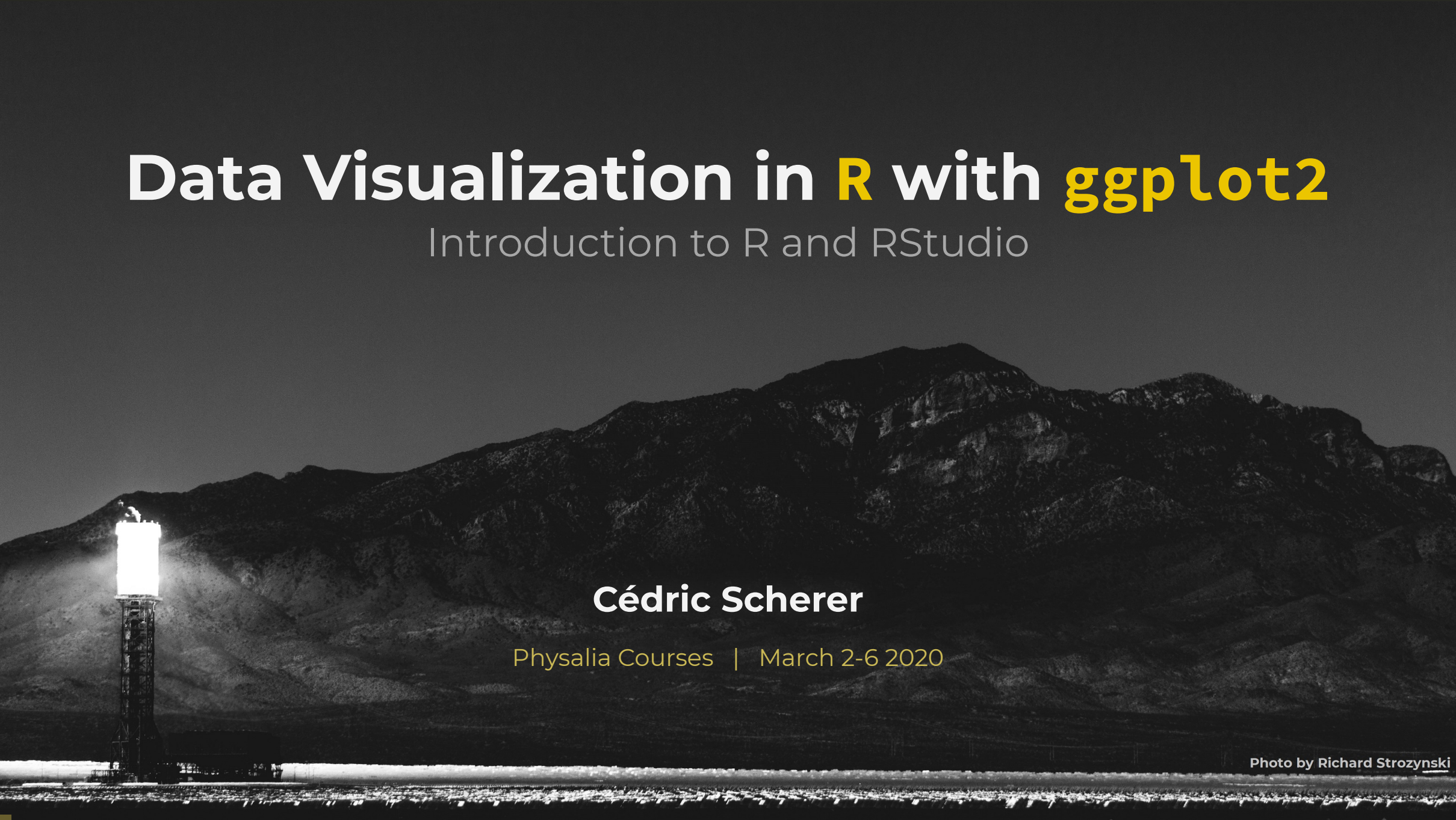
Data Visualization in R with ggplot2

Introduction to R and RStudio



Cédric Scherer

Physalia Courses | March 2-6 2020

Photo by Richard Strozynski

cedricscherer.netlify.com

@CedScherer @Z3tt

About Me

- **B.Sc. in *Life Sciences*** (2008-2011 in Potsdam)
- **M.Sc. in *Ecology, Evolution & Nature Conservation*** (2011-2014 in Potsdam)

About Me

- **B.Sc. in *Life Sciences*** (2008-2011 in Potsdam)
- **M.Sc. in *Ecology, Evolution & Nature Conservation*** (2011-2014 in Potsdam)
- **Ph.D. in *Ecology*** (2015-2019 in Berlin)
- **PostDoc in *Computational Ecology*** at Leibniz Institute for Zoo and Wildlife Research (IZW) in Berlin
  - movement ecology of animals
  - dynamics of populations and communities
  - ecology of wildlife diseases

About Me

- **B.Sc. in *Life Sciences*** (2008-2011 in Potsdam)
- **M.Sc. in *Ecology, Evolution & Nature Conservation*** (2011-2014 in Potsdam)
- **Ph.D. in *Ecology*** (2015-2019 in Berlin)
- **PostDoc in *Computational Ecology*** at Leibniz Institute for Zoo and Wildlife Research (IZW) in Berlin
  - movement ecology of animals
  - dynamics of populations and communities
  - ecology of wildlife diseases
- **Data Visualization Designer** (Practitioner/Engineer/Expert/...)
  - own and collaborative research data *(for a living)*
  - freelancing client projects *(for a living)*
  - data challenges and personal projects *(for fun & training)*

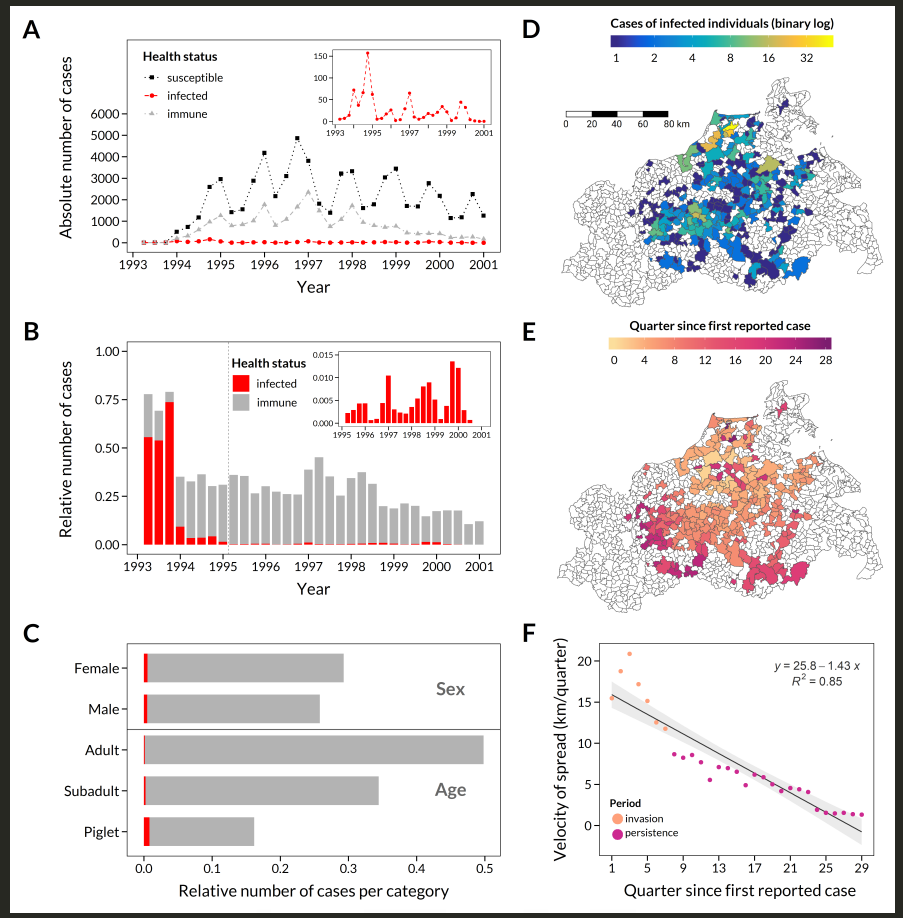
About Me

For the preparation and visualization of all these datasets,
I use mostly `R` in combination with the `tidyverse` and `ggplot2`.

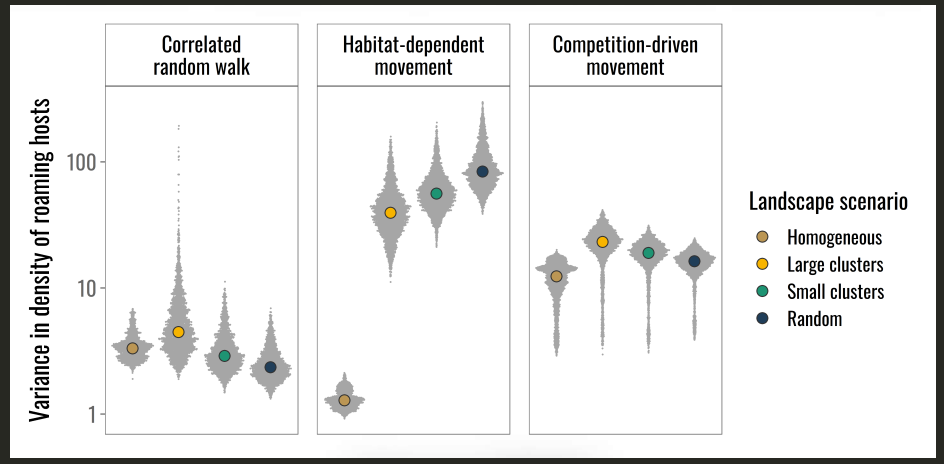
Source: dicook.org/files/rstudio/#3

Your Turn!

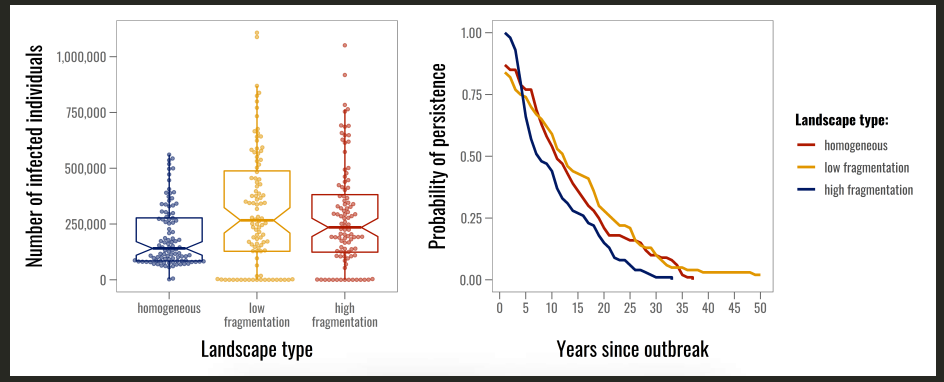
Data Visualizations for Scientific Publications & Talks



