Open Science Starter Pack

Filippo Gambarota

@Winter School 4Ms

February 17, 2025

Materials

The materials will be available on the WS website/repository. But you can access the slides and the materials at:

github.com/stat-teaching/open-science-starter-pack

About me…

- I am a **post-doctoral researcher in Psychometrics** and a Clinical Psychologist at the Department of Developmental Psychology and Socialization, University of Padova
- My research interests are **meta-analysis**, **statistical methods for replicability**, **Monte Carlo simulations for power analysis** and **R programming**.
- I did a PhD in Experimental Psychology about neural correlates of unconscious processing

If you want to know more about my work check my website

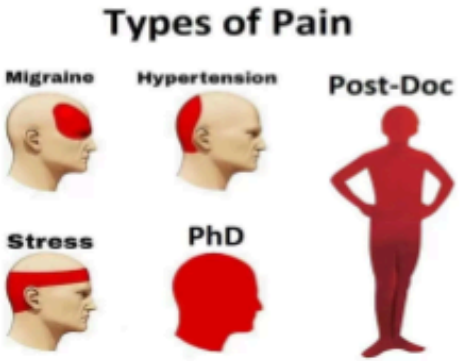
filippogambarota.github.io

> Doing research is hard…

Doing research is hard…

- you have to **read** papers, textbooks, slides and track information
- you have to **plan** your experiment or research
- you have to **collect, organize and manage your data**
- you have to **analyze data**, create **figures** and **tables**
- you have to **write** reports, papers, slides, etc.
- you have to **keep track of reviews** from reviewers, co-authors, supervisor, etc.

Doing research is hard…



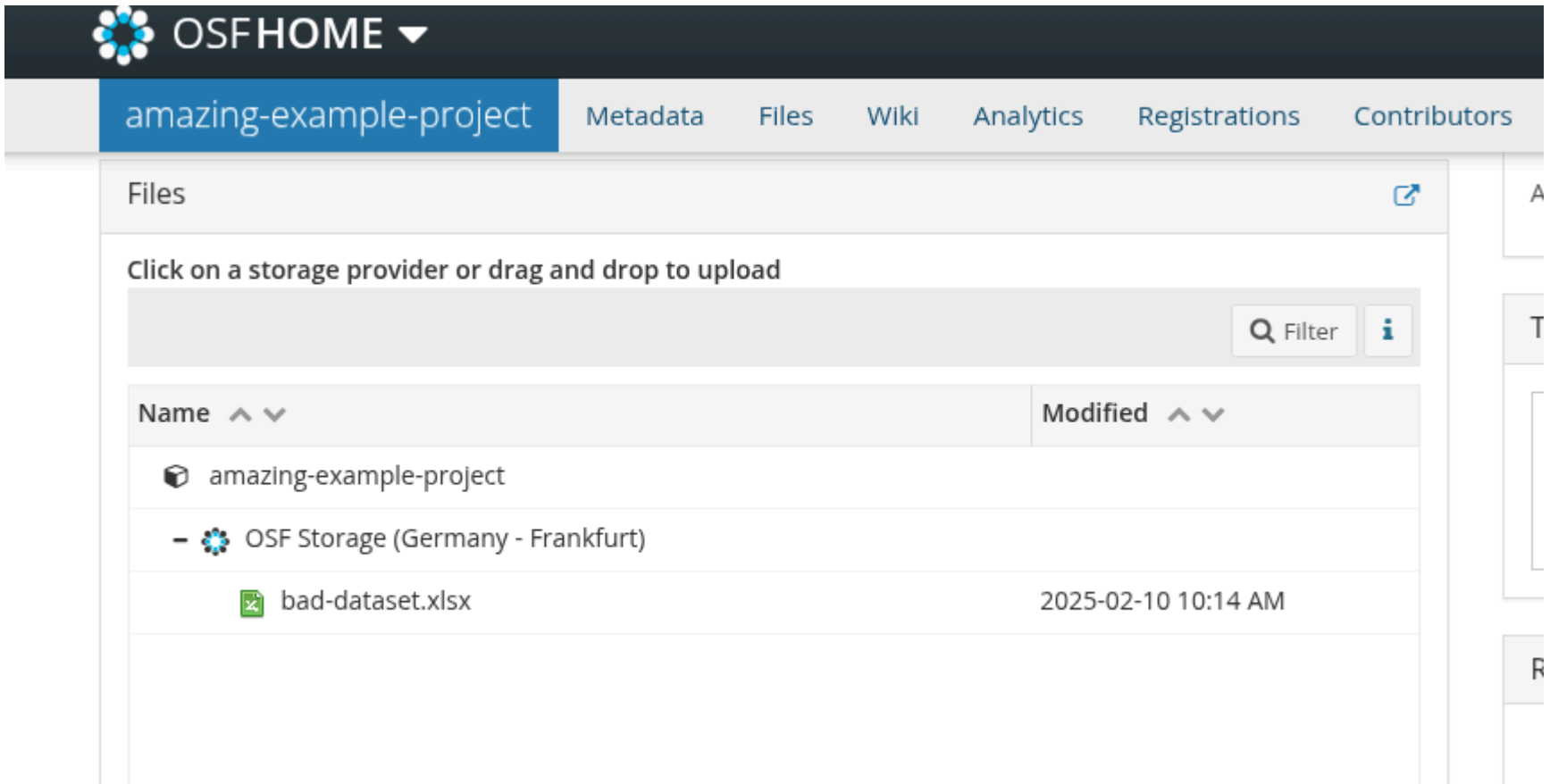
Doing reproducible research is even harder 😱

- organize and share data in a **comprehensive format**
- choose a **future-proof place** to share data
- analyze data using reproducible tools i.e., **scripting**
- create research reports in multiple formats: **slides**, **reports**, **papers**

> Examples of not optimal data sharing

Sharing the dataset

You find a nice paper with an interesting dataset. There is an Open Science Framework link. Awesome! Let's open it:



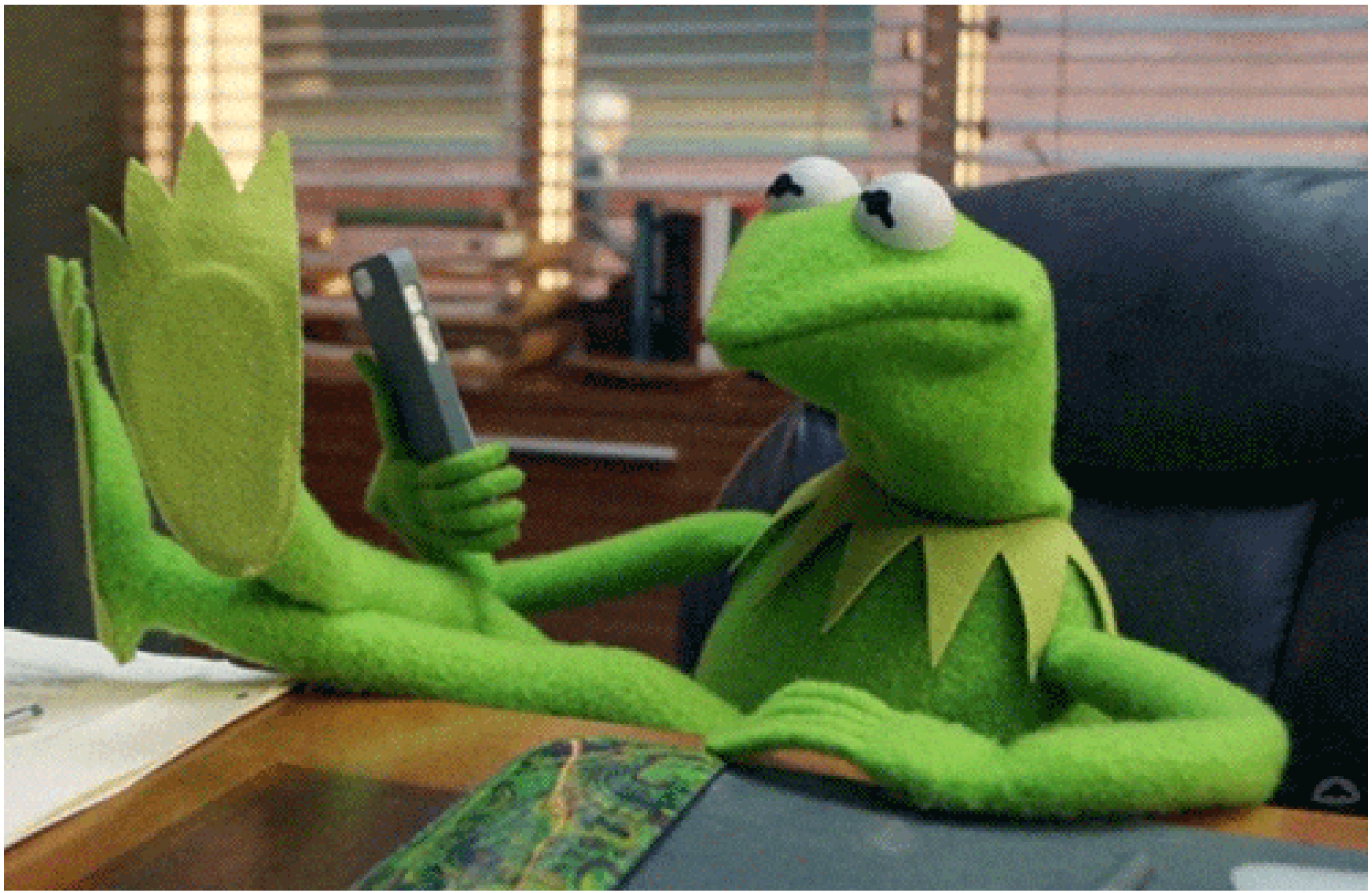
Sharing the dataset

Amazing! there is a single file on the OSF repository. Then you open the dataset:

x1	x2	x3	x4	x5	x6	x7
0.398	13.912	a	0	-0.678	0.876	-0.205
-0.143	1.094	c	0	0.706	0.252	1.882
-0.253	4.898	c	0	0.474	-0.563	0.325
-1.227	14.717	b	0	-0.513	-1.137	-0.136
...	...	...	...	...	...	...
0.937	5.2	b	1	2.087	0.16	-0.172
-0.019	0.582	b	0	-2.164	-0.519	-1.911
0.58	14.435	a	1	-1.641	-1	-0.229
2.011	13.175	a	0	-0.866	1.761	-0.696

Sharing the dataset

Where is the data-dictionary? What are 0 and 1? How missing values are coded?



Sharing is important, but do it appropriately!

- Putting a dataset on OSF is not doing reproducible research. The dataset need to be usable
- Create a data dictionary with variables description and important details
- Add a README file with important information
- Prefer a plain-text format e.g., `csv`, `txt`, etc.

datadictionary

The `datadictionary` package can be used to create a data dictionary starting from a dataframe.

```
library(datadictionary)

# You can also specify labels with a named vector
iris.labels <- c(Sepal.Length = "Sepal length in mm",
                 Sepal.Width = "Sepal width in mm",
                 Petal.Length = "Petal length in mm",
                 Petal.Width = "Petal width in mm",
                 Species = "Species of iris")
create_dictionary(iris, var_labels = iris.labels)
```

datadictionary

Then you can visualize, put in into a document or save as a separated file.

```
#>            item                label   class             summary value
#> 1                                                  Rows in dataset   150
#> 2                                               Columns in dataset     5
#> 3  Sepal.Length Sepal length in mm numeric               mean     6
#> 4                                                      median     6
#> 5                                                         min   4.3
#> 6                                                         max   7.9
#> 7                                                     missing     0
#> 8   Sepal.Width  Sepal width in mm numeric               mean     3
#> 9                                                      median     3
#> 10                                                        min     2
#> 11                                                        max   4.4
#> 12                                                    missing     0
#> 13 Petal.Length Petal length in mm numeric               mean     4
```

> Reproducibility starter pack

Reproducibility starter pack 👷

- A general purpose (or flexible enough) programming language such as R or 🐍
- A literate programming framework to integrate code and text
- A version control system to track projects
- An online repository for future-proof sharing

Disclaimers

The best tool is the tool that does the job.

- But there are some features that makes a tool better in terms of reproducibility, reducing the probability of errors and improve your coding skills.
- There is nothing bad about using SPSS, Jasp or Jamovi. The real problem is that using a point-and-click software reduce the reproducibility. If you can use the scripting part, whatever the tool.
- A general suggestion is to invest some of your time learning/improving a programming language for data pre-processing, analysis and reporting (tables, figures, etc.)

