Testing R code

Brendan Clarke, NHS Education for Scotland, brendan.clarke2@nhs.scot

07/08/2024

KIND Learning Network

NHS Education for Scotland

Welcome

- this session is for 🌶🌶 intermediate R users
- we'll get going properly at 15.05
- you'll need R & Rstudio / posit.cloud / Posit Workbench to follow along
- if you can't access the chat, you might need to join our Teams channel: tinyurl.com/kindnetwork
- you can find session materials at tinyurl.com/kindtrp

KIND Learning Network
NHS Education for Scotland

The KIND network

- a social learning space for staff working with **k**nowledge, **i**nformation, and **d**ata across health, social care, and housing in Scotland
- we offer social support, free training, mentoring, community events, …
- Teams channel / mailing list

KIND Learning Network

R training sessions

Session	Date	Area	Level
Hacker Stats (AKA Resampling Methods)	14:00-15:00 Wed 14th August 2024	R	🌶🌶🌶 : **advanced-level**
Flexdashboard	13:00-14:30 Thu 15th August 2024	R	🌶🌶 : **intermediate-level**

KIND Learning Network
NHS Education for Scotland

Session outline

- introduction: why test?
- informal testing
- unit testing
  - introduction - why automate your tests?
  - `testthat` walkthrough

Introduction: why test?

- code goes wrong
  - functions change
  - data sources change
  - usage changes
- testing guards against the commonest problems
- that makes for more reliable code
  - more reliable code opens the way to nicer development patterns

A note

- most discussions about testing come about as part of package development
  - we'll avoid that area here, but please see the three excellent chapters in the R packages book for guidance
  - we'll also steer clear of Shiny/Rmarkdown/Quarto, as things can be a bit more tricky to test there
- we also won't talk about debugging here (although do look out for the future training session on that)

Informal testing

- a real-world example: Teams transcripts
- Teams transcripts can be very useful data-sources
- but they’re absolutely horrible to work with:

```
WEBVTT
Q1::>
00:00:00.000 --> 00:00:14.080
<v Brendan Clarke> this was the first question in the transcript

00:00:14.080 --> 00:00:32.180
<v Someone Else> then someone replied with this answer

Q2::>
00:00:32.180 --> 00:00:48.010
<v Brendan Clarke> then there was another question

00:00:48.010 --> 00:00:58.010
<v Someone Else> and another tedious response
```

KIND Learning Network

NHS Education for Scotland

Informal testing

- imagine that you've written a (horrible) Teams transcript parser:
- how would you test this code to make sure it behaves itself?

```
file <- "data/input.txt"

readLines(file) |>
    as_tibble() |>
    filter(!value == "") |>
    filter(!value == "WEBVTT") |>
    mutate(question = str_extract(value, "^(Q.*?)::>$")) |>
    fill(question, .direction = 'down') |>
    filter(!str_detect(value,  "^(Q.*?)::>$")) |>
    mutate(ind = rep(c(1, 2),length.out = n())) |>
    group_by(ind) |>
    mutate(id = row_number()) |>
    spread(ind, value) |>
    select(-id) |>
    separate("1", c("start_time", "end_time"), " --> ") |>
    separate("2", c("name", "comment"), ">") |>
    mutate(source = str_remove_all(file, "\\.txt"),
```

Informal testing

question	start_time	end_time	name	comment	source
Q1	00:00:00.000	00:00:14.080	Brendan Clarke	this was the first question in the transcript	data/input
Q1	00:00:14.080	00:00:32.180	Someone Else	then someone replied with this answer	data/input
Q2	00:00:32.180	00:00:48.010	Brendan Clarke	then there was another question	data/input
Q2	00:00:48.010	00:00:58.010	Someone Else	and another tedious response	data/input

Informal testing

- we could change the inputs, and look at the outputs
  - so twiddle our input file, and manually check the output
- maybe we could also change the background conditions
  - change the R environment, or package versions, or whatever
- but that gets tedious and erratic very quickly

Unit testing = automated, standardised, testing

- the best place to start is with `testthat`:

```
1 library(testthat)
```

- helpful man pages
- nice vignette
- more ambitious guide to ad hoc testing with `testthat`

First steps with testthat

- built for R package developers
- but readily usable for non-package people

```
1 test_that("multiplication works", {
2   expect_equal(2 * 2, 4)
3 })
```

```
Test passed 🥇
```

KIND Learning Network

Functions and testthat

- testthat works best when you're testing functions
- functions in R are easy:

```
1 function_name <- function(arg1 = default1, arg2 = default2){
2       arg1 * arg2 # using our argument names
3 }
```

or include the body inline for simple functions:

```
1 function_name <- function(arg1 = default1, arg2 = default2) arg1 * arg2
```

Transform your code into functions

```
1 multo <- function(n1, n2){
2   n1 * n2
3 }
```

