Reproducible Science

Margherita Calderan
Replicability School

June 6, 2025

About me 👋

- Post-doctoral researcher in **Cognitive Psychology**, University of Padova.
- Research: Computational modeling of **cognitive** and **learning processes**, **Bayesian** hypothesis testing.
- PhD in Psychological Science, completed **March 6, 2025**.
- Passionate about reproducible science after struggling with disorganized datasets in my early research!

Our job is hard 🔥

- Running experiments
- Analyzing data
- Managing trainees
- Writing papers
- Responding to reviewers

Reproducibility helps!

- **Organizes** your workflow.
- **Saves time** by documenting steps.
- **Builds trust** in your findings.
- Enables others to **reproduce** and **extend** your work.

Keys to Reproducible Science 🔐

- **Programming**: Use R or Python for transparent analysis.
- **Literate Programming**: Combine code and text with Quarto.
- **Version Control**: Track changes with Git/GitHub.
- **Repositories**: Share data and code via OSF.

So... Is reproducible science even harder?

At first, yes - but then…🧯🔥

- Helps you stay **organized**.
- Makes it **easier to remember** what you did.
- Allows others to **understand**, **reproduce**, and **build on** your work.

> *Learning the tools takes effort but once you do, your workflow becomes smoother, clearer, and more reliable.*

Outline

Data

Code

R projects

Literate Programming

Version Control

Data

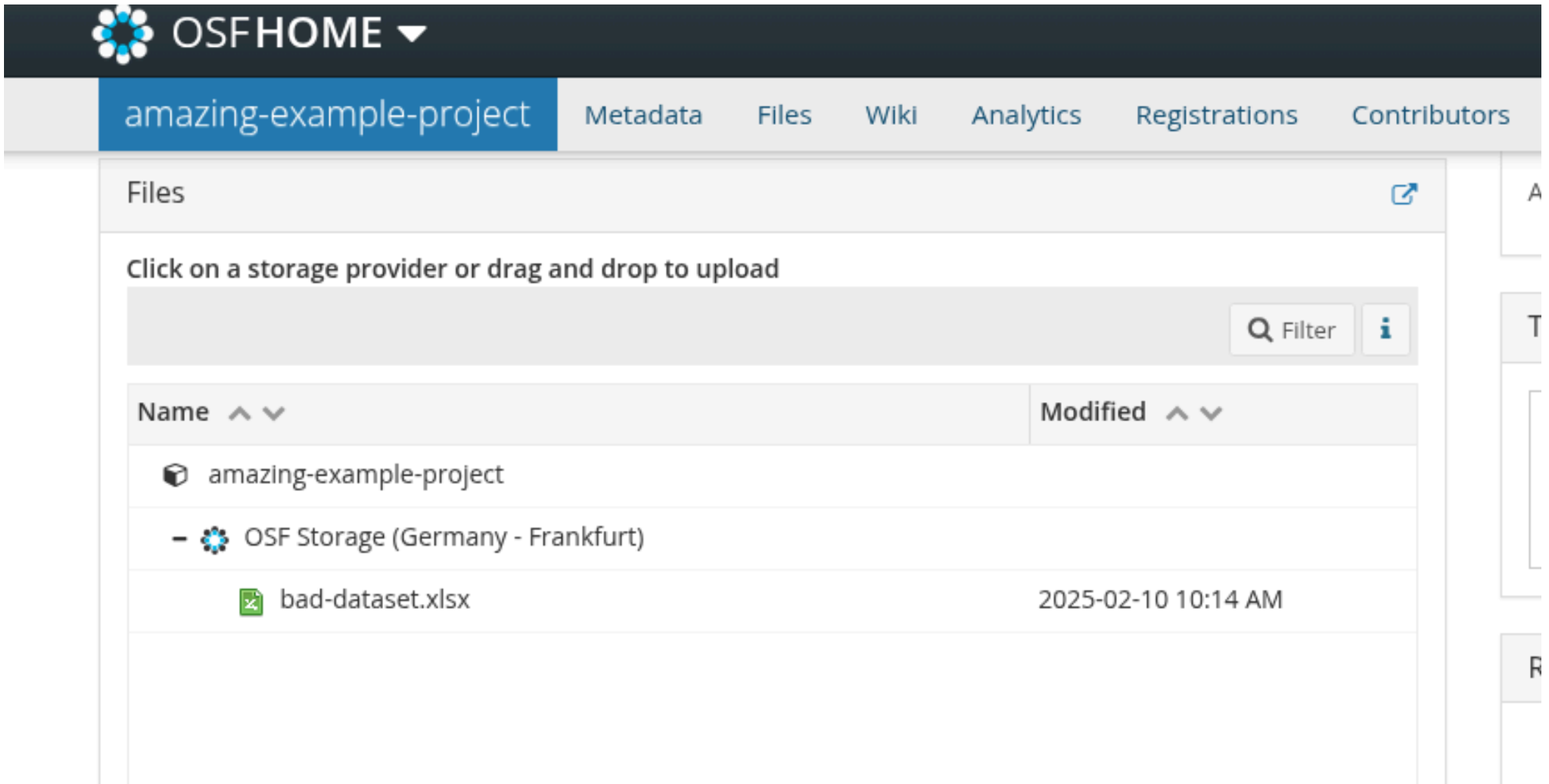
Data Types in Research

Open Science Framework

- Free platform to **organize**, **document**, and **share** research.
- Supports **preregistration**, **archiving**, and **collaboration**.
- Integrates with GitHub, Dropbox, Google Drive.

