Week 01: Foundations & Reproducibility

Introduction to reproducible workflows (R + Quarto)

Bartosz Maćkiewicz

Reproducible workflows: why are we doing this?

In this course you will conduct statistical tests, fit models, compute effect sizes and confidence intervals, and run simulations.

In research a result is not just a number - it is an **argument** supported by data, code, and a chain of decisions.

A reproducible workflow is how you keep that argument inspectable. It makes it possible to **check**, **debug**, **learn from**, and **build on** an analysis without relying on memory or trust.

It also has a very practical payoff:

- you spend less time “reconstructing what happened”
- you catch mistakes earlier (often during rendering)
- collaboration becomes easier (your future self is a collaborator, too)

Reproducibility vs replication (two different problems)

People use similar words for different things, so let's be explicit.

- **Reproducibility** (computational): same data + same code + same environment $\Rightarrow$ the same numbers/figures/tables.
- **Replication** (empirical): new data from the same design $\Rightarrow$ a similar conclusion (within uncertainty).

We will focus on **computational reproducibility** because it is (a) under your control and (b) a prerequisite for doing serious replication work.

Why reproducibility matters methodologically

Statistical analysis is often full of reasonable choices:

- inclusion/exclusion rules,
- outlier handling, transformations,
- coding decisions,
- model specification,
- reporting decisions.

Different choices can lead to different results — sometimes substantially.

A reproducible workflow does not guarantee that choices are “correct”, but it makes them **visible** and therefore discussable:

- you can justify decisions (and reconsider them later)
- others can verify what you did instead of guessing
- you can separate disagreement about theory from disagreement about analysis

Open science: reproducible workflows are the mechanics

Open science is set of practices that make research easier to evaluate and reuse. A reproducible workflow is not the whole of open science, but it is the part of it that make open practices feasible.

In practice, openness usually means some combination of:

- sharing data (when ethically and legally possible)
- sharing analysis code and materials
- writing analyses so they can be rerun end-to-end
- documenting decisions so others can understand what happened

Even when you cannot share raw data (e.g., privacy constraints), you can often still share **code**, **analysis decisions**, and **simulated examples** that make your work more transparent.

What “reproducible” means in this course

The goal is not only to get *an answer*, but to be able to **re-create** the answer later.

In this course, “reproducible” means:

- someone else can run your code and get the same results, and
- *you* can rerun it in 2 months and understand what you did.

Because of that deliverables in this course are **Quarto source files** (`.qmd`) that render successfully.

Course workflow (high level)

Majority of the analysis we do follows the same pipeline:

1. Import
2. Tidy / transform
3. Visualize
4. Model
5. Diagnose / validate
6. Report (tables, figures, written interpretation)

Using reproducible workflow (such as Quarto) makes sure that results have a clear origin: every figure and number in the report is traceable to a specific sequence of steps in the workflow.

Tools we use

We will rely on a small set of tools

- **R** for data analysis and simulation
- **RStudio** as the development environment (recommended)
- **Quarto** for reproducible reports and worksheets (`.qmd`)
- (Optional) **Git** for version control (recommended if you do research projects)

What is a `.qmd`?

Quarto is a modern authoring system for data analysis reports. A Quarto document (`.qmd`) integrates text, code, and the results of code execution (tables, printouts, figures, etc.) into a single, reproducible report.

A Quarto document combines:

- Markdown text
- R code chunks
- Output (plots, tables, printed results)

Markdown is just plain text

Quarto documents utilize Markdown syntax. You do not need to memorize everything — you only need the basics.

```
# This is a top-level header
## This is a second-level header

*This text is italic*, **this text is bold**.

1. Ordered lists are automatically numbered.
* You can create hierarchical lists
  * by indenting
```

Quarto code chunks (example)

Embedding R code in Quarto documents is straightforward:

```r
set.seed(1)
x <- rnorm(10)
mean(x)
```