> R Programming Language

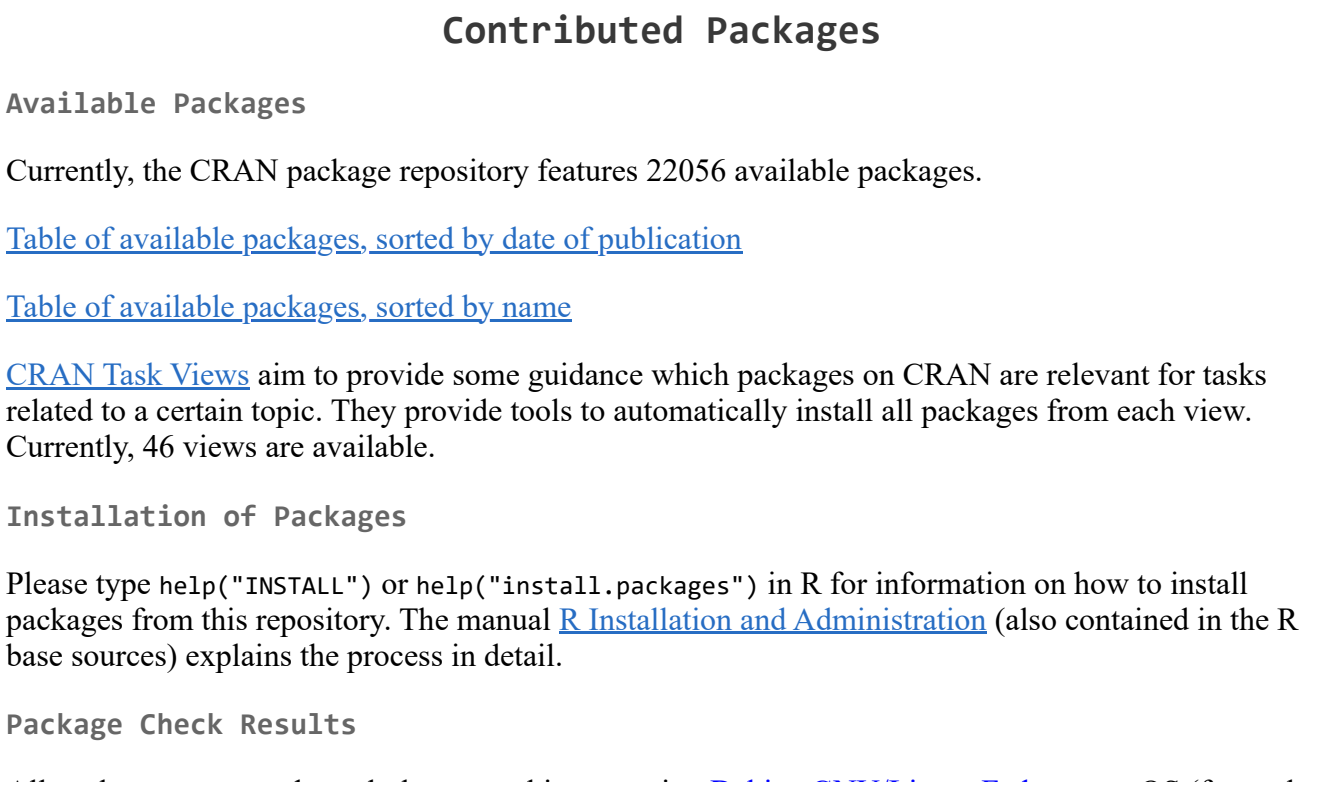
R

> R is a free software environment for statistical computing and graphics.

- (TBH) It is not a proper general purpose programming language (such as C++ or Python).
- R *packages* allow to do almost everything (file manager, image processing, webscraping, sending emails, coffee 😄, etc.)
- It is free and open-source
- The community is wide, active thus solving problems is very easy
- Force you to learn scripting but the are R-based GUI software (e.g., JAMOVI)

R - CRAN

The CRAN is the repository where package developers upload their packages and other users can install them.



Contributed Packages

Available Packages

Currently, the CRAN package repository features 22056 available packages.

Table of available packages, sorted by date of publication

Table of available packages, sorted by name

CRAN Task Views aim to provide some guidance which packages on CRAN are relevant for tasks related to a certain topic. They provide tools to automatically install all packages from each view. Currently, 46 views are available.

Installation of Packages

Please type `help("INSTALL")` or `help("install.packages")` in R for information on how to install packages from this repository. The manual R Installation and Administration (also contained in the R base sources) explains the process in detail.

Package Check Results

> As the saying goes: if something exist, there is an R package for doing it! 😄

R - PYPL Index

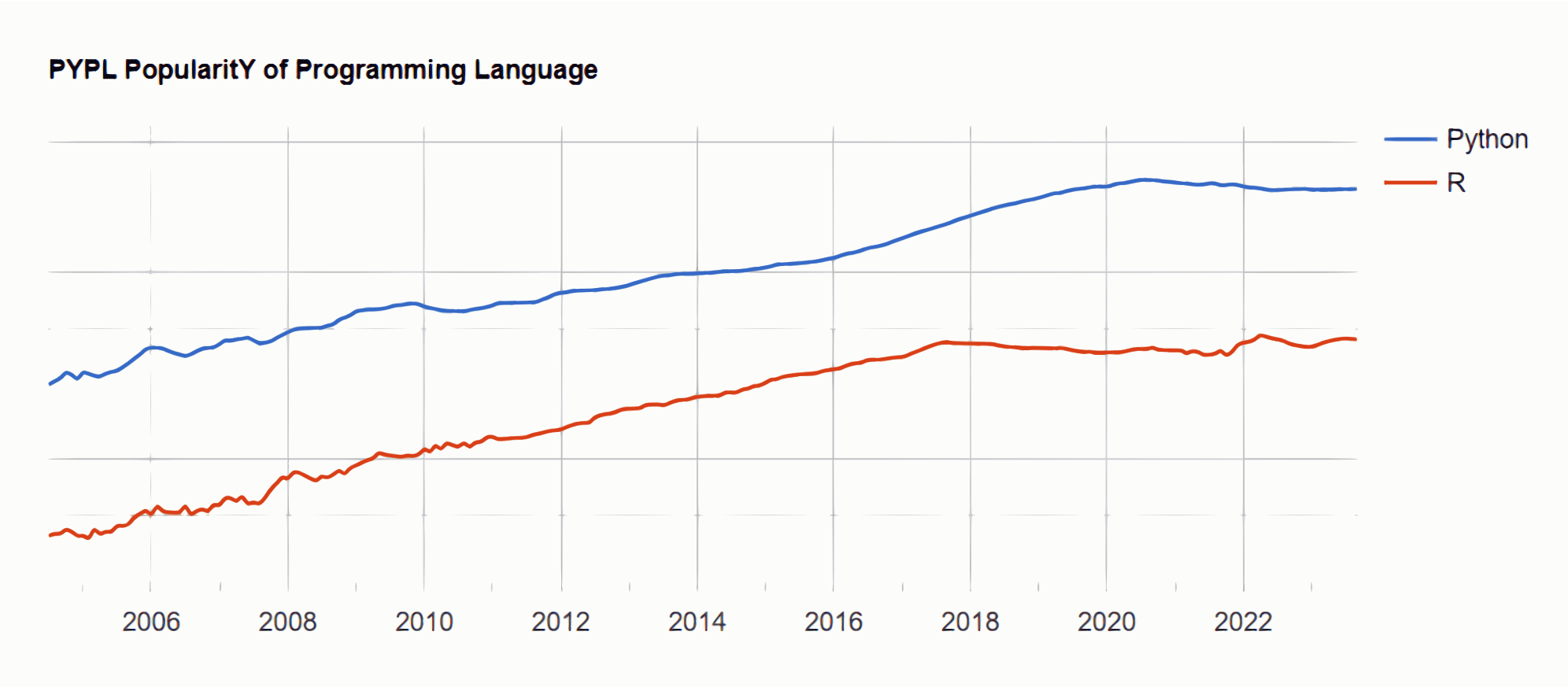
Worldwide, Feb 2025 :

Rank	Change	Language	Share	1-year trend
1		Python	29.85 %	+1.6 %
2		Java	15.15 %	-0.7 %
3		JavaScript	7.92 %	-0.8 %
4		C/C++	7.19 %	+0.5 %
5		C#	6.13 %	-0.5 %
6		R	4.55 %	-0.1 %
7		PHP	3.72 %	-0.8 %
8	↑↑	Rust	3.07 %	+0.6 %
9	↑↑	Objective-C	2.86 %	+0.5 %
10	↓↓	TypeScript	2.74 %	-0.1 %
11	↓↓	Swift	2.46 %	-0.3 %
12		Go	2.07 %	-0.1 %
13		Kotlin	1.87 %	-0.0 %
14		Matlab	1.7 %	+0.1 %

Source: https://pypl.github.io/PYPL.html

R - PYPL Index

The popularity is on a different scale compared to Python but still increasing:



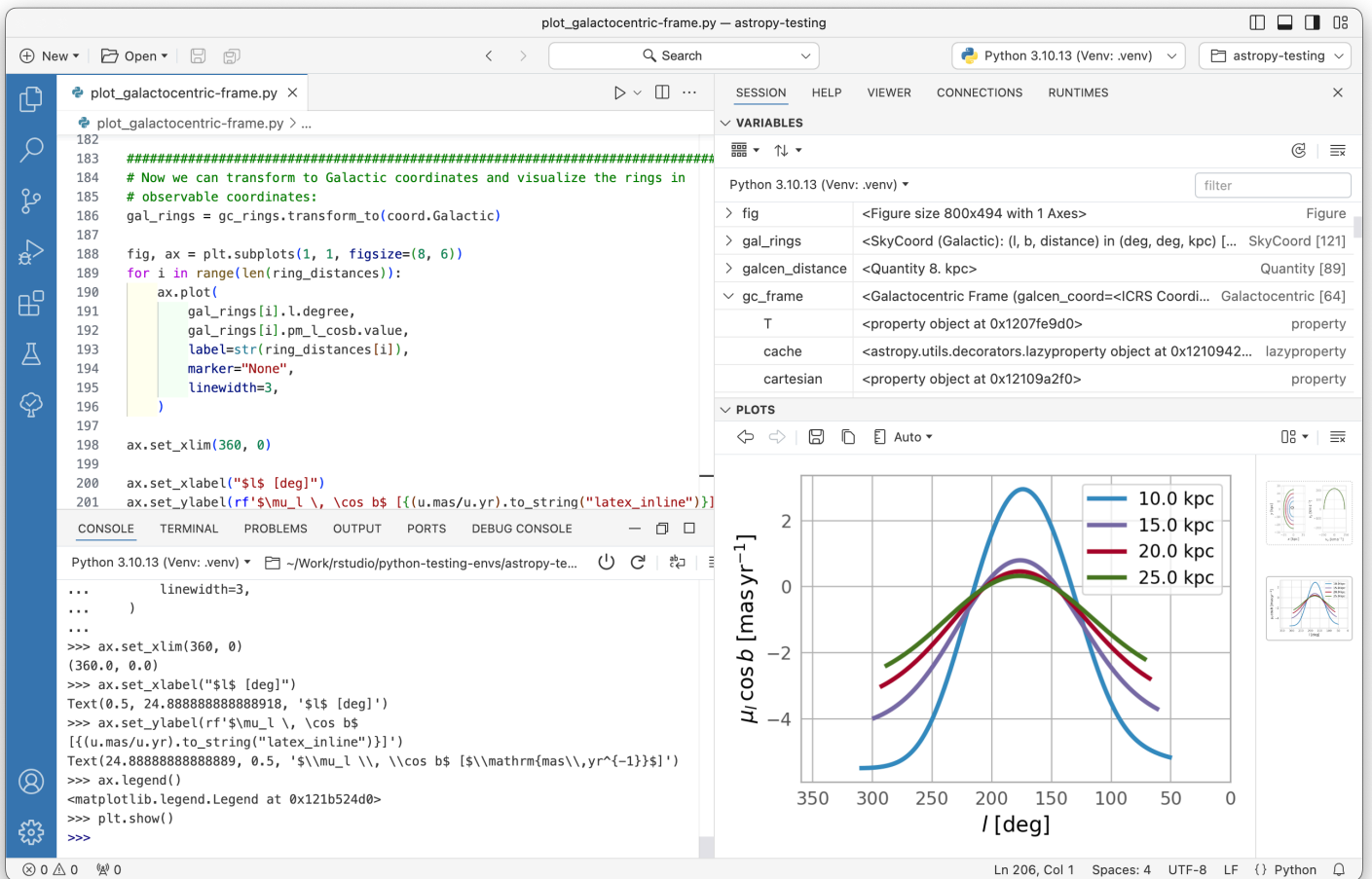
Source: https://pypl.github.io/PYPL.html

R or Python?

- Python is a very general-purpose language more powerful for general tasks.
- I find python very useful for programming experiments, image processing, automatizing tasks and interacting with the operating system
- R is still a little bit superior in terms of data manipulation and visualization. Python is faster and more powerful for complex models (e.g., machine learning, etc.)

Positron

Sometimes Python is not so easy to setup. In addition is not as interactive as R (i.e., line by line evaluation). Posit (ex. R Studio) recently created Positron that is a new IDE working with R and Python at the same way.



Modern R

- For purist programmers, R is weird: arrays starts with 1, object-oriented programming is hidden, a lot of built-in vectorized functions, etc. The The R Inferno book is really funny showing the strange R-stuff.
- Despite the weirdness, R is widely used because it is intuitive (for non-programmers) and made for statistics and data manipulation
- R is a language and as in spoken languages you can elegant, rude, ambiguous, funny, etc.
- There are some tips to improve the readability and reproducibility of your code

Functional Programming

> In computer science, functional programming is a programming paradigm where programs are constructed by applying and composing functions.

- Despite R can be used both with an **imperative** and **object-oriented approach**, the functional side is quite powerful.
- The basic idea is to decompose your code into small, testable and re-usable functions

Functional Programming, example...