Bad Data Sharing Example

You find a paper with an OSF link, but…



Bad Data Sharing Example

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

x1	x2	x3	x4	x5	x6	x7
0.3981105	13.912435	a	0	-0.6775811	0.8759740	-0.2051604
-0.1434733	1.093743	c	0	0.7055193	0.2521987	1.8816947
-0.2526000	4.898035	c	0	0.4744651	-0.5628840	0.3245589
-1.2272588	14.717053	b	0	-0.5132792	-1.1368242	-0.1355150
-0.4360417	8.547025	c	1	-0.1736804	-0.7120962	-1.2714320

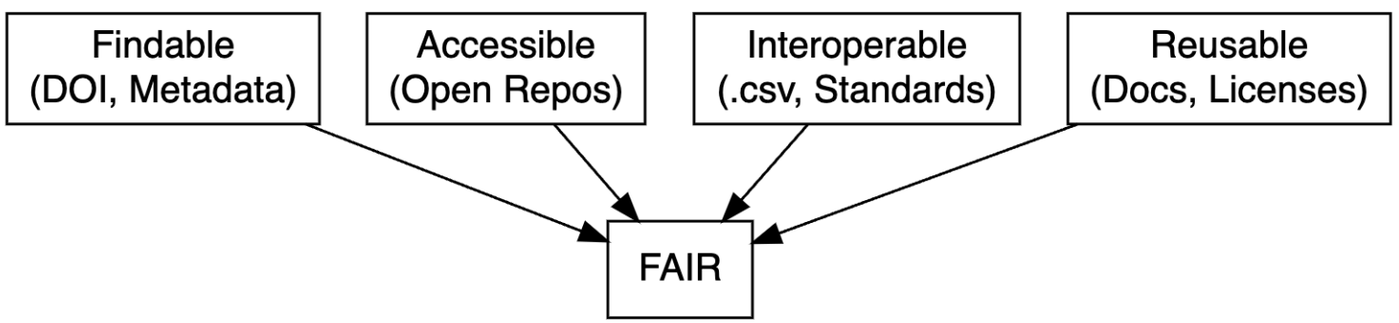
Where is the data-dictionary?
What are 0 and 1?
How missing values are coded?

Good Data Sharing Practices

- Use **plain-text formats** (e.g., `.csv`, `.txt`).
- Include a **data dictionary** with variable descriptions.
- Add a **README** with key details.
- Follow **FAIR principles** (Findable, Accessible, Interoperable, Reusable).

FAIR Data Principles 🔍

- **Findable**: Use metadata and DOIs to make data easy to locate.
- **Accessible**: Ensure data is retrievable via open repositories.
- **Interoperable**: Use standard formats (e.g., `.csv`, `.txt`) for compatibility.
- **Reusable**: Include clear documentation and open licenses.



The Role of a Data Dictionary 📂

- A **data dictionary** defines each variable in your dataset.
- Boosts **transparency** and **collaboration**.
- Saves time for **collaborators** and **future-you**.

datadictionary 📦

```r
library(datadictionary)
df <- data.frame( id = factor(letters[1:5]),
                  anxi = rnorm(5, 0, 1),
                  edu = factor(c("PhD", "BSc", "MSc", "PhD", "BSc")))
df_labels <- list(
  anxi = "Beck Anxiety Inventory, standardized",
  edu = "Last degree obtained"
)
create_dictionary(df, id_var = "id", var_labels = df_labels)
```

	item	label	class	summary	value
1				Rows in dataset	5
2				Columns in dataset	3
3	id	Unique identifier		unique values	5
4				missing	0
5	anxi	Beck Anxiety Inventory, standardized	numeric	mean	0
6				median	0
7				min	-1.85
8				max	0.51
9				missing	0
10	edu	Last degree obtained	factor	BSc (1)	2
11				MSc (2)	1
12				PhD (3)	2
13				missing	0

Best Practices for Data Dictionaries

- Provide context (e.g., how variables were measured).
- Define codes and abbreviations (e.g., “NA” for missing).
- Note special values (e.g., -999 for invalid data).

Variable	Description	Type	Values	Missing	Invalid
id	Participant ID	Factor	a-e	NA	-999
anxi	Beck Anxiety Inventory, standardized	Numeric	-3 to 3	NA	-999
edu	Last degree obtained	Factor	PhD, BSc, MSc	NA	-999

Data Licensing 🔒

- **Licenses** clarify how others can use your data.
- Common licenses:
  - CC BY: Requires attribution.
  - CC0: No restrictions.
  - ODC-BY: Database-specific attribution.
- Choose based on **discipline norms** and **data sensitivity**.

