Reproducible Data Analysis

Tristan Mahr

August 20, 2025

What is reproducibility?

I don’t have a precise definition. Here’s a rough one:

Being able to run and rerun a data analysis and get the same results each time

(This is not *replicability* which is about whether some empirical finding is repeated between studies.)

Kind of a loaded question

“data analysis”? (data cleaning?)

“results”? (recruitment and attrition?)

“run” and “rerun”?

“same”?

Reproducibility comes in degrees

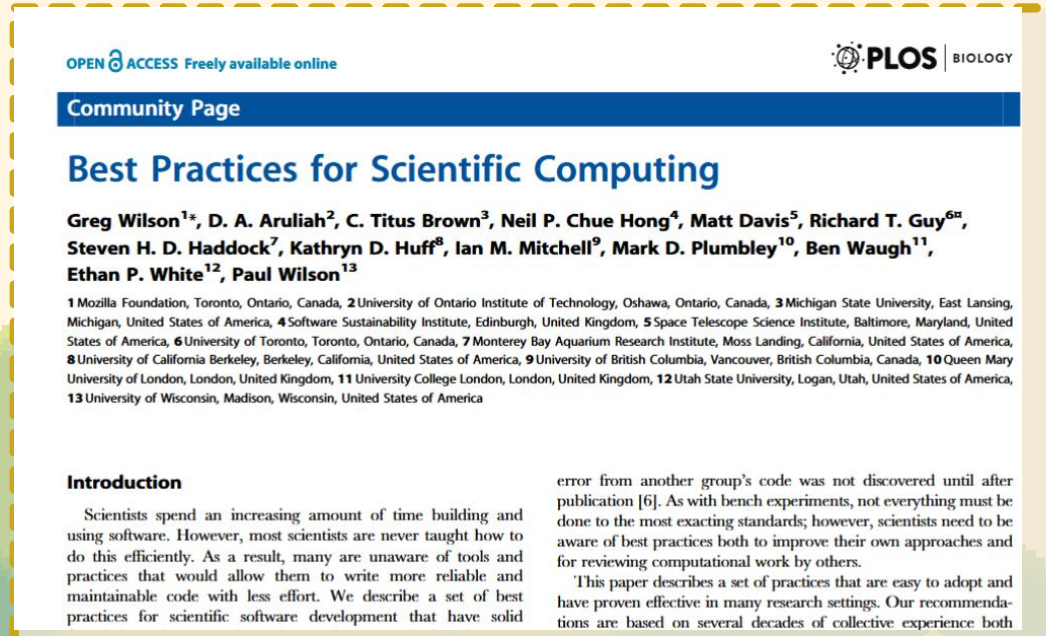
It’s very easy to overthink, overengineer analyses to make them reproducible.

Let’s remember why we want reproducibility.

Reproducibility provides scientific documentation

- Our scientific obligation is to describe our work in enough detail that someone else in principle can carry out the method themselves.
  - “We regressed intelligibility onto age in months using beta regression”
- But the code describes precisely what we did.
  - `betareg::betareg(mean_intel ~ age_months, data)`
  - We are usually the “someone else” anyway!
- And if it “reproduces” (runs with stable results), it is doing something correctly.

Let's start with good enough



OPEN ACCESS Freely available online

PLOS | BIOLOGY

Community Page

Best Practices for Scientific Computing

Greg Wilson[1]*, D. A. Aruliah[2], C. Titus Brown[3], Neil P. Chue Hong[4], Matt Davis[5], Richard T. Guy[6¤], Steven H. D. Haddock[7], Kathryn D. Huff[8], Ian M. Mitchell[9], Mark D. Plumbley[10], Ben Waugh[11], Ethan P. White[12], Paul Wilson[13]

1 Mozilla Foundation, Toronto, Ontario, Canada, **2** University of Ontario Institute of Technology, Oshawa, Ontario, Canada, **3** Michigan State University, East Lansing, Michigan, United States of America, **4** Software Sustainability Institute, Edinburgh, United Kingdom, **5** Space Telescope Science Institute, Baltimore, Maryland, United States of America, **6** University of Toronto, Toronto, Ontario, Canada, **7** Monterey Bay Aquarium Research Institute, Moss Landing, California, United States of America, **8** University of California Berkeley, Berkeley, California, United States of America, **9** University of British Columbia, Vancouver, British Columbia, Canada, **10** Queen Mary University of London, London, United Kingdom, **11** University College London, London, United Kingdom, **12** Utah State University, Logan, Utah, United States of America, **13** University of Wisconsin, Madison, Wisconsin, United States of America

Introduction

Scientists spend an increasing amount of time building and using software. However, most scientists are never taught how to do this efficiently. As a result, many are unaware of tools and practices that would allow them to write more reliable and maintainable code with less effort. We describe a set of best practices for scientific software development that have solid

error from another group's code was not discovered until after publication [6]. As with bench experiments, not everything must be done to the most exacting standards; however, scientists need to be aware of best practices both to improve their own approaches and for reviewing computational work by others.

This paper describes a set of practices that are easy to adopt and have proven effective in many research settings. Our recommendations are based on several decades of collective experience both

PLOS | COMPUTATIONAL BIOLOGY

PERSPECTIVE

Good enough practices in scientific computing

Greg Wilson[1☯]*, Jennifer Bryan[2☯], Karen Cranston[3☯], Justin Kitzes[4☯], Lex Nederbragt[5☯], Tracy K. Teal[6☯]

1 Software Carpentry Foundation, Austin, Texas, United States of America, **2** RStudio and Department of Statistics, University of British Columbia, Vancouver, British Columbia, Canada, **3** Department of Biology, Duke University, Durham, North Carolina, United States of America, **4** Energy and Resources Group, University of California, Berkeley, Berkeley, California, United States of America, **5** Centre for Ecological and Evolutionary Synthesis, University of Oslo, Oslo, Norway, **6** Data Carpentry, Davis, California, United States of America

☯ These authors contributed equally to this work.
* gvwilson@software-carpentry.org

Author summary

Computers are now essential in all branches of science, but most researchers are never taught the equivalent of basic lab skills for research computing. As a result, data can get lost, analyses can take much longer than necessary, and researchers are limited in how

Low hanging fruit

Super Mario World © Nintendo

Make an analysis artifact

Have a simple script that reads in the data, does the work, and makes the things.