We have a dataset (`mtcars`) and we want to calculate the mean, median, standard deviation, minimum and maximum of each column and store the result in a table.

```
head(mtcars)
```

```
#>                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
#> Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
#> Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
#> Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
#> Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
#> Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
#> Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```
str(mtcars)
```

```
#> 'data.frame':    32 obs. of  11 variables:
#>  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
#>  $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
#>  $ disp: num  160 160 108 258 360 ...
#>  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
#>  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
#>  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
#>  $ qsec: num  16.5 17 18.6 19.4 17 ...
#>  $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
```

```
#>  $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
#>  $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
#>  $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

Functional Programming

The standard (~imperative) option is using a `for` loop, iterating through columns, calculate the values and store into another data structure.

```
ncols <- ncol(mtcars)
means <- medians <- mins <- maxs <- rep(0, ncols)

for(i in 1:ncols){
  means[i] <- mean(mtcars[[i]])
  medians[i] <- median(mtcars[[i]])
  mins[i] <- min(mtcars[[i]])
  maxs[i] <- max(mtcars[[i]])
}

results <- data.frame(means, medians, mins, maxs)
results$col <- names(mtcars)

results
```

```
#>         means medians   mins    maxs  col
#> 1   20.090625  19.200 10.400  33.900  mpg
#> 2    6.187500   6.000  4.000   8.000  cyl
#> 3  230.721875 196.300 71.100 472.000 disp
#> 4  146.687500 123.000 52.000 335.000   hp
```

```
#> 5    3.596563   3.695  2.760   4.930 drat
#> 6    3.217250   3.325  1.513   5.424   wt
#> 7   17.848750  17.710 14.500  22.900 qsec
#> 8    0.437500   0.000  0.000   1.000   vs
#> 9    0.406250   0.000  0.000   1.000   am
#> 10   3.687500   4.000  3.000   5.000 gear
#> 11   2.812500   2.000  1.000   8.000 carb
```

Functional Programming

The main idea is to decompose the problem writing a function and loop over the columns of the dataframe:

```
summ <- function(x){
  data.frame(means = mean(x),
             medians = median(x),
             mins = min(x),
             maxs = max(x))
}
ncols <- ncol(mtcars)
dfs <- vector(mode = "list", length = ncols)

for(i in 1:ncols){
  dfs[[i]] <- summ(mtcars[[i]])
}
```

Functional Programming

```
results <- do.call(rbind, dfs)
results
```

```
#>          means medians   mins    maxs
#> 1   20.090625  19.200 10.400  33.900
#> 2    6.187500   6.000  4.000   8.000
#> 3  230.721875 196.300 71.100 472.000
#> 4  146.687500 123.000 52.000 335.000
#> 5    3.596563   3.695  2.760   4.930
#> 6    3.217250   3.325  1.513   5.424
#> 7   17.848750  17.710 14.500  22.900
#> 8    0.437500   0.000  0.000   1.000
#> 9    0.406250   0.000  0.000   1.000
#> 10   3.687500   4.000  3.000   5.000
#> 11   2.812500   2.000  1.000   8.000
```

Functional Programming

The actual real functional way require using the built-in iteration tools `*apply`. In this way you avoid writing the verbose `for` loop.

```
results <- lapply(mtcars, summ)
results <- do.call(rbind, results)
results
```

```
#>            means medians   mins    maxs
#> mpg   20.090625  19.200 10.400  33.900
#> cyl    6.187500   6.000  4.000   8.000
#> disp 230.721875 196.300 71.100 472.000
#> hp   146.687500 123.000 52.000 335.000
#> drat   3.596563   3.695  2.760   4.930
#> wt     3.217250   3.325  1.513   5.424
#> qsec  17.848750  17.710 14.500  22.900
#> vs     0.437500   0.000  0.000   1.000
#> am     0.406250   0.000  0.000   1.000
#> gear   3.687500   4.000  3.000   5.000
#> carb   2.812500   2.000  1.000   8.000
```

Functional Programming, `*apply`

- The `*apply` family is one of the best tool in R. The idea is pretty simple: apply a function to each element of a list.
- The powerful side is that in R everything can be considered as a list. A vector is a list of single elements, a dataframe is a list of columns etc.
- Internally, R is still using a `for` loop but the verbose part (preallocation, choosing the iterator, indexing) is encapsulated into the `*apply` function.

```
means <- rep(0, ncol(mtcars))
for(i in 1:length(means)){
  means[i] <- mean(mtcars[[i]])
}

# the same with sapply
means <- sapply(mtcars, mean)
```

`for` loops are bad?

`for` loops are the core of each operation in R (and in every programming language). For complex operation thery are more readable and effective compared to `*apply`. In R we need extra care for writing efficent `for` loops.

Extremely slow, no preallocation:

```
res <- c()
for(i in 1:1000){
  # do something
  res[i] <- x
}
```

Very fast, no difference compared to `*apply`

With *apply you can do crazy stuff!

```
funs <- list(mean = mean, sd = sd, min = min, max = max, median = median)
sapply(funs, function(f) lapply(mtcars, function(x) f(x)))
```

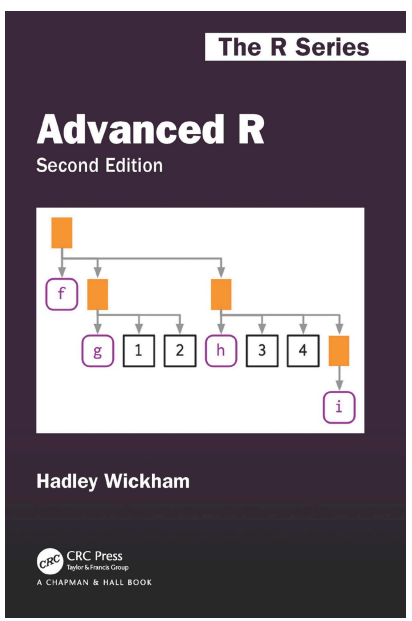
```
#>       mean     sd         min   max   median
#> mpg  20.09062 6.026948  10.4  33.9  19.2
#> cyl  6.1875   1.785922  4     8     6
#> disp 230.7219 123.9387  71.1  472   196.3
#> hp   146.6875 68.56287  52    335   123
#> drat 3.596563 0.5346787 2.76  4.93  3.695
#> wt   3.21725  0.9784574 1.513 5.424 3.325
#> qsec 17.84875 1.786943  14.5  22.9  17.71
#> vs   0.4375   0.5040161 0     1     0
#> am   0.40625  0.4989909 0     1     0
#> gear 3.6875   0.7378041 3     5     4
#> carb 2.8125   1.6152    1     8     2
```

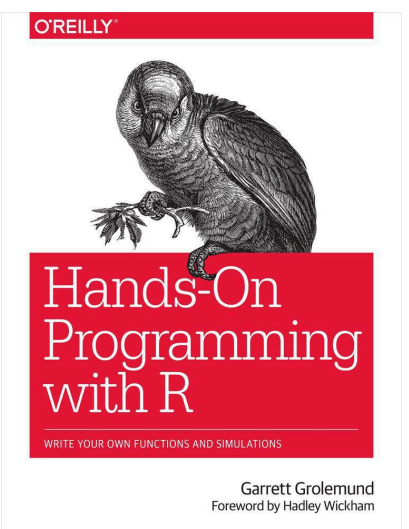
Why functional programming?

- We can write less and reusable code that can be shared and used in multiple projects
- The scripts are more compact, easy to modify and less error prone (imagine that you want to improve the `summ` function, you only need to change it once instead of touching the `for` loop)
- Functions can be easily and consistently documented (see roxygen documentation) improving the reproducibility and readability of your code

More about functional programming in R

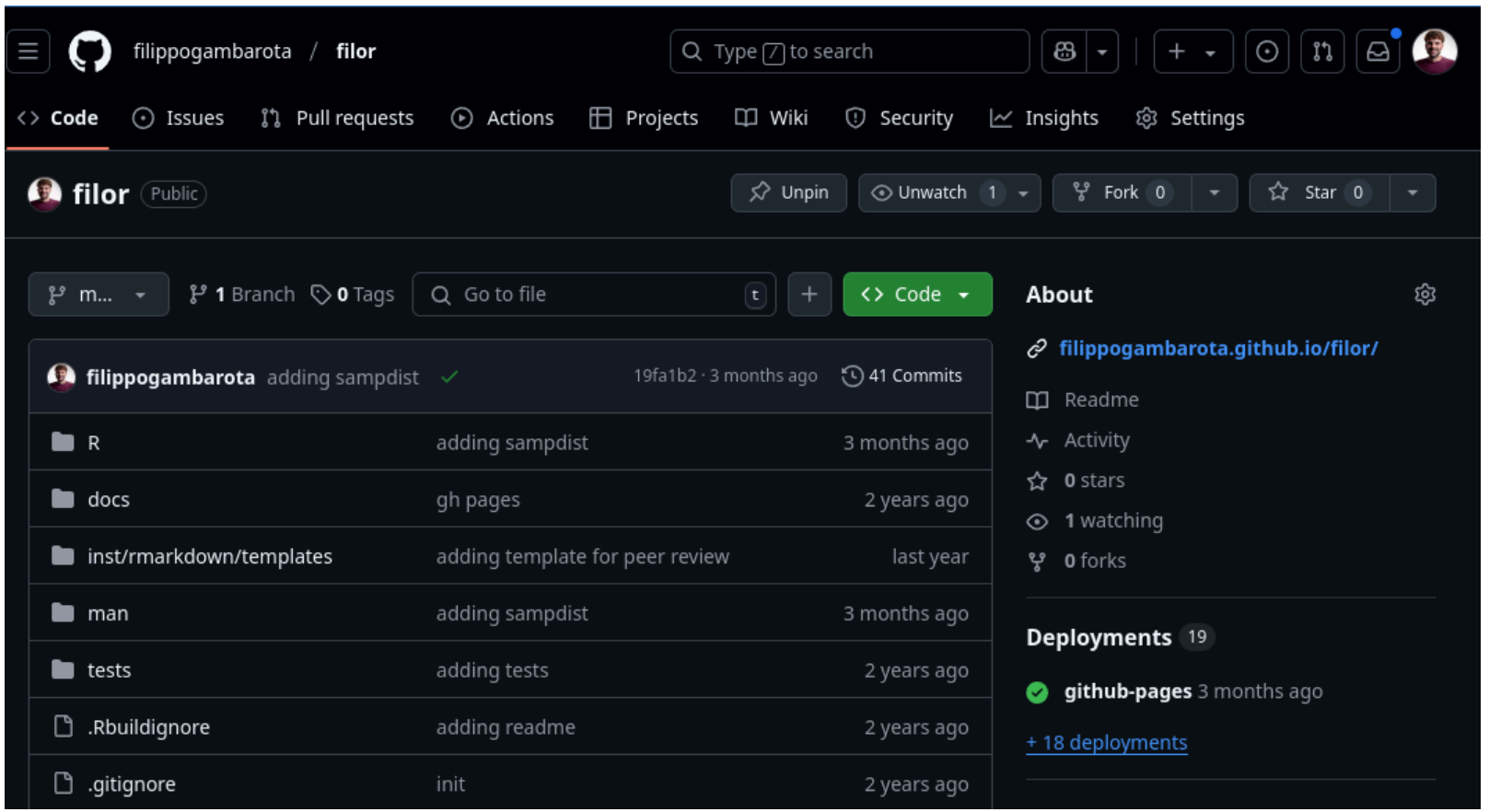
- Advanced R by Hadley Wickham, section on Functional Programming (https://adv-r.hadley.nz/fp.html)
- Hands-On Programming with R by Garrett Grolemund https://rstudio-education.github.io/hopr/
- Hadley Wickham: The Joy of Functional Programming (for Data Science)
- Bruno Rodrigues Youtube Channel





A more advanced approach, R packages

R packages are not only on CRAN. You can (pretty) easily create a package and put it on Github. For example, if you keep using some functions in your project, write a general version and put them into a package.



github.com/filippogambarota/filor

A more advanced approach, R packages

If your functions are project-specific you can define them into your scripts or write some R scripts only with functions and `source()` them into the global environment.

