My Own Private ~~Idaho~~ Repo

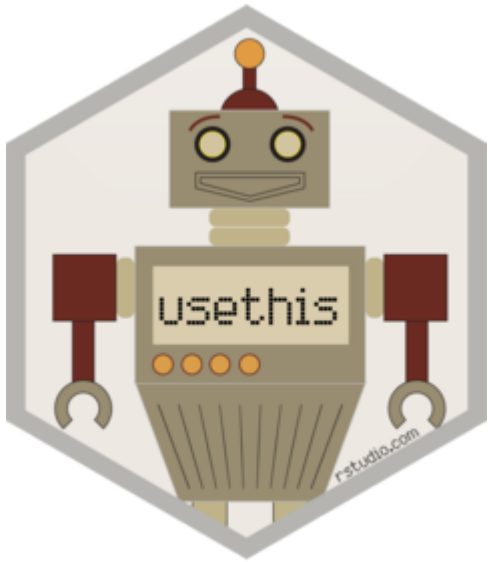
Writing a Personal R Package with Rstudio, {usethis}, and {roxygen2}

Daniel D. Sjoberg

Memorial Sloan Kettering Cancer Center
Department of Epidemiology and Biostatistics

June 4, 2019

RStudio®



Outline

- Why an R Package?
- R Package Structure
- Getting Started
- Adding Functions
- Build your Package
- Other Tips
- Create Package Website
- Additional Resources

Outline

- **Why an R Package?**
- R Package Structure
- Getting Started
- Adding Functions
- Build your Package
- Other Tips
- Create Package Website
- Additional Resources

Why a Personal R Package?

- Encourages good coding practices
- Built-in documentation
- Reprodicibility
- Collaboration
- And more!

Outline

- Why an R Package?
- **R Package Structure**
- Getting Started
- Adding Functions
- Build your Package
- Other Tips
- Create Package Website
- Additional Resources

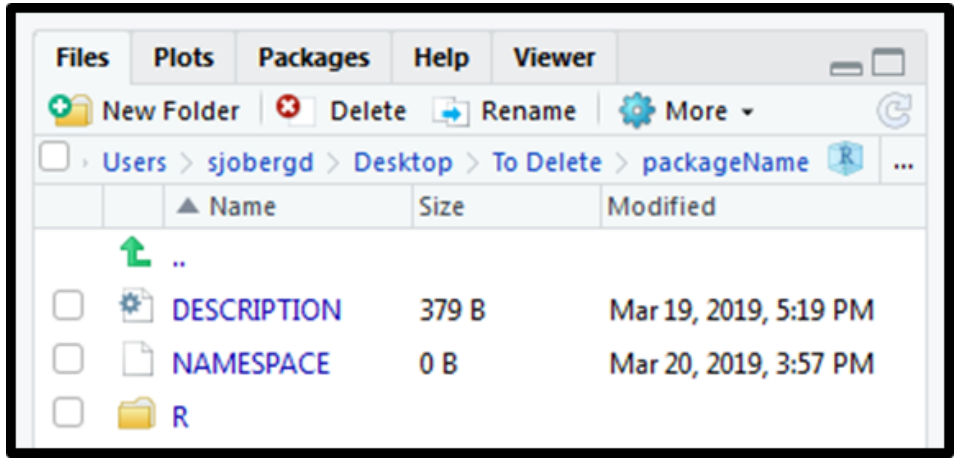
Package Structure

An R package needs 3 components

1. DESCRIPTION file

2. NAMESPACE file

3. R code folder



Package Structure

An R package needs 3 components

1. ***DESCRIPTION file***
2. NAMESPACE file
3. R code folder

```
Package: mypackage
Title: What The Package Does (one line)
Version: 0.1
Authors@R: person(
  "First", "Last",
  email = "first.last@example.com",
  role = c("aut", "cre"))
Description: What the package does
  (one paragraph)
Depends: R (>= 3.5)
Imports: dplyr
License: What license is it under?
LazyData: true
```

- store metadata about the package
- list dependencies
- specify version number of package

