Introduction to R packages

Mikhail Dozmorov

Summer 2018

DRY, don't repeat yourself

- If you're repeating the same lines of code in multiple places, you should turn those minimal repetitive tasks into functions – reuse your code
- A package is a collection of frequently used functions
- Package = easiest way to distribute code and data
- Package = easiest way to reuse other's code

Do One Thing and Do It Well

- Functions are minimal bits of repeated code that do one thing well
- Should be universal – applied to a variety of problems
- Scalability – should handle small and large tasks equally well

Package repositories

- `CRAN` - Comprehensive R Archive Network – a collection of $>$ 12,500 (May 2018) packages
- `Bioconductor` – genomics-oriented free and open source project hosting $>$ 1,500 specialized R packages (May 2018)
- `MRAN` - Microsoft R Application Network, includes CRAN packages and more
- `GitHub` – code hosting repository, packages for everyone and by everyone

https://cran.r-project.org/web/packages/

https://www.bioconductor.org/

https://mran.microsoft.com/

https://github.com/

Installing packages

- `install.packages("<package_name>")` – install from CRAN
- `install.packages("<package_name.tar.gz>", repos = NULL)` – install from a tarball archive
- `R CMD INSTALL <package_name.tar.gz>` - install from a command line
- `source("https://bioconductor.org/biocLite.R"); biocLite("<package_name>")` – install from Bioconductor (alternatively, install `BiocInstaller` package and use `biocLite()` function from it)
- `devtools::install_github('mdozmorov/MDmisc')` – install from GitHub

Loading packages

- `library(package_name)` – load library to use its functions
- `library()` vs. `require()`
  - `require()` *tries* to load the package, returns TRUE or FALSE
  - `library()` just loads the package, fails if the package is not available

> Ladies and gentlemen, I've said this before: require() is the wrong way to load an R package; use library() instead #useR2014
>
> — Yihui Xie (@xieyihui) July 2, 2014

https://yihui.name/en/2014/07/library-vs-require/

Using functions from other packages

- You can access functions without loading the package using the `::` operator, e.g., `Hmisc::rcorr()`
- Entering function name without parentheses will output its code

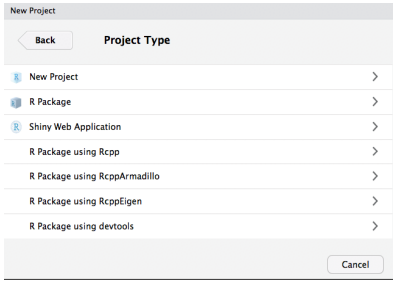
```
> biocLite
function (pkgs = c("Biobase", "IRanges", "AnnotationDbi"), sup
    suppressAutoUpdate = FALSE, siteRepos = character(), ask =
    ...)
{
    if (missing(pkgs))
        pkgs <- pkgs[!pkgs %in% rownames(installed.packages())
...
```

- You can access internal functions of a package with the `:::` operator

Starting an R package using RStudio

Ideally, create packages from scratch as soon as you begin on a project

- RStudio->File->New project->New Directory->R Package



Package made simple with devtools

- Two packages, `devtools` (creating package skeleton) and `roxygen2` (documenting your code) help creating good packages

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

Creating a bare bone structure of the package: `create("cats")`

Writing your functions

- Each function better be in a separate file, e.g., `cat_function.R`
- Should contain code and documentation
- Placed in `R` subfolder

Example function: cat_function.R

```
#' A Cat Function
#'
#' This function allows you to express your love of cats.
#' @param love Do you love cats? TRUE/FALSE. Defaults to TRUE.
#' @keywords cats
#' @export
#' @examples
#' cat_function()

cat_function <- function(love = TRUE){
  if(love == TRUE){
    print("I love cats!")
  }
  else {
    print("I will love cats!")
  }
```

Starting an R package: DESCRIPTION

- Edit the DESCRIPTION file. *Title*, *Author* and *role*, *Description* (as verbose as you can), *License*
- If some of your functions use functions from other packages, you should add `imports` (forced install) and/or `suggests` (suggested install) sections to the DESCRIPTION file

```
# Adding dplyr to Imports
devtools::use_package("dplyr")
# Adding dplyr to Suggests
devtools::use_package("dplyr", "Suggests")
```

- Functions from packages declared in the DESCRIPTION file should be used with the `::` sign, e.g., `dplyr::left_join()`

Starting an R package: DESCRIPTION

Short-term: Keeps track of imports (dependencies)

