Good practices for sharing research code

Scientific workflows: Tools and Tips 🛠️

2026-07-16

What is this lecture series?

Scientific workflows: Tools and Tips 🛠️

🗓️ Every 3rd Thursday 🕓 4-5 p.m. 📍 Webex

- One topic from the world of scientific workflows
- Material provided online
- If you don't want to miss a lecture
  - Subscribe to the mailing list

Code sharing in Ecology

BRITISH ECOLOGICAL SOCIETY | Methods in Ecology and Evolution

PERSPECTIVE | Open Access

Data- and code-archiving in the British Ecological Society journals: Present status and recommendations for future improvements

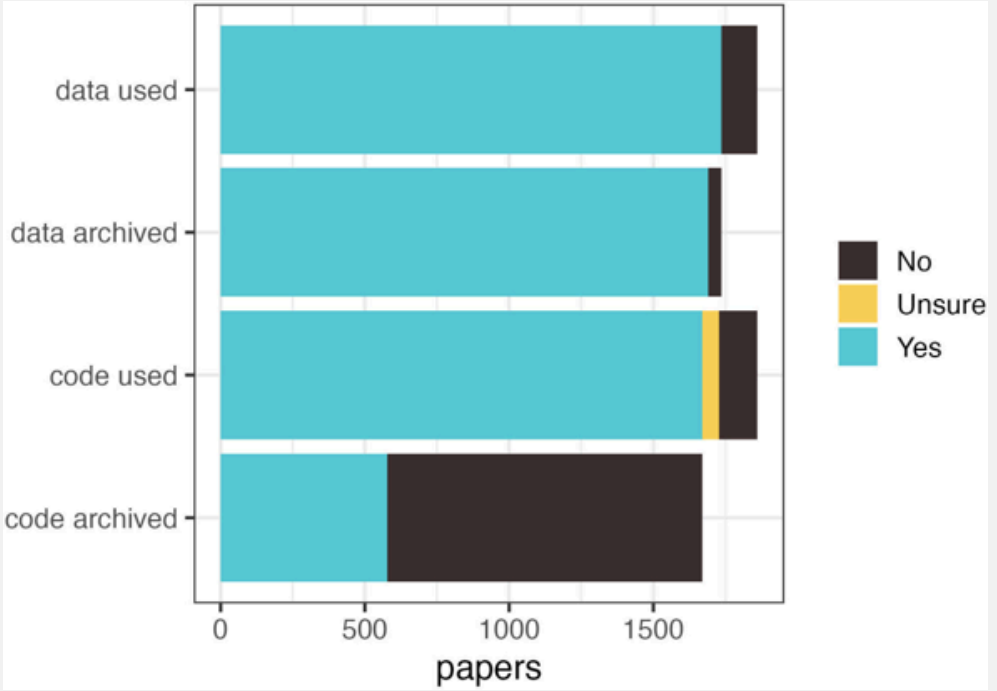
Natalie Cooper, the BES Data and Code Hackathon Group

First published: 02 June 2026 | https://doi.org/10.1111/2041-210x.70338 | VIEW METRICS

1861 ecology papers from 7 journals (2017-2024)

Sharing code is still the exception (?)

- Data is often shared, code not
- Shared code is often not sufficient
  - low reproducibility
  - missing a README (39%)
  - low-quality READMEs
  - missing license (26%) or wrong license
  - …



Cooper et al. (2026), *Methods in Ecology and Evolution*

Why?

- Time pressure - not a priority
- Journals still often don't require it
- Code is not part of peer review
- Not sure how to do it properly

Code sharing is worth it

- Open science creates trust
- Reproducibility: Results can be verified
- Reusability: Others (including future you) can build on your work
- Visibility: Credit, collaborations
- A portfolio of your coding skills

Today

- How to build and publish a good practice repository
- Checklist with the most important things to do
- Focus on data analysis projects
- Where AI tools can help

What we will not talk about

- Research software (packages, libraries, tools)
- Writing code (Good coding practices, style, functions, testing, etc …)
  - For this check out the lecture R code that lasts and some other resources linked on the website
- Too many language specific details (but some R and Python hints)

Our example project

My paper is accepted:
"Temperature and antibiotic resistance in soil bacteria".

