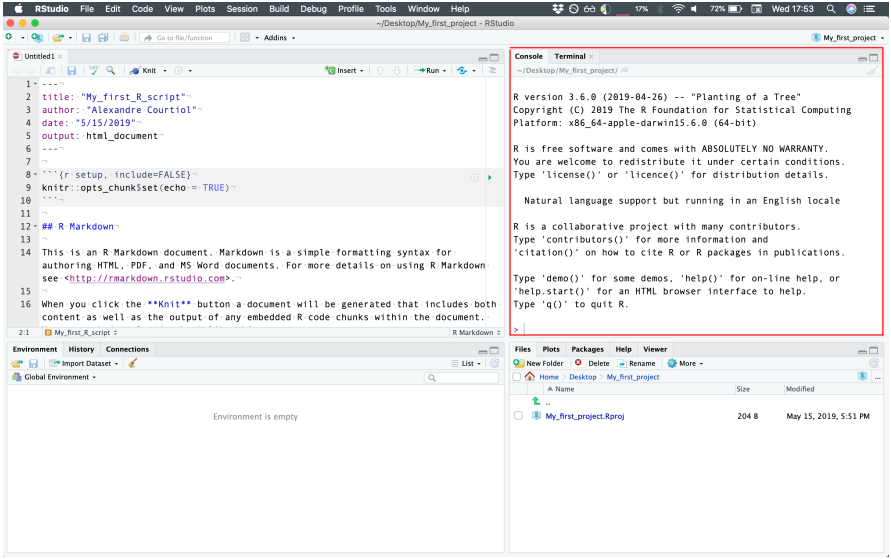
The Source pane

This is where you type the code you want to keep



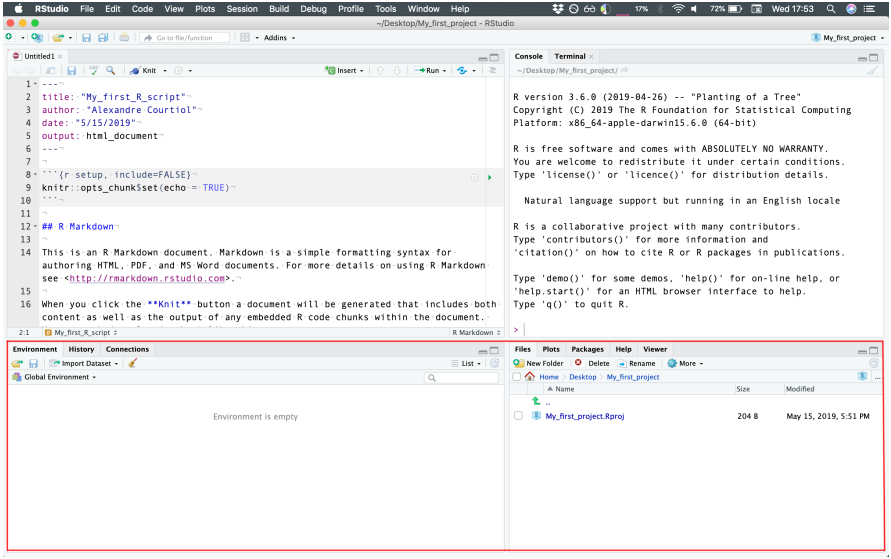
The Console pane

This is where you can run **R** code directly



The other panes

This is where everything else is



Tips

1. only use the `Console` pane to mess around
2. write proper **R** code in the `Source` pane
3. start your script with `rm(list = ls())`
4. comment thoroughly your **R** script using # signs
5. re-run frequently your entire script to make sure that it works

Getting started with **R**

Arithmetic & Logic

R can perform basic arithmetic:

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

Arithmetic & Logic

R can perform basic arithmetic:

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

R can perform logical operations:

```
1 == 1
## [1] TRUE
(1 == 1) & (1 == 2)
## [1] FALSE
(1 == 1) | (1 == 2)
## [1] TRUE
1 != 2
## [1] TRUE
!(1 == 2)
## [1] TRUE
2 >= 1
## [1] TRUE
2 < 1
## [1] FALSE
```

Arithmetic & Logic

R can perform basic arithmetic:

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

R can perform logical operations:

```
1 == 1
## [1] TRUE
(1 == 1) & (1 == 2)
## [1] FALSE
(1 == 1) | (1 == 2)
## [1] TRUE
1 != 2
## [1] TRUE
!(1 == 2)
## [1] TRUE
2 >= 1
## [1] TRUE
2 < 1
## [1] FALSE
```

As for most other programming languages, avoid equality tests for floating-point numbers!

```
0.8 - 0.3 - 0.5 == 0.8 - 0.5 - 0.3
## [1] FALSE
## check later ?all.equal() and ?dplyr::near()
```

Practice

Compute $\sqrt{\frac{2^{3+1}}{\frac{4}{5 \times 6}} - 20}$

Creating objects

Objects are being assigned using the “arrow” operator:

```
one_plus_one <- 1 + 1 ## storing the result
```

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
```

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
```

Tips:

```
(one_times_two <- 1 * 2) ## storing and displaying the result at once
## [1] 2
```

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
```

Tips:

```
(one_times_two <- 1 * 2) ## storing and displaying the result at once
## [1] 2
```

- -> works too (if you swith the lefthand side and the righthand side)

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
```

Tips:

```
(one_times_two <- 1 * 2) ## storing and displaying the result at once
## [1] 2
```

- -> works too (if you swith the lefthand side and the righthand side)
- "_" and "." are OK but avoid spaces & other weird characters in names

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
```

Tips:

```
(one_times_two <- 1 * 2) ## storing and displaying the result at once
## [1] 2
```

- -> works too (if you swith the lefthand side and the righthand side)
- "_" and "." are OK but avoid spaces & other weird characters in names
- names are case sensitive

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
```

Tips:

```
(one_times_two <- 1 * 2) ## storing and displaying the result at once
## [1] 2
```

- -> works too (if you swith the lefthand side and the righthand side)
- "_" and "." are OK but avoid spaces & other weird characters in names
- names are case sensitive
- use <- for assignments, not =

Functions

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

```
##
## To cite R in publications use:
##
##   R Core Team (2019). 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 = {2019},
##     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.
```

Functions

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

```
##
## To cite R in publications use:
##
##   R Core Team (2019). 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 = {2019},
##     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.
```

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

```r
?citation() ## same but shorter (syntactic sugar)
```

Functions

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

```
##
## To cite R in publications use:
##
##   R Core Team (2019). 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 = {2019},
##     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.
```

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

```r
?citation() ## same but shorter (syntactic sugar)
```

NB: best to look at the help before using a function new to you!

Practice

Display the path of any file you want using `file.choose()`

Syntax for functions that take arguments

Basic syntax:

```
sign(x = -5)
## [1] -1
sign(-5)
## [1] -1
```

Syntax for functions that take arguments

Basic syntax:

```
sign(x = -5)
## [1] -1
sign(-5)
## [1] -1
```

Both are synonymous, but:

- the first syntax is safer
- the second syntax removes visual clutter

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

All these calls are similar:

```
mean(x = c(1, 5, 3, 4))
## [1] 3.25
```

```
mean(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
```

Practice

```r
new.vector <- c(1, 2, 3, NA, 5)
```

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

Finding functions

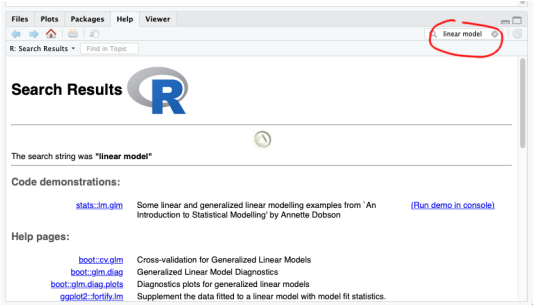
To find the name of the function you are look for, you may try:

```
??"linear model" ## if you want something that fits a linear model
```

or

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

or



NB: this is failing on some 1.2.xxx versions of RStudio

Getting started with **R**

The concept of an **R** package

Packages extend **R** functionalities:

- for most users; e.g. `{dplyr}`, `{ggplot2}`
- for specific users; e.g. `{IsoriX}`, `{hyenaR}`
- for developers; e.g. `{devtools}`, `{Rcpp}`

The concept of an **R** package

Packages extend **R** functionalities:

- for most users; e.g. `{dplyr}`, `{ggplot2}`
- for specific users; e.g. `{IsoriX}`, `{hyenaR}`
- for developers; e.g. `{devtools}`, `{Rcpp}`

Key facts about packages:

- a package is just a folder (often compressed) containing functions, data & documentation
- a library is the installed version of the package (also a folder)
- there are tons of packages out there:
  `https://rdrr.io`; `https://www.rdocumentation.org`

Installing a package

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

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

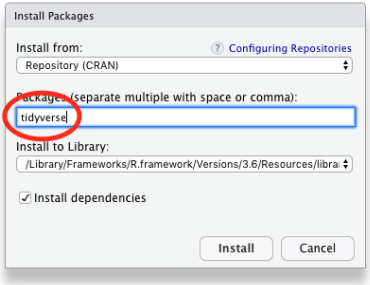
Tip: in order to be able to install packages that require compilation (and thus have access to more or newer versions of packages), you need to install:

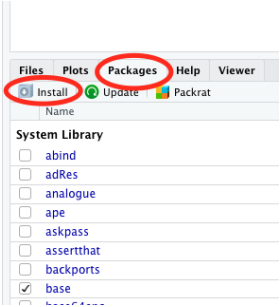
- `Rtools` if you use Windows (`https://cran.r-project.org/bin/windows/Rtools`)
- `clang` and `gfortran` (`https://cran.r-project.org/bin/macosx/tools`) or `Xcode` (`https://developer.apple.com/xcode`) if you use macOS
- if you use Linux or Unix-based system, you should already have everything you need

Installing a package

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

You can use RStudio:

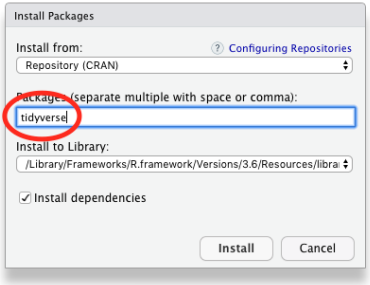


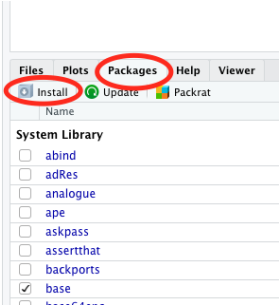


Installing a package

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

You can use RStudio:





or you can type in the `Console` pane:

```
install.packages("tidyverse")  ## install {tidyverse}
```

Practice

Install the {tidyverse}!

Loading a library

That is always simple:

```
library(tidyverse)
## - Attaching packages -------------- tidyverse 1.3.0 -
## v ggplot2 3.2.1            v purrr  0.3.3
## v tibble  2.99.99.9014     v dplyr  0.8.99.9000
## v tidyr   1.0.2            v stringr 1.4.0
## v readr   1.3.1            v forcats 0.4.0
## - Conflicts ----------------- tidyverse_conflicts() -
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

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!

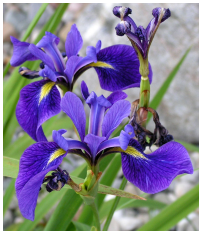
Getting started with **R**

The iris dataset

The iris dataset is readily available in **R**; it is often used to illustrate statistical procedures (https://en.wikipedia.org/wiki/Iris_flower_data_set)

Iris setosa

©Miya.m



Iris versicolor

©D.G.E. Robertson

Iris virginica

©F. Mayfield

```
colnames(iris)  ## shows the column names of the iris dataset
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"
```

Data representation in **R**

The most common class of objects used for storing data in **R** is the data frame!

Data representation in **R**

The most common class of objects used for storing data in **R** is the data frame!

Example: the `iris` dataset

```
iris
##     Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
## 1            5.1         3.5          1.4         0.2     setosa
## 2            4.9         3.0          1.4         0.2     setosa
## 3            4.7         3.2          1.3         0.2     setosa
## 4            4.6         3.1          1.5         0.2     setosa
## 5            5.0         3.6          1.4         0.2     setosa
## 6            5.4         3.9          1.7         0.4     setosa
## 7            4.6         3.4          1.4         0.3     setosa
## 8            5.0         3.4          1.5         0.2     setosa
## 9            4.4         2.9          1.4         0.2     setosa
## 10           4.9         3.1          1.5         0.1     setosa
## 11           5.4         3.7          1.5         0.2     setosa
## 12           4.8         3.4          1.6         0.2     setosa
## 13           4.8         3.0          1.4         0.1     setosa
## 14           4.3         3.0          1.1         0.1     setosa
## 15           5.8         4.0          1.2         0.2     setosa
## 16           5.7         4.4          1.5         0.4     setosa
## 17           5.4         3.9          1.3         0.4     setosa
## 18           5.1         3.5          1.4         0.3     setosa
```

Data representation in **R**

The most common class of objects used for storing data in **R** is the data frame!

Example: the `iris` dataset

```
head(iris)
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

Data representation in **R**

The most common class of objects used for storing data in **R** is the data frame!

Example: the `iris` dataset

```
str(iris)
## 'data.frame': 150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

Data representation in **R**

The most common class of objects used for storing data in **R** is the data frame!

- each column is a variable, usually corresponding to a vector (a serie of elements of 1 type)
- all columns have the same length (rectangular format)
- contains column names (usually informative) and row names (usually not informative)

Data frames come in 2 flavours

original data frames (class `data.frame`)

- perfectly fine to work with
- don't rely on extra packages

modern data frames: tibbles (class `tbl_df`)

- improved display (see `?formatting`)
- lazier & safer (see `?`tbl_df-class``)
- more flexible (work well with groupings)

Data frames come in 2 flavours

original data frames (class `data.frame`)

- perfectly fine to work with
- don't rely on extra packages

modern data frames: tibbles (class `tbl_df`)

- improved display (see `?formatting`)
- lazier & safer (see `?`tbl_df-class``)
- more flexible (work well with groupings)

Tip: if some function struggle with tibble, try using original data frames

Tibbles

You can easily turn an original data frame into a tibble

```
library(tidyverse)
iris_tbl <- as_tibble(iris) ## as.data.frame() does the opposite: tbl -> df
iris_tbl
## # A tibble: 150 x 5
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##          <dbl>       <dbl>        <dbl>       <dbl> <fct>
## 1          5.1         3.5          1.4         0.2 setosa
## 2          4.9         3            1.4         0.2 setosa
## 3          4.7         3.2          1.3         0.2 setosa
## # ... with 147 more rows
```

Tibbles

You can easily turn an original data frame into a tibble

```
library(tidyverse)
iris_tbl <- as_tibble(iris) ## as.data.frame() does the opposite: tbl -> df
iris_tbl
## # A tibble: 150 x 5
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##           <dbl>       <dbl>        <dbl>       <dbl> <fct>
## 1          5.1         3.5          1.4         0.2 setosa
## 2          4.9         3            1.4         0.2 setosa
## 3          4.7         3.2          1.3         0.2 setosa
## # ... with 147 more rows
```

Note: the tidyverse is a meta package that loads several packages we are going to use

Working with data frames

There are two competing paradigms for manipulating data in **R**:

Working with data frames

There are two competing paradigms for manipulating data in **R**:

Vector-focussed

- easier for developers
- more error prone, but easier to debug
- more efficient (often)
- dominant paradigm for first generations of **R** users

Working with data frames

There are two competing paradigms for manipulating data in **R**:

Vector-focussed

- easier for developers
- more error prone, but easier to debug
- more efficient (often)
- dominant paradigm for first generations of **R** users

Data frame-focussed

- easier for users
- closer to Excel, SPSS, ...
- *functional programming* friendly
- dominant paradigm for new generations of **R** users

Vector based approach

Goal: computing the mean and SD of sepal length in mm for per species

