

# Statistics 360: Advanced R for Data Science

## Lecture 5

Brad McNeney



# R packages, CRAN and GitHub

- ▶ CRAN and the R package system is the key to R's success, allowing contributions from hundreds of scientists outside the “R core team”.
- ▶ CRAN has strict quality-control checks and requirements.
- ▶ It is possible to distribute R packages *via* GitHub without the CRAN checks.

# R packages

- ▶ What is an R package?
  - ▶ An easy-to-install collection of R functions, documentation and example data.
- ▶ Why make an R package?
  - ▶ Share your work with others
  - ▶ Get credit for your work
  - ▶ Make it easier for you to use your own work.

# Making R packages

- ▶ The definitive source for making R packages is the “Writing R Extensions” document, which you can find at <https://mirror.rcg.sfu.ca/mirror/CRAN/doc/manuals/r-release/R-exts.html>
- ▶ A minimal R package is a folder containing DESCRIPTION and NAMESPACE files and an R subfolder.
- ▶ Other possible subfolders are data, demo, exec, inst, man, po, src, tests, tools and vignettes.
- ▶ Other possible files in the directory are INDEX, configure, cleanup, LICENSE, LICENCE and NEWS.

# Getting Started

- ▶ A helper function called `package.skeleton()` in the `utils` package that comes with R takes a package name and list of R objects as arguments and sets up directories, files and skeleton help files.

```
rm(list=ls()) # clear workspace  
source("../mars.R") # location of my mars source file  
# package.skeleton(name="mars",ls()) # Can't be re-run
```

Creating directories ...  
Creating DESCRIPTION ...  
Creating NAMESPACE ...  
Creating Read-and-delete-me ...  
Saving functions and data ...  
Making help files ...  
Done.  
Further steps are described in './mars/Read-and-delete-me'.

```
$ more mars/Read-and-delete-me
```

- \* Edit the help file skeletons in 'man', possibly combining help files for multiple functions.
- \* Edit the exports in 'NAMESPACE', and add necessary imports.
- \* Put any C/C++/Fortran code in 'src'.
- \* If you have compiled code, add a useDynLib() directive to 'NAMESPACE'.
- \* Run R CMD build to build the package tarball.
- \* Run R CMD check to check the package tarball.

Read "Writing R Extensions" for more information.



# devtools

- ▶ A package called `devtools` provides more help.
  - ▶ `devtools` was started by Hadley Wickham and has since expanded into a collection of tools for developing R packages, described in a work-in-progress book called “R Packages”  
<https://r-pkgs.org/index.html>
  - ▶ We'll skim Chapter 2, which provides an overview of functionality.
- ▶ In addition to automating more of the process, `devtools` uses `roxygen2` to create documentation from structured comments in your R source files.
  - ▶ C++ programmers will recognize the approach from Doxygen

## Getting started with `create_package()`

- ▶ Call `create_package()` to initialize an R package directory and new R project
- ▶ Recommended that this new directory **not** be part of an existing R project or be under version control.
- ▶ Launches a new RStudio session in the newly-created directory/project
  - ▶ Switch to this new session.
  - ▶ If needed, call `use_git()` to initialize a git repository

```
# install.packages("devtools")
library(devtools)
```

```
## Loading required package: usethis
create_package("/Users/mcneney/MyRpackages/mars")
```

```
## v Setting active project to '/Users/mcneney/MyRpackages/mars'
```

```
## v Leaving 'DESCRIPTION' unchanged
```

```
## Package: mars
```

```
## Title: What the Package Does (One Line, Title Case)
```

```
## Version: 0.0.0.9000
```

```
## Authors@R (parsed):
```

```
## * First Last <first.last@example.com> [aut, cre] (YOUR-ORCID-ID)
```

```
## Description: What the package does (one paragraph).
```

```
## License: `use_mit_license()`, `use_gpl3_license()` or friends to pick a
## license
```

```
## Encoding: UTF-8
```

```
## Roxygen: list(markdown = TRUE)
```

```
## RoxygenNote: 7.1.2
```

```
## v Leaving 'NAMESPACE' unchanged
```

```
## v Setting active project to '<no active project>'
```

# Start your R source files

- ▶ Copy your R source files to the R sub-folder.
  - ▶ For now there is just mars.R, but later we will have predict.mars.R, plot.mars.R, etc.
- ▶ Call `load_all()` to load the source into your R session.
  - ▶ Rather than just source in your R code, `load_all()` loads the R functions as a package.
  - ▶ Gives a better sense of how the code will behave when loaded by a user.

```
library(devtools) # call in new R session  
# load_all()
```

## check

- ▶ R CMD check from the command line
- ▶ `check()`
- ▶ Will throw a warning of a non-standard licence

## Edit DESCRIPTION file

- ▶ Add your name, collaborators, etc.

## Add a licence

- ▶ Helper functions add a copy of the relevant license to your package main directory and update the Licence field of the DESCRIPTION file.

```
# use_gpl3_license()
```

## Add documentation comments to your source file.

- Open your .R file(s), place your cursor in one of the functions and click Code->Insert Roxygen Skeleton to insert a skeleton of the roxygen2-style comments above the function.

```
#' Title
#'  
#' @param formula  
#' @param data  
#' @param control  
#' @param ...  
#'  
#' @return  
#' @export  
#'  
#' @examples
```



```
#' Multivariate Adaptive Regression Splines (MARS)
#'  
#'  
#' @param formula an R formula  
#' @param data a data frame containing the data for the model  
#' @param control an object of class 'mars.control'  
#' @param ... other arguments -- currently not used  
#'  
#' @return an object of class 'mars'  
#' @export  
#'  
#' @examples  
#' mm <- mars(wage ~ age,data=ISLR::Wage)  
#' @import stats  
#' @import ISLR
```

## Call document()

- ▶ Calling document() will create the .Rd file from your comments, and add mars as an “export” and stats and ISLR as “imports” in the mars NAMESPACE file.
  - ▶ See below for Imports

```
> document()  
Updating mars documentation  
Loading mars  
Writing NAMESPACE  
Writing mars.Rd  
> load_all()  
Loading mars  
> ?mars  
Rendering development documentation for 'mars'
```

## Add other packages you depend on

- ▶ Reference: R packages, Chapter 8
- ▶ Your package's reliance on other packages can be as
  - ▶ “Depends” – you expect that users will always want to call `library(package)` when loading your package – not common these days
  - ▶ “Imports” – you need to call functions from the `NAMESPACE` of another package. Add these to the `DESCRIPTION` file with `use_package()`.
  - ▶ “Suggests” – Packages that are not crucial, but helpful (e.g., used in examples). Add these to the `DESCRIPTION` file yourself
  - ▶ “Enhances” – Packages that are enhanced by yours – not common

```
# use_package("stats")  
# use_package("ISLR") # used in mars example  
# document()
```

## check

- ▶ Call `check()` to check your package using the same checks as R CMD check from the command line.
- ▶ Should return no errors, warnings or notes.

```
# check()
```

# install

- ▶ Call `install()` to install the package in your R library.

```
# install()
```

## Topics not covered

- ▶ You have enough to get you started, but you will need to learn more as you go . . .
- ▶ Our MARS implementation will use the S3 object oriented programming system.
  - ▶ You will need to read about documenting classes, generics and methods in Chapter 10, SS 7
- ▶ Your final project document will include a “vignette”
  - ▶ Chapter 11
- ▶ You will need to write “unit tests” using the `testthat` package
  - ▶ Chapter 12