Project layout and reproducibility for R analyses

Risk assessment

– An edited raw data file cannot be reliably restored, and results derived from it are not reproducible.

▷ Treat `data/raw/` as read-only.

Reviewed: May 14, 2026

This is a bench card. Full protocol available online.

›› *Starting a new project*

(1.) Decide whether the project uses Conda, which pins the R version and system libraries SOP 0042, and Git, which tracks the code history SOP 0101. If plain R is sufficient, skip the optional steps.

(2.) Create a project folder.

- In RStudio, choose "New Project" and a (new) directory. For Git, tick "Create a git repository".
- Otherwise, create and navigate to the project folder in a terminal:

```
mkdir my-project && cd my-project
```

(3.) *Optional:* With Conda, create an environment named after the project folder and activate it.

```
mamba create -n my-project r-base=4.5 r-usethis r-here
mamba activate my-project
```

(4.) *Optional:* Initialize Git and add a `.gitignore` (see Supplementary Resources) before the first commit.

```
git init -b main
```

(5.) Install the packages the scripts need, never from inside a script.

- With Conda, install into the environment from a terminal, for example:

```
mamba install -n my-project r-dplyr r-tidyr r-readr r-ggplot2 r-broom r-knitr r-rmarkdown r-waldo
```

- Otherwise, install the packages from within an R session, for example:

```
install.packages(c("dplyr", "tidyr", "readr", "ggplot2", "broom",
                   "knitr", "rmarkdown", "waldo", "usethis", "here"))
```

(6.) Start R in the project folder. In R, create an `.Rproj` file and an empty `R/` subfolder for function definitions, and turn off saving and restoring of the workspace.

```
usethis::create_project(getwd(), rstudio = TRUE)
usethis::use_blank_slate()   # RStudio: never save or restore .RData
```

Critical: With Conda, start R from the terminal with the project environment activated. To use RStudio for editing, run `open -a RStudio my-project.Rproj` (macOS only) once the file exists. RStudio started any other way runs the system R.

›› *Folder conventions*

(1.) Create the folders the project needs from this layout; a small analysis uses only some of them:

```
my-project/
├── .gitignore              # with Git
├── conda-lock.yml          # with Conda: locked environment
├── environment.yml         # with Conda: direct dependencies
```

```
├── my-project.Rproj
├── README.md
├── data/
│   ├── raw/                # this project's own data; read-only
│   │   ├── README.md       # provenance of each file
│   │   └── samples.csv     # sample sheet
│   ├── external/           # obtainable again from a named source
│   │   └── README.md       # source, version, and date of each file
│   └── processed/          # derived; regenerated by the pipeline
├── R/                      # function definitions, one file per purpose
│   ├── preprocess.R
│   └── model.R
├── scripts/                # runnable steps, numbered in execution order
│   ├── 00_params.R         # pinned values, each with its source
│   ├── 01_filter.R
│   ├── 02_model.R
│   ├── 03_figures.R
│   └── 99_checks.R         # assertions over the derived outputs
├── notebook/               # log and notebooks
│   ├── _log.md             # running record
│   └── report.qmd
└── output/                 # figures, tables, reports; regenerated
```

(2.) Document raw data provenance and anything that affects interpretation in `data/raw/README.md`.

```
plate_1.csv
    Source:   Plate reader export, BMG CLARIOstar, 2026-04-15
    Obtained: 2026-04-15 by BB
    Notes:    Unmodified export; well assignment in data/raw/samples.csv.
```

Critical: Sensitive data (patient identifiers, unpublished sequences, restricted-access metadata) must not be tracked!

(3.) Map samples and experimental factors in `data/raw/samples.csv` (Broman and Woo, 2018).

```
plate,well,sample_id,replicate,concentration_uM
plate_1,A1,control,1,1000
plate_1,A2,control,1,100
plate_1,A3,control,1,10
plate_1,A4,control,1,1
plate_1,A5,control,1,0
```

(4.) Copy or fetch files that come from another project or a public source into `data/external/`, and record source, version, and date in `data/external/README.md`.

Critical: Do not read another project's files by a path that leaves the project root: the path breaks when either folder moves, and the input changes without notice when the other project is rerun.

Writing an analysis script

(1.) *Critical:* Keep `setwd()`, `rm(list = ls())`, `attach()`, and `install.packages()` out of every script. Each depends on or alters state outside the script: one user's folder layout, the session, the search path, the package library.