Code

R and RStudio 💻

- **R**: Free, open-source, with thousands of packages for analysis.
- **RStudio**: Intuitive interface for coding, plotting, and debugging.
- Vibrant **community** for support and resources.
- Scripting ensures **transparent, reproducible** workflows.

Writing Better Code 

- **Organize scripts**: Load packages and data upfront.
- **Comment clearly**: Document your logic for clarity.
- **Name descriptively**: Use `snake_case` or `camelCase` for readability.

```r
x1 = rep(c("Psy","Med","Bio"), 4)  # What does 'x' mean?
DepUni = x1       # CamelCase
dep_uni = x1      # snake_case
```

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(...)
```

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.

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.

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.

```
1 head(mtcars, n = 3)
```

```
               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
```

```
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)

head(results, n = 3)
```

```
      means medians mins  maxs  col
1  20.09062    19.2 10.4  33.9  mpg
2   6.18750     6.0  4.0   8.0  cyl
3 230.72188   196.3 71.1 472.0 disp
```

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

```
1 results <- do.call(rbind, dfs)
2 head(results, n = 6)
```

```
       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
```

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.

```
1 results <- lapply(mtcars, summ)
2 results <- do.call(rbind, results)
3 head(results, n = 6)
```

```
          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
```

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] <- i^2
}
```

Very fast:

```
res <- rep(0, 1000)
for(i in 1:length(res)){
  # do something
  res[i] <- i^2
}
```

microbenchmark 📦

```r
library(microbenchmark)

microbenchmark(
  grow_in_loop = {
    res <- c()
    for (i in 1:10000) {
      res[i] <- i^2
    }
  },
  preallocated = {
    res <- rep(0, 10000)
    for (i in 1:length(res)) {
      res[i] <- i^2
    }
  }, times = 100)
```

```
Unit: microseconds
         expr      min        lq      mean    median        uq      max neval
 grow_in_loop 1183.957 1273.7675 1498.0432 1332.1925 1404.7215 7168.932   100
 preallocated  658.624  693.0025  736.0115  708.2135  727.8935 2417.647   100
 cld
  a
   b
```

With *apply you can do crazy stuff!

```
1 funs <- list(mean = mean, sd = sd, min = min, max = max, median = median)
2 sapply(funs, function(f) apply(mtcars, MARGIN = 2, function(x) f(x)))
```

```
            mean           sd    min     max  median
mpg    20.090625    6.0269481 10.400  33.900  19.200
cyl     6.187500    1.7859216  4.000   8.000   6.000
disp  230.721875  123.9386938 71.100 472.000 196.300
hp    146.687500   68.5628685 52.000 335.000 123.000
drat    3.596563    0.5346787  2.760   4.930   3.695
wt      3.217250    0.9784574  1.513   5.424   3.325
qsec   17.848750    1.7869432 14.500  22.900  17.710
vs      0.437500    0.5040161  0.000   1.000   0.000
am      0.406250    0.4989909  0.000   1.000   0.000
gear    3.687500    0.7378041  3.000   5.000   4.000
carb    2.812500    1.6152000  1.000   8.000   2.000
```

Pure vs. Impure functions

Pure function

Same input, same output, no side effects.

```r
x = 4
add_pure<- function(x) {
  return(x + 1)
}
add_pure(2)
```

```
[1] 3
```

```r
print(x)
```

```
[1] 4
```

Impure function

Modifies external variables.

```r
add_impure <- function(x) {
  x <<- x + 1
}
add_impure(x)
print(x)
```

```
[1] 5
```

Test your functions - `fuzzr` 📦

Define your function…

```r
my_mean <- function(x, na.rm = TRUE) {
  if (!is.numeric(x)) stop("`x` must be numeri
  if (length(x) == 0) return(NA)
  if (na.rm) x <- x[!is.na(x)]
  if (length(x) == 0) return(NA)
  sum(x) / length(x)
}
```

Define properties that should always hold true…

```r
property_mean_correct <- function(x) {
  x_no_na <- x[!is.na(x)] #remove NA
  if (length(x_no_na) == 0) return(TRUE)
  abs(my_mean(x) - mean(x, na.rm = TRUE)) < 1e
}
```

Test the function across many randomly generated inputs…

```r
# Property-based testing with 'fuzzr'
library(fuzzr)
test = fuzz_function(fun = property_mean_correct,
                     arg_name = "x",
                     tests = test_dbl())
lapply(test, function(res) res$test_result$value)
```

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

[[2]]
[1] TRUE

[[3]]
[1] TRUE

[[4]]
[1] TRUE

[[5]]
[1] TRUE

[[6]]
[1] TRUE
```

```r
fuzzr::test_dbl()
```

```
$dbl_empty
numeric(0)

