Code Publication

Last modified: 2026-10-31 04:53:06 (PDT)

Checklist overview

1. Fill out file headers
2. Clean up comments
3. Document functions
4. Remove deprecated filepaths
5. Ensure project runs via bash
6. Complete the README
7. Clean up feature branches
8. Create GitHub release

Fill out file headers

Every file in a project should have a header that allows it to be interpreted on its own. It should include the name of the project and a short description for what this file (among the many in your project) does specifically. See template here.

Clean up comments

Make sure comments in the code are for code documentation purposes only. Do not leave comments to self in the final script files.

Document functions

Every function you write must include a header to document its purpose, inputs, and outputs. See template for the function documentation here.

Remove deprecated filepaths

All file paths should be defined in 0-config.R, and should be set relative to the project working directory. All absolute file paths from your local computer should be removed, and replaced with a relative path. If a third party were to re-run this analysis, if they need to download data from a separate source and change a filepath in the 0-config.R to match, make sure to specify in the README which line of 0-config.R needs to be substituted.

Ensure project runs via bash

The project should be configured to be entirely reproducible by running a master bash script, run-project.sh, which should live at the top directory. This bash script can call other bash scripts in subfolders, if necessary. Bash scripts should use the runFileSaveLogs utility script, which is a wrapper around the Rscript command, allowing you to specify where .Rout log files are moved after the R scripts are run.

See usage and documentation here.

Complete the README

A `README.md` should live at the top directory of the project. Include a repository status badge from repostatus.org near the top of the file to communicate whether the codebase is actively maintained, a work in progress, or an archived analysis (see Repository Status Badges). This file also includes a Project Overview and a Directory Structure, along with contributor names and the project license. See below for a template:

> **Overview**
>
> To date, coronavirus testing in the US has been extremely limited. Confirmed COVID-19 case counts underestimate the total number of infections in the population. We estimated the total COVID-19 infections – both symptomatic and asymptomatic – in the US in March 2020. We used a semi-Bayesian approach to correct for bias due to incomplete testing and imperfect test performance.
>
> **Directory structure**
>
> - 0-config.R: configuration file that sets data directories, sources base functions, and loads required libraries
> - 0-base-functions: folder containing scripts with functions used in the analysis
>   - 0-base-functions.R: R script containing general functions used across the analysis

  - 0-bias-corr-functions.R: R script containing functions used in bias correction
  - 0-bias-corr-functions-undertesting.R: R script containing functions used in bias correction to estimate the percentage of underestimation due to incomplete testing vs. imperfect test accuracy
  - 0-prior-functions.R: R script containing functions to generate priors
- 1-data: folder containing data processing scripts NOTE: some scripts are deprecated
- 2-analysis: folder containing analysis scripts. To rerun all scripts in this subdirectory, run the bash script 0-run-analysis.sh.
  - 1-obtain-priors-state.R: obtain priors for each state
  - 2-est-expected-cases-state.R: estimate expected cases in each state
  - 3-est-expected-cases-state-perf-testing.R: estimate expected cases in each state, estimate the percentage of underestimation due to incomplete testing vs. imperfect test accuracy
  - 4-obtain-testing-protocols.R: find testing protocols for each state.
  - 5-summarize-results.R: summarize results; obtain results for in text numerical results.
- 3-figure-table-scripts: folder containing figure scripts. To rerun all scripts in this subdirectory, run the bash script 0-run-figs.sh.
  - 1-fig-testing.R: creates plot of testing patterns by state over time
  - 2-fig-cases-usa-state-bar.R: creates bar plot of confirmed vs. estimated infections by state
  - 3a-fig-map-usa-state.R: creates map of confirmed vs. estimated infections by state
  - 3b-fig-map-usa-state-shiny.R: creates map of confirmed vs. estimated infections by state with search functionality by state
  - 4-fig-priors.R: creates figure with priors for US as a whole
  - 5-fig-density-usa.R: creates figure of distribution of estimated cases in the US
  - 6-table-data-quality.R: creates table of data quality grading from COVID Tracking Project

  - 7-fig-testpos.R: creates figure of the probability of testing positive among those tested by state
  - 8-fig-percent-undertesting-state.R: creates figure of the percentage of under estimation due to incomplete testing
- 4-figures: folder containing figure files.
- 5-results: folder containing analysis results objects.
- 6-sensitivity: folder containing scripts to run the sensitivity analyses

Contributors: UCD-SeRG team (adapted from original contributors: Jade Benjamin-Chung, Sean L. Wu, Anna Nguyen, Stephanie Djajadi, Nolan N. Pokpongkiat, Anmol Seth, Andrew Mertens)

Wu SL, Mertens A, Crider YS, Nguyen A, Pokpongkiat NN, Djajadi S, et al. Substantial underestimation of SARS-CoV-2 infection in the United States due to incomplete testing and imperfect test accuracy. medRxiv. 2020; 2020.05.12.20091744. doi:10.1101/2020.05.12.20091744

When possible, also include a description of the RDS results that are generated, detailing what data sources were used, where the script lives that creates it, and what information the RDS results hold.

