Developing Your First R Package

A Case Study with esvis

Daniel Anderson
04-10-2018

UNIVERSITY OF OREGON

Want to follow along?

If you'd like to follow along, pleae make sure you have the following packages installed

```
install.packages(c("tidyverse", "esvis",
                   "devtools", "roxygen2",
                   "usethis"))
```

#whomi

- Research Assistant Professor (newly) in the College of Education
- Work a lot in growth modeling and measurement
- Preacher of the R gospel
- Dad of two amazing girls


the fam

Some review

- What is an R function?

Some review

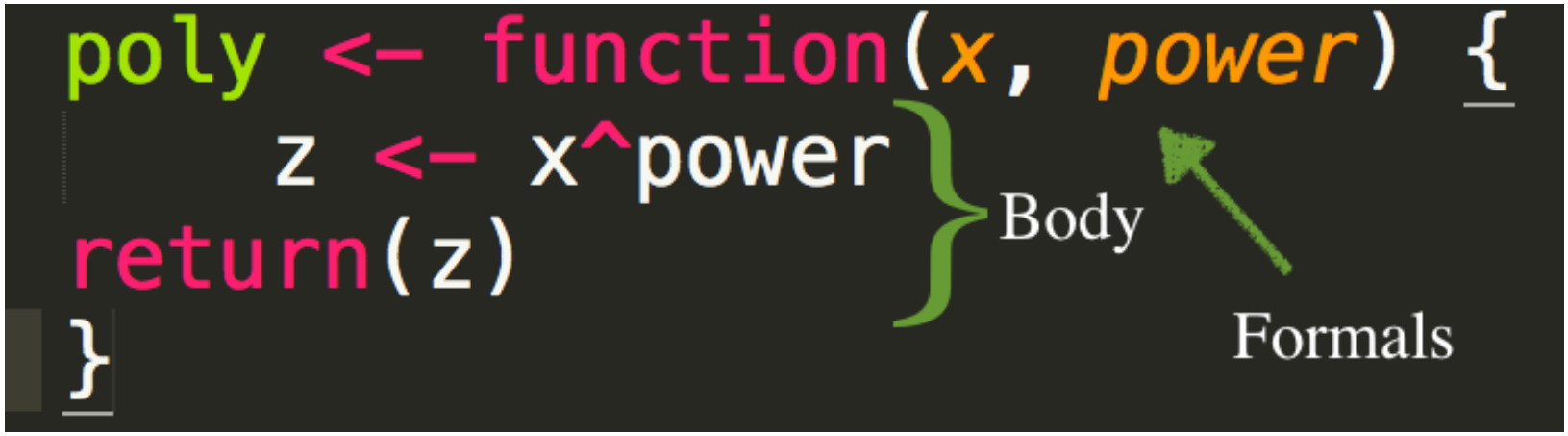
- What is an R function?
    - Anything that carries out an operation in R, including `+` and `<-`

Some review

- What is an R function?
    - Anything that carries out an operation in R, including `+` and `<-`
- What are the components of a function?

Some review

- What is an R function?
  - Anything that carries out an operation in R, including `+` and `<-`
- What are the components of a function?
  - Formals (arguments), Body (everything between the braces), and Environment (where the function lives)

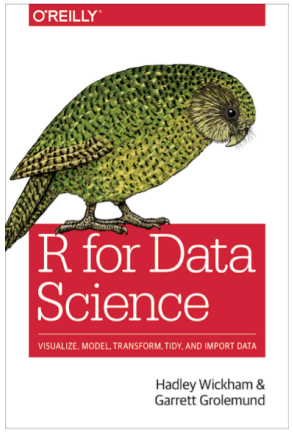


When should you write a function?

Hadley's rule

> You should consider writing a function whenever you've copied and pasted a block of code more than twice (i.e. you now have three copies of the same code).

r4ds



Hadley Wickham

@hadleywickham

Time to write a function

Fluff Society @FluffSociety

"Ctr+C, Ctr+V, Ctr+V, Ctr+V."

7:29 PM - 16 Sep 2017

Bundle your functions

Once you've written more than one function, you may want to bundle them. There are two general ways to do this:

Bundle your functions

Once you've written more than one function, you may want to bundle them. There are two general ways to do this:

source?

Write a package

Bundle your functions

Once you've written more than one function, you may want to bundle them. There are two general ways to do this:

source? Write a package

Reasons to avoid `sourceing`

- Documentation is generally more sparse
- Directory issues
  - Which leads to reproducibility issues

More importantly

Bundling functions into a package is not that hard!

my journey with esvis

Background

Effect sizes

Standardized mean differences

Background

Effect sizes

Standardized mean differences

- Assumes reasonably normally distributed distributions (mean is a good indicator of central tendency)

Background

Effect sizes

Standardized mean differences

- Assumes reasonably normally distributed distributions (mean is a good indicator of central tendency)
- Differences in means may not reflect differences at all points in scale if variances are different

Background

Effect sizes

Standardized mean differences

- Assumes reasonably normally distributed distributions (mean is a good indicator of central tendency)
- Differences in means may not reflect differences at all points in scale if variances are different
- Substantive interest may also lie with differences at other points in the distribution.

