

# tests/

*December 2016*

Hadley Wickham  
[@hadleywickham](#)  
Chief Scientist, RStudio

# Motivation

# Your turn

You already know how to debug. Recall four helpful functions/features.

# Four useful debugging tools

# Where is the problem?

`traceback()`

# Give me an interactive prompt here

`browser()`

# Give me an interactive prompt on error

`options(error = recover)`

# Convert warnings to errors

`options(warn = 2)`

# Your turn

Why should you write tests?

# Why do we test?

## Correctness

**Ensure current correctness**

Interactive feedback

Reduce maintenance burden

## Working with others

Guard against future developers

Communicate with colleagues

Guard against complex interactions

Social pressure

Establish interfaces

## Design

Think adversarially

Enable refactoring

Force good design

# Other benefits

Code that can be tested easily, often has a better, more modular, design

When you stop working, leave a test failing. You'll know what to work on when you come back

Make big changes without fear of accidentally breaking anything

# testthat

Provides easy transition from informal to formal tests.  
Can be used in wide variety of situations.

Wide range of expectations/assertions.

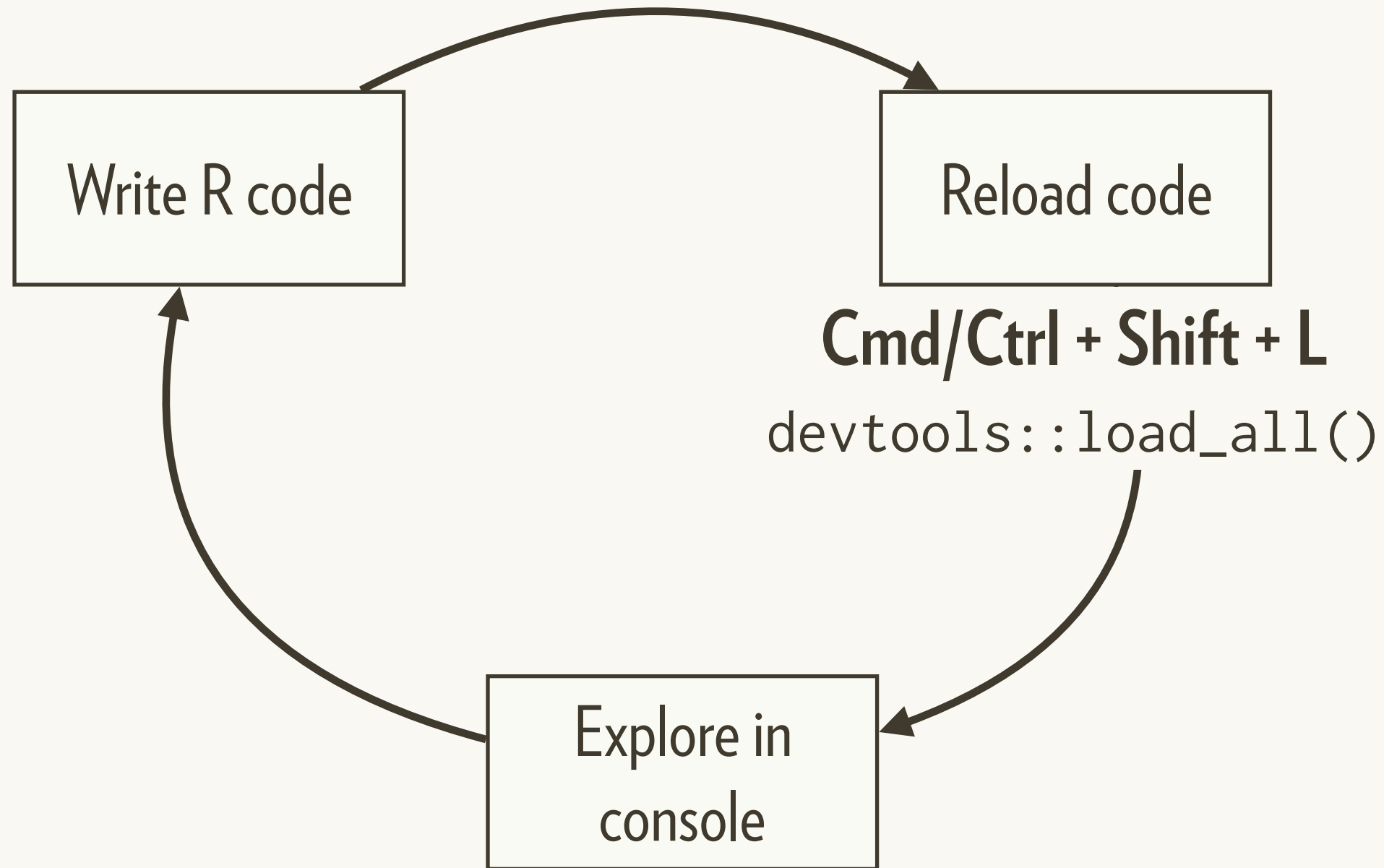
Fun output designed to keep you motivated.