Now I need to publish code and data

Let's make this repo publishable!

```
temperature-resistance/
├── clean.R
├── analysis_final.R
├── makeplots.R
├── growth_assay.csv
├── growth_assay_clean.csv
├── results.csv
└── Rplot01.png
```

Structure & orientation

> Can someone understand it?

Separate your files into folders

- Most important: separate data, code, and output
- Separate raw data (read-only) from processed data (produced by scripts)
- Follow a template or use your own structure, as long as it is clear and consistent (keyword: research compendium)

```
temperature-resistance/
├── data/
│   ├── raw/
│   │   └── growth_assay.csv
│   └── clean/
│       └── growth_assay_clean.csv
├── analysis/
│   ├── clean.R
│   ├── analysis_final.R
│   └── makeplots.R
└── output/
    ├── figures/
    │   └── Rplot01.png
    └── tables/
        └── results.csv
```

Our messy project, sorted into folders

Further reading: rrtools (R), research compendia with computational notebooks (R), Cookiecutter Data

Use informative file names

- Human readable: reveal the content
- Machine readable: no spaces, no special characters
- Work with default ordering: left-padded numbers or YYYY-MM-DD dates first

```
temperature-resistance/
├── data/
│   ├── raw/
│   │   └── growth_assay.csv
│   └── clean/
│       └── cleaned_data.csv
├── analysis/
│   ├── 01_clean-data.R
│   ├── 02_fit-models.R
│   └── 03_make-figures.R
└── output/
    ├── figures/
    │   └── linear-model-fit.png
    └── tables/
        └── model-results.csv
```

Our project with nicer file names

Add a thorough README

- README's are the entry point to your project
- A README should be a simple text format (plain text or Markdown)

```
temperature-resistance/
├── README.md
├── data/
│   ├── raw/
│   │   └── growth_assay.csv
│   └── clean/
│       └── cleaned_data.csv
├── analysis/
│   ├── 01_clean-data.R
│   ├── 02_fit-models.R
│   └── 03_make-figures.R
└── output/
    ├── figures/
    │   └── linear-model-fit.png
    └── tables/
        └── model-results.csv
```

What a README should do

At minimum, a README should answer:

- **What** the project is, and which **paper** it belongs to
- **How to run it** (how to install, which order to run the scripts)
- **How the files are structured** (what is in each folder, what the files are)
- **How to cite** the project
- **Who to ask** when it breaks

A README skeleton

```
# Temperature and antibiotic resistance in soil bacteria

Code and data for: Baldauf et al. (2026), *Journal*, doi:10.xxxx/xxxxx

Cite as: Baldauf et al. (2026). Data and code for "Temperature and antibiotic resistance in soil bacteria".

Code is licensed under MIT, data under CC-BY-4.0. See `LICENSE` and `data/LICENSE` for details.

## How to run

Download or clone the repository to run it locally.

The project needs R version 4.0.0 or higher.

1. Restore the environment: `renv::restore()` - This will create a project library and install all required
2. Run `main.R`, which runs all analysis scripts in order

## Project structure

- `analysis/`      analysis scripts, run in numerical order
- `data/raw/` the original growth-assay measurements (read-only)
```

Some examples of good README files

- https://github.com/lciernik/attentive-layer-fusion
- https://github.com/openplantpathology/Reproducibility_in_Plant_Patholog
- https://github.com/paocorrales/t-drop-trends

Running the code

> Can someone rerun it?

Make the run order obvious

Level 1: Number your scripts `01_`, `02_`, etc. and add this to the "How to run" section in the README.

```
## How to run

1. Install dependencies with `renv::restore()`
2. Run the scripts in numerical order:
    1. `01_clean-data.R`
    2. `02_fit-models.R`
    3. `03_make-figures.R`
```

```
temperature-resistance/
├── README.md
├── data/
│   ├── raw/
│   │   └── growth_assay.csv
│   └── clean/
│       └── cleaned_data.csv
├── analysis/
│   ├── 01_clean-data.R
│   ├── 02_fit-models.R
│   └── 03_make-figures.R
└── output/
    ├── figures/
    │   └── linear-model-fit.png
    └── tables/
        └── model-results.csv
```

Make the run order obvious

Level 2: Create a **main script** that calls all scripts in order, so this is the only file someone needs to open.