Scherer et al. 2019 Journal of Animal Ecology

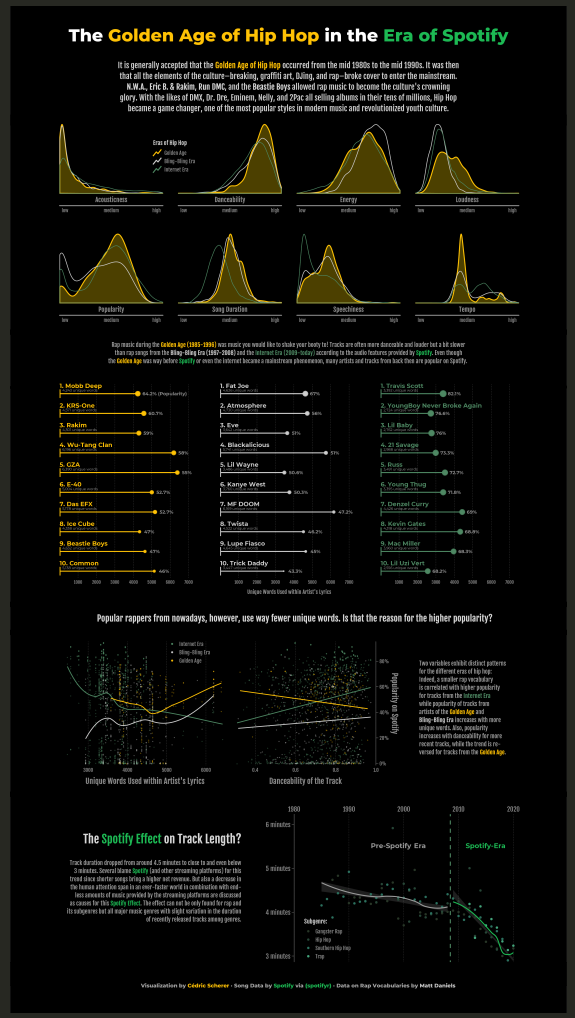


Scherer et al. 2020 Oikos

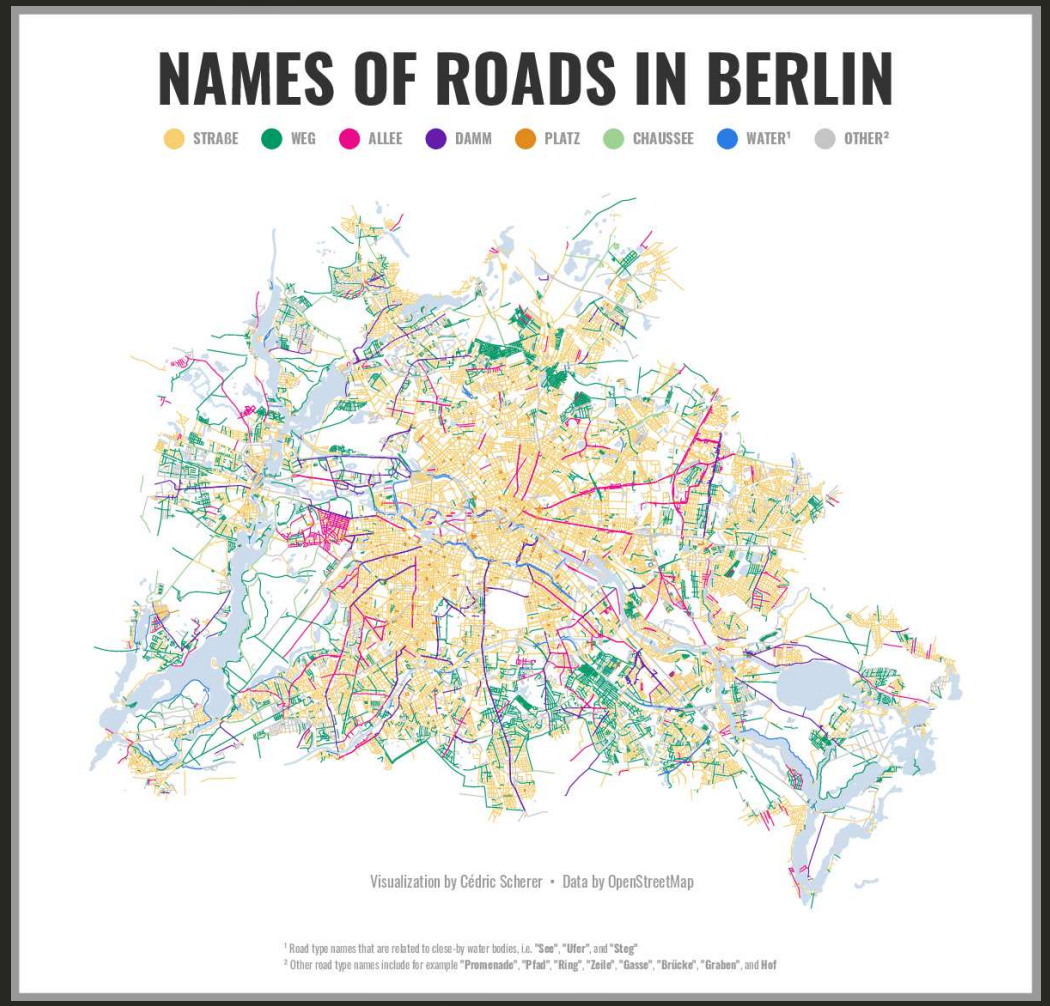


Sciaini et al. 2019 Methods in Ecology & Evolution

Data Visualizations as Challenge Contributions

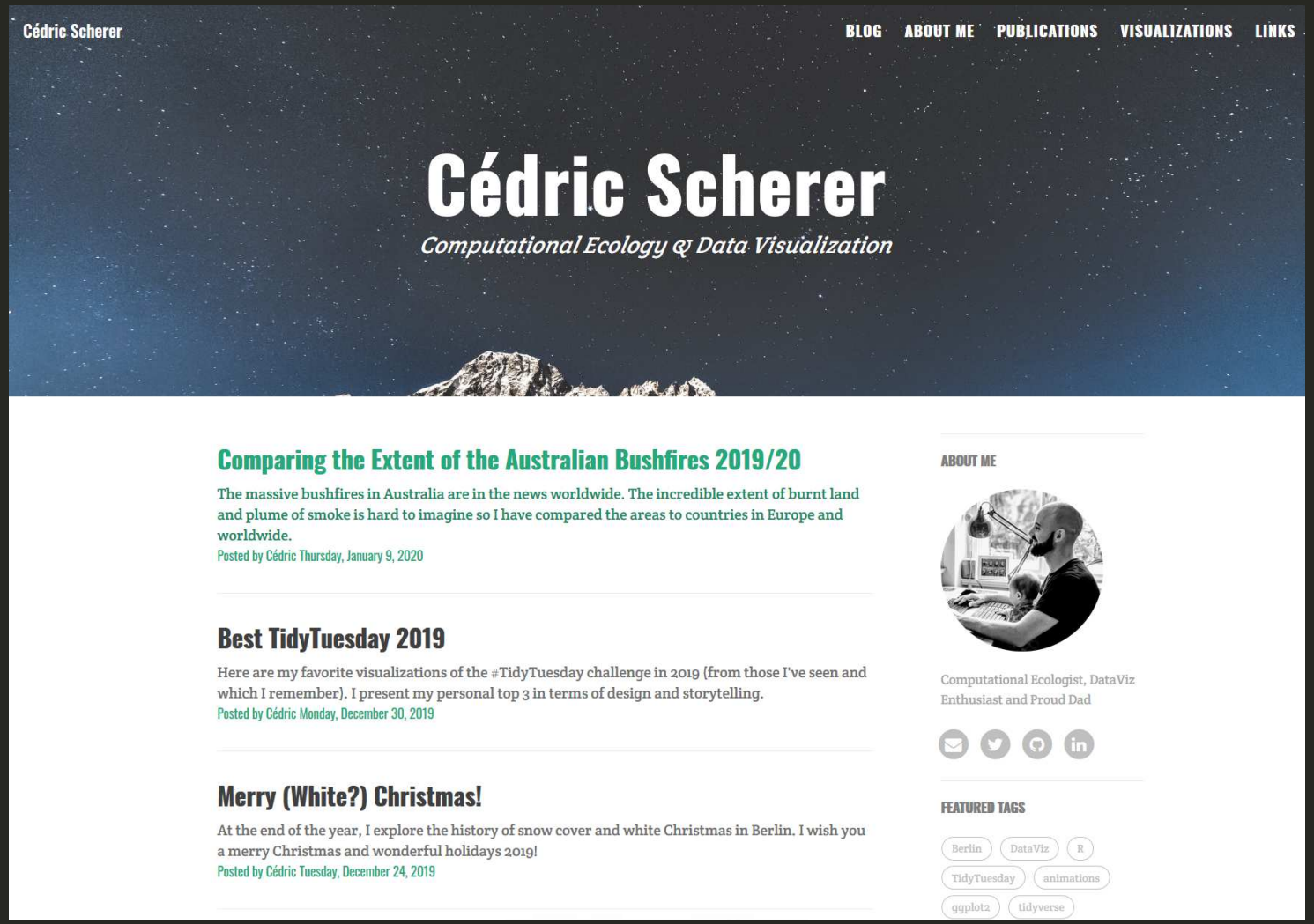


Contribution to #TidyTuesday



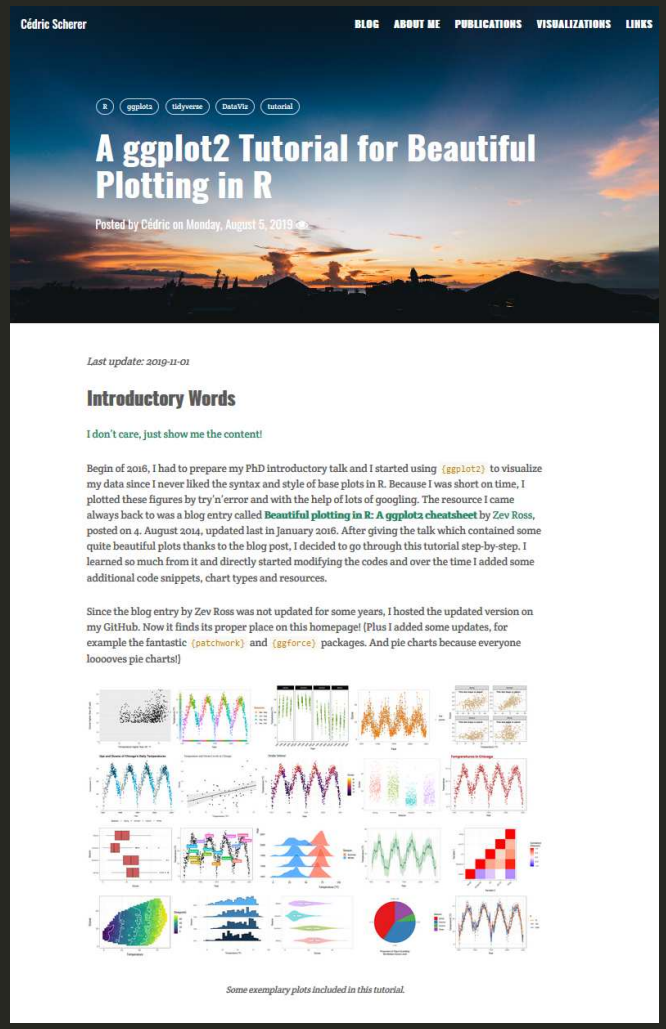
Contribution to #30DaymapChallenge

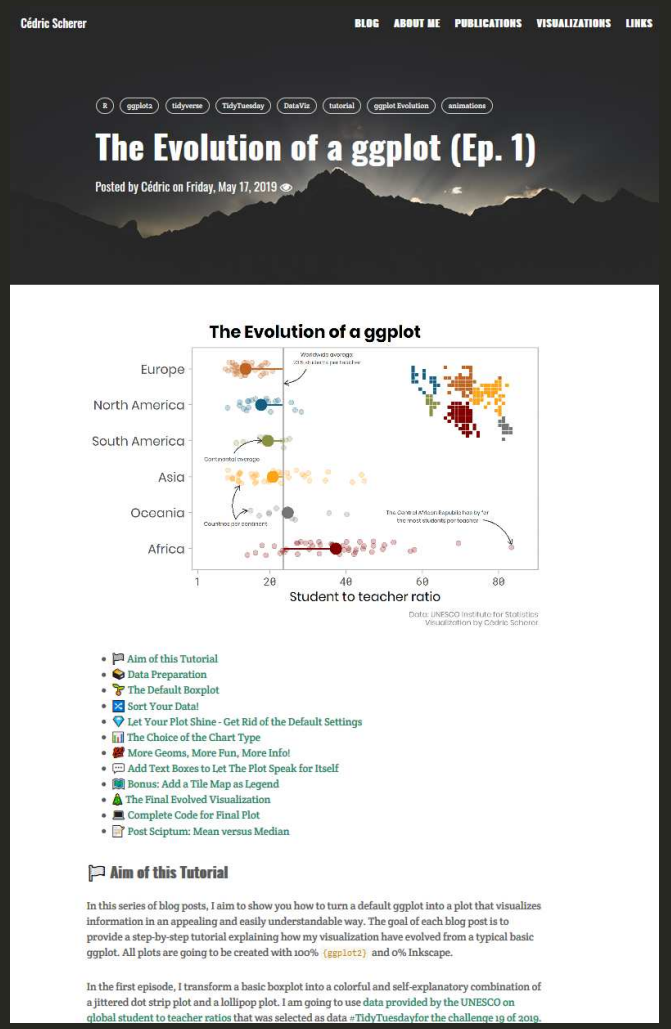
My Blog



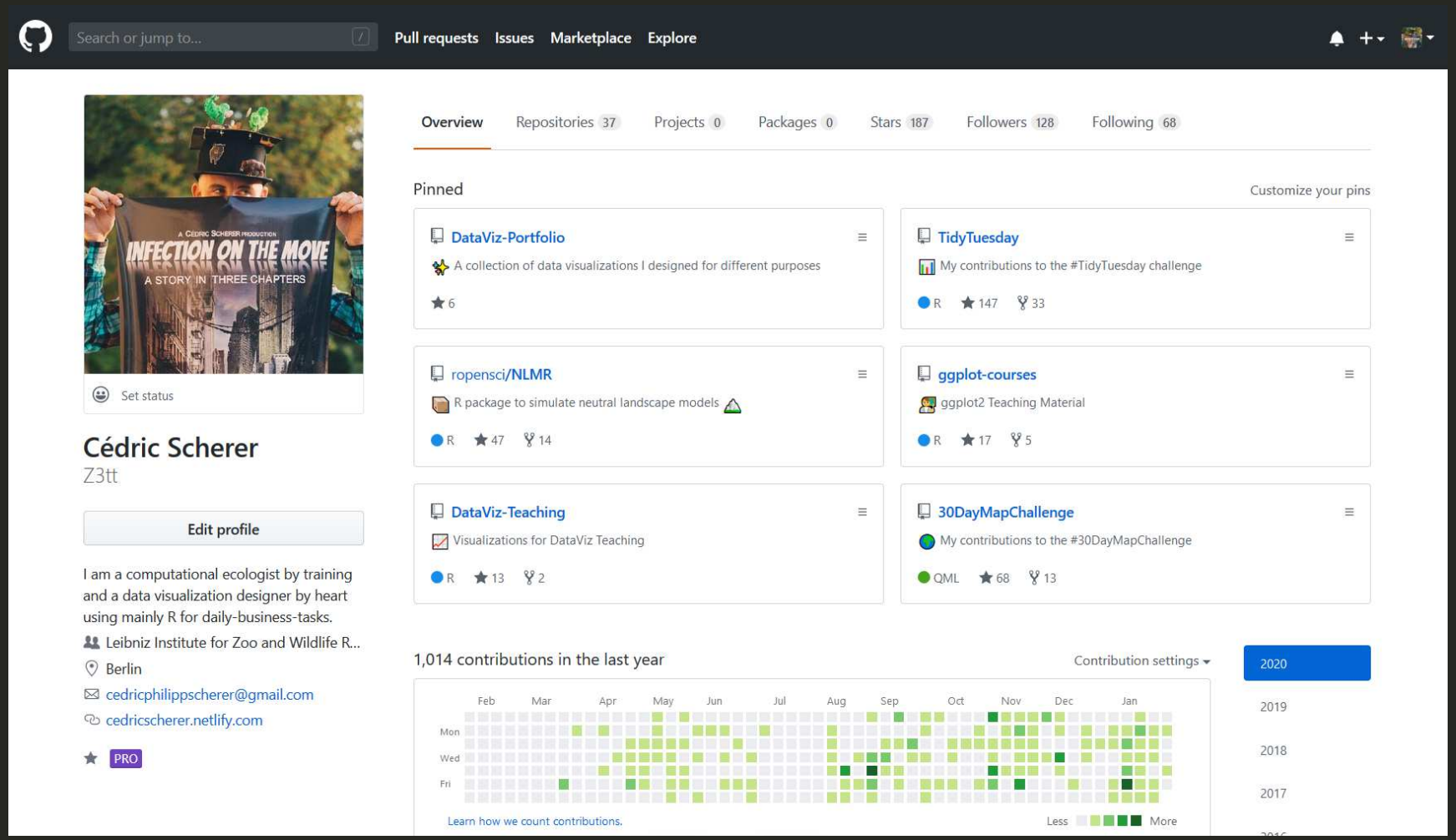
cedricscherer.netlify.com

My Blog


Cédric Scherer
BLOG ABOUT ME PUBLICATIONS VISUALIZATIONS LINKS
R ggplot2 tidyverse DataViz tutorial
A ggplot2 Tutorial for Beautiful Plotting in R
Posted by Cédric on Monday, August 5, 2019
Last update: 2019-11-01
Introductory Words
I don't care, just show me the content!
Begin of 2016, I had to prepare my PhD introductory talk and I started using {ggplot2} to visualize my data since I never liked the syntax and style of base plots in R. Because I was short on time, I plotted these figures by try'n'error and with the help of lots of googling. The resource I came always back to was a blog entry called Beautiful plotting in R: A ggplot2 cheatsheet by Zev Ross, posted on 4. August 2014, updated last in January 2016. After giving the talk which contained some quite beautiful plots thanks to the blog post, I decided to go through this tutorial step-by-step. I learned so much from it and directly started modifying the codes and over the time I added some additional code snippets, chart types and resources.
Since the blog entry by Zev Ross was not updated for some years, I hosted the updated version on my GitHub. Now it finds its proper place on this homepage! (Plus I added some updates, for example the fantastic {patchwork} and {ggforce} packages. And pie charts because everyone looooves pie charts!)
Some exemplary plots included in this tutorial.


Cédric Scherer
BLOG ABOUT ME PUBLICATIONS VISUALIZATIONS LINKS
R ggplot2 tidyverse TidyTuesday DataViz tutorial ggplot Evolution animations
The Evolution of a ggplot (Ep. 1)
Posted by Cédric on Friday, May 17, 2019
The Evolution of a ggplot
Europe
North America
South America
Asia
Oceania
Africa
1
20
40
60
80
Student to teacher ratio
Aim of this Tutorial
Data Preparation
The Default Boxplot
Sort Your Data!
Let Your Plot Shine - Get Rid of the Default Settings
The Choice of the Chart Type
More Geoms, More Fun, More Info!
Add Text Boxes to Let The Plot Speak for Itself
Bonus: Add a Tile Map as Legend
The Final Evolved Visualization
Complete Code for Final Plot
Post Sciptum: Mean versus Median
Aim of this Tutorial
In this series of blog posts, I aim to show you how to turn a default ggplot into a plot that visualizes information in an appealing and easily understandable way. The goal of each blog post is to provide a step-by-step tutorial explaining how my visualization have evolved from a typical basic ggplot. All plots are going to be created with 100% {ggplot2} and 0% Inkscape.
In the first episode, I transform a basic boxplot into a colorful and self-explanatory combination of a jittered dot strip plot and a lollipop plot. I am going to use data provided by the UNESCO on global student to teacher ratios that was selected as data #TidyTuesdayfor the challenge 19 of 2019.

My GitHub



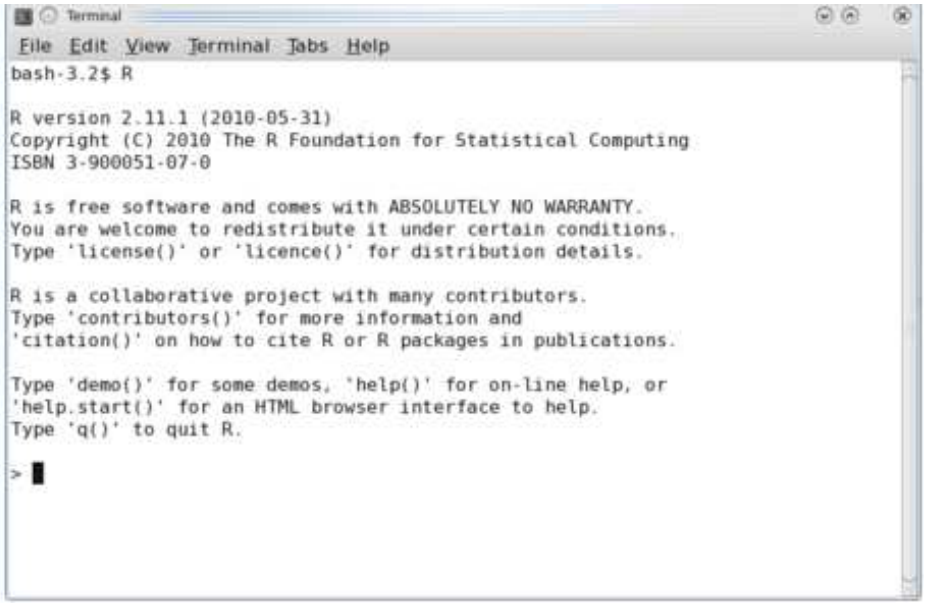
github.com/Z3tt

Introduction to R

What is R ?

A programming language and free software environment for statistical computing and graphics.

R is very popular among, scientists, statisticians, and data miners
for developing statistical software and analyse and visualize data



What is R?

R was developed by **Ross Ihaka** and **Robert Gentleman** as an open source implementation of the S programming language. R was conceived in 1992 and released in 1995.

What is R?

R was developed by **Ross Ihaka** and **Robert Gentleman** as an open source implementation of the S programming language. R was conceived in 1992 and released in 1995.

Pros:

- free and open source
- platform-independent
- dedicated packages that provide extra functionality
- highly compatible with many other programming languages
- huge online community
- often experienced as *easy to code* (from a non-programmer perspective)

What is R?

R was developed by **Ross Ihaka** and **Robert Gentleman** as an open source implementation of the S programming language. R was conceived in 1992 and released in 1995.