Varying differences

Quick simulated example

```
library(tidyverse)
common_var <- tibble(low  = rnorm(1000, 10, 1),
                     high = rnorm(1000, 12, 1),
                     var  = "common")
diff_var <- tibble(low  = rnorm(1000, 10, 1),
                   high = rnorm(1000, 12, 2),
                   var  = "diff")
d <- bind_rows(common_var, diff_var)
head(d)
```

```
## # A tibble: 6 x 3
##     low  high var
##   <dbl> <dbl> <chr>
## 1 11.6   11.3 common
## 2 10.6   11.5 common
## 3 10.2   13.9 common
## 4  9.23  12.5 common
## 5  9.44  12.0 common
## 6 11.4   10.8 common
```

Restructure the data for plotting

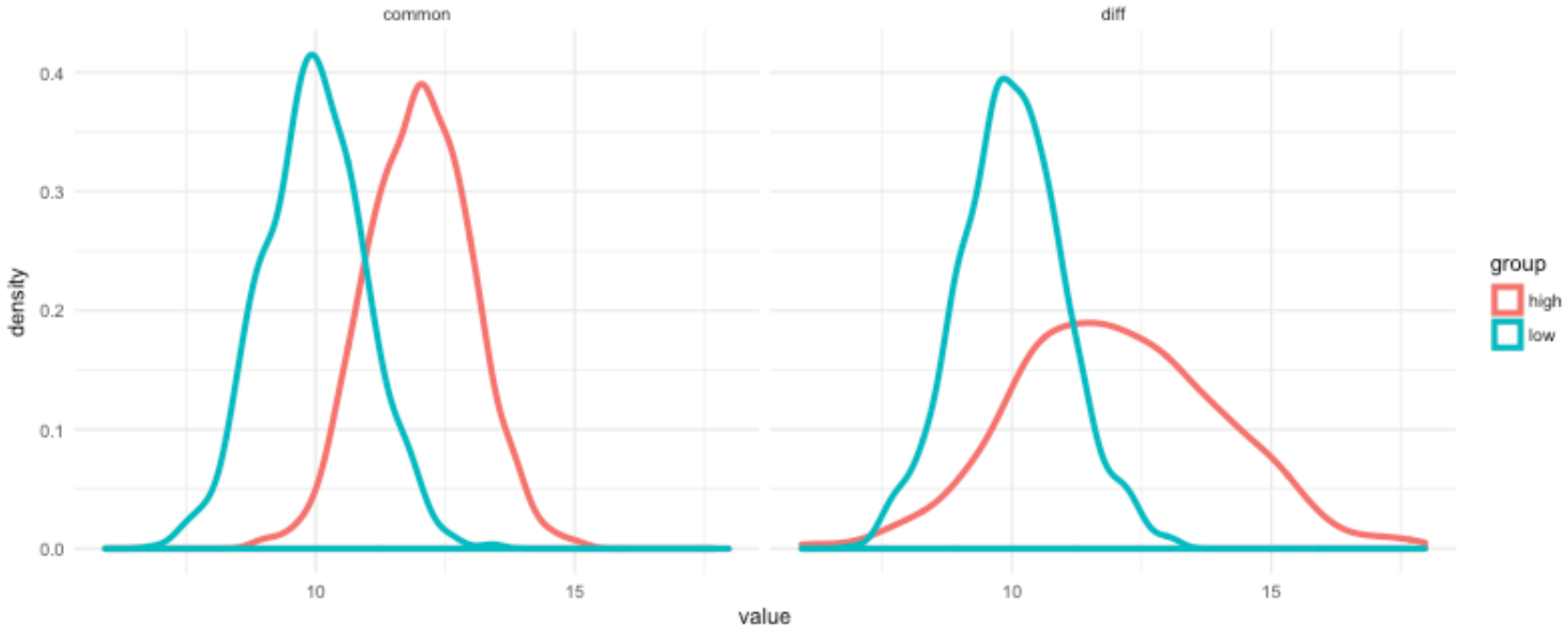
```
d <- d %>%
  gather(group, value, -var)
d
```

```
## # A tibble: 4,000 x 3
##    var    group value
##    <chr>  <chr> <dbl>
##  1 common low   11.6
##  2 common low   10.6
##  3 common low   10.2
##  4 common low    9.23
##  5 common low    9.44
##  6 common low   11.4
##  7 common low   10.6
##  8 common low    8.81
##  9 common low    9.21
## 10 common low    9.80
## # ... with 3,990 more rows
```

Plot the distributions

```
theme_set(theme_minimal())

ggplot(d, aes(value, color = group)) +
  geom_density(lwd = 1.5) +
  facet_wrap(~var)
```



Binned effect sizes

1. Cut the distributions into n bins (based on percentiles)
2. Calculate the mean difference between paired bins
3. Divide each mean difference by the overall pooled standard deviation

$$d_{[i]} = \frac{\bar{X}_{foc_{[i]}} - \bar{X}_{ref_{[i]}}}{\sqrt{\frac{(n_{foc}-1)Var_{foc}+(n_{ref}-1)Var_{ref}}{n_{foc}+n_{ref}-2}}}$$

Binned effect sizes

1. Cut the distributions into n bins (based on percentiles)
2. Calculate the mean difference between paired bins
3. Divide each mean difference by the overall pooled standard deviation

$$d_{[i]} = \frac{\bar{X}_{foc_{[i]}} - \bar{X}_{ref_{[i]}}}{\sqrt{\frac{(n_{foc}-1)Var_{foc}+(n_{ref}-1)Var_{ref}}{n_{foc}+n_{ref}-2}}}$$

visualize it!