main.R | main.py

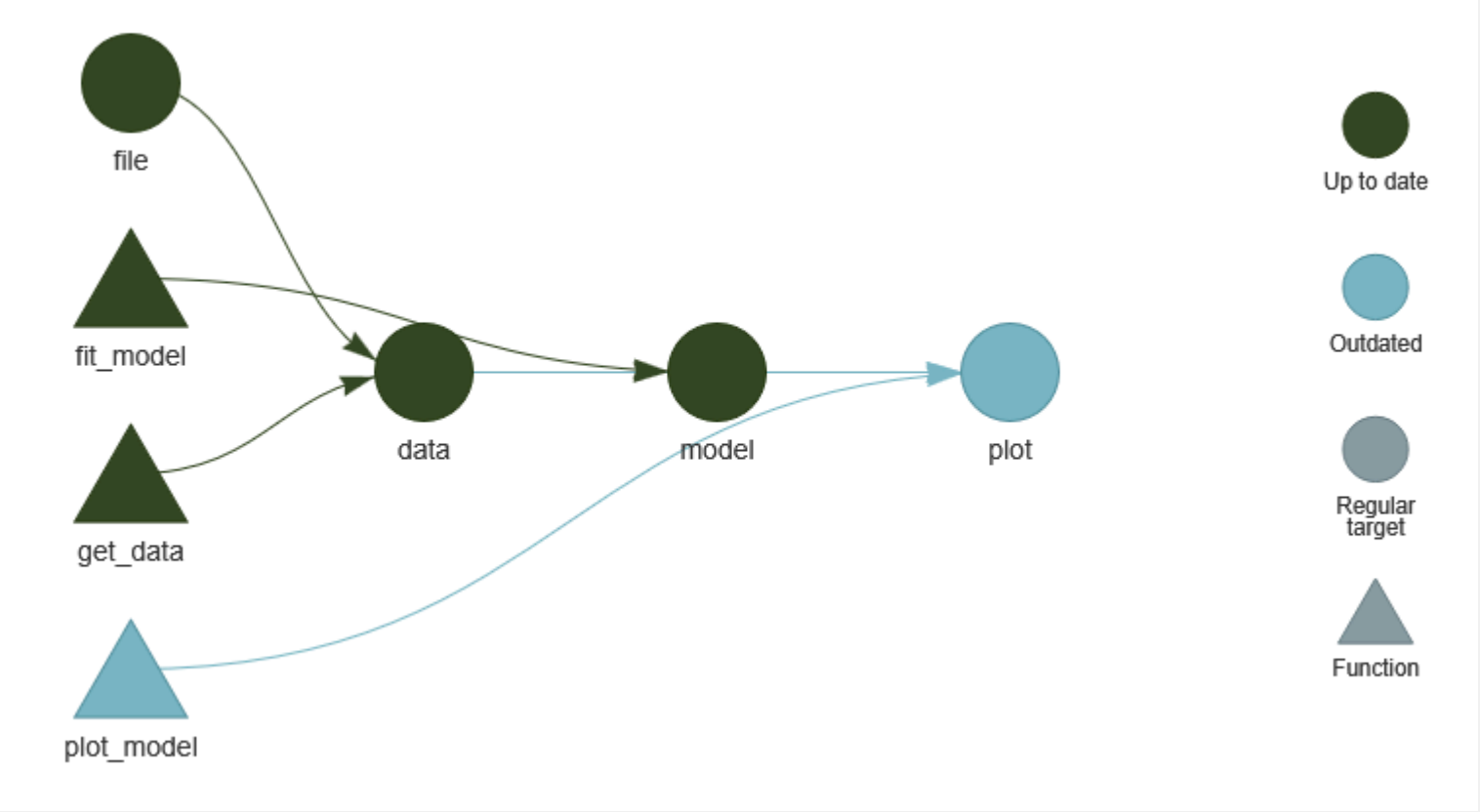
```r
# Run the whole analysis pipeline

# Step 1: Clean the data
# Input: `data/raw/growth_assay.csv`
# Output: `data/clean/cleaned_data.csv`
source("analysis/01_clean-data.R")
# Step 2: Fit the models and produce table 1
# Input: `data/clean/cleaned_data.csv`
# Output: `output/tables/model-results.csv`
source("analysis/02_fit-models.R")
# Step 3: Make all manuscript figures
# Input: `data/clean/cleaned_data.csv`, `output/table
# Output: `output/figures/linear-model-fit.png` (Figu
source("analysis/03_make-figures.R")
```

```
temperature-resistance/
├── README.md
├── main.R
├── data/
│   ├── raw/
│   │   └── growth_assay.csv
│   └── clean/
│       └── cleaned_data.csv
├── analysis/
│   ├── 01_clean-data.R
│   ├── 02_fit-models.R
│   └── 03_make-figures.R
└── output/
    ├── figures/
    │   └── linear-model-fit.png
    └── tables/
        └── model-results.csv
```

