

# Testing and debugging

## Tools for Reproducible Research

Karl Broman

Biostatistics & Medical Informatics, UW–Madison

`biostat.wisc.edu/~kbroman`

`github.com/kbroman`

`@kwbroman`

Course web: [bit.ly/tools4rr](https://bit.ly/tools4rr)

"I tried it, and it worked."

"It's not that we don't test our code, it's that we don't store our tests so they can be re-run automatically."

– Hadley Wickham

R Journal 3(1):5–10, 2011

# Types of tests

- ▶ Unit tests
  - For each small function: does it give the right results in specific cases?
- ▶ Integration tests
  - Check that larger multi-function tasks are working.
- ▶ Regression tests
  - Compare output to saved results, to check that things that worked continue working.

# Types of tests

- ▶ Check inputs
  - Stop if the inputs aren't as expected.
- ▶ Unit tests
  - For each small function: does it give the right results in specific cases?
- ▶ Integration tests
  - Check that larger multi-function tasks are working.
- ▶ Regression tests
  - Compare output to saved results, to check that things that worked continue working.

# Check inputs

```
winsorize <-  
function(x, q=0.006)  
{  
  if(!is.numeric(x)) stop("x should be numeric")  
  
  if(!is.numeric(q)) stop("q should be numeric")  
  if(length(q) > 1) {  
    q <- q[1]  
    warning("length(q) > 1; using q[1]")  
  }  
  if(q < 0 || q > 1) stop("q should be in [0,1]")  
  
  lohi <- quantile(x, c(q, 1-q), na.rm=TRUE)  
  if(diff(lohi) < 0) lohi <- rev(lohi)  
  
  x[!is.na(x) & x < lohi[1]] <- lohi[1]  
  x[!is.na(x) & x > lohi[2]] <- lohi[2]  
  x  
}
```

# Check inputs

```
winsorize <-  
function(x, q=0.006)  
{  
  stopifnot(is.numeric(x))  
  stopifnot(is.numeric(q), length(q)==1, q>=0, q<=1)  
  
  lohi <- quantile(x, c(q, 1-q), na.rm=TRUE)  
  if(diff(lohi) < 0) lohi <- rev(lohi)  
  
  x[!is.na(x) & x < lohi[1]] <- lohi[1]  
  x[!is.na(x) & x > lohi[2]] <- lohi[2]  
  x  
}
```

# assertthat package

```
#' import assertthat
winsorize <-
function(x, q=0.006)
{
  if(all(is.na(x)) || is.null(x)) return(x)

  assert_that(is.numeric(x))
  assert_that(is.number(q), q>=0, q<=1)

  lohi <- quantile(x, c(q, 1-q), na.rm=TRUE)
  if(diff(lohi) < 0) lohi <- rev(lohi)

  x[!is.na(x) & x < lohi[1]] <- lohi[1]
  x[!is.na(x) & x > lohi[2]] <- lohi[2]
  x
}
```



# Tests in R packages

- ▶ Examples in .Rd files
- ▶ Vignettes
- ▶ tests/ directory
  - `some_test.R` and `some_test.Rout.save`

R CMD check is your friend.

# An example example

```
#' @examples
#' x <- sample(c(1:10, rep(NA, 10), 21:30))
#' winsorize(x, 0.2)
```

# A tests/ example

```
library(qtl)

# read data
csv <- read.cross("csv", "", "listeria.csv")

# write
write.cross(csv, "csv", filestem="junk")

# read back in
csv2 <- read.cross("csv", "", "junk.csv",
                   genotypes=c("AA", "AB", "BB",
                               "not BB", "not AA"))

# check for a change
comparecrosses(csv, csv2)

unlink("junk.csv")
```

# testthat package

- ▶ Expectations

```
expect_equal(10, 10 + 1e-7)
expect_identical(10, 10)
expect_equivalent(c("one"=1), 1)
expect_warning(log(-1))
expect_error(1 + "a")
```

- ▶ Tests

```
test_that("winsorize small vectors", { ... })
```

- ▶ Contexts

```
context("Group of related tests")
```

- ▶ Store tests in inst/tests/

- ▶ tests/run-all.R file containing

```
library(testthat)
library(mypkg)
test_package("mypkg")
```

# Example testthat test

```
context("winsorise")

test_that("winsorize works for small vectors", {

  x <-      c(2, 3, 7, 9, 6, NA, 5, 8, NA, 0, 4, 1, 10)
  result1 <- c(2, 3, 7, 9, 6, NA, 5, 8, NA, 1, 4, 1, 7)
  result2 <- c(2, 3, 7, 8, 6, NA, 5, 8, NA, 2, 4, 2, 8)

  expect_identical(winsorize(x, 0.1), result1)
  expect_identical(winsorize(x, 0.2), result2)

})
```

# Example testthat test

```
test_that("winsorize works for a long vector", {  
  
  set.seed(94745689)  
  n <- 1000  
  nmis <- 10  
  p <- 0.05  
  input <- rnorm(n)  
  input[sample(1:n, nmis)] <- NA  
  quL <- quantile(input, p, na.rm=TRUE)  
  quH <- quantile(input, 1-p, na.rm=TRUE)  
  
  result <- winsorize(input, p)  
  middle <- !is.na(input) & input >= quL & input <= quH  
  low <- !is.na(input) & input <= quL  
  high <- !is.na(input) & input >= quH  
  
  expect_identical(is.na(input), is.na(result))  
  expect_identical(input[middle], result[middle])  
  expect_true( all(result[low] == quL) )  
  expect_true( all(result[high] == quH) )  
  
})
```

# Workflow

- ▶ Write tests as you're coding.
- ▶ Run `test()`
  - with `devtools`, and working in your package directory
- ▶ Consider `auto_test("R", "inst/tests")`
  - automatically runs tests when any file changes

# What to test?

- ▶ You can't test **everything**.
- ▶ Focus on the **boundaries**
  - Depends on the nature of the problem
  - Vector of length 0 or 1
  - Things exactly matching; things with no matches.
- ▶ Test handling of missing data.
  - NA, Inf, -Inf
- ▶ Automate the construction of test cases, or at least create a table of inputs and expected outputs and then run through the values in the table.



# Debugging tools

- ▶ `cat`, `print`
- ▶ `traceback`, `browser`, `debug`
- ▶ RStudio breakpoints
- ▶ Eclipse/StatET
- ▶ `gdb`

# Debugging

Step 1: Reproduce the problem

# Debugging

Step 1: Reproduce the problem

Step 2: Turn it into a test

# Debugging

Isolate the problem: where do things go bad?

# Debugging

Don't make the same mistake twice.

# The most pernicious bugs

The code is right, but your thinking is wrong.

# The most pernicious bugs

The code is right, but your thinking is wrong.

You were mistaken about what the code would do.

# The most pernicious bugs

The code is right, but your thinking is wrong.

You were mistaken about what the code would do.

→ Write trivial programs to test your understanding.



# Summary

- ▶ If you don't test your code, how do you know it works?
- ▶ If you test your code, save and automate those tests.
- ▶ Check the input to each function.
- ▶ Write unit tests for each function.
- ▶ Write some larger regression tests.
- ▶ Turn bugs into tests.