

Getting started with R

Alexandre Courtiol @rdataberlin

Leibniz Institute of Zoo and Wildlife Research

February 2023



**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)
- 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)

What is R good for?

R is good for:

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

What is R good for?

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

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

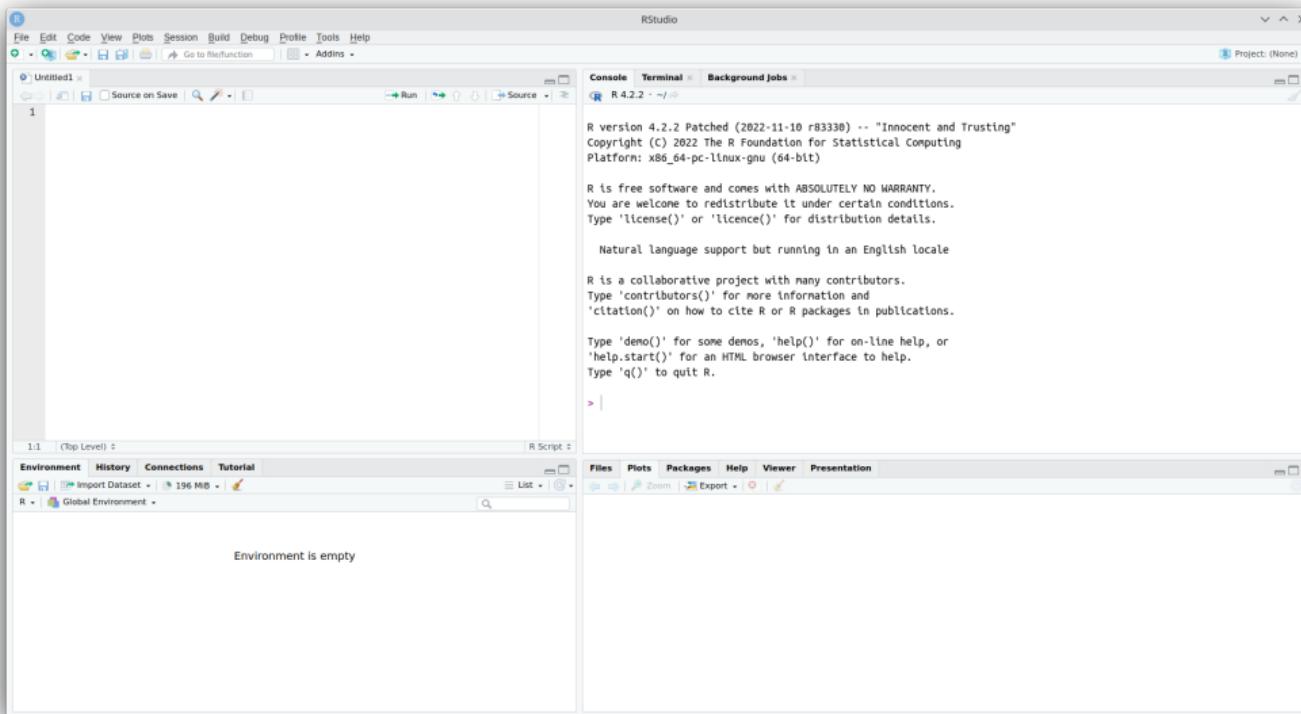
Installation steps

- ① install R: <https://cran.r-project.org/>
- ② install RStudio: <https://posit.co/download/rstudio-desktop/>
- ③ open RStudio

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

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

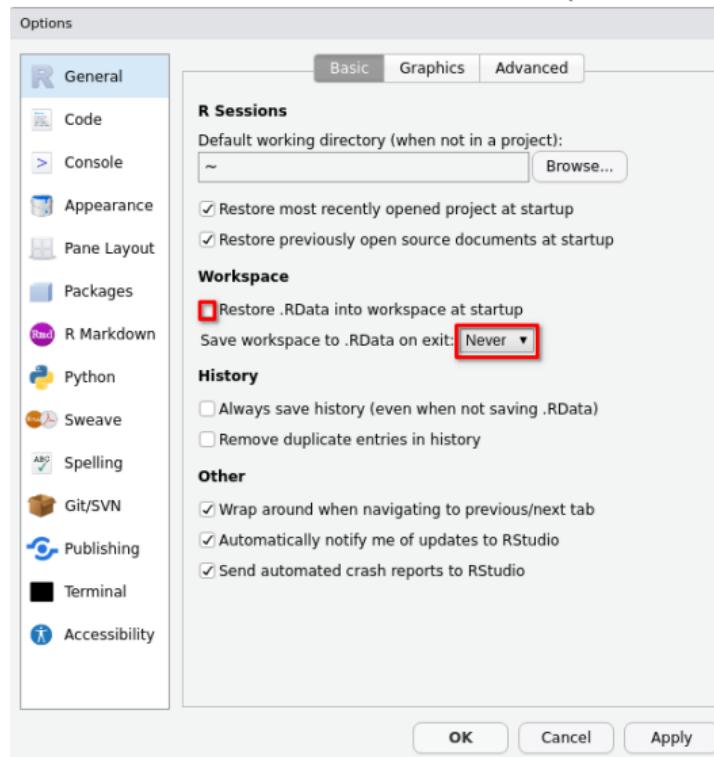
- ① create a new project
- ② create a new **R** Script file
- ③ save the created **R** Script file into the project folder directory
- ④ have a look at the RStudio cheatsheet