Currently used by over 1395 packages. Look at `tidyr`,  
`devtools`, `roxygen2`, `ffbase`, `ISOweek` for good  
examples.

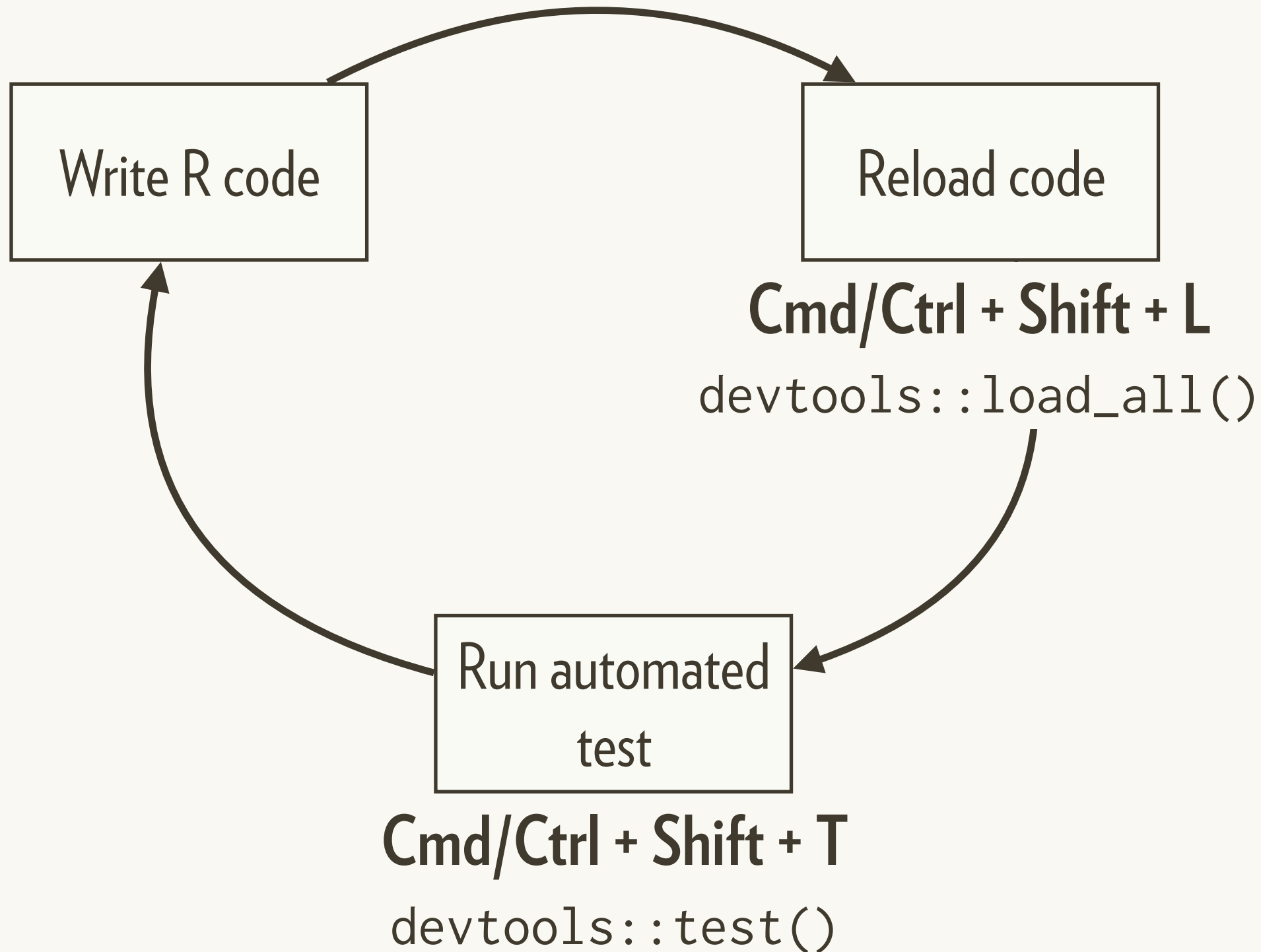
<http://r-pkgs.had.co.nz/tests.html>



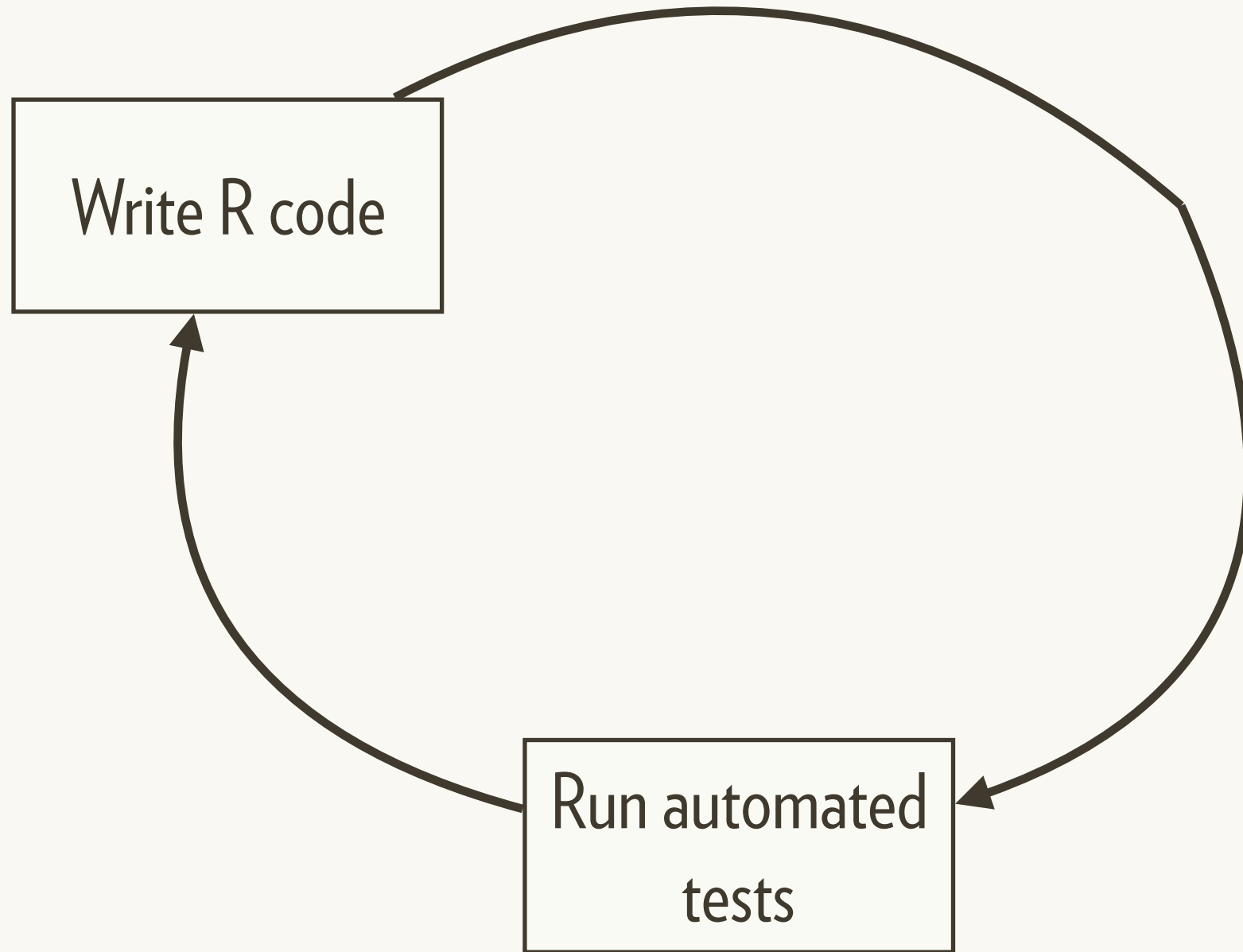
# Workflow so far



# Workflow so far



# But why load the code?



**Cmd/Ctrl + Shift + T**  
`devtools::test()`

Change working directory/project to:

[test-me]

# Your turn

Where are the tests? Which tests are failing? Which function is probably the cause? Can you fix it?