Pros:

- free and open source
- platform-independent
- dedicated packages that provide extra functionality
- highly compatible with many other programming languages
- huge online community
- often experienced as *easy to code* (from a non-programmer perspective)

Cons:

- data handling (physical memory storage + speed)
- security issues (relevant for web applications)
- often experienced as *strange to code* (from a programmer perspective)

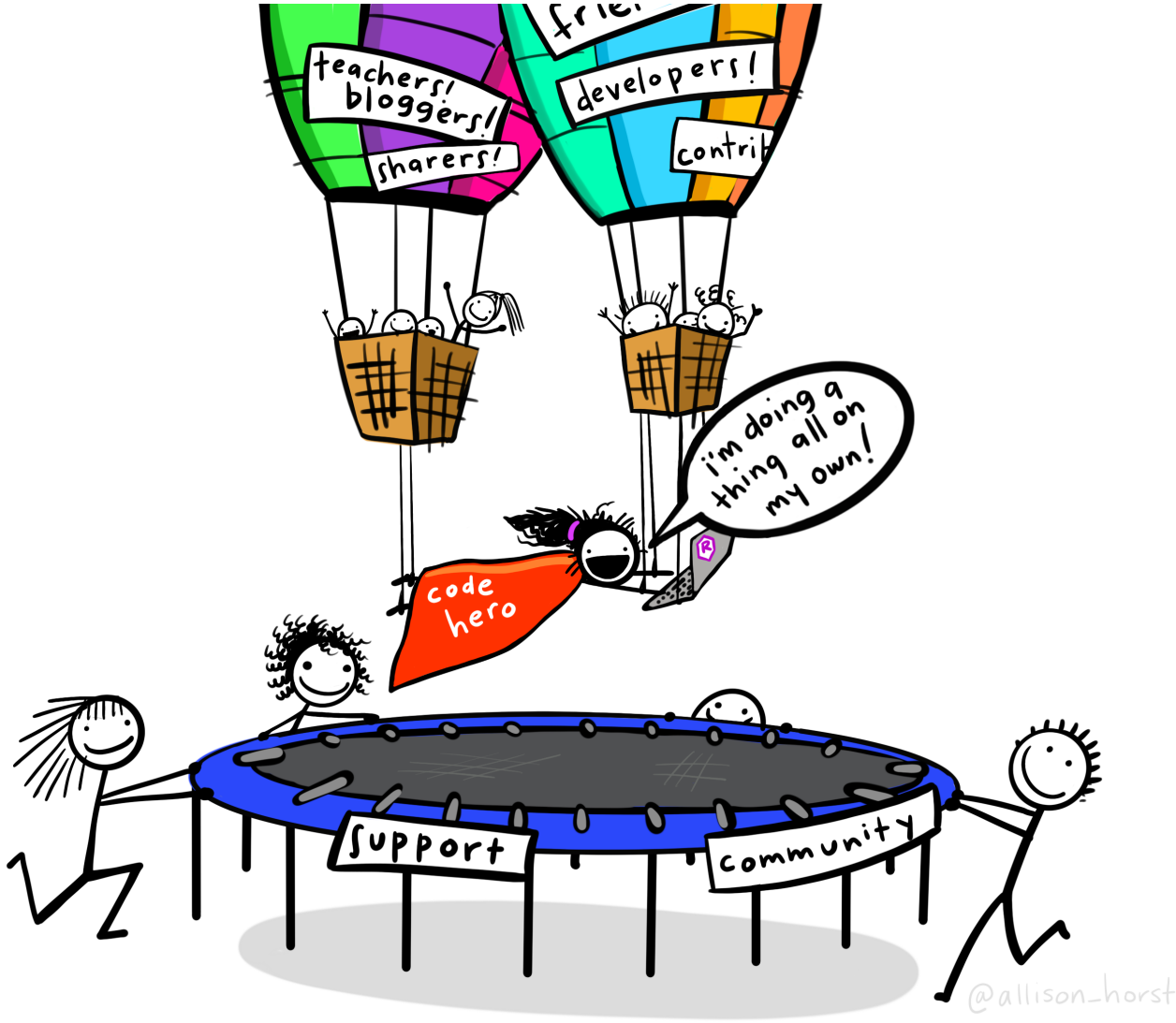


Illustration by Allison Horst (github.com/allisonhorst/stats-illustrations)

Getting Help with R

- R Foundation Help Page: r-project.org/help.html
- RStudio Community: community.rstudio.com
- R For Data Science Online Learning Community: rfordatasci.com
- Stack Overflow: stackoverflow.com
- R User Groups (local): jumpingrivers.github.io/meetingsR/r-user-groups.html
- R Ladies (global & local): rladies.org
- Twitter: #rstats, #tidyverse, #ggplot2

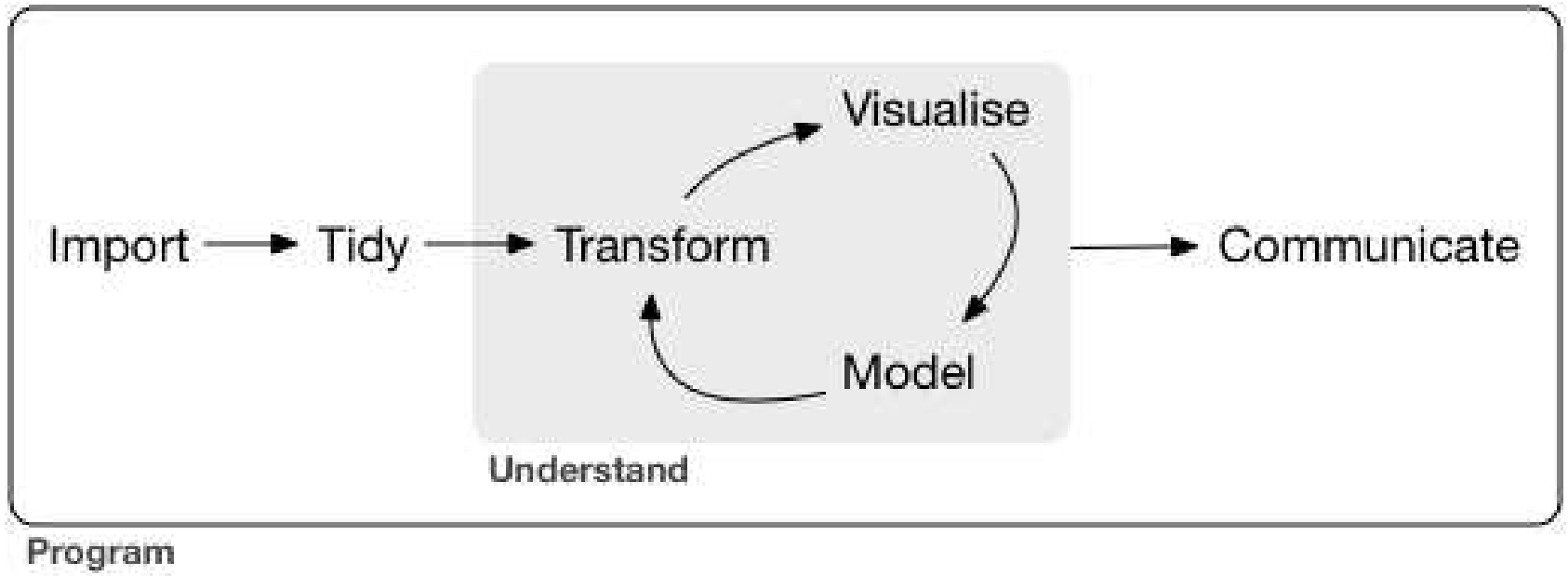
#rstats on Twitter

.pull-right[

- #rstats, #tidyverse, #ggplot2
- R For Data Science Online Learning Community: @R4DScommunity
- R Weekly: @rweekly_org
- ROpenSci: @rOpenSci
- TidyTuesday challenge: @thomas_mock and @tidypod
- R For The Rest Of Us: @rfortherest
- R Function A Day: @Rfunction_a_day
- Mara Averick: @dataandme

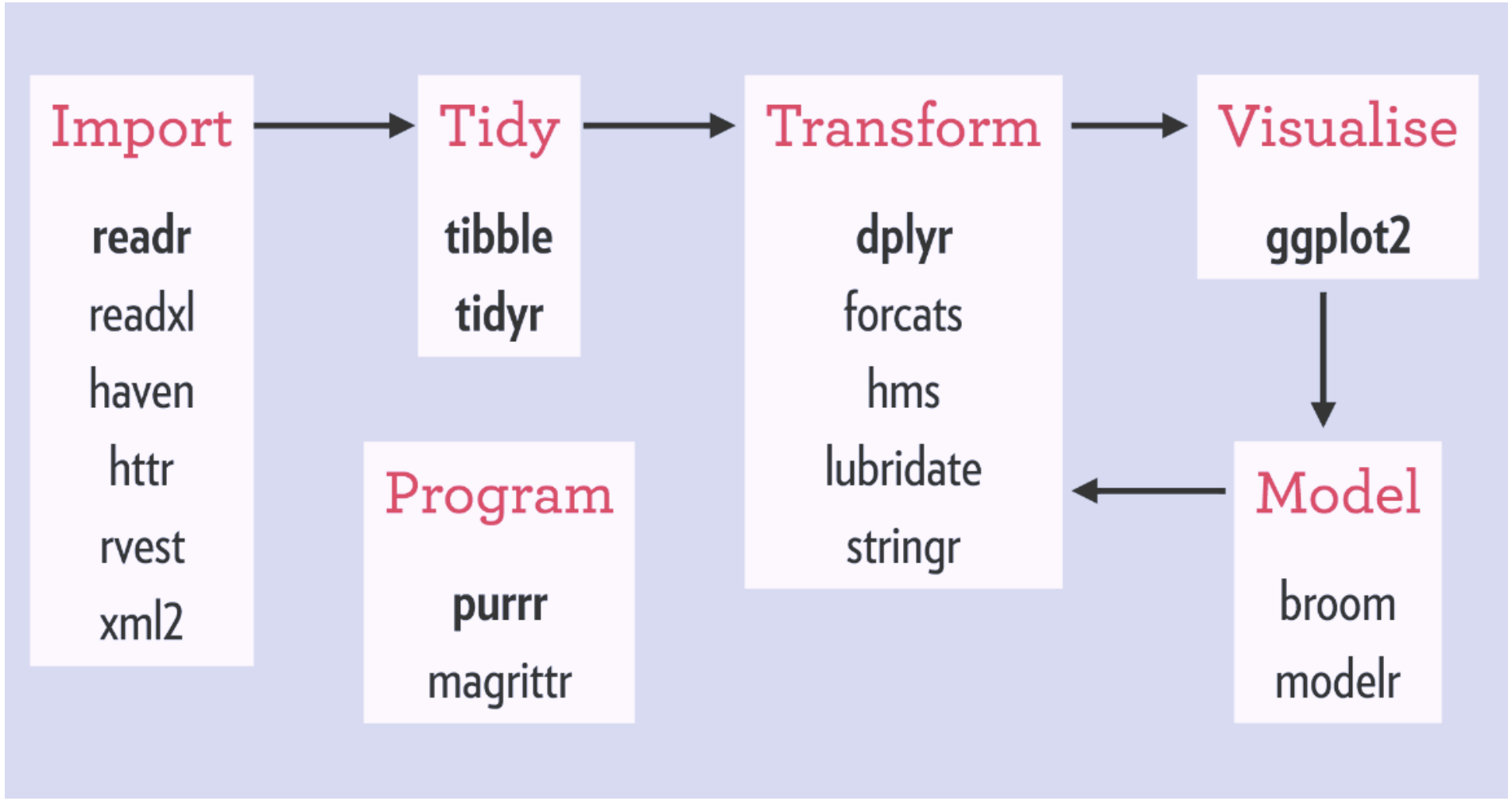
What is the `tidyverse`?

The `tidyverse` provides exemplary support for data wrangling and is the main reason for the recent popularity of `R`, especially in data-driven environments.



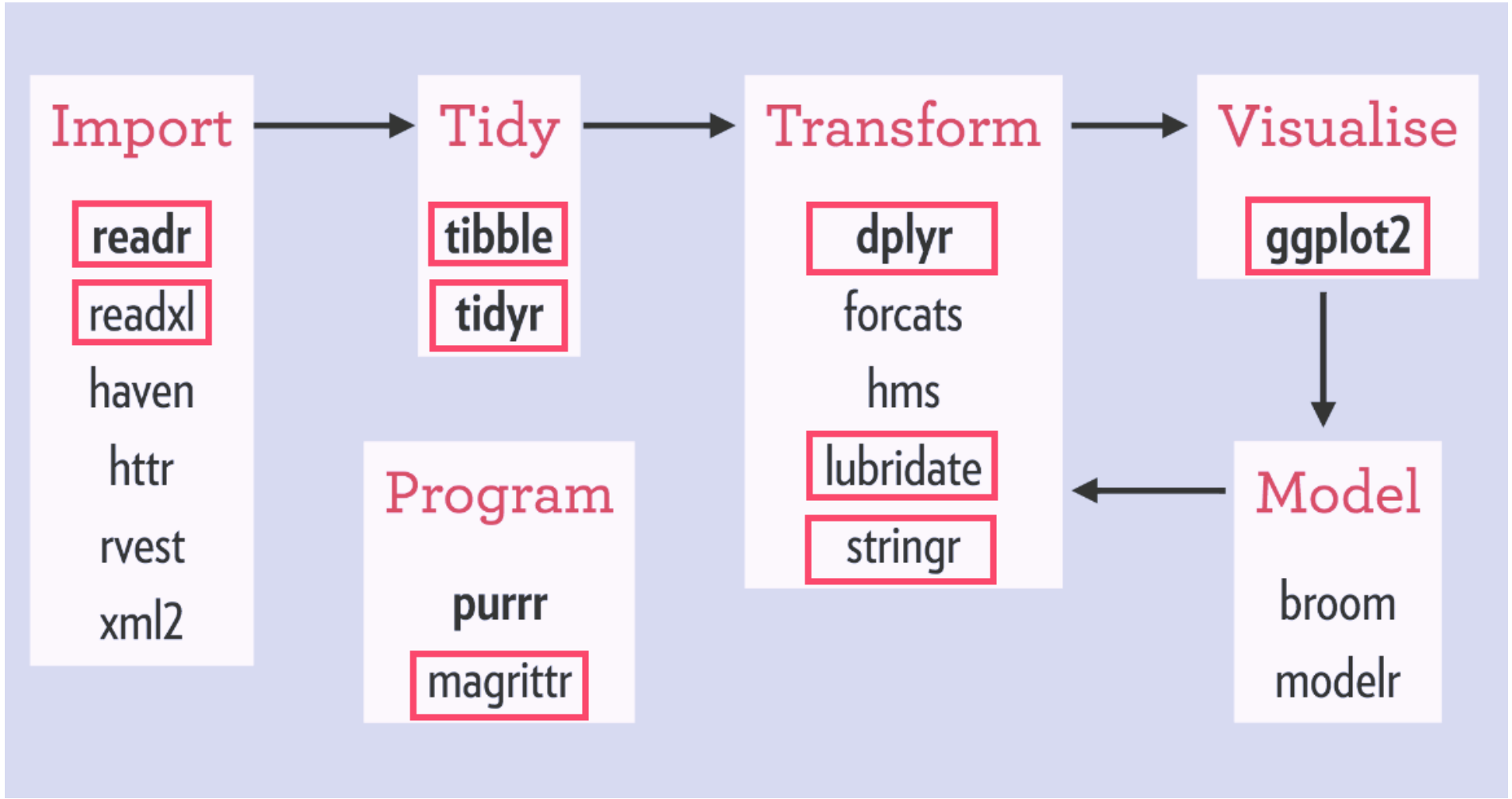
Source: Hadley Wickham's "R for Data Science" (R4DS)

The Typical Data Science Project Flow



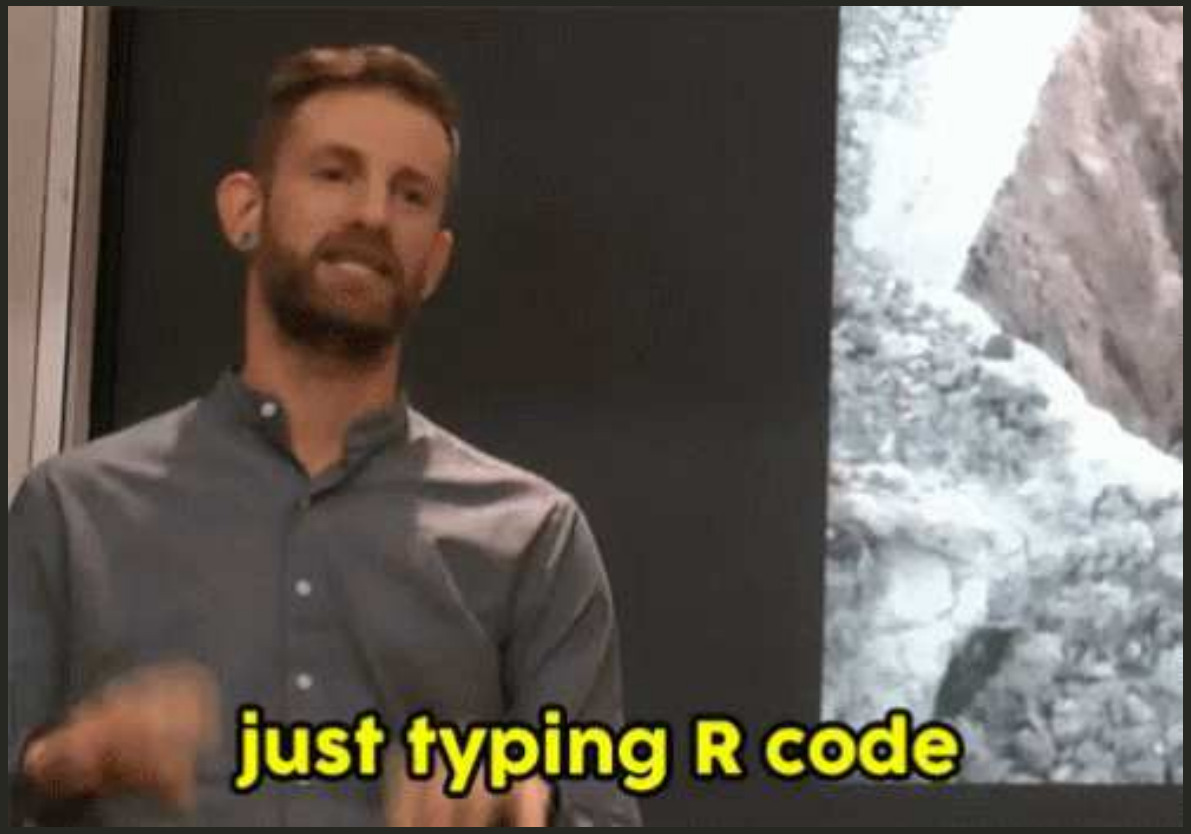
Source: www.rviews.rstudio.com/post/2017-06-09-What-is-the-tidyverse_files/tidyverse1.png

The Typical Data Science Project Flow



Source: www.rviews.rstudio.com/post/2017-06-09-What-is-the-tidyverse_files/tidyverse1.png

Who is Hadley Wickham?