Back to the simultated example

```
common <- filter(d, var == "common")
diff   <- filter(d, var == "diff")
```

```
library(esvis)
qtile_es(value ~ group, common)
```

```
##   ref_group foc_group low_qtile high_qtile midpoint         es         se
## 1      high       low      0.00       0.33    0.165 -2.029997 0.09597083
## 2      high       low      0.33       0.66    0.495 -2.060516 0.09631753
## 3      high       low      0.66       0.99    0.825 -2.065950 0.09640567
```

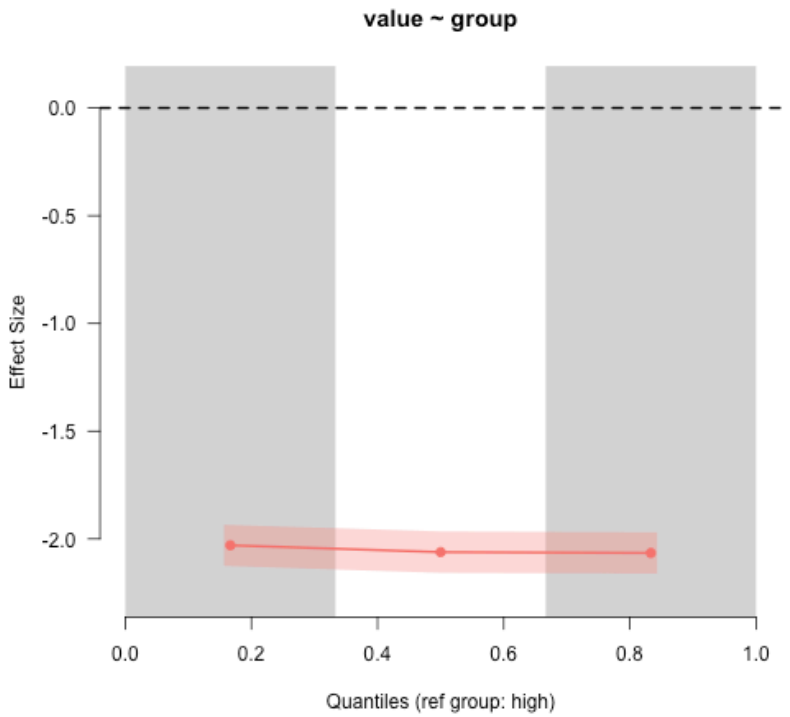
```
qtile_es(value ~ group, diff)
```

```
##   ref_group foc_group low_qtile high_qtile midpoint         es         se
## 1      high       low      0.00       0.33    0.165 -0.6246141 0.07984666
## 2      high       low      0.33       0.66    0.495 -1.2220534 0.08480555
## 3      high       low      0.66       0.99    0.825 -1.9406473 0.09441250
```

Visualize it

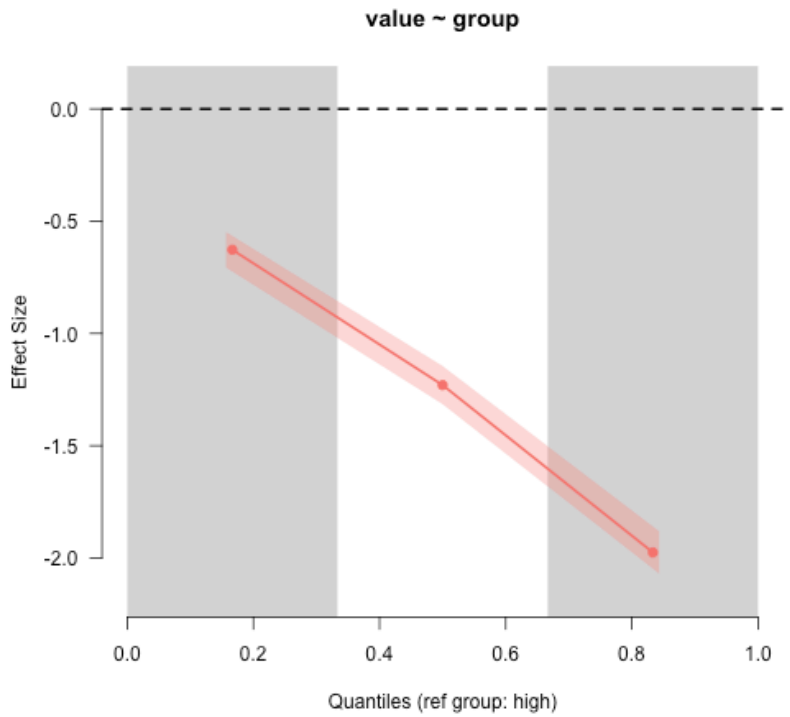
Common Variance

```
binned_plot(value ~ group, common)
```