$dbl_single
[1] 1.5

$dbl_mutliple
[1] 1.5 2.5 3.5

$dbl_with_na
[1] 1.5 2.5  NA

$dbl_single_na
[1] NA

$dbl_all_na
[1] NA NA NA
```

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.

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:

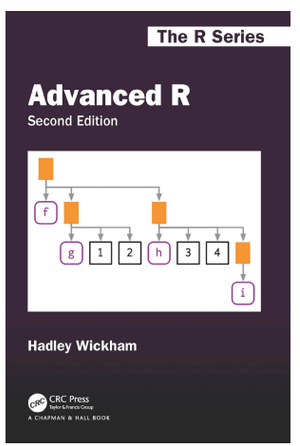
```r
myfun <- function(x) {
  # something
}
```

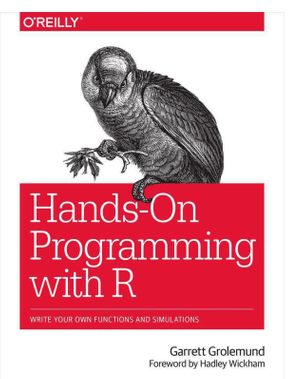
Then you can load the function using `source("R/utils.R)` at the beginning of `analysis.R`:

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

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)





Organize your project

R Projects

R Projects are a feature implemented in RStudio to organize a working directory.

- They automatically set the working directory
- They allow the use of ***relative paths*** instead of ***absolute paths***
- They provide quick access to a specific project

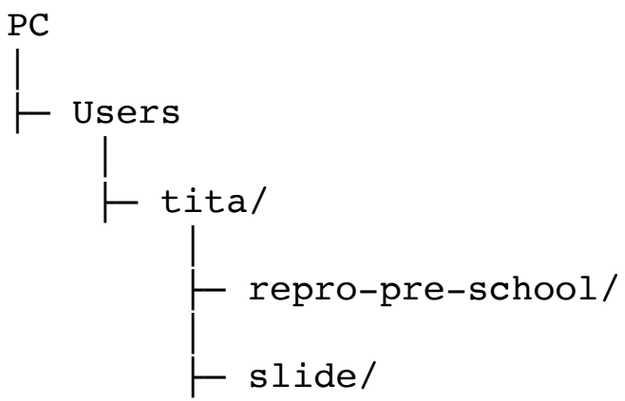
Working Directory

Where is R currently working?

```
1 getwd()
```

```
[1] "/Users/tita/repro-pre-school/slide"
```

Absolute Path



We are working inside the folder **slide**.

Relative Path (to the working directory)

Since we are working inside this folder, if we want to load a file located in this folder we can simply write the file name in quotes.

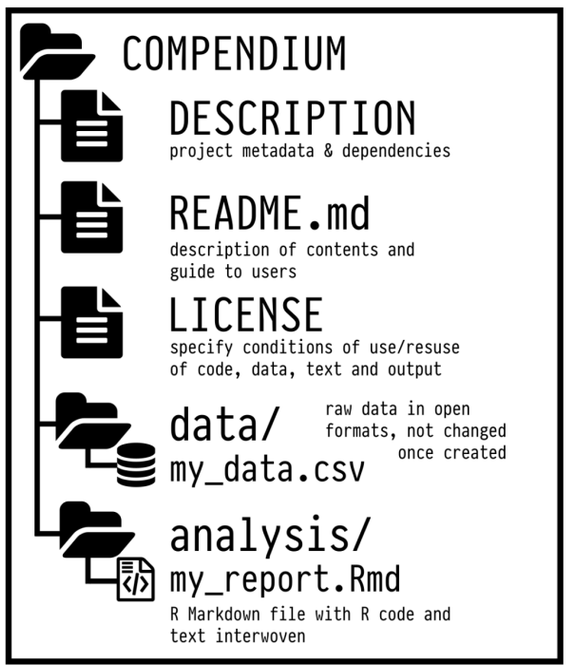
Absolute path: Users/tita/repro-pre-school/slide/df.csv
Relative path: df.csv