Who is Hadley Wickham?

Statistician from New Zealand, living in Houston, TX

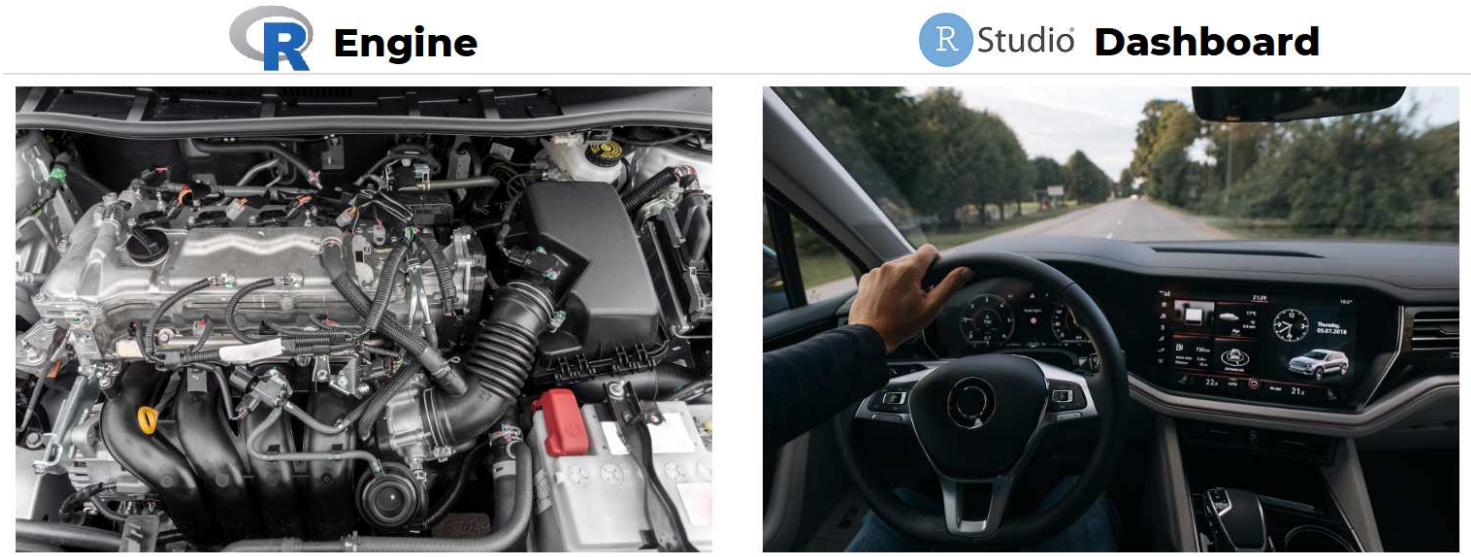
Adjunct Professor of Statistics at the University of Auckland,
Stanford University, and Rice University

Chief Scientist at RStudio

Inventor/Developer of {ggplot2} and the tidyverse

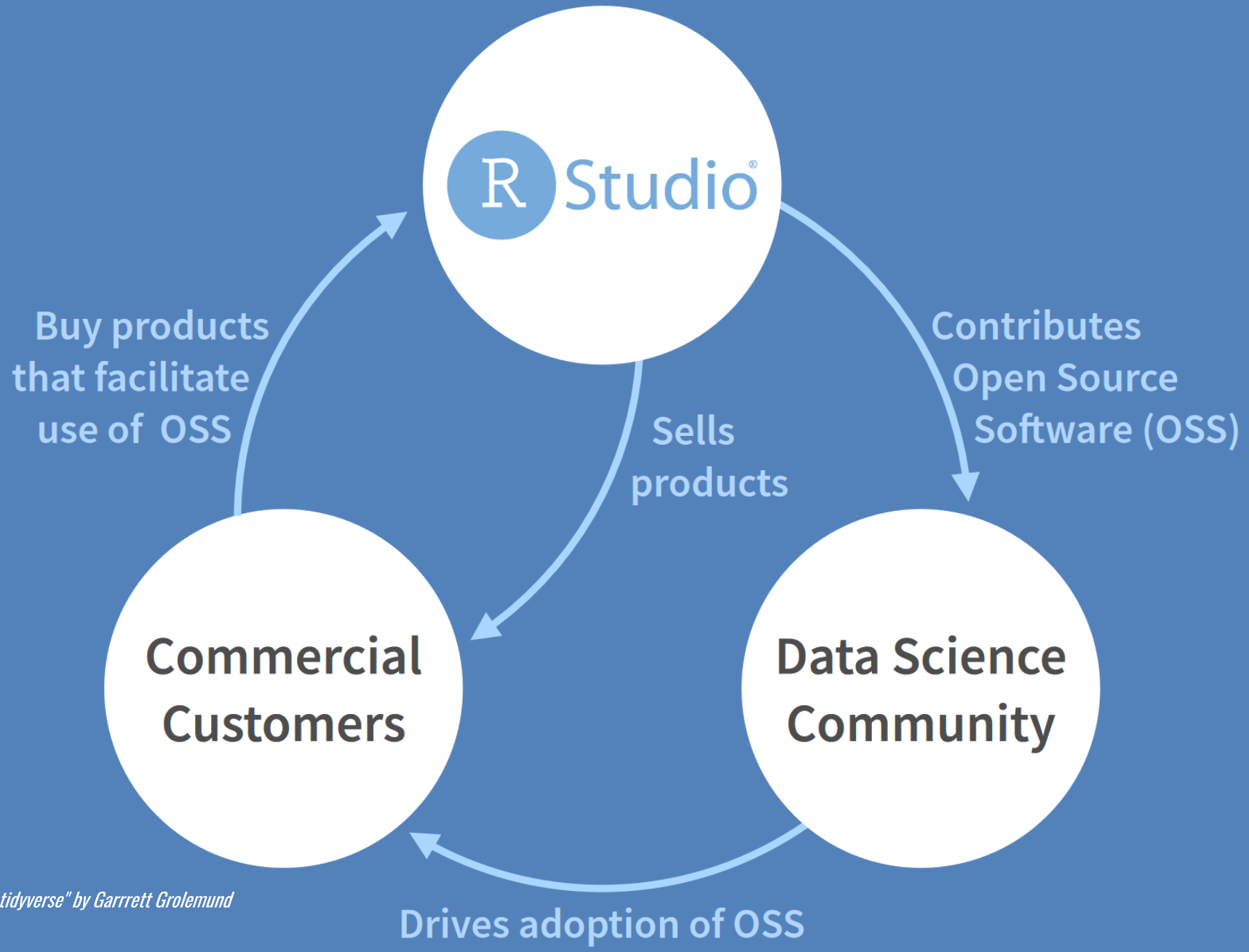
RStudio IDE

R and RStudio



Modified from moderndive.netlify.com/1-getting-started.html

- **RStudio** is an open-source IDE (integrated development environment) for R
- most popular R IDE since several years *(what is tinnR?!)*
- many features + extensions to facilitate workflows *(projects, notebooks, toc, ...)*
- availability of **R projects** and **Rmarkdown** *(later more)*



Source: "Remaster the tidyverse" by Garrrett Grolemund

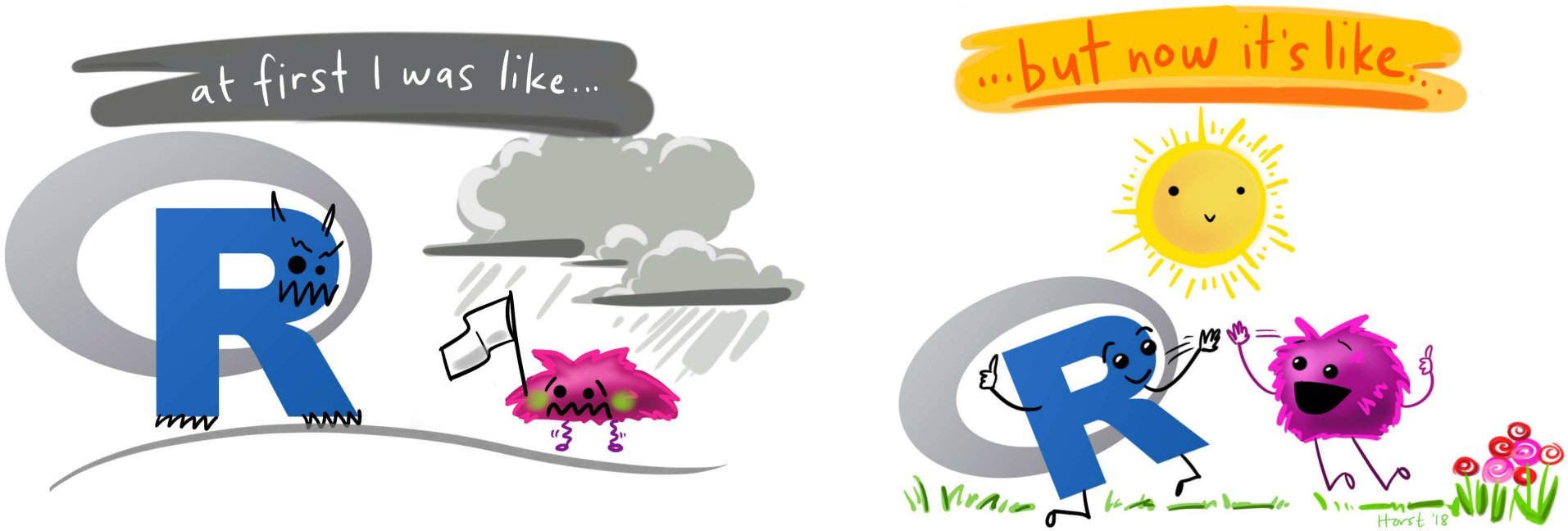


Illustration by Allison Horst (github.com/allisonhorst/stats-illustrations)

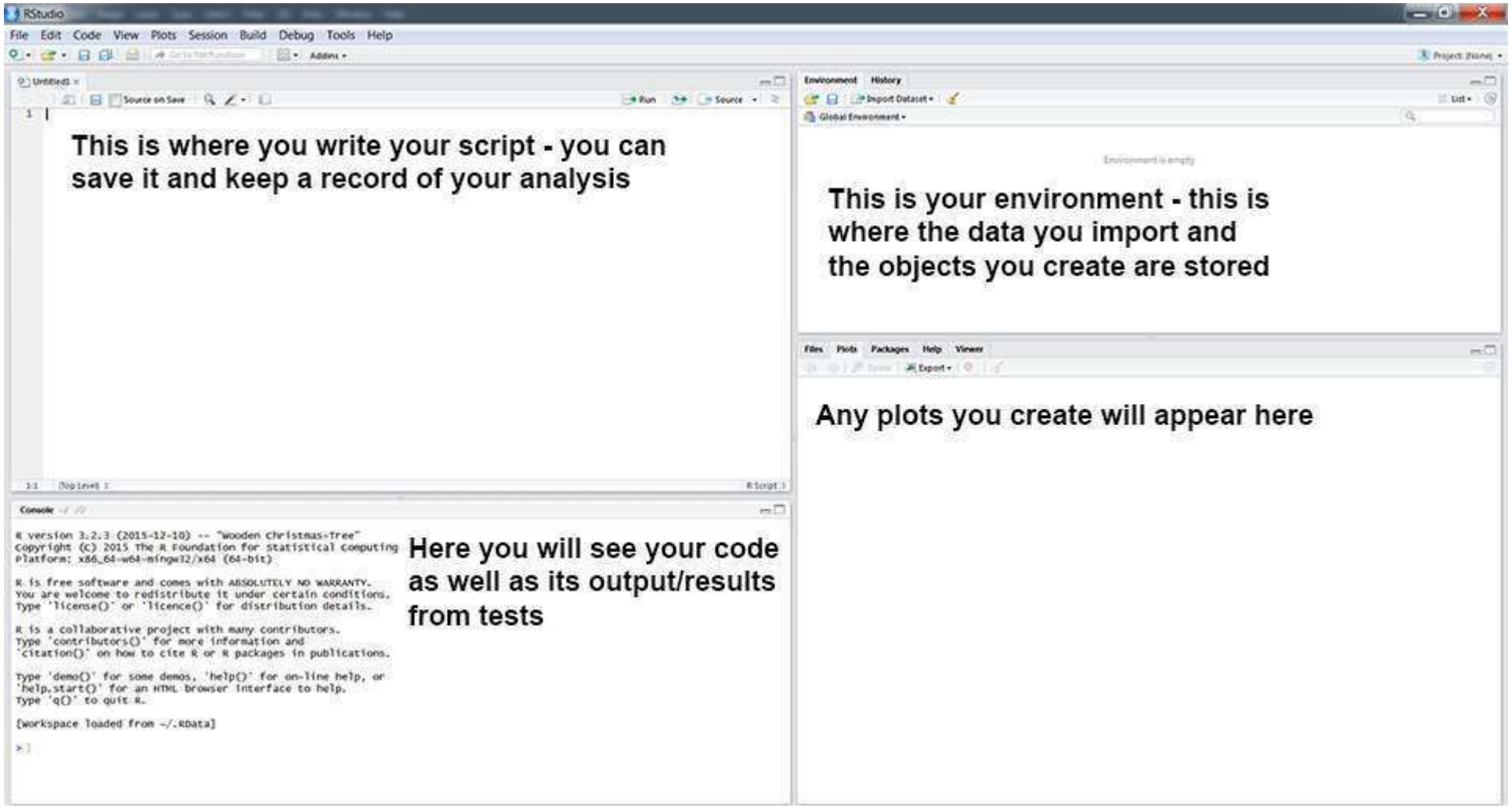
Installation of R and RStudio

- Download and install R via
  https://cloud.r-project.org/
- Download and install **RStudio Desktop** via
  https://www.rstudio.com/products/rstudio/download/

Your Turn!

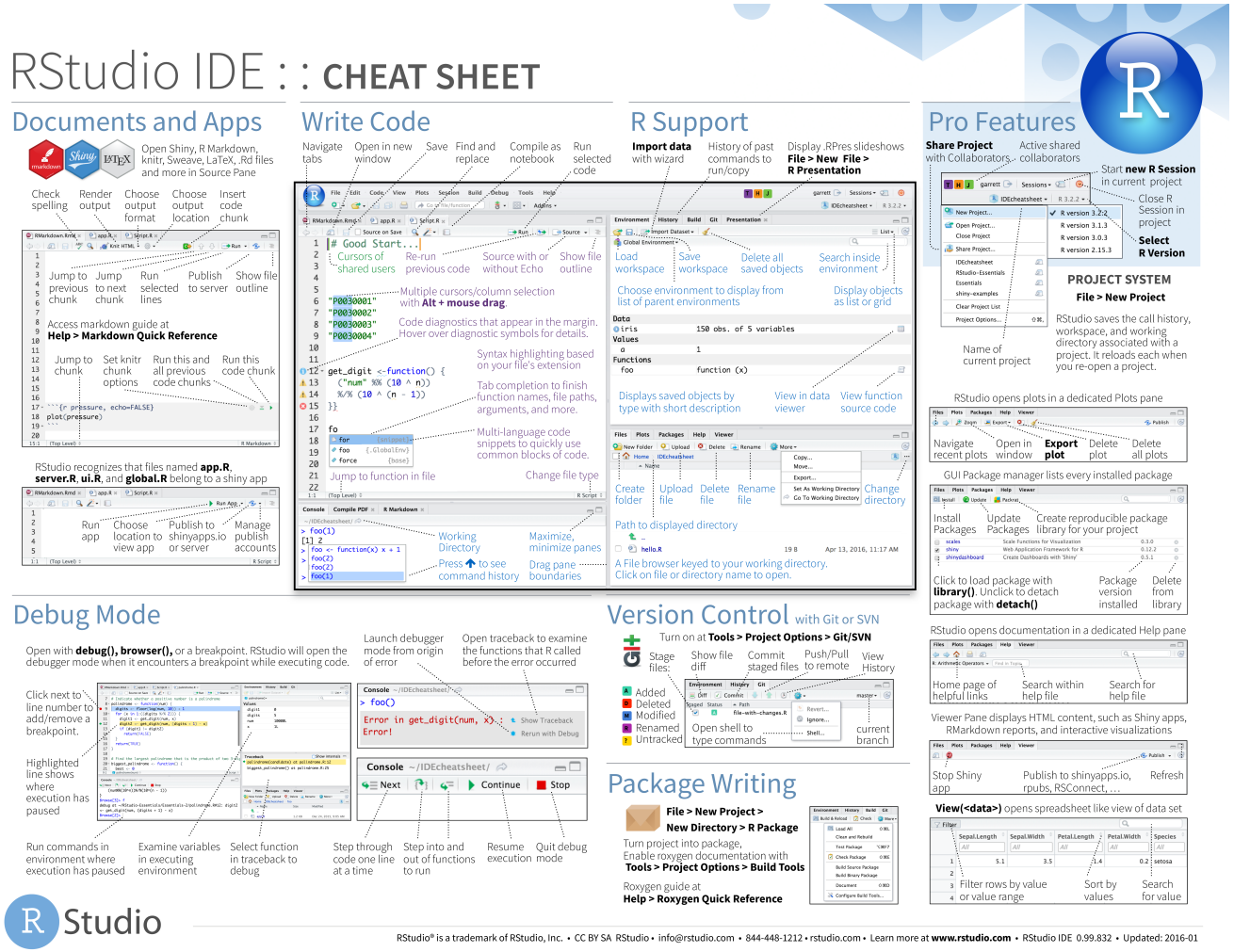
- Open RStudio and get familiar with its environment.
- Go to *Help > Cheatsheets > RStudio IDE Cheat Sheet* and study the document.

