STAT 33B Workbook 1

Ming Fong (3035619833)

Sep 3, 2020

This workbook is due **Sep 3, 2020** by 11:59pm PT.

The workbook is organized into sections that correspond to the lecture videos for the week. Watch a video, then do the corresponding exercises *before* moving on to the next video.

Workbooks are graded for completeness, so as long as you make a clear effort to solve each problem, you'll get full credit. That said, make sure you understand the concepts here, because they're likely to reappear in homeworks, quizzes, and later lectures.

As you work, write your answers in this notebook. Answer questions with complete sentences, and put code in code chunks. You can make as many new code chunks as you like.

In the notebook, you can run the line of code where the cursor is by pressing `Ctrl` + `Enter` on Windows or `Cmd` + `Enter` on Mac OS X. You can run an entire code chunk by clicking on the green arrow in the upper right corner of the code chunk.

Please do not delete the exercises already in this notebook, because it may interfere with our grading tools.

Data Types

Watch the "Data Types" lecture video.

Exercise 1

In R, if you pass vectors with different lengths to a binary operator, the shorter one will be **recycled**. This means the elements of the shorter vector will be repeated to match the length of the longer vector.

Use the recycling rule to explain what's happening in each of these lines of code:

```r
c(1, 2) - c(3, 4, 5, 6)
```

```
## [1] -2 -2 -4 -4
```

```r
c(20, 30, 40) / 10
```

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

```r
c(1, 3) + c(0, 0, 0, 0, 0)
```

```
## Warning in c(1, 3) + c(0, 0, 0, 0, 0): longer object length is not a multiple of
## shorter object length
```

```
## [1] 1 3 1 3 1
```

YOUR ANSWER GOES HERE:

1. The first vector is recycled once and becomes `1 2 1 2` before being subtracted by the second vector.
2. The second vector recycled 3 times into `10 10 10`. Each of the values in the first vector are divided by the corresponding 10 in the second vector.

3. The `1 3` is recycled into `1 3 1 3 1`. The last `3` is cut off and throws a warning.

Exercise 2

Run each line in the following code chunk and inspect the result. For each one, state the type and class of the result, and explain why the result has that type.

```
c(TRUE, "hello", 3, 6)
```

```
## [1] "TRUE"  "hello" "3"     "6"
```

```
3L + 3i
```

```
## [1] 3+3i
```

```
c(3L, 4L, 5L) / TRUE
```

```
## [1] 3 4 5
```

YOUR ANSWER GOES HERE:

1. type: character, class: character. Each value is coerced into a character because there is a character in the argument. Character is the most dominant type/class.
2. type: complex, class: complex. The interger is coerced into numeric then complex.
3. type: double, class: numeric. The division function returns a double which is also a numeric.

Exercise 3

Another way to create vectors is with the `rep()` function. The `rep()` function creates a vector by replicating a value or vector of values.

1. The first parameter of `rep()` is the thing to replicate. The second parameter, `times`, is the number of times to to replicate. Use `rep()` to make a vector with 10 elements, all equal to 78.

2. What happens if you pass a vector as the first argument to `rep()`? Give some examples.

3. Skim the help file `?rep`. What happens if you pass a vector as the second argument to `rep()`? The help file might seem a bit cryptic, so you'll also need to experiment. Give some examples.

YOUR ANSWER GOES HERE:

1.

```
x = rep(78, 10)
x
```

```
##  [1] 78 78 78 78 78 78 78 78 78 78
```

2. It will repeat the argument vector `times` times.

```
y = c(1, 2, 3)
y
```

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

```
z = rep(y, 3)
z
```

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

3. When the first argument and the `times` argument have the same length, the new vector will repeat the first element `times[1]` times, the second `times[2]` times, and the nth element `times[n]` times.

```
x = c(1, 2, 3)
x
```

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

```
y = rep(c(2, 4, 8), x)
y
```

```
## [1] 2 4 4 8 8 8
```

Exercise 4

Yet another way to create vectors is with the `seq()` function. The `seq()` function creates a vector that contains a sequence of numbers.

