Developing Your First R Package

Daniel Anderson

Week 10

Agenda

Want to follow along?

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

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

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

Why avoid `source`ing

- Documentation is generally more sparse
- Directory issues
    - Which leads to reproducibility issues
    - This is also less of an issue if you're using RStudio Projects and {here}

More importantly

Bundling functions into a package is not that hard!

My journey with {esvis}

My first CRAN package

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

```r
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  9.549110 13.50911 common
## 2  9.350832 12.93887 common
## 3 10.40181  12.20612 common
## 4 10.81037  10.52069 common
## 5 10.79728  12.71338 common
## 6  9.672677 10.68060 common
```

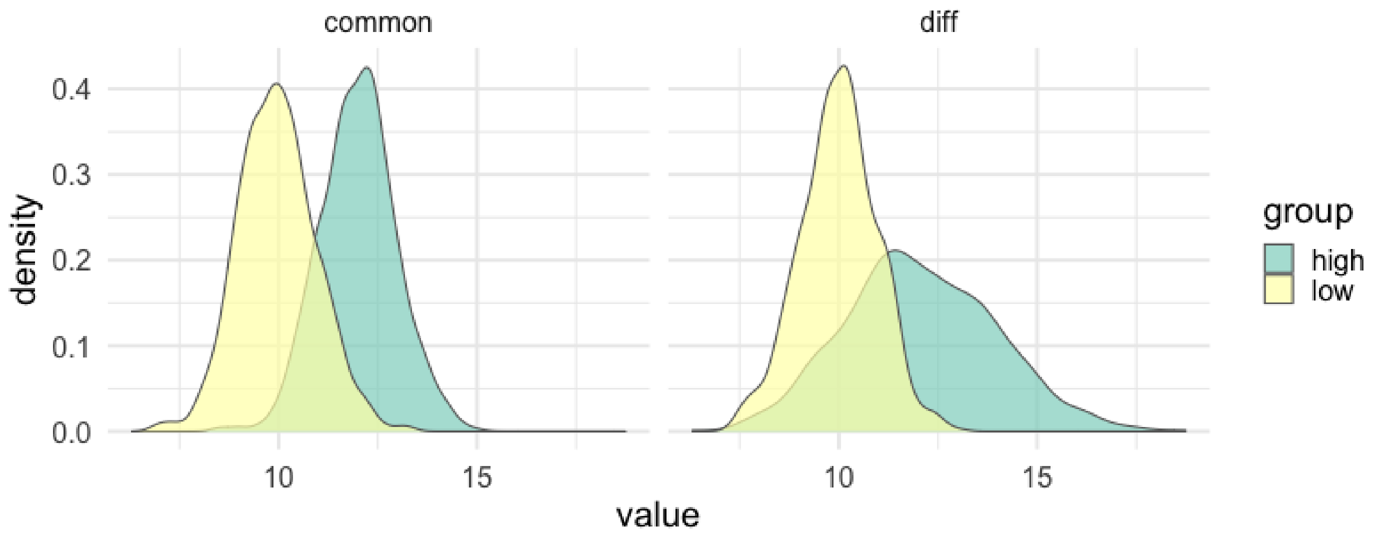
Restructure for plotting

```r
d <- d %>%
  pivot_longer(
    -var,
    names_to = "group",
    values_to = "value"
  )
d
```

```
## # A tibble: 4,000 x 3
##    var    group      value
##    <chr>  <chr>      <dbl>
##  1 common low     9.549110
##  2 common high   13.50911
##  3 common low     9.350832
##  4 common high   12.93887
##  5 common low    10.40181
##  6 common high   12.20612
##  7 common low    10.81037
##  8 common high   10.52069
##  9 common low    10.79728
## 10 common high   12.71338
## # … with 3,990 more rows
```

Plot the distributions

```r
ggplot(d, aes(value, fill = group)) +
  geom_density(alpha = 0.7,
               color = "gray40") +
  facet_wrap(~var) +
  scale_fill_brewer(palette = "Set3")
```



Binned effect sizes

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

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

visualize it!

