Packing up, shacking up's (going to be) all you wanna do! Building packages in R and Github

Gianluca Baio

University College London
Department of Statistical Science

g.baio@ucl.ac.uk

http://www.ucl.ac.uk/statistics/research/statistics-health-economics/
http://www.statistica.it/gianluca
https://github.com/giabaio

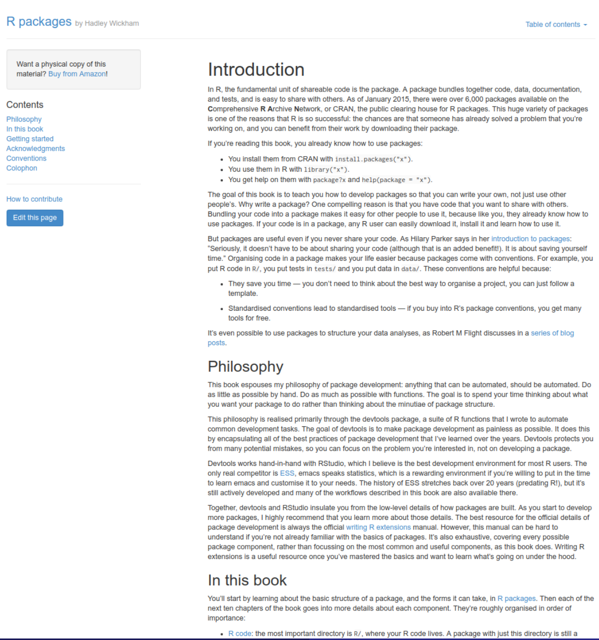
R for trial and model-based cost-effectiveness analysis

University College London
Tuesday 9 July 2019

"Tell me why
Everything turned around
Packing up
Shacking up is all you want to do*" (Go your own way. Fleetwood Mac, 1976)*

What's a R package?

http://r-pkgs.had.co.nz/intro.html



R packages by Hadley Wickham

Table of contents

Want a physical copy of this material? Buy from Amazon!

Contents

How to contribute

Edit this page

Introduction

In R, the fundamental unit of shareable code is the package. A package bundles together code, data, documentation, and tests, and is easy to share with others. As of January 2015, there were over 6,000 packages available on the **C**omprehensive **R A**rchive **N**etwork, or CRAN, the public clearing house for R packages. This huge variety of packages is one of the reasons that R is so successful: the chances are that someone has already solved a problem that you're working on, and you can benefit from their work by downloading their package.

If you're reading this book, you already know how to use packages:

- You install them from CRAN with `install.packages("x")`.
- You use them in R with `library("x")`.
- You get help on them with `package?x` and `help(package = "x")`.

The goal of this book is to teach you how to develop packages so that you can write your own, not just use other people's. Why write a package? One compelling reason is that you have code that you want to share with others. Bundling your code into a package makes it easy for other people to use it, because like you, they already know how to use packages. If your code is in a package, any R user can easily download it, install it and learn how to use it.

But packages are useful even if you never share your code. As Hilary Parker says in her introduction to packages: "Seriously, it doesn't have to be about sharing your code (although that is an added benefit!). It is about saving yourself time." Organising code in a package makes your life easier because packages come with conventions. For example, you put R code in `R/`, you put tests in `tests/` and you put data in `data/`. These conventions are helpful because:

- They save you time — you don't need to think about the best way to organise a project, you can just follow a template.
- Standardised conventions lead to standardised tools — if you buy into R's package conventions, you get many tools for free.

It's even possible to use packages to structure your data analyses, as Robert M Flight discusses in a series of blog posts.

Philosophy

This book espouses my philosophy of package development: anything that can be automated, should be automated. Do as little as possible by hand. Do as much as possible with functions. The goal is to spend your time thinking about what you want your package to do rather than thinking about the minutiae of package structure.

This philosophy is realised primarily through the devtools package, a suite of R functions that I wrote to automate common development tasks. The goal of devtools is to make package development as painless as possible. It does this by encapsulating all of the best practices of package development that I've learned over the years. Devtools protects you from many potential mistakes, so you can focus on the problem you're interested in, not on developing a package.

Devtools works hand-in-hand with RStudio, which I believe is the best development environment for most R users. The only real competitor is ESS, emacs speaks statistics, which is a rewarding environment if you're willing to put in the time to learn emacs and customise it to your needs. The history of ESS stretches back over 20 years (predating R!), but it's still actively developed and many of the workflows described in this book are also available there.

Together, devtools and RStudio insulate you from the low-level details of how packages are built. As you start to develop more packages, I highly recommend that you learn more about those details. The best resource for the official details of package development is always the official writing R extensions manual. However, this manual can be hard to understand if you're not already familiar with the basics of packages. It's also exhaustive, covering every possible package component, rather than focussing on the most common and useful components, as this book does. Writing R extensions is a useful resource once you've mastered the basics and want to learn what's going on under the hood.

