Base-R–First, Workflow–First: A Minimal-Tooling Pedagogy for Teaching Probability with R

Martin Summer (notes drafted with ChatGPT)

January 17, 2026

Context. These notes respond to a pedagogical question motivated by my January bootcamp course in probability with R for a mixed cohort in quantitative and computational finance:

`martin-summer-1090.github.io/Probability_Introduction/`.

The key issue is how to update the course to reflect current best practice in teaching R and RStudio, while avoiding a rigid "tidyverse-first" approach and instead emphasizing principles, computational thinking, and minimal tooling.

Position in one sentence

A *workflow-first* course design (Quarto, reproducibility, projects, simulation-as-thinking) is the genuinely modern part; a *tidyverse-first* syntax is optional. In fact, a *base-R-first* approach can be more future-proof today, especially in the age of LLM-assisted coding.

Why base-R-first makes even more sense now (especially with LLMs)

If "syntax is cheap," then the scarce skills students must develop are not primarily about remembering verbs or piping rules, but about reasoning:

- **Computational problem formulation:** mapping a question into inputs → transformations → outputs.
- **Simulation literacy:** seeds, replication, Monte Carlo error, convergence intuition.
- **Verification habits:** sanity checks, invariants, dimensions, edge cases.
- **Code comprehension:** reading and interrogating code (including code generated by an LLM).
- **Reproducible practice:** organizing work so results can be rerun, shared, and audited.

Base R supports these goals cleanly because it keeps attention on objects, data structures, and semantics rather than on a parallel "verb universe."

A modern but minimal "tooling" stack (not a Hadleyverse commitment)

Minimalism need not mean a 2005 workflow. A lean, contemporary setup can be:

1. **Quarto project** as the course spine (notes, exercises, solutions, labs).
2. **Base R** for the computational core: `read.csv`, indexing `[ ]`, `subset`, `aggregate`/`tapply`, `apply`, `replicate`, the RNG family (`rbinom`, `runif`, `rnorm`, ...).
3. `renv` to freeze package versions (even with few packages, it prevents January chaos).
4. **Optional Git/GitHub** for distributing templates and collecting assignments (even if students do not love Git, seeing it once is valuable).

Crucially, none of this requires a tidyverse-first curriculum.

A base-R-first structure for a probability bootcamp course

For a mixed audience in a bootcamp phase, a coherent base-R curriculum can be organized around three pillars.

Pillar A: Simulation as the primary microscope

Lean hard into simulation early:

- `set.seed()`, `sample()`, `rbinom()`, `runif()`, `rnorm()` as foundational tools.
- `replicate()` and vectorization; explicit loops when they improve clarity.
- Monte Carlo error as a first-class concept: repeated simulations to quantify noise and stability.

This aligns with the core probabilistic ideas (LLN/CLT) and builds computational confidence quickly.

Pillar B: "Thinking in objects" and data structures

Students should become fluent in what R *is*:

- atomic vectors vs. lists vs. matrices vs. data frames;
- indexing as a superpower: `x[i]`, `x[x > 0]`, `x[order(x)]`, `which()`.
- writing small functions early: `f <- function(...) { ... }`.

This supports both probability content (random variables as objects) and later professional work.

Pillar C: Verification habits (the under-taught skill)

The "LLM era" upgrade is to teach students how to trust results appropriately:

- invariants (probabilities sum to one; expectations in bounds; monotonicity);
- dimension checks (`stopifnot()`, `length()`, `dim()`);
- compare simulation to known results (LLN/CLT as unit tests for reasoning).

Students will generate code; the course teaches them to *audit* it.

Where the tidyverse can still appear (without becoming a religion)

A pragmatic compromise is:

- **Base track** as the core course (required).
- **Tidy track** as an appendix or optional translation layer: "here is the same idea in a pipeline style."

This frames tidyverse as a convenience layer and avoids turning it into "what R is."

One cutting-edge, minimal addition: "LLM-generated code audit" exercises

A high-payoff exercise format is to give students a short code snippet (possibly produced by an LLM) and ask:

1. What is it trying to compute?
2. What assumptions does it make (distributional, independence, parameterization, scaling)?
3. What checks would you add (invariants, dimensions, edge cases)?
4. Can you construct a counterexample where it breaks or misleads?

This trains transferable competence: reading code, validating results, and reasoning under uncertainty.

Closing remark

In short: keep the modern *workflow* (Quarto, reproducibility, project structure), keep the *thinking* (simulation and verification), and keep the *tooling minimal*. Base R is not a retreat from modernity—it can be the cleanest way to teach durable principles.