```
## create a new column storing the sepal length in mm
iris$Sepal.Length.mm <- iris$Sepal.Length * 10

## create data frame to store results
results <- data.frame(Species = unique(iris$Species))

## compute the means and SD per species
for (sp in results$Species) {
  results$meanSL[results$Species == sp] <- mean(iris$Sepal.Length.mm[iris$Species == sp])
  results$sdSL[results$Species == sp]   <- sd(iris$Sepal.Length.mm[iris$Species == sp])
}

## display the result
results
##      Species meanSL     sdSL
## 1     setosa  50.06 3.524897
## 2 versicolor  59.36 5.161711
## 3  virginica  65.88 6.358796
```

Data frame based approach

Goal: computing the mean and sd of sepal length in mm for per species

```
## create a new column storing the sepal length in mm
iris <- mutate(iris, Sepal.Length.mm = Sepal.Length * 10)

## declare that you will perform operations within species
iris <- group_by(iris, Species)

## compute the means and SD per species
results <- summarise(iris, meanSL = mean(Sepal.Length.mm),
                           sdSL   = sd(Sepal.Length.mm))

## display the result
results
## # A tibble: 3 x 3
##   Species    meanSL  sdSL
##   <fct>       <dbl> <dbl>
## 1 setosa       50.1  3.52
## 2 versicolor   59.4  5.16
## 3 virginica    65.9  6.36
```

The {tidyverse}

The tidyverse packages (`https://www.tidyverse.org`) are developed by RStudio

There are 8 core tidyverse packages:

The {tidyverse}

The tidyverse packages (`https://www.tidyverse.org`) are developed by RStudio

There are 8 core tidyverse packages:

- philosophy: making **R** more accessible and more modern
- more functions, more focussed: 1 function = 1 action = 1 verb
- backward compatibility is not the absolute priority
- financed and controlled by RStudio

Data frame based approach with pipes (%>%) from {magrittr}

Goal: computing the mean and SD of sepal length in mm for per species

```
iris %>%
  mutate(Sepal.Length.mm = Sepal.Length * 10) %>%
  group_by(Species) %>%
  summarise(meanSL = mean(Sepal.Length.mm),
            sdSL   = sd(Sepal.Length.mm)) -> results
results
## # A tibble: 3 x 3
##   Species    meanSL  sdSL
##   <fct>       <dbl> <dbl>
## 1 setosa       50.1  3.52
## 2 versicolor   59.4  5.16
## 3 virginica    65.9  6.36
```

Data frame based approach with pipes (%>%) from {magrittr}

Goal: computing the mean and SD of sepal length in mm for per species

```
iris %>%
  mutate(Sepal.Length.mm = Sepal.Length * 10) %>%
  group_by(Species) %>%
  summarise(meanSL = mean(Sepal.Length.mm),
            sdSL   = sd(Sepal.Length.mm)) -> results
results
## # A tibble: 3 x 3
##   Species    meanSL  sdSL
##   <fct>       <dbl> <dbl>
## 1 setosa       50.1  3.52
## 2 versicolor   59.4  5.16
## 3 virginica    65.9  6.36
```

Note: pipes capture what is on their left and forward this as the first argument of the function that comes on their right

Manipulating data frames with {dplyr}

You can do a lot with 5 functions:

Manipulating data frames with {dplyr}

You can do a lot with 5 functions:

- `select()` to keep or discard columns
- `group_by()` to define groups of rows for the following verbs
- `filter()` to keep or discard rows
- `mutate()` to create new columns
- `summarise()` to compute summary statistics

Manipulating data frames with {dplyr}

You can do a lot with 5 functions:

- `select()` to keep or discard columns
- `group_by()` to define groups of rows for the following verbs
- `filter()` to keep or discard rows
- `mutate()` to create new columns
- `summarise()` to compute summary statistics

Note: this is much more than that in {dplyr}

select()

Keep columns using `select()`:

```
iris_tbl %>%
  select(Sepal.Length, Sepal.Width)
## # A tibble: 150 x 2
##   Sepal.Length Sepal.Width
##          <dbl>       <dbl>
## 1          5.1         3.5
## 2          4.9         3
## 3          4.7         3.2
## # ... with 147 more rows
```

select()

Keep columns using `select()`:

```
iris_tbl %>%
  select(Sepal.Length, Sepal.Width)
## # A tibble: 150 x 2
##   Sepal.Length Sepal.Width
##          <dbl>       <dbl>
## 1          5.1         3.5
## 2          4.9         3
## 3          4.7         3.2
## # ... with 147 more rows
```

Discard columns using `select()`:

```
iris_tbl %>%
  select(-Sepal.Length, -Sepal.Width)
## # A tibble: 150 x 3
##   Petal.Length Petal.Width Species
##          <dbl>       <dbl> <fct>
## 1          1.4         0.2 setosa
## 2          1.4         0.2 setosa
## 3          1.3         0.2 setosa
## # ... with 147 more rows
```

filter()

Keep rows using `filter()`:

```
iris_tbl %>% filter(Sepal.Length > 7, Sepal.Width > 3)
## # A tibble: 4 x 5
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##          <dbl>       <dbl>        <dbl>       <dbl> <fct>
## 1          7.2         3.6          6.1         2.5 virginica
## 2          7.7         3.8          6.7         2.2 virginica
## 3          7.2         3.2          6           1.8 virginica
## # ... with 1 more row
```

filter()

Keep rows using `filter()`:

```
iris_tbl %>% filter(Sepal.Length > 7, Sepal.Width > 3)
## # A tibble: 4 x 5
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##          <dbl>       <dbl>        <dbl>       <dbl> <fct>
## 1          7.2         3.6          6.1         2.5 virginica
## 2          7.7         3.8          6.7         2.2 virginica
## 3          7.2         3.2          6           1.8 virginica
## # ... with 1 more row
```

Keep rows using `group_by() %>% filter()`:

```
iris_tbl %>%  group_by(Species) %>%  filter(Sepal.Length == max(Sepal.Length)) %>%  group_by()
## # A tibble: 3 x 5
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##          <dbl>       <dbl>        <dbl>       <dbl> <fct>
## 1          5.8         4            1.2         0.2 setosa
## 2          7           3.2          4.7         1.4 versicolor
## 3          7.9         3.8          6.4         2   virginica
```

filter()

Keep rows using `filter()`:

```
iris_tbl %>% filter(Sepal.Length > 7, Sepal.Width > 3)
## # A tibble: 4 x 5
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##          <dbl>       <dbl>        <dbl>       <dbl> <fct>
## 1          7.2         3.6          6.1         2.5 virginica
## 2          7.7         3.8          6.7         2.2 virginica
## 3          7.2         3.2          6           1.8 virginica
## # ... with 1 more row
```

Keep rows using `group_by() %>% filter()`:

```
iris_tbl %>%  group_by(Species) %>%  filter(Sepal.Length == max(Sepal.Length)) %>%  group_by()
## # A tibble: 3 x 5
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##          <dbl>       <dbl>        <dbl>       <dbl> <fct>
## 1          5.8         4            1.2         0.2 setosa
## 2          7           3.2          4.7         1.4 versicolor
## 3          7.9         3.8          6.4         2   virginica
```

Tip: ungrouping prevents further (unwanted) grouped operations

mutate()

Create a column using `mutate()`:

```
iris_tbl %>% mutate(Sepal.Length.mm = Sepal.Length * 10)
## # A tibble: 150 x 6
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species Sepal.Length.mm
##          <dbl>       <dbl>        <dbl>       <dbl> <fct>              <dbl>
## 1          5.1         3.5          1.4         0.2 setosa                51
## 2          4.9         3            1.4         0.2 setosa                49
## 3          4.7         3.2          1.3         0.2 setosa                47
## # ... with 147 more rows
```

mutate()

Create a column using `mutate()`:

```
iris_tbl %>% mutate(Sepal.Length.mm = Sepal.Length * 10)
## # A tibble: 150 x 6
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species Sepal.Length.mm
##          <dbl>       <dbl>        <dbl>       <dbl> <fct>              <dbl>
## 1          5.1         3.5          1.4         0.2 setosa              51
## 2          4.9         3            1.4         0.2 setosa              49
## 3          4.7         3.2          1.3         0.2 setosa              47
## # ... with 147 more rows
```

Create a column using `group_by() %>% mutate()`:

```
iris_tbl %>% group_by(Species) %>% mutate(Sepal.Length.max = max(Sepal.Length)) %>% group_by()
## # A tibble: 150 x 6
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species Sepal.Length.max
##          <dbl>       <dbl>        <dbl>       <dbl> <fct>              <dbl>
## 1          5.1         3.5          1.4         0.2 setosa               5.8
## 2          4.9         3            1.4         0.2 setosa               5.8
## 3          4.7         3.2          1.3         0.2 setosa               5.8
## # ... with 147 more rows
```

mutate()

Create a column using `mutate()`:

```
iris_tbl %>% mutate(Sepal.Length.mm = Sepal.Length * 10)
## # A tibble: 150 x 6
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species Sepal.Length.mm
##          <dbl>       <dbl>        <dbl>       <dbl> <fct>              <dbl>
## 1          5.1         3.5          1.4         0.2 setosa              51
## 2          4.9         3            1.4         0.2 setosa              49
## 3          4.7         3.2          1.3         0.2 setosa              47
## # ... with 147 more rows
```

Create a column using `group_by() %>% mutate()`:

```
iris_tbl %>% group_by(Species) %>% mutate(Sepal.Length.max = max(Sepal.Length)) %>% group_by()
## # A tibble: 150 x 6
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species Sepal.Length.max
##          <dbl>       <dbl>        <dbl>       <dbl> <fct>              <dbl>
## 1          5.1         3.5          1.4         0.2 setosa               5.8
## 2          4.9         3            1.4         0.2 setosa               5.8
## 3          4.7         3.2          1.3         0.2 setosa               5.8
## # ... with 147 more rows
```

Note: `mutate()` only accepts functions that return a vector of length 1 or of length suitable to create a new column

summarise()

Compute summary statistics using `summarise()`:

```
iris_tbl %>% summarise(Sepal.Length.max = max(Sepal.Length), N = n())
## # A tibble: 1 x 2
##   Sepal.Length.max     N
##              <dbl> <int>
## 1              7.9   150
```

summarise()

Compute summary statistics using `summarise()`:

```
iris_tbl %>% summarise(Sepal.Length.max = max(Sepal.Length), N = n())
## # A tibble: 1 x 2
##   Sepal.Length.max     N
##              <dbl> <int>
## 1              7.9   150
```

Compute summary statistics using `group_by() %>% summarise()`:

```
iris_tbl %>% group_by(Species) %>% summarise(Sepal.Length.max = max(Sepal.Length), N = n())
## # A tibble: 3 x 3
##   Species    Sepal.Length.max     N
##   <fct>                 <dbl> <int>
## 1 setosa                  5.8    50
## 2 versicolor              7      50
## 3 virginica               7.9    50
```

summarise()

Compute summary statistics using summarise():

```
iris_tbl %>% summarise(Sepal.Length.max = max(Sepal.Length), N = n())
## # A tibble: 1 x 2
##   Sepal.Length.max     N
##              <dbl> <int>
## 1              7.9   150
```

Compute summary statistics using group_by() %>% summarise():

```
iris_tbl %>% group_by(Species) %>% summarise(Sepal.Length.max = max(Sepal.Length), N = n())
## # A tibble: 3 x 3
##   Species    Sepal.Length.max     N
##   <fct>                 <dbl> <int>
## 1 setosa                  5.8    50
## 2 versicolor              7      50
## 3 virginica               7.9    50
```

Note: summarise() only accepts functions that return a vector of length 1

Tip: summarise() ungroup automatically

Practice

Assuming petals are rectangular, compute:

- the mean...
- the median...
- the minimum...
- the maximum...

...area of petals for each species

Lists

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

```
iris %>% filter(Species %in% c("setosa", "versicolor")) -> iris2sp
test <- t.test(Sepal.Length ~ Species, data = iris2sp)
```

Lists

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

```
iris %>% filter(Species %in% c("setosa", "versicolor")) -> iris2sp
test <- t.test(Sepal.Length ~ Species, data = iris2sp)
```

```
str(test)
## List of 10
##  $ statistic  : Named num -10.5
##   ..- attr(*, "names")= chr "t"
##  $ parameter  : Named num 86.5
##   ..- attr(*, "names")= chr "df"
##  $ p.value    : num 3.75e-17
##  $ conf.int   : num [1:2] -1.106 -0.754
##   ..- attr(*, "conf.level")= num 0.95
##  $ estimate   : Named num [1:2] 5.01 5.94
##   ..- attr(*, "names")= chr [1:2] "mean in group setosa" "mean in group versicolor"
##  $ null.value : Named num 0
##   ..- attr(*, "names")= chr "difference in means"
##  $ stderr     : num 0.0884
##  $ alternative: chr "two.sided"
##  $ method     : chr "Welch Two Sample t-test"
##  $ data.name  : chr "Sepal.Length by Species"
##  - attr(*, "class")= chr "htest"
```

Lists

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

```
iris %>% filter(Species %in% c("setosa", "versicolor")) -> iris2sp
test <- t.test(Sepal.Length ~ Species, data = iris2sp)
```

```
str(test)
## List of 10
##  $ statistic  : Named num -10.5
##   ..- attr(*, "names")= chr "t"
##  $ parameter  : Named num 86.5
##   ..- attr(*, "names")= chr "df"
##  $ p.value    : num 3.75e-17
##  $ conf.int   : num [1:2] -1.106 -0.754
##   ..- attr(*, "conf.level")= num 0.95
##  $ estimate   : Named num [1:2] 5.01 5.94
##   ..- attr(*, "names")= chr [1:2] "mean in group setosa" "mean in group versicolor"
##  $ null.value : Named num 0
##   ..- attr(*, "names")= chr "difference in means"
##  $ stderr     : num 0.0884
##  $ alternative: chr "two.sided"
##  $ method     : chr "Welch Two Sample t-test"
##  $ data.name  : chr "Sepal.Length by Species"
##  - attr(*, "class")= chr "htest"
```

There are different ways to extract an element from a list:

```
test[["p.value"]]
## [1] 3.746743e-17
test$p.value
## [1] 3.746743e-17
pluck(test, "p.value") # {purrr}
## [1] 3.746743e-17
```

Lists

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

```
iris %>% filter(Species %in% c("setosa", "versicolor")) -> iris2sp
test <- t.test(Sepal.Length ~ Species, data = iris2sp)
```

```
str(test)
## List of 10
##  $ statistic  : Named num -10.5
##   ..- attr(*, "names")= chr "t"
##  $ parameter  : Named num 86.5
##   ..- attr(*, "names")= chr "df"
##  $ p.value    : num 3.75e-17
##  $ conf.int   : num [1:2] -1.106 -0.754
##   ..- attr(*, "conf.level")= num 0.95
##  $ estimate   : Named num [1:2] 5.01 5.94
##   ..- attr(*, "names")= chr [1:2] "mean in group setosa" "mean in group versicolor"
##  $ null.value : Named num 0
##   ..- attr(*, "names")= chr "difference in means"
##  $ stderr     : num 0.0884
##  $ alternative: chr "two.sided"
##  $ method     : chr "Welch Two Sample t-test"
##  $ data.name  : chr "Sepal.Length by Species"
##  - attr(*, "class")= chr "htest"
```

There are different ways to extract an element from a list:

```
test[["p.value"]]
## [1] 3.746743e-17
test$p.value
## [1] 3.746743e-17
pluck(test, "p.value") # {purrr}
## [1] 3.746743e-17
```

Note: lists are necessary because, in **R**, functions cannot output several objects are once

Getting started with **R**

Importing data in **R**

R can read (and write) many formats storing data:

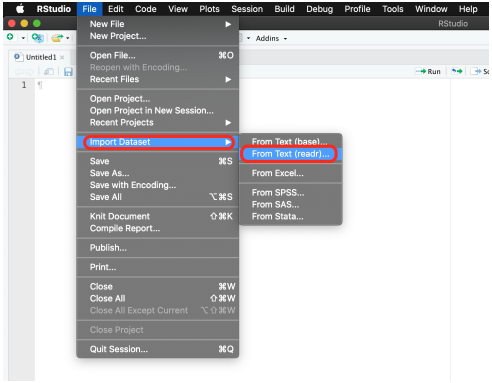
- tabulated text files (`*.csv`, `*.txt`, ...) → no pkg or `{readr}`
- MS Excel files (`*.xlsx`) → `{readxl}`
- binary R files (`*.rda`, `*.RData`, `*.rds`) → no pkg

...as well as many other formats (`*.feather`, `*.html`, `*.json`, `*.sav`, `*.zip`, ...)

Importing *.csv tabulated data with {readr}

TIP:

1. use the GUI



2. copy and paste the **R** code automatically generated into your script

NB:

- the same apply for many other file formats!
- if you don't see "From Text (readr)" then you must install {readr}

Practice

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

Getting started with **R**

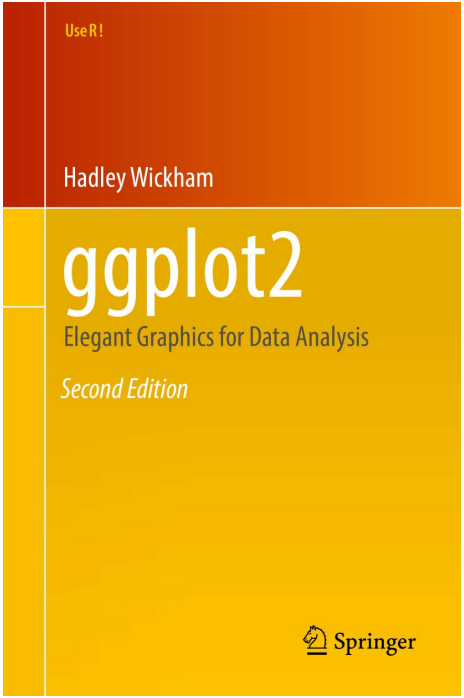
Plotting systems

There are many plotting systems in **R**, such as:

- `{graphics}` (build-in system): most efficient for small jobs, but difficult for complex tasks
- `{lattice}`: difficult, but efficient for complex tasks
- `{ggplot2}`: easy and efficient for most tasks (but a little verbose)

Practice: **R** Graph Gallery

Browse `https://www.r-graph-gallery.com` and try to reproduce one plot you like!


UseR!
Hadley Wickham
ggplot2
Elegant Graphics for Data Analysis
Second Edition
Springer

The grammar of graphics

All plots are composed of:

- Data that you want to visualise

Note: the word in **bold** font correspond to main functions of `{ggplot2}`

The grammar of graphics

All plots are composed of:

- Data that you want to visualise
- A set of **aes**thetic mappings describing which variables are mapped to which aesthetics

Note: the word in **bold** font correspond to main functions of `{ggplot2}`

The grammar of graphics

All plots are composed of:

- Data that you want to visualise
- A set of **aes**thetic mappings describing which variables are mapped to which aesthetics
- Layers made up of geometric elements (or **geom**s)

Note: the word in **bold** font correspond to main functions of `{ggplot2}`

The grammar of graphics

All plots are composed of:

- Data that you want to visualise
- A set of **aes**thetic mappings describing which variables are mapped to which aesthetics
- Layers made up of geometric elements (or **geom**s)
- **Scale**s describing how values in the data space are mapped to values in an aesthetic space

Note: the word in **bold** font correspond to main functions of `{ggplot2}`

The grammar of graphics

All plots are composed of:

- Data that you want to visualise
- A set of **aes**thetic mappings describing which variables are mapped to which aesthetics
- Layers made up of geometric elements (or **geom**s)
- **Scale**s describing how values in the data space are mapped to values in an aesthetic space
- A **coord**inate system describing how data coordinates are mapped to the plotted plane

Note: the word in **bold** font correspond to main functions of `{ggplot2}`

The grammar of graphics

All plots are composed of:

- Data that you want to visualise
- A set of **aes**thetic mappings describing which variables are mapped to which aesthetics
- Layers made up of geometric elements (or **geom**s)
- **Scale**s describing how values in the data space are mapped to values in an aesthetic space
- A **coord**inate system describing how data coordinates are mapped to the plotted plane
- A **facet**ing specification describing how data are dispatched into sub-plots

Note: the word in **bold** font correspond to main functions of `{ggplot2}`

The grammar of graphics

All plots are composed of:

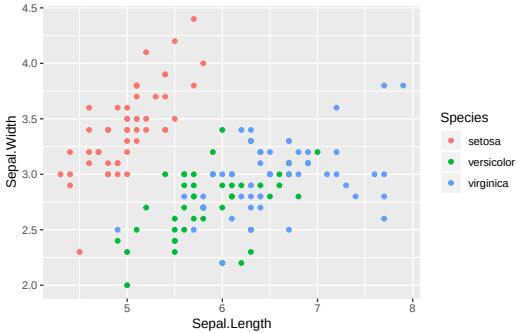
- Data that you want to visualise
- A set of **aes**thetic mappings describing which variables are mapped to which aesthetics
- Layers made up of geometric elements (or **geom**s)
- **Scale**s describing how values in the data space are mapped to values in an aesthetic space
- A **coord**inate system describing how data coordinates are mapped to the plotted plane
- A **facet**ing specification describing how data are dispatched into sub-plots
- A **theme** that controls the details of the display (font size, background colour. . .)

Note: the word in **bold** font correspond to main functions of `{ggplot2}`

Building a plot

We build a plot by adding components together:

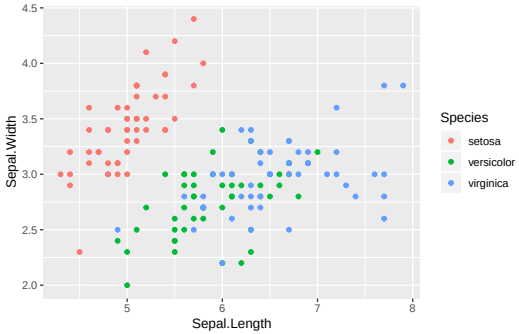
```
ggplot(iris) + ## the data
  aes(x = Sepal.Length, y = Sepal.Width, colour = Species) +  ## the aesthetic mappings
  geom_point()  ## a layer
```



Building a plot

We build a plot by adding components together:

```
ggplot(iris) + ## the data
  aes(x = Sepal.Length, y = Sepal.Width, colour = Species) +  ## the aesthetic mappings
  geom_point()  ## a layer
```



Note: here, we use the default settings for the scales, coordinate system, faceting specification, and the theme – so they don't need to be explicitly set

Building a plot: 3 main steps

Step 1: we specify the data (a tidy data frame) using `ggplot()`:

```
ggplot(iris) ## it calls the plotting device
```


Note: you can also write: `iris %>% ggplot()`

Building a plot: 3 main steps

Step 2: we specify the aesthetic mappings between the data and aesthetics using `aes()`:

```
aes(x = Sepal.Length, y = Sepal.Width, colour = Species) ## US spelling (here, color) also works!
## Aesthetic mapping:
## * `x`      -> `Sepal.Length`
## * `y`      -> `Sepal.Width`
## * `colour` -> `Species`
```

Note: sometimes more terms will be present to map other aesthetics to the data

Building a plot: 3 main steps

Step 3: we specify one or multiple layer(s) using a `geom_*()` function:

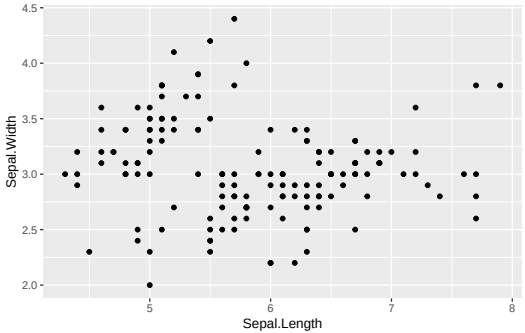
```
geom_point()
## geom_point: na.rm = FALSE
## stat_identity: na.rm = FALSE
## position_identity
```

Note: all `geom_*()` have a default statistic and position, which you can override

Layers: generalities

Changing the `geom_*()` call changes how the data are being represented!

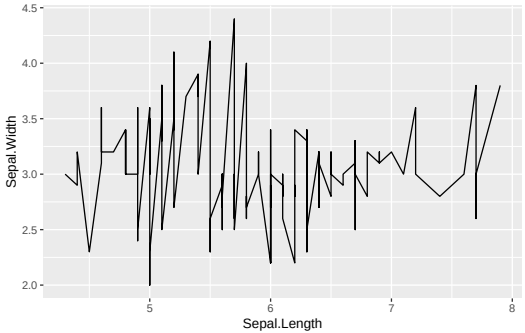
```
ggplot(iris) +
  aes(x = Sepal.Length, y = Sepal.Width) +
  geom_point()
```



Layers: generalities

Changing the `geom_*()` call changes how the data are being represented!

```
ggplot(iris) +
  aes(x = Sepal.Length, y = Sepal.Width) +
  geom_line()
```



Practice

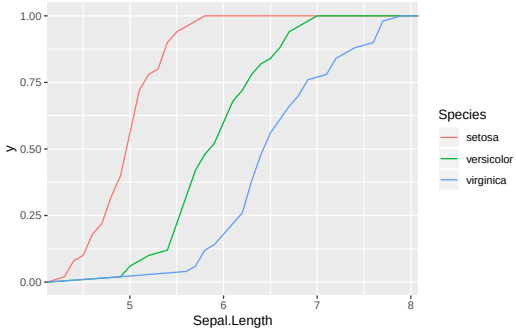
Plot the distribution of the sepal length of each species of iris using:

- `geom_boxplot()`
- `geom_violin()`
- `geom_jitter()`

Layers: statistics

Overridding the `geom_*()` default statistic is sometimes useful:

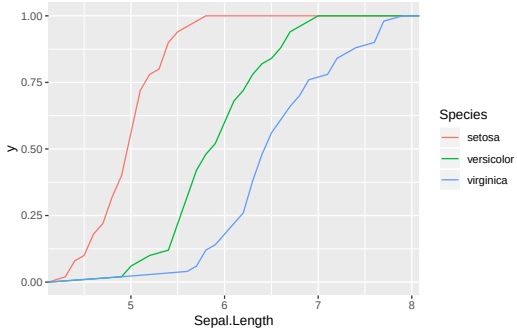
```
ggplot(iris) +
  aes(x = Sepal.Length, colour = Species) +
  geom_line(stat = "ecdf")
```



Layers: statistics

Overridding the `geom_*()` default statistic is sometimes useful:

```
ggplot(iris) +
  aes(x = Sepal.Length, colour = Species) +
  geom_line(stat = "ecdf")
```

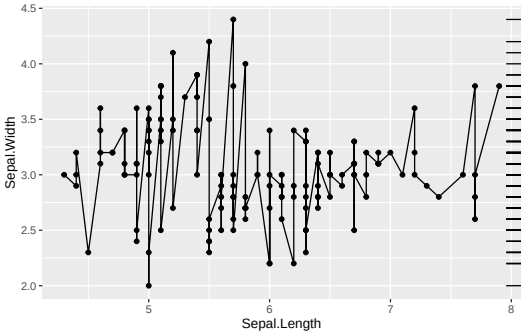


Note: plotting the Empirical Cumulative Distribution Function allows for deriving information from the distribution of the data (more on that later!)

Multiple layers

Several `geom_*()` call can be used one after another:

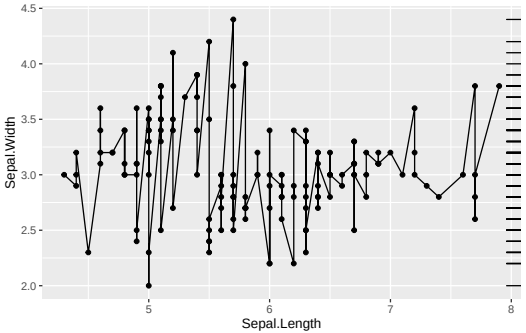
```
ggplot(iris) +
  aes(x = Sepal.Length, y = Sepal.Width) +
  geom_point() +
  geom_line() +
  geom_rug(sides = "r") ## r for right!
```



Multiple layers

Several `geom_*()` call can be used one after another:

```
ggplot(iris) +
  aes(x = Sepal.Length, y = Sepal.Width) +
  geom_point() +
  geom_line() +
  geom_rug(sides = "r") ## r for right!
```



Note: the order can matter since the layers are drawn from top to bottom

Aesthetics attributes: generalities

The `geom_*()` functions rely on aesthetics attributes.

Some aesthetics are compulsory, others are optional.

Examples of aesthetics:

- position: `x`, `y`, `xmin`, `xmax`, `ymin`, `ymax`...
- colours: `colour` and `fill`
- transparency: `alpha`
- sizes: `size` and `width`
- shape: `shape` and `linetype`

Aesthetics attributes: generalities

The `geom_*()` functions rely on aesthetics attributes.

Some aesthetics are compulsory, others are optional.

Examples of aesthetics:

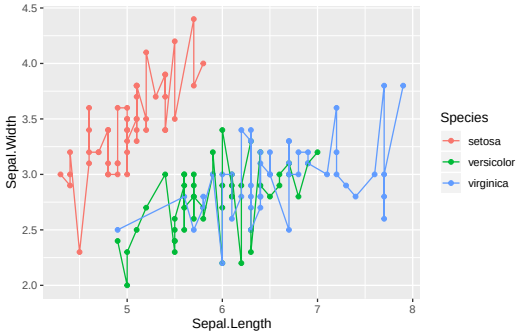
- position: `x`, `y`, `xmin`, `xmax`, `ymin`, `ymax`...
- colours: `colour` and `fill`
- transparency: `alpha`
- sizes: `size` and `width`
- shape: `shape` and `linetype`

Tip: check the help of the geoms you want to use to know which aesthetics can be used!

Aesthetics: where to provide the aesthetic mappings?

If the `aes()` call is outside a function, it applies to all geoms:

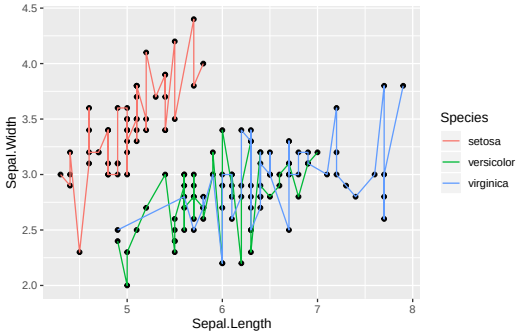
```
ggplot(iris) +
  aes(x = Sepal.Length, y = Sepal.Width, colour = Species) +
  geom_point() +
  geom_line()
```



Aesthetics: where to provide the aesthetic mappings?

You can split the definition of global and local aesthetic mappings:

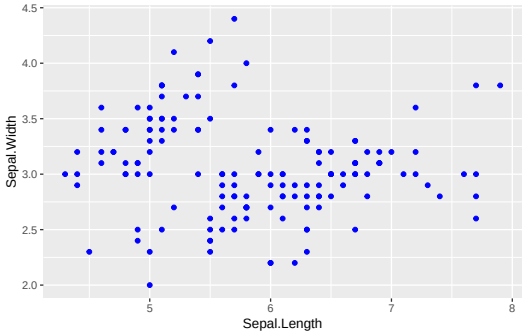
```
ggplot(iris) +
  aes(x = Sepal.Length, y = Sepal.Width) +
  geom_point() +
  geom_line(aes(colour = Species))
```



Aesthetics: where to provide the aesthetic mappings?

Aesthetics not mapped to the data should not be part of any `aes()` call!

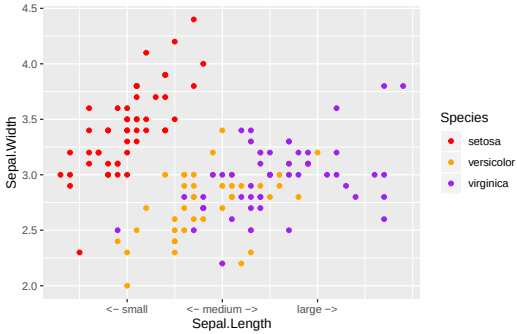
```
ggplot(iris) +
  aes(x = Sepal.Length, y = Sepal.Width) +
  geom_point(colour = "blue")
```



Scales

You can use `scale_*()` to change properties of each aesthetic attribute; e.g.:

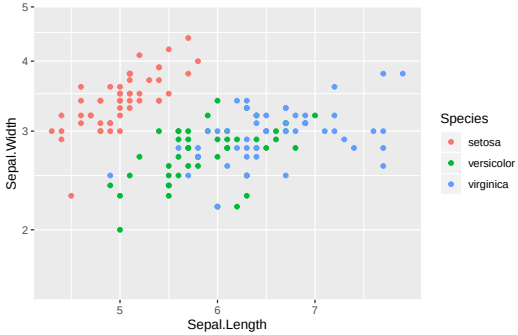
```
ggplot(iris) +
  aes(x = Sepal.Length, y = Sepal.Width, colour = Species) +
  geom_point() +
  scale_x_continuous(breaks = c(5, 6, 7), labels = c("<- small", "<- medium ->", "large ->")) +
  scale_colour_manual(values = c("red", "orange", "purple"))
```



Coordinates

You can use `coord_*()` to modify how graphical elements are projected on a plan; e.g.:

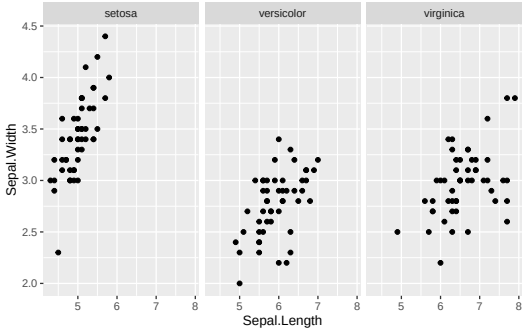
```
ggplot(iris) +
  aes(x = Sepal.Length, y = Sepal.Width, colour = Species) +
  geom_point() +
  coord_trans(y = "log10", limy = c(1.5, 5))
```



Facets

You can use `facet_*()` to shows different subset of the data separately; e.g.:

