Intro to Writing Packages

Data Wrangling and Husbandry

04/13/2020

Why Write a Package?

Packages are a fundamental way to distribute functions and datasets, but they are also a good way to store your favorite custom functions.

- Official location is CRAN ("Comprehensive R Archive Network")
- Increasingly popular to store on GitHub
- Can live just on your computer(s)

Package creation

Creating R packages used to quite finicky, but now is rather easy thanks to the `devtools` package. Note that there is a cheatsheet available from the RStudio help menu.

To create the start of a package, first think up a name. Ideally, it should be something Google-able and not easily confused with an existing R package (and definitely not the same as an existing R package). Figure out a location where you can create an empty directory with the same name, *but don't create the directory*. If your package names is `JustSayNoFactors` in the directory `~/tmp`, then you can get started with

```
library(devtools)
create(path = "~/tmp/JustSayNoFactors")
```

You'll find that the directory now exists, with files

```
DESCRIPTION
JustSayNoFactors.Rproj
NAMESPACE
```

and a sub-directory `R`.

You can open the `JustSayNoFactors.Rproj` file in RStudio and then setup for git with `usethis::use_git()`. Once that is done, quit RStudio, reopen it by opening `JustSayNoFactors.Rproj` and RStudio will recognize it as having version control via Git. More information: https://support.rstudio.com/hc/en-us/articles/200532077?version=1.2.1206-2&mode=server —

Functions

Start with a new function (or two) in a new file `R/functions.R`

```
nf.data.frame <- function(...) {
  data.frame(stringsAsFactors = FALSE, ...)
}
nf.read.table <- function(...) {
  utils::read.table(as.is = TRUE, ...)
}
```

Once you have done that, you can try them out *as if you had loaded a package with those functions in it*, by typing

```
load_all()
```

```
> load_all()
Loading JustSayNoFactors
> nf.data.frame(x = c("A","B","C"), y = 1:3)
  x y
1 A 1
2 B 2
3 C 3
> summary(.Last.value)
      x                   y      
 Length:3           Min.   :1.0  
 Class :character   1st Qu.:1.5  
 Mode  :character   Median :2.0  
                    Mean   :2.0  
                    3rd Qu.:2.5  
                    Max.   :3.0  
```

Check and Install

You can check your package with `check()` or using the menu of "Build $>$ Check". There will be a warning of an empty DESCRIPTION file.

As long as there are no errors, you can install the package with `install()` or RStudio's "Build $>$ Build & Reload".

Towards a more complete package

You should edit the `DESCRIPTION` file. Hadley Wickham has some advice here. Note that if you want to use the MIT license there is a function `use_mit_license()` to make that easy. The `check()` function is *very* finicky about the `DESCRIPTION` file; for example, the Description text cannot use the word “package” and must be written in sentences.

Importing functions

In our example, `read.table()` comes from the `utils` package. If you submit a package to CRAN, you cannot assume that any packages are available, so you need to

- explicitly state in DESCRIPTION which packages you need, and
- call the functions with the package name, e.g., `utils::read.table()`

The first part can be done with `use_package()`:

```
> use_package("utils")
* Adding utils to Imports
Next:
Refer to functions with utils::fun()
```

Documenting functions

Next up, we need to document the functions. The `roxygen2` package makes this easier. In RStudio, put the cursor on the file with the function that you want to document and select Code > Insert roxygen skeleton. It really is just a skeleton, however, and you need to add text to end up with a result like

```
#' Make data frame
#'
#' Create a data frame with character variables left as is
#' and not converted to factors
#'
#'
#' @param ...
#'
#' @return data frame
#' @export
#'
#' @examples
#'   nf.data.frame(x = c("A","B","C"), y = 1:3)
```

When you are done, type `document()` to create the additional required files. At this point, if you type `?nf.data.frame` for example, you should get an actual help file.

Package documentation

You can describe the package as a whole, although not all authors do this. You can get started with `use_package_doc()`, which creates a skeleton file to get started.

See the section in R Packages for more details

Including data

If you want to include datasets in your package, create a subdirectory called `data` and store your datasets there as `*.RData` files by using R's `save()` command once per dataset. It's easier to use the `devtools` function `use_data()`. There are a few additional complications that are explained at http://r-pkgs.had.co.nz/data.html.

Github

You can either

- Create the GitHub repo (using the browser) and then add the GitHub remote repo
- Use `use_github()` to set things up. The documentation `?use_github` explains how to use it; you'll need to get a token from github as described at https://github.com/settings/tokens

If your package is on GitHub, then other users can install it with

```
devtools::install_github("username/packagename")
```

usethis package

The usethis package is a newer package that pulls out the package tools from devtools, with plans for expansion of those tools. Take a look at it.

Stepped in-class exercise

Figure out a good directory for a demo package, and then type in the RStudio console (if your directory location is “~/tmp”")

```
library(devtools)
create(path = "~/tmp/aphorismr")
```

Go to `~/tmp/aphorismr/` and notice the various files.

Open the `~/tmp/aphorismr/aphorismr.Rproj` file in RStudio and then setup for git with `usethis::use_git()`. Once that is done, quit RStudio, reopen it by opening `aphorismr.Rproj` and RStudio will recognize it as having version control via Git.

Create a file `~/tmp/aphorismr/R/aphorism.R` with the following code

```
aphorism <- function(n = 1){
  aphorisms <- c("The basis of optimism is sheer terror. --
                 "You can observe a lot by just watching. -
                 "If you can't say something good about son
  sample(aphorisms, size = n)
}
```

Type

```r
library(devtools)
load_all()
```

in the console and try out `aphorism()`.

Try `check()` and “Build > Build & Reload”

Use “Code > Insert roxygen skeleton” to document your function, and then run `document()`, `check()`, and “Build > Build & Reload”

Edit your DESCRIPTION file, followed by `check()` and “Build > Clean & Rebuild”