```
1 read.csv('df.csv')
```

```
  X id       anxi        dep edu
1 1  a  0.3127232  1.0862664 PhD
2 2  b  0.7644206  0.5783888 BSc
3 3  c  0.6950641 -0.7888920 MSc
4 4  d -0.2936099 -0.8288001 PhD
5 5  e -0.3144039  0.2714690 BSc
```

Project Organization with `rrtools` 📦

> *… the goal is to provide a standard and easily recognisable way for organising the digital materials of a project to enable others to inspect, reproduce, and extend the research… (Marwick et al., 2018)*



Research compendium `rrtools` 📦

- Organize its files according to the prevailing conventions.
- Maintain a clear separation of data, method, and output, while unambiguously expressing the relationship between those three (original data is untouched!).
- Specify the computational environment that was used for the original analysis

Click here for Tutorial

`rrtools::create_compendium()` builds the basic structure for a research compendium.

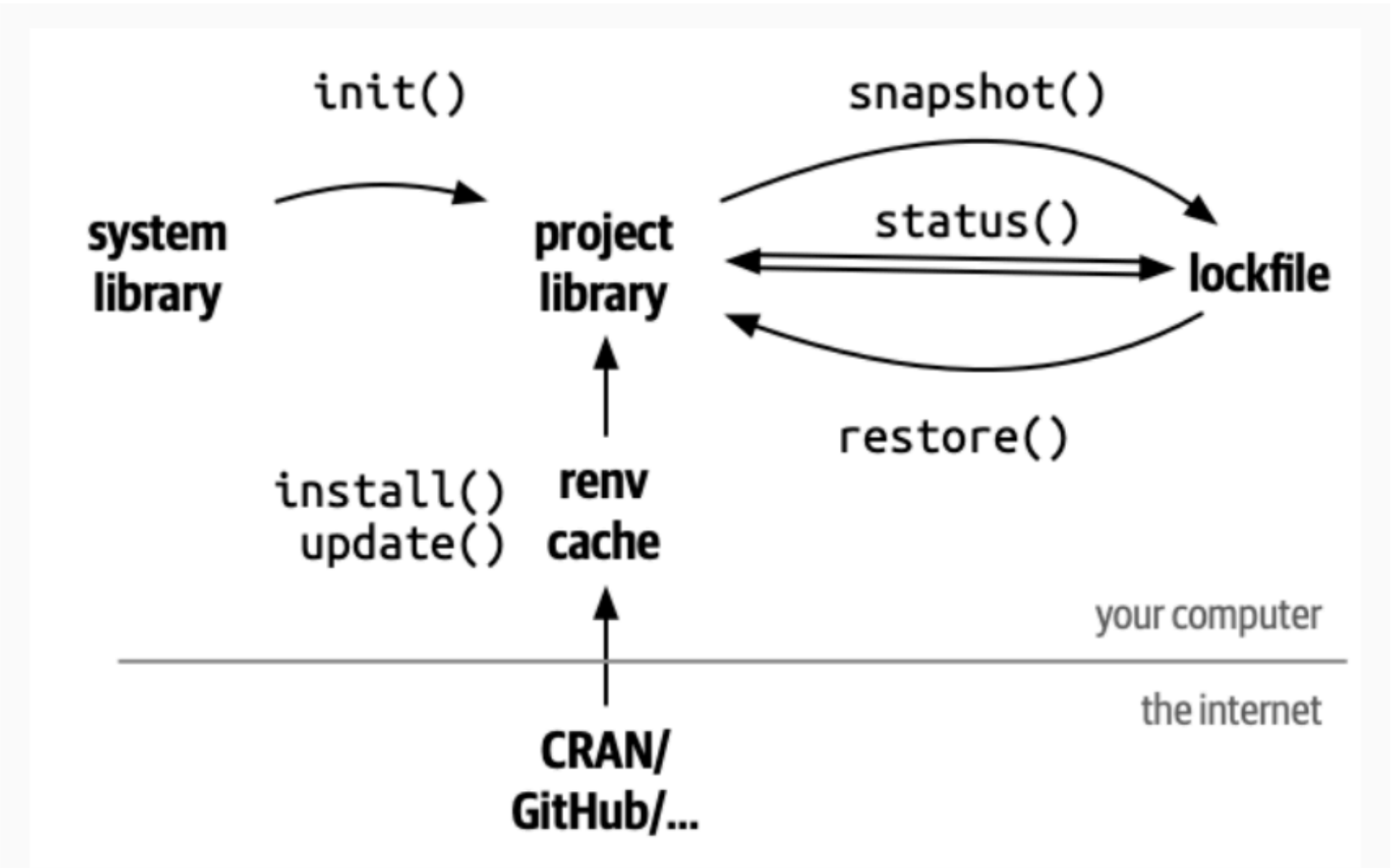
- Storage for general metadata (e.g., citation details)
- Dependency management via **DESCRIPTION** file
- Function storage and documentation in R/ folder

These features enable managing, installing, and sharing project-related functionality.

example

renv 📦

`renv` helps you create reproducible environments for your R projects.



Project specific library

```
`install.packages('microbenchmark')`

The following package(s) will be installed:

\- microbenchmark \[1.5.0\]

These packages will be installed into

\~/repro-pre-school/example-renv/renv/library/macos/R-4.4/aarch64-apple-darwin20
```

example

Research `rrtools` + `renv` 💣

- `rrtools`: Organizes your project into a **reproducible compendium** with clear folders.
- `renv`: Locks **R package versions** for consistent environments.
- Together, they ensure **structure** and **reproducibility** across teams and time.
- Run `rrtools::create_compendium()` to start, then `renv::init()` to lock dependencies.

Docker

- Packages your project's **software**, **dependencies**, and **system settings** into a *container*.
- Ensures **consistency** across different computers or servers.
- Ideal for **sharing** complex analyses with others.

Documenting Your Environment ℹ️

- `sessionInfo()`: Captures your **R version**, **packages**, and **platform** in one command.
- Easy way to **document** and **share** your environment.
- Perfect for **beginners** or when avoiding `renv` or Docker.