Back to the simulated example

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

```r
library(esvis)
binned_es(common, value ~ group)
```

```
## # A tibble: 6 x 11
##       q  qtile_lb  qtile_ub group_ref group_foc mean_diff length length1
##   <dbl>     <dbl>     <dbl> <chr>     <chr>         <dbl>  <int>   <int>
## 1     1 0         0.3333333 high      low       -2.048922   1000    1000
## 2     2 0.3333333 0.6666667 high      low       -2.080733   1000    1000
## 3     3 0.6666667 1         high      low       -1.992408   1000    1000
## 4     1 0         0.3333333 low       high       2.048922   1000    1000
## 5     2 0.3333333 0.6666667 low       high       2.080733   1000    1000
## 6     3 0.6666667 1         low       high       1.992408   1000    1000
```

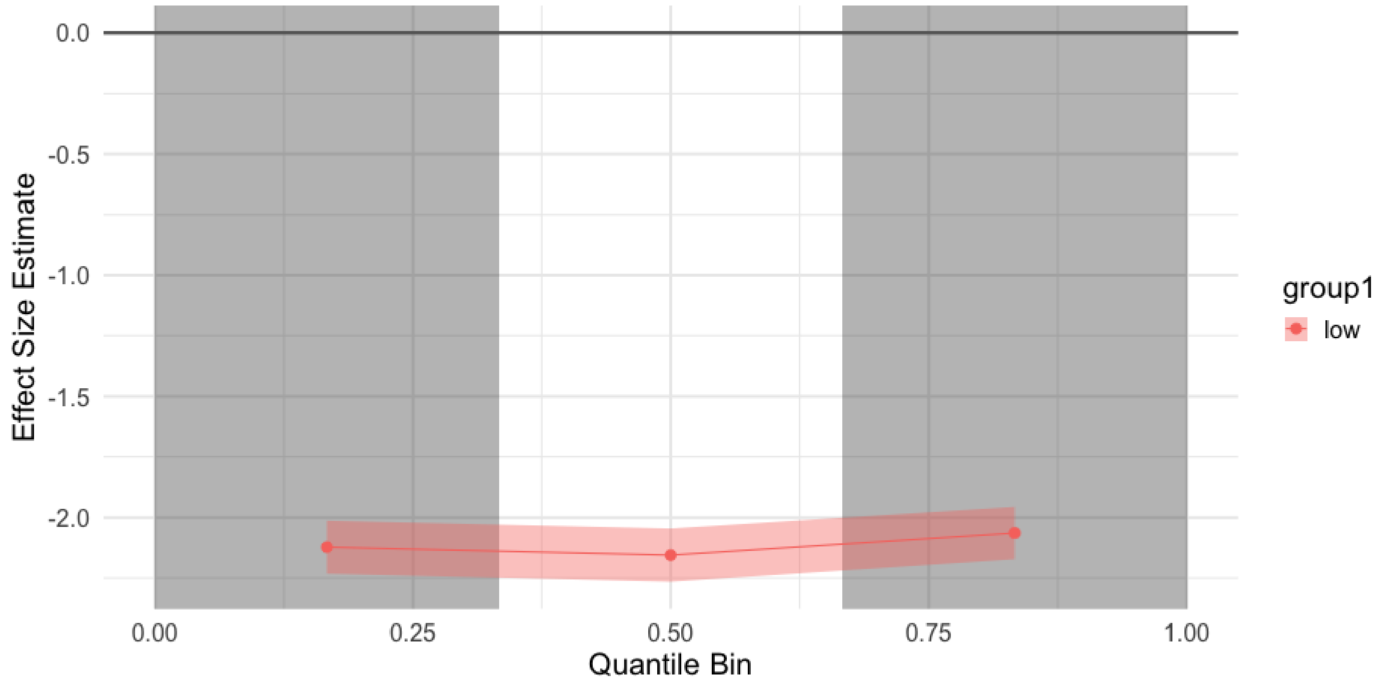
```r
binned_es(diff, value ~ group)
```

```
## # A tibble: 6 x 11
##       q  qtile_lb  qtile_ub group_ref group_foc mean_diff length length1
##   <dbl>     <dbl>     <dbl> <chr>     <chr>         <dbl>  <int>   <int>
## 1     1 0         0.3333333 high      low       -1.098328   1000    1000
## 2     2 0.3333333 0.6666667 high      low       -1.967218   1000    1000
## 3     3 0.6666667 1         high      low       -3.133526   1000    1000
## 4     1 0         0.3333333 low       high       1.098328   1000    1000
## 5     2 0.3333333 0.6666667 low       high       1.967218   1000    1000
## 6     3 0.6666667 1         low       high       3.133526   1000    1000
```

