

# **AUTOMATE CODE**

## **R functions**

# 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 > 8,400 (as of June 2016) packages
- **Bioconductor** – genomics-oriented free and open source project hosting > 1,200 specialized R packages
- **MRAN** - Microsoft R Application Network
- **GitHub** – code hosting repository

# 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
- `devtools::install_github('mdozmorov/MDmisc')` – install from GitHub

# Using functions from other packages

- `library(package_name)` – load library to use its functions
- You can access functions without loading the package using the `::` operator

`Hmisc::rcorr()`

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

# Package made simple

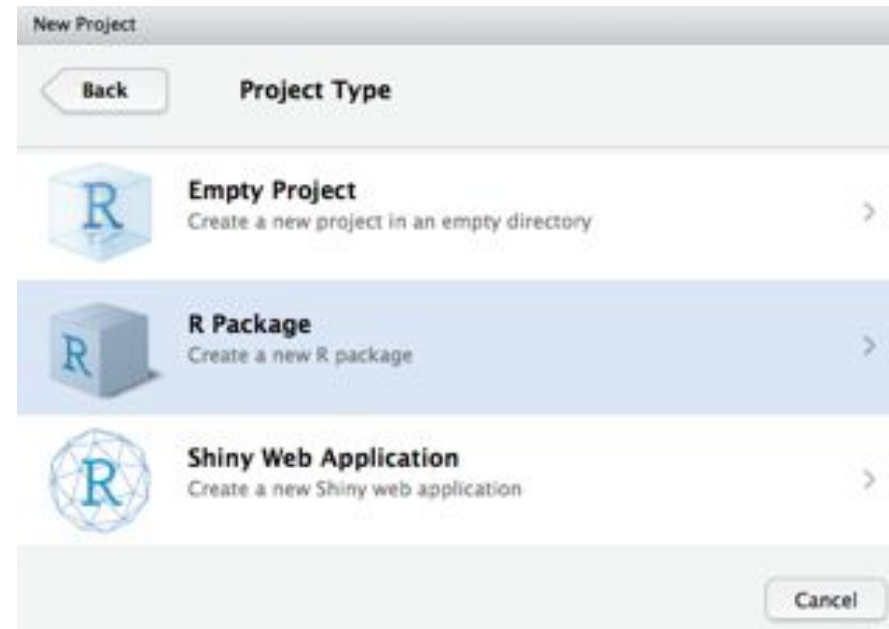
Two packages, `devtools` (creating package skeleton) and `roxygen2` (documenting your code) help creating good packages

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

Creating a bare bone structure of the package

- `create("cats")`

```
cats
|-- DESCRIPTION
|-- NAMESPACE
|-- R
`-- cats.Rproj
```







# Customizing your package

- 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()**



# 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 (".")
```

# 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 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()
```

```
cats
|-- DESCRIPTION
|-- NAMESPACE
|-- R
|   |-- cat_function.R
|-- cats.Rproj
|-- man
|   |-- cat_function.Rd
```



# 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

---

## Documentation

<a href="#">HTML</a>	<a href="#">R Script</a>	ChIPseeker: an R package for ChIP peak Annotation, Comparison and Visualization
<a href="#">PDF</a>		Reference Manual
<a href="#">Text</a>		NEWS

