Automating Reproducibility

Challenges and what it takes to meet them

Aaron Peikert[1,2],
Andreas M. Brandmaier[1,2], Maximilian S. Ernst[1]

[1]Center for Lifespan Psychology—Max Planck Institute for Human Development, Berlin, Germany
[2]Max Planck UCL Centre for Computational Psychiatry and Ageing Research, Berlin, Germany and London, UK

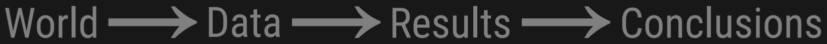
12.05.2022 Max Planck Institute for Evolutionary Biology

Slides: `https://github.com/aaronpeikert/repro-talk`

Why do we trust science?

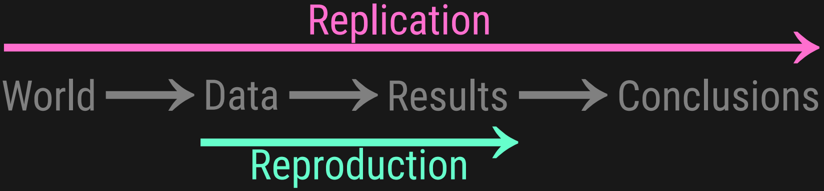
Scientific claims should not be credible because of their originators’ authority but by the transparency and replicability of their supporting evidence.

Reproduction ≠ Replication



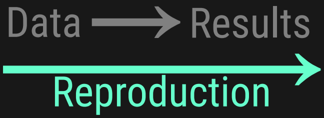
World → Data → Results → Conclusions

Reproduction ≠ Replication



Replication can not be automated, but reproducibility can and should be automated.

Reproduction ≠ Replication



Data → Results

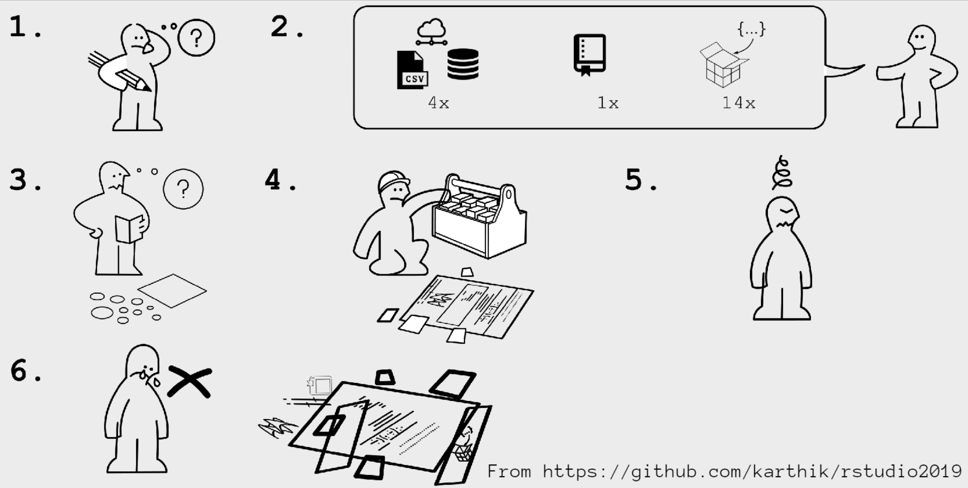
Reproduction

“Insanity is doing the same thing over and over again and expecting different results.”

– Albert Einstein (disputed)

> “Insanity is doing the same thing over and over again and expecting different results.”
>
> – Albert Einstein (disputed)

As it turns out, doing the
same thing
is pretty complicated.



From https://github.com/karthik/rstudio2019

Why do I mistrust supplementary code?

Four different problems are all to common:

1. Multiple inconsistent versions of code and data

Why do I mistrust supplementary code?

Four different problems are all to common:

1. Multiple inconsistent versions of code and data
2. Copy-and-paste errors

Why do I mistrust supplementary code?

Four different problems are all to common:

1. Multiple inconsistent versions of code and data
2. Copy-and-paste errors
3. Ambiguous order of code execution

Why do I mistrust supplementary code?

Four different problems are all to common:

1. Multiple inconsistent versions of code and data
2. Copy-and-paste errors
3. Ambiguous order of code execution
4. Broken dependencies

Lessons from software engeniering

Four solutions:

1. Version control
2. Dynamic document creation
3. Dependency tracking
4. Software management

Peikert, A., & Brandmaier, A. M. (2021). A Reproducible Data Analysis Workflow. *Quantitative and Computational Methods in Behavioral Sciences, 1, Article e3763.* https://doi.org/10.5964/qcmb.3763

Specify Everything

The relations between
code, data, results and their environment
need to be unambiguously specified.

Why should I care?

