ETC5523: Communicating with Data

Stylishly communicating with code

Lecturer: *Emi Tanaka*

Department of Econometrics and Business Statistics

emi.tanaka@monash.edu

Week 9

cwd.numbat.space

Aim

- Document your functions
- Create vignettes for your package
- Implement testing framework for your package
- Distribute your package

Why

- Documentation informs users of how to use your package
- Adopting best practice development workflow will make package development easier
- Testing can increase the trust worthiness of the package
- Distributing your package is needed for adoption of your package by others

Thanks to Stuart Lee for developing the initial content in this slide, which has been subsequently modified a fair amount by me.

Demo R Package `praise.me`

Communicating about your R package

- What is the goal of the package?
- What does your function(s) do?
- How do we use it?
- Why should we use it?
- Where do we find and install it?

Documentation is vital

`praise.me` package

❗ This package is for teaching demo only

- The goal of the `praise.me` package is to give you or someone else a random word of praise.

```r
library(praise.me)
praise_me()
```

```
You are astonishing!
```

```r
praise_me()
```

```
You are delightful!
```

```r
praise_someone("Patrick")
```

```
Patrick is extraordinary!
```

```r
praise_someone("Harriet")
```

```
Harriet is delightful!
```

`praise_me()` function

```r
praise_me <- function() {
  praises <- c(
    "exceptional",
    "remarkable",
    "extraordinary",
    "delightful",
    "wonderful",
    "fantastic",
    "phenomenal",
    "brilliant",
    "astonishing",
    "splendid"
  )
  affirmation <- sample(praises, 1)
  paste0("You are ", affirmation, "!")
}
```

```r
praise_me()
```

```
[1] "You are phenomenal!"
```

`praise_someone()` function

```r
praise_someone <- function(who = NULL) {
  praises <- c(
    "exceptional",
    "remarkable",
    "extraordinary",
    "delightful",
    "wonderful",
    "fantastic",
    "phenomenal",
    "brilliant",
    "astonishing",
    "splendid"
  )
  affirmation <- sample(praises, 1)
  ifelse(is.null(who),
    paste0(tools::toTitleCase(affirmation), "!"),
    paste0(who, " is ", affirmation, "!")
  )
```

```r
praise_someone()
```

```
[1] "Brilliant!"
```

```r
praise_someone("Patrick")
```

```
[1] "Patrick is astonishing!"
```

Reduce repetition

`data-raw/praises.R`

```r
praises <- data.frame(words = c(
  "exceptional",
  "remarkable",
  "extraordinary",
  "delightful",
  "wonderful",
  "fantastic",
  "phenomenal",
  "brilliant",
  "astonishing",
  "splendid"
))

usethis::use_data(praises, overwri
```

`R/praise.R`

```r
#' @export
praise_me <- function() {
  affirmation <- sample(praises$words, 1)
  paste0("You are ", affirmation, "!")
}

#' @export
praise_someone <- function(who = NULL) {
  affirmation <- sample(praises$words, 1)
  ifelse(is.null(who),
    paste0(tools::toTitleCase(affirmation), "!"),
    paste0(who, " is ", affirmation, "!")
  )
}
```

Or put code for `praises` in a file under `R/` if not using as exported data.

Custom `print` method

```r
praise_me()
```

```
[1] "You are extraordinary!"
```

- `print` is an S3 method and above is actually using `print.default()`

```r
print
```

```
function (x, ...)
UseMethod("print")
<bytecode: 0x7f87890af698>
<environment: namespace:base>
```

- Custom `print` method for class `praise`:

```r
#' @export
print.praise <- function(x, ...) {
  cat(x, ...)
}
```

Internal functions to reduce repetition

R/praise.R

```r
#' @export
praise_me <- function() {
  affirmation <- sample(praises$words, 1)
  out <- paste0("You are ", affirmation, "!")
  praise_now(out)
}

#' @export
praise_someone <- function(who = NULL) {
  affirmation <- sample(praises$words, 1)
  out <- ifelse(is.null(who),
    paste0(tools::toTitleCase(affirmation), "!"),
    paste0(who, " is ", affirmation, "!")
  )
  praise_now(out)
}

praise_now <- function(praise) {
```