Skim the help file `?seq`. Give some examples of creating vectors with the `seq()` function.

YOUR ANSWER GOES HERE:

```
x = seq(2, 10, 2)
x
```

```
## [1]  2  4  6  8 10
```

```
y = seq(2, 100, 2)
y
```

```
##  [1]   2   4   6   8  10  12  14  16  18  20  22  24  26  28  30  32  34  36  38
## [20]  40  42  44  46  48  50  52  54  56  58  60  62  64  66  68  70  72  74  76
## [39]  78  80  82  84  86  88  90  92  94  96  98 100
```

Exercise 5

In R, `T` and `F` are shortcuts for `TRUE` and `FALSE`.

1. What happens if you try to assign a value to `TRUE`?
2. What happens if you try to assign a value to `T`?
3. Check that what you observed in #1 and #2 is also true for `FALSE` and `F`. Why might it be safer to use `TRUE` and `FALSE` rather than `T` and `F` in code?

YOUR ANSWER GOES HERE:

1. It throws an error: invalid (do_set) left-hand side to assignment.

```
TRUE = 15
```

```
## Error in TRUE = 15: invalid (do_set) left-hand side to assignment
```

2. The value is assigned successfully

```
T = 15
```

3. Using the full word could avoid bugs from accidentally assigning values to `T` and `F`.

```
FALSE = 888
```

```
## Error in FALSE = 888: invalid (do_set) left-hand side to assignment
```

```
F = 888
```

Matrices, Arrays, & Lists

Watch the "Matrices, Arrays, & Lists" lecture video.

Exercise 6

Recall that many of R's functions are vectorized, which means they are applied element-by-element to vectors.

1. What happens if you call a vectorized function on a matrix?
2. What happens if you call a vectorized function on an array?

Give examples to support your answer.

YOUR ANSWER GOES HERE:

1. Each element in `a` is added to the corresponding element in `b`. Vectorized functions only work on matrices with the proper dimensions.

```
m1 = matrix(seq(1, 10), 2)
m2 = matrix(seq(2, 20, 2), 2)
m3 = matrix(c(1, 2, 3), 3)
m1
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    3    5    7    9
## [2,]    2    4    6    8   10
```

```
m2
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    2    6   10   14   18
## [2,]    4    8   12   16   20
```

```
m3
```

```
##      [,1]
## [1,]    1
## [2,]    2
## [3,]    3
```

```
m1 + m2
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    3    9   15   21   27
## [2,]    6   12   18   24   30
```

```
m1 + m3
```

```
## Error in m1 + m3: non-conformable arrays
```

2. The same happens as with a matrix.

```
a = array(1:8, c(2, 2, 2))
b = array(9:16, c(2, 2, 2))
a
```

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

```
## , , 2
##
##      [,1] [,2]
## [1,]    5    7
## [2,]    6    8
```

```
b
```

```
## , , 1
##
##      [,1] [,2]
## [1,]    9   11
## [2,]   10   12
##
## , , 2
##
##      [,1] [,2]
## [1,]   13   15
## [2,]   14   16
```

```
a + b
```

```
## , , 1
##
##      [,1] [,2]
## [1,]   10   14
## [2,]   12   16
##
## , , 2
##
##      [,1] [,2]
## [1,]   18   22
## [2,]   20   24
```

Exercise 7

Suppose we want to multiply a length-2 vector with a 2-by-2 matrix.

What happens if you use `*` to multiply them? What happens if you use `%*%`?

Give some examples that show the difference, including for vectors and matrices of other sizes.