Don't skip me until you've tried to fix the test!

```
# Tests located in tests/testthat
# All failing tests in tests/testthat/test-variance.R
# So variance function probably to blame
VAR <- function(x) E((x - E(x) ^ 2))

# Should be
VAR <- function(x) E((x - E(x)) ^ 2)
```

# Organisation



Test

Expectation  
Expectation  
Expectation  
Expectation

Test

Expectation  
Expectation  
Expectation

Test

Expectation  
Expectation  
Expectation  
Expectation  
Expectation  
Expectation

Test

Expectation  
Expectation  
Expectation  
Expectation

```
# Expectations can test many things. Two common expectations  
# are expect_equal and expect_error
```

```
test_that("probabilities must be positive and add to 1", {  
  expect_error(rv(1, 0.9), "sum to 1")  
  expect_error(rv(1, -0.1), "positive")  
})
```

```
test_that("duplicate values have probabilities collapsed", {  
  expect_equal(rv(c(1, 1)), rv(1))  
  expect_equal(rv(c(1, 1, 2), c(0.25, 0.25, 0.5)), rv(1:2))  
})
```

# Your turn

Add the expectations on the following slide. What file should they go in? Modify the code to make the tests pass.

```
test_that("expectation throws error if input not an rv", {  
  expect_error(E("a"), "must be an rv object")  
})
```

```
test_that("expectation of a number is that number", {  
  expect_equal(E(5), 5)  
})
```

```
test_that("variance of a constant is 0", {  
  expect_equal(VAR(10), 0)  
})
```

```
test_that("variance throws error if input not an rv", {  
  expect_error(VAR("a"), "must be an rv object")  
})
```

Abbreviation	Test
<code>expect_equal</code>	Uses <code>all.equal()</code> , ignores floating point differences
<code>expect_identical</code>	Uses <code>identical()</code> for stricter numerical testing.
<code>expect_equivalent</code>	Like <code>expect_equal()</code> , but also ignores differences in attributes.
<code>expect_s3_class</code> <code>expect_s4_class</code>	Check that inherits from a given class.
<code>expect_true</code> / <code>expect_false</code>	Catch all expectations for anything not otherwise covered

Abbreviation	Test
<code>expect_matches</code>	Does any value match the supplied regular expression?
<code>expect_output</code>	Does printed output match the supplied regular expression?
<code>expect_message</code>	Does displayed messages match the supplied regular expression?
<code>expect_warning</code>	Do any warnings match supplied regular expression?
<code>expect_error</code>	Do any errors match supplied regular expression?

# How to create tests?

1. Compare with known outputs.
2. Compare with results calculated another way.
3. Whenever you find a bug, first figure out the right answer and write a test.
4. Test areas that are likely to fail (i.e. complicated bits).
5. Focus on improving tests over time, not being perfect when you first start.
6. (Exciting work going on at <https://github.com/jimhester/covr>)

# Your turn

How might you test the probability function?  
Brainstorm with 2 minutes with your neighbours.



# Your turn

Create a new file (and new context) for testing `P()`. Do you discover any new bugs?

Read `?devtools::use_test()`

```
context("Probability")
```

```
test_that("0 probability of being infinite", {  
  X <- rv(1:10)  
  expect_equal(P(X > -Inf), 1)  
  expect_equal(P(X < -Inf), 0)  
  expect_equal(P(X > Inf), 0)  
  expect_equal(P(X < Inf), 1)  
})
```

```
test_that("missing comparison means 100% of NA", {  
  X <- rv(1:5)  
  expect_equal(P(X > NA), NA_real_)  
  expect_true(is.na(P(X > NA)))  
})
```

# Other challenges

How could you test that (e.g) `dice + dice` gives the correct result? How would you test that `lm()` gives the correct result?

How would you test `rsim()`?

How would you test `plot.rv()`?

How would you test connecting to a website?

# Other challenges

How could you test that (e.g) `dice + dice` gives the correct result? How would you test that `lm()` gives the correct result? (`expect_equal_to_reference`)

How would you test `rsim()`?  
(use pure functions as much possible)

How would you test `plot.rv()`?  
(see `vdiffr`)

How would you test connecting to a website?  
(`try_again`)

Putting it all  
together

# Challenge: make this into a package

```
col_summary <- function(df, fun) {  
  stopifnot(is.data.frame(df))  
  
  df_num <- purrr::keep(df, purrr::is_numeric)  
  as.data.frame(purrr::map(df, fun))  
}
```

1. Brainstorm a name for this package.
2. Create a new package.
3. Add `col_summary()` & verify it works.
4. Add `purrr` as a dependency.
5. Document `col_summary()`.
6. Convert your informal tests to formal tests.

# Remember these functions:

Back at 1530

```
devtools::create()
```

```
devtools::use_package()
```

```
devtools::use_testthat()
```

```
devtools::use_test()
```

```
# Some common problems are listed on the next slide
```



**Problem:** Accidentally pressed source

**Symptoms:** load\_all() stops working

**Diagnosis:** Check your environment pane

**Cure:** Restart R



This work is licensed under the  
Creative Commons Attribution-Noncommercial 3.0  
United States License.

To view a copy of this license, visit  
<http://creativecommons.org/licenses/by-nc/3.0/us/>