Importantly, we can re-run it. And it was can **capture its output.**

My analysis.R

```
library(tidyverse)
library(mgcv)
data <- readr::read_csv("C:/Users/mahr/Documents/td-mwi-means.csv")

# descriptives
data |>
  mutate(age_years = age_months %/% 12) |>
  group_by(age_years) |>
  summarise(
    n_participants = n(),
    intel_mean = mean(mean_intel),
    intel_sd = sd(mean_intel)
  )

# model
model <- mgcv::gam(
  mean_intel ~ s(age_months),
  family = betar(),
  data = data)
summary(model)
plot(model)

sessionInfo()
```

RStudio:
File > Compile Report…

☑ A document

☑ What we did

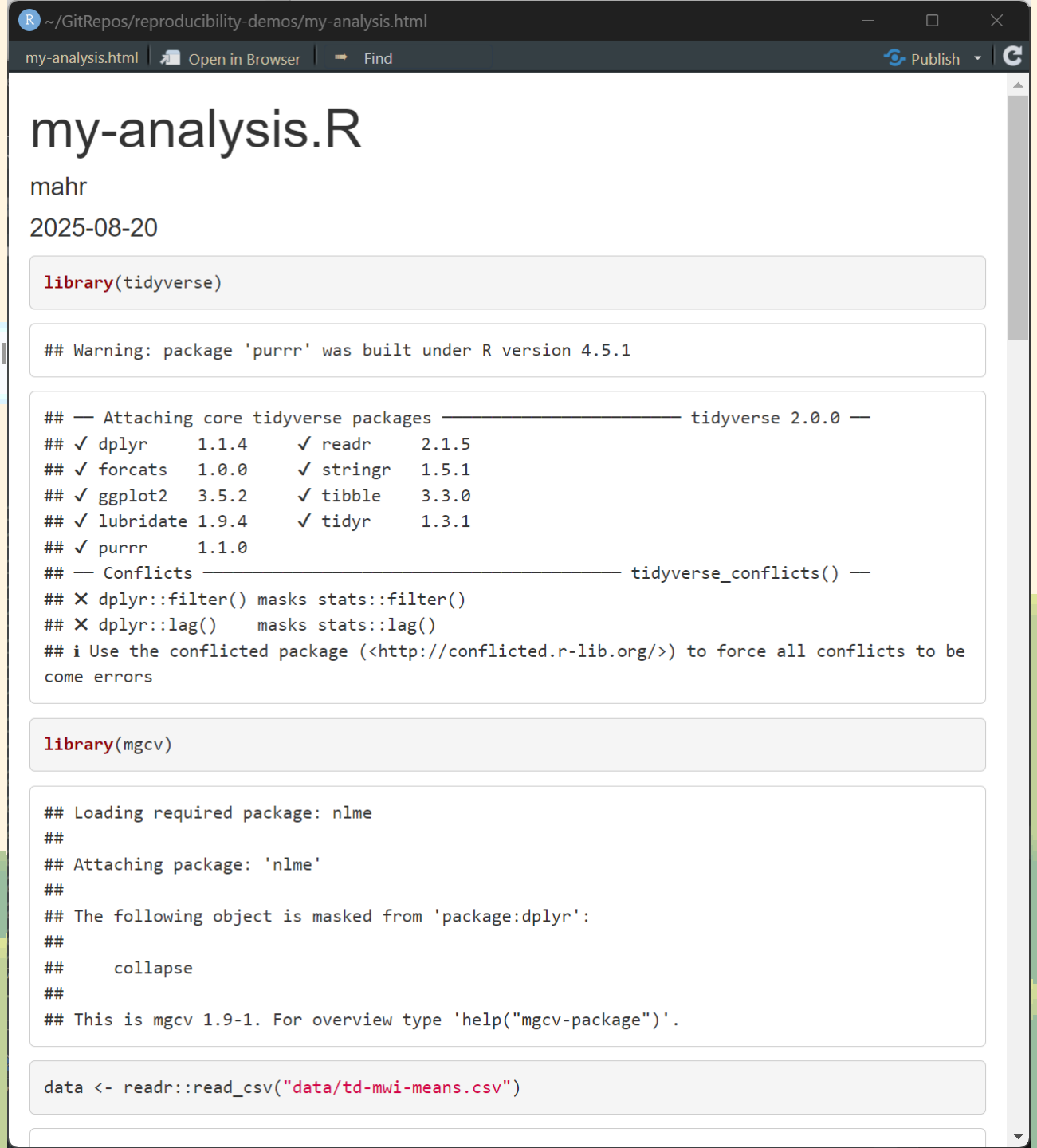
☑ When we did it

☑ What versions we used

But:

☒ It will not work on another computer

❗ A single script may grow to be unwieldy.



Use Projects
Write Documents

Use an Rstudio project or some other way of avoiding hard-coded locations.

Use an RMarkdown/Quarto document to allow longform prose with code

my-analysis.qmd

````
---
title: "Intelligibility GAM"
author: "Tristan Mahr"
format: html
---

```{r}
library(tidyverse)
library(mgcv)

