An Introduction to Creating R Packages

Denver R Users Group
www.meetup.com/DenverRUG

Peter DeWitt
peter.dewitt@ucdenver.edu

18 June 2015

Common Tasks?

- Think about your life in code.
  - How many tasks do you code regularly?
  - How many of these tasks have you created a general function for?
  - How do you reuse code?
    - Rewrite?
    - `helpful-stuff.R`?
    - A package?
- Let's consider two common tasks for examples:
  1. Report the mean and standard deviation as a formated character string.
  2. Construct a receiver operating curve (ROC) for a logistic regression model.

Please don't do this

```r
data('diamonds', package = 'ggplot2')
mean_price <- mean(diamonds$price)
mean_carat <- mean(diamonds$carat)
mean_depth <- mean(diamonds$depth)
sd_price   <- sd(diamonds$price)
sd_carat   <- sd(diamonds$carat)
sd_depth   <- sd(diamonds$depth)
paste0(formatC(mean_price, digits = 2, format = "f"), " (",
       formatC(sd_price,   digits = 2, format = "f"), ")")

## [1] "3932.80 (3989.44)"

paste0(formatC(mean_carat, digits = 2, format = "f"), " (",
       formatC(sd_carat,   digits = 2, format = "f"), ")")

## [1] "0.80 (0.47)"

paste0(formatC(mean_depth, digits = 2, format = "f"), " (",
       formatC(sd_depth,   digits = 2, format = "f"), ")")

## [1] "61.75 (1.43)"
```

Better, but . . .

```r
mean_sd <- function(x) {
  m <- mean(x)
  s <- sd(x)
  paste0(formatC(m, digits = 2, format = "f"), " (",
         formatC(s, digits = 2, format = "f"), ")")
}

mean_sd(diamonds$price)

## [1] "3932.80 (3989.44)"

mean_sd(diamonds$carat)

## [1] "0.80 (0.47)"

mean_sd(diamonds$depth)

## [1] "61.75 (1.43)"
```

- Good for a one-off project. Documentation? Reuse? Share?

Create a Package

- Well documented, shareable functions.
- Easy to use on multiple projects.
- Many very helpful tools exist to make package authorship easy.
  - Thank you, Hadley Wickham, for
    - `devtools` package
    - `roxygen2` package
    - The book *R packages* `http://r-pkgs.had.co.nz/`

Create a Package

Create the skeleton of an R package.

```
devtools::create("mypackage")
```

```
mypackage/
 |-- R
 |---DESCRIPTION
 |---mypackage.Rproj
 `---NAMESPACE

 1 directory, 3 files
```

Edit the DESCRIPTION file

- Package Meta Data
- http://r-pkgs.had.co.nz/description.html

Generated File:

```
Package: mypackage
Title: What the Package Does (one line, title case)
Version: 0.0.0.9000
Authors@R: person("First", "Last", , "first.last@example.com", role = c("aut", "cre"))
Description: What the package does (one paragraph)
Depends: R (>= 3.2.0)
License: What license is it under?
LazyData: true
```

Edited file:

```
Package: mypackage
Title: A collection of helper functions
Version: 0.0.0.9000
Authors@R: person("Peter", "DeWitt", , "peter.dewitt@ucdenver", role = c("aut", "cre"))
Description: Commonly used formatting functions.  A minimalist set of functions
    used to show an example of building an R package.
Depends: R (>= 3.0.2)
License: GPL-2
LazyData: true
```

Add R Code

- To add R code to your package add a file to the `R/` directory:

```
mypackage/R/mean_sd.R
```

- Within this file we will author the R code and the corresponding documention via roxygen comments (prefaced with `#'`).
- The function `devtools::document()` will parse the R file(s) and populate the needed `man/` files.