```
ggplot(iris) +
  aes(x = Sepal.Length, y = Sepal.Width) +
  geom_point() +
  facet_wrap(~ Species)
```



Themes: generalities

Themes control the display of all elements of the plot that do not depend on your data:

- titles
- labels
- fonts
- background
- gridlines
- legends
- ...

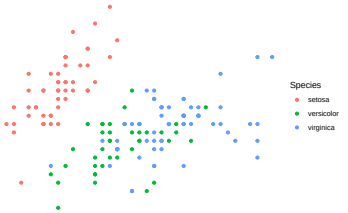
Note: see `?theme()` for an exhaustive list of all that can be changed

Themes: pre-defined themes

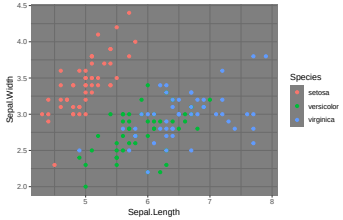
There are many predefined themes in {ggplot2} and companion packages (e.g. {ggthemes}):

```
ggplot(iris) +
  aes(x = Sepal.Length, y = Sepal.Width, colour = Species) +
  geom_point() -> myplot
```

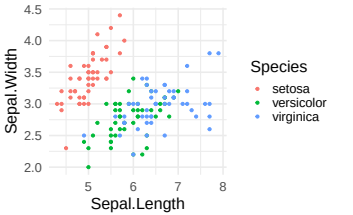
```
myplot +
  theme_void()
```




```
myplot +
  theme_dark()
```




```
myplot +
  theme_minimal(base_size = 20)
```

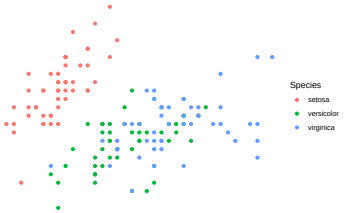


Themes: pre-defined themes

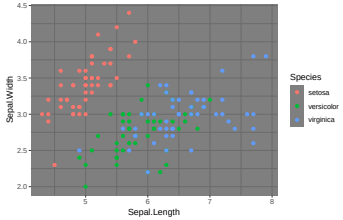
There are many predefined themes in {ggplot2} and companion packages (e.g. {ggthemes}):

```
ggplot(iris) +
  aes(x = Sepal.Length, y = Sepal.Width, colour = Species) +
  geom_point() -> myplot
```

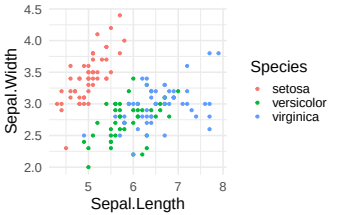
```
myplot +
  theme_void()
```




```
myplot +
  theme_dark()
```




```
myplot +
  theme_minimal(base_size = 20)
```

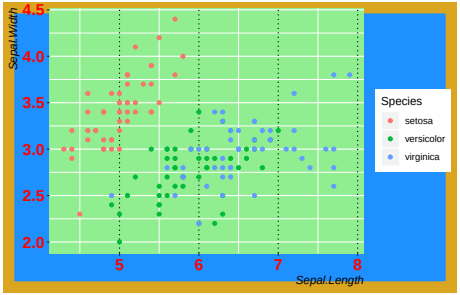


Tip: `base_size` controls the base font size for any theme

Themes: modifying themes

To modify the theme yourself, use `theme()` in combination with `element_*()`:

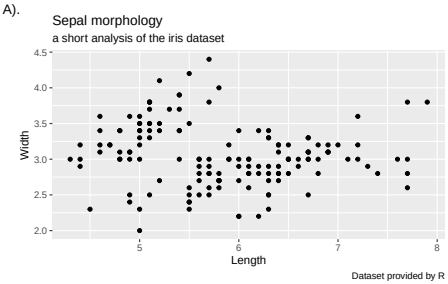
```
ggplot(iris) +
  aes(x = Sepal.Length, y = Sepal.Width, colour = Species) +
  geom_point() +
  theme(axis.text = element_text(size = 15, face = "bold", colour = "red"),
        axis.title = element_text(face = "italic", hjust = 1),
        panel.grid.major.x = element_line(linetype = "dotted", colour = "black"),
        panel.background = element_rect(fill = "lightgreen"),
        plot.background = element_rect(fill = "dodgerblue", colour = "goldenrod", size = 10))
```



Adding labels

You can set all the labels easily using `labs()`:

```
ggplot(iris) +
  aes(x = Sepal.Length, y = Sepal.Width) +
  geom_point() +
  labs(x = "Length", y = "Width",
       title = "Sepal morphology", subtitle = "a short analysis of the iris dataset",
       caption = "Dataset provided by R",
       tag = "A).")
```



Saving plots

You can export your plots using `ggsave()`:

```
ggplot(iris) +
  aes(x = Sepal.Length, y = Sepal.Width, colour = Species) +
  geom_point()

ggsave("my_first_plot.pdf", width = 15, height = 10, units = "cm")
```

Note: by default, the plot is saved in your project directory

Getting started with **R**

Usual programming commands exist in **R**

```
for (i in 1:4) {
  if (i == 2) {
    print(x = "found 2!")
  } else {
    print(x = "found something other than 2")
    }
}
## [1] "found something other than 2"
## [1] "found 2!"
## [1] "found something other than 2"
## [1] "found something other than 2"
```

Usual programming commands exist in **R**

```
for (i in 1:4) {
  if (i == 2) {
    print(x = "found 2!")
  } else {
    print(x = "found something other than 2")
    }
}
## [1] "found something other than 2"
## [1] "found 2!"
## [1] "found something other than 2"
## [1] "found something other than 2"
```

You can check the help files for such functions as follow:

```
?"for"
```

Note: the quotes are necessary because the parser considers that such words are special

You can create your own **R** functions!