Long-term: Help others find your package

```
Package: examplepackage
Type: Package
Title: What the Package Does (Title Case)
Version: 0.1.0
Author: Who wrote it
Maintainer: The package maintainer <yourself@somewhere.net>
Description: More about what it does (maybe more than one line
    Use four spaces when indenting paragraphs within the Descr
License: What license is it under?
Encoding: UTF-8
LazyData: true
```

Making your functions available

- All packages have a `NAMESPACE` file: a collection of objects to be exported and imported
  - To avoid overwriting users' variables
  - To avoid ambiguity in function calls
  - To ensure the package has everything it needs to run
  - To encourage modular code

```
# Generated by roxygen2: do not edit by hand

S3method(t,test2)
export(TCGA_corr)
export(Venn2)
export(Venn3)
export(Venn4)
export(Venn5)
export(gene_enrichment)
```

Making your functions available

- A `NAMESPACE` file specifies which functions are available to the user, and which are hidden (helper functions, minimize naming conflicts)

```
export(function_name)
```

- A minimal `NAMESPACE` file

```
# Export all names
exportPattern(".")
```

- Your `NAMESPACE` is auto generated using *tags*; never directly modify your `NAMESPACE` file

Package priorities

Question: What is more important?

- Usability, solves real problem
- Statistical (methodological) superiority
- Documentation
- Speed

Package priorities

Question: What is more important?

- Usability, solves real problem
- Statistical (methodological) superiority
- **Documentation**
- Speed

Documenting functions: the old way

- Originally, documentation was written in LaTeX-like format, stored in `man/*.Rd` files

```
\name{cat_function}
\alias{cat_function}
\title{A Cat Function}
\usage{
cat_function(love = TRUE)
}
\arguments{
\item{love}{Do you love cats? Defaults to TRUE.}
}
\description{
This function allows you to express your love of cats.
}
\examples{
cat_function()
}
\keyword{cats}
```