Make the run order obvious

Level 3: Use **workflow management tools** like targets (R) or Snakemake (any language). Define workflows, automatically run the steps in the right order and only rerun what changed.



An example workflow from the targets package

Record all dependencies and versions

Let people know what they need to install to make the code work.

- The language itself and its version (e.g. R 4.4.1, Python 3.12)
- All libraries/packages and their versions
- Any other programs your code calls

Record all dependencies and versions

Level 1: Copy-paste dependencies into the README. Easy, but less convenient for people who need to install the software.

R | Python

- Run `sessionInfo()` at the **end** of your pipeline (so every package is loaded), then paste the package list into the README.
- `renv::dependencies()` also lists the packages your code uses, but without versions.

```
sessionInfo()
#> R version 4.4.1 (2024-06-14)
#> ...
#> other attached packages:
#> [1] ggplot2_3.5.1 dplyr_1.1.4   readr_2.1.5
```

Record all dependencies and versions

Level 2: Snapshot your local environment so users can recreate them easily.

R | Python

Use the `renv` package to:

- `renv::init()`: set up a project library and record all packages in a `renv.lock` file
- `renv::snapshot()`: update your lockfile with new packages
- `renv::restore()`: recreate the environment on another machine

Record all dependencies and versions

Level 3: Package the whole environment in a **container** (e.g. Docker) so the code runs identically on any machine. Powerful but also more to learn and maintain.

R | Python

- The Rocker project gives you different R images to build on.

Use paths relative to project root

- Paths anchored to the project root work on any machine, from any working directory
- Absolute paths (`C:/Users/selina/PhD/...`) will break the project

R | Python

Use an R project (`.Rproj`, in RStudio or Positron) and the `here` package. `here()` finds the root by searching upward for a marker (`.Rproj`, `.git`, `.here`), independent of the working directory:

```
library(here)
read.csv(here("data", "raw", "growth_assay.csv"))
```

Data

> Can someone understand the data situation?

Save your data in open formats

- Save data in **open formats**: `.csv`, `.tsv`, `.txt`, `.json`, `.parquet`, etc.
- Avoid proprietary formats: `.xlsx`, `.sav`, `.dta`, etc.

```
temperature-resistance/
├── README.md
├── main.R
├── renv.lock
├── data/
│   ├── raw/
│   │   └── growth_assay.csv
│   └── clean/
│       └── cleaned_data.csv
├── analysis/
│   ├── 01_clean-data.R
│   ├── 02_fit-models.R
│   └── 03_make-figures.R
└── output/
    ├── figures/
    │   └── linear-model-fit.png
    └── tables/
        └── model-results.csv
```

Document your data files

Describe all your data files: Variable names, units, missing data codes, and other relevant information.

E.g. one table per file:

Variable	Meaning	Unit / coding
`sample_id`	Unique sample identifier	text
`temp`	Incubation temperature	°C
`resistant`	Grew on antibiotic plate	0 = no, 1 = yes
`mic`	Minimum inhibitory concentration	mg/L

Document your data files

- Add documentation to the main README
- Or create a separate `data/README.md` for all data files

Main README:

```
### `data/`

The `data/` folder contains the raw and cleaned
growth assay measurements.
See `data/README.md` for all variables.
```