RStudio IDE



Source: ourcodingclub.github.io/2016/11/13/intro-to-r.html

RStudio IDE

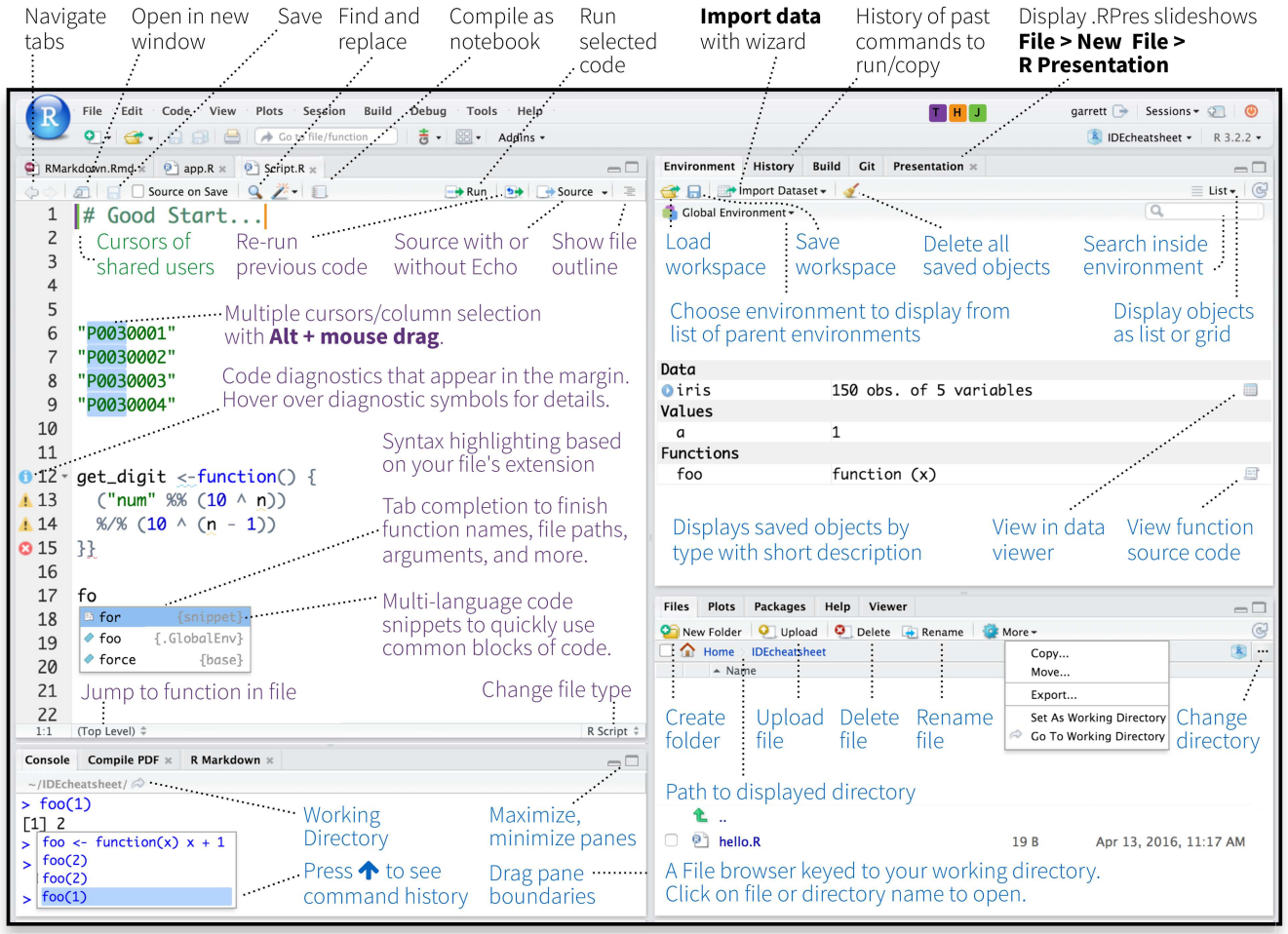


Source: rstudio.com/resources/cheatsheets/

RStudio IDE

Write Code

R Support



Source: rstudio.com/resources/cheatsheets/

RStudio IDE

1 LAYOUT

	Windows/Linux	Mac
Move focus to Source Editor	Ctrl+1	Ctrl+1
Move focus to Console	Ctrl+2	Ctrl+2
Move focus to Help	Ctrl+3	Ctrl+3
Show History	Ctrl+4	Ctrl+4
Show Files	Ctrl+5	Ctrl+5
Show Plots	Ctrl+6	Ctrl+6
Show Packages	Ctrl+7	Ctrl+7
Show Environment	Ctrl+8	Ctrl+8
Show Git/SVN	Ctrl+9	Ctrl+9
Show Build	Ctrl+0	Ctrl+0

2 RUN CODE

	Windows/Linux	Mac
Search command history	**Ctrl+↑**	**Cmd+↑**
Navigate command history	**↑/↓**	**↑/↓**
Move cursor to start of line	Home	Cmd+←
Move cursor to end of line	End	Cmd+→
Change working directory	Ctrl+Shift+H	Ctrl+Shift+H
Interrupt current command	**Esc**	**Esc**
Clear console	**Ctrl+L**	**Ctrl+L**
Quit Session (desktop only)	Ctrl+Q	Cmd+Q
Restart R Session	**Ctrl+Shift+F10**	**Cmd+Shift+F10**
Run current line/selection	**Ctrl+Enter**	**Cmd+Enter**
Run current (retain cursor)	Alt+Enter	Option+Enter
Run from current to end	Ctrl+Alt+E	Cmd+Option+E
Run the current function	Ctrl+Alt+F	Cmd+Option+F
Source a file	Ctrl+Alt+G	Cmd+Option+G
Source the current file	**Ctrl+Shift+S**	**Cmd+Shift+S**
Source with echo	Ctrl+Shift+Enter	Cmd+Shift+Enter

3 NAVIGATE CODE

	Windows /Linux	Mac
Goto File/Function	**Ctrl+.**	**Ctrl+.**
Fold Selected	Alt+L	Cmd+Option+L
Unfold Selected	Shift+Alt+L	Cmd+Shift+Option+L
Fold All	Alt+O	Cmd+Option+O
Unfold All	Shift+Alt+O	Cmd+Shift+Option+O
Go to line	Shift+Alt+G	Cmd+Shift+Option+G
Jump to	Shift+Alt+J	Cmd+Shift+Option+J
Switch to tab	Ctrl+Shift+.	Ctrl+Shift+.
Previous tab	Ctrl+F11	Ctrl+F11
Next tab	Ctrl+F12	Ctrl+F12
First tab	Ctrl+Shift+F11	Ctrl+Shift+F11
Last tab	Ctrl+Shift+F12	Ctrl+Shift+F12
Navigate back	Ctrl+F9	Cmd+F9
Navigate forward	Ctrl+F10	Cmd+F10
Jump to Brace	Ctrl+P	Ctrl+P
Select within Braces	Ctrl+Shift+Alt+E	Ctrl+Shift+Option+E
Use Selection for Find	Ctrl+F3	Cmd+E
Find in Files	Ctrl+Shift+F	Cmd+Shift+F
Find Next	Win: F3, Linux: Ctrl+G	Cmd+G
Find Previous	W: Shift+F3, L:	Cmd+Shift+G
Jump to Word	Ctrl+←/→	Option+←/→
Jump to Start/End	Ctrl+↑/↓	Cmd+↑/↓
Toggle Outline	Ctrl+Shift+O	Cmd+Shift+O

4 WRITE CODE

	Windows /Linux	Mac
Attempt completion	**Tab or Ctrl+Space**	**Tab or Cmd+Space**
Navigate candidates	**↑/↓**	**↑/↓**
Accept candidate	Enter, Tab, or→	Enter, Tab, or→
Dismiss candidates	Esc	Esc
Undo	Ctrl+Z	Cmd+Z
Redo	Ctrl+Shift+Z	Cmd+Shift+Z
Cut	Ctrl+X	Cmd+X
Copy	Ctrl+C	Cmd+C
Paste	Ctrl+V	Cmd+V
Select All	Ctrl+A	Cmd+A
Delete Line	Ctrl+D	Cmd+D
Select	Shift+[Arrow]	Shift+[Arrow]
Select Word	Ctrl+Shift+←/→	Option+Shift+←/→
Select to Line Start	Alt+Shift+←	Cmd+Shift+←
Select to Line End	Alt+Shift+→	Cmd+Shift+→
Select Page Up/Down	Shift+PageUp/Down	Shift+PageUp/Down
Select to Start/End	Shift+Alt+↑/↓	Cmd+Shift+↑/↓
Delete Word Left	Ctrl+Backspace	Ctrl+Opt+Backspace
Delete Word Right		Option+Delete
Delete to Line End		Ctrl+K
Delete to Line Start		Option+Backspace
Indent	Tab (at start of line)	Tab (at start of line)
Outdent	Shift+Tab	Shift+Tab
Yank line up to cursor	Ctrl+U	Ctrl+U
Yank line after cursor	Ctrl+K	Ctrl+K
Insert yanked text	Ctrl+Y	Ctrl+Y
Insert <-	**Alt+-**	**Option+-**
Insert %>%	**Ctrl+Shift+M**	**Cmd+Shift+M**
Show help for function	F1	F1
Show source code	F2	F2
New document	Ctrl+Shift+N	Cmd+Shift+N
New document (Chrome)	Ctrl+Alt+Shift+N	Cmd+Shift+Opt+N
Open document	Ctrl+O	Cmd+O
Save document	Ctrl+S	Cmd+S
Close document	Ctrl+W	Cmd+W
Close document (Chrome)	Ctrl+Alt+W	Cmd+Option+W
Close all documents	Ctrl+Shift+W	Cmd+Shift+W
Extract function	Ctrl+Alt+X	Cmd+Option+X
Extract variable	Ctrl+Alt+V	Cmd+Option+V
Reindent lines	Ctrl+I	Cmd+I
(Un)Comment lines	**Ctrl+Shift+C**	**Cmd+Shift+C**
Reflow Comment	Ctrl+Shift+/	Cmd+Shift+/
Reformat Selection	Ctrl+Shift+A	Cmd+Shift+A
Select within braces	Ctrl+Shift+E	Ctrl+Shift+E
Show Diagnostics	Ctrl+Shift+Alt+P	Cmd+Shift+Opt+P
Transpose Letters		Ctrl+T
Move Lines Up/Down	Alt+↑/↓	Option+↑/↓
Copy Lines Up/Down	Shift+Alt+↑/↓	Cmd+Option+↑/↓
Add New Cursor Above	Ctrl+Alt+Up	Ctrl+Option+Up
Add New Cursor Below	Ctrl+Alt+Down	Ctrl+Option+Down
Move Active Cursor Up	Ctrl+Alt+Shift+Up	Ctrl+Option+Shift+Up
Move Active Cursor Down	Ctrl+Alt+Shift+Down	Ctrl+Opt+Shift+Down
Find and Replace	Ctrl+F	Cmd+F
Use Selection for Find	Ctrl+F3	Cmd+E
Replace and Find	Ctrl+Shift+J	Cmd+Shift+J

WHY RSTUDIO SERVER PRO?

RSP extends the the open source server with a commercial license, support, and more:

- open and run multiple R sessions at once
- tune your resources to improve performance
- edit the same project at the same time as others
- see what you and others are doing on your server
- switch easily from one version of R to a different version
- integrate with your authentication, authorization, and audit practices

Download a free 45 day evaluation at
www.rstudio.com/products/rstudio-server-pro/

5 DEBUG CODE

	Windows/Linux	Mac
Toggle Breakpoint	Shift+F9	Shift+F9
Execute Next Line	F10	F10
Step Into Function	Shift+F4	Shift+F4
Finish Function/Loop	Shift+F6	Shift+F6
Continue	Shift+F5	Shift+F5
Stop Debugging	Shift+F8	Shift+F8

6 VERSION CONTROL

	Windows/Linux	Mac
Show diff	Ctrl+Alt+D	Ctrl+Option+D
Commit changes	Ctrl+Alt+M	Ctrl+Option+M
Scroll diff view	Ctrl+↑/↓	Ctrl+↑/↓
Stage/Unstage (Git)	Spacebar	Spacebar
Stage/Unstage and move to next	Enter	Enter

7 MAKE PACKAGES

	Windows/Linux	Mac
Build and Reload	Ctrl+Shift+B	Cmd+Shift+B
Load All (devtools)	**Ctrl+Shift+L**	**Cmd+Shift+L**
Test Package (Desktop)	**Ctrl+Shift+T**	**Cmd+Shift+T**
Test Package (Web)	Ctrl+Alt+F7	Cmd+Opt+F7
Check Package	Ctrl+Shift+E	Cmd+Shift+E
Document Package	**Ctrl+Shift+D**	**Cmd+Shift+D**

8 DOCUMENTS AND APPS

	Windows/Linux	Mac
Preview HTML (Markdown, etc.)	Ctrl+Shift+K	Cmd+Shift+K
Knit Document (knitr)	**Ctrl+Shift+K**	**Cmd+Shift+K**
Compile Notebook	Ctrl+Shift+K	Cmd+Shift+K
Compile PDF (TeX and Sweave)	Ctrl+Shift+K	Cmd+Shift+K
Insert chunk (Sweave and Knitr)	Ctrl+Alt+I	Cmd+Option+I
Insert code section	Ctrl+Shift+R	Cmd+Shift+R
Re-run previous region	Ctrl+Shift+P	Cmd+Shift+P
Run current document	Ctrl+Alt+R	Cmd+Option+R
Run from start to current line	**Ctrl+Alt+B**	**Cmd+Option+B**
Run the current code section	**Ctrl+Alt+T**	**Cmd+Option+T**
Run previous Sweave/Rmd code	Ctrl+Alt+P	Cmd+Option+P
Run the current chunk	Ctrl+Alt+C	Cmd+Option+C
Run the next chunk	Ctrl+Alt+N	Cmd+Option+N
Sync Editor & PDF Preview	Ctrl+F8	Cmd+F8
Previous plot	Ctrl+Alt+F11	Cmd+Option+F11
Next plot	Ctrl+Alt+F12	Cmd+Option+F12
Show Keyboard Shortcuts	**Alt+Shift+K**	**Option+Shift+K**

RStudio

RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more at **www.rstudio.com** • RStudio IDE 0.1.0 • Updated: 2017-09

Source: rstudio.com/resources/cheatsheets/

Hello World!

Your Turn!

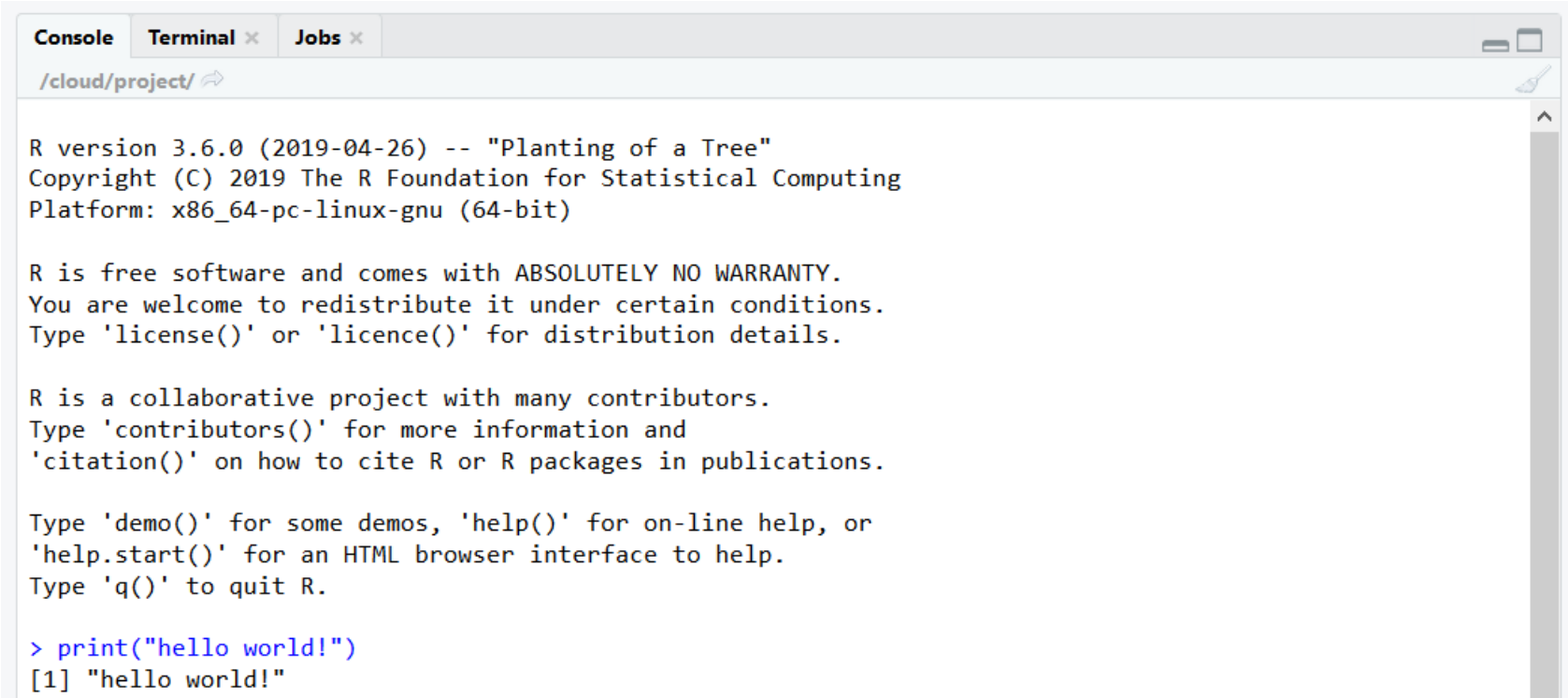
- Open a script by clicking *File > New File > R script*
- Type the classic `print("hello world!")` and hit the *Run* button.
  (alternatively: `Ctrl/Cmd + Enter`)

- What happens if you just run "hello world!"?