YOUR ANSWER GOES HERE:

Using `*` turns the vector into a 2x1 matrix and multiplies across. Using `%*%` performs matix multiplication with dot products.

```
v = c(1, 2)
m = matrix(1:4, 2)
v
```

```
## [1] 1 2
```

```
m
```

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

```
v * m
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    4    8
```

```r
v %*% m
```

```
##      [,1] [,2]
## [1,]    5   11
```

```r
v = c(3, 6, 9)
m = matrix(1:9, 3)
v
```

```
## [1] 3 6 9
```

```r
m
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

```r
v * m
```

```
##      [,1] [,2] [,3]
## [1,]    3   12   21
## [2,]   12   30   48
## [3,]   27   54   81
```

```r
v %*% m
```

```
##      [,1] [,2] [,3]
## [1,]   42   96  150
```

Exercise 8

The `c()` function combines vectors, but it can also combine lists. Use `list()` to create two lists, and show that `c()` can be used to combine them.

YOUR ANSWER GOES HERE:

```r
a = list(1, 4, 7, 12)
b = list(TRUE, "asdf", 5L)

c(a, b)
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 4
##
## [[3]]
## [1] 7
##
## [[4]]
## [1] 12
##
## [[5]]
## [1] TRUE
```

```
##
## [[6]]
## [1] "asdf"
##
## [[7]]
## [1] 5
```

Special Values

Watch the "Special Values" lecture video.

Exercise 9

Skim the help file for the `mean()` function.

1. What happens if you call the mean function on a vector that contains missing values? Is there a way to override this behavior?
2. What happens if you call the mean function on a vector that contains `NaN` values or infinite values?

In each case, provide examples to suport your answers.

YOUR ANSWER GOES HERE:

1. The output will be `NA`. This can be prevented by passing `na.rm = TRUE` to strip `NA` values first.

```r
x = c(1, 2, 3, NA, 5)
mean(x)
```

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

```r
mean(x, na.rm = TRUE)
```

```
## [1] 2.75
```

2. Passing `NaN` returns `NaN` from the mean function. Passing `Inf` returns `Inf`, which makes sense mathematically.

```r
mean(c(1, 2, 3, NaN, 4))
```

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

```r
mean(c(1, 2, 3, NaN, 4), na.rm = TRUE) # NaN can be removed with na.rm = TRUE
```

```
## [1] 2.5
```

```r
mean(c(1, 2, 3, Inf, 4))
```

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

```r
mean(c(1, 2, 3, Inf, 4), na.rm = TRUE) # na.rm will not remove Inf
```

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

Making Comparisons

Watch the "Making Comparisons" lecture video.

Exercise 10

Each of the following lines of code produces a result that, at a glance, you might not expect. Explain the reason for each result.

```
3 == "3"
```

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

```
50 < '6'
```

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

```
isTRUE("TRUE")
```

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

YOUR ANSWER GOES HERE:

1. `3` is coerced into the character `"3"` which makes the two sides the same character.
2. The `50` is coerced into a character `50`. `"6"` is greater because the operator orders characters alphabetically (based on locale settings).
3. `"TRUE"` in quotes is a character, not a logical. Thus the character is not `TRUE`.

Exercise 11

Suppose you want to check whether any of the values in `c(1, 2, 3)` appear in the vector `c(4, 1, 3, 1)`.

Novice R users often expect they can check with the code:

```
c(1, 2, 3) == c(4, 1, 3, 1)
```

```
## Warning in c(1, 2, 3) == c(4, 1, 3, 1): longer object length is not a multiple
## of shorter object length
```

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

1. Explain why the code above is not correct, and what's actually happening.

2. The correct way is to use the `%in%` operator. Give some examples of using the `%in%` operator. Recall that you can access its help page with `?"%in%"`.

YOUR ANSWER GOES HERE:

1. The nth value of the left vector is checked with the nth value of the right vector. The first vector is also recycled so the 1st left value is checked with the 4th right value.
2.

```
c(1, 2, 3) %in% c(4, 1, 3, 1)
```

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

```
c(1, 44) %in% c(2, 4, 12, 1, 0)
```

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

Submitting Your Work

Congratulations, you made it through the first workbook!

You need to submit your work in two places:

- Submit this Rmd file with your edits on bCourses.
- Knit and submit the generated PDF file on Gradescope.