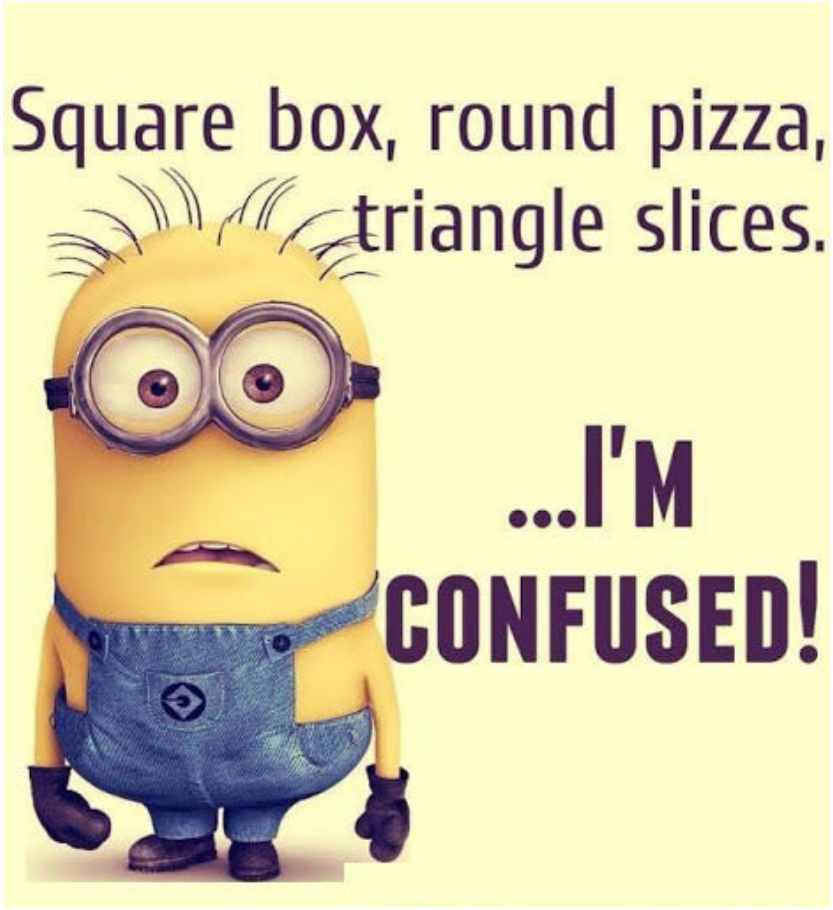
Different Variance

```
binned_plot(value ~ group, diff)
```



Wait a minute...

- The *esvis* package will (among other things) calculate and visually display binned effect sizes.
- But how did we get from an idea, to functions, to a package?



taking a step back

Package Creation

The (a) recipe

1. Come up with ~~a brilliant~~ an idea
    - can be boring and mundane but just something you do a lot

Package Creation

The (a) recipe

1. Come up with ~~a brilliant~~ an idea
    - can be boring and mundane but just something you do a lot
2. Write a function! or more likely, a set of functions

Package Creation

The (a) recipe

1. Come up with ~~a brilliant~~ an idea
    - can be boring and mundane but just something you do a lot
2. Write a function! or more likely, a set of functions
3. Create package skelton

Package Creation

The (a) recipe

1. Come up with ~~a brilliant~~ an idea
    - can be boring and mundane but just something you do a lot
2. Write a function! or more likely, a set of functions
3. Create package skelton
4. Document your function

Package Creation

The (a) recipe

1. Come up with ~~a brilliant~~ an idea
   - can be boring and mundane but just something you do a lot
2. Write a function! or more likely, a set of functions
3. Create package skelton
4. Document your function
5. Install/fiddle/install

Package Creation

The (a) recipe

1. Come up with ~~a brilliant~~ an idea
    - can be boring and mundane but just something you do a lot
2. Write a function! or more likely, a set of functions
3. Create package skelton
4. Document your function
5. Install/fiddle/install
6. Write tests for your functions

Package Creation

The (a) recipe

1. Come up with ~~a brilliant~~ an idea
   - can be boring and mundane but just something you do a lot
2. Write a function! or more likely, a set of functions
3. Create package skelton
4. Document your function
5. Install/fiddle/install
6. Write tests for your functions
7. Host your package somewhere public (GitHub is probably best) and promote it - leverage the power of open source!

Package Creation

The (a) recipe

1. Come up with ~~a brilliant~~ an idea
   - can be boring and mundane but just something you do a lot
2. Write a function! or more likely, a set of functions
3. Create package skelton
4. Document your function
5. Install/fiddle/install
6. Write tests for your functions
7. Host your package somewhere public (GitHub is probably best) and promote it - leverage the power of open source!

Use tools throughout (which we'll talk about momentarily) to help automate many of the steps, and make the whole thing less painful

A really good point

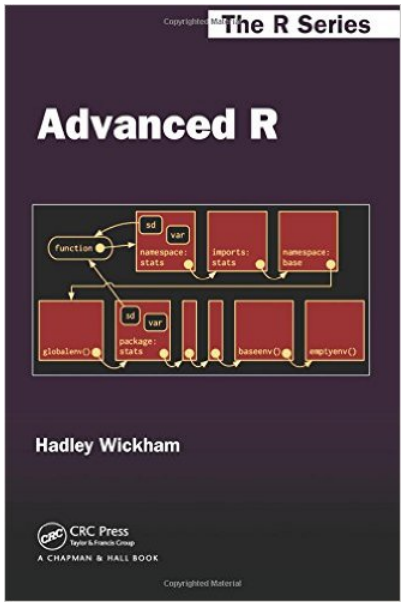
> 1a) check that no one had the same idea 😇
>
> — Maëlle Salmon 🐟 (@ma_salmon) April 10, 2018

