Lab 7: Unit tests with testthat

Stat 159, Fall 2016, Prof. Sanchez

October 10, 2016

Learning Objectives

- Introduction to the R package "testthat"
- Write simple functions and their unit tests
- Test your code

R package `"testthat"`

`"testthat"` is one of the packages in R that helps you write tests for your functions. One of the main references is the paper *testthat: Get Started with Testing* by Hadley Wickham (see link below). This paper clearly describes the philisoply and workflow of `"testthat"`. But keep in mind that since the introduction of the package, many more functions haven been added to it.

https://journal.r-project.org/archive/2011-1/RJournal_2011-1_Wickham.pdf

About `"testthat"`

- `"testthat"` provides a testing framework for R that is easy to learn and use
- `"testthat"` has a hierarchical structure made up of:
  - expectations
  - tests
  - contexts
- A **context** involves **tests** formed by groups of **expectations**
- Each structure has associated functions:
  - `expect_that()` for expectations
  - `test_that()` for groups of tests
  - `context()` for contexts

List of common expectation functions

Function	Description
`expect_true(x)`	expects that `x` is `TRUE`
`expect_false(x)`	expects that `x` is `FALSE`
`expect_null(x)`	expects that `x` is `NULL`
`expect_type(x)`	expects that `x` is of type `y`
`expect_is(x, y)`	expects that `x` is of class `y`
`expect_length(x, y)`	expects that `x` is of length `y`
`expect_equal(x, y)`	expects that `x` is equal to `y`
`expect_equivalent(x, y)`	expects that `x` is equivalent to `y`
`expect_identical(x, y)`	expects that `x` is identical to `y`
`expect_lt(x, y)`	expects that `x` is less than `y`
`expect_gt(x, y)`	expects that `x` is greater than `y`
`expect_lte(x, y)`	expects that `x` is less than or equal to `y`

Function	Description
`expect_gte(x, y)`	expects that `x` is greater than or equal `y`
`expect_named(x)`	expects that `x` has names `y`
`expect_matches(x, y)`	expects that `x` matches `y` (regex)
`expect_message(x, y)`	expects that `x` gives message `y`
`expect_warning(x, y)`	expects that `x` gives warning `y`
`expect_error(x, y)`	expects that `x` throws error `y`

Practice

- To start the practice, create a new directory, e.g. `test-that`
- `cd` into the directory `test-that/`
- Create two subdirectories: `functions` and `tests`
- Sample content is in the folder `test-that/` of this lab

Functions

Let's start with a couple of basic functions: `range_value()` and `missing_values()`:

```
range_value <- function(x) {
  max(x) - min(x)
}
```

- *description:* computes the range of a numeric vector (i.e. max - min)
- *input:* a numeric vector
- *output:* the range value (max - min)

```
missing_values <- function(x) {
  sum(is.na(x))
}
```

- *description:* computes the number of missing values
- *input:* a numeric vector
- *output:* the number of missing values

Write these functions in files `range-value.R` and `missing-values.R` inside `functions/`

Tests

In the folder `tests/`, create a file `tests.R` that includes tests for `range_value()` and `missing_values()`.

To write the unit tests, we are going to consider the following testing vectors:

- `x <- c(1, 2, 3, 4, 5)`
- `y <- c(1, 2, 3, 4, NA)`
- `z <- c(TRUE, FALSE, TRUE)`
- `w <- letters[1:5]`

The typical structure of the tests has the following form:

```
# load the source code of the functions to be tested
source("../functions/functions.R")

# context with one test that groups expectations
context("Test for range value")

test_that("range works as expected" {
  x <- c(1, 2, 3, 4, 5)

  expect_equal(range_value(x), 4)
  expect_length(range_value(x), 1)
  expect_type(range_value(x), 'double')
})
```

- use `context()` to describe what the test are about
- use `test_that()` to group expectations:
  - output equal to 4
  - output of length one
  - output of type `double`
- to run the tests from the R console, use the function `test_file()`

```
# assuming that your working directory is "test-that/"
# from the R console
library(testthat)
test_file("tests/tests.R")
```

Your Turn

Write more tests—`test_that()`—to test `range_value()` with the rest of the testing vectors `y`, `z`, `w`:

- Using `y`, write expectations for:
  - output of length one
  - output is `NA_real_`
- Using `z`, write expectations for:
  - output of length one
  - output of type `integer`
  - output equal to `1L`
- Using `w`, write expectations for:
  - throws an error

Missing Values

Now consider the function `missing_value()`. Write a context for testing the following expectations:

- output of length one
- output of type `double`
- output greater than or equal to zero

Improving `range_value()`

Modify the function `range_value()` to include an argument `na.rm` that takes a logical value indicating whether missing values should be removed before computing the range.

Adapt the tests for the improved `range_value()` function, especially when using the testing vector `y <- c(1, 2, 3, 4, NA)`.

Extra Challenges

Try writing the following functions and come up with unit tests:

- `center_measures()`
  - *description:* computes measures of center such as Median and Mean
  - *input:* a numeric vector
  - *output:* a numeric vector with median and mean
- `spread_measures()`
  - *description:* computes measures of spread such as Range, IQR, Std Dev
  - *input:* a numeric vector
  - *output:* a numeric vector with range, iqr, and stdev
- `descriptive_stats()`
  - *description:* computes descriptive statistics
  - *input:* a numeric vector
  - *output:* a numeric vector with median, mean, range, iqr, stdev, and number of missing values.