Clean up feature branches

In the remote repository on GitHub, all feature branches aside from master should be merged in and deleted. All outstanding PRs should be closed.

Create GitHub release

Once all of these items are verified, create a tag to make a GitHub release, which will tag the repository, creating a marker at this specific point in time.

Detailed instructions: Managing releases in a repository.

Releasing to CRAN

If your R package is intended for public distribution, you may want to release it on CRAN (the Comprehensive R Archive Network). This section summarizes the release process and adds guidance on versioning and tagging. For full details, see the R Packages book chapter on releasing to CRAN by Hadley Wickham and Jennifer Bryan.

The CRAN submission pipeline goes through several stages: initial upload, automated checks, human review by CRAN maintainers, and final acceptance or rejection. For a visual overview of the full pipeline, see the CRAN stages diagram by Edgar Ruiz.

Versioning

R package versions follow a `major.minor.patch` scheme (e.g., `1.2.3`). Use a fourth development component (e.g., `1.2.3.9000`) on the development version to distinguish it from the released version.

- Increment the **patch** version (`1.2.3` → `1.2.4`) for bug fixes
- Increment the **minor** version (`1.2.3` → `1.3.0`) for new features that are backward-compatible
- Increment the **major** version (`1.2.3` → `2.0.0`) for breaking changes

Update the version in `DESCRIPTION` before each release.

Pre-release Checklist

Before submitting to CRAN, work through the following steps:

1. **Update `NEWS.md`**: Document all changes since the last release in `NEWS.md`. Use `usethis::use_news_md()` to create this file if it doesn't exist. Each release should have a top-level heading (e.g., `# mypackage 1.2.0`).
2. **Run `devtools::check()`**: Ensure your package passes `R CMD CHECK` with no errors, warnings, or notes. Use `cran = TRUE` to enable the stricter checks that more closely mirror what CRAN runs. Address any issues before submitting.

   ```
   devtools::check(cran = TRUE)
   ```

3. **Check spelling**: Run `devtools::spell_check()` to catch typos in documentation.

   ```
   devtools::spell_check()
   ```

4. **Check URLs**: Run `urlchecker::url_check()` to catch broken or redirected URLs in your documentation, which CRAN flags.

```
urlchecker::url_check()
```

5. **Test on multiple platforms**: CRAN requires packages to work across operating systems and R versions. Use the following to test on additional platforms:

   - `devtools::check_win_devel()` - tests on Windows with the development version of R
   - R-hub - tests on a wide range of platforms. In rhub v2, this runs via GitHub Actions: run `rhub::rhub_setup()` once to add the workflow to your repository, then `rhub::rhub_check()` to trigger a check run. (If you previously used older rhub APIs, see the rhub v2 documentation for migration details.)

6. **Review CRAN policies**: Ensure your package complies with the CRAN Repository Policy.

Submitting to CRAN

Once all checks pass, submit with:

```
devtools::submit_cran()
```

This bundles your package and uploads it to CRAN's submission portal. You will receive an email asking you to confirm the submission. Review time varies; watch for email from CRAN maintainers with acceptance or requested changes.

On the CRAN web submission form that `devtools::submit_cran()` opens, briefly describe any `R CMD CHECK` notes in the submission comments and explain why they are acceptable. Be concise and factual.

For first-time submissions, also check the new submission checklist in the R Packages book for additional requirements.

After CRAN Acceptance

Once CRAN accepts your package:

1. **Tag the release and create a GitHub release**: Follow the steps in Section , using a tag named `v` followed by the version number (e.g., `v1.2.0`). Copy the relevant section of `NEWS.md` as the release notes. Tags permanently mark the exact commit that was released, making it easy to return to or compare against any released version.

2. **Bump the development version**: After releasing, append the `.9000` development suffix to the current version in `DESCRIPTION`:

```
usethis::use_version("dev")
```

   For example, after releasing `1.2.0`, the `DESCRIPTION` version becomes `1.2.0.9000`. This signals to contributors that the repository reflects development work toward the next release.

3. **Commit the version bump**: Stage and commit the updated `DESCRIPTION` (and `NEWS.md` if you added a `# mypackage (development version)` header):

```
git add DESCRIPTION NEWS.md
git commit -m "Start development toward next release (1.2.0.9000)"
git push
```

4. **Track adoption over time**: the {Visualize.CRAN.Downloads} package plots a CRAN package's download counts (retrieved via {cranlogs}), producing static, interactive, and multi-package comparison plots, with moving averages and confidence bands to make trends visible. With no date range given, it defaults to the year of downloads ending today:

```
Visualize.CRAN.Downloads::processPckg("yourpackage")
```