Package Structure

An R package needs 3 components

1. DESCRIPTION file

2. ***NAMESPACE file***

3. R code folder

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

export(tbl_regression)
export(tbl_summary)
export(tbl_uvregression)
importFrom(glue,glue)
importFrom(knitr,knit_print)
importFrom(magrittr,"%>%")
```

- {roxygen2} will take care of this for you!
- lists functions that will be exported by your package
- lists functions imported from other packages

Package Structure

An R package needs 3 components

1. DESCRIPTION file
2. NAMESPACE file
3. ***R code folder***

- R folder contains R code for each function in your package
  - typically, one code file for each exported function (although not required)
  - e.g. `myfirstfunction.R`
- also contains code for helper or utility functions
  - these functions are not exported, that is, not available to users of the package
  - utility function files begin with utils- prefix
  - e.g. `utils-myfirstfunction.R`

Package Structure

An R package needs 3 components

1. DESCRIPTION file
2. NAMESPACE file
3. R code folder

The {usethis} package has functions that create the package structure for you

After any function in {usethis} is run, it prints additional information into the console. READ AND FOLLOW THE DIRECTIONS!

- lists files created
- lists files modified
- lists user instructions

{usethis} makes package development a breeze

Outline

- Why an R Package?
- R Package Structure
- **Getting Started**
- Adding Functions
- Build your Package
- Other Tips
- Create Package Website
- Additional Resources

Getting Started

A few functions to get you started with a new package

- `usethis::create_package()`
- `usethis::use_package_doc()`
- `usethis::use_git()`
- `usethis::use_github()`

Getting Started

A few functions to get you started with a new package

- ***usethis::create_package()***
- usethis::use_package_doc()
- usethis::use_git()
- usethis::use_github()

```
> usethis::create_package("~/myPackage")
✓ Setting active project to '~/myPackage'
✓ Creating 'R/'
✓ Creating 'man/'
✓ Writing 'DESCRIPTION'
✓ Writing 'NAMESPACE'
✓ Writing 'myPackage.Rproj'
✓ Adding '.Rproj.user' to '.gitignore'
✓ Adding '^myPackage\\.Rproj$', '^\\.Rproj\\.
   to '.Rbuildignore'
✓ Opening new project 'myPackage' in RStudio
```

- Create package folder and a skeleton of the folder structure

Getting Started

A few functions to get you started with a new package

- `usethis::create_package()`
- ***`usethis::use_package_doc()`***
- `usethis::use_git()`
- `usethis::use_github()`

```
> usethis::use_package_doc()
✓ Writing 'R/myPackage-package.R'
```

'R/myPackage-package.R' contents

```
#' @keywords internal
"_PACKAGE"
```

- writes a basic documentation file for you package
- we will add more to this later

Getting Started

A few functions to get you started with a new package

- `usethis::create_package()`
- `usethis::use_package_doc()`
- ***`usethis::use_git()`***
- `usethis::use_github()`

```
> usethis::use_git()
✓ Initialising Git repo
✓ Adding '.Rhistory', '.RData' to '.gitignore
OK to make an initial commit of 6 files?
1: Negative
2: Not now
3: Yeah

Selection: 3
✓ Adding files and committing
```

- create a git repository
- commit existing files to the repo

Getting Started

A few functions to get you started with a new package

- `usethis::create_package()`
- `usethis::use_package_doc()`
- `usethis::use_git()`
- ***`usethis::use_github()`***

```
> usethis::use_github()
● Check title and description
  Name:        myPackage
  Description: What the Package Does (One Lin
Are title and description ok?
1: No
2: Nope
3: Yeah

Selection: 3
✓ Creating GitHub repository
✓ Adding GitHub remote
✓ Adding GitHub links to DESCRIPTION
✓ Setting URL field in DESCRIPTION to
  'https://github.com/ddsjoberg/myPackage'
✓ Setting BugReports field in DESCRIPTION to
  'https://github.com/ddsjoberg/myPackage/iss
✓ Pushing to GitHub and setting
  remote tracking branch
```

Getting Started

A few functions to get you started with a new package

- `usethis::create_package()`
- `usethis::use_package_doc()`
- `usethis::use_git()`
- ***`usethis::use_github()`***

- THIS ONLY WORKS IF YOU'VE PREVIOUSLY CONFIGURED THE `use_github()` FUNCTION
- recommend you setup Rstudio to play nicely with GitHub. Read *Happy Git and GitHub for the useR* for details (https://happygitwithr.com/)
- you can create your GitHub repo manually and add the package contents if you haven't yet configured RStudio and GitHub

Outline

- Why an R Package?
- R Package Structure
- Getting Started
- **Adding Functions**
- Build your Package
- Other Tips
- Create Package Website
- Additional Resources

Adding Functions

`usethis::use_r()`

- creates a new code file for you write your function
- places file correctly in the `R` folder
- the new file is entirely blank