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

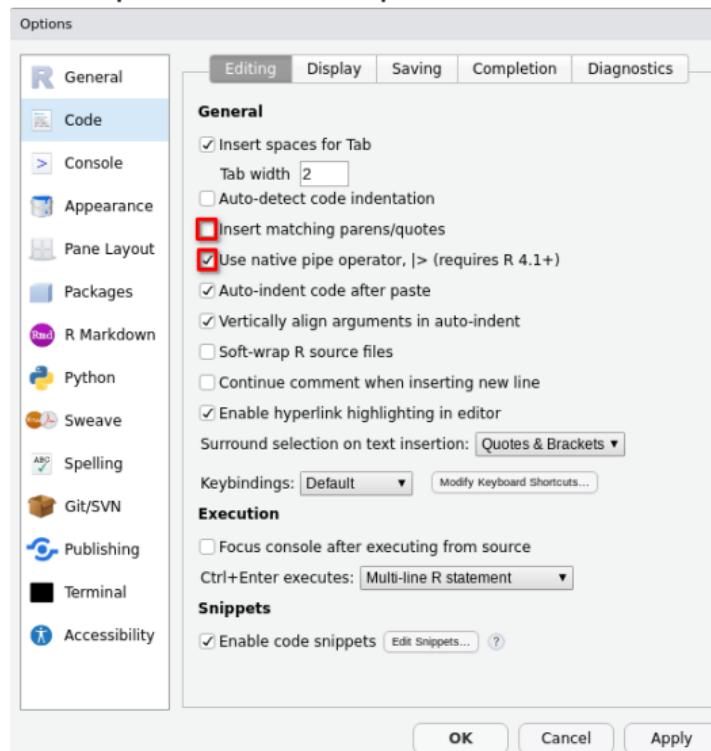
My tips about the global options

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



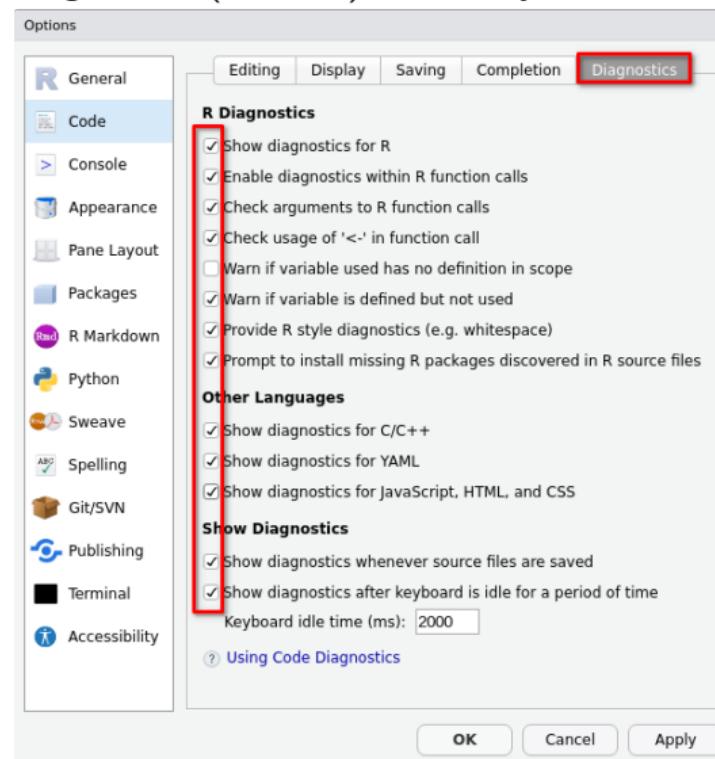
My tips about the global options

2. I don't like parentheses and quotes auto-completion & I like the native pipe



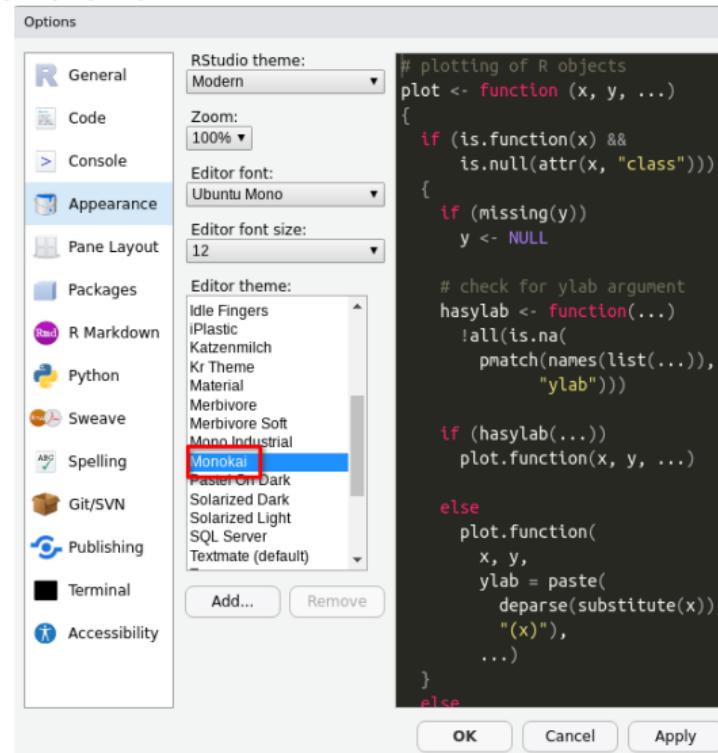
My tips about the global options

3. I activate all the code diagnostics (but one) offered by RStudio!



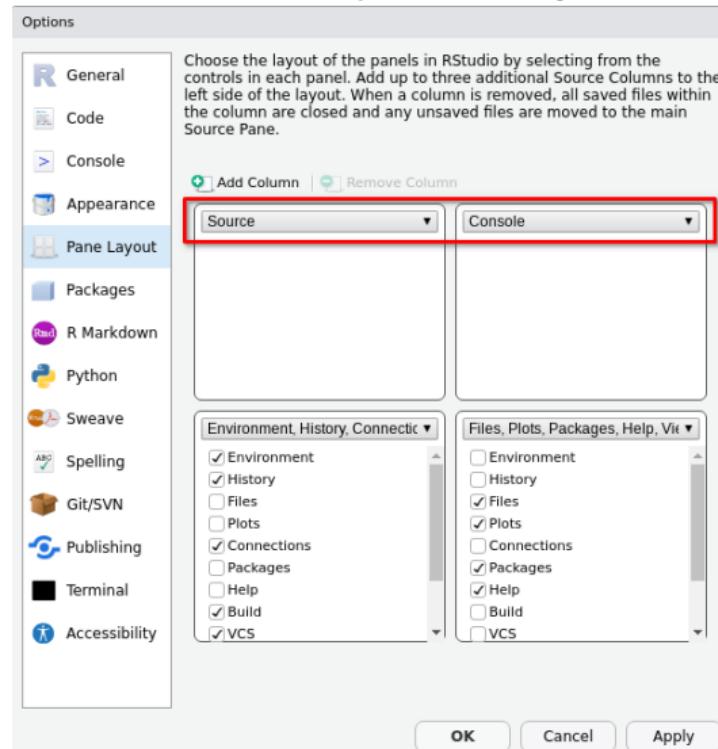
My tips about the global options

4. I sometimes use a black theme



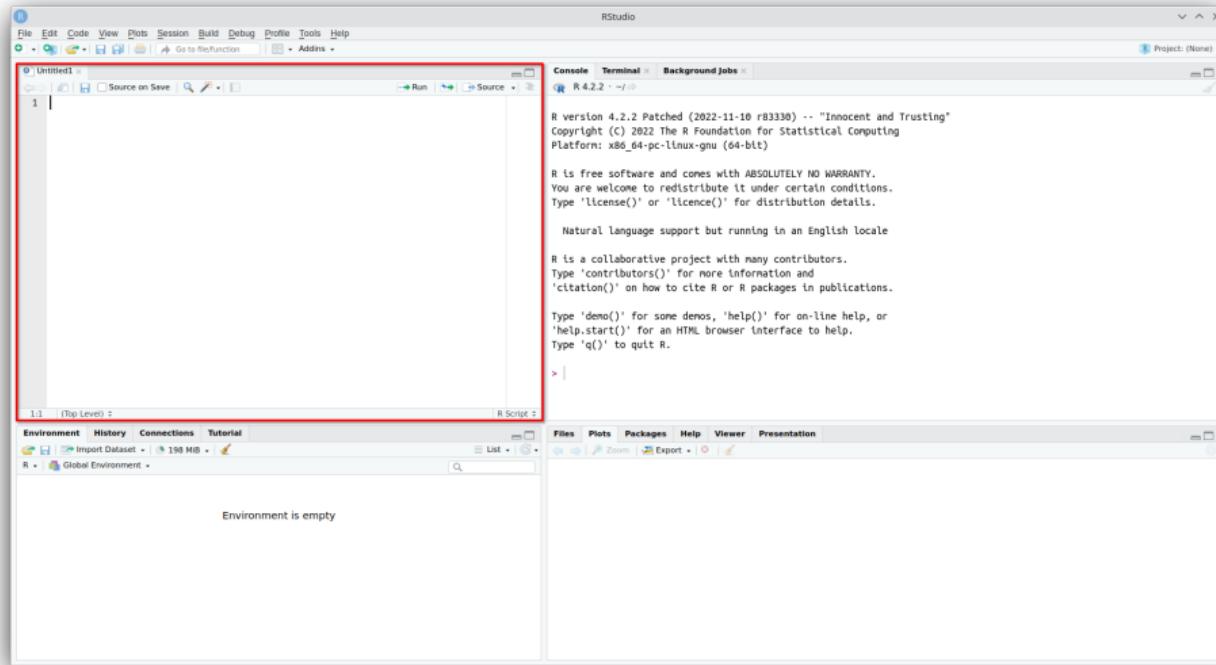
My tips about the global options

5. I usually put the Source and the Console panes side-by-side



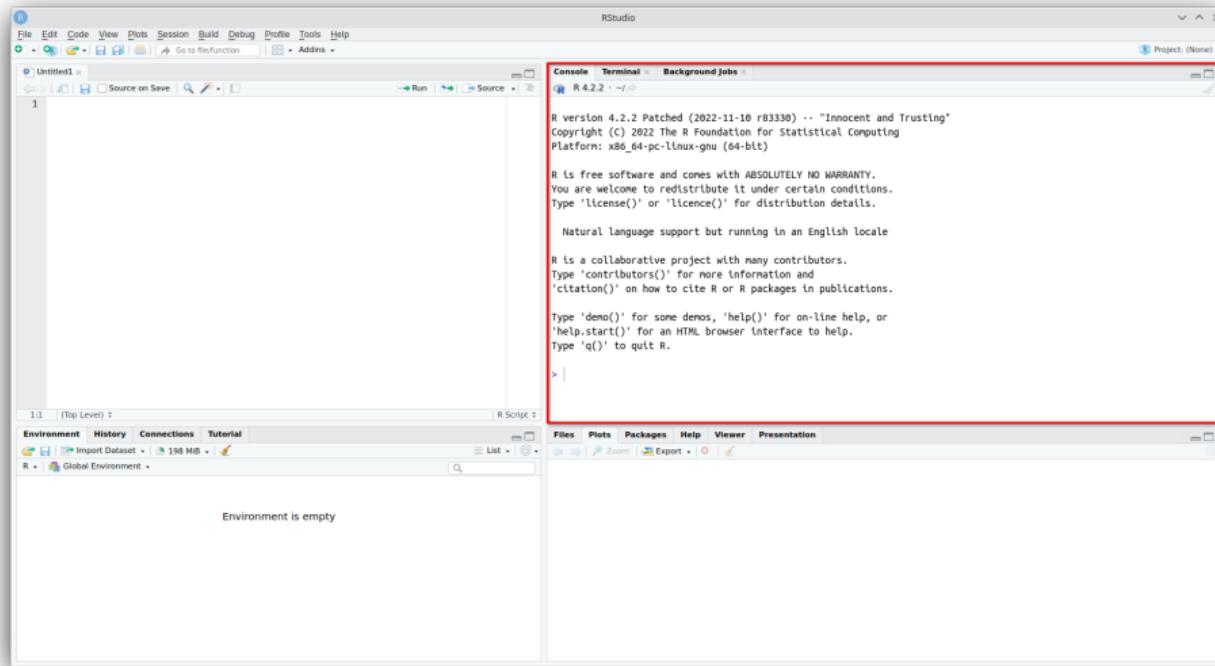
The Source pane

This is where you type the code you want to keep



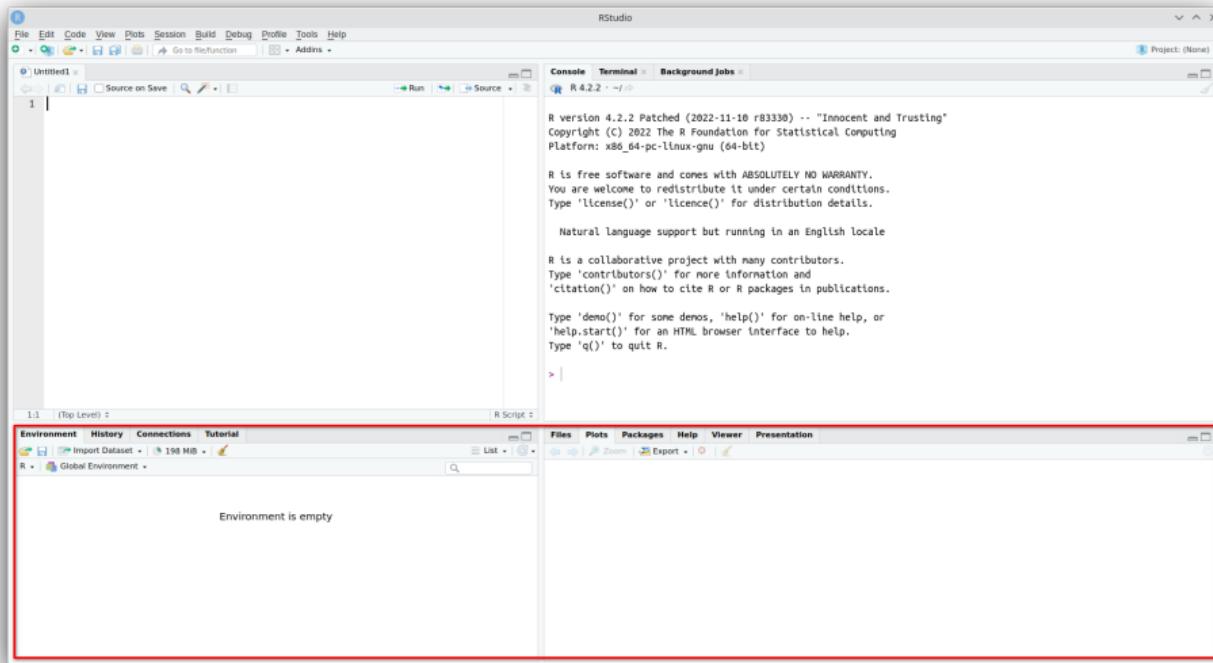
The Console pane

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 comment thoroughly your **R** script using `#` signs
- ④ re-run frequently your entire script to make sure that it works
(after restarting the session: Session/Rerun R)

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

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 $\sqrt{\frac{2^{3+1}}{4} - 20}$

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
```

- \rightarrow works too (if you switch the left hand side and the right hand 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 switch the left hand side and the right hand 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 switch the left hand side and the right hand side)
- “`_`” and “`.`” are OK but avoid spaces & other weird characters in names
- names are CaSe sensITIVE

Functions

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

##
## To cite R in publications use:
##
##   R Core Team (2022). 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 = {2022},
##     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

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

##
## To cite R in publications use:
##
##   R Core Team (2022). 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 = {2022},
##     url = {https://www.R-project.org/},
##   }
##
## We have invested a lot of time and effort in creating R, please cite it when using it for
## data analysis. See also 'citation("pkgname")' for citing R packages.
```