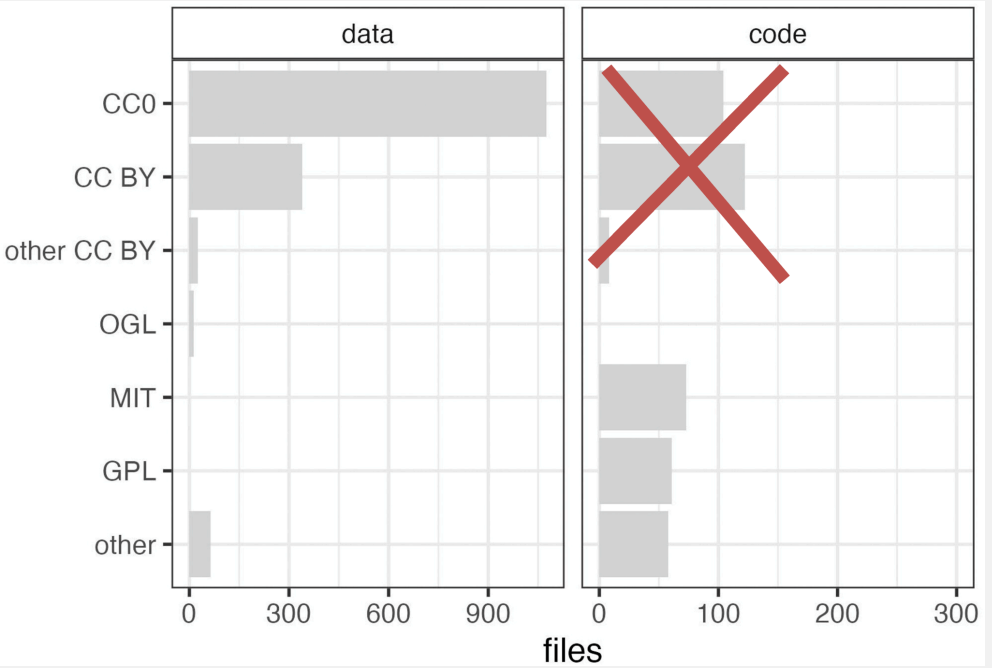
```
temperature-resistance/
├── README.md
├── main.R
├── renv.lock
├── data/
│   ├── README.md
│   ├── raw/
│   │   └── growth_assay.csv
│   └── clean/
│       └── cleaned_data.csv
├── analysis/
│   ├── 01_clean-data.R
│   ├── 02_fit-models.R
│   └── 03_make-figures.R
└── output/
    ├── figures/
    │   └── linear-model-fit.png
    └── tables/
```

Licence & citation

> Can someone legally reuse and cite it?

Add a licence

- Without a licence file, people can not reuse your code
- Most popular licences in ecology are:
  - Creative commons licences (CC0, CC-BY)
  - MIT & GNU General Public License (GPL)
- Careful: CC licenses are NOT for code, only data & text



Number of licences used for code and data in BES ecology papers. Cooper et al. (2026), *Methods in Ecology and Evolution*

Which licence to choose?

- Are there specific requirements for your project?
- Checkout choosealicense.com to guide through licence choice

Standard open licences:

Licence	What it does	For
CC0	Public-domain waiver, no attribution required	Data
CC BY	Reuse for any purpose, attribution required	Data
MIT	Permissive: reuse freely, keep the notice	Code
GNU GPL v3	Copyleft: derivatives stay open, same licence	Code

How to add a licence

Many different ways. In general: Add a `LICENSE` file containing the licence text into the repository.

Manual | GitHub

1. Create a `LICENSE` file
2. Go to choosealicense.com and copy the text of your chosen licence into the file
3. Add your name and the year to the licence text (if needed, e.g. for MIT license)

```
temperature-resistance/
├── README.md
├── LICENSE
├── main.R
├── renv.lock
├── data/
│   ├── README.md
│   ├── LICENSE
│   ├── raw/
│   └── clean/
├── analysis/
└── output/
```

Make it clear in the README what is the licence for code and data, e.g.:

```
All files in `/data` are licensed under CC-BY-4.0, all other files are licensed under MIT. See
`data/LICENSE` and `LICENSE` for details.
```

Make your code citable

Level 1: add a "How to cite" section in the README.

```
## Citation
Baldauf et al. (2026). Code and data for "Temperature and antibiotic resistance in
soil bacteria". https://doi.org/10.xxxx/xxxxx
```

Make your code citable

Level 2: add a `CITATION.cff` file

- small YAML file (create with `cffinit` website and add to project)
- Also add your Orcid ID
- this cff file holds all citation information
  - GitHub shows a "Cite this repository" button
  - Zenodo reads it when you archives your project, so the credit is correct automatically

Ready to publish?