Comparison with new print

Old

```r
praise_me()
```

```
[1] "You are splendid!"
```

```r
praise_someone()
```

```
[1] "Delightful!"
```

New

```r
praise_me()
```

```
You are excpetional!
```

```r
praise_someone()
```

```
Remarkable!
```

Note: the return object is still a `character` so you can store the object.

```r
x <- cat("Hello")
```

```
Hello
```

```r
x
```

```
NULL
```

```r
x <- praise_now("Hello")
x
```

```
Hello
```

Using pipe operator in your package

- To add the `%>%` operator in your package, you can import the `magrittr` package (used to import pipe operator in all `tidyverse` packages).
- Add all essential elements automatically with:

```
usethis::use_pipe()
```

Documentation

Documenting R functions with `roxygen2`

- use `#'` above a function to write documentation for that function
- `roxygen2` uses `@` tags to structure documentation, e.g.
  - any text after `@description` is the description
  - any text after `@param` describes the arguments of the function
  - `@export` signals that it is an exported function
  - any text after `@return` describes the return object
  - the full list of Rd tags are found here
- `devtools::document()` converts the Rd tags to appropriate sections of `.Rd` files written in the `man/` folder

Documenting praise.me package

R/praise.R

```r
#' Praises you or someone
#'
#' @description
#' Praises you or someone with a random word.
#'
#' @param who A character of who to praise.
#'
#' @return An object of class `praise` and `character`.
#'
#' @examples
#' praise_me()
#' praise_someone()
#' praise_someone("Joanna")
#'
#' @export
praise_me <- function() {
  affirmation <- sample(praises$words, 1)
  out <- paste0("You are ", affirmation, "!")
```

Documenting data

- `usethis::use_data_raw()` to store R code to process raw data,
- `usethis::use_data()` to save a binary file in `data/` directory,
- The data is named `praises`.
- Documentation is contained in `data.R` or `name-of-data.R`

`R/data.R`

```r
#' A list of praises
#'
#'
#' @format A data frame with a single column.
#' \describe{
#'   \item{words}{A list of praises.}
#' }
#' @source \url{https://www.vocabulary.com/lists/5167}
"praises"
```

Make package documentation

- Add documentation of the “big picture” of your package

```
usethis::use_package_doc()
```

- Above creates the file below

R/praise.me-package.R

```
#' @keywords internal
"_PACKAGE"

## usethis namespace: start
## usethis namespace: end
NULL
```

- Default package documentation is built from your DESCRIPTION file

```
library(praise.me)
?praise.me
```

Vignette: a long-form documentation

- Some documentation doesn’t fit as a package or function documentation.
- You may want to built a vignette (article) for these cases.

```r
usethis::use_vignette(name = "my-amazing-package",
                      title = "My amazing package")
```

- Edit the created Rmd file
- Knit the vignette to see what it looks like
- Use `devtools::build()` to build package with vignettes included

Dependencies

Adding dependencies

- Dependencies are specified in DESCRIPTION file under three categories:
  - `Depends`: Specify the version of R that the package will work with or package that it is dependent on (e.g. for ggplot2 extension packages, it depends on ggplot2).
  - `Imports`: External packages that are imported to use in your package. Most external packages are in this category.
  - `Suggests`: Packages that are not strictly needed but are nice to have, i.e. you use them in examples or vignettes.
- You can add easily add this via `usethis::use_package()`

Importing `cowsay`

```r
cowsay::say("Hello", by = "cow")
```

```
 -----
Hello
 ------
    \   ^__^
     \  (oo)\ ________
        (__)\         )\ /\
             ||------w|
             ||      ||
```

```r
usethis::use_package("cowsay", type = "Imports") # default is Imports
```

This adds a line in the `DESCRIPTION` file:

```
Imports:
    cowsay
```

Using imported packages

1. Refer to it with `pkg::fun()`.

```r
#' Praises you or someone
#'
#' @description
#' Praises you or someone with a random word.
#'
#' @param who A character of who to praise.
#' @param by A character to say the praise. See the full
#'   list of character by `list_character()`.
#'
#' @return An object of class `cheer`, which is
#'   just a character with special print method.
#'
#' @examples
#' praise_me()
#' praise_me(by = "cow")
#' praise_someone()
#' praise_someone("Joanna", by = "cat")
#'
```