Visualize it

Common Variance

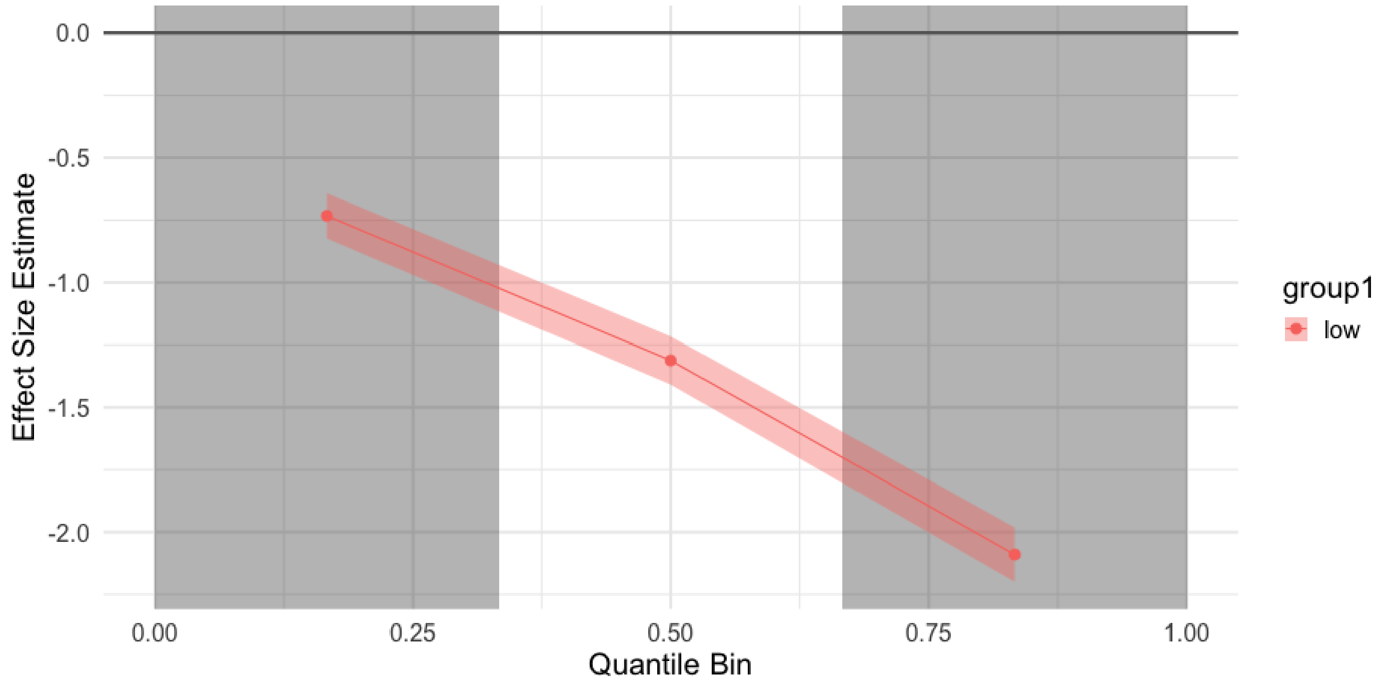
```
binned_plot(common, value ~ group)
```