```
project/
├─ R/
│  ├─ utils.R
├─ analysis.R
```

And inside `utils.R` you have some functions:

```
myfun <- function(x) {
  # something
}
```

Then you can load the function using `source("R/utils.R)` at the beginning of `analysis.R`:

```
source("R/utils.R")
```

Analysis project as R package

The R project structure is really interesting to organize a data analysis pipeline. In fact, you can use the project structure. Vuorre & Crump (2021) and Marwick et al. (2018) describe in details the idea.

The general approach is:

1. Create an R Studio project `.Rproj` file
2. Create your directories, put scripts, data, etc.
3. Create an `R/` folder and put your scripts with functions
4. Create a `DESCRIPTION` file using `usethis::use_description(check_name = FALSE)`
5. Then you can load your functions without source and with `devtools::load_all()` (same as `library()`)

> Let’s see an example!

The Tidy approach

The `tidyverse` is a series of high-quality R packages to do modern data science:

- data manipulation (`dplyr`, `tidyr`)
- plotting (`ggplot2`)
- reporting (`rmarkdown`)
- string manipulation (`stringr`)
- functionals (`purrr`)
- …

The Tidy approach - Pipes

One of the great improvement from the `tidyverse` is the usage of the pipe `%>%` now introduced in base R as `|>`. You will se these symbols a lot when looking at modern R code.

The idea is very simple, the standard pattern to apply a function is `function(argument)`. The pipe can reverse the pattern as `argument |> function()`. Normally when we apply multiple functions progressively the pattern is this:

```
x <- rnorm(100)
x <- round(x, 3)
x <- abs(x)
x <- as.character(x)
```

The Tidy approach - Pipes

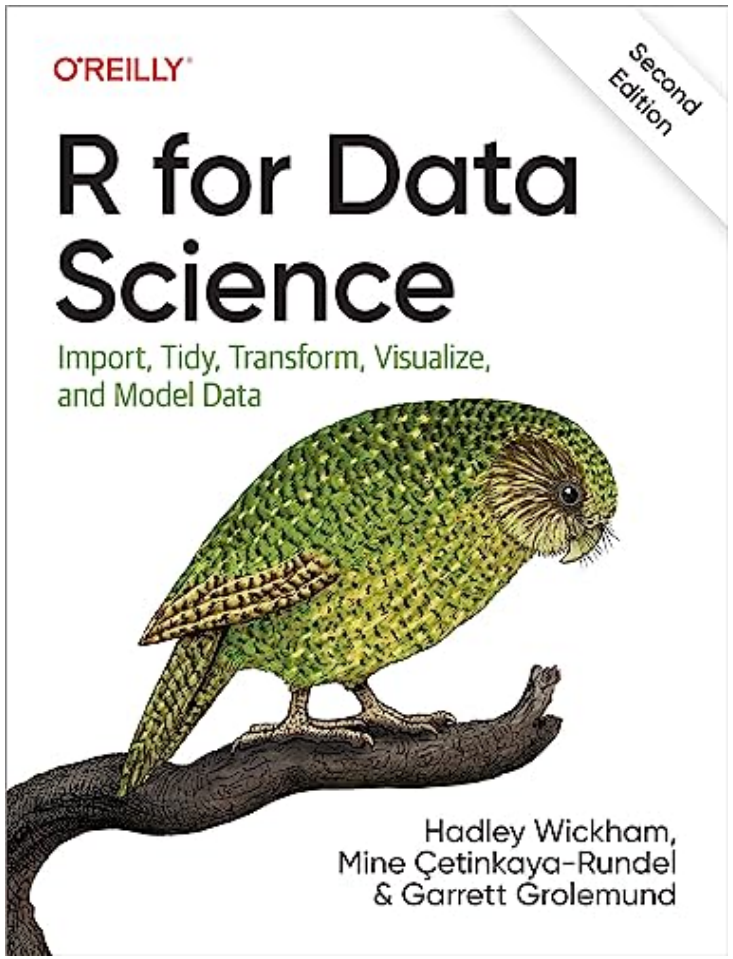
When using the pipe, we remove the redundand assignment `<-` pattern:

```
x <- rnorm(100)
x |>
  round(3) |>
  abs() |>
  as.character()
```

The pipe can be read as *"from **x** apply `round`, then `abs`, etc."*. The first argument of the piped function is assumed to be the result of the previus call.

More about the Tidy approach

The `tidy` approach contains tons of functions and packages. The overall philosophy can be deepen in the R for Data Science book.



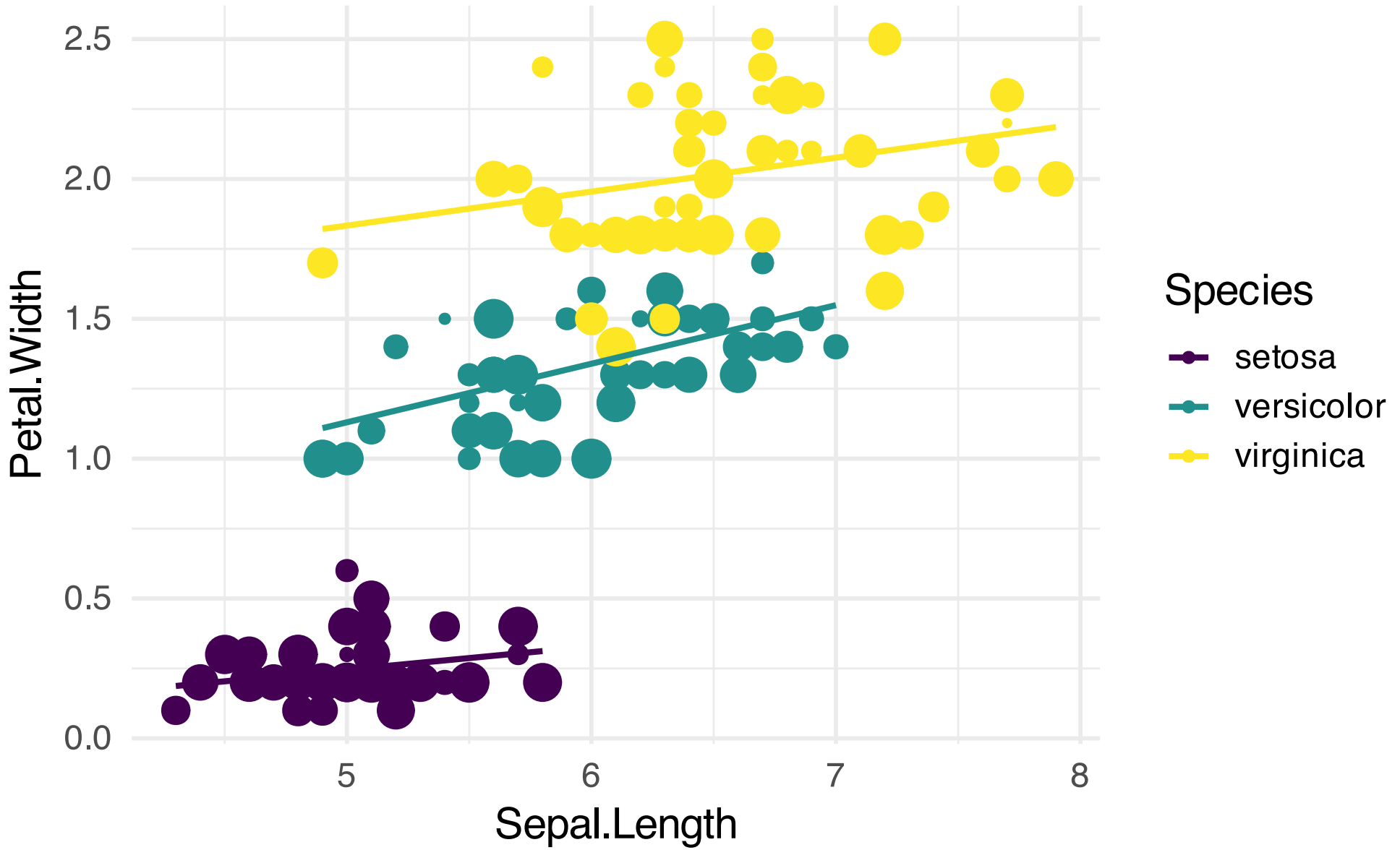
https://r4ds.hadley.nz/

ggplot2

Only an quick mention to `ggplot2` https://ggplot2-book.org/ (part of the `tidyverse`) that is an amazing package for data visualization following the *piping* and *tidy* approach. Is the implementation of the **grammar of graphics** idea.

```
library(tidyverse)

iris |>
  mutate(wi = runif(n())) |>
  ggplot(aes(x = Sepal.Length, y = Petal.Width, color = Species)) +
  geom_point(aes(size = wi)) +
  geom_smooth(method = "lm", se = FALSE)
  guides(size = "none") +
  theme_minimal(15)
```

ggplot2

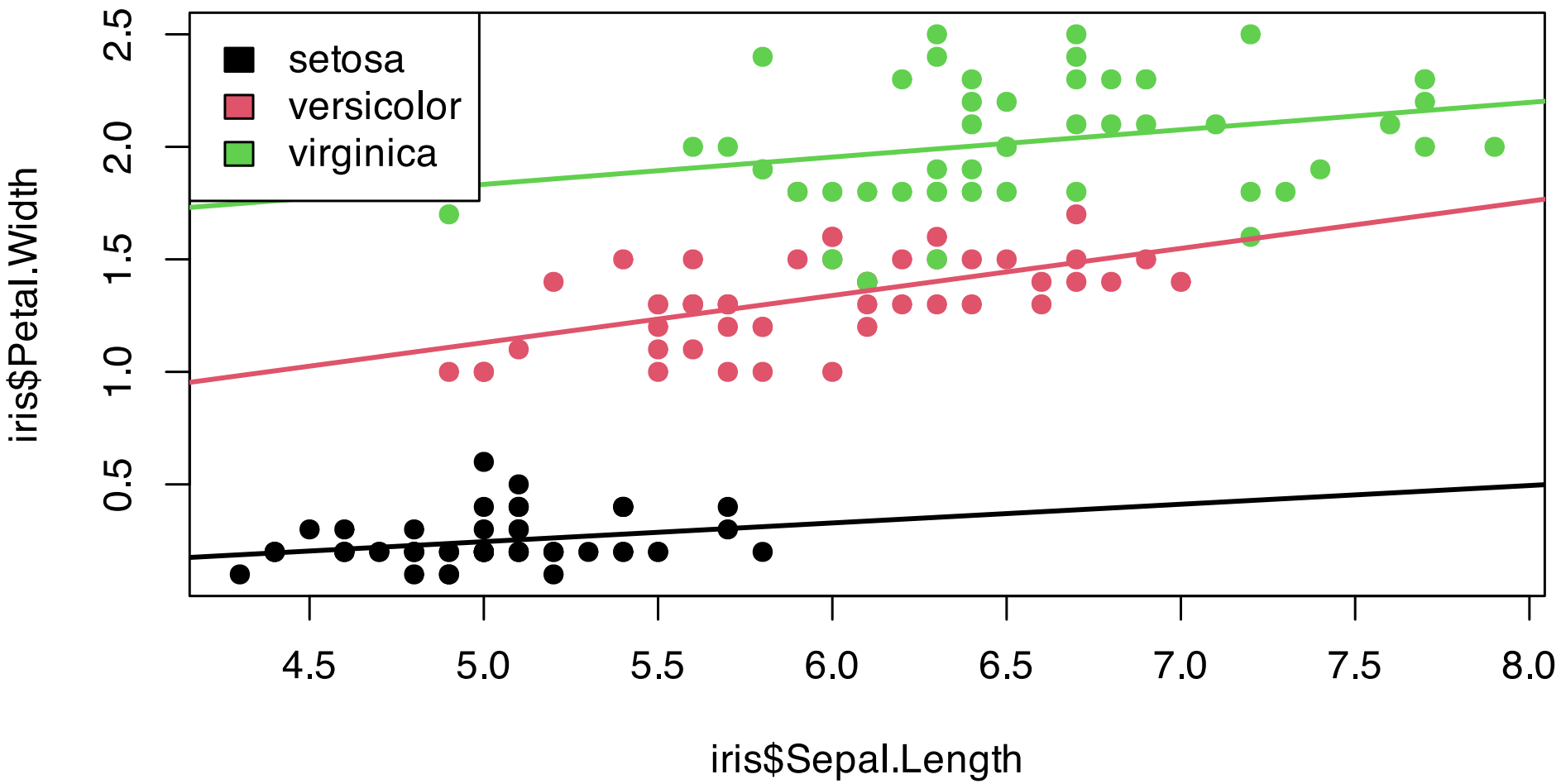

2.5
2.0
1.5
1.0
0.5
0.0
Petal.Width
5
6
7
8
Sepal.Length
Species
setosa
versicolor
virginica

Base R version

More verbose, more hard coding, more steps and intermediate objects.

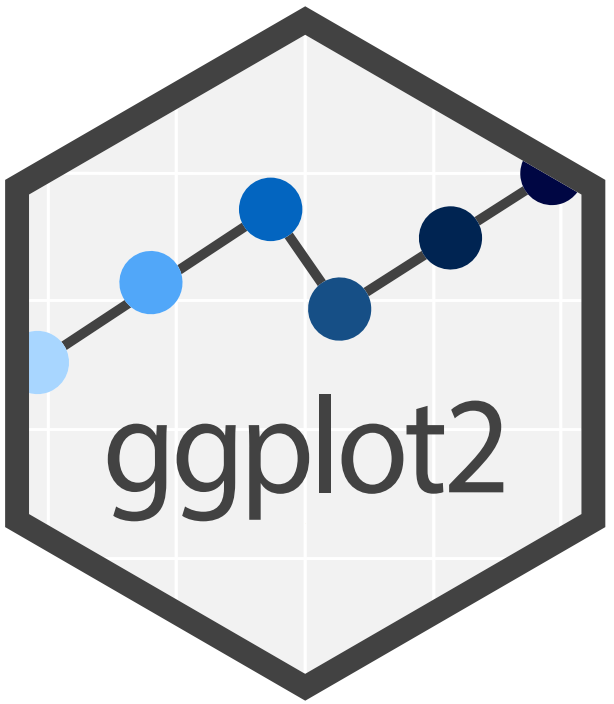
```
iris_l <- split(iris, iris$Species)
lms <- lapply(iris_l, function(x) lm(Petal.Width ~ Sepal.Length, data = x))

plot(iris$Sepal.Length,
     iris$Petal.Width,
     col = as.numeric(iris$Species), pch = 19)

abline(lms[[1]], col = 1, lwd = 2)
abline(lms[[2]], col = 2, lwd = 2)
abline(lms[[3]], col = 3, lwd = 2)