data <- readr::read_csv("data/td-mwi-means.csv")
```

We examined intelligibility from `r nrow(data)` children between
`r min(data$age_months)` and `r max(data$age_months)` months in age.

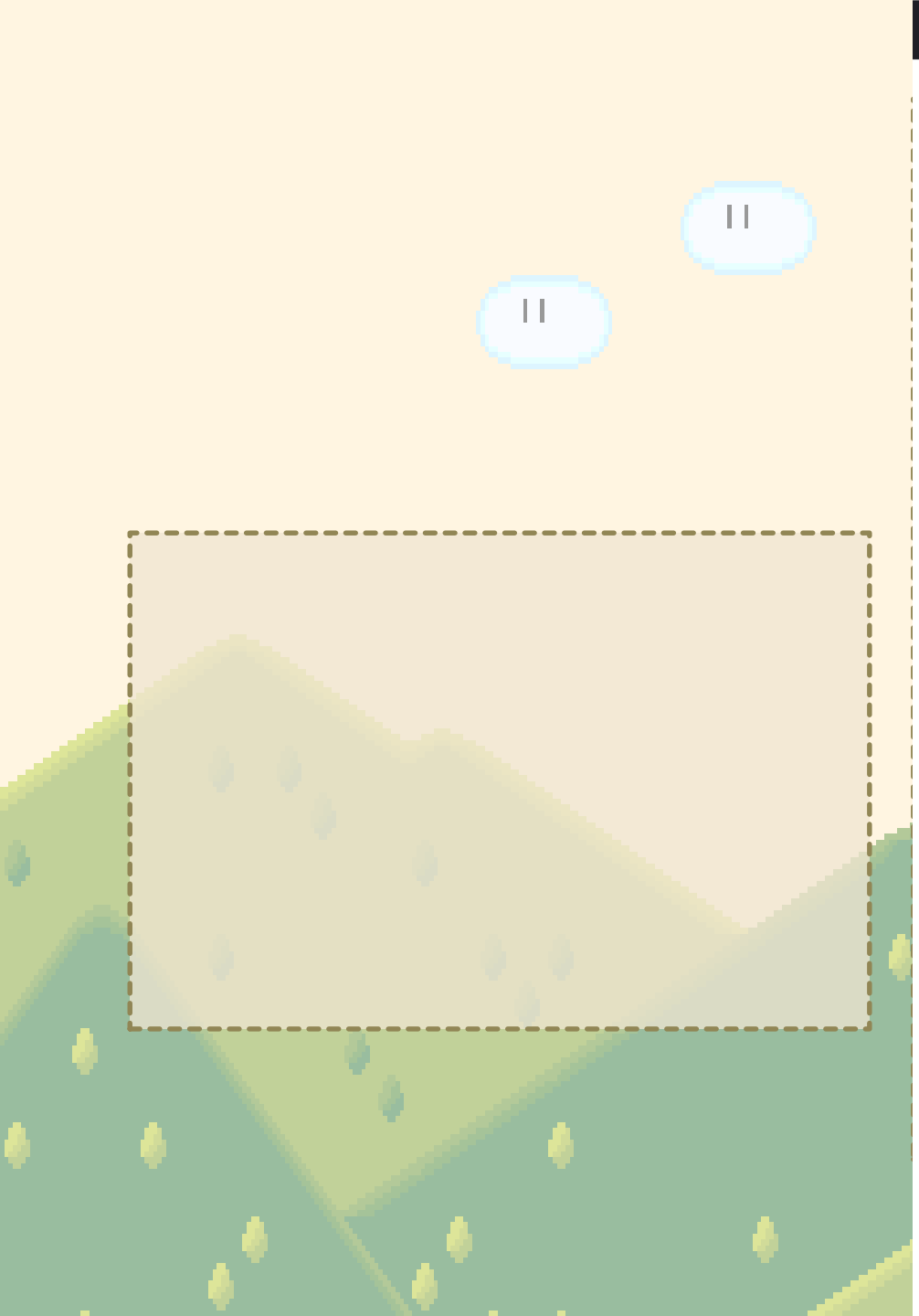
```{r}
data |>
  mutate(age_years = age_months %/% 12) |>
  group_by(age_years) |>
  summarise(
    n_participants = n(),
    intel_mean = mean(mean_intel),
    intel_sd = sd(mean_intel)
  ) |>
  knitr::kable(digits = 3)

model <- mgcv::gam(
  mean_intel ~ s(age_months),
  family = betar(),
  data = data
)
````

No hardcoded paths

Dynamic generated text contents

Output a nice table instead of an R console printout
````



http://localhost:3691

```
ℹ Use `spec()` to retrieve the full column specification for this data.
ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

We examined intelligibility from 538 children between 30 and 119 months in age.

```
data |>
  mutate(age_years = age_months %/% 12) |>
  group_by(age_years) |>
  summarise(
    n_participants = n(),
    intel_mean = mean(mean_intel),
    intel_sd = sd(mean_intel)
  ) |>
  knitr::kable(digits = 3)
```

age_years	n_participants	intel_mean	intel_sd
2	57	0.481	0.196
3	108	0.665	0.178
4	112	0.830	0.108
5	103	0.898	0.070
6	101	0.945	0.039
7	23	0.959	0.037
8	16	0.975	0.014
9	18	0.979	0.012

```
model <- mgcv::gam(
  mean_intel ~ s(age_months),
  family = betar(),
  data = data
)

summary(model)
```

You can get very far with this approach

Write chunks for papers to paste in

demo: <github.com/tjmahr/2015_Coartic/>

Write a whole manuscript

demo: <github.com/tjmahr/mahr-edwards-2017/>

Goal

- You have an analysis artifact
- You can re-create it on more than one computer

Super Mario World © Nintendo

Techniques for other computers

Don’t refer to computer-specific file locations

- "./data/td-mwi-means.csv"
- (. is your current directory)
- (RStudio projects handle your current directory for you)

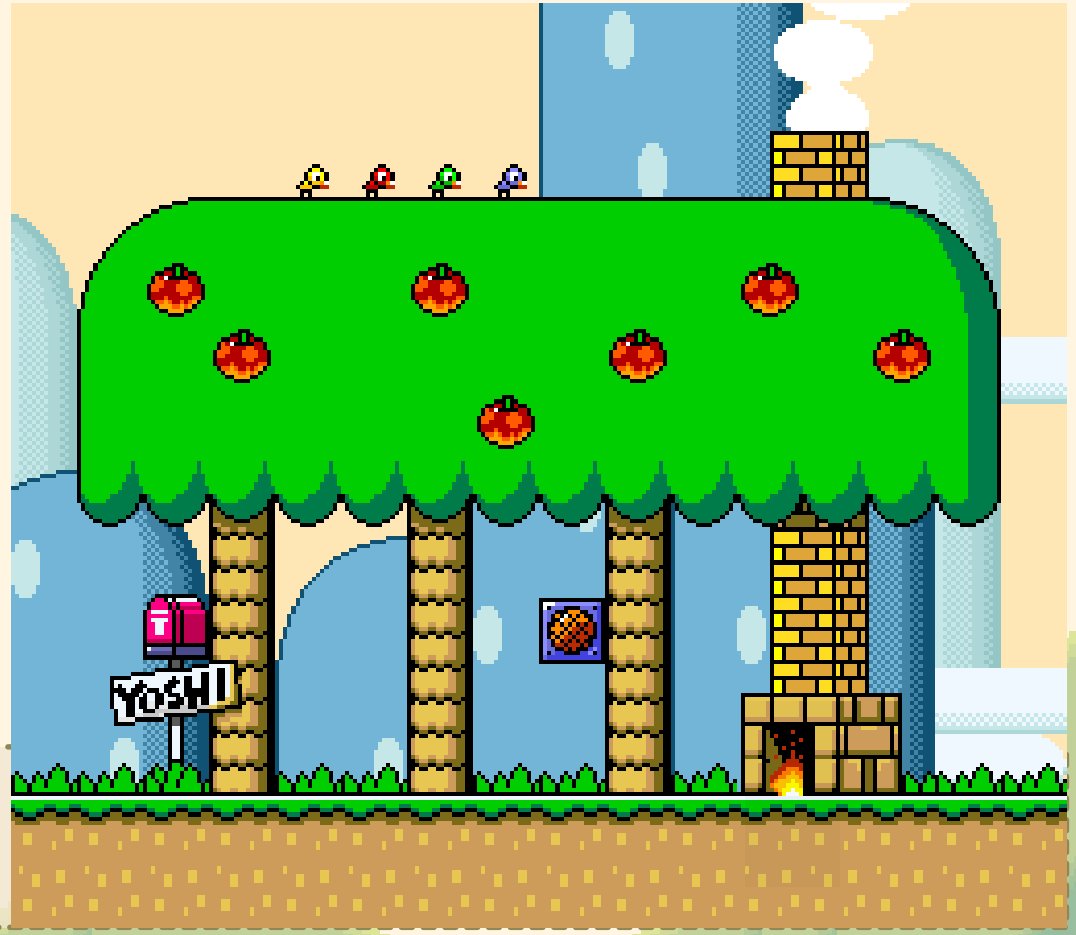
More techniques

Retrieve or sync the data

- Read a csv from the network drive
- Sync project with Onedrive
- Read from box
  - boxr::box_read_csv()

Embed small data (dput(), datapasta pkg)

High hanging fruit



Super Mario World © Nintendo

Use version control

How do we know if anything has *changed* since we reran the analysis?

1. Also create a plaintext version of artifact
2. Track with version control

my-analysis.qmd