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

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

Functions

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

##
## To cite R in publications use:
##
##   R Core Team (2022). 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 = {2022},
##     url = {https://www.R-project.org/},
##   }
##
## We have invested a lot of time and effort in creating R, please cite it when using it for
## data analysis. See also 'citation("pkgname")' for citing R packages.
```

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

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

NB: it is 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 equivalent:

```
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
```

```
c(1, 5, 3, 4) |> mean() ## see ?pipeOp  
## [1] 3.25
```

```
vector.of.numbers |> mean()  
## [1] 3.25
```

```
vector.of.numbers |> mean(x = _)  
## [1] 3.25
```

Practice

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

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

Finding functions

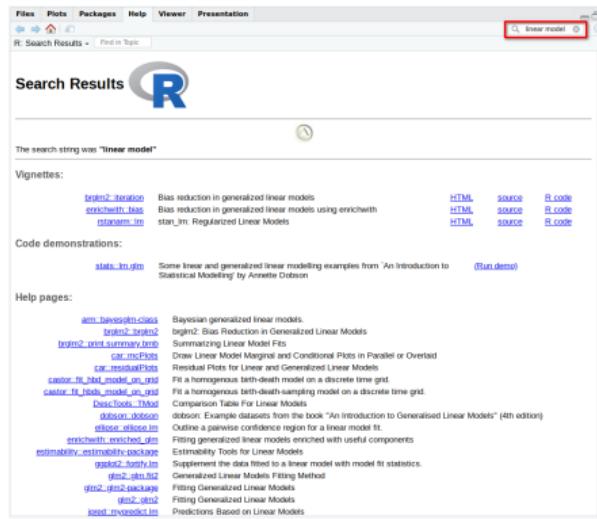
To find the name of the function you are looking for, you may try:

```
?"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



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

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

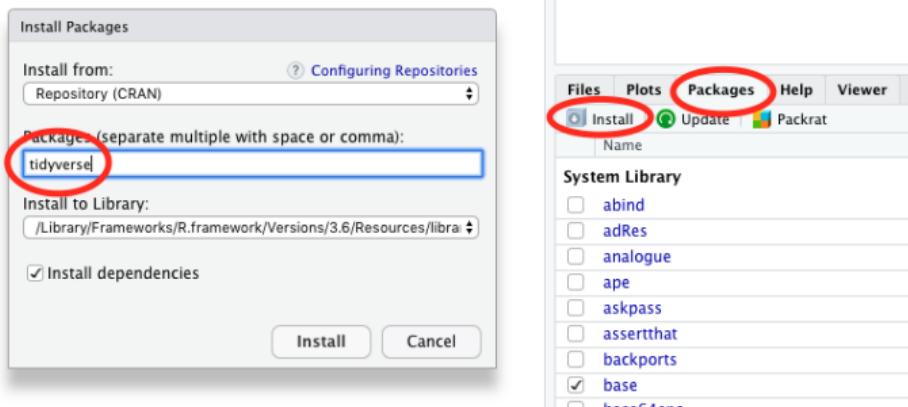
Tip: in order to be able to install packages that require compilation (and thus have access to more or newer versions of packages), you need to install:

- 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

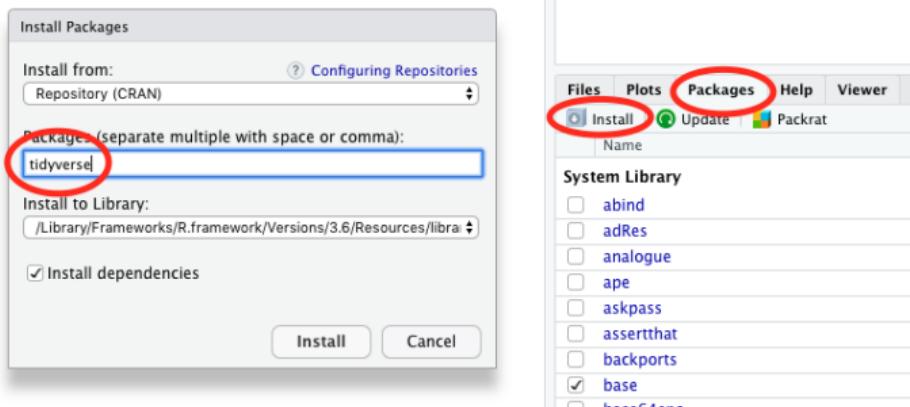
You can use RStudio:



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 1.3.2 -
## v ggplot2 3.4.0     v purrrr   1.0.1
## v tibble  3.1.8     v dplyr    1.1.0
## v tidyrr   1.3.0     v stringr   1.5.0
## v readr    2.1.3     v forcats  1.0.0
## - 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, you need to load the libraries each time you open a new **R** session!