legend("topleft", legend = levels(iris$Species), fill = 1:3)
```

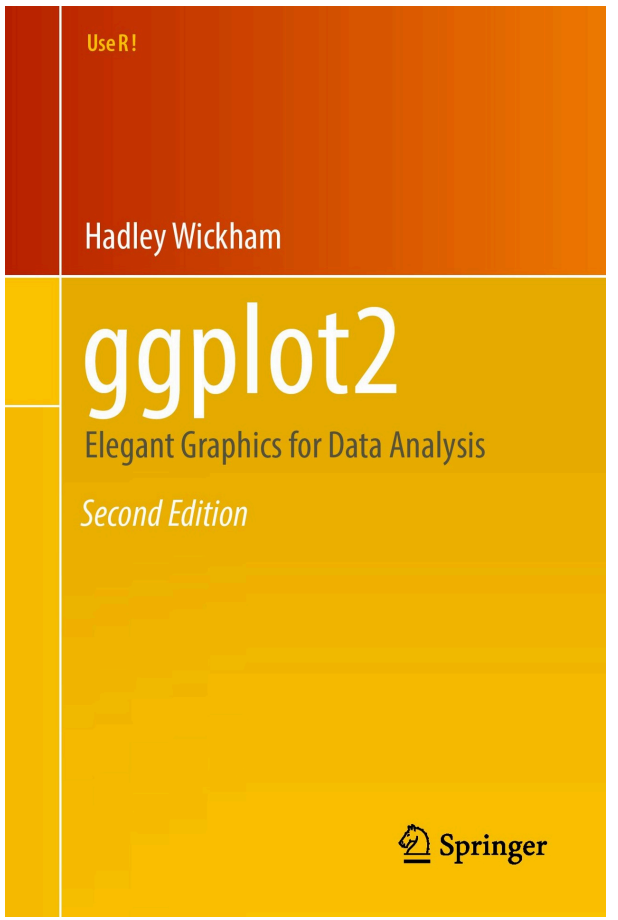
Base R version



More on ggplot2

The `ggplot2` book https://ggplot2-book.org/ is a great resource to produce high-quality, publication ready plots. Clearly, the advantage of producing the figures entirely writing code are immense in terms of reusability and reproducibility.





Something crazy in the `tidyverse`

Without going into details, I want to show you a very interesting approach that you can do with the `tidyverse` functions.

Let's assume you want to do a leave-one-out analysis thus fitting the same models on a dataset, removing one observation at time.

You can do it in base R with a loop or other methods, but the see so-called *many-models* approach. See https://r4ds.had.co.nz/many-models.html and https://www.youtube.com/watch?v=rz3_FDVt9eg.

Something crazy in the `tidyverse`

Let's define some functions:

```
leave1out <- function(data){
  idx <- 1:nrow(data)
  ll <- lapply(idx, function(i) data[-i, ])
  names(ll) <- paste0("no", idx)
  c(no0 = list(data), ll)
}

fit_model <- function(data){
  lm(Sepal.Length ~ Petal.Width, data = data)
}
```

Something crazy in the `tidyverse`

```
dat <- tibble(data = leave1out(iris[1:20, ]))
dat |>
  mutate(removed = names(data)) |>
  head()
```

```
#> # A tibble: 6 × 2
#>   data           removed
#>   <named list>   <chr>
#> 1 <df [20 × 5]> no0
#> 2 <df [19 × 5]> no1
#> 3 <df [19 × 5]> no2
#> 4 <df [19 × 5]> no3
#> 5 <df [19 × 5]> no4
#> 6 <df [19 × 5]> no5
```

Something crazy in the `tidyverse`

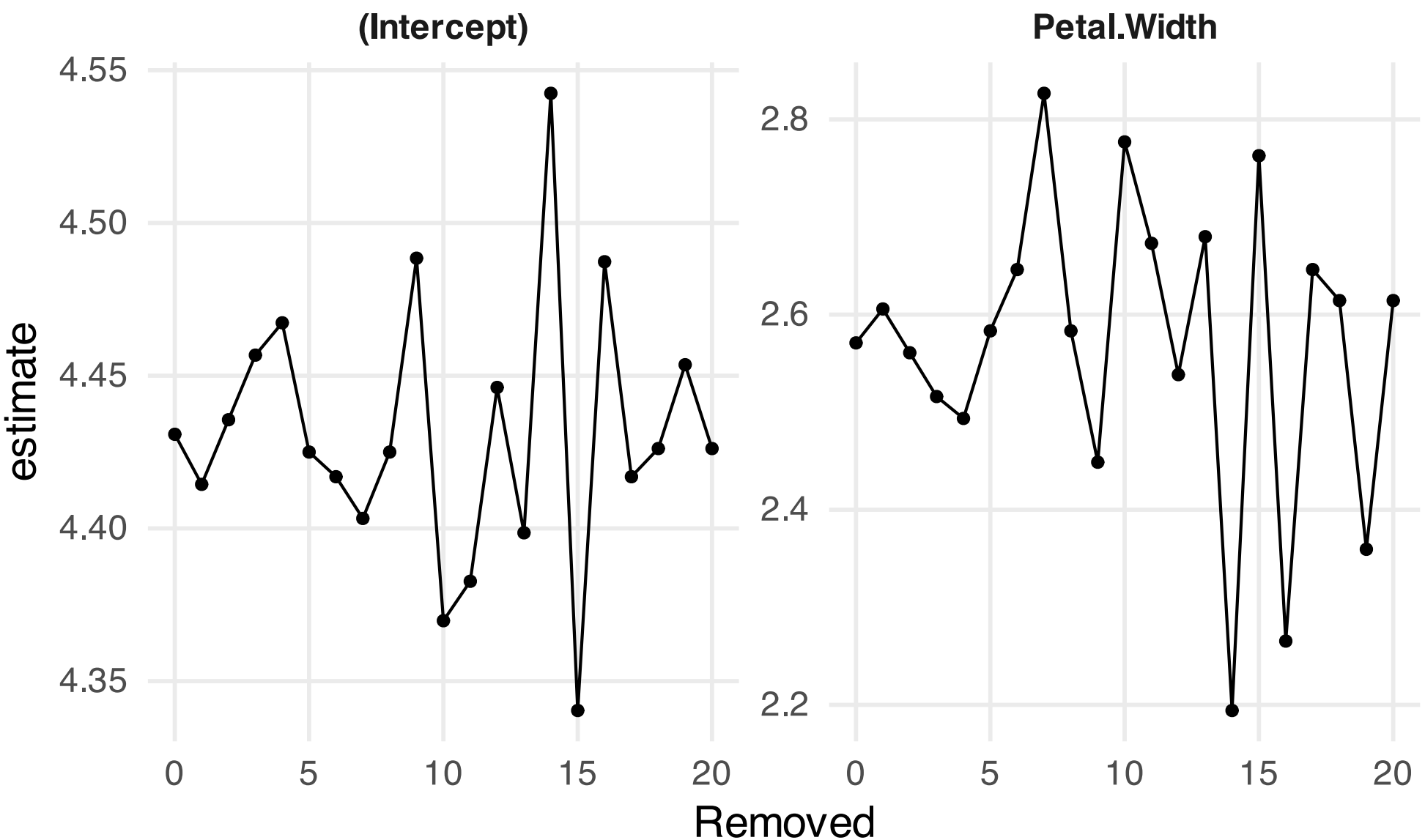
```
dat |>
  mutate(removed = names(data)) |>
  mutate(fit = map(data, fit_model),
         results = map(fit, broom::tidy)) |>
  head()
```

```
#> # A tibble: 6 × 4
#>   data           removed fit           results
#>   <named list>   <chr>   <named list> <named list>
#> 1 <df [20 × 5]> no0     <lm>         <tibble [2 × 5]>
#> 2 <df [19 × 5]> no1     <lm>         <tibble [2 × 5]>
#> 3 <df [19 × 5]> no2     <lm>         <tibble [2 × 5]>
#> 4 <df [19 × 5]> no3     <lm>         <tibble [2 × 5]>
#> 5 <df [19 × 5]> no4     <lm>         <tibble [2 × 5]>
#> 6 <df [19 × 5]> no5     <lm>         <tibble [2 × 5]>
```

Something crazy in the `tidyverse`

```
dat |>
  mutate(removed = names(data)) |>
  mutate(fit = map(data, fit_model),
         results = map(fit, broom::tidy)) |>
  unnest(results) |>
  ggplot(aes(x = removed, y = estimate)) +
  geom_point() +
  geom_line() +
  facet_wrap(~term, scales = "free")
```

Something crazy in the `tidyverse`



Quick tables

```
gtsummary::tbl_summary(iris)
```

Characteristic	N = 150[1]
Sepal.Length	5.80 (5.10, 6.40)
Sepal.Width	3.00 (2.80, 3.30)
Petal.Length	4.35 (1.60, 5.10)
Petal.Width	1.30 (0.30, 1.80)
Species	
setosa	50 (33%)
versicolor	50 (33%)
virginica	50 (33%)

[1] Median (Q1, Q3); n (%)

Quick tables from models

```
fit <- lm(Sepal.Length ~ Petal.Width, data = iris)
sjPlot::tab_model(fit)
```

	Sepal.Length		
Predictors	*Estimates*	*CI*	*p*
(Intercept)	4.78	4.63 – 4.92	**<0.001**
Petal Width	0.89	0.79 – 0.99	**<0.001**
Observations	150		
R^2 / R^2 adjusted	0.669 / 0.667		

Quick tables from models

```
gtsummary::tbl_regression(fit)
```

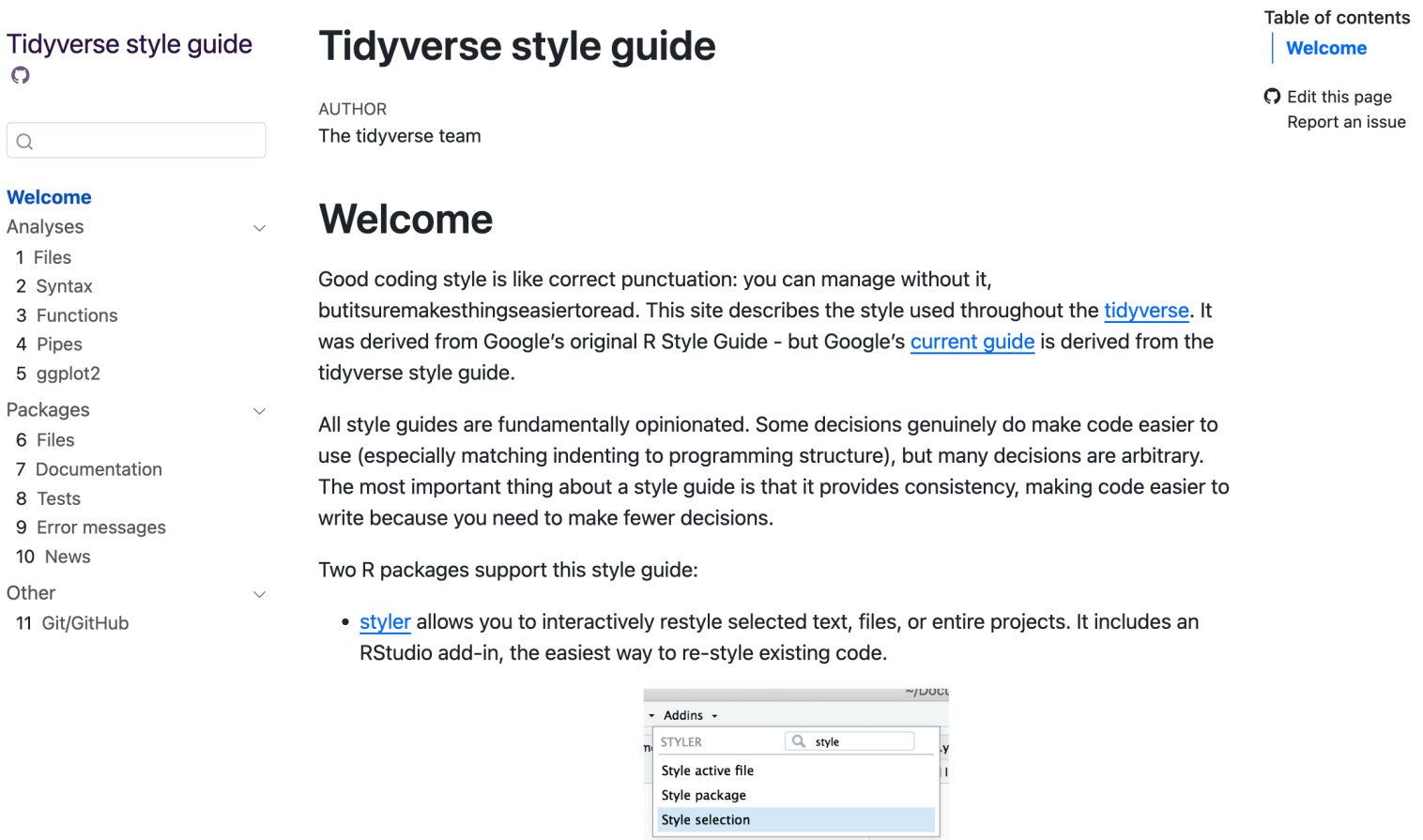
Characteristic	Beta	95% CI[1]	p-value
Petal.Width	0.89	0.79, 0.99	<0.001

[1] CI = Confidence Interval

> Tips on writing good R code

The `tidyverse` style guide

It's a series of best practices and suggestion to create consistent and readable R code.



http://style.tidyverse.org/

What is good (R) code

- organized scripts
- commenting and documenting
- consistent and self-explanatory variables and functions naming

Organized scripts

Global operations at the beginning of the script:

- loading datasets
- loading packages
- changing general options (`options()`)

```
# packages
library(tidyverse)
library(lme4)

# options

options(scipen = 999)