The Console

R code you are running appears here - the line starts with > (by default).

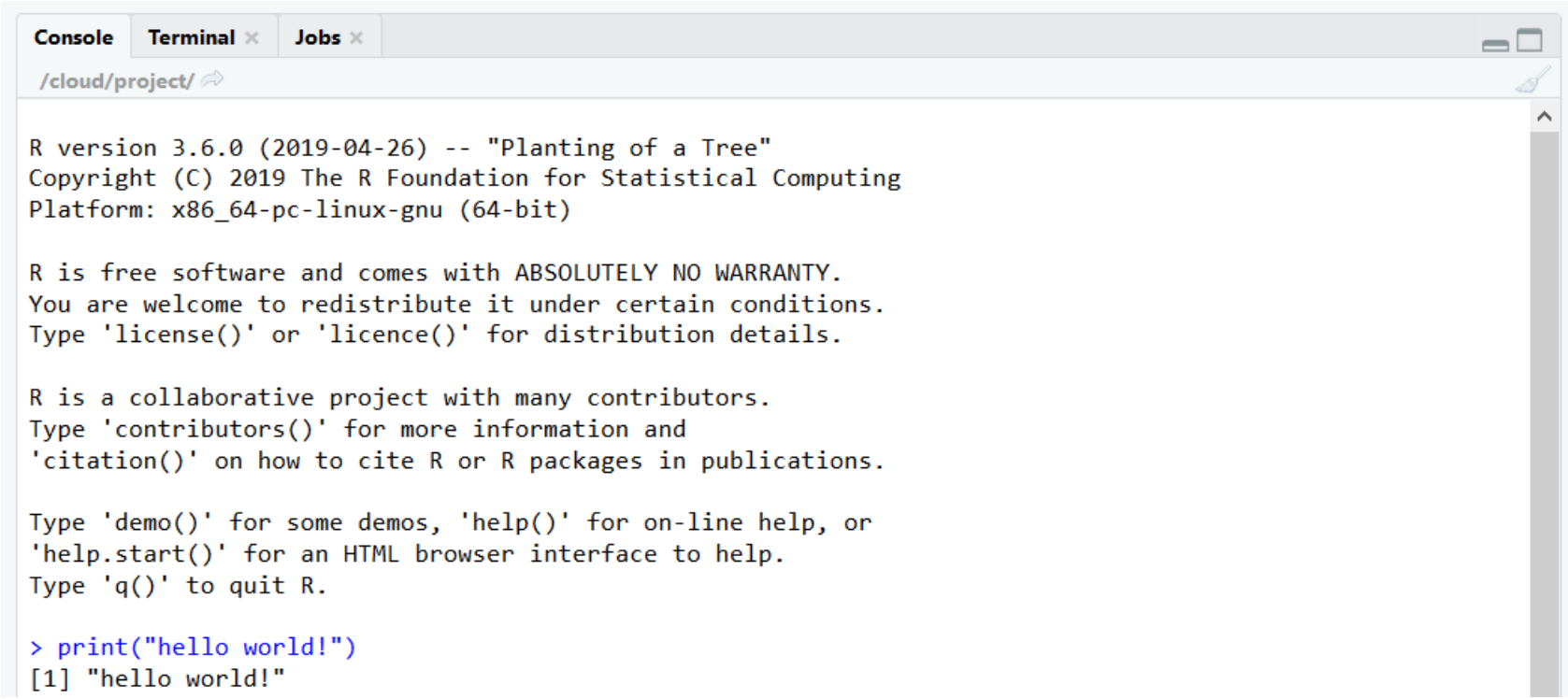
If there is anything R *returns*, it will also be printed here after your call - here starting with [1] .



The Console

R code you are running appears here - the line starts with `>` (by default).

If there is anything R *returns*, it will also be printed here after your call - here starting with `[1]` .



Btw, you can also type code directly in the console and hit `Enter` to evaluate it.

Values

```
1
```

```
"hello world!"
```

```
"2020-03-02 09:00:00"
```

Your Turn!

- Type `2 + 3` and run it.
- Play around with other calculus operators such as `-`, `*`, `/`, `^`, or `sqrt`.

- What is the difference between the first 5 and the `sqrt` calculation?

R is a calculator!

```
2 + 2
## [1] 4

(59 + 73 + 2) * 5
## [1] 670

1 / 200 * 30
## [1] 0.15

sin(pi / 2)
## [1] 1

10^12 * sqrt(4312)
## [1] 6.566582e+13

log(exp(5))
## [1] 5
```

Your Turn!

- Run `x <- 1` and afterwards `x`
- Next, run `x + 2`.
- Now run `x <- 5` and again `x + 2`.

- Why is there no output when you run the first command?
- What is the arrow symbol `<-` doing?
- What is the value of x in the end?
- What if you type `y <- x` and what if `y <- x <- 2`
- And what happens if you type `y <- 2 <- x` ?

Assignments

Values - 1, "Florida", "2010-01-25"

Objects - x <− 22/7

Values - 1, "Florida", "2010-01-25"

Objects - x <− 22/7

Assignments

New objects are created with `<-` and the process is called **assignment**:

```
x <- 24
x
## [1] 24
```

Assignments

New objects are created with `<-` and the process is called **assignment**:

```
x <- 24
x
## [1] 24
```

Assignments

All **R** statements where you create objects have the same form:

```
object_name <- value
```

(When reading that code above say *"object x gets value 24"* in your head.)

Assignments

All R statements where you create objects ~~have the same form:~~

```
object_name <- value
```

You can also create them by using the equal sign:

```
object_name = value
```

This practice is controversially discussed with many people arguing it can cause confusion.
However, the = way is much more common in other programming languages thus some prefer it.

```
y = "hello world"
y
## [1] "hello world"
```

Your Turn!

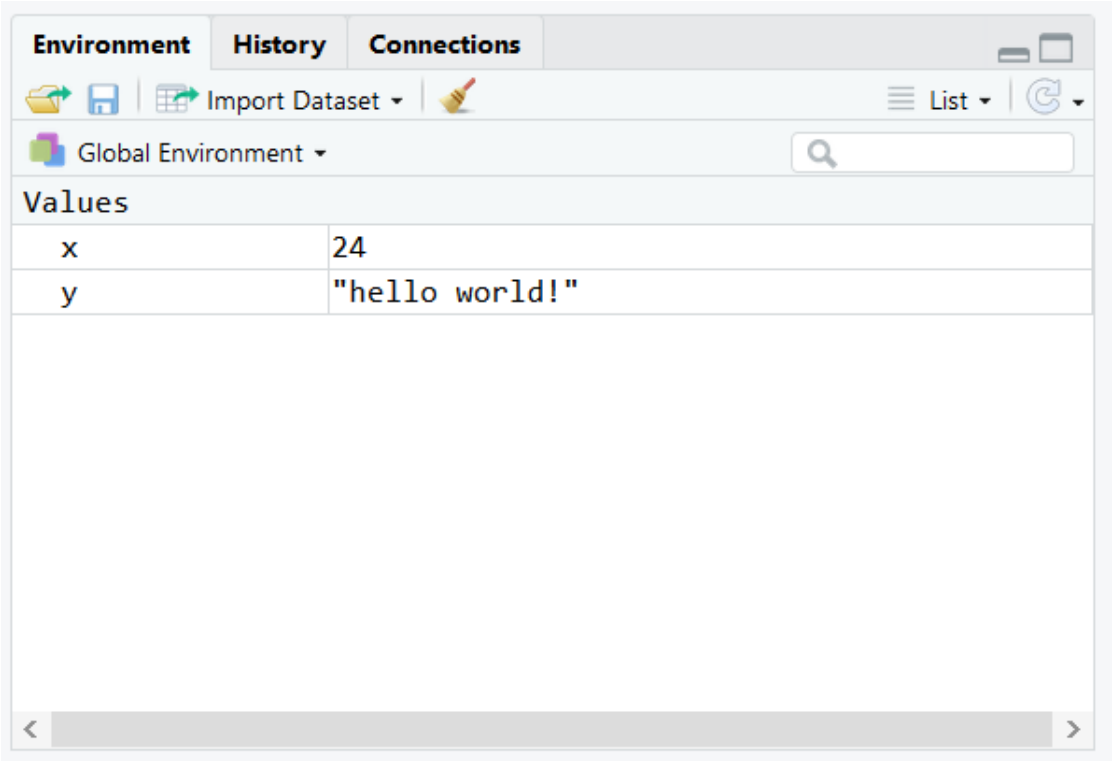
Which of these are numbers?

`1` `"1"` `"one"` `one`

The Environment

All assigned objects belong to the *global environment* called `R_GlobalEnv`*.

You can find an overview of everything you have defined in the *environment pane*.



** There are also other types of environments but we do not bother with that.*

Object Names

Object Names

Object names

- must start with a letter
- can only contain letters, numbers, _ and .
- should be descriptive
- ideally follow a convention for multiple words
- ideally follow a convention for different classes/types

```
i_use_snake_case
otherPeopleUseCamelCase
some.people.use.periods
And_aFew.People_RENOUNCEconvention
```

```
df_us_population <- a
DfUsPopulation <- b
df.us.population <- c
dfUS_population <- d
```

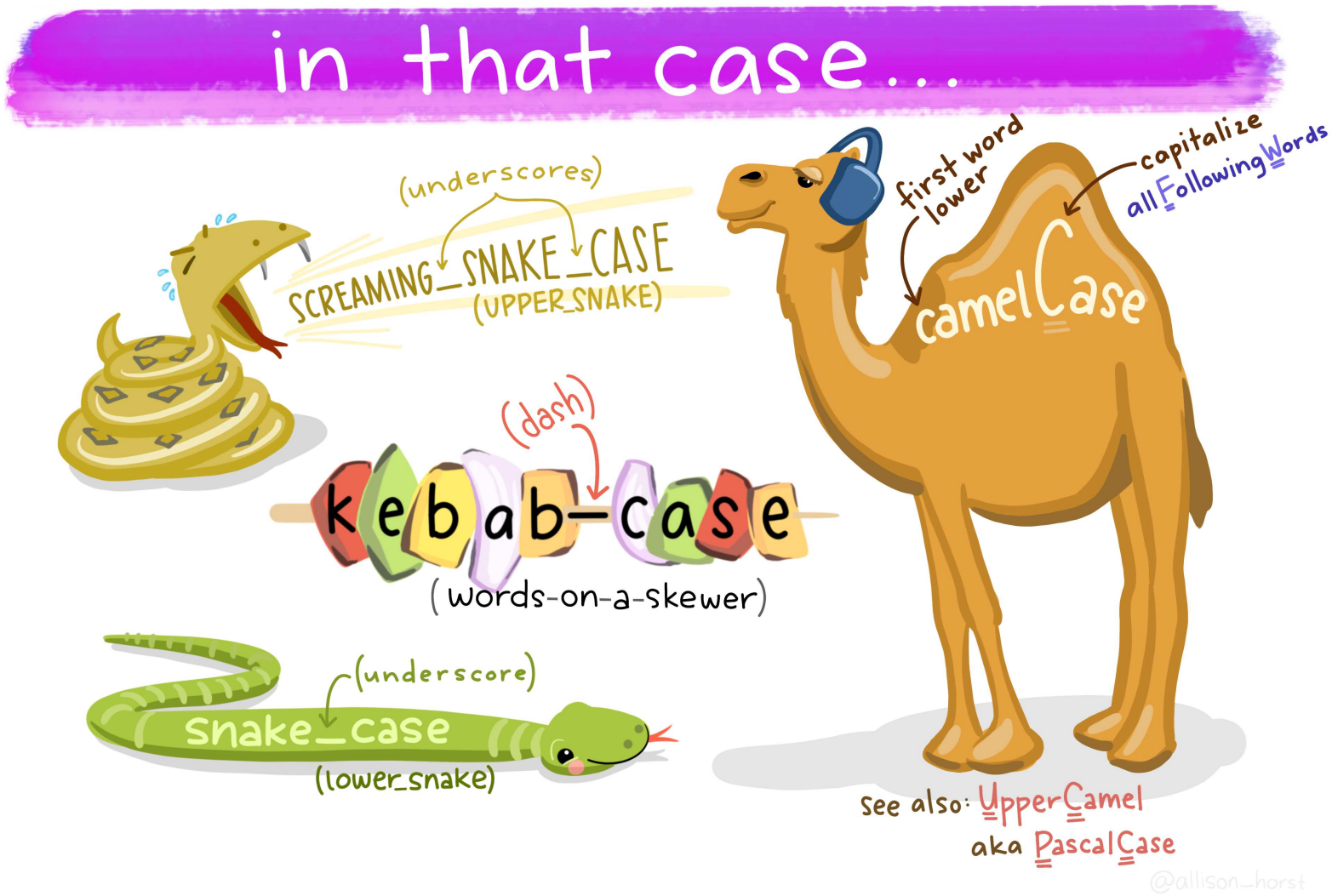


Illustration by Allison Horst (github.com/allisonhorst/stats-illustrations)

Functions

Functions

R has a large collection of built-in functions that are called like this:

```
function_name(arg1 = val1, arg2 = val2, ...)
```

Functions

R has a large collection of built-in functions that are called like this:

```
function_name(arg1 = val1, arg2 = val2, ...)
```

We already have seen some functions in the exercise before: `+`, `-`, and `sqrt` all are functions!

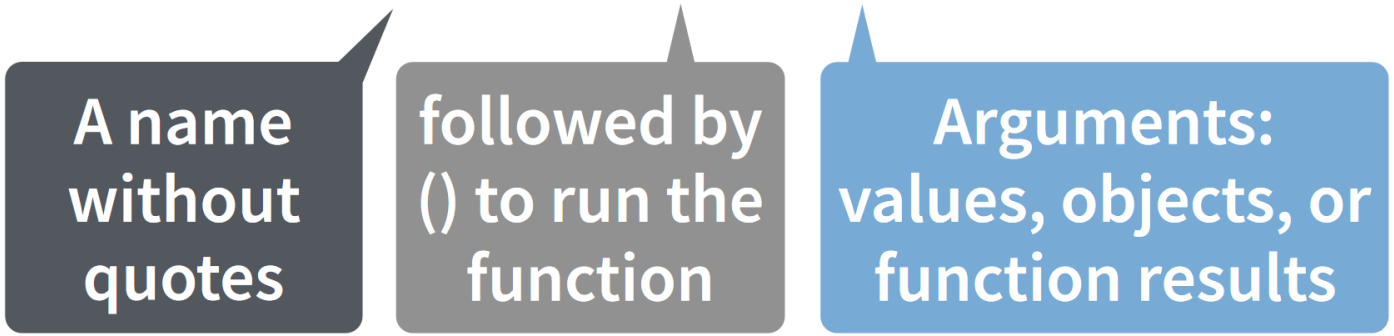
Functions

```
`+`
## function (e1, e2)  .Primitive("+")

`-`
## function (e1, e2)  .Primitive("-")

sqrt
## function (x)  .Primitive("sqrt")

mean
## function (x, ...)
## UseMethod("mean")
## <bytecode: 0x0000000061d3068>
## <environment: namespace:base>
```

Functions

```
citation
## function (package = "base", lib.loc = NULL, auto = NULL)
## {
##     if (!is.null(auto) && !is.logical(auto) && !any(is.na(match(c("Package",
##         "Version", "Title"), names(meta <- as.list(auto))))) &&
##         !all(is.na(match(c("Authors@R", "Author"), names(meta))))) {
##         auto_was_meta <- TRUE
##         package <- meta$Package
##     }
##     else {
##         auto_was_meta <- FALSE
##         dir <- system.file(package = package, lib.loc = lib.loc)
##         if (dir == "")
##             stop(packageNotFoundError(package, lib.loc, sys.call()))
##         meta <- packageDescription(pkg = package, lib.loc = dirname(dir))
##         citfile <- file.path(dir, "CITATION")
##         test <- file_test("-f", citfile)
##         if (!test) {
##             citfile <- file.path(dir, "inst", "CITATION")
##             test <- file_test("-f", citfile)
##         }
##         if (is.null(auto))
```

Functions - round(x, digits = 3)



Your Turn!