Productivity:

- reuse
- easier collaboration
- avoid trouble (during review, questions after publication, etc.)

Why should I care?

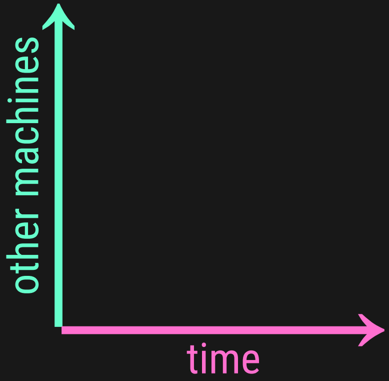
Good scientific practice:

- reproducibility is a precondition for replication
- increases transparency and (longterm) accessibility

Lessons from software engeniering

Four solutions:

1. Version control
2. Dynamic document creation
3. Dependency tracking
4. Software management



Tools for R Users

In the R universe and beyond, the most flexible tools are:

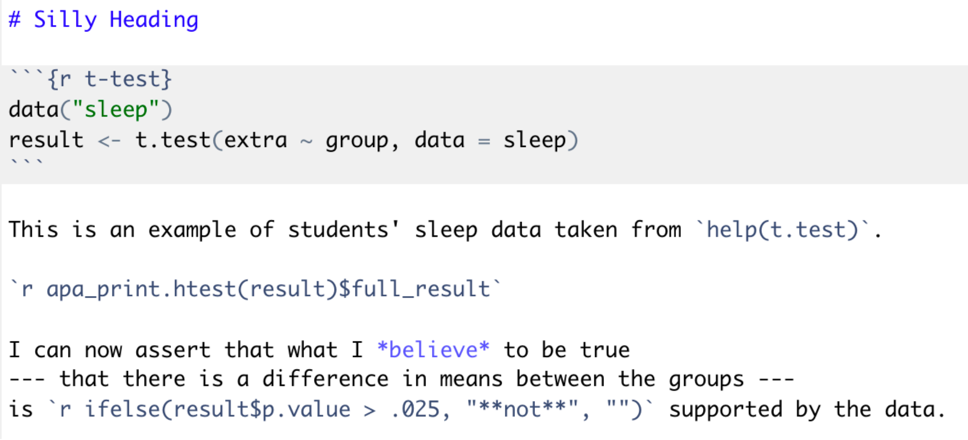
- Dynamic document creation = RMarkdown*
- Version control = Git**
- Dependency tracking = Make**
- Software management = Docker**

* RMarkdown supports more then 40 languages e.g.:
Python, Julia, SAS, Scala & Octave
** Language agnostic

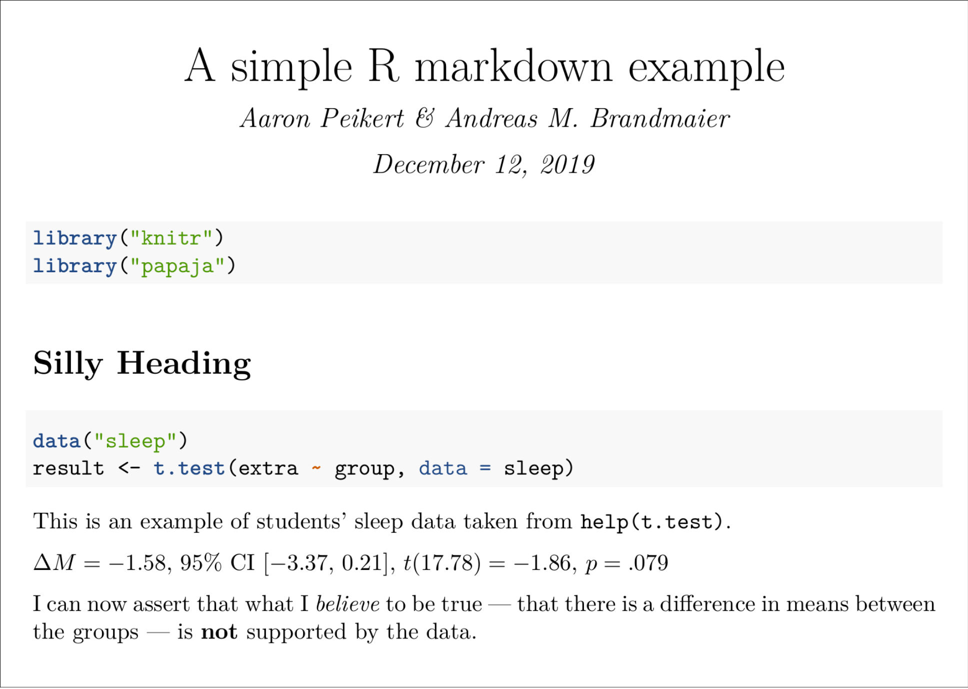
RMarkdown—Literate Programming

Text and code are mixed
in a single source document
that can be dynamically compiled
into various representations:

- (APA conformable) manuscripts
- presentations
- websites
- books
- posters
- CV



~~~
# Silly Heading

```{r t-test}
data("sleep")
result <- t.test(extra ~ group, data = sleep)
```

This is an example of students' sleep data taken from `help(t.test)`.

`r apa_print.htest(result)$full_result`

I can now assert that what I *believe* to be true
--- that there is a difference in means between the groups ---
is `r ifelse(result$p.value > .025, "**not**", "")` supported by the data.
~~~



A simple R markdown example

Aaron Peikert & Andreas M. Brandmaier

December 12, 2019

```r
library("knitr")
library("papaja")
```

Silly Heading

```r
data("sleep")
result <- t.test(extra ~ group, data = sleep)
```

This is an example of students' sleep data taken from `help(t.test)`.

$\Delta M = -1.58$, 95% CI $[-3.37, 0.21]$, $t(17.78) = -1.86$, $p = .079$

I can now assert that what I *believe* to be true — that there is a difference in means between the groups — is **not** supported by the data.

Git/GitHub—Version Control

Version control is a system that records changes to a set of files over time so that you can recall specific versions later.

It guarantees that code and data are exactly the same version as used for publication.

Make—Dependency Tracking

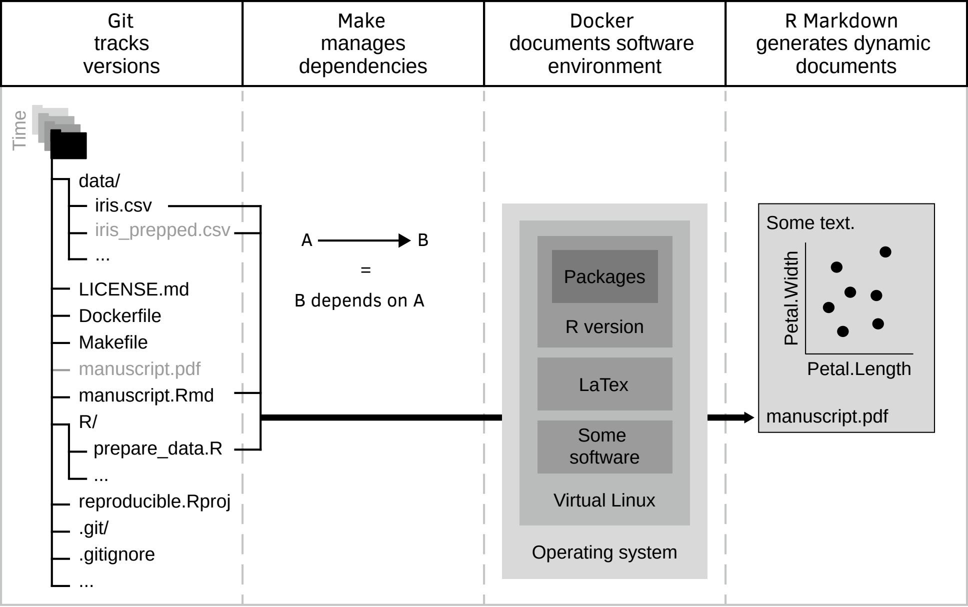
Make is a “recipe” language that describes how files depend on each other and how to resolve these dependencies.

```
spaghetti_arrabiata.pdf: spagetti_arrabiata.Rmd arrabiata_sauce.csv pa
    Rscript -e 'rmarkdown::render("spaghetti_arrabiata.Rmd")'

pasta.csv: cook_pasta.R
    Rscript -e 'source("cook_pasta.R")'

arrabiata_sauce.csv: cook_sauce.R canned_tomatoes.csv
    Rscript -e 'source("cook_sauce.R")'
```

Docker—Containerization

Docker is a lightweight virtual computer.
Dockerfiles are “recipes” that describe what to install on that virtual computer:

```
FROM rocker/verse:3.6.1
ARG BUILD_DATE=2019-11-11
RUN install2.r --error --skipinstalled\
  here lavaan
WORKDIR /home/rstudio
```


Git tracks versions

Make manages dependencies

Docker documents software environment

R Markdown generates dynamic documents

From Peikert & Brandmaier (2019) under CC-BY4.0

Advantages

Unambiguous

Advantages

Unambiguous Standardized

Advantages

Unambiguous Standardized Portable

Advantages

Unambiguous Standardized Portable Automated

Simplifying the tools

These tools require extensive training and need much time to configure correctly.

Simplifying the tools

These tools require extensive training and need much time to configure correctly.