- The following slides show the contents of the `mypackage/R/mean_sd.R` file. This version was copied from the `qwraps2` package I'm developing.
  - `http://cran.r-project.org/web/packages/qwraps2/`
  - `https://github.com/dewittpe/qwraps2`

```
#' @title Mean and Standard deviation
#'
#' @description A function for calculating and formatting means and
#' standard deviations.
#'
#' @details
#' Given a numeric vector, \code{mean_sd} will return a character string with
#' the mean and standard deviation.  Formating of the output will be extended in
#' future versions.
#'
#' @param x a numeric vector
#' @param digits digits to the right of the decimal point to return in the
#' percentage estimate.
#' @param na_rm if true, omit NA values
#' @param show_n defaults to "ifNA".  Other options are "always" or "never".
#' @param denote_sd a character string set to either "pm" or "paren" for reporting
#' 'mean \eqn{\pm} sd' or 'mean (sd)'
#' @param markup latex or markdown
#'
#' @return a character vector of the formatted values
#'
#' @examples
#' set.seed(42)
#' x <- rnorm(1000, 3, 4)
#' mean(x)
#' sd(x)
#' mean_sd(x)
#' mean_sd(x, show_n = "always")
#' mean_sd(x, show_n = "always", denote_sd = "paren")
#'
#' x[187] <- NA
#' mean_sd(x, na_rm = TRUE)
#'
#' @export
mean_sd <- function(x,
                    digits = getOption("qwraps2_frmt_digits", 2),
                    na_rm = FALSE,
                    show_n = "ifNA",
                    denote_sd = "pm",
                    markup = getOption("qwraps2_markup", "latex")) {
  n <- sum(!is.na(x))
  m <- mean(x, na.rm = na_rm)
  s <- sd(x, na.rm = na_rm)

  if (show_n =="always" | any(is.na(x))) {
    rtn <- paste0(frmt(as.integer(n), digits), "; ", frmt(m, digits),
                  " $\\pm$ ", frmt(s, digits))
  } else {
    rtn <- paste0(frmt(m, digits), " $\\pm$ ", frmt(s, digits))
  }
  if (denote_sd == "paren") {
    rtn <- gsub("\\$\\\\pm\\$\\s(.*)", "\\(\\1\\)", rtn)
  }

  if (markup == "markdown") {
    rtn <- gsub("\\$\\\\pm\\$", "&plusmn;", rtn)
  }

  return(rtn)
}
```

Current Directory Structure

```
mypackage/
 |-- R
 |    `-- mean_sd.R
 |-- DESCRIPTION
 |-- mypackage.Rproj
 `-- NAMESPACE

 1 directory, 4 files
```

Generate the required documentation

```r
devtools::document('mypackage')

# Updating mypackage documentation
# Loading mypackage
# First time using roxygen2 4.0 Upgrading automatically...
# Writing NAMESPACE
# Writing mean_sd.Rd
```

```
mypackage/
  |-- man
  |    `-- mean_sd.Rd
  |-- R
  |    `-- mean_sd.R
  |-- DESCRIPTION
  |-- mypackage.Rproj
  `-- NAMESPACE

  2 directory, 5 files
```

The next couple slides show the contents of `mean_sd.Rd`

```
% Generated by roxygen2 (4.1.0): do not edit by hand
% Please edit documentation in R/mean_sd.R
\name{mean_sd}
\alias{mean_sd}
\title{Mean and Standard deviation}
\usage{
mean_sd(x, digits = getOption("qwraps2_frmt_digits", 2), na_rm = FALSE,
  show_n = "ifNA", denote_sd = "pm", markup = getOption("qwraps2_markup",
  "latex"))
}
\arguments{
\item{x}{a numeric vector}

\item{digits}{digits to the right of the decimal point to return in the
percentage estimate.}

\item{na_rm}{if true, omit NA values}

\item{show_n}{defaults to "ifNA".  Other options are "always" or "never".}

\item{denote_sd}{a character string set to either "pm" or "paren" for reporting
'mean \eqn{\pm} sd' or 'mean (sd)'}

\item{markup}{latex or markdown}
}
```

```
\value{
a character vector of the formatted values
}
\description{
A function for calculating and formatting means and
standard deviations.
}
\details{
Given a numeric vector, \code{mean_sd} will return a character string with
the mean and standard deviation.  Formating of the output will be extended in
future versions.
}
\examples{
set.seed(42)
x <- rnorm(1000, 3, 4)
mean(x)
sd(x)
mean_sd(x)
mean_sd(x, show_n = "always")
mean_sd(x, show_n = "always", denote_sd = "paren")

x[187] <- NA
mean_sd(x, na_rm = TRUE)
}
```