And some further recommendations/good advice

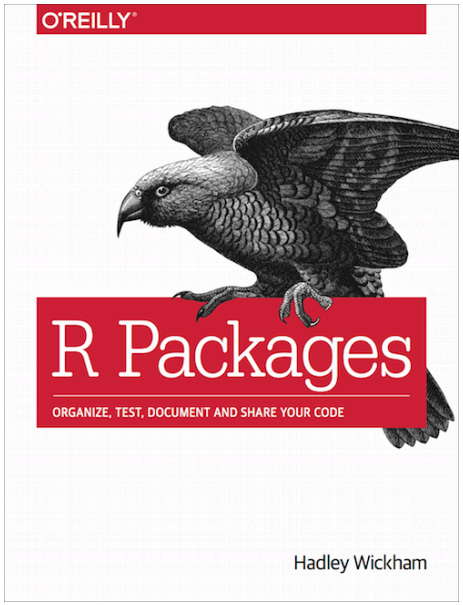
Some resources

We surely won't get through all the steps tonight. In my mind, the best resources are:

Advanced R



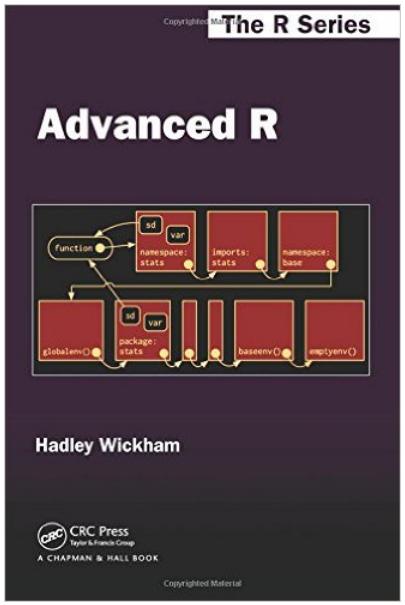
R Packages



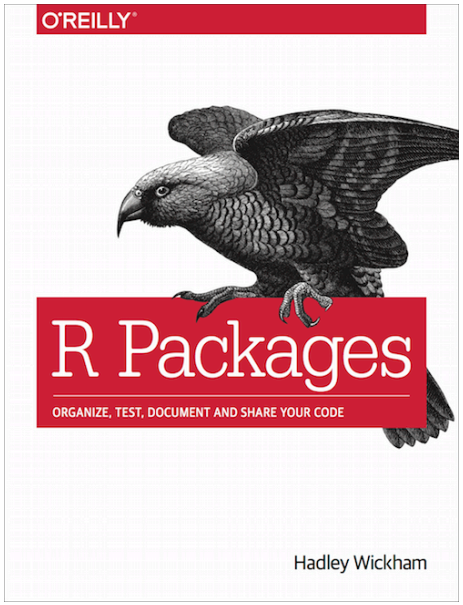
Some resources

We surely won't get through all the steps tonight. In my mind, the best resources are:

Advanced R



R Packages



For a really quick but really good intro, see Hilary Parker's blog post

Our package

We're going to write a package today! Let's keep it really simple...

1. Idea: Report basic descriptive statistics for a vector, `x`: `n`, `mean`, and `sd`. Let's also have it report on the number of missing observations.

Our function

- Let's have it return either (a) a named vector, or (b) a dataframe (whichever you prefer is fine)
- What will be the formal arguments?
- What will the body look like?

Our function

- Let's have it return either (a) a named vector, or (b) a dataframe
  (whichever you prefer is fine)
- What will be the formal arguments?
- What will the body look like?

Want to give it a go?

The approach I took...

```
describe <- function(x) {
  n     <- as.integer(length(na.omit(x)))
  nmiss <- as.integer(sum(is.na(x)))
  mn    <- mean(x, na.rm = TRUE)
  stdev <- sd(x, na.rm = TRUE)

  out <- tibble::tibble(n_valid   = n,
                        n_missing = nmiss,
                        mean      = mn,
                        sd        = stdev)
  out
}
```

The approach I took...

```
describe <- function(x) {
   n     <- as.integer(length(na.omit(x))) # Count number of valid cases
  nmiss <- as.integer(sum(is.na(x)))
  mn    <- mean(x, na.rm = TRUE)
  stdev <- sd(x, na.rm = TRUE)

  out <- tibble::tibble(n_valid   = n,
                        n_missing = nmiss,
                        mean      = mn,
                        sd        = stdev)
  out
}
```

The approach I took...

```
describe <- function(x) {
  n     <- as.integer(length(na.omit(x)))
   nmiss <- as.integer(sum(is.na(x))) # Count the number of missing
  mn    <- mean(x, na.rm = TRUE)
  stdev <- sd(x, na.rm = TRUE)

  out <- tibble::tibble(n_valid   = n,
                        n_missing = nmiss,
                        mean      = mn,
                        sd        = stdev)
  out
}
```

The approach I took...

```
describe <- function(x) {
  n     <- as.integer(length(na.omit(x)))
  nmiss <- as.integer(sum(is.na(x)))
   mn    <- mean(x, na.rm = TRUE) # Calculate mean
  stdev <- sd(x, na.rm = TRUE)

  out <- tibble::tibble(n_valid   = n,
                        n_missing = nmiss,
                        mean      = mn,
                        sd        = stdev)
  out
}
```

The approach I took...

```
describe <- function(x) {
  n     <- as.integer(length(na.omit(x)))
  nmiss <- as.integer(sum(is.na(x)))
  mn    <- mean(x, na.rm = TRUE)
   stdev <- sd(x, na.rm = TRUE) # Standard deviation

  out <- tibble::tibble(n_valid   = n,
                        n_missing = nmiss,
                        mean      = mn,
                        sd        = stdev)
  out
}
```