KIND Learning Network
NHS Education for Scotland

Test your function

- then test. We think that `multo(2,2)` should equal 4, so we use:
  - `test_that()` to set up our test environment
  - `expect_equal()` inside the test environment to check for equality

```
# then run as a test

test_that("multo works with 2 and 2", {
    expect_equal(multo(2, 2), 4)
})
```

```
Test passed 😸
```

Raise your expectations

- we can add more expectations

```
test_that("multo works in general", {
    expect_equal(multo(2, 2), 4)
    expect_identical(multo(2,0.01), 0.02)
    expect_type(multo(0,2), "double")
    expect_length(multo(9,2), 1)
    expect_gte(multo(4,4), 15)
})
```

```
Test passed 🥳
```

Equal and identical

- beware the floating point error

```
1  3 - 2.9
```

```
[1] 0.1
```

```
1  3 - 2.9 == 0.1
```

```
[1] FALSE
```

- happily, there's a sufficiently sloppy way of checking equality:

```
1  test_that("pedants corner", {
2    expect_equal(multo(2, 0.01), 0.020000001)
3    expect_identical(multo(2, 0.01), 0.02)
4  })
```

```
Test passed 🎉
```

KIND Learning Network

NHS Education for Scotland

Beyond single values

- if you want to work with vectors, there are a number of tools for checking their contents:

```
x <- rownames(as.matrix(eurodist, labels=TRUE)) # odd built in dataset

test_that("check my vec", {
    expect_equal(x[1:2], c("Athens", "Barcelona"))
})
```

```
Test passed 😸
```

Beyond single values

- you can get **much** more fancy with a bit of set theory (not really set theory):

```
y <- x

test_that("check my vec sets", {
    expect_success(expect_setequal(x, y)) # all x in y
    expect_failure(expect_mapequal(x, y)) # same names, y is proper subset x) # all x in y)
    show_failure(expect_contains(x[1:19], y)) # y proper subset x)
    expect_success(expect_in(x, y)) # x proper subset y
})
```

```
Failed expectation:
x[1:19] (`actual`) doesn't fully contain all the values in `y` (`expected`).
* Missing from `actual`: "Stockholm", "Vienna"
* Present in `actual`:   "Athens", "Barcelona", "Brussels", "Calais", "Cherbourg", "Cologne", "Copenhagen", "Geneva",
"Gibraltar", ...

Test passed 😀
```

Beyond single values

```
1 y <- sample(x, length(x)-2)
2
3 test_that("check my vec sets", {
4     expect_failure(expect_setequal(x, y)) # all x in y
5     expect_failure(expect_mapequal(x, y)) # same names, y is proper subset x) # all x in y)
6     expect_success(expect_contains(x, y)) # y proper subset x)
7     expect_failure(expect_in(x, y)) # x is a proper subset y
8 })
```

```
Test passed
```

KIND Learning Network

NHS Education for Scotland

Testing tibbles

- because most of the tests are powered by `waldo`, you shouldn't have to do anything fancy to test on tibbles:

```
library(palmerpenguins)

my_pengs <- penguins

test_that("penguin experiments", {
    expect_equal(my_pengs, penguins)
})
```

```
Test passed 😅
```

Types and classes etc

- one *massive* corollary to that: if you don't do a lot of base-R, expect a fiercely stringent of your understanding of types and classes.

```
typeof(penguins)
```

```
[1] "list"
```

```
class(penguins)
```

```
[1] "tbl_df"     "tbl"        "data.frame"
```

```
is.object(names(penguins)) # vectors are base types
```

```
[1] FALSE
```

```
attr(names(penguins), "class") # base types have no class
```

```
NULL
```

```
is.object(penguins) # this is some kind of object
```

```
[1] TRUE
```

```
attr(penguins, "class") # so it definitely does have a class
```

```
[1] "tbl_df"     "tbl"        "data.frame"
```

Testing tibbles

```
test_that("penguin types", {
    expect_type(penguins, "list")
    expect_s3_class(penguins, "tbl_df")
    expect_s3_class(penguins, "tbl")
    expect_s3_class(penguins, "data.frame")
    expect_type(penguins$island, "integer")
    expect_s3_class(penguins$island, "factor")
})
```

```
Test passed 😸
```

- there's also an `expect_s4_class` for those with that high clear mercury sound ringing in their ears

Last tip

- you can put bare expectations in pipes if you're looking for something specific

```
penguins |>
  expect_type("list") |>
  pull(island) |>
  expect_length(344)
```

Feedback and resources

- please can I ask for some feedback - takes less than a minute, completely anonymous, helps people like you find the right training for them

Session	Date	Area	Level
Hacker Stats (AKA Resampling Methods)	14:00-15:00 Wed 14th August 2024	R	🌶🌶🌶 : **advanced-level**
Flexdashboard	13:00-14:30 Thu 15th August 2024	R	🌶🌶 : **intermediate-level**

KIND Learning Network
NHS Education for Scotland