Testing with testit

## Learning Objectives

* Write assertion statements with assert
* Confirm errors using has\_error
* Confirm warnings using has\_warning
* Use unit tests to confirm code is working as expected

Using assertion statements is a good first step to writing more reliable code. Going the next step, we can pass inputs to a function and confirm that the result is what we expect. Tests that check a function works properly are called unit tests (because each function is a unit of the overall code we are writing). Writing tests gives us confidence that our code works in different situations, serves as explicit documentation of how a function is supposed to work, and alerts us to any changes due to software updates.

In this lesson we will use the simple testing framework in the package [testit](https://cran.r-project.org/web/packages/testit/index.html). There are other more elaborate testing frameworks such as [RUnit](https://cran.r-project.org/web/packages/RUnit/index.html) and [testthat](https://cran.r-project.org/web/packages/testthat/index.html) if you need more complicated testing in the future. Also, as a caveat, R testing frameworks work best in the context of an R package. They will be less flexible in our context testing functions that are not part of an R package.

### Using the testit package

Let's first load the package.

library("testit")

In the last lesson, we wrote assertion statements using stopifnot. However, the error messages generated by stopifnot are cryptic unless we are familiar with the intricacies of a specific function. This can be difficult when writing lots of code or returning to code written long ago. The main function of the testit package is assert, which allows us to include a message that is printed if the assertion fails. This makes it easier to interpret what went wrong.

assert("one equals one", 1 == 1)  
assert("two plus two equals five", 2 + 2 == 5)

assertion failed: two plus two equals five

Error: 2 + 2 == 5 is not TRUE

And similar to stopifnot, we can list multiple statements to check.

assert("these statements are TRUE", 1 == 1, 2 == 2, 3 == 3)

And if any of them fails, it throws an error.

assert("these statements are TRUE", 0 == 1, 2 == 2, 3 == 3)

assertion failed: these statements are TRUE

Error: 0 == 1 is not TRUE

Furthermore, the package has functions to that detect if an error or warning was produced. From their documentation:

# No warning  
has\_warning(1 + 1)

[1] FALSE

# Issues warning because vectors have different lengths  
has\_warning(1:2 + 1:3)

[1] TRUE

# No error  
has\_error(2 - 3)

[1] FALSE

# Throws error because cannot add numeric and character vectors  
has\_error(1 + "a")

[1] TRUE

We can combine these with assert to confirm that our functions throw errors or warnings when given certain inputs.

assert("Throws error", has\_error(1 + "a"))

### Writing unit tests for a function

In the challenge for the last lesson we added assertion statements using stopifnot to the function calc\_sum\_stat. Also we fixed the problem with NAs by adding the argument na.rm = TRUE to mean. The result should have looked something like the following:

calc\_sum\_stat <- function(df, cols) {  
 stopifnot(dim(df) > 0,  
 is.character(cols),  
 cols %in% colnames(df))  
 if (length(cols) == 1) {  
 warning("Only one column specified. Calculating the mean will not change anything.")  
 }  
 df\_sub <- df[, cols, drop = FALSE]  
 stopifnot(is.data.frame(df\_sub))  
 sum\_stat <- apply(df\_sub, 1, mean, na.rm = TRUE)  
 stopifnot(!is.na(sum\_stat))  
 return(sum\_stat)  
}

We checked that our function worked properly by testing some different inputs. Let's convert these into formal units tests that we can automatically run to test our function.

When passing an empty data frame, we expect it to throw an error.

# Empty data frame  
sum\_stat <- calc\_sum\_stat(data.frame(), c("wosCountThru2010", "f1000Factor"))

Error: dim(df) > 0 are not all TRUE

Which we can convert to a test:

assert("Empty data frame throws error",  
 has\_error(calc\_sum\_stat(data.frame(), c("wosCountThru2010",  
 "f1000Factor"))))

We also expect an error when passing a non-character vector for the argument cols.