The R package ’repro’ abstracts away the concrete technical implementation:

```
repro:
  packages:
    - ggplot2
    - aaronpeikert/repro@adb5fa569
  scripts:
    - R/clean.R
  data:
    mycars: data/mtcars.csvrepro:
```

The function `repro::automate()` automatically infers Docker- and Makefile.

Disadvantages

- requires complex software infrastructure
- depends on for-profit services
- diverges from the standard manuscript workflow

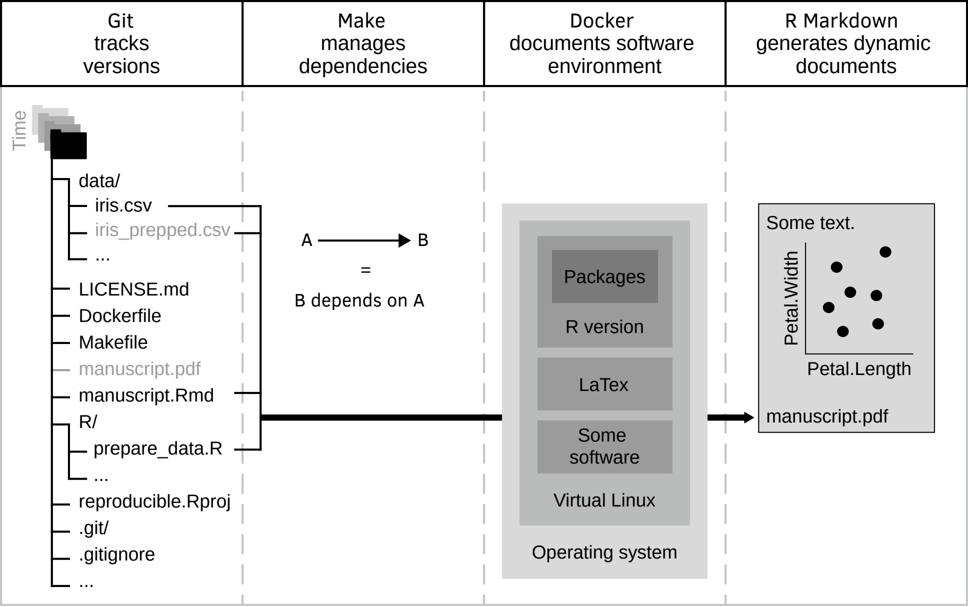
Your melange may vary

Different requirements regarding:
archivation + number of machines

Your melange may vary

Different programming languages:

- Python
- R
- Julia
- Matlab



Git tracks versions

Make manages dependencies

Docker documents software environment

R Markdown generates dynamic documents

Time

- data/
  - iris.csv
  - iris_prepped.csv
  - ...
- LICENSE.md
- Dockerfile
- Makefile
- manuscript.pdf
- manuscript.Rmd
- R/
  - prepare_data.R
  - ...
- reproducible.Rproj
- .git/
- .gitignore
- ...

A → B = B depends on A

Packages

R version

LaTex

Some software

Virtual Linux

Operating system

Some text.

Petal.Width

Petal.Length

manuscript.pdf

From Peikert & Brandmaier (2019) under CC-BY4.0

Focus: Computing infrastructure

Dependency tracking + software management

=

distributed computation

Focus: Computing infrastructure

distributed computation

Dependency tracking enables intelligent task scheduling

Software management guarantees compatible software environment

Focus: Computing infrastructure

distributed computation
on
Cloud Computing infrastructure

Focus: Computing infrastructure

distributed computation
on
High Performance Computing cluster (HPC)

HPC—Container

- repro supports Singularity as a Docker alternative
- developing environment matches HPC environment exactly
- full freedome to use any software, even when not supported by HPC admin

HPC—Dependency tracking

making dependencies between tasks explicit enables:

- intelligent caching
- automatic parralelization
- dynamic job scheduling

Make is well supported by several job schedulers.

Pure R solutions like the packages targets + futureverse offer even more convinience and are compatible with repro

Focus: Modularity

- repro is a modular system

Focus: Modularity

- repro is a modular system
- potential integration of other workflows

Focus: Modularity

- repro is a modular system
- potential integration of other workflows
- “Lego system of reproducibility tools”

Focus: Longterm Archive

All software is bundled into the container, therefore all we need is:

- container software
- storage infrastructure

Focus: Longterm Archive

All software is bundled into the container, therefore all we need is:

- container software
- storage infrastructure

What happens when Docker and co. are not supported anymore?
Containers can be converted into a full system image ensuring support for decades.

References

Slides:
https://github.com/aaronpeikert/repro-talk
Package:
https://github.com/aaronpeikert/repro-thesis
Workflow:
https://doi.org/10.31234/osf.io/8xzqy

Thank you

Questions?