````
---
title: "Intelligibility GAM"
author: "Tristan Mahr"
format:
  html:
    keep-md: true
---

```{r}
library(tidyverse)
library(mgcv)

data <- readr::read_csv("data/td-mwi-means.csv")
```

We examined intelligibility from `r nrow(data)` children between
`r min(data$age_months)` and `r max(data$age_months)` months in age.

```{r}
data |>
  mutate(age_years = age_months %/% 12) |>
  group_by(age_years) |>
  summarise(
    n_participants = n(),
    intel_mean = mean(mean_intel),
    intel_sd = sd(mean_intel)
  ) |>
  knitr::kable(digits = 3)

model <- mgcv::gam(
  mean_intel ~ s(age_months),
  family = betar(),
````

Quarto saves the plaintext .md file that gets converted into the HTML file
````

git: big idea

- Version control system
- Create or check out (clone) a code project
- “commit” changes (a very deliberate kind of save/submit action)
- git tracks line-by-line changes
- Made for software developers so very easy to get lost in the sauce

Scenario

- We use git to take a snapshot of the my-analysis.html.md file
- Now suppose 20 (random) children were remove from the data-set.
- We can see line-by-line changes in the results.

I do this obsessively

I commit analysis results when they are in a good place.

I iterate and work on new things.

I make sure I didn't break anything by monitoring these changes.

I confirm reproducibility by looking for changes.

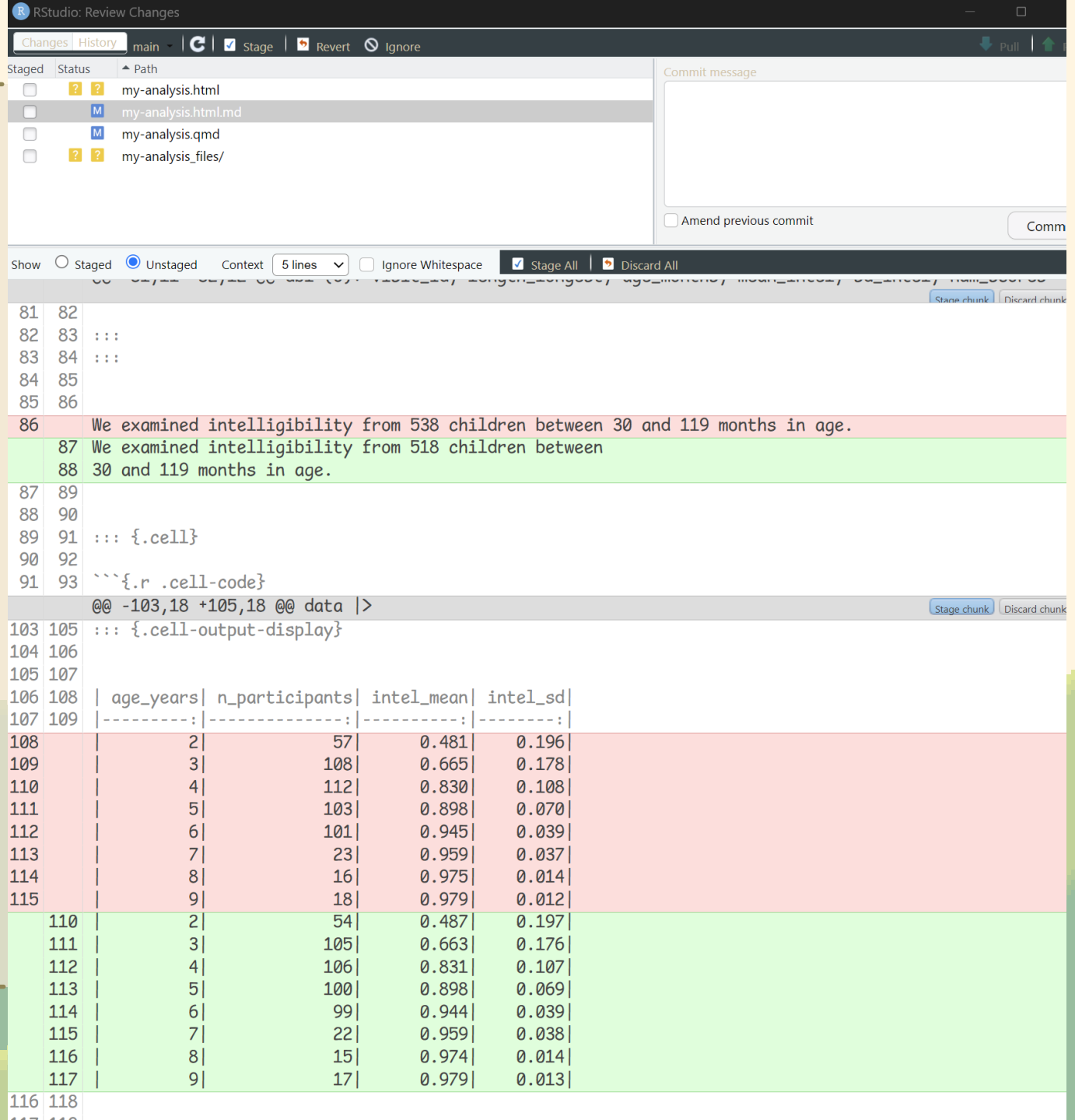
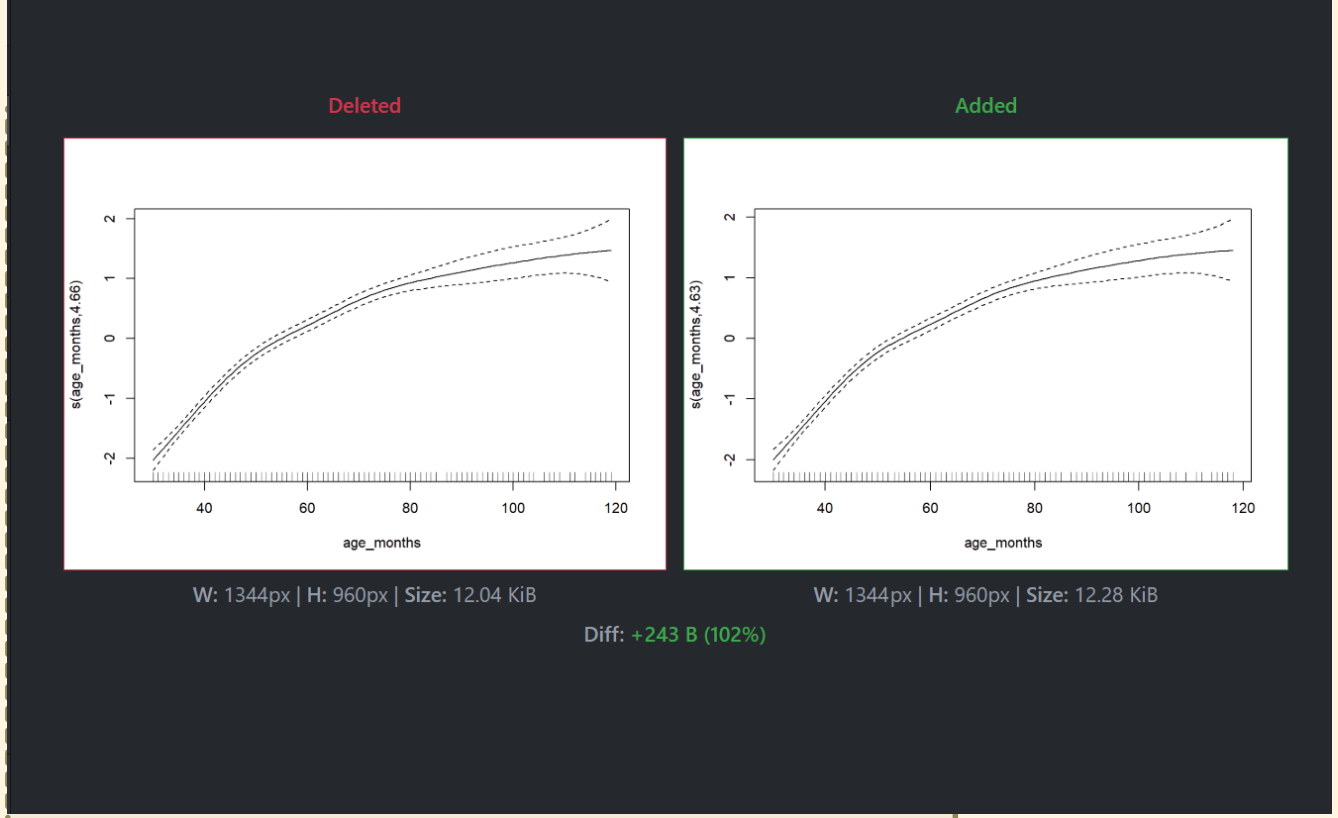
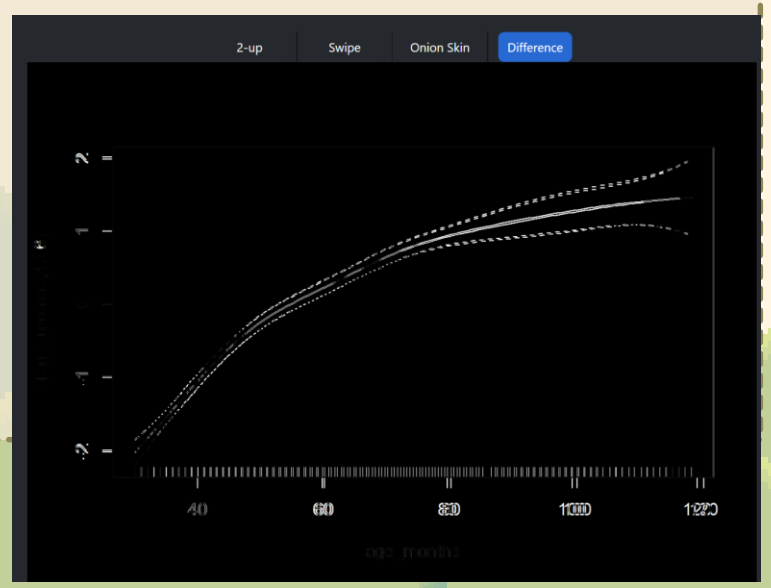


Image changes

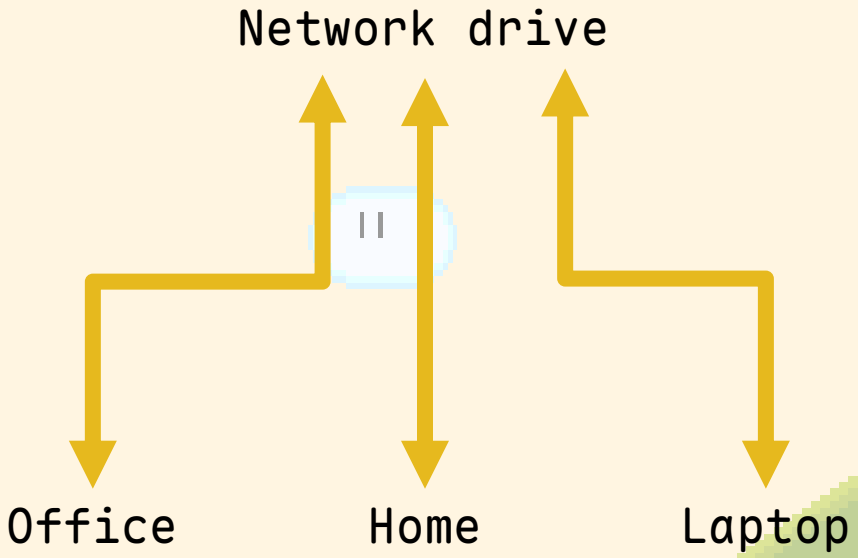
If I commit a figure, I can use the GitHub Desktop app to see a side-by-side comparison or the difference between image pixels.





git is also how I manage my syncing

- Each project is a folder with a git “origin” on the network drive
- Push and pull changes with network copy
- Onedrive-type systems don’t work well with git or projects with many, many files.



Add a workflow/build system

Once a project gets large, we wind up with different datasets and sub-analyses along the way.

How do we make sure everything is up-to-date?

_targets.R

Initialized with targets::use_targets()