The approach I took...

```
describe <- function(x) {
  n     <- as.integer(length(na.omit(x)))
  nmiss <- as.integer(sum(is.na(x)))
  mn    <- mean(x, na.rm = TRUE)
  stdev <- sd(x, na.rm = TRUE)

   out <- tibble::tibble(n_valid   = n,      # Bundle it all
                         n_missing = nmiss,
                         mean      = mn,
                         sd        = stdev)
  out
}
```

The approach I took...

```
describe <- function(x) {
  n     <- as.integer(length(na.omit(x)))
  nmiss <- as.integer(sum(is.na(x)))
  mn    <- mean(x, na.rm = TRUE)
  stdev <- sd(x, na.rm = TRUE)

  out <- tibble::tibble(n_valid   = n,
                        n_missing = nmiss,
                        mean      = mn,
                        sd        = stdev)
   out # Return the tibble
}
```

Informal testing

```
describe(rnorm(100))
```

```
## # A tibble: 1 x 4
##   n_valid n_missing     mean    sd
##     <int>     <int>    <dbl> <dbl>
## 1     100         0 -0.00975 0.924
```

```
describe(c(rnorm(1000, 10, 4), rep(NA, 27)))
```

```
## # A tibble: 1 x 4
##   n_valid n_missing  mean    sd
##     <int>     <int> <dbl> <dbl>
## 1    1000        27  9.76  3.96
```

Demo

Package skeleton:

- `usethis::create_package`
- `usethis::use_r`
- Use `roxygen2` special comments for documentation
- Run `devtools::document`
- Install and restart, play around

roxygen2 comments

Typical arguments

- `@param`: Describe the formal arguments. State argument name and the describe it.

  ```
  #' @param x Vector to describe
  ```

- `@return`: What does the function return

  ```
  #' @return A tibble with descriptive data
  ```

- `@example` or more commonly `@examples`: Provide examples of the use of your function.
- `@export`: Export your function

If you don't include `@export`, your function will be *internal*, meaning others can't access it easily.

Other docs

- `.gitignore`: Files to ignore for git commits with some pre-slugged entries
- `NAMESPACE`: Created by **{roxygen2}**. Don't edit it. If you need to, trash it and it will be reproduced.
- `DESCRIPTION`: Describes your package (more on next slide)
- `man/`: The documentation files. Created by **{roxygen2}**. Don't edit.

DESCRIPTION

Metadata about the package. Default fields for our package are

```
Package: practice
Version: 0.0.0.9000
Title: What the Package Does (One Line, Title Case)
Description: What the package does (one paragraph).
Authors@R: person("First", "Last", email = "first.last@example.com", role = c("aut
License: What license is it under?
Encoding: UTF-8
LazyData: true
ByteCompile: true
RoxygenNote: 6.0.1
```

DESCRIPTION

Metadata about the package. Default fields for our package are

```
Package: practice
Version: 0.0.0.9000
Title: What the Package Does (One Line, Title Case)
Description: What the package does (one paragraph).
Authors@R: person("First", "Last", email = "first.last@example.com", role = c("aut
License: What license is it under?
Encoding: UTF-8
LazyData: true
ByteCompile: true
RoxygenNote: 6.0.1
```

This is where the information for `citation(package = "practice")` will come from.

DESCRIPTION

Metadata about the package. Default fields for our package are

```
Package: practice
Version: 0.0.0.9000
Title: What the Package Does (One Line, Title Case)
Description: What the package does (one paragraph).
Authors@R: person("First", "Last", email = "first.last@example.com", role = c("aut
License: What license is it under?
Encoding: UTF-8
LazyData: true
ByteCompile: true
RoxygenNote: 6.0.1
```

This is where the information for `citation(package = "practice")` will come from.

Some advice - edit within RStudio, or a good text editor like sublimetext.
"Fancy" quotes and things can screw this up.

Description File Fields

> The 'Package', 'Version', 'License', 'Description', 'Title', 'Author', and 'Maintainer' fields are mandatory, all other fields are optional.
>
> - Writing R Extensions

Some optional fields include

- Imports and Suggests (we'll do this in a minute).
- URL
- BugReports
- License (we'll have {usethis} create this for us).
- LazyData

DESCRIPTION for {esvis}

```
Package: esvis
Type: Package
Title: Visualization and Estimation of Effect Sizes
Version: 0.1.0.9000
Authors@R: person("Daniel", "Anderson", email = "daniela@uoregon.edu",
       role = c("aut", "cre"))
Description: A variety of methods are provided to estimate and visualize
    distributional differences in terms of effect sizes. Particular emphasis
    is upon evaluating differences between two or more distributions across
    the entire scale, rather than at a single point (e.g., differences in
    means). For example, Probability-Probability (PP) plots display the
    difference between two or more distributions, matched by their empirical
    CDFs (see Ho and Reardon, 2012; <doi:10.3102/1076998611411918>), allowing
    for examinations of where on the scale distributional differences are
    largest or smallest. The area under the PP curve (AUC) is an effect-size
    metric, corresponding to the probability that a randomly selected
    observation from the x-axis distribution will have a higher value
    than a randomly selected observation from the y-axis distribution.
    Binned effect size plots are also available, in which the distributions
    are split into bins (set by the user) and separate effect sizes (Cohen's
    d) are produced for each bin - again providing a means to evaluate the
    consistency (or lack thereof) of the difference between two or more
    distributions at different points on the scale. Evaluation of empirical
    CDFs is also provided, with  built-in arguments for providing annotations
```

DESCRIPTION for {esvis} (continued)

```
Depends:
    R (>= 3.1)
Imports:
    sfsmisc
URL: https://github.com/DJAnderson07/esvis
BugReports: https://github.com/DJAnderson07/esvis/issues
License: MIT + file LICENSE
LazyData: true
RoxygenNote: 6.0.1
Suggests:
    testthat,
    viridisLite
```

Demo

- Change the author name.
  - Add a contributer just for fun.
- Add a license. We'll go for MIT license using `usethis::use_mit_license("First and Last Name")`
- Install and reload.

