R packages (/) by Hadley Wickham

Table of contents

Want a physical copy of this material? Buy from Amazon (http://amzn.com/1491910593?tag=r-pkgs-20)!

Contents

How to contribute (/contribute.html)

Edit this page (https://github.com/hadley/r-pkgs/edit/master/data.rmd)

External data

It’s often useful to include data in a package. If you’re releasing the package to a broad audience, it’s a way to provide compelling use cases for the package’s functions. If you’re releasing the package to a more specific audience, interested either in the data (e.g., NZ census data) or the subject (e.g., demography), it’s a way to distribute that data along with its documentation (as long as your audience is R users).

There are three main ways to include data in your package, depending on what you want to do with it and who should be able to use it:

- If you want to store binary data and make it available to the user, put it in `data/`. This is the best place to put example datasets.
- If you want to store parsed data, but not make it available to the user, put it in `R/sysdata.rda`. This is the best place to put data that your functions need.
- If you want to store raw data, put it in `inst/extdata`.

A simple alternative to these three options is to include it in the source of your package, either creating by hand, or using `dput()` to serialise an existing data set into R code.

Each possible location is described in more detail below.

Exported data

The most common location for package data is (surprise!) `data/`. Each file in this directory should be a `.RData` file created by `save()` containing a single object (with the same name as the file). The easiest way to adhere to these rules is to use `devtools::use_data()`:

```
x <- sample(1000)
devtools::use_data(x, mtcars)
```

It's possible to use other types of files, but I don't recommend it because `.RData` files are already fast, small and explicit. Other options are described in `data()`. For larger datasets, you may want to experiment with the compression setting. The default is `bzip2`, but sometimes `gzip` or `xz` can create smaller files (typically at the expense of slower loading times).

If the `DESCRIPTION` contains `LazyData: true`, then datasets will be lazily loaded. This means that they won't occupy any memory until you use them. The following example shows memory usage before and after loading the nycflights13 package. You can see that memory usage doesn't change until you inspect the flights dataset stored inside the package.

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

```
## 31.6 MB
```

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

```
## 33.3 MB
```

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

```
## 74 MB
```

I recommend that you always include `LazyData: true` in your `DESCRIPTION`. `devtools::create()` does this for you.

Often, the data you include in `data/` is a cleaned up version of raw data you've gathered from elsewhere. I highly recommend taking the time to include the code used to do this in the source version of your package. This will make it easy for you to update or reproduce your version of the data. I suggest that you put this code in `data-raw/`. You don't need it in the bundled version of your package, so also add it to `.Rbuildignore`. Do all this in one step with:

```
devtools::use_data_raw()
```

You can see this approach in practice in some of my recent data packages. I've been creating these as packages because the data will rarely change, and because multiple packages can then use them for examples:

- babynames (https://github.com/hadley/babynames)
- fueleconomy (https://github.com/hadley/fueleconomy)

- nasaweather (https://github.com/hadley/nasaweather)
- nycflights13 (https://github.com/hadley/nycflights13)
- usdanutrients (https://github.com/hadley/usdanutrients)

Documenting datasets

Objects in `data/` are always effectively exported (they use a slightly different mechanism than `NAMESPACE`'s but the details are not important). This means that they must be documented. Documenting data is like documenting a function with a few minor differences. Instead of documenting the data directly, you document the name of the dataset and save it in `R/`. For example, the roxygen2 block used to document the diamonds data in ggplot2 is saved as `R/data.R` and looks something like this:

```
#' Prices of 50,000 round cut diamonds.
#'
#' A dataset containing the prices and other attributes of almost 54,000
#' diamonds.
#'
#' @format A data frame with 53940 rows and 10 variables:
#' \describe{
#'   \item{price}{price, in US dollars}
#'   \item{carat}{weight of the diamond, in carats}
#'   ...
#' }
#' @source \url{http://www.diamondse.info/}
"diamonds"
```

There are two additional tags that are important for documenting datasets:

- `@format` gives an overview of the dataset. For data frames, you should include a definition list that describes each variable. It's usually a good idea to describe variables' units here.

- `@source` provides details of where you got the data, often a `\url{}`.

Never `@export` a data set.

Internal data

Sometimes functions need pre-computed data tables. If you put these in `data/` they'll also be available to package users, which is not appropriate. Instead, you can save them in `R/sysdata.rda`. For example, two colour-related packages, munsell (https://github.com/cwickham/munsell) and dichromat (http://cran.r-project.org/web/packages/dichromat/index.html), use `R/sysdata.rda` to store large tables of colour data.

You can use `devtools::use_data()` to create this file with the argument `internal = TRUE`:

```
x <- sample(1000)
devtools::use_data(x, mtcars, internal = TRUE)
```

Again, to make this data reproducible it's a good idea to include the code used to generate it. Put it in `data-raw/`.

Objects in `R/sysdata.rda` are not exported (they shouldn't be), so they don't need to be documented. They're only available inside your package.

Raw data

If you want to show examples of loading/parsing raw data, put the original files in `inst/extdata`. When the package is installed, all files (and folders) in `inst/` are moved up one level to the top-level directory (so they can't have names like `R/` or `DESCRIPTION`). To refer to files in `inst/extdata` (whether installed or not), use `system.file()`. For example, the testdat (https://github.com/ropensci/testdat) package uses `inst/extdata` to store a UTF-8 encoded csv file for use in examples:

```
system.file("extdata", "2012.csv", package = "testdat")
```

```
## [1] "/home/travis/R/Library/testdat/extdata/2012.csv"
```

Beware: by default, if the file does not exist, `system.file()` does not return an error - it just returns the empty string:

```
system.file("extdata", "2010.csv", package = "testdat")
```

```
## [1] ""
```

If you want to have an error message when the file does not exist, add the argument `mustWork = TRUE`:

```
system.file("extdata", "2010.csv", package = "testdat", mustWork = TRUE)
```

```
## Error in system.file("extdata", "2010.csv", package = "testdat", mustWork = TRUE): no file found
```

Other data

- Data for tests: it's ok to put small files directly in your test directory. But remember unit tests are for testing correctness, not performance, so keep the size small.
- Data for vignettes. If you want to show how to work with an already loaded dataset, put that data in `data/`. If you want to show how to load raw data, put that data in `inst/extdata`.

CRAN notes

Generally, package data should be smaller than a megabyte - if it's larger you'll need to argue for an exemption. This is usually easier to do if the data is in its own package and won't be updated frequently. You should also make sure that the data has been optimally compressed:

1. Run `tools::checkRdaFiles()` to determine the best compression for each file.
2. Re-run `devtools::use_data()` with `compress` set to that optimal value. If you've lost the code for recreating the files, you can use `tools::resaveRdaFiles()` to re-save in place.