```
> usethis::use_r("my_mean")
● Modify 'R/my_mean.R'
```

Adding Functions

Let's write our first function

```
my_mean <- function(x) {
  mean(na.omit(x))
}
```

For EVERY non-base R function you need to either *import* the function, or use `::` to reference the function

```
my_mean <- function(x) {
  mean(stats::na.omit(x))
}
```

We will now use {roxygen2} comments in our code to document our new function (aka write the help file)!

Adding Functions: Documentation

- R function help files (*.Rd) are saved in the *man* folder
- the *man* folder already exists courtesy of `usethis::create_package()`
- R processes the *.Rd files to create plain text, PDF, and HTML versions of the help files
- the code in *.Rd files looks somewhat like LaTeX: it's verbose and cumbersome to write
- we will automate the creation of these files with {roxygen2} comments
- by automating, we link the function code to the documentation
- this helps keep the documentation up to date

```
> usethis::use_roxygen_md()
✓ Setting Roxygen field in DESCRIPTION to 'list(markdown = TRUE)'
✓ Setting RoxygenNote field in DESCRIPTION to '6.1.1'
● Run `devtools::document()`
```

Adding Functions: Documentation

- roxygen comments appear above a function
- roxygen comment lines always begins with `#'`
- two common roxygen tags
  - `@param` used to document a function argument
  - `@export` tells roxygen to export the function when the package is built

```
#' The first line is the title
#'
#' The second section is a longer description of the function.
#' This can go on for multiple lines.
#'
#' @param x numeric vector
#' @export
my_mean <- function(x) {
  mean(stats::na.omit(x))
}
```

Adding Functions: Documentation

- other notable roxygen tags
  - `@seealso` list related references (typically used to reference related functions)
  - `@family` similar to `@seealso`, but creates a list of "see also" functions that belong to the same family
  - `@examples` add examples to function help file
  - `@author` list author(s) of the function
  - `@return` specify the returned object
- link to other functions in help file
  - function in the same package `[my_mean]`
  - function in another package `[another_package::fun_name]`
- additional resources
  - *Rd* vignette in {roxygen2} package
  - Blog post http://kbroman.org/pkg_primer/pages/docs.html

Adding Functions: Documentation

```r
#' The first line is the title
#'
#' The second section is a longer
#' description of the function.
#' This can go on for multiple lines.
#'
#' @param x numeric vector
#' @export
#' @seealso \code{\link[base]{mean}}
#' @author Daniel D. Sjoberg
#' @examples
#' my_mean(1:5)
my_mean <- function(x) {
  mean(stats::na.omit(x))
}
```

my_mean {myPackage} R Documentation

The first line is the title

Description

The second section is a longer description of the function. This can go on for multiple lines.

Usage