```
1 sessionInfo()
```

```
R version 4.4.2 (2024-10-31)
Platform: aarch64-apple-darwin20
Running under: macOS Sequoia 15.5

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRblas.0.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRlapack.dylib;  LAPACK
version 3.12.0

locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

time zone: Europe/Rome
tzcode source: internal
```

```
attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base
```

Literate Programming

Literate Programming

> 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.

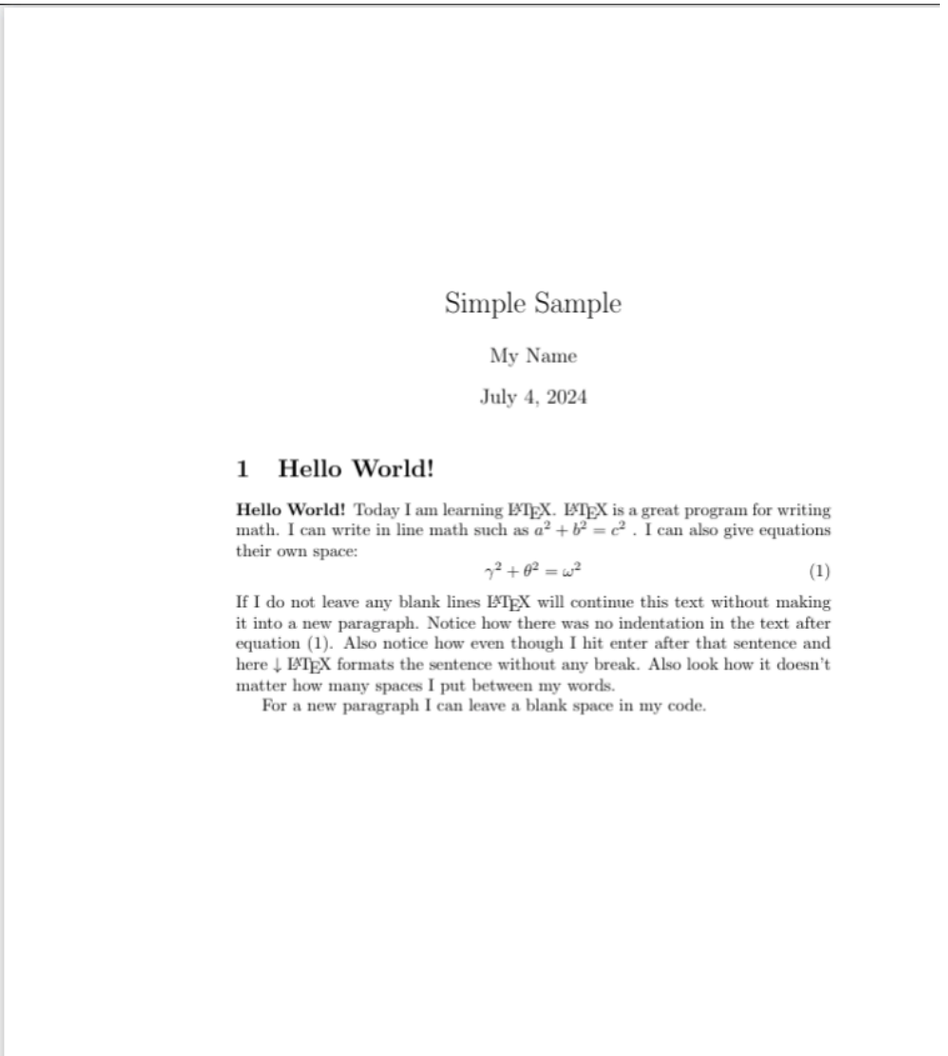
Literate Programming, the markup language

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

```
% 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 stampa. Lorem Ips

<h2>My Second Heading</h2>

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

Lorem Ipsum è considerato il testo segnaposto standard sin dal sedicesimo secolo, quando un anonimo

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

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

Markdown

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
```

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)

Also the source code can be used to take notes and read.