Using imported packages

2. Use `#' @importFrom pkg fun` to drop the `pkg::`.
3. Use `#' @import pkg` to import *all* functions in `pkg` (not recommended).

```r
#' @importFrom cowsay say
praise_now <- function(praise, by = NULL) {
  if (is.null(by)) {
    out <- praise
  } else {
    out <- say(praise, by = by, type = "string")
  }
  structure(out, class = c("praise", "character"))
}
```

Unit Tests

Testing

- When we check a function works in the console, we are informally testing the function.
- We can formalise and automate this process using unit tests.
- This checks your assumptions - does your code do what you think it does?
- Ensure code works as intended as you develop the package.

Unit testing with `testthat`

- To create a file for testing for the active R file:

```r
usethis::use_test()
```

- This creates a file `test-active-filename.R` in `tests/testthat/` directory

```
praise.me
|- R
|   |- praise.R
|- tests
|   |- testthat
|       |- test-praise.R
|- ...
```

Writing tests with `testthat`

`tests/testthat/test-praise.R`

```r
test_that("praise works", {
  library(stringr)
  expect_true(str_detect(praise_me(), "^You are [a-z]+!$"))
  expect_true(str_detect(praise_someone(), "^[A-Z][a-z]+!$"))
  expect_true(str_detect(
    praise_someone(who = "Emi"),
    "^Emi is [a-z]+!$"
  ))
})
```

Test as you make changes to code:

```r
devtools::test_active_file()
devtools::test() # to test whole package
```

Sharing

Share and collaborate on your package

- Track changes to your code with Git

```r
usethis::use_git()
```

- Collaborate with others via GitHub (or otherwise)

```r
usethis::use_github()
```

or for existing repo, run from the terminal:

```
git remote add origin https://github.com/user/repo.git
```

- You can install your R package now using:

```r
devtools::install_github("user/repo")
```

Installing `praise.me` package

```
devtools::install_github("emitanaka/praise.me")
```

- The package is found at `https://github.com/emitanaka/praise.me`.
- It's a good idea to add a `README` file with installation instructions – this is displayed in the GitHub repo.
- You can create a `README.Rmd` file with

```
usethis::use_readme_rmd()
# OR usethis::use_readme_md() if you have no code
```

- Make sure you knit the `README.Rmd` when you modify its contents.

Package documentation website with pkgdown

- Automatically turns all package documentation into a website.
- Documentation can now be easily viewable outside of R.
- Easy to customise appearance of the site using YAML

Using `pkgdown`

```r
usethis::use_pkgdown()
```

- Build site locally with `pkgdown::build_site()`
- Site appearance is modified in the `_pkgdown.yml` file
  - bootswatch themes for the appearance of the whole site
  - organising function / vignette documentation with reference
- See the vignette for more details
- Automatically build and deploy your site with GitHub actions

```r
usethis::use_pkgdown_github_pages() # if using this, no need for usethis::use_pkgdown()
```

The whole package development workflow

```r
available::available("pkgname") # check if package name is available (if planning to publish
usethis::create_package("pkgname")
usethis::use_git() # set up version control
usethis::use_github() # optional
usethis::use_r("myfile")
# write some functions in a script
usethis::use_data_raw() # if adding data
devtools::load_all() # try it out in the console
usethis::use_package("import-pkgname") # add package to import (or depends or suggests)
usethis::use_package_doc() # add package documentation
usethis::use_pipe() # if you want to add %>% from `magrittr`
usethis::use_vignette("vignette-name") # add vignette
usethis::use_test() # make test file for active R file
# write some test
devtools::test_active_file() # test active file
devtools::test() # test whole package
devtools::build() # build vignettes
devtools::install() # to install package
```

Week 9 Lesson

Summary

- Package documentation is important to let others know about the goal of the package, what your function does, and how to use your package.
- Sharing your package by making it easy to install, implementing unit tests, commiting to good documentation, and making the documentation accessible helps to build trust to use your package.
- You can make package development and distribution easy with `usethis`, `devtools`, `roxygen2`, `testthat` and `pkgdown`.

Resources

- `testthat` reference
- `roxygen2` documentation tags
- Customising your `pkgdown` site