In this book

You'll start by learning about the basic structure of a package, and the forms it can take, in R packages. Then each of the next ten chapters of the book goes into more details about each component. They're roughly organised in order of importance:

- R code: the most important directory is `R/`, where your R code lives. A package with just this directory is still a

Package DESCRIPTION

Used to store the package **metadata**

```
Package: survHE
Title: Survival Analysis in Health Economic Evaluation
Version: 1.0.65
Date: 2018-11-07
Authors@R:
    person(given = "Gianluca",
           family = "Baio",
           role = c("aut", "cre"),
           email = "gianluca@stats.ucl.ac.uk")
URL: https://github.com/giabaio/survHE, http://www.statistica.it/gianluca
BugReports: https://github.com/giabaio/survHE/issues
Description: Contains a suite of functions for survival analysis in health economics under a frequentist or a Bayesian approach.
License: GPL (>=3)
Depends:
    methods,
    R (>= 3.4.0),
    Rcpp (>= 0.12.19),
    flexsurv
Imports:
    rms,
    xlsx,
    tools,
    rstan (>= 2.18.1),
Suggests:
    INLA
LinkingTo:
    BH (>= 1.66.0-1),
    Rcpp (>= 0.12.19),
    RcppEigen (>= 0.3.3.4.0),
    rstan (>= 2.18.1),
    StanHeaders (>= 2.18.0)
Additional_repositories: https://inla.r-inla-download.org/R/stable
Encoding: UTF-8
LazyData: true
NeedsCompilation: yes
SystemRequirements: GNU make
RoxygenNote: 6.1.1
```

R code

- The R folder contains all the functions in the package
  - Could simply have one big .R file with all the functions

```
func1 <- function(inputs,...) {
   code here
   ...
}

func2 <- function(inputs,...) {
   code here
   ...
}
...
```

  - **BUT**: probably best to separate out the functions (one per .R file)

```
mypackage/
  DESCRIPTION
  NAMESPACE
  R/
    func1.R
    func2.R
    ...
  ...
```

- See http://r-pkgs.had.co.nz/r.html for tips on code style

Code documentation (the man subfolder)

- This is what you get when you type `?(function)` or `help(function)` on your R terminal
- The information is stored in suitable `.Rd` files

```
\name{add}
\alias{add}
\title{Add together two numbers}
\usage{
add(x, y)
}
\arguments{
  \item{x}{A number}
  \item{y}{A number}
}
\value{
The sum of \code{x} and \code{y}
}
\description{
Add together two numbers
}
\examples{
add(1, 1)
add(10, 1)
}
```

- In the old days, you would have to make one such files for each function in your package!

devtools & roxygen2

- In the new days, you don't need to create the folder structure or the tedious manual files yourself (kind-of...)
  - The structure of the package folder is automatically created using

```
> install.packages("devtools")    # (Only needed one time)
> devtools::create("path/to/package/pkgname")
```

- The R functions and their documentation can also be automatically integrated
  - First create the file `add.R` with the following formatting (mark-up)

```
#' Add together two numbers.
#'
#' @param x A number.
#' @param y A number.
#' @return The sum of \code{x} and \code{y}.
#' @examples
#' add(1, 1)
#' add(10, 1)
add <- function(x, y) {
  x + y
}
```

  - Then run in the terminal either

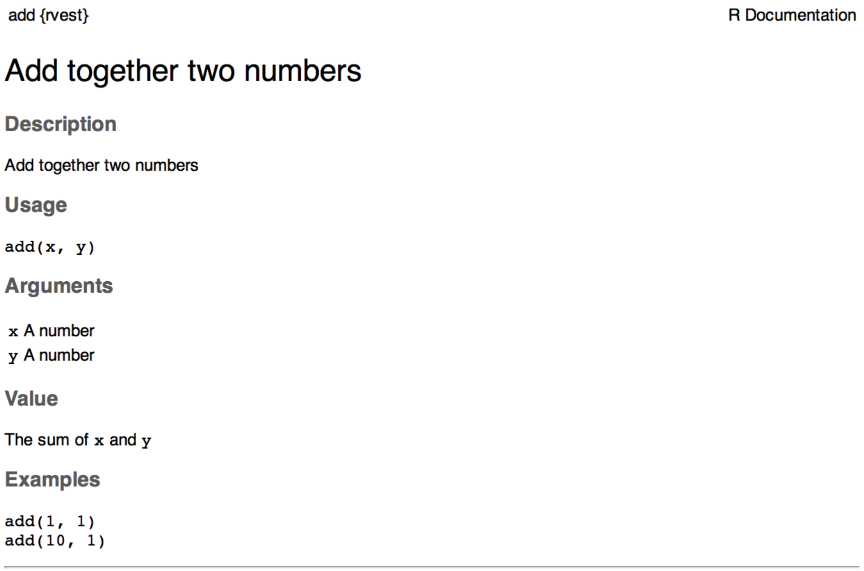
```
> devtools::document()
```

or

```
> roxygen2::roxygenise()
```

to auto-generate the .Rd file

devtools & roxygen2



add {rvest} R Documentation

Add together two numbers

Description

Add together two numbers

Usage

```
add(x, y)
```

Arguments

`x`	A number
`y`	A number

Value

The sum of `x` and `y`

Examples

```
add(1, 1)
add(10, 1)
```

When you're finished. . . (It's a bit more complicated than I made it sound. . .)

