PART IX

Appendices

♣ A ♣

Create your own R Package

package creation

While this book focuses on the work-flow of the data scientist and it is not a programming book per se; it still makes a lot of sense to introducing you to building your own package. A package is an ideal solution to keep together the functions and data that you need regularly for specific tasks. Of course, it is possible to keep these functions in a file that can be read in via the function `source()`, but having it all in a package has the advantage of being more standardized, structured and easier to document and so it becomes more portable so that also others can use your code. Eventually, if you have built something great and unique then it might be worth to share it via the "Comprehensive R Archive Network" (CRAN).[1]

package

`source()`

Further information – More on creating packages

This chapter provides only a quick introduction. More information can be found online, for example `http://r-pkgs.had.co.nz`, which is the website version of Hadley Wickham's book "R-Packages" — see Wickham (2015).

Before we can get started, we need to install two packages first: `devtools` – that provides the essentials to build the package – and `roxygen` – that facilitates the documentation. This is done as follows:

`devtools`
`roxygen`

```
install.packages("devtools")
install.packages("roxygen2")
```

`devtools` is the package that is the workhorse to develop packages, it is built and designed to make the process of building packages easier. For readers that use C++, it might suffice to say that `roxygen2` is the equivalent in R for Doxygen.[2] This means that `roxygen2` will be able to create documentation for R-source code if that code is documented according to a specific syntax.

[1] The link to submit a package is: `https://cran.r-project.org/submit.html`.

[2] Doxygen is the de facto standard tool to generate documentation directly form C++ source code. This is achieved by annotating the code in a specific way. `doxygen` also supports other languages such as C, C#, PHP, Java, Python, IDL, Fortran, etc. Its website is here: `http://www.doxygen.nl`.

The Big R-Book: From Data Science to Learning Machines and Big Data, First Edition. Philippe J.S. De Brouwer.

Companion Website: www.wiley.com/go/De Brouwer/The Big R-Book

There are multiple work-flows possible to create a package. One can start from an existing .R file or first create the empty skeleton of a package and then fill in the code. In any case, the two packages are needed and hence, the first step is loading the packages.

```
library(devtools)

## Loading required package: usethis
```

```
library(roxygen2)
```

A.1 Creating the Package in the R Console

We could select the all the functions used in this book to create a package and name it after the book, but it makes more sense to use a narrower scope, such as the functionality around asset pricing (see Chapter 30 *"Asset Valuation Basics"* on page 597), diversity (see Chapter 36.3.1 *"The Business Case: a Diversity Dashboard"* on page 726) or multi criteria decision analysis (see Chapter 27 *"Multi Criteria Decision Analysis (MCDA)"* on page 511). The functions for multi criteria decision analysis in this book follow a neat convention where each function name starts with `mcda_`, this would be a first step of good house-holding that is necessary for a great package in R.

We choose the functions around the diversity dashboard to illustrate how a package in R is built. The code below walks through the different steps. First, we have the function `setwd()`, that sets the working directory on the file system of the computer to the place where we want to create the package. Then, in step one, it defines the function – as it was done before, not using the `roxygen` formatting commenting style for now.

```
setwd(".")  # replace this with your desired directory

# Step 1: define the function(s)
# -- below we repeat the function as defined earlier.
# diversity
# Calculates the entropy of a system with discrete states.
# Arguments:
#   x     -- numeric vector -- observed probabilities of classes
#   prior -- numeric vector -- prior probabilities of the classes
# Returns:
#    numeric -- the entropy / diversity measure
diversity <- function(x, prior = NULL) {
  if (min(x) <= 0) {return(0);} # the log will fail for 0
  # If the numbers are higher than 1, then not probabilities but
  # populations are given, so we rescale to probabilities:
  if (sum(x) != 1) {x <- x / sum(x)}
  N <- length(x)
  if(!is.null(prior)) {
    for (i in (1:N)) {
      a <- (1 - 1 / (N * prior[i])) / (1 - prior[i])
      b <- (1 - N * prior[i]^2) / (N * prior[i] * (1 - prior[i]))
      x[i] <- a * x[i]^2 + b * x[i]
    }
   }
  f <- function(x) x * log(x)
  x1 <- mapply(FUN = f, x)
  - sum(x1) / log(N)
  }
```

Now, we are working from the correct working directory and we have the function `diversity()` in the working memory of R. This is the moment where we can create the package with the function `package.skeleton()` of `devtools`.

```
# Step 2: create the empty package directory:
package.skeleton(list = c("diversity"), # list all the functions
                                        # from the current
                                        # environment that we want
                                        # in the package.
                 name = "div"           # name of the package
                 )
```

package.skeleton()

The function parameter `list` is a vector of the names of all functions that we want to be included in the package. The argument `name` supplied to the function `package.skeleton()` is the name of the package and it will become a subdirectory in the current directory of our file-system. In that directory there you will find now the following files and directories:

```
list.files(path = "div", all.files = TRUE)
## [1] "."                 ".."
## [3] "DESCRIPTION"       "man"
## [5] "NAMESPACE"         "R"
## [7] "Read-and-delete-me"
```

If you find these files, it means that you have created your first package. Those files are the package and can be used as any other package.