Which of these will work supposing `one <- 1`?

```
log(1)   log("1")   log("one")   log(one)
```

Functions

```r
one <- 1

log(1)
## [1] 0

log("1")
## Error in log("1"): non-numeric argument to mathematical function

log("one")
## Error in log("one"): non-numeric argument to mathematical function

log(one)
## [1] 0
```

Functions

```
citation()
##
## 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

```
sqrt(9)
## [1] 3
```

Functions

```
sqrt(9)
## [1] 3
mean(5, 1, 3)
## [1] 5
```

Functions

```
sqrt(9)
## [1] 3
mean(5, 1, 3)
## [1] 5
```

via NBA on GIPHY

Functions

```
?mean()
```

mean {base} R Documentation

Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic 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. 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.
`...`	further arguments passed to or from other methods.

Value

If `trim` is zero (the default), the arithmetic mean of the values in `x` is computed, as a numeric or complex vector of length one. If `x` is not logical (coerced to numeric), numeric (including integer) or complex, `NA_real_` is returned, with a warning.

If `trim` is non-zero, a symmetrically trimmed mean is computed with a fraction of `trim` observations deleted from each end before the mean is computed.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

`weighted.mean`, `mean.POSIXct`, `colMeans` for row and column means.

Examples

```
x <- c(0:10, 50)
xm <- mean(x)
c(xm, mean(x, trim = 0.10))
```

Functions

```
?mean()
```

mean {base} R Documentation

Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic 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. 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.
`...`	further arguments passed to or from other methods.

Value

If `trim` is zero (the default), the arithmetic mean of the values in `x` is computed, as a numeric or complex vector of length one. If `x` is not logical (coerced to numeric), numeric (including integer) or complex, `NA_real_` is returned, with a warning.

If `trim` is non-zero, a symmetrically trimmed mean is computed with a fraction of `trim` observations deleted from each end before the mean is computed.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

`weighted.mean`, `mean.POSIXct`, `colMeans` for row and column means.

Examples

```
x <- c(0:10, 50)
xm <- mean(x)
c(xm, mean(x, trim = 0.10))
```

Data Types

Data Types: Classes

Atomic classes in R:

```
class(1)
## [1] "numeric"

class("a")
## [1] "character"

class(TRUE) ## T is short for 'TRUE', F is short for 'FALSE',
## [1] "logical"

class(1L)
## [1] "integer"

class(1 + 0i)
## [1] "complex"
```

Data Types: Vectors

As in the case of the **mean** function we need a *numerical/logical vector*:

```
v <- c(5, 1)
class(v)
## [1] "numeric"

v2 <- c(TRUE, FALSE)
class(v2)
## [1] "logical"
```

Data Types: Vectors

As in the case of the **mean** function we need a *numerical/logical vector*:

```
v <- c(5, 1)
class(v)
## [1] "numeric"

v2 <- c(TRUE, FALSE)
class(v2)
## [1] "logical"

v3 <- c(1L, 2L)
class(v3)
## [1] "integer"
```

Data Types: Vectors

As in the case of the `mean` function we need a *numerical/logical vector*:

```
v <- c(5, 1)
class(v)
## [1] "numeric"

v2 <- c(TRUE, FALSE)
class(v2)
## [1] "logical"

v3 <- c(1L, 2L)
class(v3)
## [1] "integer"
```

... and there are also *character vectors*:

```
v4 <- c("Physalia", "DataViz")
class(v4)
## [1] "character"
```

Data Types: Vectors

You create *atomic vectors* by using `c(val1, val2, ...)`.
Atomic vectors only contain one data type and all values will be *coerced implicitly* to the same data type:

```
(v5 <- c("ggplot", 2020, TRUE))
## [1] "ggplot" "2020"   "TRUE"
class(v5)
## [1] "character"

(v6 <- c(1L, 0.2, 3 + 0i))
## [1] 1.0+0i 0.2+0i 3.0+0i
class(v6)
## [1] "complex"
```

Data Types: Vectors

You create *atomic vectors* by using `c(val1, val2, ...)`.
One can *explicitly coerce* atomic vectors to a particular data type:

```
(v5 <- c("ggplot", 2020, TRUE))
## [1] "ggplot" "2020"   "TRUE"
class(v5)
## [1] "character"

(v6 <- c(1L,0.2, 3 + 0i))
## [1] 1.0+0i 0.2+0i 3.0+0i
class(v6)
## [1] "complex"

as.numeric(v5)
## [1]   NA 2020   NA

as.character(v6)
## [1] "1+0i"   "0.2+0i" "3+0i"

as.logical(v6)
## [1] TRUE TRUE TRUE
```

Your Turn!

- Fix our calculation of the mean of 5, 1, and 3.
- Guess what the following vectors look like without running them first:
  - `a <- c(1.7, "Physalia")` and `as.numeric(a)`
  - `b <- c(TRUE, 2020, 1.375)` and `as.complex(b)`
  - `c <- c(TRUE, "TRUE", "T", FALSE, "False", "f")` and `as.logical(c)`
  - `as.integer(c(45L, 0.237, 4.9))`
- What does `as.factor(c("red", "blue"))` do?

Function Arguments

Function Arguments

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

Function Arguments

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

## or:
my_vector <- c(5, 1, 3)
mean(x = my_vector)
## [1] 3
```

Function Arguments

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

max(x = 5)
## [1] 5
max(x = c(5, 1, 3))
## [1] 5
```

Function Arguments

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

max(x = 5)
## [1] 5
max(x = c(5, 1, 3))
## [1] 5

quantile(x = c(5, 1, 3), probs = 0.25)
## 25%
##   2
```

Implicit Matching of Arguments

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

max(x = 5)
## [1] 5
max(x = c(5, 1, 3))
## [1] 5

quantile(x = c(5, 1, 3), probs = 0.25)
## 25%
##   2
```

```
mean(c(5, 1, 3))
## [1] 3

max(5)
## [1] 5
max(c(5, 1, 3))
## [1] 5

quantile(c(5, 1, 3), 0.25)
## 25%
##   2
```

Implicit Matching of Arguments

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

max(x = 5)
## [1] 5
max(x = c(5, 1, 3))
## [1] 5

quantile(x = c(5, 1, 3), probs = 0.25)
## 25%
##   2
```

```
mean(c(5, 1, 3))
## [1] 3

max(5)
## [1] 5
max(c(5, 1, 3))
## [1] 5

quantile(c(5, 1, 3), 0.25)
## 25%
##   2
```

BUT:

```
quantile(0.25, c(5, 1, 3))
## Error in quantile.default(0.25, c(5, 1, 3)): 'probs' outside [0,1]
quantile(0.25, c(1, 0.345))
##  100% 34.5%
##  0.25  0.25
```

Coercion

Implicit and Explicit Coercion

```
(a <- c(1.7, "Physalia"))
## [1] "1.7"      "Physalia"

(b <- c(TRUE, 2020, 1.375))
## [1]    1.000 2020.000    1.375

(c <- c(TRUE, "TRUE", "T", FALSE, "False", "f"))
## [1] "TRUE"  "TRUE"  "T"     "FALSE" "False" "f"
```

Implicit and Explicit Coercion

```r
(a <- c(1.7, "Physalia"))
## [1] "1.7"      "Physalia"

(b <- c(TRUE, 2020, 1.375))
## [1]    1.000 2020.000    1.375

(c <- c(TRUE, "TRUE", "T", FALSE, "False", "f"))
## [1] "TRUE"  "TRUE"  "T"     "FALSE" "False" "f"

as.numeric(a)
## [1] 1.7  NA

as.complex(b)
## [1]    1.000+0i 2020.000+0i    1.375+0i

as.logical(c)
## [1]  TRUE  TRUE  TRUE FALSE FALSE    NA

as.integer(c(45L, 0.237, 4.9))
## [1] 45  0  4
```

Implicit Coercion

The coersion rule goes:

logical → **integer** → **numeric** → **complex** → **character**

```
class(c(TRUE, 1L, 1.2, 1 + 0i, "a"))
## [1] "character"

class(c(TRUE, 1L, 1.2, 1 + 0i))
## [1] "complex"

class(c(TRUE, 1L, 1.2))
## [1] "numeric"

class(c(TRUE, 1L))
## [1] "integer"

class(c(TRUE))
## [1] "logical"
```

Factors

Factors

Conceptually, factors are variables which take only a predefined number of values (i.e. categorical variables).

Factors are important for *ordered categorical data*, a thing we will need often when working with **ggplot2**:

```
v4
## [1] "Physalia" "DataViz"
as.factor(v4)
## [1] Physalia DataViz
## Levels: DataViz Physalia
factor(v4)
## [1] Physalia DataViz
## Levels: DataViz Physalia
```

Factors

Conceptually, factors are variables which take only a predefined number of values (i.e. categorical variables).

Factors are important for *ordered categorical data*, a thing we will need often when working with **ggplot2**:

```
v4
## [1] "Physalia" "DataViz"
as.factor(v4)
## [1] Physalia DataViz
## Levels: DataViz Physalia
factor(v4)
## [1] Physalia DataViz
## Levels: DataViz Physalia

factor(v4, levels = c("Physalia", "DataViz"))
## [1] Physalia DataViz
## Levels: Physalia DataViz
```

Factors

Conceptually, factors are variables which take only a predefined number of values (i.e. categorical variables).

Factors are important for *ordered categorical data*, a thing we will need often when working with **ggplot2**:

```
v4
## [1] "Physalia" "DataViz"
as.factor(v4)
## [1] Physalia DataViz
## Levels: DataViz Physalia
factor(v4)
## [1] Physalia DataViz
## Levels: DataViz Physalia

factor(v4, levels = c("Physalia", "DataViz"))
## [1] Physalia DataViz
## Levels: Physalia DataViz

factor(v4, levels = c("Physalia", "DataViz"),
       labels = c("Physalia Courses Berlin", "Data Visualization"))
## [1] Physalia Courses Berlin Data Visualization
## Levels: Physalia Courses Berlin Data Visualization
```

Factors

Conceptually, factors are variables which take only a predefined number of values (i.e. categorical variables).

Factors are important for *ordered categorical data*, a thing we will need often when working with **ggplot2**:

```
f <- factor(v4, levels = c("Physalia", "DataViz"),
            labels = c("Physalia Courses Berlin", "Data Visualization"))

class(f)
## [1] "factor"
levels(f)
## [1] "Physalia Courses Berlin" "Data Visualization"
```

Factors

Conceptually, factors are variables which take only a predefined number of values (i.e. categorical variables).

Factors are important for *ordered categorical data*, a thing we will need often when working with **ggplot2**:

```
f <- factor(v4, levels = c("Physalia", "DataViz"),
            labels = c("Physalia Courses Berlin", "Data Visualization"))

class(f)
## [1] "factor"
levels(f)
## [1] "Physalia Courses Berlin" "Data Visualization"

as.character(f)
## [1] "Physalia Courses Berlin" "Data Visualization"
as.numeric(f)
## [1] 1 2
```

Factors

Conceptually, factors are variables which take only a predefined number of values (i.e. categorical variables).

Factors are important for *ordered categorical data*, a thing we will need often when working with **ggplot2**:

```
factor(c(0, 0, 1, 0, 1), levels = c("1", "0"), labels = c("no", "yes"))
## [1] yes yes no  yes no
## Levels: no yes
```

Factors

Conceptually, factors are variables which take only a predefined number of values (i.e. categorical variables).

Factors are important for *ordered categorical data*, a thing we will need often when working with **ggplot2**:

```
factor(c(0, 0, 1, 0, 1), levels = c("1", "0"), labels = c("no", "yes"))
## [1] yes yes no  yes no
## Levels: no yes

factor(c(0, 0, 1, 0, 1, 2), levels = c("1", "0"), labels = c("no", "yes"))
## [1] yes  yes  no   yes  no   <NA>
## Levels: no yes
```

Missing Data

Missing Values

Missing values are denoted by **NA** *(not available)*,
undefined mathematical operations by **NaN** *(not a number)* in **R** .

```
(my_experiment <- c(0.61, 0.43, NA, 0.96))
## [1] 0.61 0.43   NA 0.96
is.na(my_experiment)
## [1] FALSE FALSE  TRUE FALSE
```

Missing Values

Missing values are denoted by **NA** *(not available)*,
undefined mathematical operations by **NaN** *(not a number)* in **R** .

```
(my_experiment <- c(0.61, 0.43, NA, 0.96))
## [1] 0.61 0.43   NA 0.96
is.na(my_experiment)
## [1] FALSE FALSE  TRUE FALSE

0 / 0
## [1] NaN
is.nan(0 / 0)
## [1] TRUE
is.na(0 / 0)
## [1] TRUE
```

Missing Values

Missing values are denoted by **NA** *(not available)*,
undefined mathematical operations by **NaN** *(not a number)* in **R** .

```
(my_experiment <- c(0.61, 0.43, NA, 0.96))
## [1] 0.61 0.43   NA 0.96
is.na(my_experiment)
## [1] FALSE FALSE  TRUE FALSE

0 / 0
## [1] NaN
is.nan(0 / 0)
## [1] TRUE
is.na(0 / 0)
## [1] TRUE

is.nan(my_experiment)
## [1] FALSE FALSE FALSE FALSE
```

Missing Values

Since **NA** values are unknown, most comparisons and calculations will return **NA** as well if there is any missing data:

```
max(my_experiment)
## [1] NA
mean(my_experiment)
## [1] NA
```

Missing Values

Since `NA` values are unknown, most comparisons and calculations will return `NA` as well if there is any missing data:

```
max(my_experiment)
## [1] NA
mean(my_experiment)
## [1] NA

max(my_experiment, na.rm = TRUE)
## [1] 0.96
mean(my_experiment, na.rm = TRUE)
## [1] 0.6666667
```

Missing Values

Since `NA` values are unknown, most comparisons and calculations will return `NA` as well if there is any missing data:

```
max(my_experiment)
## [1] NA
mean(my_experiment)
## [1] NA

max(my_experiment, na.rm = TRUE)
## [1] 0.96
mean(my_experiment, na.rm = TRUE)
## [1] 0.6666667

NA == 1
## [1] NA
NA == NA
## [1] NA
NA == NaN
## [1] NA
```

Tabular Data

Data Frames

You can store several vectors (with different data types) as tabular data in so-called *data frames*:

```
(my_df <- data.frame(v, v2, v3, v4))
##   v    v2 v3       v4
## 1 5  TRUE  1 Physalia
## 2 1 FALSE  2  DataViz
```

Data Frames

You can store several vectors (with different data types) as tabular data in so-called *data frames*:

```
(my_df <- data.frame(v, v2, v3, v4))
##   v    v2 v3       v4
## 1 5  TRUE  1 Physalia
## 2 1 FALSE  2  DataViz

## explicitly declaring column names
(my_df <- data.frame(numeric = v, logical = v2, integer = v3, character = v4))
##   numeric logical integer character
## 1       5    TRUE       1  Physalia
## 2       1   FALSE       2   DataViz
```

The New Data Frame: Meet the Tibble!

> Tibbles are a modern reimagining of the data.frame, keeping what time has proven to be effective, and throwing out what is not.

```
(my_tbl <- tibble::tibble(numeric = v, logical = v2, integer = v3, character = v4))
## # A tibble: 2 x 4
##   numeric logical integer character
##     <dbl> <lgl>     <int> <chr>
## 1       5 TRUE          1 Physalia
## 2       1 FALSE         2 DataViz
```

*The name comes from the way you originally created these objects: **`tbl_df()`**, which was most easily pronounced as "tibble diff" or "tibble d. f.".*

The New Data Frame: Meet the Tibble!

Note:

Similarly as the functions we have used before, `tibble()` is a function.

While `mean()` and `+` are *base functions*, `tibble()` belongs to an *add-on package* which is named *tibble* as well.

To use use functions from installed packages, you either call them as I did before or by loading the package into your session via another function called `library()`:

```
## install the add-on package from the CRAN server
install.packages("tibble")

## without loading the package but using "namespace"
my_tbl <- tibble::tibble(numeric = v, logical = v2, integer = v3, character = v4)

## loading the package
library(tibble)
## all functions contained in the library are available now
my_tbl <- tibble(numeric = v, logical = v2, integer = v3, character = v4)
```

Data Frame vs Tibble

> Tibbles are a modern reimagining of the data.frame, keeping what time has proven to be effective, and throwing out what is not.

Main differences in the usage of a data frame vs a tibble:

- tibbles have a refined print method
  - show only the first 10 rows, and all the columns that fit on screen
  - in addition to its name, each column reports its type
  - also, the type of the data and its dimensions are shown

Data Frame vs Tibble

```
tbl_flights
## # A tibble: 100 x 13
##     year month   day sched_dep_time
##    <int> <int> <int>          <int>
##  1  2013     1     1            515
##  2  2013     1     1            529
##  3  2013     1     1            540
##  4  2013     1     1            545
##  5  2013     1     1            600
##  6  2013     1     1            558
##  7  2013     1     1            600
##  8  2013     1     1            600
##  9  2013     1     1            600
## 10  2013     1     1            600
## # ... with 90 more rows, and 9 more
## #   variables: dep_delay <dbl>,
## #   sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>,
## #   flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>,
## #   time_hour <dttm>
```

```
as.data.frame(tbl_flights)
##     year month day sched_dep_time
## 1   2013     1   1            515
## 2   2013     1   1            529
## 3   2013     1   1            540
## 4   2013     1   1            545
## 5   2013     1   1            600
## 6   2013     1   1            558
## 7   2013     1   1            600
## 8   2013     1   1            600
## 9   2013     1   1            600
## 10  2013     1   1            600
## 11  2013     1   1            600
## 12  2013     1   1            600
## 13  2013     1   1            600
## 14  2013     1   1            600
## 15  2013     1   1            600
## 16  2013     1   1            559
## 17  2013     1   1            600
## 18  2013     1   1            600
## 19  2013     1   1            600
## 20  2013     1   1            600
```

Data Frame vs Tibble

> Tibbles are a modern reimagining of the data.frame, keeping what time has proven to be effective, and throwing out what is not.