```
## My Section
- Write **bold** text.
- Include a [link](https://quarto.org)
- Run code: `r mean(mtcars$mpg)`.
```

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.

Markdown Basics

Overview

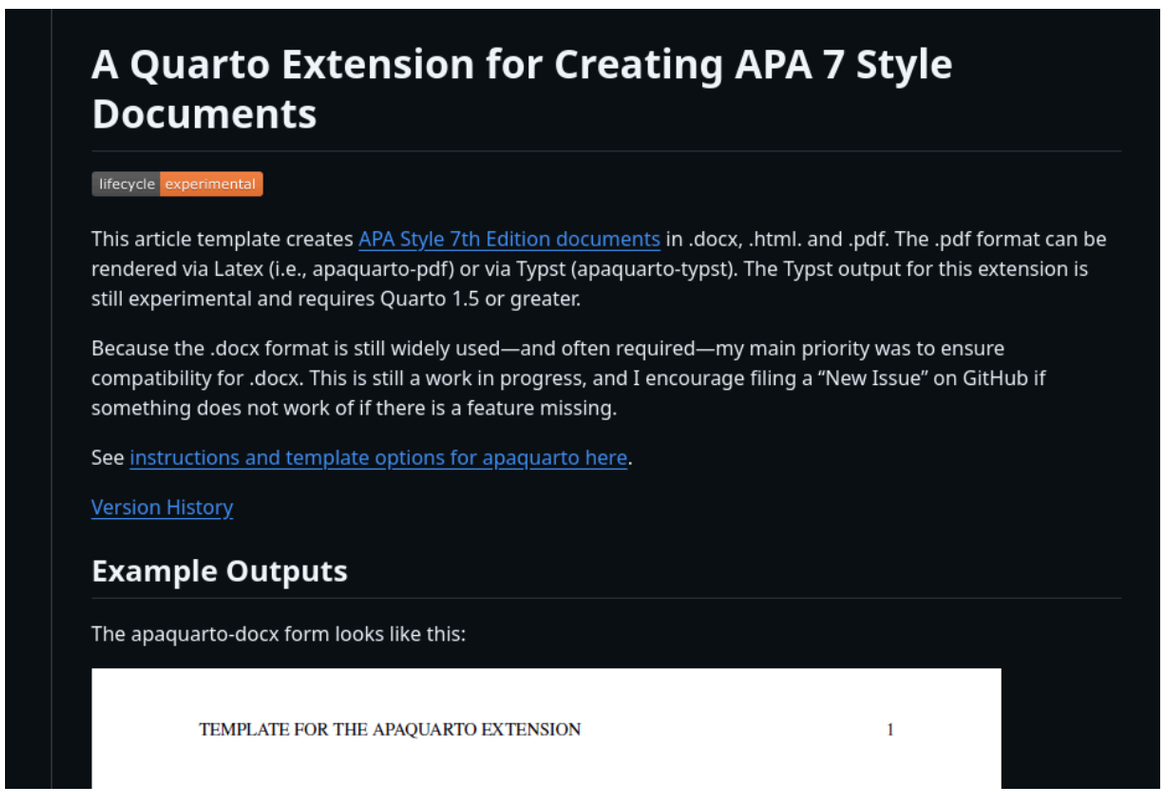
Quarto is based on Pandoc and uses its variation of markdown as its underlying document syntax. Pandoc markdown is an extended and slightly revised version of John Gruber's Markdown syntax.

Markdown is a plain text format that is designed to be easy to write, and, even more importantly, easy to read:

> A Markdown-formatted document should be publishable as-is, as plain text, without looking like it's been marked up with tags or formatting instructions. – John Gruber

Writing Papers - APA quarto

APA Quarto is a Quarto extension that makes it easy to write documents in APA 7th edition style — with automatic formatting for title pages, headings, citations, references, tables, and figures.

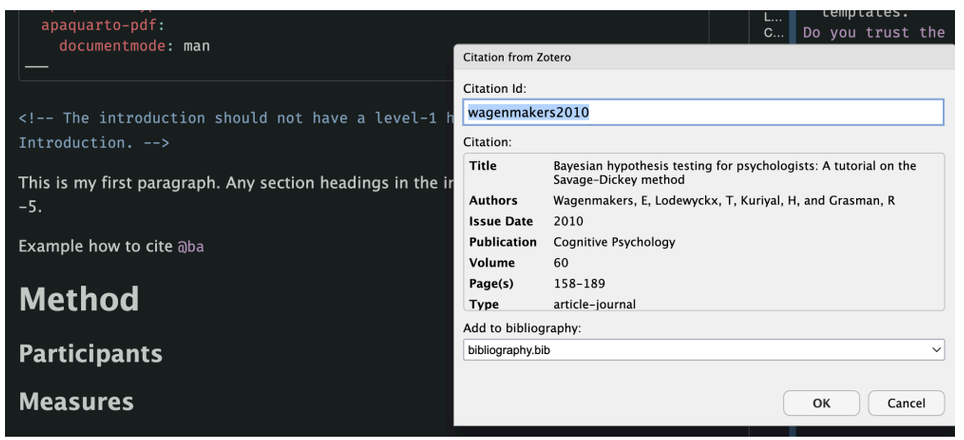


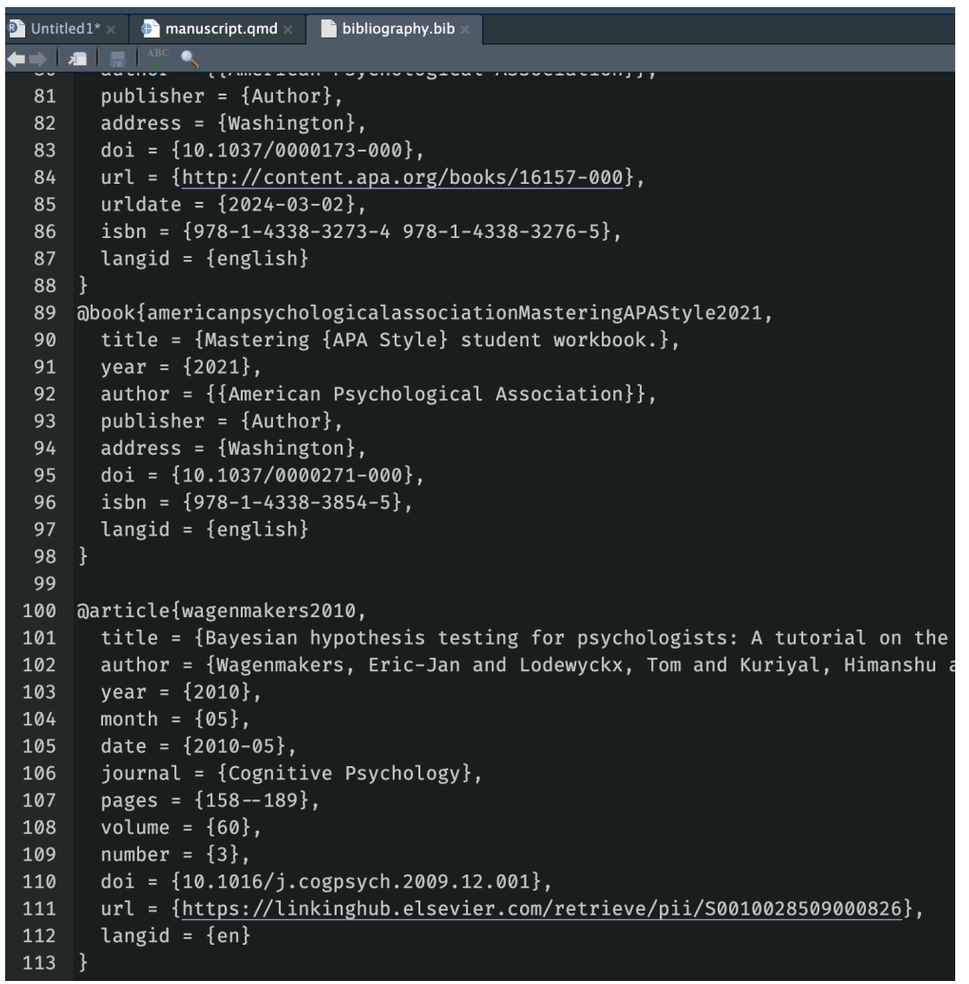
Let’s see an example...

.qmd source

docx output

Quarto + Zotero





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/

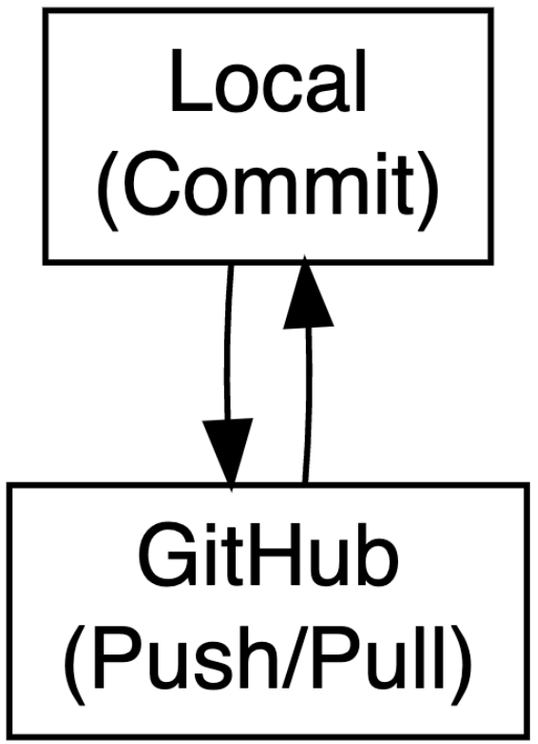
Version Control

Version Control with Git & GitHub 🔀

- Git tracks changes in your code and text files.
- GitHub makes it easy to share, collaborate, and back up your project.
- You can revert to previous versions at any point.
- Encourages clean, documented workflows.

> 💡 Use meaningful commit messages and structure your repositories with clarity.

GitHub in Practice