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

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 usually corresponds to a vector (a series 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 `?pillar_options`)
- 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 `?pillar_options`)
- lazier & safer (see `?`tbl_df-class``)
- more flexible (work well with groupings)

Tip: functions working with data frames usually work fine with tibbles, but in case of problems just try converting your tibble into an original data frame using `as.data.frame()`

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        3.0        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        3.0        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 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 mean 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 per species

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

## compute the mean and SD per species
results <- summarise(iris, meanSL = mean(Sepal.Length.mm),
                      sdSL    = sd(Sepal.Length.mm),
                      .by = Species) # declare that you will perform operations within species

## 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 with pipes (|> or %>%)

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

```
iris |>
  mutate(Sepal.Length.mm = Sepal.Length * 10) |>
  summarise(meanSL = mean(Sepal.Length.mm),
            sdSL   = sd(Sepal.Length.mm),
            .by = Species) -> results
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 with pipes (|> or %>%)

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

```
iris |>
  mutate(Sepal.Length.mm = Sepal.Length * 10) |>
  summarise(meanSL = mean(Sepal.Length.mm),
            sdSL   = sd(Sepal.Length.mm),
            .by = Species) -> results
results
##      Species  meanSL    sdSL
## 1    setosa  50.06 3.524897
## 2 versicolor  59.36 5.161711
## 3 virginica  65.88 6.358796
```

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

The {tidyverse}

The tidyverse (<https://www.tidyverse.org>) is an ecosystem of R packages.

There are 8 core 'tidy' packages:



The {tidyverse}

The tidyverse (<https://www.tidyverse.org>) is an ecosystem of R packages.

There are 8 core 'tidy' 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
- open-source (collaborative) development supervised by Posit (formerly known as RStudio)

Manipulating data frames with {dplyr}

You can do a lot with 4 functions:

Manipulating data frames with {dplyr}

You can do a lot with 4 functions:

- `select()` to keep or discard columns
- `filter()` to keep or discard rows
- `mutate()` to create or modify columns
- `summarise()` to compute summary statistics

Manipulating data frames with {dplyr}

You can do a lot with 4 functions:

- `select()` to keep or discard columns
- `filter()` to keep or discard rows
- `mutate()` to create or modify columns
- `summarise()` to compute summary statistics

Notes:

- this is much more than that in {dplyr}
- you can use the argument `.by` to define groups of rows when using `filter()`, `mutate()` or `summarise()`

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 per group using filter(..., .by = xxx):

```
iris_tbl |> filter(Sepal.Length == max(Sepal.Length), .by = Species)
## # 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 per group using filter(..., .by = xxx):

```
iris_tbl |> filter(Sepal.Length == max(Sepal.Length), .by = Species)
## # 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: using .by is safer than group_by() because the grouping is not persistent

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      3.0       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        3.0       1.4       0.2  setosa        49
## 3         4.7        3.2       1.3       0.2  setosa        47
## # ... with 147 more rows
```

Create a column based on values per group using `mutate(..., .by = xxx)`:

```
iris_tbl |> mutate(Sepal.Length.max = max(Sepal.Length), .by = Species)
## # 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        3.0       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        3.0       1.4       0.2  setosa        49
## 3         4.7        3.2       1.3       0.2  setosa        47
## # ... with 147 more rows
```

Create a column based on values per group using `mutate(..., .by = xxx)`:

```
iris_tbl |> mutate(Sepal.Length.max = max(Sepal.Length), .by = Species)
## # 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        3.0       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 per group using summarise(..., .by = xxx):

```
iris_tbl |> summarise(Sepal.Length.max = max(Sepal.Length), N = n(), .by = Species)
## # 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 per group using `summarise(..., .by = xxx)`:

```
iris_tbl |> summarise(Sepal.Length.max = max(Sepal.Length), N = n(), .by = Species)
## # 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
```

Tip: `summarise()` can only return one row per group; `reframe()` is more flexible

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 between group setosa and group versicolor"
## $ 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 between group setosa and group versicolor"
## $ 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 between group setosa and group versicolor"
## $ 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 at once

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

Importing data in R

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

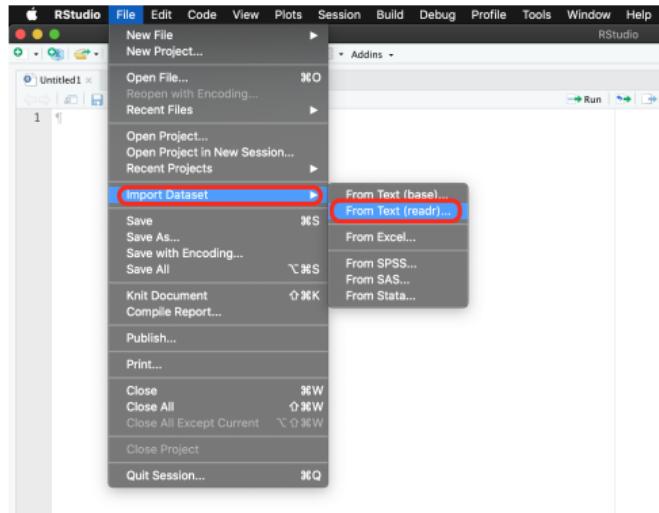
- tabulated text files (*.csv, *.txt, ...) → no pkg or {readr}
- MS Excel files (*.xlsx) → {readxl}
- binary R files (*.rda, *.RData, *.rds) → no pkg

... 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 data frame using your favourite spreadsheet software
and import the data in R!

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

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://r-graph-gallery.com> and try to reproduce one plot you like!

 UseR!

Hadley Wickham

 ggplot2

Elegant Graphics for Data Analysis

Second Edition

You can make nice plots with {ggplot2} and companion packages

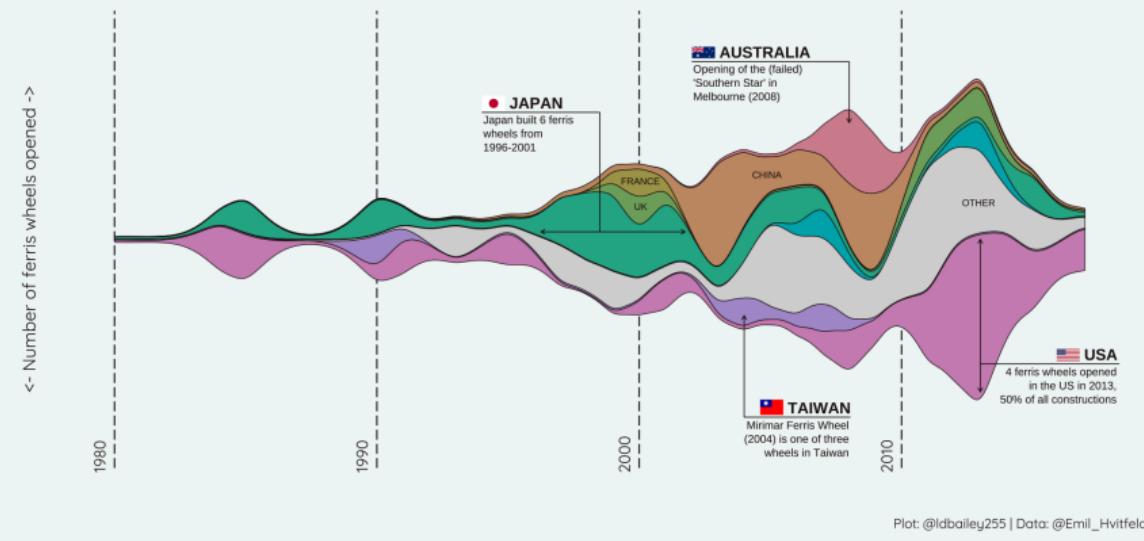
@ldbailey255

2013 WAS PEAK FERRIS WHEEL

8 ferris wheels were opened in 2013 across 5 countries.

The US opened 16 ferris wheels between 1980 and 2017.

Japan has opened the second most (11), but only 1 in the last decade.

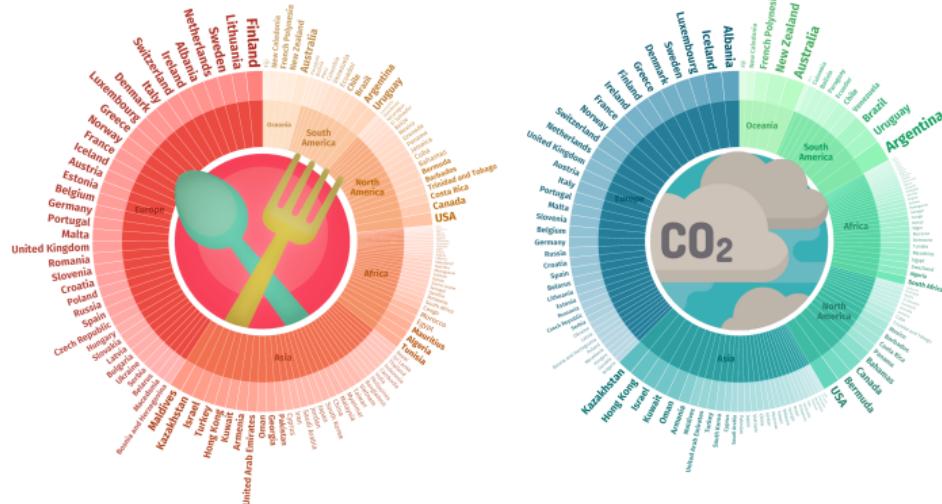


You can make nice plots with {ggplot2} and companion packages

@CedScherer / @CedScherer@vis.social

Food Carbon Footprint Index 2018

Global comparison of different diets in terms of **Average Consumption** (kg/person/year) of both animal and non-animal products as well as **Carbon Emissions** (kg CO₂/person/year) per continent and country. Font size and color intensity indicate each country's estimate with **countries printed in bold** belonging to the upper 50% of consumers and CO₂ emitters, respectively.