(2.) Begin each script with a one-line purpose, the libraries, the parameters, and the functions it uses.

```
# 01_filter.R - import the plate, annotate the wells, drop failed readings
library(dplyr)
```

```
source(here::here("scripts", "00_params.R"))
source(here::here("R", "preprocess.R"))
```

(3.) Read data inputs at the top of the script, by paths relative to the project root with `here::here()`, so dependencies are easily discovered.

```
# inputs
samples <- readr::read_csv(here::here("data", "raw", "samples.csv"))
plate   <- readr::read_csv(here::here("data", "raw", "plate_1.csv"))
```

(4.) Write every output at the bottom of the script. Save an intermediate R object with `readr::write_rds()`:

```
# outputs
readr::write_rds(filtered, here::here("data", "processed", "filtered.rds"))
```

Export a result that is read outside R as CSV, here from `02_model.R`:

```
fit <- lm(signal ~ sample_id, data = filtered)

broom::tidy(fit) |>
  readr::write_csv(here::here("output", "tables", "02_model_summary.csv"))
```

(5.) Save each figure from code with explicit dimensions. Name every file in `output/` after its script.

```
p <- ggplot2::ggplot(filtered,
                     ggplot2::aes(concentration_uM, signal, colour = sample_id)) +
  ggplot2::geom_point()

ggplot2::ggsave(here::here("output", "figures", "03_titration.pdf"),
                plot = p, width = 85, height = 60, units = "mm")
```

(6.) Collect pinned values (thresholds, reference versions, the random seed) in `scripts/00_params.R` with their source. The file computes nothing.

```
# 00_params.R - pinned values, each with its source
params <- list(
  seed       = 20260514L,     # arbitrary; fixed for all stochastic steps
  min_signal = -0.5,          # QC threshold; notebook/_log.md, 2026-05-12
  n_boot     = 2000L          # bootstrap resamples; notebook/_log.md, 2026-05-13
)
```

(7.) Set the seed of the random number generator *once* in each script that draws random numbers (bootstraps, permutations, simulations), after `library()` and `source()`:

```
set.seed(params$seed)
```

> ***Critical:*** Do not set the seed inside a function or a sourced file. Resetting the random number generator there silently couples unrelated random draws.

(8.) Replace constructs that go wrong without an error:

Construct	Use instead	What goes wrong
`as.numeric(f)` on a factor	`as.numeric(as.character(f))`	Returns the internal level codes, not the values.
`x == y` on decimal numbers	`dplyr::near(x, y)`	`sqrt(2)^2 == 2` is `FALSE`.
`1:length(x)`	`seq_along(x)`	Counts 1, 0 when `x` is empty.
`T` and `F`	`TRUE` and `FALSE`	Ordinary variables that any script can overwrite.
`sapply()`	`vapply()` with the type declared	The type of the result depends on the input.

(9.) When joining data tables, state the expected relationship, so that duplicated or unmatched keys stop the script instead of adding or losing rows:

```
annotated <- readings |>
  inner_join(samples, by = c("plate", "well"),
             relationship = "one-to-one", unmatched = "error")
```

(10.) Check for missing values and decide what they mean before filtering or summarizing; do not pass `na.rm = TRUE` by default. Report how many rows each filter removes.

```
stopifnot("signal has missing values" = !anyNA(annotated$signal))

n_before <- nrow(annotated)
filtered <- drop_low_signal(annotated, params$min_signal)
message("low signal: removed ", n_before - nrow(filtered), " of ", n_before, " rows")
```

(11.) Write each function in `R/` so that it takes its data as arguments and returns a value. It reads no variable from the calling script and writes no file.

```
# R/preprocess.R
drop_low_signal <- function(data, min_signal) {
  dplyr::filter(data, signal >= min_signal)
}
```

(12.) State what must hold for the derived outputs as assertions in `scripts/99_checks.R`:

```
# 99_checks.R - assertions over the derived outputs
filtered  <- readr::read_rds(here::here("data", "processed", "filtered.rds"))
estimates <- readr::read_csv(here::here("output", "tables", "02_model_summary.csv"))

stopifnot(
  "a reading appears twice"   = !anyDuplicated(filtered[c("plate", "well")]),
  "an estimate is not finite" = all(is.finite(estimates$estimate))
)
```

Critical: Run after every code change. When a check fails, fix the analysis or record why the check is expected to fail. Do not loosen a check to make it pass. Remove a check that cannot fail.

›› *Running the pipeline*

(1.) Create the folders for derived files and run the numbered scripts from the project root, in order:

```
mkdir -p data/processed output/figures output/tables

Rscript scripts/01_filter.R
Rscript scripts/02_model.R
Rscript scripts/03_figures.R
Rscript scripts/99_checks.R
```

(2.) *Optional:* When tracking stale outputs by hand becomes error-prone, let a workflow manager do it: `targets` for R, `snakemake` for pipelines in several languages.

From exploration to pipeline

(1.) Keep each file in `notebook/` to one of three purposes.

