L13 Vectors and Iteration

Data Science I (STAT 301-1)

Shay Lebovitz

Contents

Overview 1
Datasets 1
Exercises 1

Overview

The goal of this lab is to improve our understanding of vectors (a basic structure in R) and iterative programming skills, using the `purrr` package from the tidyverse. This resource may help:

- `purrr` tidyverse homepage

Datasets

Datasets referenced in this lab are either from the tidyverse or from the familiar `nycflights13` package.

Exercises

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

```
# Loading package(s)
library(tidyverse)
library(nycflights13)
library(readr)
```

Load Packages

Exercise 1 (Website: 20.4.6 Ex. 1)

What does `mean(is.na(x))` tell you about a vector `x`? What about `sum(!is.finite(x))`? **`mean(is.na(x))` would give you the proportion of `NA` in a vector. `sum(!is.finite(x))` would give you a count of all the infinite elements in a vector.**

Exercise 2 (Website: 20.4.6 Ex. 5)

Why is `x[-which(x > 0)]` not the same as `x[x <= 0]`?

```
x <- c(-3, -1, 0, 1, 4, Inf, -Inf, NA, NaN)
x[-which(x > 0)]
```

```
## [1]   -3   -1    0 -Inf   NA  NaN
```

```
x[x <= 0]
```

```
## [1]   -3   -1    0 -Inf   NA   NA
```

`x[-which(x > 0)]` leaves `NA`'s and `NaN`s as is, whereas `x[x <= 0]` replaces `NaN`'s with `NA`'s.

Exercise 3 (Website: 20.5.4 Ex. 2)

What happens if you subset a tibble as if you're subsetting a list? What are the key differences between a list and a tibble?

```
myTibble <- tibble('col1' = c(1, 2, 3, 4), 'col2' = c(5, 6, 7, 8))
myTibble[1]
```

```
## # A tibble: 4 x 1
##    col1
##   <dbl>
## 1     1
## 2     2
## 3     3
## 4     4
```

```
myTibble[[1]]
```

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

Subsetting a tibble with one bracket returns a tibble, whereas doing the same for a list returns a list. The double bracket does the same for both tibbles and list: they return the individual elements.

Exercise 4 (Website: 20.7.4 Ex. 2)

Try to make a tibble containing columns with different lengths. What happens?

```
myTibble2 <- tibble('col1' = c(1, 2, 3), 'col2' = c(5, 6, 7, 8))
```

You can not create a tibble with different column lengths, you get an error.

Exercise 5 (Website: 20.7.4 Ex. 3)

Is it possible to have a list as a column of a tibble?

```
a <- list(4, 5, NA, 'a', 'b')
myTibble3 <- tibble('col1' = a, 'col2' = c(1, 2, 3, 4, 5))
myTibble3
```

```
## # A tibble: 5 x 2
##   col1       col2
```

```
##   <list>    <dbl>
## 1 <dbl [1]>     1
## 2 <dbl [1]>     2
## 3 <lgl [1]>     3
## 4 <chr [1]>     4
## 5 <chr [1]>     5
```

Theoretically you can, but it is not practically useful due to nesting issues.

Exercise 6 (Website: 21.2.1 Ex. 1)

Write for loops to:

- Compute the mean of every column in `mtcars`.
- Determine the type of each column in `nycflights13::flights`.
- Compute the number of unique values in each column of `iris`.
- Generate 10 random draws from a normal distribution with $\mu = -10$, 0, 10, and 100.

Think about the output, sequence, and body of each loop before you start writing it.

```
means <- vector('double', ncol(mtcars))
for (i in seq_along(mtcars)) {
  means[i] = mean(mtcars[[i]])
}
means
```

```
##  [1]  20.090625   6.187500 230.721875 146.687500   3.596563   3.217250
##  [7]  17.848750   0.437500   0.406250   3.687500   2.812500
```

```
col_type <- vector('character', ncol(flights))
for (i in seq_along(flights)) {
  col_type[i] = typeof(flights[[i]])
}
col_type
```

```
##  [1] "integer"   "integer"   "integer"   "integer"   "integer"   "double"
##  [7] "integer"   "integer"   "double"    "character" "integer"   "character"
## [13] "character" "character" "double"    "double"    "double"    "double"
## [19] "double"
```

```
view(flights)

num_unique <- vector('integer', ncol(iris))
for (i in seq_along(iris)) {
  num_unique[i] = length(unique(iris[[i]]))
}
num_unique
```

```
## [1] 35 23 43 22  3
```

```
norm_dist <- vector('list', 4)
mu <- c(-10, 0, 10, 100)
for (i in seq_along(norm_dist)) {
  norm_dist[[i]] = rnorm(10, mean = mu[i])
}
norm_dist
```

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

```
##  [1]  -8.570702  -9.868547 -10.782991  -9.135421 -10.327494  -9.640191
##  [7]  -8.998848 -11.227847 -10.035393 -11.486441
##
## [[2]]
##  [1] -3.1378973  2.1591393 -0.2440069 -0.1406911  1.1891850  0.1144654
##  [7]  0.5604635  1.0042404  0.4802929  1.2633342
##
## [[3]]
##  [1] 10.567096 10.890726 10.623337  9.660844  9.669222 11.338140 10.445854
##  [8] 12.371605 10.050879  9.223419
##
## [[4]]
##  [1]  99.92805  99.17429 100.20113 100.86725 100.21087  98.77748 101.37475
##  [8]  99.41844  99.78901 101.97932
```