# Non-character cols  
sum\_stat <- calc\_sum\_stat(counts\_raw, 1:3)

Error: is.character(cols) is not TRUE

And the test looks like:

assert("Non-character vector input for columns throws error",  
 has\_error(calc\_sum\_stat(counts\_raw, 1:3)))

Testing non-existent column names:

# Bad column names  
sum\_stat <- calc\_sum\_stat(counts\_raw, c("a", "b"))

Error: cols %in% colnames(df) are not all TRUE

Converted to a test:

assert("Column names not in data frame throws error",  
 has\_error(calc\_sum\_stat(counts\_raw, c("a", "b"))))

Issue a warning if only one column is given to the function.

# Issue warning since only one column  
sum\_stat <- calc\_sum\_stat(counts\_raw, "mendeleyReadersCount")

Warning in calc\_sum\_stat(counts\_raw, "mendeleyReadersCount"): Only one  
column specified. Calculating the mean will not change anything.

And the test:

assert("Selecting only one column issues warning",  
 has\_warning(calc\_sum\_stat(counts\_raw, "mendeleyReadersCount")))

Lastly, we passed a column that contains NAs. We fixed this so that mean ignores NAs and returns a numeric answer.

# NA output  
sum\_stat <- calc\_sum\_stat(counts\_raw, c("wosCountThru2010", "facebookLikeCount"))  
anyNA(sum\_stat)

[1] FALSE

We can also test this:

assert("NA input does not result in NA output",  
 !anyNA(calc\_sum\_stat(counts\_raw, c("wosCountThru2010",  
 "facebookLikeCount"))))

We have now built an entire test suite that we can automatically run whenever we update the function or update the version of R. This way we always know that the function still works as we initially planned.

assert("Empty data frame throws error",  
 has\_error(calc\_sum\_stat(data.frame(), c("wosCountThru2010",  
 "f1000Factor"))))  
assert("Non-character vector input for columns throws error",  
 has\_error(calc\_sum\_stat(counts\_raw, 1:3)))  
assert("Column names not in data frame throws error",  
 has\_error(calc\_sum\_stat(counts\_raw, c("a", "b"))))  
assert("Selecting only one column issues warning",  
 has\_warning(calc\_sum\_stat(counts\_raw, "mendeleyReadersCount")))  
assert("NA input does not result in NA output",  
 !anyNA(calc\_sum\_stat(counts\_raw, c("wosCountThru2010",  
 "facebookLikeCount"))))

### Challenge

## Write some tests

Write unit tests for the function my\_mean that you wrote in an earlier lesson. It should look something like this:

my\_mean <- function(x) {  
 result <- sum(x) / length(x)  
 return(result)  
}

The input x is a numeric vector, and the output is the mean of the vector of numbers. Some ideas to get started:

* Pass a vector where you know what the mean is, and assert that the result is correct.
* Add some assertion statments to check the input x. Use has\_error to test that the function throws an error when given bad input.
* Issue a warning if the user passes a vector of length one. Test that the warning is properly issued using has\_warning.
* Include an NA in the vector where you know the result to see what happens. Do you need to modify the code to pass the test?

my\_mean <- function(x) {  
 assert("x is numeric", is.numeric(x))  
 if (length(x) == 1) {  
 warning("Input vector had only one element")  
 }  
 result <- sum(x, na.rm = TRUE) / length(x[!is.na(x)])  
 return(result)  
}  
  
assert("Mean is calculated correctly",  
 my\_mean(1:3) == 2, my\_mean(c(2, 4, 6)) == 4)  
assert("Non-numeric input throws error",  
 has\_error(my\_mean("hello")))

assertion failed: x is numeric

assert("Input vector with one element issues warning",  
 has\_warning(my\_mean(5)))  
assert("Mean is calculated correctly when given NAs",  
 my\_mean(c(2, 4, 6, NA)) == 4)