- To explore: a dated notebook (Quarto, R Markdown, or Jupyter), `notebook/eda_<date>_<question>.qmd`, with the question above each chunk and the observation below it.

````
## Do the edge wells read low?

```{r}
readings |> group_by(edge = row %in% c("A", "D")) |> summarize(mean(signal))
```

Observation: the edge rows read lower than the rest of the plate.
````

- To record: a running log, `notebook/_log.md`, with dated entries stating the question, what was run, what was found or retracted, and what comes next (Noble, 2009).
- To report: `notebook/report.qmd`, which combines narrative and analysis.

(2.) *Critical:* Render a notebook from the project root before closing or sharing it, and repair or delete what fails. Rendering starts a fresh session and writes to `output/`.

```
quarto render notebook/eda_2026-05-12_edge-wells.qmd --output-dir ../output
```

(3.) Move settled code from a notebook into a numbered script as soon as something else depends on its result (another notebook, a figure, a statement in the log), or as soon as it is slow enough to be skipped on a rerun. The script saves an intermediate output to `data/processed/`; the notebook reads it.

(4.) Copy a function that a second project needs into that project, with its origin and commit in the header.

```
# R/read_platereader.R - copied from my-project, commit 3f2a9c1
```

When a third project needs the code, turn it into a package (`usethis::create_package()`) with tagged versions, and install a named tag in each project. From the R console:

```
remotes::install_github("my-lab/platereader@v0.2.0")
```

Capturing the environment

(1.) Record the R version, the packages, and the system libraries the analysis needs.

- With Conda, re-export and re-lock after every change to the installed packages, and commit `environment.yml` and `conda-lock.yml` SOP 0042.
- Without Conda, or with packages from CRAN or GitHub, use `renv` to take a snapshot. Commit `renv.lock`, `.Rprofile`, `renv/activate.R`, and `renv/settings.json`.

```
renv::init()       # creates renv/library/ and renv.lock
renv::snapshot()   # writes the current state to renv.lock

# on another machine:
renv::restore()    # rebuilds the library from renv.lock
```

- Without a lockfile, write a `sessionInfo()` record at the end of each script.

```
writeLines(capture.output(utils::sessionInfo()), here::here("session_info-02_model.txt"))
```

›› *Documenting the project*

(1.) Write a `README.md` at the project root covering what the analysis does, how to recreate and run it, where the data comes from, and which files are not in the repository.

````
# my-project

Bootstrap confidence intervals for binding affinity from titration curves (Expt 042, 2026).

## Recreate the environment

```bash
conda-lock install --name my-project conda-lock.yml
mamba activate my-project
```

## Run the pipeline

From the project root:

```bash
mkdir -p data/processed output/figures output/tables
Rscript scripts/01_filter.R
Rscript scripts/02_model.R
Rscript scripts/03_figures.R
Rscript scripts/99_checks.R
```

## Data

Provenance in `data/raw/README.md`; sample identity in `data/raw/samples.csv`.

## Files not in git

| pattern                       | where it lives        | how to get it back  |
|-------------------------------|-----------------------|---------------------|
| `data/raw/**/*.tif`           | lab file share        | copy from the share |
| `data/external/`              | sources in its README | fetch again         |
| `data/processed/`, `output/` | local disk only       | rerun the pipeline  |
````

›› *Case study: updating an upstream function mid-analysis*

(1.) In a project under Git, open a new branch. Keep a copy of the output the edit should change, edit the function in its file under `R/`, and rerun the affected scripts.

```
git checkout -b fix-preprocess
cp data/processed/filtered.rds data/processed/filtered.rds.bak
# ... edit R/preprocess.R, then rerun the pipeline ...
```

Critical: Edit in place; Git holds the old version. A `drop_low_signal_v2()` alongside the original becomes dead code, and which of the two is active depends on what was sourced last.

(2.) Compare the regenerated output with the copy taken before the edit.

```
waldo::compare(
  readr::read_rds(here::here("data", "processed", "filtered.rds.bak")),
  readr::read_rds(here::here("data", "processed", "filtered.rds"))
```

```
)
```

(3.) Delete the copy, commit the function edit, and merge the branch SOP 0101.

```
rm data/processed/*.bak
git add R/preprocess.R
git commit -m "drop_low_signal(): exclude readings equal to the threshold"
```

List of references

K.W. Broman and K.H. Woo, *Am. Stat.* **72**(1), 2-10 (2018).

W. Noble, *PLoS Comput. Biol.* **5**(7), e1000424 (2009).

Resources (available online) Notes (available online)

Lab Protocols — © B. C. Buchmuller. Licensed CC-BY-SA-4.0. Research use; provided as-is, without warranty. Current when printed. Visit https://benjbuch.github.io/check/?q=9bdfa37 or scan the QR code to read the full protocol or to check for updates.