```
library(targets)
library(tarchetypes)
library(mgcv)

tar_assign({
  file_td_mwi <- tar_file("data/td-mwi-means.csv")

  data_td_mwi <- file_td_mwi |>
    readr::read_csv() |>
    tar_target()

  model_td_mwi <- gam(
    mean_intel ~ s(age_months),
    data_td_mwi,
    family = betar()
  ) |>
    tar_target()

  doc_td_mwi <- tar_render(path = "my-analysis.qmd")
})
```

Declare the things to make with special tar_functions()

my-analysis.qmd

````
---
title: "Intelligibility GAM"
author: "Tristan Mahr"
format:
  html:
    keep-md: true
---

```{r}
library(tidyverse)
library(mgcv)

data <- targets::tar_read(data_td_mwi)
model <- targets::tar_read(model_td_mwi)
```

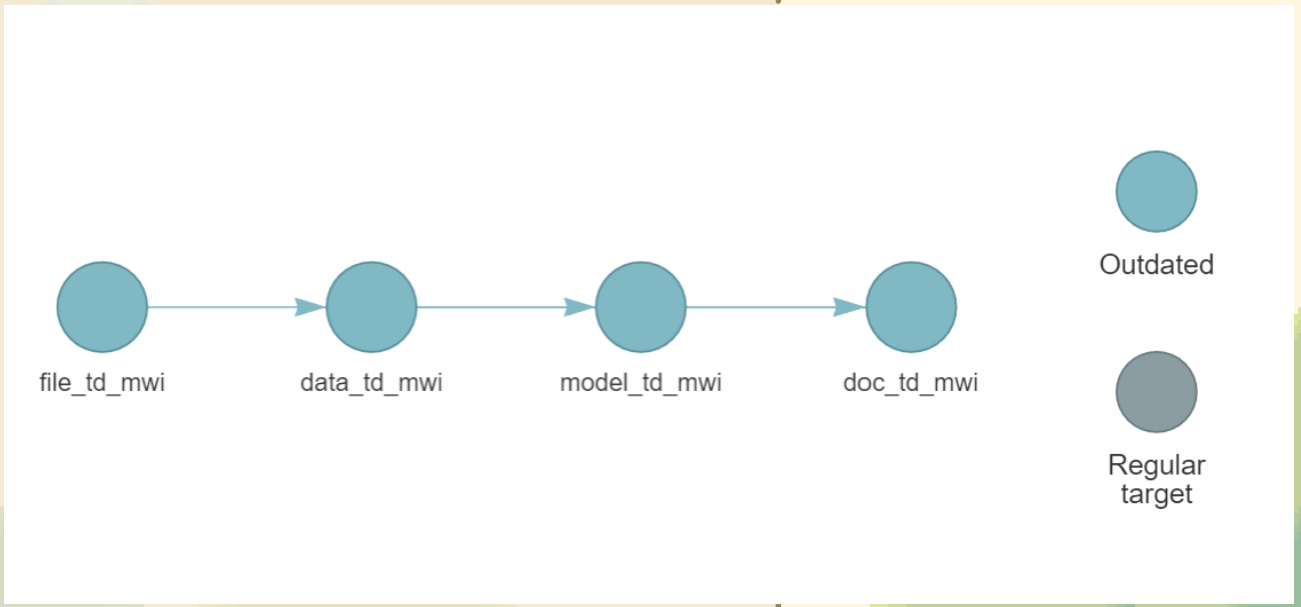
We examined intelligibility from `r nrow(data)` children between
`r min(data$age_months)` and `r max(data$age_months)` months in age.

