L12 Pipes and Functions

Data Science I (STAT 301-1)

Shay Lebovitz

Contents

Overview

The goal of this lab is to improve your programming skills and develop an appreciation for writing code. Remember that code is a communication tool, and that being able to write clear and concise code is extremely important.

Datasets

There are no datasets for this lab.

Exercises

Please complete the following exercises. Be sure that your solutions are clearly indicated and that the document is neatly formatted.

Load Packages It shouldn't be necessary to load all the `tidyverse` packages, but you may want to anyway since you'll likely want to build functions that can be piped and that play nicely with the `tidyverse`. The only required tidyverse package is `dplyr`, which contains the `case_when()` function, but you could opt to use `dplyr::case_when()` to avoid loading the package.

```
library(lubridate)
library(dplyr)
```

Exercise 1 (Website: 19.2.1 Ex. 2)

In the second variant of `rescale01()`, infinite values are left unchanged. Rewrite `rescale01()` so that `-Inf` is mapped to 0 and `Inf` is mapped to 1.

```
rescale01 <- function(x) {
  rng <- range(x, na.rm = TRUE, finite = TRUE)
  val <- (x - rng[1]) / (rng[2] - rng[1])
  val[val == -Inf] <- 0
  val[val == Inf] <- 1
```

```
  val
}
x <- c(1:10, Inf, -Inf)
rescale01(x)
```

```
##  [1] 0.0000000 0.1111111 0.2222222 0.3333333 0.4444444 0.5555556 0.6666667
##  [8] 0.7777778 0.8888889 1.0000000 1.0000000 0.0000000
```

Exercise 2 (Website: 19.2.1 Ex. 3)

Practice turning the following code snippets into functions. Think about what each function does. What would you call it? How many arguments does it need? Can you rewrite it to be more expressive or less duplicative?

```
mean(is.na(x))

x / sum(x, na.rm = TRUE)

sd(x, na.rm = TRUE) / mean(x, na.rm = TRUE)
```

```
prop_na <- function(x) {
  mean(is.na(x))
}
prop_na(c(1, 2, NA, 3, 4, NA, NA, 5))
```

```
## [1] 0.375
```

Because `is.na(x)` is a boolean operator and thus returns either 0 or 1, this code calculates the proportion of values of `x` that are `NA`.

```
standardize <- function(x) {
  x / sum(x, na.rm = TRUE)
}
standardize(c(1, 2, 3, 4, NA))
```

```
## [1] 0.1 0.2 0.3 0.4  NA
```

This code snippet computes the proportion of each value in `x` to the total value of `x`, disregarding `NA`. Essentially, it standardizes the vector so that it's sum is 1.

```
coef_var <- function(x) {
  sd(x, na.rm = TRUE) / mean(x, na.rm = TRUE)
}
coef_var(c(3, 4, 5, 6, 7, NA))
```

```
## [1] 0.3162278
```

This code snippet computes the coefficient of variation, which is the standard deviation over the mean of a vector.

Exercise 3 (Website: 19.2.1 Ex. 5)

Write `both_na()`, a function that takes two vectors of the same length and returns the number of positions that have an `NA` in both vectors.

```
both_na <- function(x, y) {
  sum(is.na(x) & is.na(y))
```

```
}

both_na(c(1, NA, 2, NA, 3, 4, NA), c(NA, NA, 4, 5, NA, NA, 6))
```

```
## [1] 1
```

Exercise 4 (Website: 19.3.1 Ex. 1)

Read the source code for each of the following three functions. Describe what each function does, and then brainstorm better names for them.

```
is_prefix <- function(string, prefix) {
  substr(string, 1, nchar(prefix)) == prefix
}
is_prefix(c('rescale', 'descale'), 're')
```

```
## [1]  TRUE FALSE
```

Tests whether the string starts with the inputted prefix

```
remove_last <- function(x) {
  if (length(x) <= 1) return(NULL)
  x[-length(x)]
}
remove_last(c(1,2,3,4,5))
```

```
## [1] 1 2 3 4
```

```
remove_last(1)
```

```
## NULL
```

removes the last element of `x`. if `x` is length 1, return `NULL`

```
length_rep <- function(x, y) {
  rep(y, length.out = length(x))
}
length_rep(c(2,2,2), 10)
```

```
## [1] 10 10 10
```

repeats `y` the length of `x` times.