Visualize it

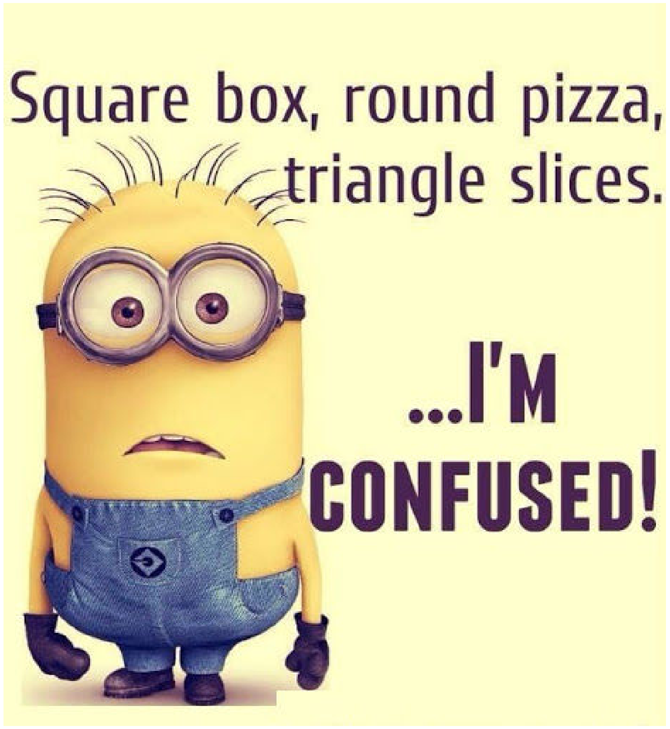
Different Variance

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



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

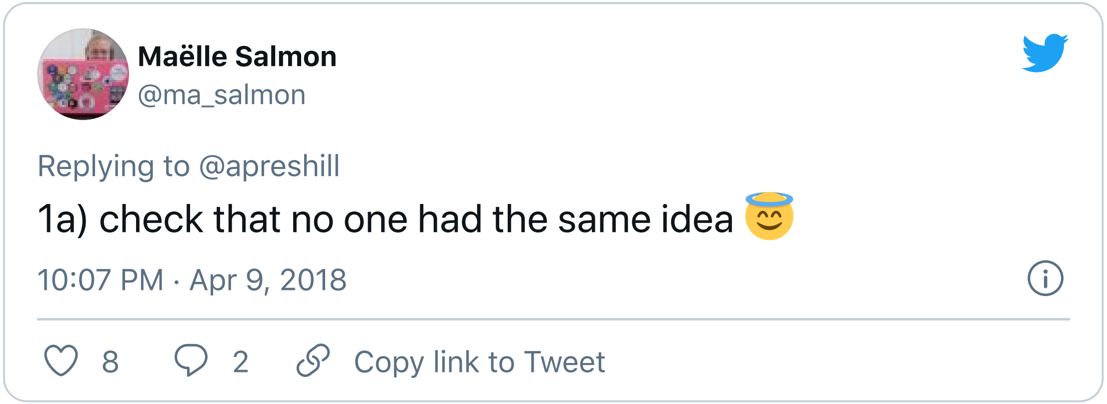
Package Creation

The (or rather 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 to automate

A really good point



Maëlle Salmon
@ma_salmon

Replying to @apreshill

1a) check that no one had the same idea 😇

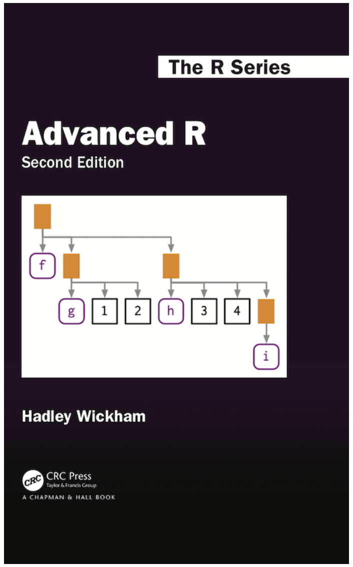
10:07 PM · Apr 9, 2018

And some further recommendations/good advice

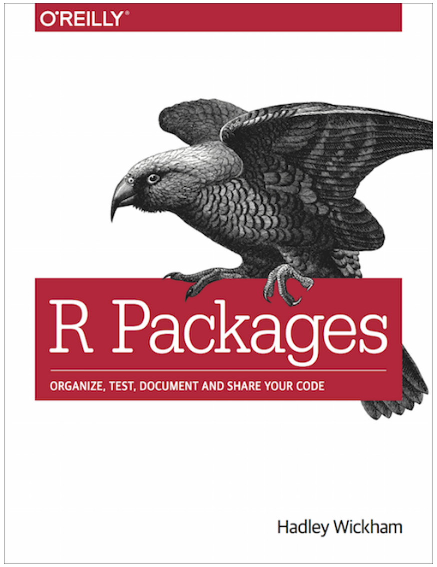
Some resources

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

Advanced R



R Packages



Our package

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

1. Idea (which we've actually used before): Report basic descriptive statistics for a vector, `x`: `N`, `n-valid`, `n-missing`, `mean`, and `sd`.