```
my_mean(x)
```

Arguments

x	numeric vector

Author(s)

Daniel D. Sjoberg

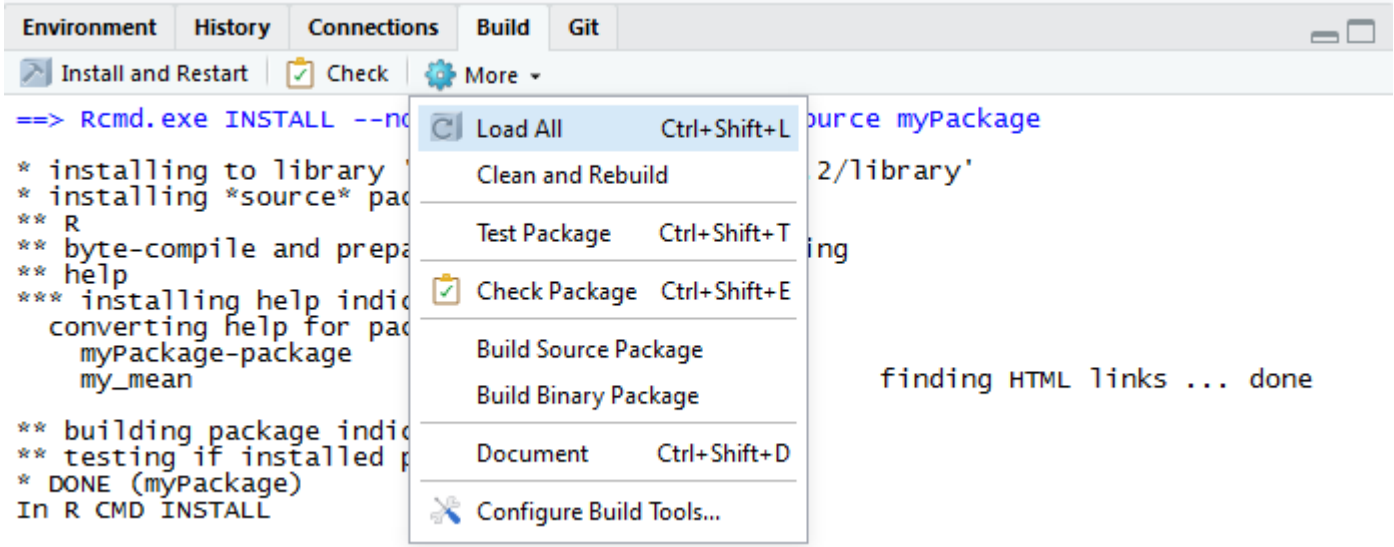
See Also

mean

Examples

```
my_mean(1:5)
```

Adding Functions: Rstudio Build Tab



Document each time you update roxygen comments

Adding Functions

- new function calculates mean of every column in a data frame `usethis::use_r("df_mean")`
- use the `map()` function in the {purrr} package
- use `::` to refer to the function: `purrr::map()`
- document that our package now depends on {purrr}
- `usethis::use_package("purrr")`

```
> usethis::use_package("purrr")
✓ Setting active project to '~/myPackage'
✓ Adding 'purrr' to Imports field
  in DESCRIPTION
● Refer to functions with `purrr::fun()`
```

DESCRIPTION (truncated)

```
Imports:
    purrr
```

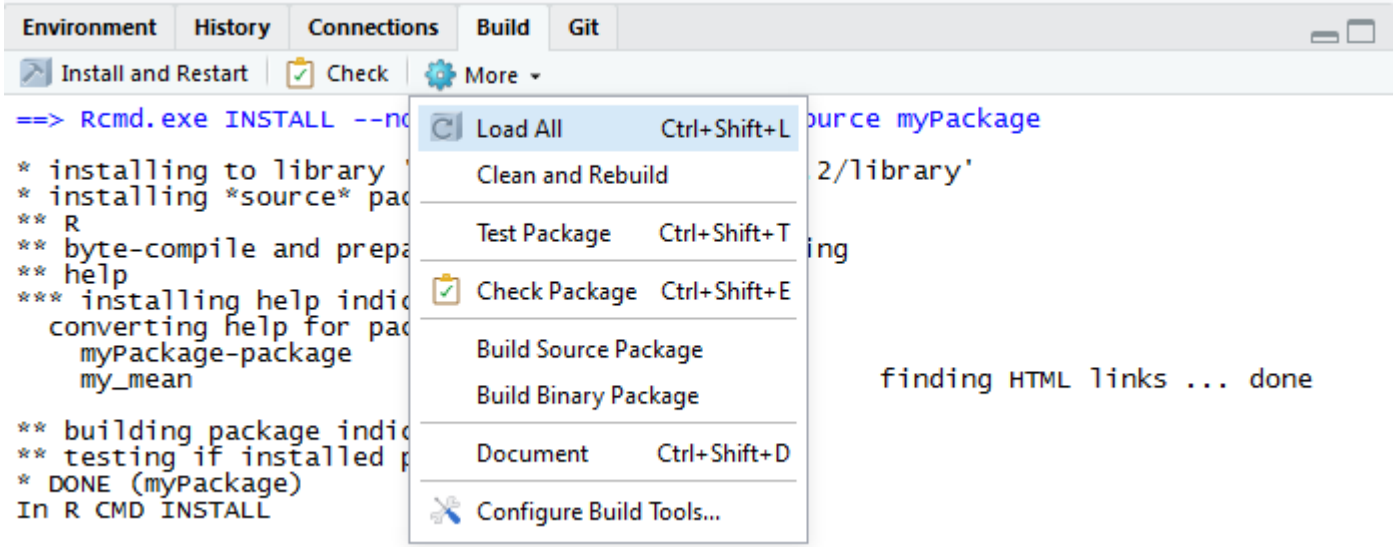
```
df_mean <- function(data) {
  purrr::map(data, my_mean)
}
```