Visualization by Cédric Scherer • Data by Food and Agriculture Organization of the United Nations (FAO) via nuz • Icons by FreePik

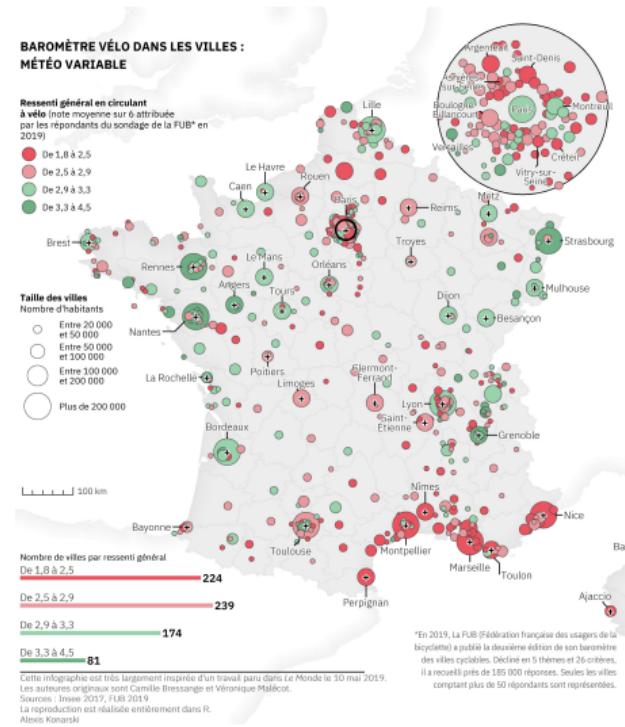
You can make nice plots with {ggplot2} and companion packages

@geokaramanis



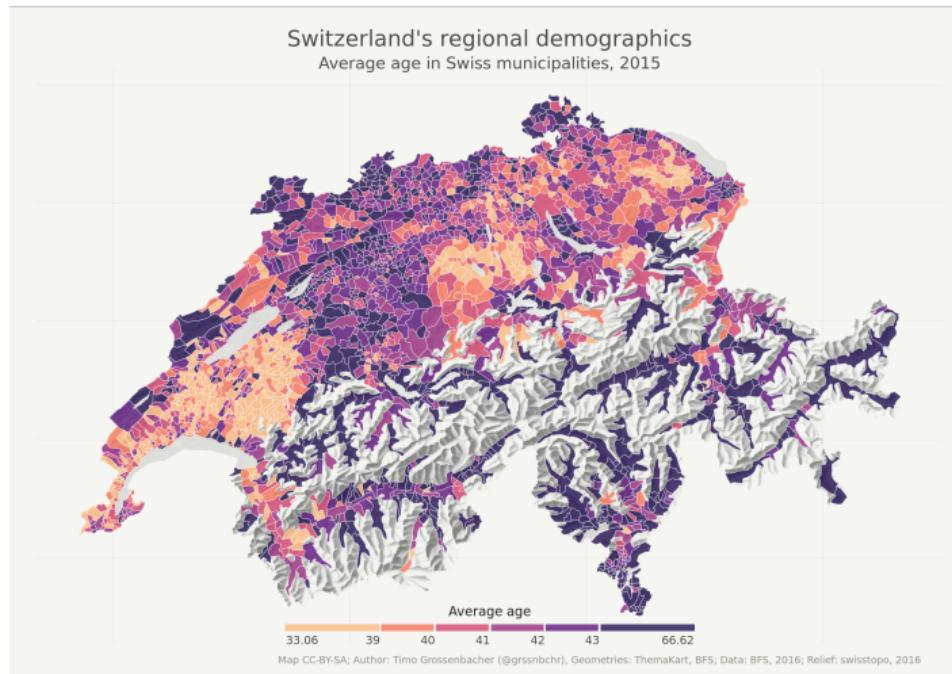
You can make nice plots with {ggplot2} and companion packages

@_Alexis_69_



You can make nice plots with {ggplot2} and companion packages

@grssnbchr



The grammar of graphics

All plots are composed of:

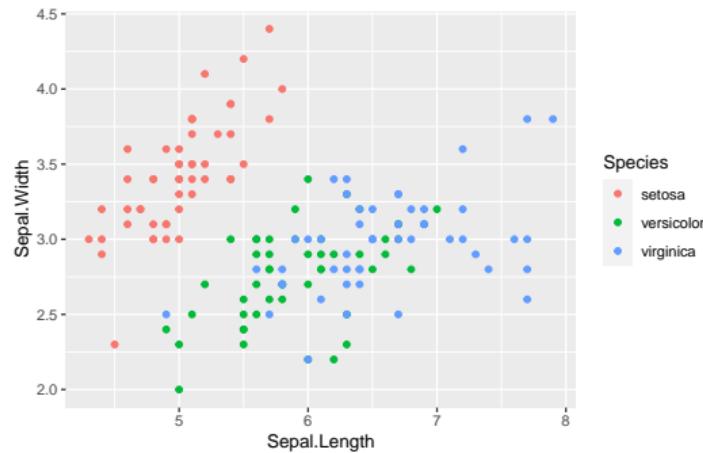
- Data that you want to visualise
- A set of **aesthetic** mappings describing which variables are mapped to which visual properties
- Layers made up of **geometric** elements
- **Scales** describing how values in the data space are mapped to values in an aesthetic space
- A **coordinate** system describing how data coordinates are mapped to the plotted plane
- A **faceting** 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:

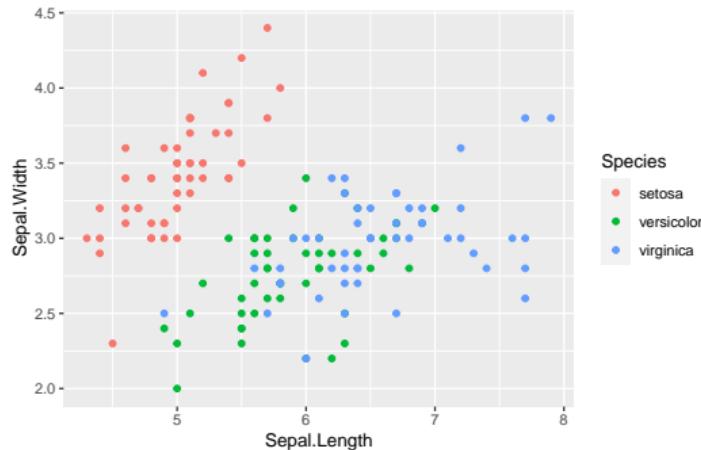
```
ggplot(iris) + ## the data  
  aes(x = Sepal.Length, y = Sepal.Width, colour = Species) +  ## the aesthetic mappings  
  geom_point()  ## a layer
```