- I've added code for a formating function `frmt()` to the example package too.
- Evaluated `devtools::document('mypackage/')` and

```
mypackage/
 |-- man
 |    |-- frmt.Rd
 |    `-- mean_sd.Rd
 |-- R
 |    |-- frmt.R
 |    `-- mean_sd.R
 |-- DESCRIPTION
 |-- mypackage.Rproj
 `-- NAMESPACE

2 directory, 7 files
```

Building the Package

```
devtools::build("mypackage/")
# 'usr/lib/R/bin/R' --vanilla CMD build \
# '/home/dewittpe/drug--r-pkg-talk/mypackage' \
# --no-resave-data --noanual
#
# * checking for file '/home/dewittpe/drug-r-pkg-talk/mypackage/DESCRIPTION'
# * preparing 'mypackage':
# * checking DESCRIPTION meta-information ... OK
# * checking for LF line-endings in source and make files
# * checking for empty or unneeded directories
# * building 'mypackage_0.0.0.9000.tar.gz'
#
# [1] "/home/dewittpe/drug-r-pkg-talk/mypackage_0.0.0.9000.tar.gz"
```

- Can also be done from the command line via `R CMD build`.
- `devtools::install()` is calling `R CMD build`.
- Send the `.tar.gz` files to colaborators to install the package on their machine(s).

Install the Package

- Do so via `R CMD INSTALL`, or
- `install.packages()`, or
- `devtools::install()`.

```r
# use with_libpaths() to change the library the package is installed too
devtools::with_libpaths("r-dev", devtools::install("mypackage/"))
## Installing mypackage
## '/usr/lib/R/bin/R' --vanilla CMD INSTALL  \
##   '/home/dewittpe/drug-r-pkg-talk/mypackage'  \
##   --library='/home/dewittpe/drug-r-pkg-talk/r-dev' --install-tests
##
## * installing *source* package mypackage ...
## ** R
## ** preparing package for lazy loading
## ** help
## *** installing help indices
## ** building package indices
## ** testing if installed package can be loaded
## * DONE (mypackage)
```

Check the install

```r
rm(list = ls())
data("diamonds", package = "ggplot2")

# errors... package not loaded and attached
mean_sd(diamonds$price, markup = "markdown")

## Error in eval(expr, envir, enclos):  could not find function "mean_sd"

# Load and attach the package
library(package = "mypackage", lib.loc = "r-dev/")
mean_sd(diamonds$price)

## [1] "3,932.80 $\\pm$ 3,989.44"

mean_sd(diamonds$price, markup = "markdown")

## [1] "3,932.80 &plusmn; 3,989.44"
```

Using Code from Other Packages

- First, you'll need to edit the `DESCRIPTION` file for your package.
  - `Imports` packages (loaded by namespace),
  - `Depends` on is 'poor form.'
- Second, the `::` operator is your friend.
  - Requires the package to be loaded.
  - Does not require attaching a package.
  - Robust to end user's attached, and order of attaching, packages.
- Next slide: updated `DESCRIPTION` file.
- We'll look at the file mypackage/R/qroc.R

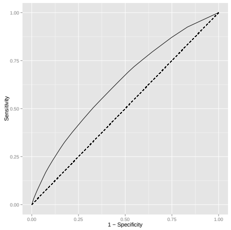
Updated DESCRIPTION file

```
Package: mypackage
Title: A collection of helper functions
Version: 0.0.0.9000
Authors@R: person("Peter", "DeWitt", , "peter.dewitt@ucdenver", role = c("aut", "cre'
Description: Commonly used formatting functions.  A minimalist set of functions
    used to show an example of building an R package.
Depends: R (>= 3.0.2)
License: GPL-2
LazyData: true
Imports:
  ggplot2
```

Impact of calling code form other packages via ::

We can plot a ROC curve, but cannot change the theme to black and white becuase `ggplot2` is not attached.