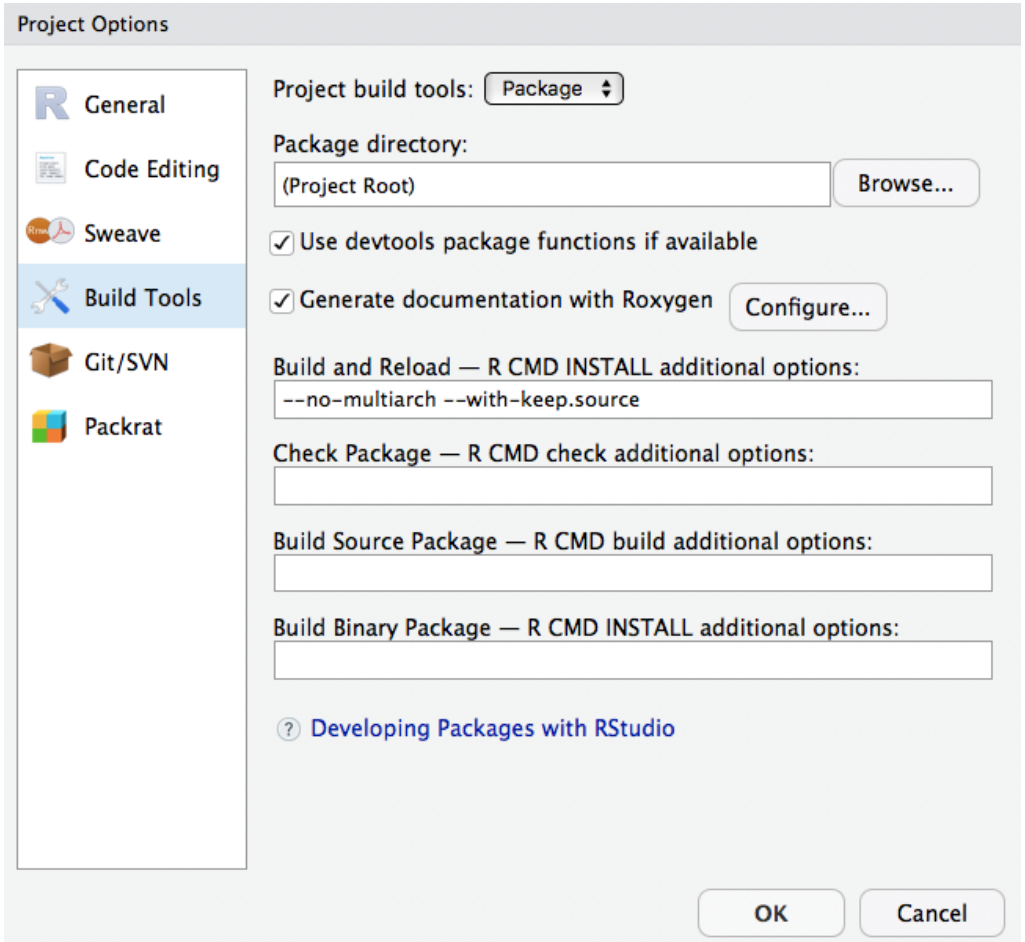
Documenting functions: the simple way

- The package `roxygen2` greatly simplifies documentation

```
#' A Cat Function
#'
#' This function allows you to express your love of cats.
#' @param love Do you love cats? Defaults to TRUE.
#' @keywords cats
#' @export
#' @examples
#' cat_function()
```

Roxygen2

Under the `Build` tab, under Build tools, check `Generate documentation with Roxygen`



Documenting functions

- The package roxygen2 greatly simplifies documentation
- Roxygen2 docstrings start with #'
- Keywords defining pieces of documentation start with @
  - `@param` - parameter description
  - `@return` - what the function returns
  - `@export` - must be to make the function available
  - `@examples` - how-to use the function
- Can (must) use LaTeX syntax in special cases
  - `\code{ <R code here> }` - code highlight
  - `\url{ http:// ... }` - URL
  - `\email{name@...}` - e-mail

Generating documentation

- Documentation is processed with a wrapper of `roxygenize` function

```
setwd("./cats")
devtools::document()
```

Writing detailed documentation

- **Vignette** – an instructive tutorial demonstrating practical uses of the software with discussion of the interpretation of the results (vignette = tutorial). Critical to get a user started with your package
- A short introduction that explains
  - The type of data the package can be used on
  - The general purpose of the functions in the package
  - One or more example analyses with
  - A small, real data set
  - An explanation of the key functions
  - An application of these functions to the data
  - A description of the output and how it can be used

Writing vignettes

- Written using Markdown syntax
- Saved in `vignettes/*.Rmd` files
- Add YAML header to each vignette file

```
---
title: "Vignette title"
date: "2018-06-13"
output: rmarkdown::html_vignette
vignette: >
  %\VignetteIndexEntry{Vignette title}
  %\VignetteEngine{knitr::rmarkdown}
  \usepackage[utf8]{inputenc}
---
```

- Build your vignettes with the `devtools::build_vignettes()` command

Package building pipeline using devtools

```
create("cats")
document("cats")
build_vignettes("cats")
build("cats")
install("cats")
check("cats")
```

Package building pipeline using R

- `R CMD build cats` – will create a tarball of the package, with its version number encoded in the file name
- `R CMD install cats_0.0.0.9000.tar.gz`
- `R CMD check --as-cran cats_0.0.0.9000.tar.gz`

Building your package with RStudio

- The **Build and Reload** command performs several steps in sequence to ensure a clean and correct result
  - Unloads any existing version of the package (including shared libraries if necessary)
  - Builds and installs the package using R CMD INSTALL
  - Restarts the underlying R session to ensure a clean environment for re-loading the package
  - Reloads the package in the new R session by executing the library function