Exercise 5 (Website: 19.3.1 Ex. 4)

Make a case for why `norm_r()`, `norm_d()` (etc.) might be better names than `rnorm()`, `dnorm()`. Then make a case for the opposite. **`norm_r()` and `norm_d()` place the emphasis on the distribution, rather than the function. `rnorm()` and `dnorm()` do the opposite, they place the emphasis on the function. I guess which one you use depends on the context, I wouldn't say one is better or clearer than the other.**

Exercise 6 (Website: 19.4.4 Ex. 2)

Write a greeting function that says "good morning", "good afternoon", or "good evening", depending on the time of day. (Hint: Try using a `time` argument that defaults to `lubridate::now()`. That will make it easier to test your function.)

```r
greeting <- function(time = now()) {
  if (hour(time) < 12) {
    'good morning'
  }
  else if (hour(time) < 17) {
    'good afternoon'
  }
  else {
    'good evening'
  }
}

greeting()
```

```
## [1] "good evening"
```

Exercise 7 (Website: 19.4.4 Ex. 3)

Implement a `fizzbuzz` function. It should take a single number as input. If the number is divisible by three, the function should return "fizz". If the number is divisible by five, it should return "buzz". If the number is divisible by three and five, it returns "fizzbuzz". Otherwise, it returns the number. **Hint: Don't try and write the function all at once, and consider using `case_when()`.**

```r
fizzbuzz <- function(x) {
  if (x %% 3 == 0 & x %% 5 == 0) {
    'fizzbuzz'
  }
  else if (x %% 3 == 0) {
    'fizz'
  }
  else if (x %% 5 == 0) {
    'buzz'
  }
  else {
    x
  }
}
fizzbuzz(10)
```

```
## [1] "buzz"
```

```r
fizzbuzz(6)
```

```
## [1] "fizz"
```

```r
fizzbuzz(7)
```

```
## [1] 7
```

```r
fizzbuzz(15)
```

```
## [1] "fizzbuzz"
```

Exercise 8 (Website: 19.4.4 Ex. 4 — Modified)

Demonstrate how to use `case_when()` to simplify the nested if-else statements below.

```
if (temp <= 0) {
  "freezing"
} else if (temp <= 10) {
  "cold"
} else if (temp <= 20) {
  "cool"
} else if (temp <= 30) {
  "warm"
} else {
  "hot"
}
```

```
x <- 22
case_when(
  x <= 0 ~ 'freezing',
  x <= 10 ~ 'cold',
  x <= 20 ~ 'cool',
  x <= 30 ~ 'warm',
  x > 30 ~ 'hot'
)
```

```
## [1] "warm"
```

Exercise 9 (Website: 19.4.4 Ex. 6)

What does this `switch()` call do? What happens if `x` is "e"?

```
switch(x,
  a = ,
  b = "ab",
  c = ,
  d = "cd"
)
```

This `switch()` call will return `'ab'` if `x` is `'a'` or `'b'`, and will return `'cd'` if `x` is `'ac'` or `'d'`. Otherwise, it will return `NULL`, like with `'e'`.

Exercise 10 (Website: 19.5.5 Ex. 4)

The default value for the `method` argument to `cor()` is `c("pearson", "kendall", "spearman")`. What does that mean? What value is used by default? **This means that the three methods available for calculating correlation are 'pearson', 'kendall', and 'spearman', named after statisticians. The default is 'pearson'.**