```
> df_mean(mtcars)
$mpg
[1] 20.09062

$cyl
[1] 6.1875

$disp
[1] 230.7219
```

Outline

- Why an R Package?
- R Package Structure
- Getting Started
- Adding Functions
- **Build your Package**
- Other Tips
- Create Package Website
- Additional Resources

Build your Package: Rstudio Build Tab



- **Document** each time you update roxygen comments
- **Install and Restart** each time you update R code
- **Check**

Build your Package

What is checked?
So much! Here's a very abbreviated list

- package structure
  - hidden files/folders
  - portable file names
  - executable files
  - package subdirectories
  - left-over files
- DESCRIPTION/NAMESPACE file
  - package dependencies
  - files exist
  - NAMESPACE parses properly
- R code
  - non-ASCII characters
  - syntax errors
  - dependencies in R code
  - S3 generic/method consistency
- documentation
  - Rd/help files
  - Rd file metadata
  - examples
  - undocumented function arguments

You've Successfully Created an R Package!

- Push your updates to GitHub
- Use `remotes::install_github("yourname/packagename")` to install package

Outline

- Why an R Package?
- R Package Structure
- Getting Started
- Adding Functions
- Build your Package
- **Other Tips**
- Create Package Website
- Additional Resources

Other Tips: @import

- there are a few functions from the tidyverse we use frequently
- referring to them with `::` quickly becomes cumbersome
- we can import functions and even entire packages to avoid the `::` notation
- {roxygen2} tags `@import` and `@importFrom`

Other Tips: @import

- remember that boring package-level documentation file we made with `use_package_doc()`? let's use this file to import commonly used functions

```
#' @import dplyr
#' @import purrr
#' @importFrom tidyr nest unnest
     spread gather complete
#' @importFrom tibble tibble as_tibble
#' @importFrom rlang .data %||% set_names sym
     expr enexpr quo enquo parse_expr
#' @importFrom glue glue
#' @keywords internal
"_PACKAGE"

# allowing for the use of the dot when piping
utils::globalVariables(".")
```

- when adding entire package imports, be sure to run the package checks. you may experience warnings from conflicting function names
    - this is common with tidyverse packages as many of them contain the same functions
- `@import` and `@importFrom` can be placed in any R code file
    - can import same function in multiple files. {roxygen2} will sort out duplicates when the package is documented
- use the package-level documentation file for imports that apply to all functions in your package

Other Tips: the pipe

- the {magrittr} pipe operator is so useful, you may want to both import and export it to make it available to users of your package

```
> usethis::use_pipe()
✓ Adding 'magrittr' to Imports
    field in DESCRIPTION
✓ Writing 'R/utils-pipe.R'
● Run `devtools::document()`
```

`R/utils-pipe.R`