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
```

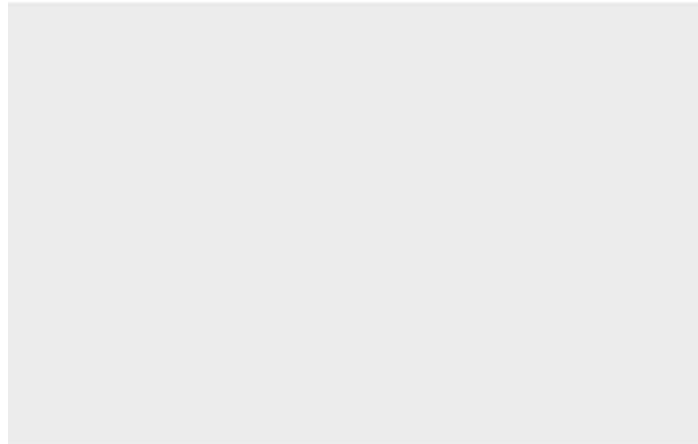


Note: here, we use the default settings for the scales, coordinate system, faceting specification, and the theme – they don't need to be explicitly set

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:

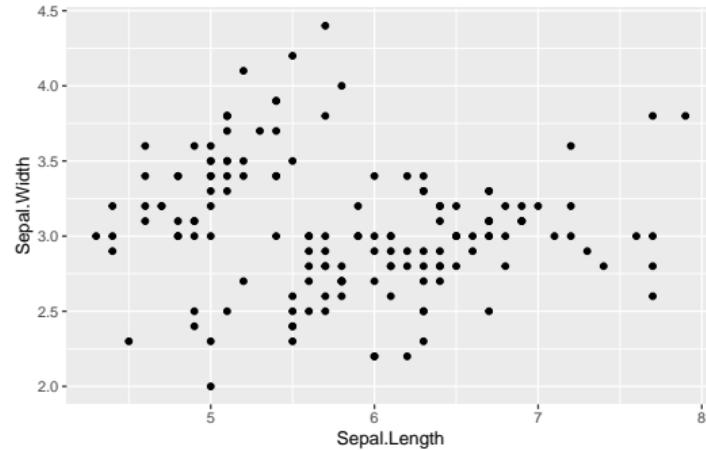
```
geom_point()  
## geom_point: na.rm = FALSE  
## stat_identity: na.rm = FALSE  
## position_identity
```

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!

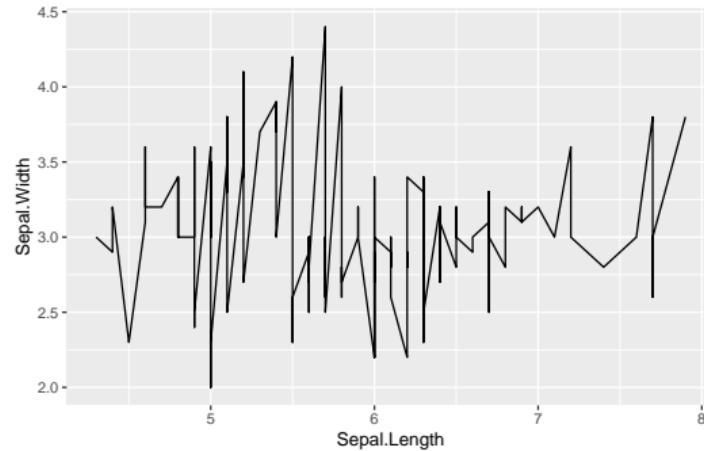
```
ggplot(iris) +  
  aes(x = Sepal.Length, y = Sepal.Width) +  
  geom_point()
```



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

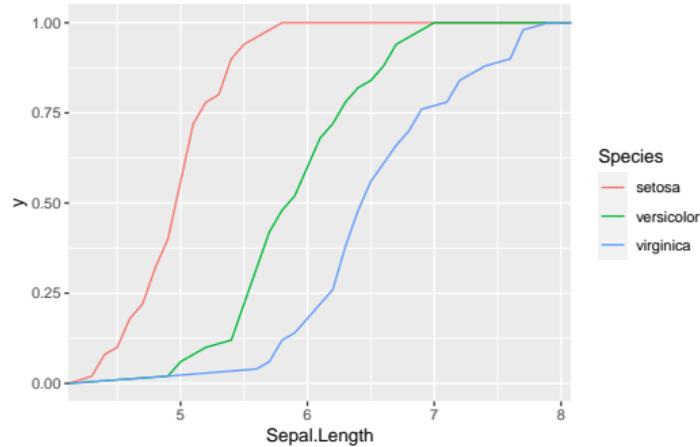
Plot the distribution of the sepal length of each species of iris using:

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

Layers: statistics

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

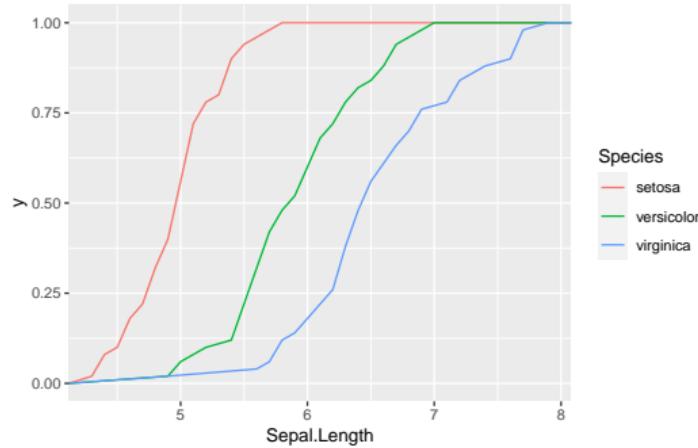
```
ggplot(iris) +  
  aes(x = Sepal.Length, colour = Species) +  
  geom_line(stat = "ecdf")
```



Layers: statistics

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

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

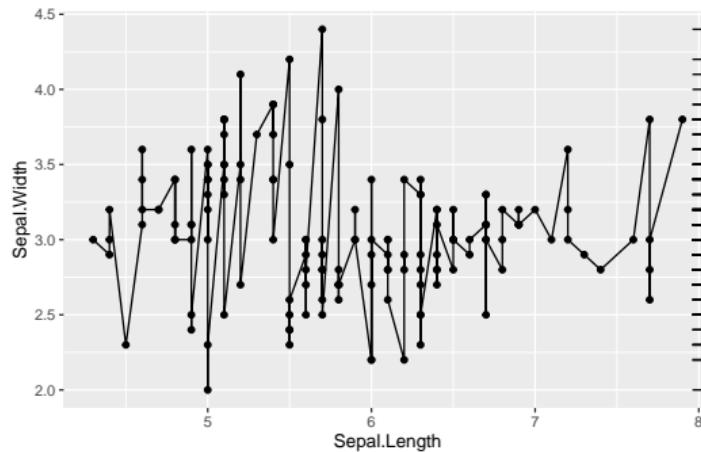


Note: plotting the Empirical Cumulative Distribution Function allows for deriving information from the distribution of the data (more on that later!)

Multiple layers

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

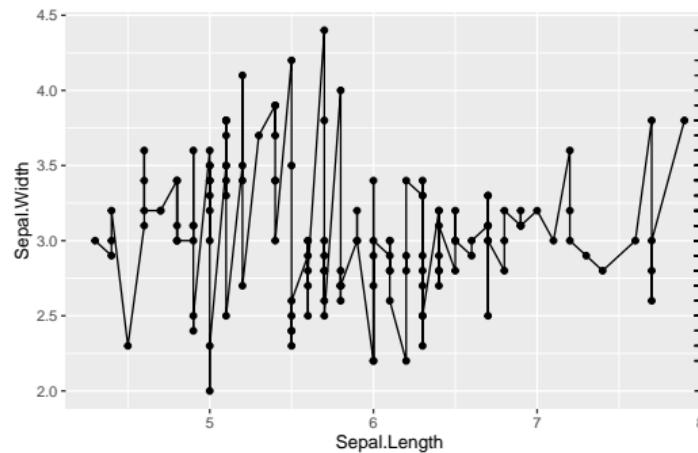
```
ggplot(iris) +  
  aes(x = Sepal.Length, y = Sepal.Width) +  
  geom_point() +  
  geom_line() +  
  geom_rug(sides = "r") ## r for right!
```



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:

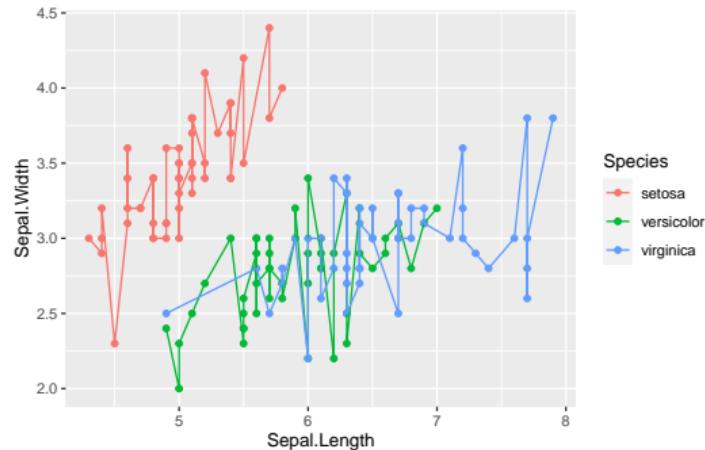
- position: x, y, xmin, xmax, ymin, ymax...
- colours: colour and fill
- transparency: alpha
- sizes: size and width
- shape: shape and linetype

Tip: check the help of the geom(s) 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:

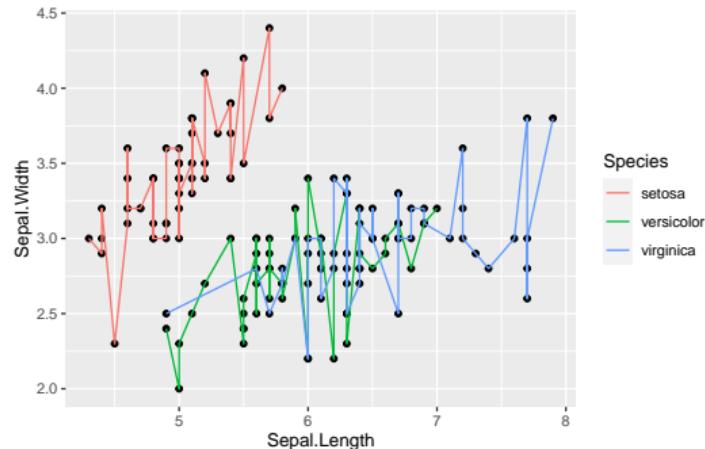
```
ggplot(iris) +  
  aes(x = Sepal.Length, y = Sepal.Width, colour = Species) +  
  geom_point() +  
  geom_line()
```



Aesthetics: where to provide the aesthetic mappings?

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

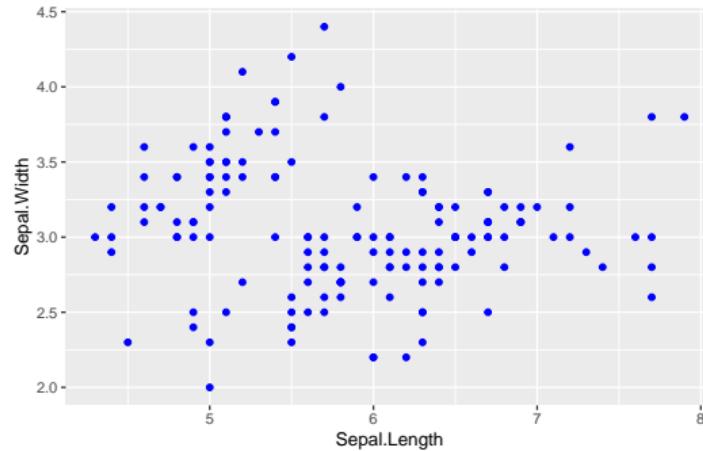
```
ggplot(iris) +  
  aes(x = Sepal.Length, y = Sepal.Width) +  
  geom_point() +  
  geom_line(aes(colour = Species))
```



Aesthetics: where to provide the aesthetic mappings?

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

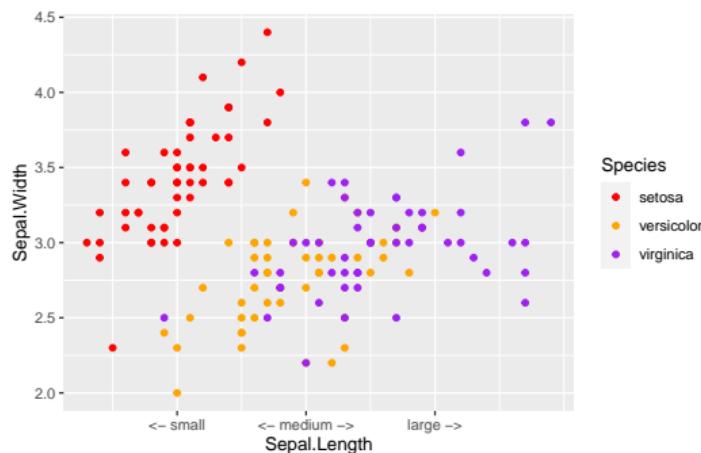
```
ggplot(iris) +  
  aes(x = Sepal.Length, y = Sepal.Width) +  
  geom_point(colour = "blue")
```



Scales

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

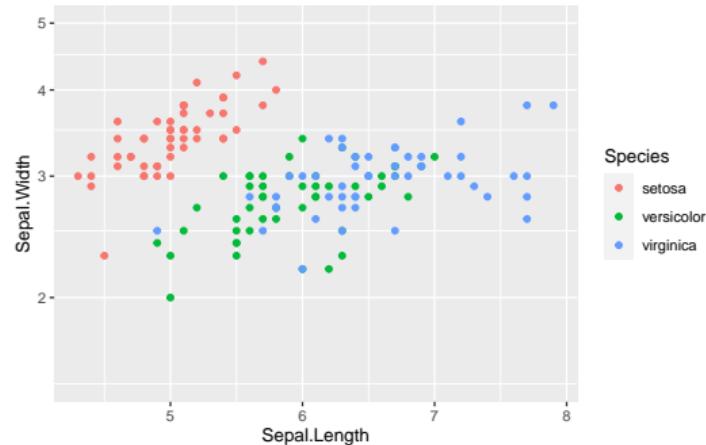
```
ggplot(iris) +  
  aes(x = Sepal.Length, y = Sepal.Width, colour = Species) +  
  geom_point() +  
  scale_x_continuous(breaks = c(5, 6, 7), labels = c("<- small", "<- medium ->", "large ->")) +  
  scale_colour_manual(values = c("red", "orange", "purple"))
```



Coordinates

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

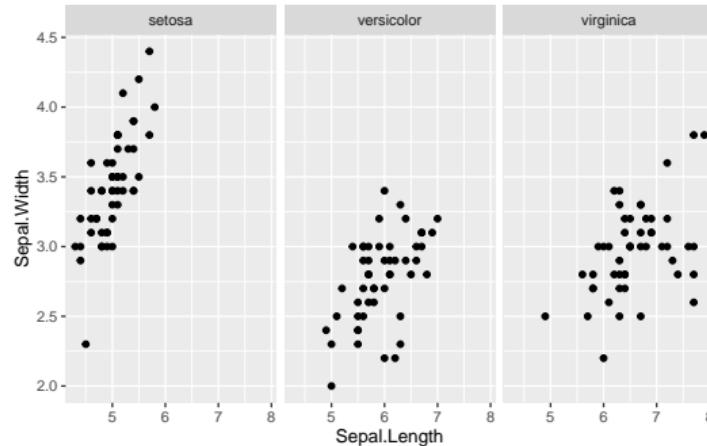
```
ggplot(iris) +  
  aes(x = Sepal.Length, y = Sepal.Width, colour = Species) +  
  geom_point() +  
  coord_trans(y = "log10", ylim = c(1.5, 5))
```



Facets

You can use `facet_*`() to show different subsets 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

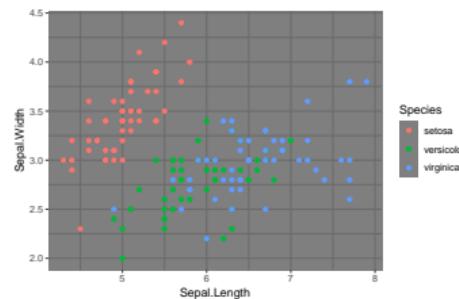
There are many predefined themes in `{ggplot2}` and companion packages (e.g. `{ggthemes}`):

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

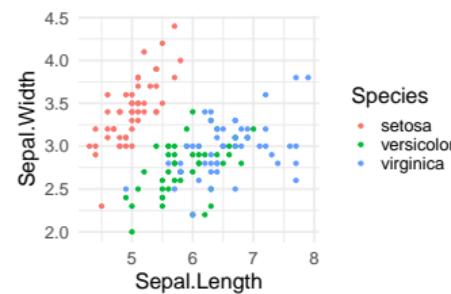
`myplot +
 theme_void()`



`myplot +
 theme_dark()`



`myplot +
 theme_minimal(base_size = 20)`



Themes: pre-defined themes

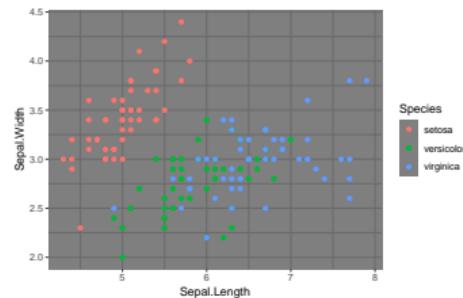
There are many predefined themes in `{ggplot2}` and companion packages (e.g. `{ggthemes}`):

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

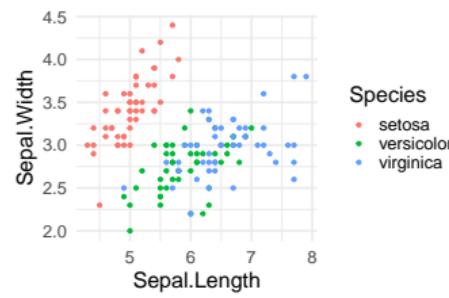
```
myplot +
  theme_void()
```



