Best Practices for Bayesian Analysis in R

Key Changes to Implement in Your Workflow

2026-01-14

Table of contents

1 Overview

This document summarizes key software development practices adapted for doing Bayesian analysis in R, particularly for scientific learning and research. These practices will make your work more reproducible, maintainable, and professional.

2 Essential Tools & Setup

2.1 VS Code Extensions

- **R Extension** - Core R support
- **R LSP Client** (languageserver) - Better code completion and help
- **R Debugger** - For troubleshooting

- **Jupyter** - For notebook support
- **Quarto** - Modern notebook experience (recommended)
- **httpgd** - Interactive R plots in VS Code

2.2 Key R Packages for Bayesian Work

- **cmdstanr** - Faster and more modern than rstan
- **brms** - User-friendly interface to Stan via R formulas
- **bayesplot** - Visualization for Bayesian models
- **posterior** - Working with posterior draws
- **tidybayes** - Tidy workflow for Bayesian models
- **renv** - Package version management

3 Version Control (Essential)

3.1 Git Basics

- Initialize Git in every project: `git init`
- Commit often with meaningful messages
- Good: "add prior sensitivity analysis for sigma parameter"
- Bad: "updates" or "fix"
- Use GitHub for backup and potential collaboration

3.2 What to Track

- **Track:** All code, scripts, documentation, README files
- **Don't track:** Large data files, fitted models, output files
- Create a `.gitignore` file for `data/raw/`, `models/fitted/`, `output/`

3.3 Benefits

- Experiment fearlessly - you can always go back
- See what changed when results differ
- Your future self will thank you

4 Project Structure

4.1 Recommended Folder Organization

```
project-name/
  README.md
  project-name.Rproj
  renv.lock
  data/
    raw/ # Never modify!
    processed/
    simulated/
  scripts/
    00_setup.R
    01_data_cleaning.R
    02_exploratory.R
    03_fit_models.R
    04_visualize_results.R
  R/ # Custom functions
    plotting_functions.R
    model_helpers.R
  models/
    model_v1.stan
    fitted/ # Saved model objects
  notebooks/ # Quarto/R Markdown
  output/
    figures/
    tables/
    reports/
  docs/
```

4.2 Key Principles

- Use R Projects (`.Rproj`) - sets working directory automatically
- Number scripts if they run in sequence (01, 02, 03…)
- Keep raw data immutable - all processing in scripts
- Save fitted models - MCMC is expensive to rerun

5 Reproducibility Practices

5.1 Package Management

```
# Initialize renv in your project
renv::init()

# After installing/updating packages
renv::snapshot()

# To restore exact package versions later
renv::restore()
```

5.2 Random Seeds

Always set seeds before stochastic operations:

```
set.seed(12345)
fit <- brm(...)
```

5.3 Relative Paths

Never do this:

```
setwd("C:/Users/YourName/Documents/project")
```

Instead: Use R Projects and relative paths:

```
data <- read_csv("data/raw/experiment.csv")
```

5.4 Document Your Environment

At the top of key scripts:

```
# R version: 4.4.0
# Platform: macOS Sonoma
# Key packages: brms 2.21.0, cmdstanr 0.8.0
# Date: 2025-01-11
```

6 Code Quality

6.1 Write Functions

Instead of copy-pasting:

```
# Repeated code...
plot(prior_predictive_1)
plot(prior_predictive_2)
plot(prior_predictive_3)
```

Write a function:

```
plot_prior_predictive <- function(model, title) {
pp_check(model, type = "dens_overlay") +
ggtitle(title)
}

plot_prior_predictive(model1, "Simple Model")
plot_prior_predictive(model2, "Hierarchical Model")
```

6.2 Meaningful Names

- **Good:** `prior_std`, `n_iterations`, `learning_rate_mean`
- **Bad:** `ps`, `n`, `x`, `thing`, `temp`

6.3 Code Style

Follow the tidyverse style guide:

```
# Use styler package to auto-format
styler::style_file("scripts/01_data_cleaning.R")
```

Key points:

- Use `<-` for assignment, not `=`
- Spaces around operators: `x <- 5`, not `x<-5`
- Indent with 2 spaces
- Line length < 80 characters when possible

6.4 Comments

Explain **why**, not **what**:

```
# Bad comment
# Set prior to normal(0, 1)
prior <- normal(0, 1)

# Good comment
# Using weakly informative prior centered at 0
# based on domain knowledge that effect sizes
# in this field are typically < 2 standard deviations
prior <- normal(0, 1)
```

7 Testing & Validation

7.1 Sanity Checks

Add assertions to catch problems early:

```
# Check data assumptions
stopifnot(all(df$age > 0))
stopifnot(all(df$temperature < 100))
stopifnot(!any(is.na(df$key_variable)))
```