Some final checks before you publish:

- No junk files: `.DS_Store`, `.Rhistory`, `__pycache__/`, … (use a `.gitignore`)
- No secrets or sensitive data: keys, passwords, personal/patient data
- Reproducibility check: send to a friend, or run on another machine

```
temperature-resistance/
├── README.md
├── LICENSE
├── CITATION.cff
├── main.R
├── renv.lock
├── data/
│   ├── raw/
│   ├── clean/
│   └── README.md
├── analysis/
└── output/
```

Archive & publish

> Can someone find the exact version you used?

Can I just publish it on Github?

- Github is a standard place to develop and share code
- The benefit of Github:
  - Easy to find and share
  - Nice display of README, LICENCE, etc.
- BUT: It is not an archive where a specific version of the code is frozen

Archive your data and get a DOI

- Archiving your data means freezing a specific version of it
- Upload your project to a repository like Zenodo, Dryad, Figshare, or a domain-specific repository
- You will get a DOI (Digital Object Identifier) for that version of your project

> **Tip**
>
> You can link your GitHub repo to Zenodo/Figshare to archive your project directly.
>
> - Connecting the repo with Zenodo to get a DOI
> - Import from Github to Figshare

Add the DOI to the README and the paper

Once you have a DOI, add it in two places:

- the code README (Zenodo gives you a nice copy-paste badge)
- the manuscript: “Code and data are archived at https://doi.org/10.5281/zenodo.xxxxx (v1.0).”

We're done 🎉

> Code is successfully shared and published!

How can AI help

- **Good at the tedious parts**: listing files and their inputs/outputs, checking the README covers everything
- Useful for preparing projects
  - Draft READMES
  - Audit your project against the checklist before publishing
- AI is **not reliable**: expect false positives and negatives -> so use it as a helper, not an authority

What to do

- **Give it access to your repo**: an IDE assistant (e.g. Claude Code, GitHub Copilot), Codex, or a browser upload
  - Carful if you have sensitive data!
- **Write a clear prompt**: say what you want and how the output should look, with an example if you can

> **Note**
>
> Find example prompts to create READMEs and to audit your projects based on the criteria we discussed today on the lecture website

Coming soon

A **repo-reviewer AI tool**: Review your repository and get a prioritized report on what to improve before publishing

- Based on clear criteria
- Actionable: tells you what to do and how to fix it
- Usable in different ways (Claude Code skill, GitHub Action, copy-paste prompt, …)

Wrap up

Take aways

- Shareable ≠ perfect: Don't aim for perfection, "just" aim to make your repository understandable and reproducible
- Follow checklist
- Implement good practices from the beginning
- Let AI help you with the annoying parts

Next lecture

Summer/Conference break in August & September!

Topic tba

🗓️ 15.10.2026 🕓 4-5/6 p.m. 📍 Webex

- For topic suggestions or feedback, send me an email
- Subscribe to the mailing list

Thank you for your attention :)

Questions?

References & resources

- Cooper et al. (2026) Data- and code-archiving in the British Ecological Society journals. *Methods in Ecology and Evolution*
- "But is the code (re)usable?" *Nature Computational Science* editorial (2021)
- BES Guide to Reproducible Code, same society as the opening figure
- The Turing Way: community handbook for reproducible research
- Wilson et al. (2017) Good enough practices in scientific computing. *PLOS Computational Biology*
- choosealicense.com · cffinit · GitHub → Zenodo guide
- CODECHECK register and ReproHack Hub: repositories whose code someone else actually ran
- Rocker project (Docker for R) · pyprojroot (the Python `here`)
- fair-software.eu: five simple first steps towards FAIR software
- How to name files, talk by Jenny Bryan

Appendix

What if you cannot share the data?

- **Data too big**: Use a separate data repository (Zenodo, Dryad, Figshare, etc.) and link to it in the README. If you can: include a download script in your code to get the data.
- **Restricted data**: Include a statement about where and how to request access in the README. Add a `data/README.md` so the code can be reviewed even without data
- **Closed data**: Include synthetic example data + the code that generated it, so the pipeline runs. Include a statement in the README that the real data cannot be shared.

Set a seed for anything random

- Many random processes like sampling, bootstrapping, simulations, even plots with jittering
- Don't forget to set a random seed so that the results are reproducible

R | Python

```r
set.seed(42)
```