# loading data
dat <- read.csv(...)
```

Functions to avoid repetition

Avoid repeating the same operation multiple times in the script. The rule is, if you are doing the same operation more than two times, write a function.

A function can be re-used, tested and changed just one time affecting the whole project.

Comments, comments and comments…

Write the code for your future self and for others, not for yourself right now.

Try to open a (not well documented) old coding project after a couple of years and you will understand :)

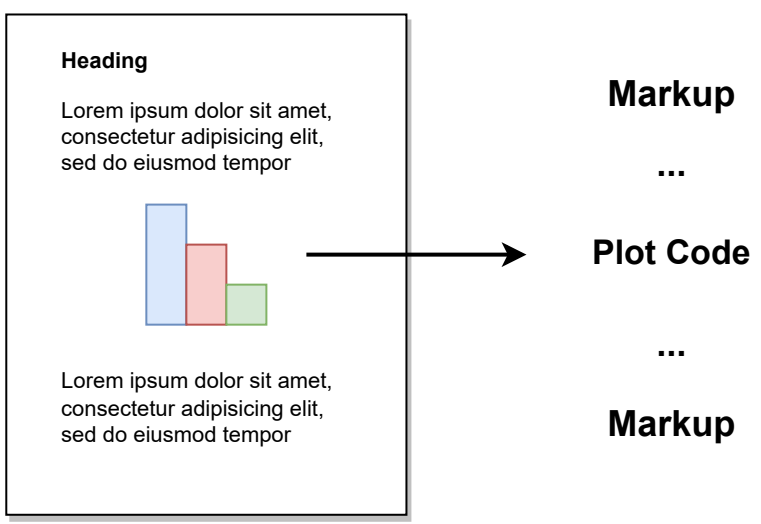
Invest time in writing more comprehensible and documented code for you and others.

> Literate Programming

Literate Programming[1]

> Donald Knuth first defined literate programming as a script, notebook, or computational document that contains an explanation of the program logic in a natural language, interspersed with snippets of macros and source code, which can be compiled and rerun

For example **jupyter notebooks**, **R Markdown** and now **Quarto** are literate programming frameworks to integrate code and text.



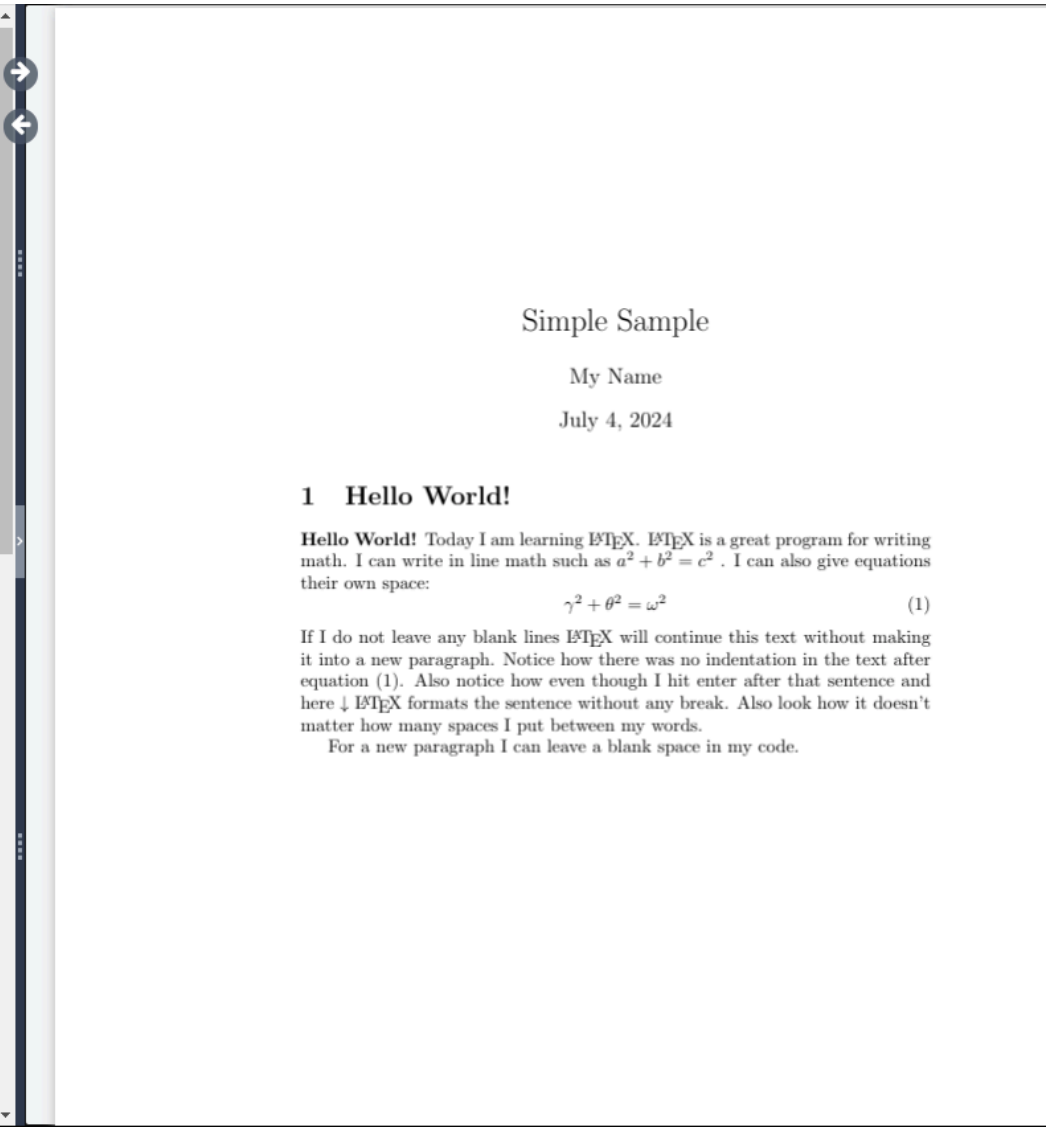
Literate Programming, the markup language

Beyond the coding part, the markup language is the core element of a literate programming framework. The idea of a markup language is separating the result from what you actually write. Some examples are:

- LaTeX
- HTML
- Markdown
- XML
- …

LaTeX [1]

```
% This is a simple sample document.  For more complicated documents take a look in the exercise tab. Note that everything that comes after a % symbol is treated as comment and ignored when the code is compiled.

\documentclass{article} % \documentclass{} is the first command in any LaTeX code.  It is used to define what kind of document you are creating such as an article or a book, and begins the document preamble

\usepackage{amsmath} % \usepackage is a command that allows you to add functionality to your LaTeX code

\title{Simple Sample} % Sets article title
\author{My Name} % Sets authors name
\date{\today} % Sets date for date compiled

% The preamble ends with the command \begin{document}
\begin{document} % All begin commands must be paired with an end command somewhere
    \maketitle % creates title using information in preamble (title, author, date)

    \section{Hello World!} % creates a section

    \textbf{Hello World!} Today I am learning \LaTeX. %notice how the command will end at the first non-alphabet charecter such as the . after \LaTeX
     \LaTeX{} is a great program for writing math. I can write in line math such as $a^2+b^2=c^2$ %$ tells LaTexX to compile as math
     . I can also give equations their own space:
    \begin{equation} % Creates an equation environment and is compiled as math
    \gamma^2+\theta^2=\omega^2
    \end{equation}
    If I do not leave any blank lines \LaTeX{} will continue  this text without making it into a new paragraph.  Notice how there was no indentation in the text after equation (1).
    Also notice how even though I hit enter after that sentence and here $\downarrow$
     \LaTeX{} formats the sentence without any break.  Also   look  how      it  doesn't     matter          how     many  spaces     I put     between  my    words.

```



Simple Sample

My Name

July 4, 2024

1 Hello World!

Hello World! Today I am learning LaTeX. LaTeX is a great program for writing math. I can write in line math such as $a^2 + b^2 = c^2$. I can also give equations their own space:

$$\gamma^2 + \theta^2 = \omega^2 \tag{1}$$

If I do not leave any blank lines LaTeX will continue this text without making it into a new paragraph. Notice how there was no indentation in the text after equation (1). Also notice how even though I hit enter after that sentence and here ↓ LaTeX formats the sentence without any break. Also look how it doesn't matter how many spaces I put between my words.

For a new paragraph I can leave a blank space in my code.

HTML

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Heading</h1>

Lorem Ipsum è un testo segnaposto utilizzato nel settore della tipografia e della stam

<h2>My Second Heading</h2>

Lorem Ipsum è un testo segnaposto utilizzato nel settore della tipografia e della stam

Lorem Ipsum è considerato il testo segnaposto standard sin dal sedicesimo secolo, quan

tipografo prese una cassetta di caratteri e li assemblò per preparare un testo campion

È sopravvissuto non solo a più di cinque secoli, ma anche al passaggio alla videoimpag
```

Markdown[1]

Markdown Live Preview Reset Copy Sync scroll

```
# Markdown syntax guide

## Headers

# This is a Heading h1
## This is a Heading h2
###### This is a Heading h6

## Emphasis

*This text will be italic*
_This will also be italic_

**This text will be bold**
__This will also be bold__

_You **can** combine them_

## Lists

### Unordered

* Item 1
* Item 2
* Item 2a
* Item 2b
    * Item 3a
    * Item 3b
```


Markdown syntax guide

Headers

This is a Heading h1

This is a Heading h2

This is a Heading h6

Emphasis

This text will be italic
This will also be italic

Markdown

Markdown is one of the most popular markup languages for several reasons:

- easy to write and read compared to Latex and HTML
- easy to convert from Markdown to basically every other format using pandoc
- easy to implement new features

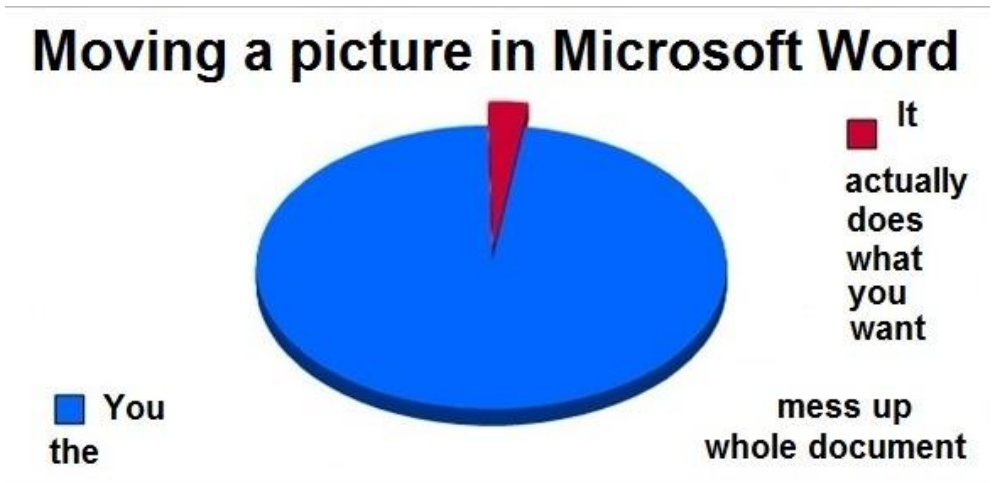
Markdown (source code)

```
## Markdown

Markdown is one of the most popular markup languages for several reasons:

- easy to write and read compared to Latex and HTML
- easy to convert from Markdown to basically every other format using `pandoc`
- easy to implement new features
```

Also the source code can be used, compared to Latex or HTML, to take notes and read. Latex and HTML need to be compiled otherwise they are very hard to read.

What’s wrong about Microsoft Word?

MS Word is a WYSIWYG (*what you see is what you get editor*) that force users to think about formatting, numbering, etc. Markup languages receive the content (plain text) and the rules and creates the final document.



What’s wrong about Microsoft Word?

Beyond the pure writing process, there are other aspects related to research data.

- writing math formulas
- reporting statistics in the text
- producing tables
- producing plots

In MS Word (or similar) we need to produce everything outside and then manually put figures and tables.

The solution… Quarto

Quarto (https://quarto.org/) is the evolution of R Markdown that integrate a programming language with the Markdown markup language. It is very simple but quite powerful.

Basic Markdown

Markdown can be learned in minutes. You can go to the following link https://quarto.org/docs/authoring/markdown-basics.html and try to understand the syntax.