```r
OddRatio <- function(proba.x, proba.y) {
  odd.x <- proba.x/(1 - proba.x)
  odd.y <- proba.y/(1 - proba.y)
  odd.x/odd.y
}
```

You can create your own **R** functions!

```
OddRatio <- function(proba.x, proba.y) {
  odd.x <- proba.x/(1 - proba.x)
  odd.y <- proba.y/(1 - proba.y)
  odd.x/odd.y
}
```

Then you can use them as any other functions:

```
OddRatio(0.1, 0.01)
## [1] 11
```

Getting started with **R**

What should you do?

- update your **R** packages frequently → at least once a month

```
update.packages(ask = FALSE)  ## or use RStudio menus
```

NB:
you can check what is being changed in `NEWS` files

What should you do?

- update your **R** packages frequently → at least once a month

```
update.packages(ask = FALSE)  ## or use RStudio menus
```

- update **R** from time to time → at least once a year in spring
  do it manually, or try {InstallR} on Windows, or {UpdateR} on macOS

NB:
you can check what is being changed in NEWS files

What should you do?

- update your **R** packages frequently → at least once a month

```
update.packages(ask = FALSE)  ## or use RStudio menus
```

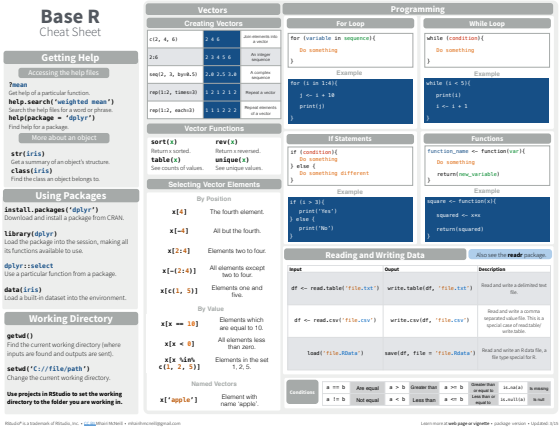
- update **R** from time to time → at least once a year in spring
  do it manually, or try {InstallR} on Windows, or {UpdateR} on macOS
- update RStudio from time to time → at least after each update of **R**

NB:
you can check what is being changed in NEWS files

Getting started with **R**

Useful resources

RStudio cheatsheets; e.g.:



NB: there are many cheatsheets covering many aspects of **R** and several packages!

Useful resources

Official documentation:

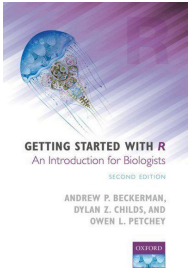
- the help files: every single (exported) function has a help file associated with it!
- official manuals: `https://cran.r-project.org/manuals.html`

Useful resources

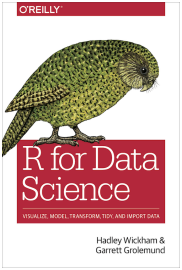
Official documentation:

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

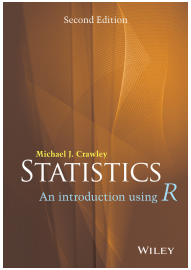
Best books for you (roughly sorted by amount of conceptual content):



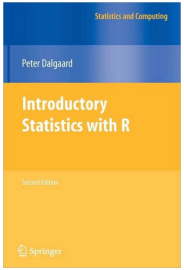
$\sim$ 30 €



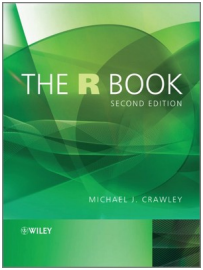
$\sim$ 60 €



$\sim$ 35 €



$\sim$ 40 €



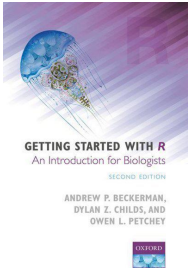
$\sim$ 60 €

Useful resources

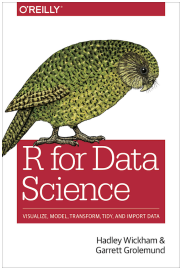
Official documentation:

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

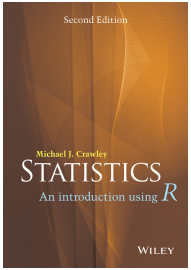
Best books for you (roughly sorted by amount of conceptual content):



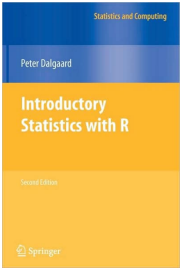
~ 30 €



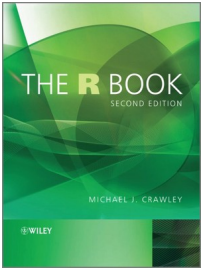
~ 60 €



~ 35 €



~ 40 €



~ 60 €

Journal of Statistical Software, **R** Journal, Journal of Open Source Software ...

Become part of the **R** community!

- face-to-face interactions:
  - Meetup Berlin R Users Group
  - Meetup R-Ladies Berlin
  - SATRDAYS (`https://satrdays.org/events`)
  - European R Users Meeting (`https://erum.io`)
  - `rstudio::conf` (`https://www.rstudio.com/conference`)
  - useR! (`@useR2020stl`)

Become part of the **R** community!

- face-to-face interactions:
  - Meetup Berlin R Users Group
  - Meetup R-Ladies Berlin
  - SATRDAYS (`https://satrdays.org/events`)
  - European R Users Meeting (`https://erum.io`)
  - `rstudio::conf` (`https://www.rstudio.com/conference`)
  - useR! (`@useR2020stl`)
- websites:
  - `https://stackoverflow.com/questions/tagged/r`
  - `https://community.rstudio.com`
  - `https://www.r-bloggers.com`

Become part of the **R** community!

- face-to-face interactions:
  - Meetup Berlin R Users Group
  - Meetup R-Ladies Berlin
  - SATRDAYS (`https://satrdays.org/events`)
  - European R Users Meeting (`https://erum.io`)
  - `rstudio::conf` (`https://www.rstudio.com/conference`)
  - useR! (`@useR2020stl`)
- websites:
  - `https://stackoverflow.com/questions/tagged/r`
  - `https://community.rstudio.com`
  - `https://www.r-bloggers.com`
- mailing lists:
  - `https://www.r-project.org/mail.html`

Become part of the **R** community!

- face-to-face interactions:
  - Meetup Berlin R Users Group
  - Meetup R-Ladies Berlin
  - SATRDAYS (`https://satrdays.org/events`)
  - European R Users Meeting (`https://erum.io`)
  - `rstudio::conf` (`https://www.rstudio.com/conference`)
  - useR! (`@useR2020stl`)
- websites:
  - `https://stackoverflow.com/questions/tagged/r`
  - `https://community.rstudio.com`
  - `https://www.r-bloggers.com`
- mailing lists:
  - `https://www.r-project.org/mail.html`
- twitter:
  - #rstats
  - #tidyverse
  - @hadleywickham (>92K followers)
  - @dataandme (>40K)
  - @drob (>39K)
  - @rdpeng (>38K)
  - @JennyBryan (>27K)
  - @statgarrett (>17K)
  - @RStudio (>84K)
  - @Rbloggers (>74K)
  - @RLangTip (>64K)
  - @rstudiotips (>53K)
  - @R_Programming (>31K)
  - @WeAreRLadies & @RLadiesGlobal (2× >13K)
  - @rstatstweet & @rstats4ds (bot like)
  - @rdataberlin (new!)

The best person who can teach you **R** may be yourself

Once you know some basics, you should be able to start learning on your own

by simply performing experiments in your **R** console!