R PackagesStatistics 506

R packages bundle up code, documentation, and data into a single file which can be distributed for installation on other computers. There are four major types of R packages:

1. Implementation of methodology, typically statistical. The majority of R packges fall into this category. Too many examples to list.
2. Interfaces to implementations in other software, or access to an external API (for example from a website). For example, **rlemon** which provides access to the C++ LEMON graph library, or **YTAnalytics** which accesses the YouTube analytic data API.
3. Packages that include a data-set for either testing or analysis purposes. For example, the **nycflights13** package we used in class.
4. Packages to assist with programming R. For example, we mentioned R6 as an object-oriented system alternative to S3/S4; it is implemented in the **R6** package.

Why write your own package?

One reason to write a package is you have some new methodology or data or API that you want to promote, so you can create a package, submit it to CRAN, and anyone can access it.

Aside from that, there are several other reasons you may want to create a package that perhaps will not go on CRAN; for example sharing with collaborators.

1. Ease of distribution. If your code or data covers many files, a package is distributed as a single archive, where you control the file structure.
2. Ease of installation. No need to explain to users where to place files or how to set them up; the installation of the package via `install.packages()` handles those details.
3. Reducing bugs. If you send users a script with many functions in it, they'll have to source the file. Asking users to simply source that document can yield issues - a user may source only part of it, or may unintentionally change it, or because the functions/objects live in the global environment, may remove or overwrite them. Installed packages place their objects in their own environment which are harder to negatively modify.
4. Embedded documentation.

5. Use of internal functions. It may make it easier for you to write code that includes many functions which the end-user doesn't need. A package allows their inclusion as internal functions, so your code can use them, but end-users generally won't have access (except by `:::`).

Writing your own package

In lecture I will create a package to demonstrate. While writing an R package can be done entirely in base R, as shown by the Writing R Extensions manual, https://cran.r-project.org/doc/manuals/r-release/R-exts.html, most modern package development is done with the **devtools** package.

The R Packages book is an essential reference for package creation.

Releasing your packages

If you are sending your package to a specific person, you can just email the built package and they can install it with a call to

```
install.packages("mypackage.tar.gz", repo = NULL)
install.packages("mypackage.tar.gz", repo = NULL, type = "source")
```

(The choice of using `type = "source"` or not depends on whether you build the package as source or binary.)

An easier option for your end-user is to release your package on GitHub or similar, then invite users to use `install_github` or similar commands to install your package.

Finally, if you want to make your package fully publically accesible, you can release it to CRAN or an alternative such as bioconductor. These repositories have differing levels of strictness regarding the packages; for example the packages on CRAN must pass all checks.