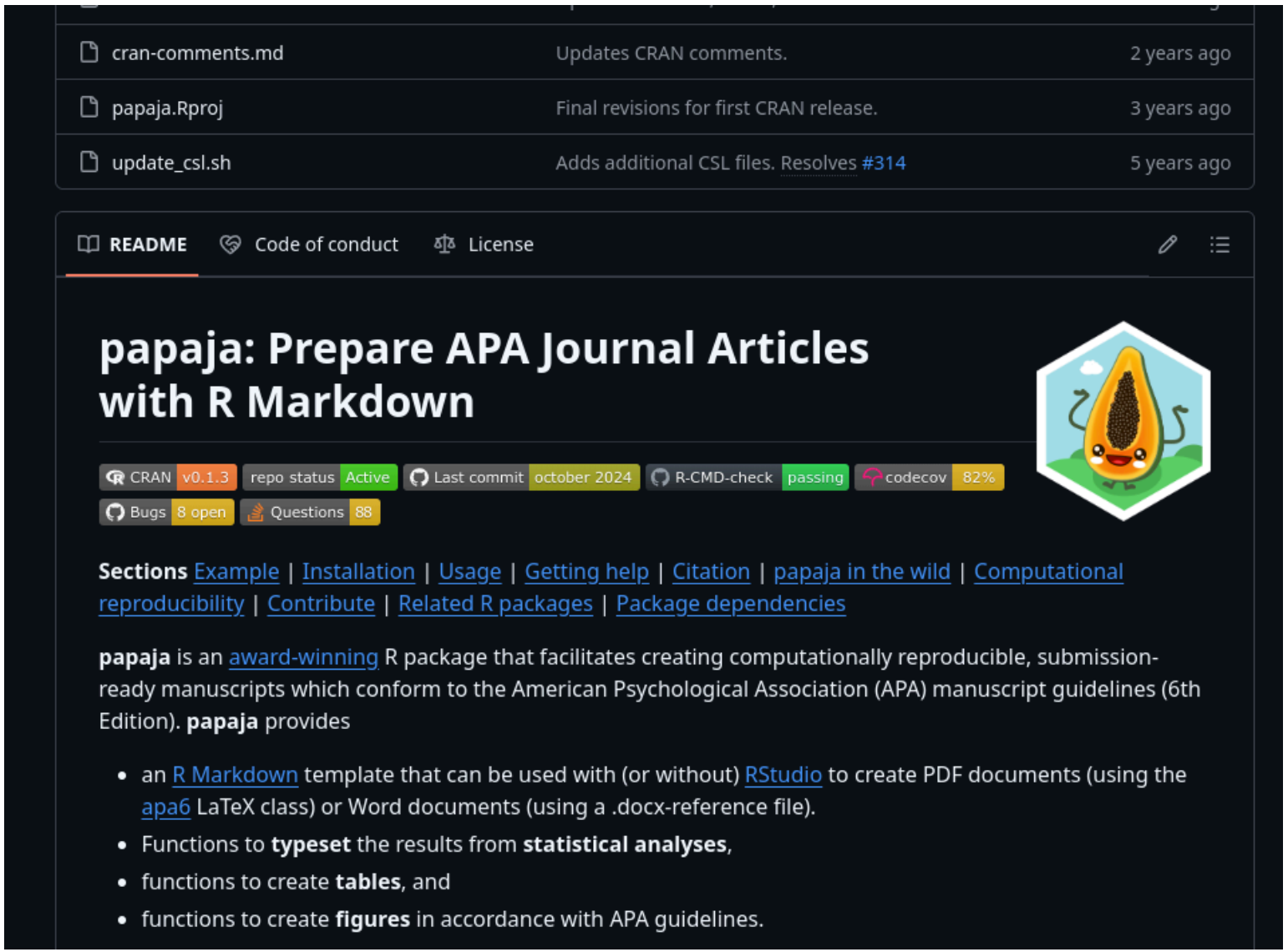
> Let’s see a practical example!

More about Quarto and R Markdown

The topic is extremely vast. You can do everything in Quarto, a website, thesis, your CV, etc.

- Yihui Xie - R Markdown Cookbook https://bookdown.org/yihui/rmarkdown-cookbook/
- Yihui Xie - R Markdown: The Definitive Guide https://bookdown.org/yihui/rmarkdown/
- Quarto documentation https://quarto.org/docs/guide/

Writing papers, papaja



cran-comments.md	Updates CRAN comments.	2 years ago
papaja.Rproj	Final revisions for first CRAN release.	3 years ago
update_csl.sh	Adds additional CSL files. Resolves #314	5 years ago

README | Code of conduct | License

papaja: Prepare APA Journal Articles with R Markdown

CRAN v0.1.3 | repo status Active | Last commit october 2024 | R-CMD-check passing | codecov 82%
Bugs 8 open | Questions 88

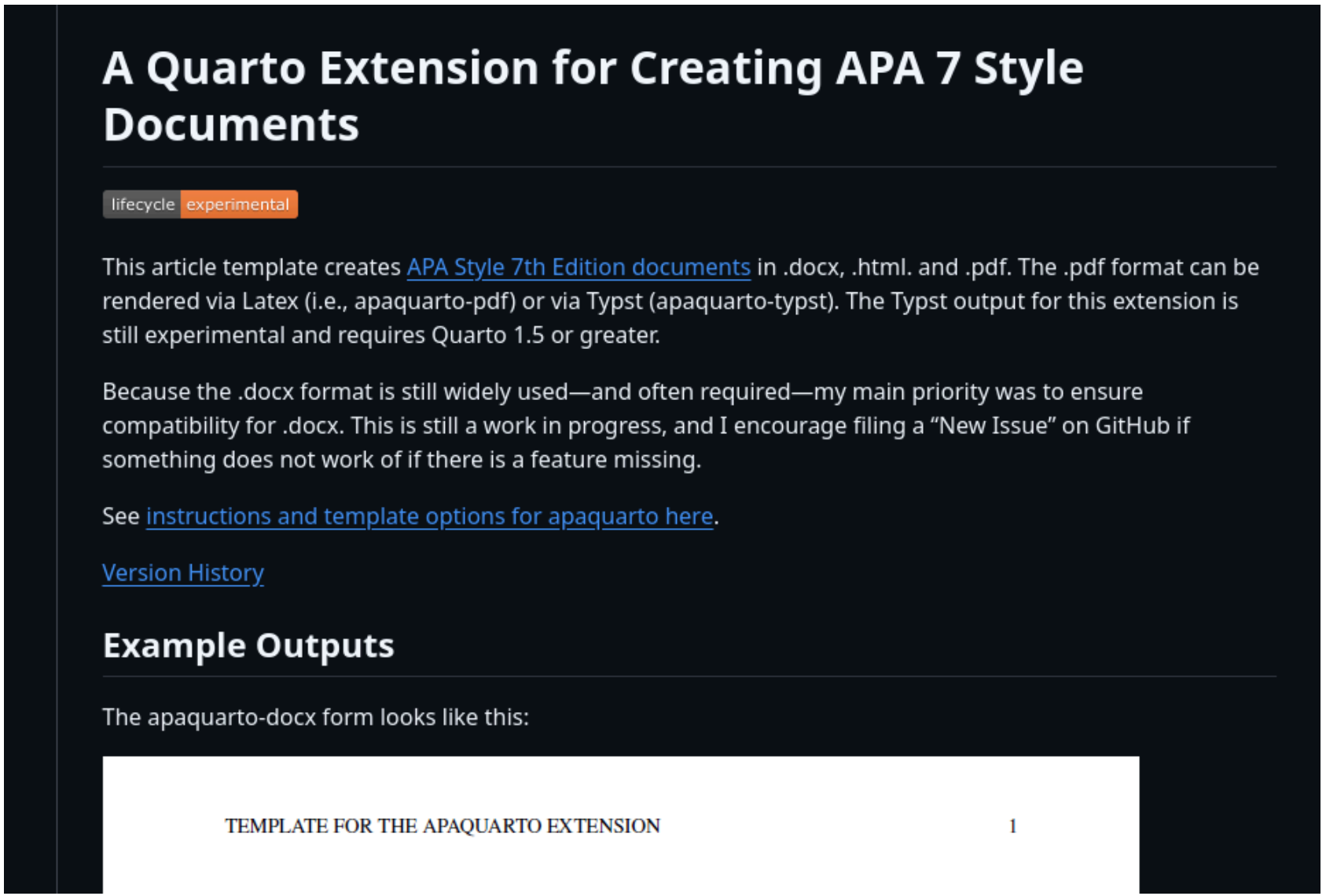
Sections Example | Installation | Usage | Getting help | Citation | papaja in the wild | Computational reproducibility | Contribute | Related R packages | Package dependencies

papaja is an award-winning R package that facilitates creating computationally reproducible, submission-ready manuscripts which conform to the American Psychological Association (APA) manuscript guidelines (6th Edition). **papaja** provides

- an R Markdown template that can be used with (or without) RStudio to create PDF documents (using the apa6 LaTeX class) or Word documents (using a .docx-reference file).
- Functions to **typeset** the results from **statistical analyses**,
- functions to create **tables**, and
- functions to create **figures** in accordance with APA guidelines.

https://github.com/crsh/papaja

Writing papers, apaquarto



A Quarto Extension for Creating APA 7 Style Documents

lifecycle experimental

This article template creates APA Style 7th Edition documents in .docx, .html. and .pdf. The .pdf format can be rendered via Latex (i.e., apaquarto-pdf) or via Typst (apaquarto-typst). The Typst output for this extension is still experimental and requires Quarto 1.5 or greater.

Because the .docx format is still widely used—and often required—my main priority was to ensure compatibility for .docx. This is still a work in progress, and I encourage filing a "New Issue" on GitHub if something does not work of if there is a feature missing.

See instructions and template options for apaquarto here.

Version History

Example Outputs

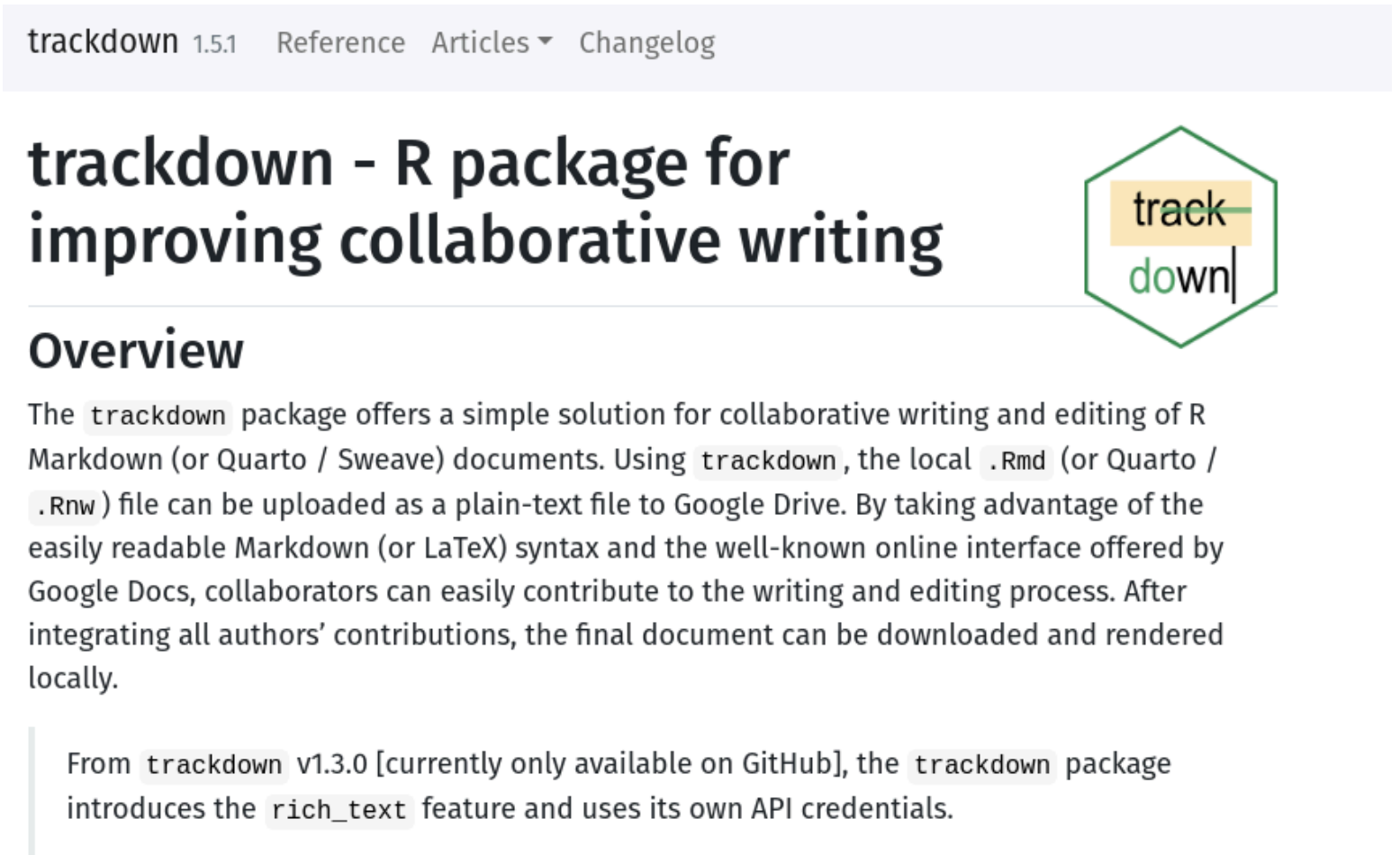
The apaquarto-docx form looks like this:

TEMPLATE FOR THE APAQUARTO EXTENSION 1

https://github.com/wjschne/apaquarto

Collaborating! (TBH not so easy)