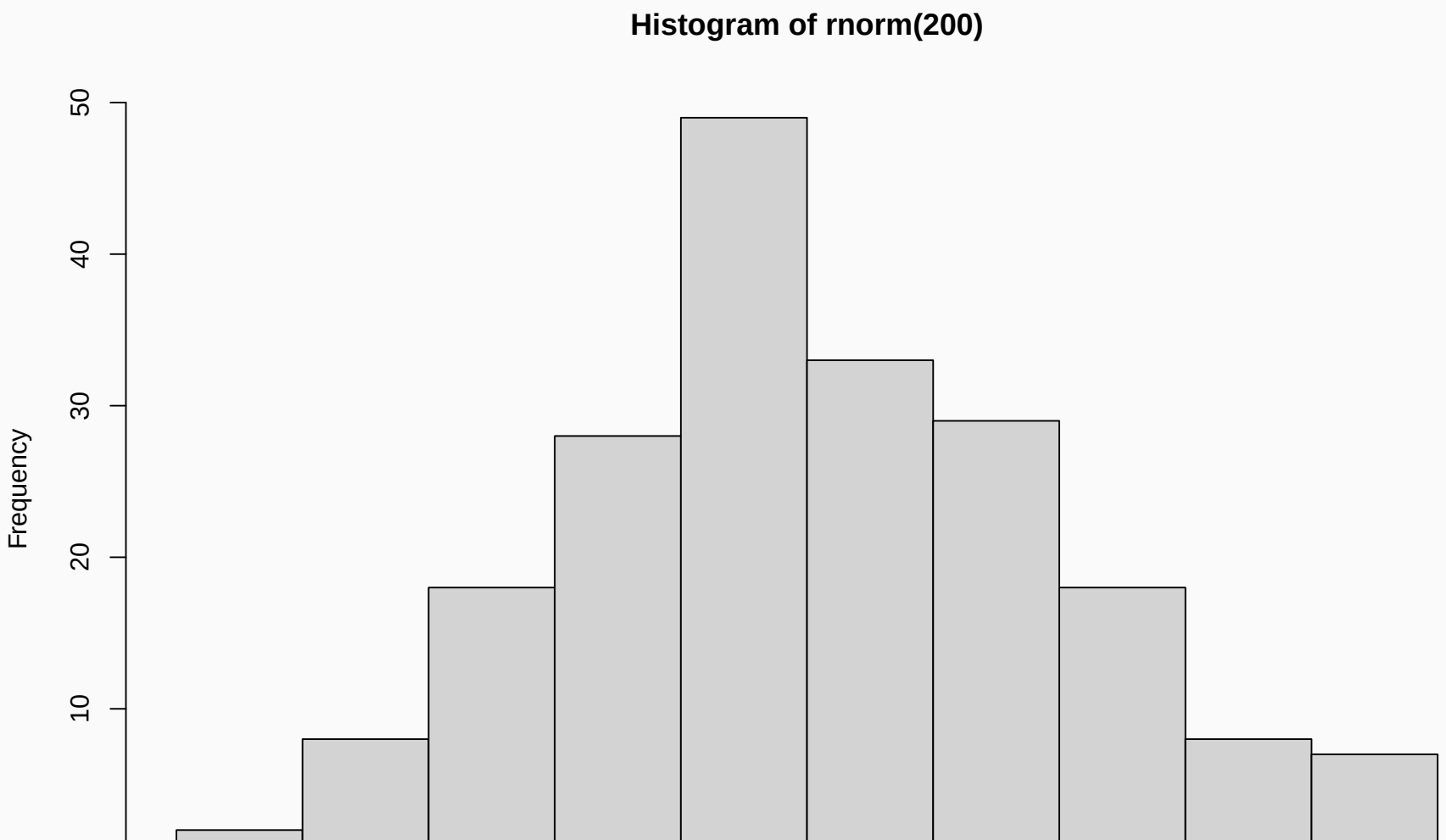
```
[1] 0.1322028
```

The key point is that the code is run during rendering, and the output becomes part of the document. This turns “notes + code” into a reproducible analysis.

Chunk options (#|) are your “presentation controls”

Chunk options let you decide what a reader sees. The recommended way is to use #| lines inside the chunk:

```
hist(rnorm(200))
```



Rendering (the one rule)

Your submission must **render** without errors.

Rendering a Quarto document involves executing R code in code blocks and compiling the document into HTML/PDF format together with the output.

- In RStudio: open the `.qmd` file and click **Render**
- In the terminal:

```
quarto render your_file.qmd
```

Rendering is also your best reproducibility test: it forces a clean "start-to-finish" run of what you wrote.

Working directory during rendering (important)

When rendering a Quarto document, the working directory is set to the location of the `.qmd` file. This detail is crucial.

So if your data is next to the `.qmd`, you can do:

```
read.csv("data.csv")
```

But if you accidentally rely on files elsewhere (or on your interactive session state), rendering will fail.

Relative paths (portability)

Use relative file paths. Avoid absolute paths like `C:\\Users\\...\\Desktop\\...` — they will not work on another computer.

If you keep a project folder with everything inside it, relative paths are simple and stable:

```
read.csv("data/data.csv")
```

Avoid these common mistakes

These are common reasons a file renders on your machine but fails elsewhere:

- Do **not** use `setwd()` in your submission (it hard-codes *your* folder structure).
- Do **not** run `install.packages()` inside the `.qmd` (installation is not analysis, and it can fail during rendering).
- Do **not** hardcode absolute paths (they will not work when someone else renders).
- Do **not** rely on objects created “in the console but not in the file” (rendering starts from a clean session).

“It works in the console, but rendering fails” — why?

This is one of the most common problems. Usually the reason is simple:

a) Is there any `install.packages()` function call? If yes, remove it.

b) Are files (for example read by `read.csv`) in the right directory? Remember: during rendering, the working directory is set to the directory where the rendered `.qmd` file is located, not to whatever you had in an interactive session.

c) Are all variables defined in your code? Maybe you defined something outside the file (in console) and forgot to put the code in the file?

Besides that, read error messages, they do not bite.

Randomness and reproducibility (seeds)

Some workflows involve randomness: simulations, bootstrapping, random splits, and (later in the course) some model-fitting procedures.

If your results depend on random numbers, your document may change each time you render it. That is not necessarily “wrong”, but it makes debugging and verification harder. When you want stable output, set a seed before the random part:

```
set.seed(123)
x <- rnorm(5)
x
```

```
[1] -0.56047565 -0.23017749  1.55870831  0.07050839  0.129287
```

Reproducibility checklist (before you upload)

1. Restart R session (Session → Restart R)
2. Open your `.qmd`
3. Render
4. Confirm the output shows:
    - code (if requested),
    - plots/tables,
    - your written answers

If something fails, fix the `.qmd` until rendering succeeds from a clean session.

What we'll do in the lab today

1. Create a new folder for Week 1 work
2. Open a provided Quarto worksheet (`.qmd`)
3. Import a small dataset from CSV using a relative path
4. Render to HTML (and optionally PDF)

Optional / Appendix: Git as a “time machine” for your analysis

You can do this course without Git, but Git becomes extremely valuable once you work on longer projects (thesis, lab projects, papers).

Version control helps because it makes changes explicit:

- you can go back to an earlier version of your analysis
- you can see what changed and when
- you can collaborate without emailing files named `final_final_v3.qmd`

If you already use Git: great. If not, you can treat it as an optional upgrade once the basics (Quarto + projects + rendering) feel natural.