Declare dependencies

- The function **depends on** the `tibble` function within the {tibble} package.
- We have to declare this dependency

Declare dependencies

- The function **depends on** the `tibble` function within the {tibble} package.
- We have to declare this dependency

My preferred approach

- Declare package dependencies: `usethis::use_package`
- Create a package documentation page: `usethis::use_package_doc`
  - Declare all dependencies for your package there
  - Only import the functions you need - not the entire package
    - Use `#' importFrom pkg fun_name`
- Generally won't have to worry about namespacing (`tibble::tibble` becomes just plain old `tibble`). The likelihood of conflicts is also reduced, so long as you don't import the full package.

Demo

Write tests!

- What does it mean to write tests?
  - ensure your package does what you expect it to

Write tests!

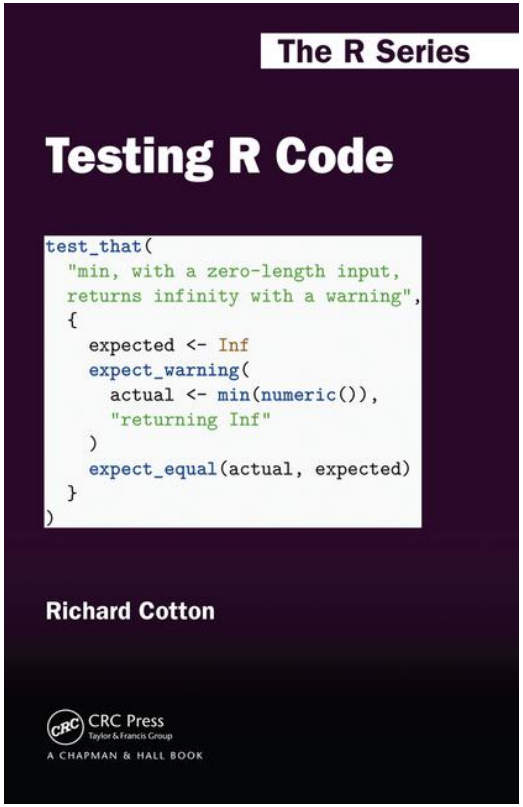
- What does it mean to write tests?
    - ensure your package does what you expect it to
- Why write tests?
    - If you write a new function, and it breaks an old one, that's good to know!
    - Reduces bugs, makes your package code more robust

Write tests!

- What does it mean to write tests?
  - ensure your package does what you expect it to
- Why write tests?
  - If you write a new function, and it breaks an old one, that's good to know!
  - Reduces bugs, makes your package code more robust
- How do you write tests?
  - `usethis::use_testthat` sets up the infrastructure
  - make assertions, e.g.: `testthat::expect_equal()`, `testthat::expect_warning()`, `testthat::expect_error()`

Testing

We'll skip over testing for today, because we just don't have time to cover everything. A few good resources:



Richie Cotton's book

r-pkgs Chapter

Karl Broman Blog Post

Check your R package

- Use `devtools::check()` to run the same checks CRAN will run on your R package.
  - Use `devtools::build_win()` to run the checks on CRAN computers.

Check your R package

- Use `devtools::check()` to run the same checks CRAN will run on your R package.
  - Use `devtools::build_win()` to run the checks on CRAN computers.

The first time, you'll likely get errors. Be patient. It will probably be frustrating, but ultimately worth the effort.

Let's check now!

🎉 Hooray! 🎉

You have a package!

A few other best practices

- Create a `README` with `usethis::use_readme_rmd`.

A few other best practices

- Create a `README` with `usethis::use_readme_rmd`.
- Try to get your code coverage up above 80%.

A few other best practices

- Create a `README` with `usethis::use_readme_rmd`.
- Try to get your code coverage up above 80%.
- Automate wherever possible ({devtools} and {usethis} help a lot with this)

A few other best practices

- Create a `README` with `usethis::use_readme_rmd`.
- Try to get your code coverage up above 80%.
- Automate wherever possible ({devtools} and {usethis} help a lot with this)
- Use the {goodpractice} package to help you package code be more robust, specifically with `goodpractice::gp()`. It will give you lots of good ideas

A few other best practices

- Create a `README` with `usethis::use_readme_rmd`.
- Try to get your code coverage up above 80%.
- Automate wherever possible ({devtools} and {usethis} help a lot with this)
- Use the {goodpractice} package to help you package code be more robust, specifically with `goodpractice::gp()`. It will give you lots of good ideas
- Host on GitHub, and capitalize on integration with other systems (all free, but require registering for an account)
    - Travis CI
    - Appveyor
    - codecov

Any time left?