RStudio

Digression – Creating packages in RStudio

If you prefer a GUI and you work with RStudio, then it is possible create packages via the menu structure. Click on "File," then "New project ...," then "New Directory" (or use an existing one), choose "R Package" and fill out the details. After confirming, RStudio will create the skeleton for the package, and you will find almost exactly the same files as directly using the function `package.skeleton()` as we did in the aforementioned code. If you have followed the RStudio work-flow, you should have now a new menu item: `build`. This menu allows to load the package and test it. Just as in our aforementioned approach, RStudio has created the following files for you:

- `DESCRIPTION` provides meta-data about your package. We edit this shortly.
- `NAMESPACE` declares the functions your package exports for external use and the external functions your package imports from other packages. At the moment, it holds temporary-yet-functional place-holder content.
- `R` contains the R-code: ideally a .R file for each function.
- `man` contains the documentation for the functions and the package.

Compared to the aforementioned example with no options in `package.skeleton()`, RStudio, has created some extra files:

- `.gitignore` anticipates Git usage and ignores some standard, behind-the-scenes files created by R and RStudio. If you do not plan to use Git, this will will have no impact on anything else.
- `.Rbuildignore` lists files that we need to have around but that should not be included when building the R package from this source.
- `foofactors.Rproj` is the file that makes this directory an RStudio Project. Even if you do not use RStudio, this file is harmless. Or you can suppress its creation with create(..., rstudio = FALSE).

A.2 Update the Package Description

Anyone, who will use your package, needs a short description to understand what the package does and why he or she would need it. Adding this documentation is done by editing the `./div/DESCRIPTION` file. For example, replace its content by:

```
Package: div
Type: Package
Title: Provides functionality for reporting about diversity
Version: 0.8
Date: 2019-07-10
Author: Philippe J.S. De Brouwer
Maintainer: Who to complain to <philippe@de-brouwer.com>
Description: This package provides functions to calculate diversity of discrete
     observations (e.g. males and females in a team) and functions that allow
    a reporting to see if there is no discrimination going on towards some
 of those categories (e.g. by showing salary distributions for the groups of
    observations per salary band).
License: LGPL
RoxygenNote: 6.1.1
```

The last line is used at by R to document the functions, so it is essential to leave it exactly as it was.

Now, that the package has a high-level explanation, it is time to document each function of the package.

A.3 Documenting the Functions

You want, of course, that the functions are neatly documented in the usual way that R recognizes so that all methods such as asking for documentation should also work for your package. For example the following line of code should lead to the documentation of the function `diversity()`, and display standard help-file formatted in the usual way.

```
?diversity
```

Each function will have its own file with documentation. R will look for these files in the `man` directory. As you will notice, these files follow syntax that is reminiscent to LaTeX. If your functions are documented via the standards of `roxygen`, these files will be created automatically, but remain editable.

Open the file `R/diversity.R` and insert right before the function the following comments.

```
#' Function to calculate a diversity index based on entropy.
#'
#' This function returns entropy of a system with discrete states
#' @param x numeric vector, observed probabilities
#' @param prior numeric vector, the prior probabilities
#' @keywords diversity, entropy
#' @return the entropy or diversity measure
#' @examples
#' x <- c(0.4, 0.6)
#' diversity(x)
```

`document()`

To process this documentation, and make it available, use the function `document()`:

```
setwd("./div")  # or your choice of package directory
document()
```

`document()`

At this point, the `document()` function is likely to generate the following errors:

```
Warning: The existing 'NAMESPACE' file was not generated by roxygen2,
and will not be overwritten.
Warning: The existing 'diversity.Rd' file was not generated by
roxygen2, and will not be overwritten.
```

If you prefer to use `roxygen`, it is safe to delete the `./div/man/diversity.Rd` as well as the `./div/NAMESPACE` files, and then execute again the `document()` command. This command will now recreate both files and from now on over-write whenever we run the `document()` command again.

Alternatively, you can use the parameter `roclets` of the function `document` to list what R exactly should do when building your package.

A.4 Loading the Package

The package can be loaded directly from the source code that resides on our hard-disk.

```
setwd(".") # the directory in which ./div is a subdirectory
install("div")
```

The package is now loaded[3] and the functions are available as is its documentation. The code below illustrates the use of the package.

```
x  <- c(0.3, 0.2, 0.5)
pr <- c(0.33, 0.33, 0.34)
diversity(x, prior = pr)
## [1] 0.9411522
```

Test the documentation by executing the following.

```
?diversity
```

This will invoke the help file of that particular function in the usual way for your system setup (a `man`-like environment in the CLI, a window in RStudio or even a page in a web-browser).

[3] Note that the output of the `install()` function is quite verbose and is therefore, suppressed in the book.

A.5 Further Steps

github

If your package does not contain confidential information or propriety knowledge, and you have all rights to it, then you might want to upload it to "github" so it can be shared with the Internet community. `devtools` has even a function that does this for you:

```
install_github('div','your_github_username')
```

And of course – in agreement with the philosophy of agile programming – you will continue to add functions, improve them, improve documentation, etc. From time to time update the version number, document, and upload again.

Also consider to read the `Read-and-delete-me` file and grab Hadley Wickham's book about R-packages. Thanks to his thought leadership packages have indeed become the easiest way to share R-code.

Further information – Further reading

The guide by MIT is very helpful to take a few more steps and this guide also includes more information about other platforms such as Windows. It is here: `http://web.mit.edu/insong/www/pdf/rpackage_instructions.pdf`.
Another great reference is Hadley Wickham's book 2015 book "R packages: organize, test, document, and share your code." It is also freely available on the Internet: `http://r-pkgs.had.co.nz`.