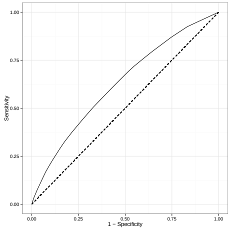
```
fit1 <- glm(formula = I(price > 2800) ~ cut * color,
            data    = diamonds,
            family  = binomial())
qroc(fit1)
```



```
qroc(fit1) + theme_bw()

## Error in eval(expr, envir, enclos):  could not find function "theme_bw"
```

```r
library(ggplot2)
qroc(fit1) + theme_bw()
```



```r
# The same graphic could be generated via
qroc(fit1) + ggplot2::theme_bw()
# with or without attaching ggplot2
```

Compiled Code

- Do you have some C++ code that you'd like to have access to in R?
- `Rcpp` will be very helpful!
- `Rcpp.package.skeleton()` and the Rcpp-package vignette for more details.

Package Checks

- Use `devtools::check()`, or
- `R CMD check`.

- Getting a package, espeically your first one, to pass the check is difficult.
- Best documentation, and notes to help prevent common errors: `http://r-pkgs.had.co.nz/check.html`

Packages on github.com

- There are many R packages on github.
    - Development versions of what is on CRAN.
    - Some only available on github.
- Version control, releases, issue tracking, ...
- Others can install your package via

```
devtools::install_github()
```

- Barebone websites
- *R packages* has a great chapter on git: `http://r-pkgs.had.co.nz/git.html`
- `devtools` has functions for installing from `bitbucket.org` and other hosting sites.

Submitting to CRAN

If you are going to submit your pakage to CRAN, the package needs to meet the CRAN Repository Policy `http://cran.r-project.org/web/packages/policies.html` and there is a web form for submission, `http://xmpalantir.wu.ac.at/cransubmit/`.

- Check your package via
  - `R CMD check`, or
  - `devtools::check()`.
- ERRORS, WARNINGS, and NOTES will be returned.
  - Correct all ERRORS before submitting
  - Correct as many WARNINGS as possible (preferably all)
  - Address all NOTES
- CRAN reviewers are mean, blunt, pompus, …(All of which are well deserved and earned traits)

My Favorite NOTE

```r
# This function, in a package, will result in two NOTES
# R CMD check
#
# * checking R code for possible problems ... NOTE
# aplot: no visible binding for global variable xvec
# aplot: no visible binding for global variable yvec
aplot <- function(x, y) {
  this_data <- data.frame(xvec = mtcars[, x], yvec = mtcars[, y])

  ggplot2::ggplot(this_data) +
  ggplot2::aes(x = xvec, y = yvec) +
  ggplot2::geom_point()
}
# Passes the R CMD check
aplot <- function(x, y) {
  this_data <- data.frame(xvec = mtcars[, x], yvec = mtcars[, y])

  ggplot2::ggplot(this_data) +
  ggplot2::aes_string(x = "xvec", y = "yvec") +
  ggplot2::geom_point()
}
```

My Suggestions

- Read *R Packages* by Hadley Wickham, `http://r-pkgs.had.co.nz/`
- Thanks to `devtools` writting a package for personal use is reasonable for all levels of R users.
- Write simple, short functions. Many robust, simple, and specific functions is preferable to a few complex functions.
- Writting a package will help you learn a lot about how R works in general. It will help improve your coding overall.
- Host on github.com
- Use TravisCI
- It's FUN!
- Can be lucritive.
- Having a package on CRAN is a nice 'feather in your hat.'

Closing

- Matt will extend this talk by showing some specific development steps in RStudio.
- Side note: We are always looking for speakers. Any one, from novice to expert, is welcome to give a talk, even a short one.

Thank you for listening.
Questions?