Exercise 7 (Website: 21.3.5 Ex. 1)

Imagine that you have a directory full of CSV files that you want to read in to R. You have their paths in a vector, `files <- dir("data/", pattern = "\\.csv$", full.names = TRUE)`, and want to read each one with `read_csv()`. Write a for loop that will load them into a single data frame.

```
files <- dir("data/", pattern = "\\.csv$", full.names = TRUE)
num_files <- length(files)
file_df <- vector('list', num_files)
for (i in seq_along(files)) {
  file_df[[i]] = read_csv(files[[i]])
}
file_df <- bind_rows(file_df)
```

Exercise 8 (Website: 21.3.5 Ex. 3)

Write a function that prints the mean of each numeric column in a data frame along with its name. For example, `show_mean(iris)` should print:

```
show_mean(iris)
#> Sepal.Length: 5.84
#> Sepal.Width:  3.06
#> Petal.Length: 3.76
#> Petal.Width:  1.20
```

```
show_mean <- function(df) {
  for (name in seq_along(df)) {
    if (is.numeric(df[[name]])) {
      print(c(names(df)[name], ": ", mean(df[[name]])))
    }
  }
}
show_mean(iris)
```

```
## [1] "Sepal.Length"     ": "               "5.84333333333333"
## [1] "Sepal.Width"      ": "               "3.05733333333333"
## [1] "Petal.Length" ": "           "3.758"
## [1] "Petal.Width"      ": "               "1.19933333333333"
```

Note: it's not pretty but it gets the job done, and I'm not very familiar with fromatting print lines.

Exercise 9 (Website: 21.5.3 Ex. 1)

Write code that uses one of the map functions to:

- Compute the mean of every column in `mtcars`.
- Determine the type of each column in `nycflights13::flights`.
- Compute the number of unique values in each column of `iris`.
- Generate 10 random normals for each of $\mu = -10$, 0, 10, and 100.

```
map_dbl(mtcars, mean)
```

```
##        mpg        cyl       disp         hp       drat         wt       qsec
##  20.090625   6.187500 230.721875 146.687500   3.596563   3.217250  17.848750
##         vs         am       gear       carb
##   0.437500   0.406250   3.687500   2.812500
```

```
map(flights, typeof)
```

```
## $year
## [1] "integer"
##
## $month
## [1] "integer"
##
## $day
## [1] "integer"
##
## $dep_time
## [1] "integer"
##
## $sched_dep_time
## [1] "integer"
##
## $dep_delay
## [1] "double"
##
## $arr_time
## [1] "integer"
##
## $sched_arr_time
## [1] "integer"
##
## $arr_delay
## [1] "double"
##
## $carrier
## [1] "character"
##
## $flight
## [1] "integer"
##
## $tailnum
## [1] "character"
##
## $origin
## [1] "character"
```

```
##
## $dest
## [1] "character"
##
## $air_time
## [1] "double"
##
## $distance
## [1] "double"
##
## $hour
## [1] "double"
##
## $minute
## [1] "double"
##
## $time_hour
## [1] "double"
```

```
map_int(iris, n_distinct)
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width      Species
##           35           23           43           22            3
```

```
map(c(-10, 0, 10, 100), ~rnorm(n = 10, mean = .))
```

```
## [[1]]
##  [1] -11.516111 -11.655894  -9.992923 -10.101011  -9.701573 -10.656301
##  [7] -10.542919  -8.254765  -9.144136  -9.796250
##
## [[2]]
##  [1]  0.12626639 -1.22695731 -0.49251532 -0.77321671 -0.72359170 -1.32026626
##  [7]  1.31965260 -0.05420319 -0.19098626 -0.59998502
##
## [[3]]
##  [1]  9.447817 10.882168 10.779855  9.281482  9.876865 10.118011 11.274408
##  [8]  9.831129 10.248930 10.646574
##
## [[4]]
##  [1]  99.81377 100.54563  99.21948  99.40902 100.63160  96.72199  99.97632
##  [8] 100.29100 100.96141  99.64610
```

Exercise 10 (Website: 21.5.3 Ex. 5)

Rewrite `map(x, function(df) lm(mpg ~ wt, data = df))` to eliminate the anonymous function.

```
x <- split(mtcars, mtcars$cyl)
map(x, ~lm(mpg ~ wt, data = .))
```

```
## $`4`
##
## Call:
## lm(formula = mpg ~ wt, data = .)
##
## Coefficients:
```

```
## (Intercept)           wt
##      39.571       -5.647
##
##
## $`6`
##
## Call:
## lm(formula = mpg ~ wt, data = .)
##
## Coefficients:
## (Intercept)           wt
##       28.41        -2.78
##
##
## $`8`
##
## Call:
## lm(formula = mpg ~ wt, data = .)
##
## Coefficients:
## (Intercept)           wt
##      23.868       -2.192
```