Why you should use git and GitHub

esvis

Quickly

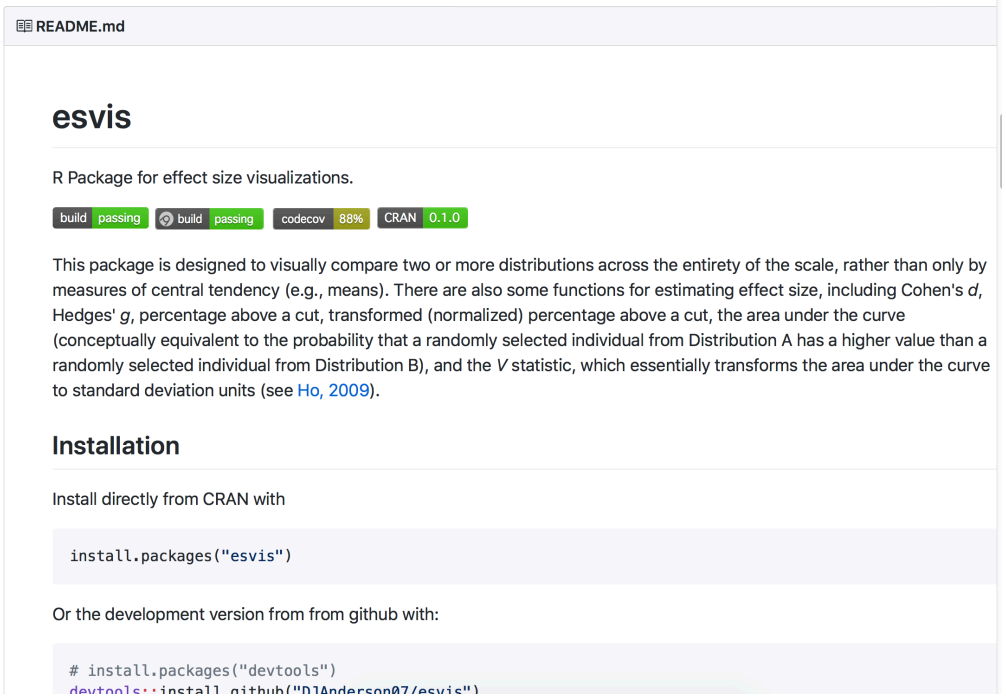
- Get started with `usethis::use_git`, followed by `usethis::use_github`.

> For this to work, you'll need to set a GITHUB_PAT environment variable in your ~/.Renviron. Follow Jenny Bryan's instructions, and use `edit_r_environ()` to easily access the right file for editing

Note: I haven't played around with this much. Standard git procedures will work too.

Create a README

- Use standard R Markdown. Setup the infrastructure with `usethis::use_readme_rmd`.
- Write it just like a normal R Markdown doc and it should all flow into the `README`.



README.md

esvis

R Package for effect size visualizations.

build passing | build passing | codecov 88% | CRAN 0.1.0

This package is designed to visually compare two or more distributions across the entirety of the scale, rather than only by measures of central tendency (e.g., means). There are also some functions for estimating effect size, including Cohen's *d*, Hedges' *g*, percentage above a cut, transformed (normalized) percentage above a cut, the area under the curve (conceptually equivalent to the probability that a randomly selected individual from Distribution A has a higher value than a randomly selected individual from Distribution B), and the *V* statistic, which essentially transforms the area under the curve to standard deviation units (see Ho, 2009).

Installation

Install directly from CRAN with

```
install.packages("esvis")
```

Or the development version from from github with:

```
# install.packages("devtools")
devtools::install_github("DJAnderson07/esvis")
```

Use Travis/Appveyor

- Register for a free account
- Run `usethis::use_travis` and `usethis::use_appveyor` to get started.
  - Go to each respective website and "turn on" the repo
  - Copy and paste the code to the badge into your `README`.

Use Travis/Appveyor

- Register for a free account
- Run `usethis::use_travis` and `usethis::use_appveyor` to get started.
  - Go to each respective website and "turn on" the repo
  - Copy and paste the code to the badge into your `README`.

- Now all your code will be automatically tested on Mac/Linux (Travis CI) and Windows (Appveyor)

codevoc

You can test your code coverage each time you push a new commit by using codecov. Initialize with `usethis::use_coverage()`. Overall setup process is pretty similar to Travis CI/Appveyor.

Easily see what is/is not covered by tests!

```
121  2          call <- as.list(match.call())
122
123  2          calcs <- pp_calcs(formula, data, ref_group)
124
125  2          if(!is.null(grid)) {
126                     if(grid & !is.null(cut)) {
127                             warning(paste("Grid suppressed when cut lines are also provided",
128                                     "(plot becomes overly busy otherwise)."))
129                     }
130  2          }
131
132  2          if(!is.null(cut)) grid <- FALSE
133  2          if(is.null(cut) & is.null(grid))  grid <- TRUE
134
135  2          if(!is.null(leg)) {
136  2                  if(leg == "side") n_1 <- TRUE
137  2          }
138  2          if(ncol(calcs$ps) > 2 & !is.null(shade)) {
139  2                  if(shade == TRUE) {
140  2                          warning(
141  2                                  paste("The area under the curve can only be shaded with two",
142  2                                          "groups. Argument `shade = TRUE` ignored")
143  2                                  )
144  2                  }
145  2          }
146  2          if(ncol(calcs$ps) > 2 & !is.null(text)) {
147  2                  if(text == TRUE) {
148  2                          warning(
```

That's all

Thanks so much!