```
myplot +
  theme_dark()
```



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

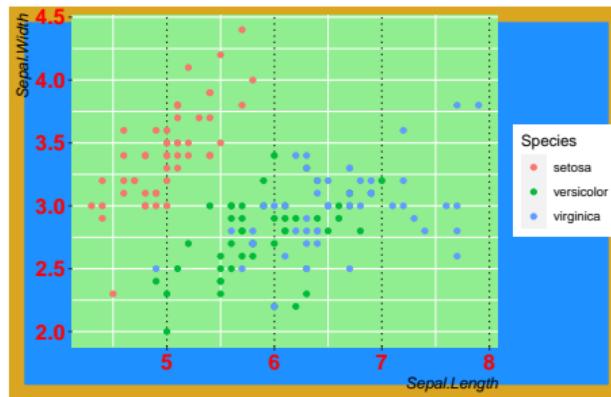


Tip: `base_size` controls the base font size for any theme

Themes: modifying themes

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

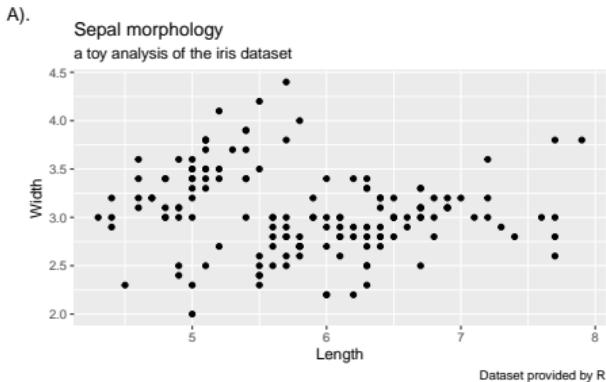
```
ggplot(iris) +  
  aes(x = Sepal.Length, y = Sepal.Width, colour = Species) +  
  geom_point() +  
  theme(axis.text = element_text(size = 15, face = "bold", colour = "red"),  
        axis.title = element_text(face = "italic", hjust = 1),  
        panel.grid.major.x = element_line(linetype = "dotted",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

- 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

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 follows:

```
?"for"
```

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

You can create your own R functions!