---

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: "`r Sys.Date()`"  
output: rmarkdown::html_vignette  
vignette: >  
  %\VignetteIndexEntry{Vignette title}  
  %\VignetteEngine{knitr::rmarkdown}  
  \usepackage[utf8]{inputenc}  
---
```

Build your vignettes with the `devtools::build_vignettes()` command  
The resulting `*.html` files will be in the `inst/doc` folde

# Package building pipeline using `devtools`

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

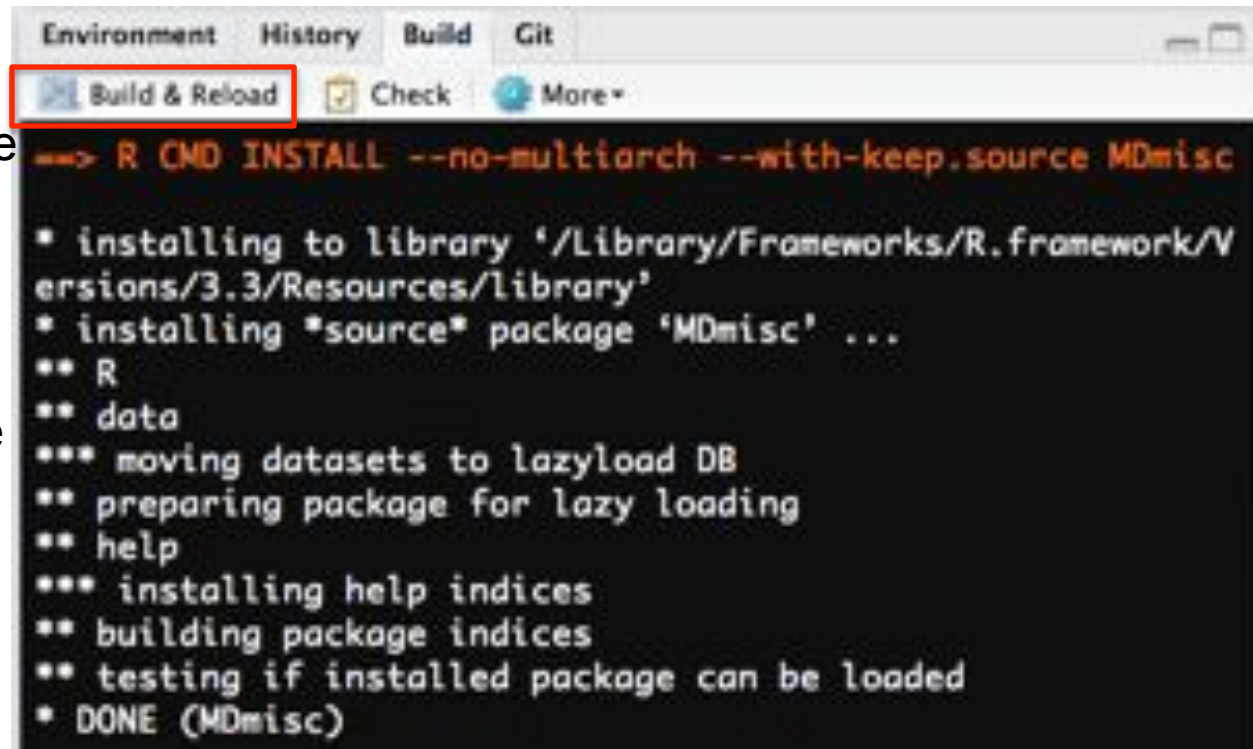
# Package building pipeline using R

- 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

A screenshot of the RStudio interface showing the 'Build & Reload' command selected in the 'Build' menu. Below the menu, a terminal window displays the output of the command. The output shows the installation of the 'MDmisc' package to the R library path, including steps for installing source packages, moving datasets to a lazyload database, preparing for lazy loading, and testing the package.