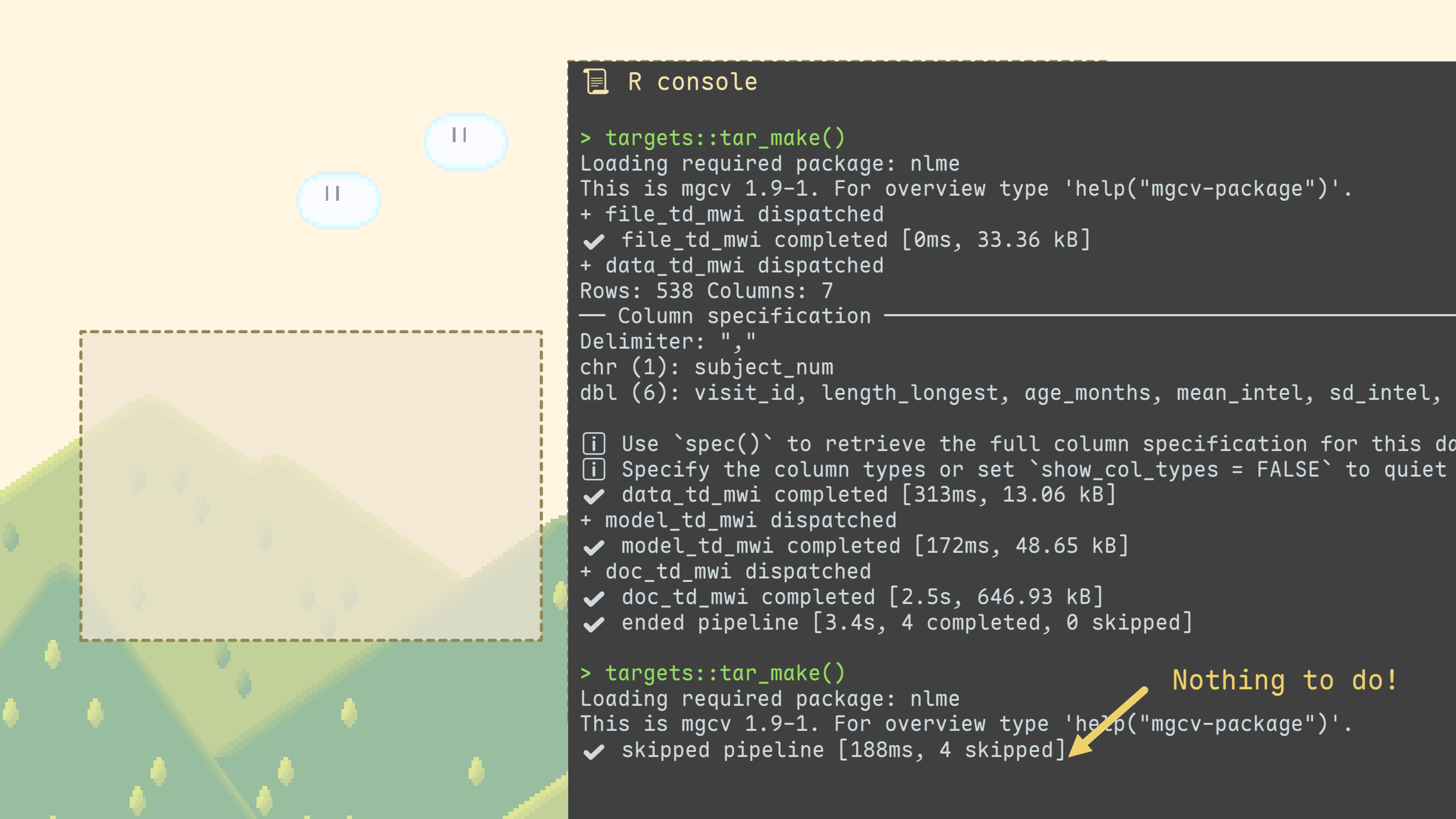
```{r}
data |>
  mutate(age_years = age_months %/% 12) |>
  group_by(age_years) |>
  summarise(
    n_participants = n(),
    intel_mean = mean(mean_intel),
    intel_sd = sd(mean_intel)
  ) |>
````