Main differences in the usage of a data frame vs a tibble:

- tibbles have a refined print method
  - show only the first 10 rows, and all the columns that fit on screen
  - in addition to its name, each column reports its type
  - also, the type of the data and its dimensions are shown
- tibbles are strict about subsetting

Data Frame vs Tibble

Tibbles are strict about subsetting:

```
my_df$integer
## [1] 1 2

my_tbl$integer
## [1] 1 2
```

Data Frame vs Tibble

Tibbles are strict about subsetting:

```
my_df$integer
## [1] 1 2

my_tbl$integer
## [1] 1 2

my_df$int
## [1] 1 2

my_tbl$int
## NULL
```

Data Import

Data Import

Usually, you create data frames by loading data files such as `.txt`, `.csv`, ...

```
## data.frames (base R)
my_data <- read.delim("./data/cool_data.txt", sep = "\t")
my_data <- read.csv("./data/cool_data.csv")
```

Data Import

Usually, you create data frames by loading data files such as `.txt`, `.csv`, ...

```
## data.frames (base R)
my_data <- read.delim("./data/cool_data.txt", sep = "\t")
my_data <- read.csv("./data/cool_data.csv")

## tibbles (tidyverse)
my_data <- readr::read_delim("./data/cool_data.txt", quote = "\t")
my_data <- readr::read_csv("./data/cool_data.csv")
```

Data Import

Usually, you create data frames by loading data files such as `.txt`, `.csv`, ...

```
## data.frames (base R)
my_data <- read.delim("./data/cool_data.txt", sep = "\t")
my_data <- read.csv("./data/cool_data.csv")

## tibbles (tidyverse)
my_data <- readr::read_delim("./data/cool_data.txt", quote = "\t")
my_data <- readr::read_csv("./data/cool_data.csv")
my_data <- readr::read_csv("https://some_webpage.com/online_data.csv")
my_data <- readr::read_csv2("./data/actually_not_a_csv_but_okay.csv") ## ';' as sep
```

Data Import

Usually, you create data frames by loading data files such as `.txt`, `.csv`, `.xlsx`, ...

```
## data.frames (base R)
my_data <- read.delim("./data/cool_data.txt", sep = "\t")
my_data <- read.csv("./data/cool_data.csv")

## tibbles (tidyverse)
my_data <- readr::read_delim("./data/cool_data.txt", quote = "\t")
my_data <- readr::read_csv("./data/cool_data.csv")
my_data <- readr::read_csv("https://some_webpage.com/online_data.csv")
my_data <- readr::read_csv2("./data/actually_not_a_csv_but_okay.csv") ## ';' as sep
my_data <- readxl::read_xls("./data/oh_no_excel_data.xls", sheet = 1)
my_data <- readxl::read_xlsx("./data/oh_no_excel_data.xlsx", sheet = 1)
```

Data Import

Usually, you create data frames by loading data files such as `.txt`, `.csv`, `.xlsx`, `.rds`, ...

```
## data.frames (base R)
my_data <- read.delim("./data/cool_data.txt", sep = "\t")
my_data <- read.csv("./data/cool_data.csv")

## tibbles (tidyverse)
my_data <- readr::read_delim("./data/cool_data.txt", quote = "\t")
my_data <- readr::read_csv("./data/cool_data.csv")
my_data <- readr::read_csv("https://some_webpage.com/online_data.csv")
my_data <- readr::read_csv2("./data/actually_not_a_csv_but_okay.csv") ## ';' as sep
my_data <- readxl::read_xls("./data/oh_no_excel_data.xls", sheet = 1)
my_data <- readxl::read_xlsx("./data/oh_no_excel_data.xlsx", sheet = 1)

## one of R's data formats (base R)
my_data <- readRDS("./data/one_of_Rs_data_formats.Rds")
```

Your Turn!

- Download or create a data set with the software of your choice and import it into R.
- Inspect the imported data using `summary()`, `str()`, `tibble::glimpse()`, and `View()`.
- Also explore other functions such as `dim()`, `length()`, `unique()`, and `range()`.
- Save your data set as .Rds (use `saveRDS()` and `?saveRDS()` to get help).

Data Summaries

Data Summaries

```
summary(tbl_flights[, c(7, 8, 13)])
##    arr_delay          carrier            time_hour
##  Min.   :-33.00   Length:100         Min.   :2013-01-01 05:00:00
##  1st Qu.:-10.00   Class :character   1st Qu.:2013-01-01 06:00:00
##  Median : -2.50   Mode  :character   Median :2013-01-01 06:00:00
##  Mean   :  1.03                      Mean   :2013-01-01 06:22:48
##  3rd Qu.: 10.25                      3rd Qu.:2013-01-01 07:00:00
##  Max.   : 49.00                      Max.   :2013-01-01 07:00:00
```

Data Summaries

```
str(tbl_flights)
## Classes 'tbl_df', 'tbl' and 'data.frame':    100 obs. of  13 variables:
##  $ year          : int  2013 2013 2013 2013 2013 2013 2013 2013 2013 2013 ...
##  $ month         : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ day           : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ sched_dep_time: int  515 529 540 545 600 558 600 600 600 600 ...
##  $ dep_delay     : num  2 4 2 -1 -6 -4 -5 -3 -3 -2 ...
##  $ sched_arr_time: int  819 830 850 1022 837 728 854 723 846 745 ...
##  $ arr_delay     : num  11 20 33 -18 -25 12 19 -14 -8 8 ...
##  $ carrier       : chr  "UA" "UA" "AA" "B6" ...
##  $ flight        : int  1545 1714 1141 725 461 1696 507 5708 79 301 ...
##  $ tailnum       : chr  "N14228" "N24211" "N619AA" "N804JB" ...
##  $ origin        : chr  "EWR" "LGA" "JFK" "JFK" ...
##  $ dest          : chr  "IAH" "IAH" "MIA" "BQN" ...
##  $ time_hour     : POSIXct, format: "2013-01-01 05:00:00" "2013-01-01 05:00:00" ...
```

Data Summaries

```
str(tbl_flights)
## Classes 'tbl_df', 'tbl' and 'data.fram
##  $ year          : int  2013 2013 2013
##  $ month         : int  1 1 1 1 1 1 1
##  $ day           : int  1 1 1 1 1 1 1
##  $ sched_dep_time: int  515 529 540 54
##  $ dep_delay     : num  2 4 2 -1 -6 -4
##  $ sched_arr_time: int  819 830 850 10
##  $ arr_delay     : num  11 20 33 -18 -
##  $ carrier       : chr  "UA" "UA" "AA"
##  $ flight        : int  1545 1714 1141
##  $ tailnum       : chr  "N14228" "N242
##  $ origin        : chr  "EWR" "LGA" "J
##  $ dest          : chr  "IAH" "IAH" "M
##  $ time_hour     : POSIXct, format:  .
```

```
tibble::glimpse(tbl_flights)
## Observations: 100
## Variables: 13
## $ year           <int> 2013, 2013, ...
## $ month          <int> 1, 1, 1, 1, ...
## $ day            <int> 1, 1, 1, 1, ...
## $ sched_dep_time <int> 515, 529, 54...
## $ dep_delay      <dbl> 2, 4, 2, -1,...
## $ sched_arr_time <int> 819, 830, 85...
## $ arr_delay      <dbl> 11, 20, 33, ...
## $ carrier        <chr> "UA", "UA", ...
## $ flight         <int> 1545, 1714, ...
## $ tailnum        <chr> "N14228", "N...
## $ origin         <chr> "EWR", "LGA"...
## $ dest           <chr> "IAH", "IAH"...
## $ time_hour      <dttm> 2013-01-01 ...
```

Data Summaries

```
dim(tbl_flights)
## [1] 100  13
length(tbl_flights)
## [1] 13
```

Data Summaries

```
dim(tbl_flights)
## [1] 100  13
length(tbl_flights)
## [1] 13

length(tbl_flights$origin)
## [1] 100
unique(tbl_flights$origin)
## [1] "EWR" "LGA" "JFK"
```

Data Summaries

```
dim(tbl_flights)
## [1] 100  13
length(tbl_flights)
## [1] 13

length(tbl_flights$origin)
## [1] 100
unique(tbl_flights$origin)
## [1] "EWR" "LGA" "JFK"

range(tbl_flights$dep_delay, na.rm = T)
## [1] -9 47
quantile(tbl_flights$dep_delay, na.rm = T)
##   0%  25%  50%  75% 100%
##   -9   -4   -2    0   47
var(tbl_flights$dep_delay, na.rm = T)
## [1] 63.41121
sd(tbl_flights$dep_delay, na.rm = T)
## [1] 7.963116
```

Generate Data in R

Generate Data Sets in R

You can combine vectors to a dataframe or, better, a tibble:

```
name <- c("Brian", "Jason", "Tyler", "Sam")
age <- seq(23, 26, by = 1)  ## or: 23:26 or c(23, 24, 25, 26)
sex <- rep("male", length(name))

my_tbl <- tibble::tibble(name, age, sex)
```

Generate Data Sets in R

From scratch, you would create these columns within the `tibble()` call:

```
my_tbl <- tibble::tibble(name, age, sex)

(my_tbl <- tibble::tibble(
  name = c("Brian", "Jason", "Tyler", "Sam"),
  age = seq(23, 26, by = 1),  ## or: 23:26 or c(23, 24, 25, 26)
  sex = rep("male", length(name))
))
## # A tibble: 4 x 3
##   name    age sex
##   <chr> <dbl> <chr>
## 1 Brian    23 male
## 2 Jason    24 male
## 3 Tyler    25 male
## 4 Sam      26 male
```

Generate Data Sets in R

You can also create tibbles using an easier to read row-by-row layout - a **tribble**:

```
my_tbl <- tibble::tibble(name, age, sex)

(my_tbl <- tibble::tribble(
  ~name,    ~age,  ~sex,
  "Brian",  23,    "male",
  "Jason",  24,    "male",
  "Tyler",  25,    "male",
  "Sam",    26,    "male"
))
## # A tibble: 4 x 3
##   name    age sex
##   <chr> <dbl> <chr>
## 1 Brian    23 male
## 2 Jason    24 male
## 3 Tyler    25 male
## 4 Sam      26 male
```

The `tidyverse`

Illustration by Allison Horst (github.com/allisonhorst/stats-illustrations)

Why *tidyverse*?

Tidy data is a standard way of mapping the meaning of a dataset to its structure.

→ **Each variable forms a column.**

→ **Each observation forms a row.**

→ **Each type of observational unit forms a table.**

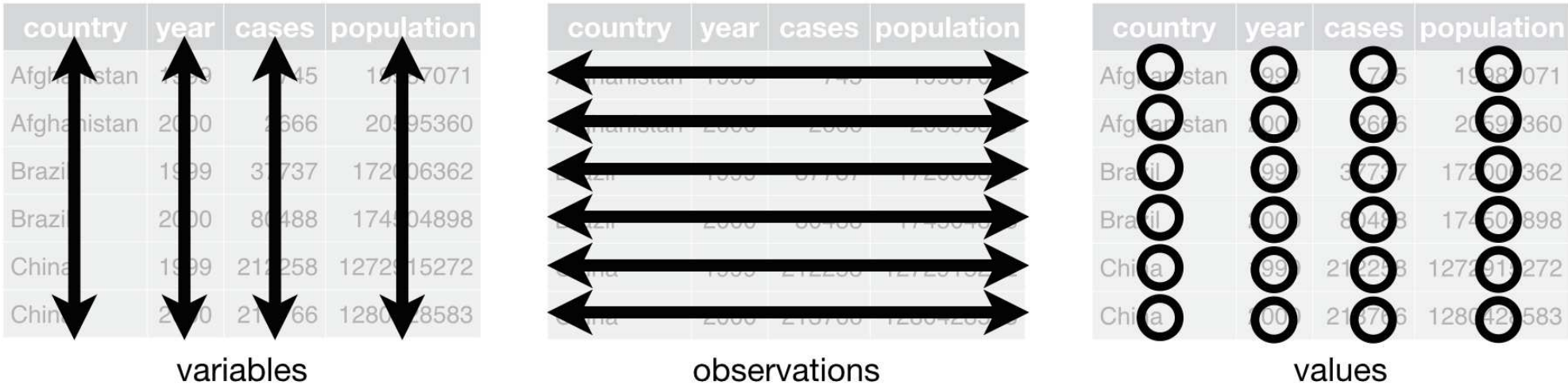
Messy data is any other arrangement of the data.

Tidy Data

→ Each variable forms a column.

→ Each observation forms a row.

→ Each type of observational unit forms a table.



Source: Hadley Wickham's "R for Data Science" (R4DS)

Messy Data

Real datasets often violate the three precepts of tidy data in almost every way imaginable:

- Column headers are values, not variable names.
- Multiple variables are stored in one column.
- Variables are stored in both rows and columns.
- Multiple types of observational units are stored in the same table.
- A single observational unit is stored in multiple tables.

Why Should I Care About *Tidy* Data?

- makes it easy for an analyst or a computer to extract needed variables
- ensures that values of different variables from the same observation are always paired
- a good ordering makes it easier to scan the raw values

Your Turn!

```
#> # A tibble: 12 x 4
#>   country      year type           count
#>   <chr>       <int> <chr>          <int>
#> 1 Afghanistan  1999 cases            745
#> 2 Afghanistan  1999 population  19987071
#> 3 Afghanistan  2000 cases           2666
#> 4 Afghanistan  2000 population  20595360
#> 5 Brazil       1999 cases           37737
#> 6 Brazil       1999 population 172006362
#> # … with 6 more rows
```

Your Turn!

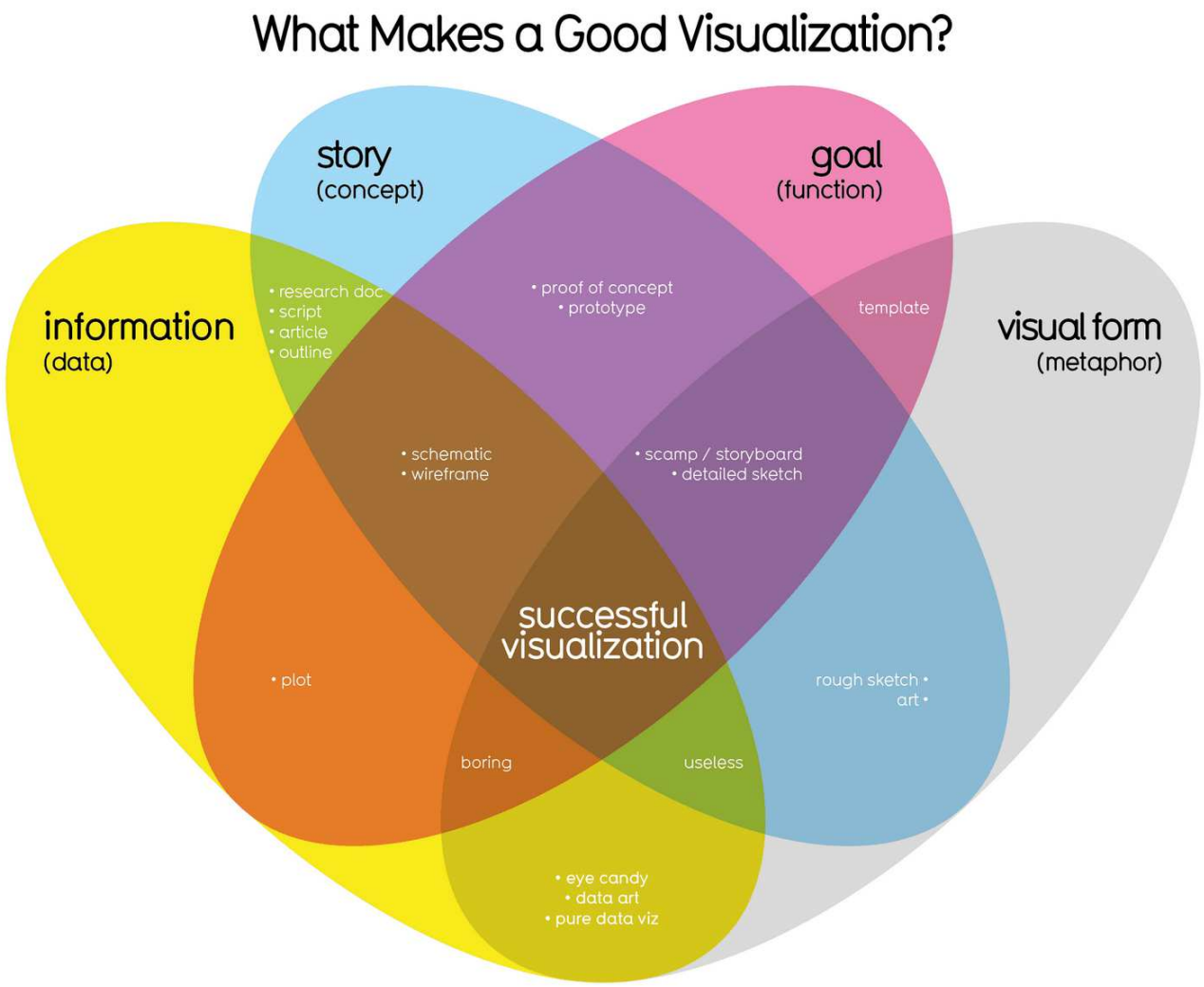
```
#> # A tibble: 6 x 3
#>   country      year rate
#> * <chr>       <int> <chr>
#> 1 Afghanistan  1999 745/19987071
#> 2 Afghanistan  2000 2666/20595360
#> 3 Brazil       1999 37737/172006362
#> 4 Brazil       2000 80488/174504898
#> 5 China        1999 212258/1272915272
#> 6 China        2000 213766/1280428583
```

Your Turn!

```
#> # A tibble: 6 x 4
#>   country      year  cases population
#>   <chr>       <int>  <int>      <int>
#> 1 Afghanistan  1999    745   19987071
#> 2 Afghanistan  2000   2666   20595360
#> 3 Brazil       1999  37737  172006362
#> 4 Brazil       2000  80488  174504898
#> 5 China        1999 212258 1272915272
#> 6 China        2000 213766 1280428583
```

Your Turn!

```
#> # A tibble: 3 x 3
#>   no_cases
#>   country     `1999` `2000`
#> * <chr>        <int>  <int>
#> 1 Afghanistan    745   2666
#> 2 Brazil       37737  80488
#> 3 China       212258 213766

#> # A tibble: 3 x 3
#>   population
#>   country          `1999`     `2000`
#> * <chr>             <int>      <int>
#> 1 Afghanistan   19987071   20595360
#> 2 Brazil       172006362  174504898
#> 3 China       1272915272 1280428583
```



Visualization by David McCandless (Information is Beautiful)