Creating functions only requires you to name the function & to rely on placeholders (i.e. arguments) to pass values:

```
OddsRatio <- 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!

Creating functions only requires you to name the function & to rely on placeholders (i.e. arguments) to pass values:

```
OddsRatio <- 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:

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

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

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} or {updater} on Windows

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} or {updater} on Windows
- 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

- 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

Useful resources

RStudio cheatsheets; e.g.:

Base R Cheat Sheet

Getting Help

- Accessing the help files
- mean**: Get help of a particular function.
- help.search("weighted mean")**: Search the help files for a word or phrase.
- help(package = "dplyr")**: Find help for a package.
- More about an object**

Using Packages

- install.packages("dplyr")**: Download and install a package from CRAN.
- library(dplyr)**: Load the package into the session, making all its functions available to use.
- dplyr::select**: Use a particular function from a package.
- data(titanic)**: Load a built-in dataset into the environment.

Working Directory

- getwd()**: Find the current working directory (where inputs are found and outputs are sent).
- setwd("C:/file/path")**: Change the current working directory.
- Use projects in RStudio to set the working directory to the folder you are working in.**

Vectors

Creating Vectors

<code>c(2, 4, 6)</code>	2 4 6	All elements have a vector
<code>2:6</code>	2 3 4 5 6	Integer sequence
<code>seq(2, 3, by=0.5)</code>	2.0 2.5 3.0	A complex sequence
<code>rep(1:2, times=3)</code>	1 2 1 2 1 2	Repeat a vector
<code>rep(1:2, each=3)</code>	1 1 1 2 2 2	Repeat elements of a vector

Programming

For Loop

```
for (variable in sequence){
  Do something
}
```

While Loop

```
while (condition){
  Do something
}
```

Vector Functions

sort(x)	Returns sorted.	rev(x)	Returns reversed.
table(x)	See counts of values.	unique(x)	See unique values.

If Statements

```
if (condition){
  Do something
} else {
  Do something different
}
```

Functions

```
function_name <- function(var){
  Do something
  return(new_variables)
}
```

Selecting Vector Elements

By Position

<code>x[4]</code>	The fourth element.
<code>x[-4]</code>	All but the fourth.
<code>x[2:4]</code>	Elements two to four.

By Value

<code>x[x == 10]</code>	Elements which are equal to 10.
<code>x[x < 0]</code>	All elements less than zero.
<code>x[x %in% c(1, 2, 5)]</code>	Elements in the set 1, 2, 5.

Named Vectors

<code>x["apple"]</code>	Element with name "apple".
-------------------------	----------------------------

Reading and Writing Data

Also see the [readr package](#).

Input	Output	Description
<code>df <- read.table("file.txt")</code>	<code>write.table(df, "file.txt")</code>	Read and write a delimited text file.
<code>df <- read.csv("file.csv")</code>	<code>write.csv(df, "file.csv")</code>	Read and write a comma-separated value file. This is a special case of readTable/writeTable.
<code>load("file.RData")</code>	<code>save(df, file = "file.RData")</code>	Read and write an R data file, a file type special for R.

Conditions

<code>a == b</code>	Are equal	<code>a > b</code>	Greater than	<code>a >= b</code>	Greater than or equal to
<code>a != b</code>	Not equal	<code>a < b</code>	Less than	<code>a <= b</code>	Less than or equal to

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

Useful resources

Official documentation:

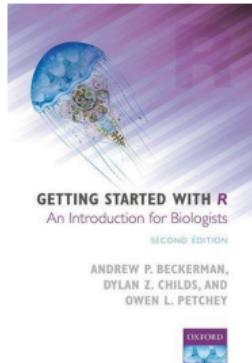
- the help files: every single (exported) function has a help file associated with it!
- official manuals: <https://cran.r-project.org/manuals.html>

Useful resources

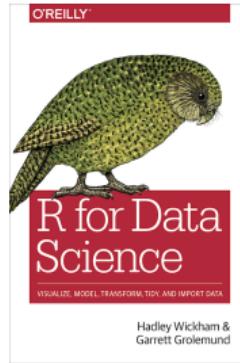
Official documentation:

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

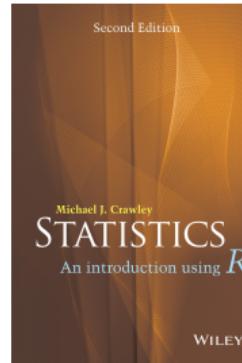
Best books for you (roughly sorted by amount of conceptual content):



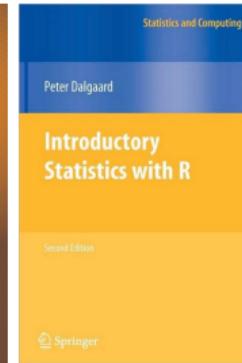
~ 30 €



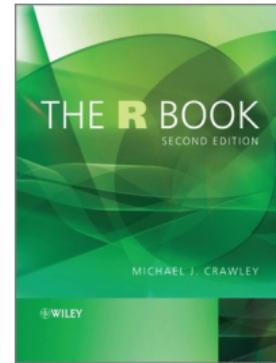
~ 60 €



~ 35 €



~ 40 €



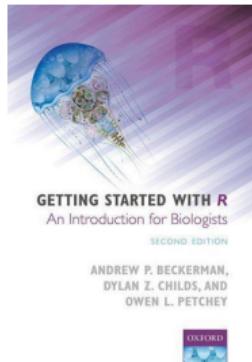
~ 60 €

Useful resources

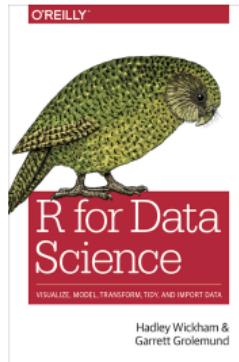
Official documentation:

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

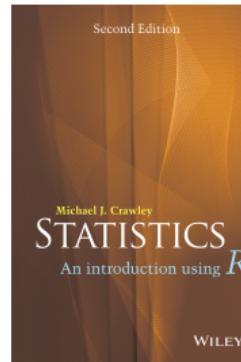
Best books for you (roughly sorted by amount of conceptual content):



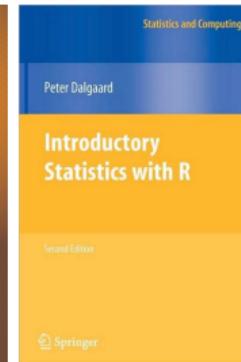
~ 30 €



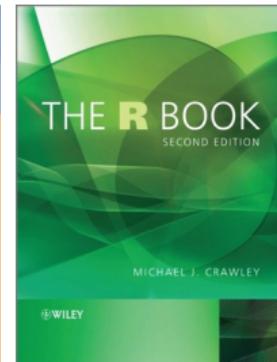
~ 60 €



~ 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 (<https://www.meetup.com/Berlin-R-Users-Group>)
- R-Ladies (<https://rladies.org/germany-rladies>)
- SATRDAYS (<https://satrdays.org/events>)
- European R Users Meeting (<https://erum.io>)
- rstudio::conf (<https://posit.co/conference>)
- useR! (<https://www.r-project.org/conferences>)

Become part of the R community!

- face-to-face interactions:

- Meetup Berlin R Users Group (<https://www.meetup.com/Berlin-R-Users-Group>)
- R-Ladies (<https://rladies.org/germany-rladies>)
- SATRDAYS (<https://satrdays.org/events>)
- European R Users Meeting (<https://erum.io>)
- rstudio::conf (<https://posit.co/conference>)
- useR! (<https://www.r-project.org/conferences>)

- websites:

- <https://stackoverflow.com/questions/tagged/r>
- <https://community.rstudio.com>
- <https://www.r-bloggers.com>
- <https://rfor datascience.slack.com>

Become part of the R community!

- face-to-face interactions:

- Meetup Berlin R Users Group (<https://www.meetup.com/Berlin-R-Users-Group>)
- R-Ladies (<https://rladies.org/germany-rladies>)
- SATRDAYS (<https://satrdays.org/events>)
- European R Users Meeting (<https://erum.io>)
- rstudio::conf (<https://posit.co/conference>)
- useR! (<https://www.r-project.org/conferences>)

- websites:

- <https://stackoverflow.com/questions/tagged/r>
- <https://community.rstudio.com>
- <https://www.r-bloggers.com>
- <https://rfor datascience.slack.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 (<https://www.meetup.com/Berlin-R-Users-Group>)
- R-Ladies (<https://rladies.org/germany-rladies>)
- SATRDAYS (<https://satrdays.org/events>)
- European R Users Meeting (<https://erum.io>)
- rstudio::conf (<https://posit.co/conference>)
- useR! (<https://www.r-project.org/conferences>)

- websites:

- <https://stackoverflow.com/questions/tagged/r>
- <https://community.rstudio.com>
- <https://www.r-bloggers.com>
- <https://rfor datascience.slack.com>

- mailing lists:

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

- twitter:

- #rstats
- #tidyverse
- @rdataberlin (and check the people I follow in there)

- mastodon:

- fosstodon.org
- #rstats

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!