Our function

- Let's have it return a data frame
- What will be the formal arguments?
- What will the body look like?

Want to give it a go?

The approach I took...

```r
describe <- function(data, column_name) {
  x <- data[[column_name]]

  nval  <- length(na.omit(x))
  nmiss <- sum(is.na(x))
  mn    <- mean(x, na.rm = TRUE)
  stdev <- sd(x, na.rm = TRUE)

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

The approach I took...

```r
describe <- function(data, column_name) {
  x <- data[[column_name]] # Extract just the vector to summariz

  nval  <- length(na.omit(x))
  nmiss <- sum(is.na(x))
  mn    <- mean(x, na.rm = TRUE)
  stdev <- sd(x, na.rm = TRUE)

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

The approach I took...

```r
describe <- function(data, column_name) {
  x <- data[[column_name]]

  nval  <- length(na.omit(x)) # Count non-missing
  nmiss <- sum(is.na(x)) # Count missing
  mn    <- mean(x, na.rm = TRUE) # Compute mean
  stdev <- sd(x, na.rm = TRUE) # Computer SD

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

The approach I took...

```r
describe <- function(data, column_name) {
  x <- data[[column_name]]

  nval  <- length(na.omit(x))
  nmiss <- sum(is.na(x))
  mn    <- mean(x, na.rm = TRUE)
  stdev <- sd(x, na.rm = TRUE)

  # Compile into a df
  out <- tibble::tibble(N         = nval + nmiss,
                        n_valid   = nval,
                        n_missing = nmiss,
                        mean      = mn,
                        sd        = stdev)
  out
}
```

The approach I took...

```r
describe <- function(data, column_name) {
  x <- data[[column_name]]

  nval     <- length(na.omit(x))
  nmiss <- sum(is.na(x))
  mn    <- mean(x, na.rm = TRUE)
  stdev <- sd(x, na.rm = TRUE)

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

Informal testing

```r
set.seed(8675309)
df1 <- tibble(x = rnorm(100))
df2 <- tibble(var_miss = c(rnorm(1000, 10, 4), rep(NA, 27)))
describe(df1, "x")
```

```
## # A tibble: 1 x 5
##       N n_valid n_missing       mean        sd
##   <int>   <int>     <int>      <dbl>     <dbl>
## 1   100     100         0 0.05230278 0.9291437
```

```r
describe(df2, "var_miss")
```

```
## # A tibble: 1 x 5
##       N n_valid n_missing     mean       sd
##   <int>   <int>     <int>    <dbl>    <dbl>
## 1  1027    1000        27 9.881107 4.090208
```

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

- `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 =
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 or VSCode. "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: 0.3.1
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 emphasi
    is upon evaluating differences between two or more distributions acros
    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 empirica
    CDFs (see Ho and Reardon, 2012; <doi:10.3102/1076998611411918>), allowi
    for examinations of where on the scale distributional differences are
    largest or smallest. The area under the PP curve (AUC) is an effect-siz
    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'
    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 annotatio
    to help evaluate distributional differences at specific points (e.g.,
    semi-transparent shading). All function take a consistent argument
    structure. Calculation of specific effect sizes is also possible. The
```

DESCRIPTION for {esvis} (continued)