7.2 Simulation-Based Calibration

Test your models by simulating fake data:

```
# 1. Simulate data from known parameters
true_params <- list(alpha = 2, beta = 0.5, sigma = 1)
sim_data <- simulate_data(true_params, n = 100)

# 2. Fit model to simulated data
fit <- brm(y ~ x, data = sim_data)

# 3. Check if you recover true parameters
posterior_summary(fit)
```

7.3 Prior & Posterior Predictive Checks

Always visualize:

```
# Prior predictive check
pp_check(fit, type = "dens_overlay", ndraws = 100,
prefix = "ppd")

# Posterior predictive check
pp_check(fit, type = "dens_overlay", ndraws = 100)
```

8 Documentation

8.1 README.md Template

```
# Project Title

## Overview
Brief description of the analysis and research question.

## Data
- Source: Where did the data come from?
- Files: What's in data/raw/?
- Processing: See scripts/01_data_cleaning.R

## Analysis Pipeline
1. Run scripts/01_data_cleaning.R
2. Run scripts/02_exploratory.R
3. Run scripts/03_fit_models.R
4. Render notebooks/analysis_report.qmd

## Key Results
- Main finding 1
- Main finding 2

## Dependencies
- R version 4.4.0
- See renv.lock for package versions
```

8.2 Document Modeling Decisions

Create `docs/modeling_decisions.md`:

```
# Model 1: Simple Linear Model

**Why this model?**
Starting simple to establish baseline.

**Prior choices:**
- beta ~ normal(0, 2): Based on pilot data showing
effects typically between -3 and 3
- sigma ~ exponential(1): Weakly informative,
allows wide range of residual variance

**Limitations:**
- Assumes independence (ignores clustering)
- Linear relationship may be too restrictive
```

9 Incremental Development

9.1 Start Simple

1. **Model v1:** Simple linear model, basic priors
2. **Model v2:** Add hierarchical structure
3. **Model v3:** Add non-linear terms or interactions
4. **Model v4:** Final model with all features

Version your model files:

- `models/model_v1_simple.stan`
- `models/model_v2_hierarchical.stan`
- `models/model_v3_nonlinear.stan`

9.2 Compare Models

```
# Fit multiple models
fit1 <- brm(y ~ x, data = df, file = "models/fitted/fit1")
fit2 <- brm(y ~ x + (1|group), data = df,
file = "models/fitted/fit2")
```

```r
# Compare
loo_compare(loo(fit1), loo(fit2))
```

10 Automation

10.1 Setup Script

Create `scripts/00_setup.R`:

```r
# Load packages
library(brms)
library(cmdstanr)
library(tidyverse)
library(bayesplot)
library(posterior)

# Source custom functions
source("R/plotting_functions.R")
source("R/model_helpers.R")

# Set options
options(mc.cores = parallel::detectCores())
theme_set(theme_minimal())

# Plotting defaults
bayesplot_theme_set(theme_minimal())
```

10.2 Reusable Diagnostic Functions

Create `R/diagnostics.R`:

```r
run_all_diagnostics <- function(fit, model_name) {
# Convergence
print(paste("Rhat diagnostics for", model_name))
print(max(rhat(fit)))

# Effective sample size
print(paste("Min ESS for", model_name))
print(min(ess_bulk(fit)))
```

```r
# Trace plots
p1 <- mcmc_trace(fit)
ggsave(paste0("output/figures/trace_", model_name, ".png"),
p1, width = 10, height = 6)

# Posterior predictive
p2 <- pp_check(fit, ndraws = 100)
ggsave(paste0("output/figures/ppcheck_", model_name, ".png"),
p2, width = 8, height = 6)
}
```

11 Quick Wins to Implement First

11.1 Week 1

1. Set up Git and make your first commit
2. Reorganize current project with folder structure above
3. Create a README.md for your project
4. Initialize renv

11.2 Week 2

1. Write a setup script (00_setup.R)
2. Number your analysis scripts in order
3. Add sanity checks to your data cleaning
4. Document one modeling decision in docs/

11.3 Week 3

1. Write one custom function for repeated code
2. Run styler on your scripts
3. Add prior and posterior predictive checks
4. Create your first Quarto document

11.4 Ongoing

- Commit to Git after each working session
- Add comments explaining modeling choices
- Run diagnostics on every model
- Keep README updated

12 Resources

12.1 Learning Bayesian Analysis

- **Statistical Rethinking** by Richard McElreath (book + lectures)
- **Bayes Rules!** by Johnson, Ott, and Dogucu
- **Stan documentation:** https://mc-stan.org/users/documentation/

12.2 R & Workflow

- **R for Data Science:** https://r4ds.hadley.nz/
- **Happy Git with R:** https://happygitwithr.com/
- **Tidyverse style guide:** https://style.tidyverse.org/
- **brms documentation:** https://paul-buerkner.github.io/brms/

12.3 VS Code with R

- **VS Code R extension docs:** Search "VS Code R" for latest docs
- **Quarto documentation:** https://quarto.org/

Remember: You don't need to implement everything at once. Pick one or two practices to focus on each week. The goal is sustainable improvement, not perfection.