Read in the targets with tar_read() or tar_load(). These functions signal to targets that a target is being used.
````

tar_visnetwork()

Targets creates a graph of dependencies and will only rebuild outdated dependencies.




R console
> targets::tar_make()
Loading required package: nlme
This is mgcv 1.9-1. For overview type 'help("mgcv-package")'.
+ file_td_mwi dispatched
✔ file_td_mwi completed [0ms, 33.36 kB]
+ data_td_mwi dispatched
Rows: 538 Columns: 7
── Column specification ──
Delimiter: ","
chr (1): subject_num
dbl (6): visit_id, length_longest, age_months, mean_intel, sd_intel,
ⓘ Use `spec()` to retrieve the full column specification for this da
ⓘ Specify the column types or set `show_col_types = FALSE` to quiet
✔ data_td_mwi completed [313ms, 13.06 kB]
+ model_td_mwi dispatched
✔ model_td_mwi completed [172ms, 48.65 kB]
+ doc_td_mwi dispatched
✔ doc_td_mwi completed [2.5s, 646.93 kB]
✔ ended pipeline [3.4s, 4 completed, 0 skipped]
> targets::tar_make()
Loading required package: nlme
This is mgcv 1.9-1. For overview type 'help("mgcv-package")'.
✔ skipped pipeline [188ms, 4 skipped]
Nothing to do!

Demo of my notebook system

<https://github.com/tjmahr/notestar-demo>

Basically a single-site blog

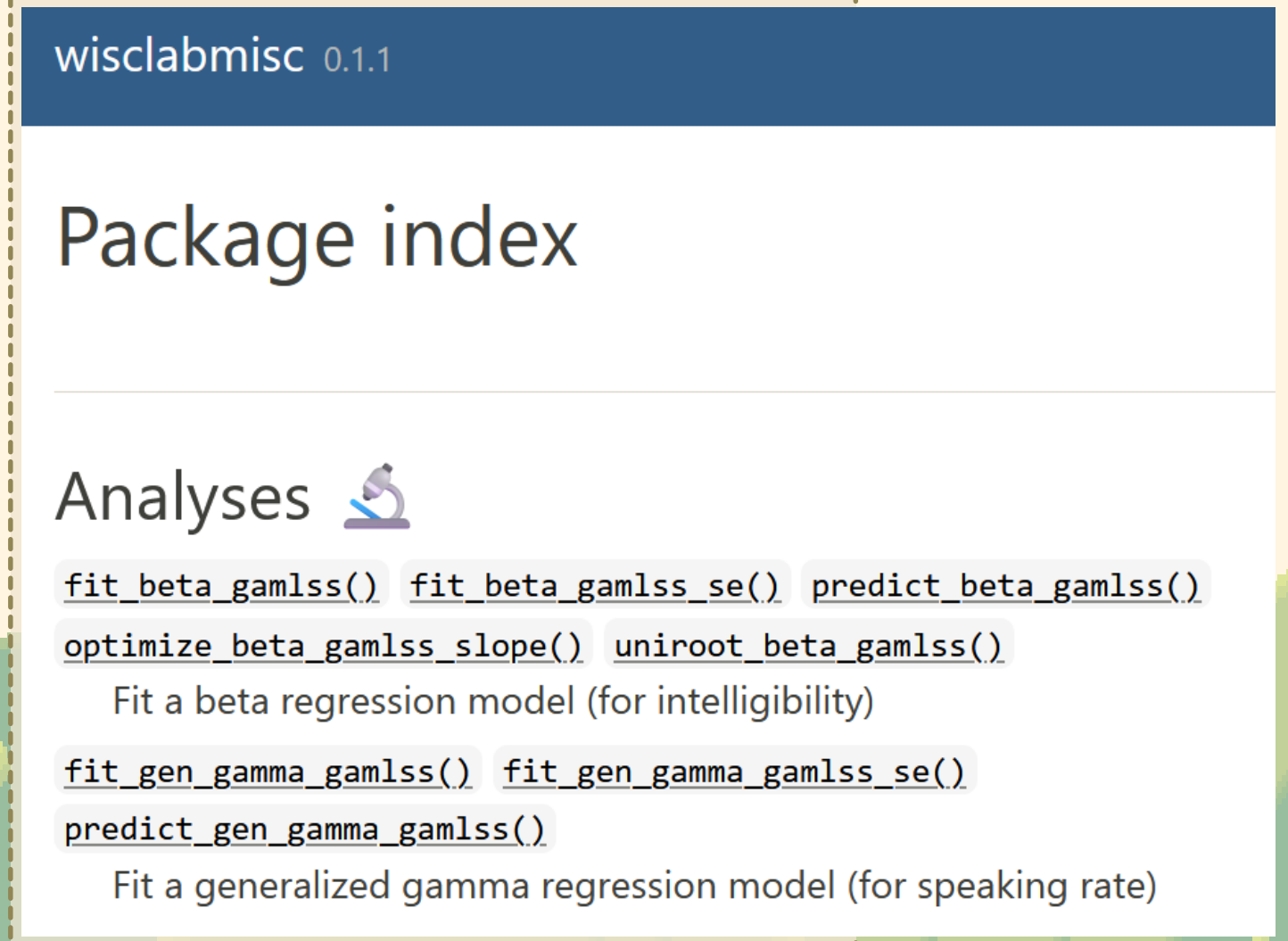
- RMarkdown documents of notebook entries that load in data/model targets
- They all get stitched together into a single HTML file

Very high fruit

Release code for public use

When there is an analysis step that we re-use in different projects, I like to distill it into a more generic version and release it in an R package.

Can plug into package development tools: documentation, testing, automated checks.



Dependencies

- When we rely on “dependencies” (3rd party packages) for data-cleaning, modeling and presentation.
- That software can change, leading to results that don’t reproduce between updates.
- (That’s why we at a minimum want to record package versions with sessionInfo())

Manage R package versions

renv (R environment) helps with some of this package management.

- Look for all the dependencies in a project
- Record those dependencies
- Restore (re-install) old dependencies

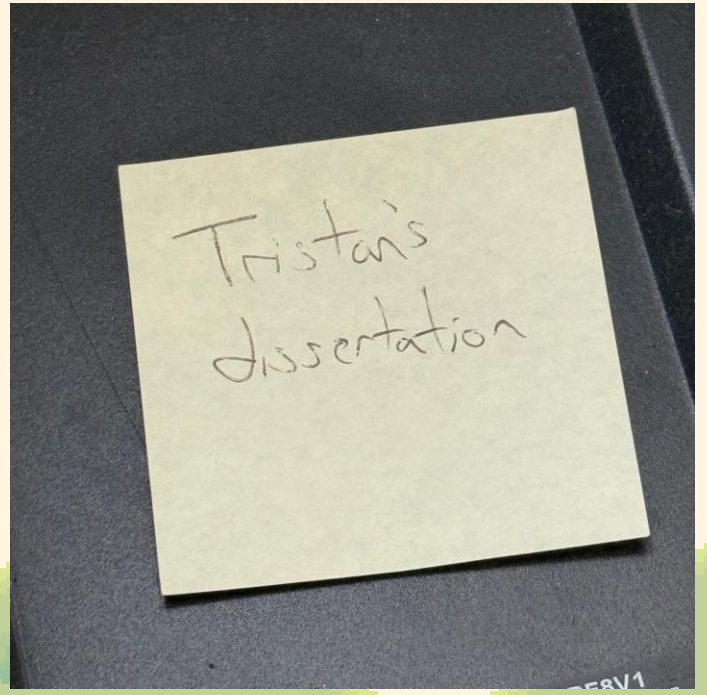
I use it to snapshot versions, but almost never to restore

R console

```
> renv::init()
- Linking packages into the project library ... [103/103] Done!
  The following package(s) will be updated in the lockfile:

  # CRAN ------------------------------------------------------------------
- askpass          [* -> 1.2.1]
- backports        [* -> 1.5.0]
- base64enc        [* -> 0.1-3]
- bit              [* -> 4.6.0]
- bit64            [* -> 4.6.0-1]
- blob             [* -> 1.2.4]
- broom            [* -> 1.0.9]
- bslib            [* -> 0.9.0]
- cachem           [* -> 1.1.0]
- callr            [* -> 3.7.6]
- cellranger       [* -> 1.1.0]
- cli              [* -> 3.6.5]
- clipr            [* -> 0.8.0]
- conflicted       [* -> 1.2.0]
- cpp11            [* -> 0.5.2]
- crayon           [* -> 1.5.3]
- curl             [* -> 6.4.0]
- data.table       [* -> 1.17.8]
- DBI              [* -> 1.2.3]
- dbplyr           [* -> 2.5.0]
- digest           [* -> 0.6.37]
- dplyr            [* -> 1.1.4]
- dtplyr           [* -> 1.3.1]
```

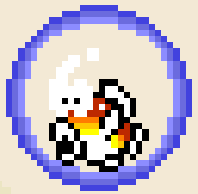
Capture the exact computing environment



Capture the computing environment

I never get this far…

- Use a virtual machine
  - Docker
  - Rix



- Use some cloud thing ?

Resources

UW Madison Data Science workshops hub.datascience.wisc.edu/workshops/

Software Carpentry software-carpentry.org/lessons/

targets docs.ropensci.org/targets/

Data management in large-scale education research datamgmtinedresearch.com/

Good enough practices in scientific computing doi:10.1371/journal.pcbi.1005510

Science as amateur software development youtube.com/watch?v=8qzVV7eEiaI

Super Mario World © Nintendo