```
Depends:
    R (>= 3.1)
Imports:
    sfsmisc,
    ggplot2,
    magrittr,
    dplyr,
    rlang,
    tidyr (>= 1.0.0),
    purrr,
    Hmisc,
    tibble
URL: https://github.com/datalorax/esvis
BugReports: https://github.com/datalorax/esvis/issues
License: MIT + file LICENSE
LazyData: true
RoxygenNote: 7.0.2
Suggests:
    testthat,
    viridisLite
```

Demo

- Change the author name.
    - Add a contributor 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

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. The likelihood of conflicts is also reduced, so long as you don't import the full package.

Demo

Write tests!

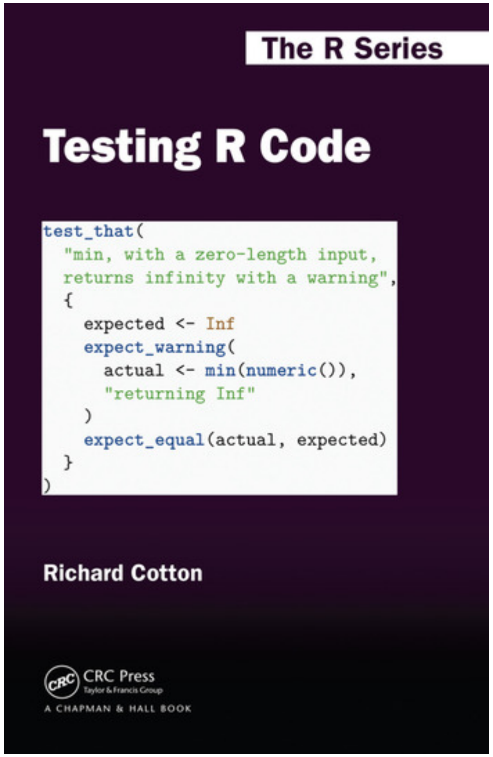
- What does it mean to write tests?
    - ensure your package does what you expect it to
- Why write tests?
    - If you write a new function, and it breaks an old one, that's good to know!
    - Reduces bugs, makes your package code more robust

How

- `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:


The R Series

Testing R Code

```
test_that(
  "min, with a zero-length input,
  returns infinity with a warning",
  {
    expected <- Inf
    expect_warning(
      actual <- min(numeric()),
      "returning Inf"
    )
    expect_equal(actual, expected)
  }
)
```

Richard Cotton

CRC Press Taylor & Francis Group
A CHAPMAN & HALL BOOK

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::check_rhub()` to test your package on https://builder.r-hub.io/ (several platforms and R versions)
    - Use `devtools::build_win()` to run the checks on CRAN computers.

I would not run the latter two until you're getting close to being ready to submit to CRAN.

Patience

The first time, you'll likely get errors. 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`.
- 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

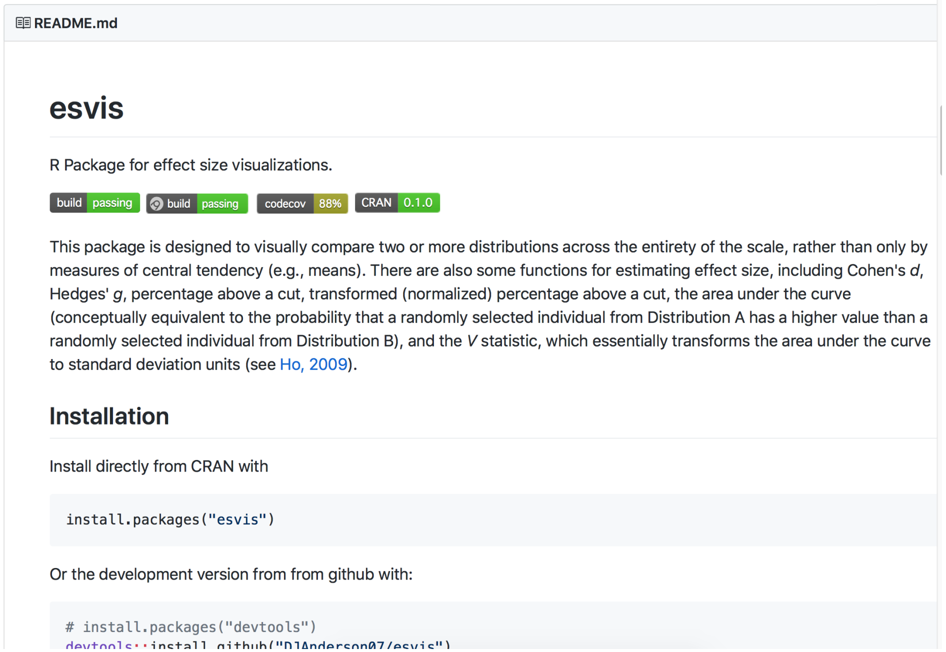
A few other best practices

- Host on GitHub, and capitalize on integration with other systems (all free, but require registering for an account)
    - Github Actions
    - codecov

Any time left?

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 GitHub Actions

- Run `usethis::use_github_actions()` to get started.
  - Go to the Actions tab on your repo
  - Copy and paste the code to the badge into your `README`.
- Now all your code will be automatically tested each time you push! You can even test on different operating systems and different versions of R!

codecov

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!

That's all

Thanks so much!