```
Environment History Build Git
Build & Reload Check More+
--> R CMD INSTALL --no-multiarch --with-keep.source MDmisc
* installing to library '/Library/Frameworks/R.framework/Versions/3.3/Resources/library'
* installing *source* package 'MDmisc' ...
** R
** data
*** moving datasets to lazyload DB
** preparing package for lazy loading
** help
*** installing help indices
** building package indices
** testing if installed package can be loaded
* DONE (MDmisc)
```

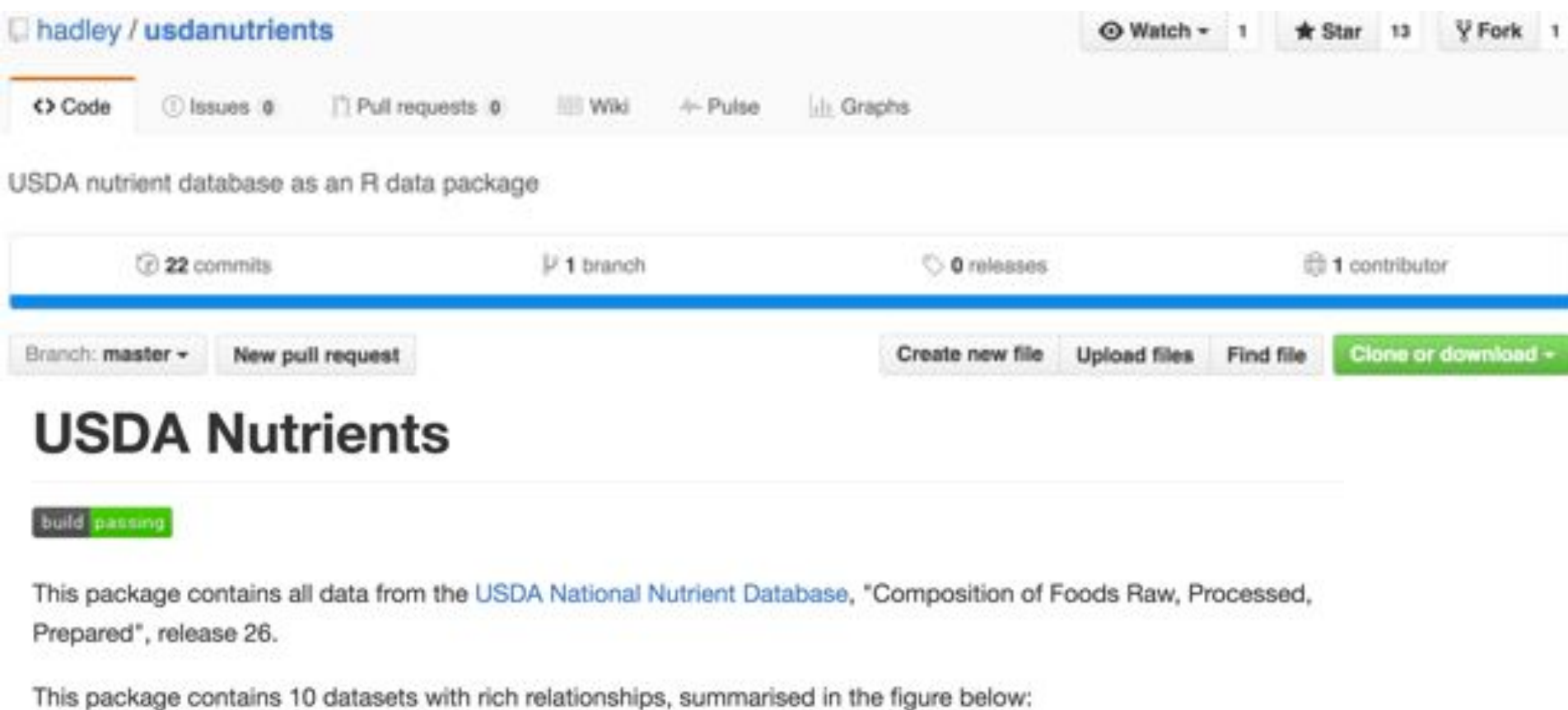
# Including datasets

- 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` or `.csv` files
- Add `LazyData: true` in the `DESCRIPTION` file – your data will be immediately available (loaded on the first use) with the package



# Example of a dataset package

- Create the whole database package



The screenshot shows the GitHub repository page for 'hadley/usdanutrients'. At the top, the repository name is displayed along with 'Watch 1', 'Star 13', and 'Fork 1' buttons. Below this is a navigation bar with 'Code', 'Issues 0', 'Pull requests 0', 'Wiki', 'Pulse', and 'Graphs' options. The repository title is 'USDA nutrient database as an R data package'. A summary bar shows '22 commits', '1 branch', '0 releases', and '1 contributor'. Below this are buttons for 'Branch: master', 'New pull request', 'Create new file', 'Upload files', 'Find file', and a prominent green 'Clone or download' button. The main heading is 'USDA Nutrients'. A 'build passing' badge is visible. The description states: 'This package contains all data from the [USDA National Nutrient Database](#), "Composition of Foods Raw, Processed, Prepared", release 26.' It also mentions: 'This package contains 10 datasets with rich relationships, summarised in the figure below:'.

# Put your package on GitHub

- 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")
```