```
#' Pipe operator
#'
#' See \code{magrittr::\link[magrittr]{\%>\%}
#' for details.
#'
#' @name %>%
#' @rdname pipe
#' @keywords internal
#' @export
#' @importFrom magrittr %>%
#' @usage lhs \%>\% rhs
NULL
```

Other Tips: tidy helpers

the {usethis} package contains helper functions to develop tidy packages

these are my favorites

- `use_tidy_description()`
  - puts fields in standard order and alphabetizes dependencies in DESCRIPTION file
- `use_tidy_versions(overwrite = FALSE)`
  - pins all dependencies to require at least the currently installed version
  - helps ensure your package will work on all systems
- `use_tidy_style(strict = TRUE)`
  - uses the {styler} package to style all code according to the tidyverse style guidelines
  - keeps your code easy to read, looking good, and collaborative

Outline

- Why an R Package?
- R Package Structure
- Getting Started
- Adding Functions
- Build your Package
- Keep it on GitHub
- **Create Package Website**
- Additional Resources

Create Package Website

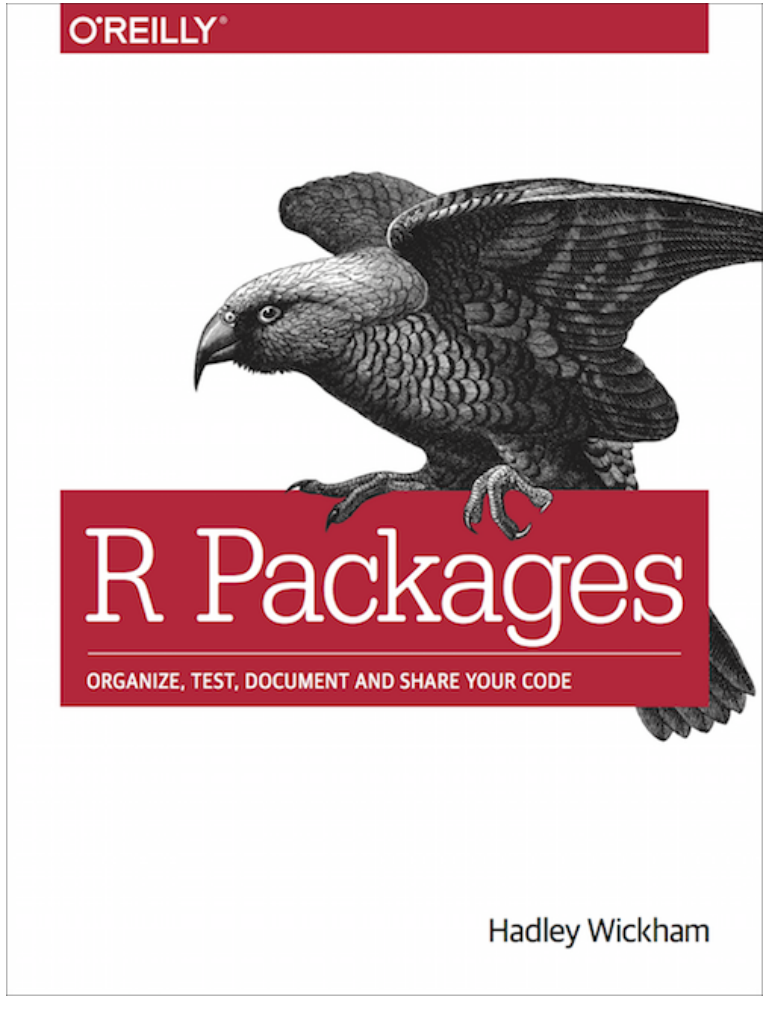
- lastly, it's easy to create a website for your package
- easier for a user to navigate, find functions, and review examples
- from the GitHub repo, select *Settings*, scroll down to *GitHub Pages*, from the *Source* menu select *master branch/docs folder*
- the location of the published site is listed under *GitHub Pages*
- run `pkgdown::build_site()`
- done! can take a few minutes to be available online
- more details at https://pkgdown.r-lib.org/

Outline

- Why an R Package?
- R Package Structure
- Getting Started
- Adding Functions
- Build your Package
- Keep it on GitHub
- Create Package Website
- **Additional Resources**

Additional Resources

- More detailed Guide
  - http://www.danieldsjoberg.com/writing-R-packages/
- {roxygen2}
  - *Rd* vignette in {roxygen2} package
  - Broman Blog https://tinyurl.com/yxj2vzkn
  - RStudio Blog https://tinyurl.com/yyrvysoy
- *R Packages*, by Hadley Wickham
  - touches on most of the topics covered here (package structures, unit tests, documentation)
  - does not include {usethis} setup
  - http://r-pkgs.had.co.nz/
- Git and GitHub
  - https://happygitwithr.com/



Thank you

slides at danieldsjoberg.com/personal-R-package
source code at github.com/ddsjoberg/personal-R-package