Including datasets

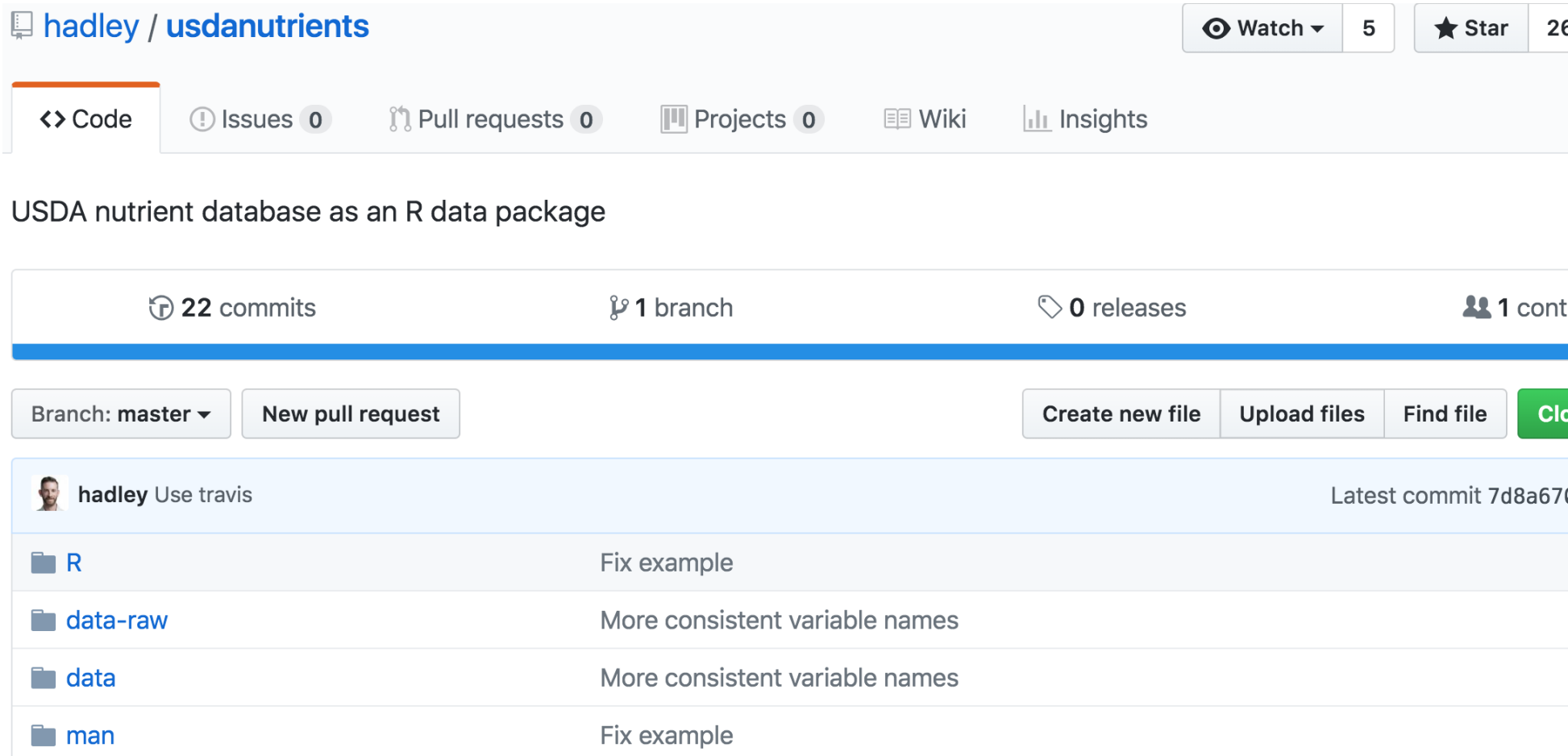
- Create `data` folder
- Save your data in R binary format, using `save(mydata, file = "data/mydata.rds")` (or, use `.RData`, or `.rda` extension)
- Can include `.txt` of `.csv` files
- Add `LazyData: true` in the `DESCRIPTION` file – your data will be immediately available (loaded on the first use) with the package

Documenting datasets

- Add `R/mydata-data.R` file
- Document with `roxygen2` syntax

```
#' My data brief info
#'
#' Longer description of my data
#'
#' @docType data
#' @usage data(mydata)
#' @format An object of class \code{"data.frame"}
#' @keywords datasets
#' @references Put reference here
#' @source \href{http://....org}{Link}
#' @examples
#' data(mydata)
"mydata" # No extension
```

Example of a dataset package



https://github.com/hadley/usdanutrients

Put your package on GitHub as a regular repository

- Use `install_github("git_username/package_name")` function to install a package from GitHub

```
devtools::install_github("hadley/usdanutrients")
```

Updating R and packages

- `installr::updateR()` - update R and the corresponding packages on Windows
- `updateR` - update R on Mac

https://cran.r-project.org/web/packages/installr/

https://github.com/AndreaCirilloAC/updateR

Upgrading R on Windows and Mac,
https://www.r-statistics.com/2018/04/r-3-5-0-is-released-major-release-with-many-new-features/

Other things to keep in mind

- `testthat` is a H.W. package to write unit tests
- `rm(list=ls(all=TRUE))` removes everything in the global environment
  - But does not unload packages! Use, e.g., `detach("package:vegan", unload=TRUE)`
- `pkgdown` is a H.W. package that can autogenerate a website for your package `build_site()`
- `pRojects` - R package for making projects
- `ProjectTemplate` - A template utility for R projects that provides a skeletal project

https://github.com/r-lib/testthat

https://github.com/r-lib/pkgdown

https://itsalocke.com/projects/, https://github.com/lockedata/pRojects

http://projecttemplate.net, https://github.com/johnmyleswhite/ProjectTemplate