```
git init
git add .
git commit -m "Initial commit"
git push origin main
```

GitHub Best Practices ✅

- Use a clear folder structure (e.g., `data/`, `scripts/`, `results/`). `rrtools` - compendium
- Write a `README.md` to explain your project's goals and usage.
- Include a `LICENSE` file to define how others can reuse your work.
- Add a `.gitignore` to avoid uploading large or irrelevant files (e.g., `.Rhistory`, `.DS_Store`).

Integrated Workflow Example 🛠️

1. Develop your analysis using **R and Quarto**.
2. Track code and scripts using **Git**.
3. Host your code on **GitHub** (public or private).
4. Upload your data and materials to **OSF**, including a data dictionary.
5. Link your GitHub repository to your OSF project.
6. Use `renv` for reproducible R environments.
7. Share the OSF project and cite it in your paper.

Start small. It's easy. 🐾

> "Let me tell you: anything you do — providing the raw data, posting any small scripts, detailing the versions of programs you used together with their parameters — will be tremendously welcome to anyone trying to validate or build off your paper. This includes you yourself, in 2 years." *C. Titus Brown*

References

Marwick, B., Boettiger, C., & Mullen, L. (2018). Packaging data analytical work reproducibly using r (and friends). *The American Statistician*, *72*(1), 80–88. https://doi.org/10.1080/00031305.2017.1375986