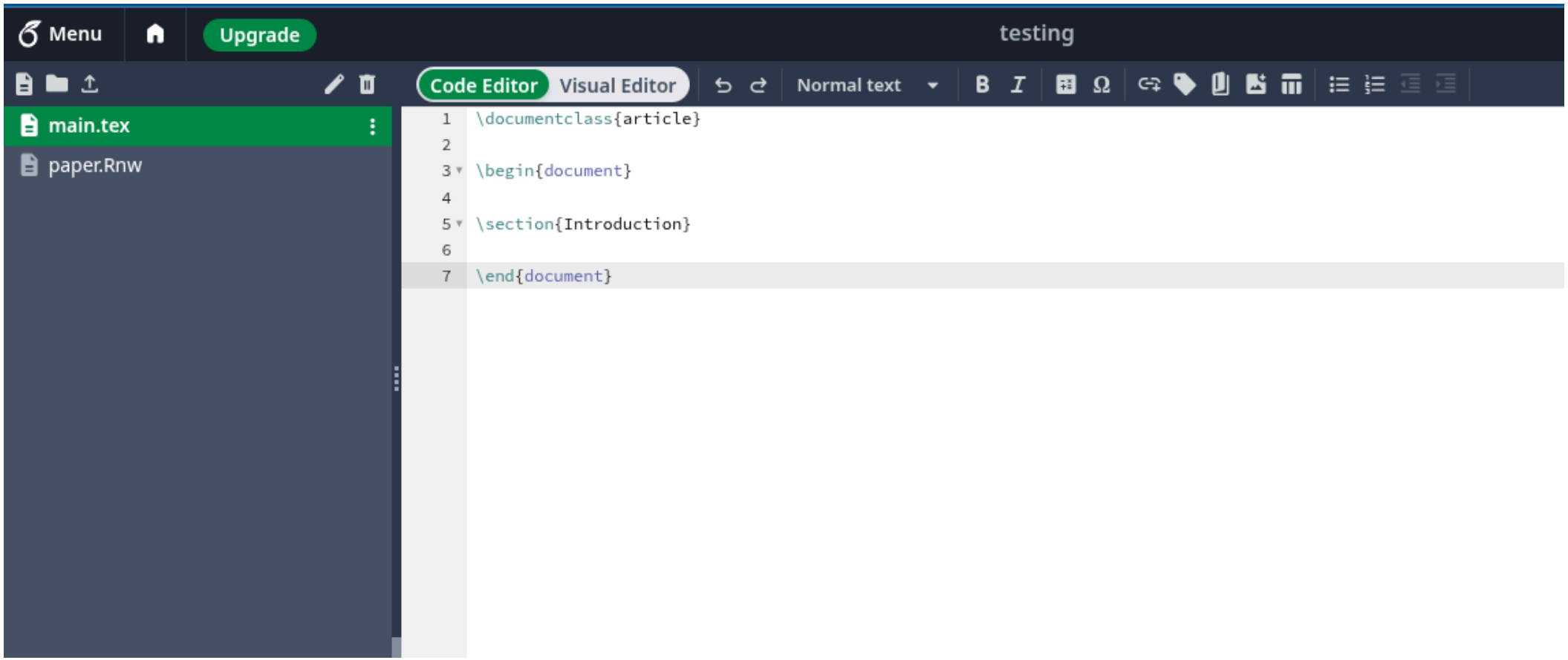
The `trackdown` package can be used to collaborate on `Rmd` or `qmd` documents using Google Docs.



https://github.com/ClaudioZandonella/trackdown

Collaborating! Overleaf

With Overleaf you can collaborate on `.tex` documents but also `.Rnw` documents. No `Rmd` or `qmd` unfortunately. See an example document.



> Git and Github

Git and Github

- The basic idea is to track changes within a folder, assign a `message` and eventually a `tag` to a specific version obtaining a version hystory. The version history is completely navigable, you can go back to a previous version of the code.
- The are advanced features like `branches` for creating an independent version of the project to test new features and then `merge` into the main streamline.
- The entire (local) Git project can be hosted on Github to improve collaboration. Other people or collaborators can `clone` the repository and `push` their changes to the project.

> Veeeery basic Git workflow

Veeeery basic Git workflow

After installing Git, you can start a new repository opening a terminal on a folder and typing `git init`. The folder is now a git project you can notice by the hidden `.git` folder.

```
cd ~/some/folder
git init
```

Then you can `add` files to the staging area. Basically these files are ready to be `committed` i.e. “written” in the Git history.

```
git add file1.txt
# git add . # add everyting
```

Finally you can commit the modified version of the file using `git commit -m message`

```
git commit -m "my first amazing commit"
```

you can see the Git hystory with all your commits:

```
git log
```

Github

Imagine to put everyting into a server with nice viewing options and advanced features. Github is just an hosting service for your `git` folder.

You can create an empty repository on Github named `git-test`. Now my repo has the path `git@github.com:filippogambarota/git-test.git`.

```
git remote add origin git@github.com:filippogambarota/git-test.git
git push
```

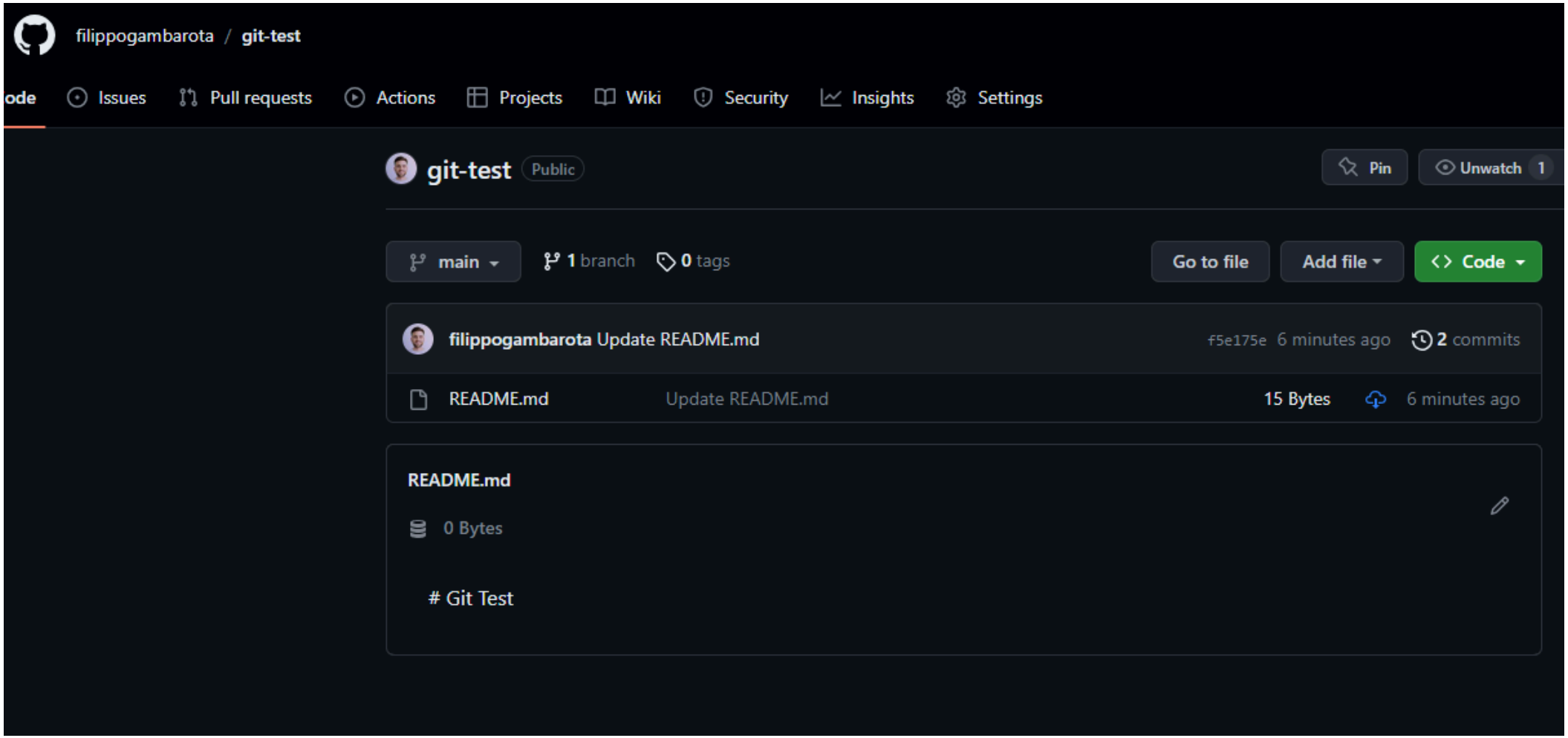
Now our local repository is linked with the remote repository. Every time we do `git push` our local commits will be uploaded.

If you worked on the repository from another machine or a colleague add some changes, you can do `git pull` and your local machine will be updated.

The repository `git-test` is online and can be seen here filippogambarota/git-test.

Github

An now let’s see on Github the result:



More about Git and Github

There are a lot of resources online:

- The Open Science Manual - Zandonella and Massidda - Git and Github chapters.
- https://agripongit.vincenttunru.com/
- https://git-scm.com/docs/gittutorial

> Open Science Framework

Open Science Framework

> OSF is a free, open platform to support your research and enable collaboration.

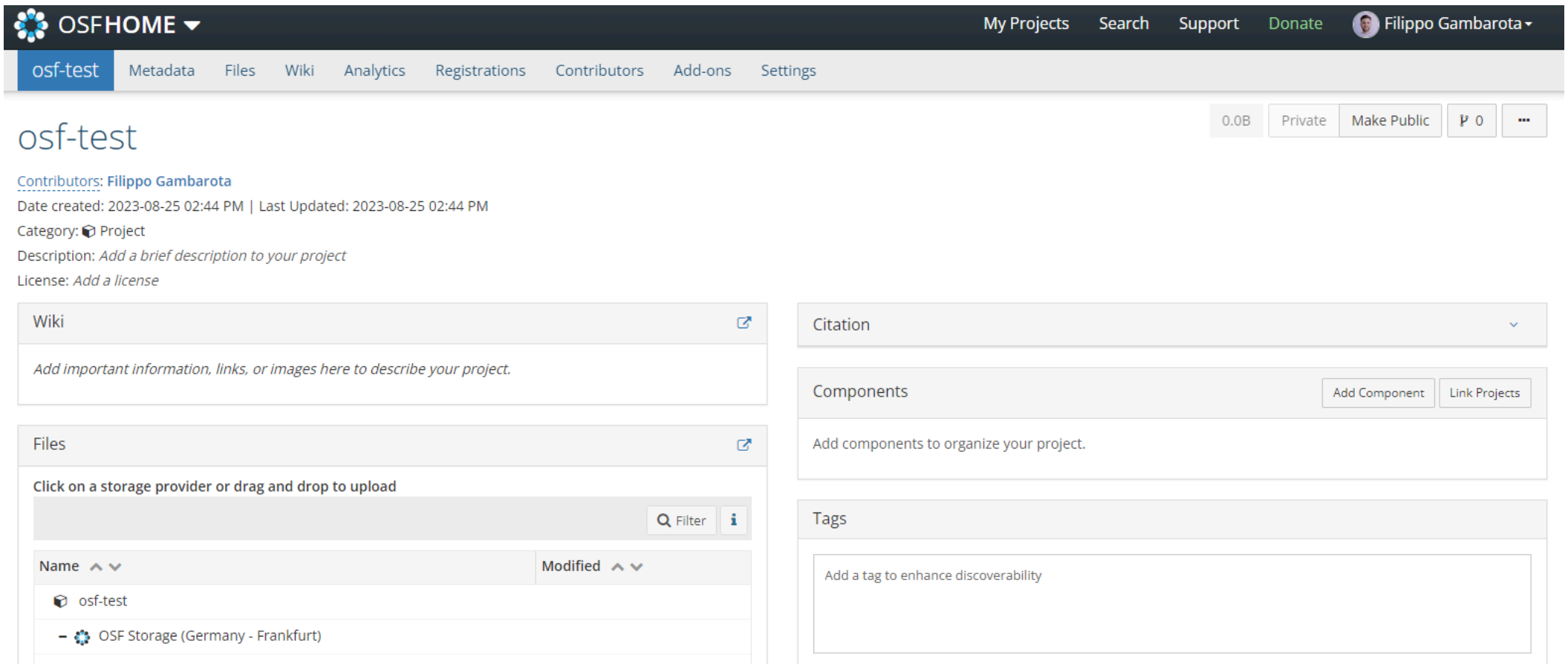
Is a great tool to upload and share materials with others and collaborate on a project. Similarly to Github you can track the changes made to a project.

The great addition is having a DOI thus the project is persistently online and can be cited.

It is now common practice to create a OSF project supporting a research paper and put the link within the paper containing supplementary materials, raw data, scripts etc.

Open Science Framework

It’s very easy to create a new project, then you simply need to add files and share it.



The project can be accessed here (depending on the visibility) https://osf.io/yf9tg/.

Open Science Framework

OSF and Github

An interesting feature is linking a Github repository to OSF. Now all changes made on Github (easier to manage) are mirrored into OSF. You can easily work in Github for the coding part and use OSF to upload other data or information and to assign a DOI to the project.

Preprints

OSF is also linked to a popular service for preprints called PsyArXiv https://psyarxiv.com/ thus you can link a preprint to an OSF project.

More on OSF

- https://help.osf.io/article/342-getting-started-on-the-osf
- https://arca-dpss.github.io/manual-open-science/osf-chapter.html

More on reproducibility

In general, I highly suggest the online book **The Open Science Manual** https://arca-dpss.github.io/manual-open-science/ written by my friend **Claudio Zandonella** and **Davide Massidda** where these and other topics are explained in details:

References

Marwick, B., Boettiger, C., & Mullen, L. (2018). Packaging data analytical work reproducibly using r (and friends). *The American Statistician*, *72*, 80–88. https://doi.org/10.1080/00031305.2017.1375986

Vuorre, M., & Crump, M. J. C. (2021). Sharing and organizing research products as r packages. *Behavior Research Methods*, *53*, 792–802. https://doi.org/10.3758/s13428-020-01436-x