

Lec 27 - More R Package Stuff

Statistical Programming

Sta 323 | Spring 2022

Dr. Colin Rundel

Package data

Exported data

Many packages contain sample data (e.g. `nycflights13`, `babynames`, etc.)

Generally these files are made available by saving a single data object as an `.Rdata` file (using `save()`) into the `data/` directory of your package.

- Use `usethis::use_data(obj)` to create the necessary files
- Data is usually compressed, for large data sets it may be worth trying different options (there is a 5 Mb package size limit on CRAN)
- Exported data must be documented (possible via Roxygen)

Lazy data

By default when attaching a package all of that package's data is loaded - however if `LazyData: true` is set in the package's `DESCRIPTION` then data is only loaded when used.

```
pryr::mem_used()
```

```
## 40.8 MB
```

```
library(nycflights13)  
pryr::mem_used()
```

```
## 50.4 MB
```

```
invisible(flights)  
pryr::mem_used()
```

```
## 91.1 MB
```

If you use `usethis::use_data()` this option will be set in `DESCRIPTION` automatically.

Raw data

The package should only contain the final data set, but it is important that the process to generate the data is documented as well as any necessary preliminary data.

- These can live anywhere but the general suggestion is to create a `data-raw/` directory which is included in `.Rbuildignore`
- Should contain scripts, data files, and anything else needed to generate the final object
- See examples `babynames` or `nycflights`
- Use `usethis::use_data_raw()` to create and ignore the `data-raw/` directory.

Internal data

If you have data that you want to have access to from within the package but not exported then it needs to live in a special Rdata object located at `R/sysdata.rda`.

- Can be created using `usethis::use_data(obj1, obj2, internal = TRUE)`
- Each call to the above will overwrite, so needs to include all objects
- No necessary for small data frames and similar objects - just create in a script. Use when you want the object to be compressed.
- Example `nflplotR` which contains team logos and colors for NFL teams.

Raw data files

If you want to include raw data files (e.g. `.csv`, shapefiles, etc.) there are generally placed in folders within `inst/` so that they are installed with the package.

- Accessed using `system.file("dir", package = "package")` after install
- Use folders to keep things organized, Hadley recommends and uses `inst/extdata/`
- Example `sf`

Advice for the final project

Your need for data within your final projects will vary depending on what you will be doing, but if your project involves collecting, cleaning, and or analyzing data then:

- If scraping or downloading data, your script and the raw data should go in `raw-data/`
 - If the process is complex feel free to use a Makefile here
 - It should be possible for us to rerun and recreate all of the data in this folder
- Any final data set that is being used in the vignette should be exported and documented

Package checking

R CMD check

Last time we saw the usage of `R CMD check`, or rather `Build > Check Package` from within RStudio.

This is a good idea to run regularly to make sure nothing is broken and you are meeting the important package quality standards, but this only in the context of your machine, your version of R, your OS, and so on.

If using GitHub it is highly recommended that you run

`usethis::use_github_action_check_standard()` to enable GitHub actions checks of the package each time it is pushed.

On each push this runs `R CMD check` on:

- Latest R on MacOS, Windows, Linux (Ubuntu)
- Previous and devel version of R on Linux (Ubuntu)

Package vigenette

Vignette

Long form documentation for your package that live in `vignette/`, use `browseVignette(pkg)` to see a package's vignettes.

- Not required, but adds a lot of value to a package
- Generally these are literate documents (`.Rmd`, `.Rnw`) that are compiled to `.html` or `.pdf` when the package is built.
- Built package retained the rendered document, the source document, and all source code
 - `vignette("colwise", package = "dplyr")` opens rendered version
 - `edit(vignette("colwise", package = "dplyr"))` opens code chunks
- Use `usethis::use_vignette()` to create a RMarkdown vignette template

Articles

These are an un-official extension to vignettes where package authors wish to include additional long form documentation that is included in their `pkgdown` site but not in the package (usually for space reasons).

- Use `usethis::use_article()` to create
- Files are added to `vignette/articles/` which is added to `.Rbuildignore`

Package testing

Basic test structure

Package tests live in `tests/`,

- Any R scripts found in the folder will be run when Checking the package (not Building)
- Generally tests fail on errors, but warnings are also tracked
- Testing is possible via base R, including comparison of output vs. a file but it is not recommended (See [Writing R Extensions](#))



testthat basics

Not the only option but probably the most widely used and with the best integration into RStudio.

Can be initialized in your project via `usethis::use_testthat()` which creates `tests/testthat/` and some basic scaffolding.

- `test/testthat.R` is what is run by the Check and runs your other tests - handles some basic config like loading package(s)
- Test scripts go in `tests/testthat/` and should start with `test_`, suffix is usually the file in R/ that is being tested.

`usethis::use_testthat()` has an `edition` argument, this is a way of maintaining backwards compatibility, generally always use the latest if starting a new project

testthat script structure

From the bottom up,

- a single test is written as an expectation (e.g. `expect_equal()`, `expect_error()`, etc.)
- multiple related expectations are combined into a test group (`test_that()`), which provides
 - a human readable name and
 - local scope to contain the expectations and any temporary objects
- multiple test groups are combined into a file

Live demo - fizzbuzz tests

Package publishing



CRAN

More details than we can get into today, but see the the Release a package chapter of R Pkgs for more details.

- `devtools::release()` will take you through a number of basic checks and if everything is good upload your package to the submission queue of CRAN.