- devtools can be used to create the actual "package" from the folder with all the files you've created

```
> # Converts a package source directory into a single bundled file
> devtools::build("path/to/package/pkgname",binary=XXX,...)
```

  - If the optional input `binary=FALSE` (default), then creates a `.tar.gz` file that can be installed cross-platforms
  - If `binary=TRUE`, then creates a platform-specific file (eg `.zip` under Windows)
  - Can specify other options, eg `manual` (default: `FALSE`)

```
> # Automatically builds and checks a *source* package, using all best practices
> devtools::check(pkgname,...)
```

  - Passing the checks is essential before the package can be submitted to the CRAN — but it is helpful even if you only intend to use the package yourself or among colleagues
  - Most likely, the check will throw lots of `NOTE`s and/or `ERROR`s — these don't necessarily stop your functions from working, but are not good signs. . .

- Once the bundle has been created (in `.tar.gz` format), it can be submitted to http://cran.r-project.org/submit.html
  - The submission is tested under different platforms and **if** all is well, uploaded on CRAN

Managing your package on GitHub

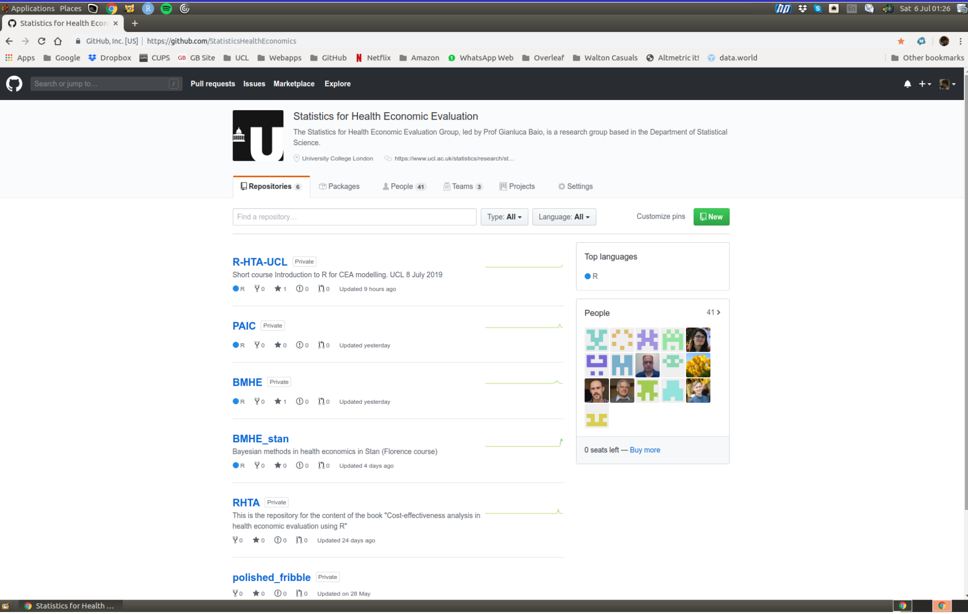
- Often it is a good idea to keep a "stable" version on CRAN and a "development" version on a software development & sharing platform (eg GitHub)
  - In fact, you may not even need CRAN at all — both versions could be managed on GitHub
  - You can create private "working groups" within GitHub, that only members can access and modify, to share packages internally (eg within companies)
- If a package is maintained on GitHub, devtools is still your best friend. . .

```
> devtools::install_github("giabaio/BCEA")
```

  - devtools has functions to install packages from other software development platforms (eg Bitbucket)
- The main advantage of having packages on GitHub is that they can be continuously updated — CRAN is a lot less dynamic
  - A typical week has over 100 submissions and only three volunteers to process them all
  - Under GitHub, you can "push" changes and modify issues whenever you like and the new version of the package is immediately available!

Managing your package on GitHub

https://github.com/



Statistics for Health Economic Evaluation

The Statistics for Health Economic Evaluation Group, led by Prof Gianluca Baio, is a research group based in the Department of Statistical Science.

University College London https://www.ucl.ac.uk/statistics/research/st...

Repositories 6 Packages People 41 Teams 3 Projects Settings

R-HTA-UCL Private
Short course Introduction to R for CEA modelling. UCL 8 July 2019
R 0 1 0 0 Updated 9 hours ago

PAIC Private
R 0 0 0 0 Updated yesterday

BMHE Private
R 0 1 0 0 Updated yesterday

BMHE_stan
Bayesian methods in health economics in Stan (Florence course)
R 0 0 0 0 Updated 4 days ago

RHTA Private
This is the repository for the content of the book "Cost-effectiveness analysis in health economic evaluation using R"
0 0 0 0 Updated 24 days ago

polished_fribble Private
0 0 0 0 Updated on 28 May

Top languages
● R

People 41

0 seats left — Buy more

Thank you!

(And now lunch!)