STAT 33A Workbook 2

CJ HINES (3034590053)

Sep 10, 2020

This workbook is due **Sep 10, 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.

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.

Data Types

Watch the "Data Types" lecture video.

Exercise 1

Recall that 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:

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

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

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

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

```
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:

> For the first one, the answer is -2 -2 -4 -4 which we get by recycling 1 and 2 (the values of the first vector) to subtract the first/third and second/fourth values respectively of the second vector (1-3) (2-4) (1-5) (2-6).
>
> For the second one, the answer is 2 3 4 which we get by recycling 10 by dividing all the values of the vector (20 30 40) by 10.
>
> For the third one, the answer is 1 3 1 3 1 which we get by recycling 1 and 3 when adding the two vectors: (1+0) (3+0) (1+0) (3+0) (1+0).

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 use the implicit coercion rule to 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:

> `c(TRUE, "hello", 3, 6)`
>
> - Type = character. Class = character.
>
> Implicit coercion tells us that data in R will automatically change classes if coerced with different data types in this order: logical -> integer -> numeric -> complex -> character. In this case, TRUE (logical) became "TRUE" (character) and 3 and 6 (numeric) became "3" and "6" (character).
>
> `3L + 3i`

- Type = complex. Class = complex.

  Implicit coercion tells us that data in R will automatically change classes if coerced with different data types in this order: logical -> integer -> numeric -> complex -> character. In this case, 3L (integer) when added to 3i (complex) became 3+3i (complex).

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

- Type = double. Class = numeric.

  Implicit coercion tells us that data in R will automatically change classes if coerced with different data types in this order: logical -> integer -> numeric -> complex -> character. In this case, c(3L, 4L, 5L) (integer) when divided by TRUE (logical) became numeric (type=double).

Exercise 3

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. `TRUE = 10` returns an Error.
2. `T = 10` assigns the value 10 to T.
3. `FALSE = 9` returns an Error and `F = 9` assigns the value 9 to F. It is safer to use `TRUE` and `FALSE` rather than `T` and `F` in code because while `T` and `F` are convenient shortcuts, it is easier to read and less likely to mistake an assigned variable (i.e `F = 9`) when using `TRUE` and `FALSE`.

Matrices, Arrays, & Lists

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

Exercise 4

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. It applies the function to every value in the matrix. Examples:

```
sin(matrix(seq(5, 7), 3))
```

```
##            [,1]
## [1,] -0.9589243
## [2,] -0.2794155
## [3,]  0.6569866
```

```
tan(matrix(seq(1, 6), 3))
```

```
##            [,1]       [,2]
## [1,]  1.5574077  1.1578213
## [2,] -2.1850399 -3.3805150
## [3,] -0.1425465 -0.2910062
```

2. It applies the function to every value in the array. Examples:

```
sin(array(seq(0,5), c(5, 5)))
```

```
##            [,1]       [,2]       [,3]       [,4]       [,5]
## [1,]  0.0000000 -0.9589243 -0.7568025  0.1411200  0.9092974
## [2,]  0.8414710  0.0000000 -0.9589243 -0.7568025  0.1411200
## [3,]  0.9092974  0.8414710  0.0000000 -0.9589243 -0.7568025
## [4,]  0.1411200  0.9092974  0.8414710  0.0000000 -0.9589243
## [5,] -0.7568025  0.1411200  0.9092974  0.8414710  0.0000000
```

```
tan(array(seq(1,10), c(2,9)))
```

```
##           [,1]       [,2]       [,3]      [,4]       [,5]      [,6]       [,7]
## [1,]  1.557408 -0.1425465 -3.3805150  0.871448 -0.4523157  1.557408 -0.1425465
## [2,] -2.185040  1.1578213 -0.2910062 -6.799711  0.6483608 -2.185040  1.1578213
##            [,8]      [,9]
## [1,] -3.3805150  0.871448
## [2,] -0.2910062 -6.799711
```

Exercise 5

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:

> When we multiply a length-2 vector with a 2-by-2 matrix using `*` it multiplies the first value of the vector with every value in row 1 of the matrix, the second value with every value in the second row, and so on.
>
> When we multiply a length-2 vector with a 2-by-2 matrix using `%*%` it multiplies the values in the vector with their respective index values in the first column of the matrix and then adds those values to create the first value of the new matrix, and then does the same with the remaining columns.

```r
vec = c(1,2)
mat = matrix(vec, 2, 2)
vec * mat
```

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

```r
vec %% mat
```

```
##      [,1] [,2]
## [1,]    0    0
## [2,]    0    0
```

```r
v1 = c(1,2,3)
m1 = matrix(seq(1,9), 3)
v1 * m1
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    4   10   16
## [3,]    9   18   27
```

```r
v1 %*% m1
```

```
##      [,1] [,2] [,3]
## [1,]   14   32   50
```

```r
v2 = c(1,2,3,4)
m2 = matrix(seq(1,24), 4)
v2 * m2
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    5    9   13   17   21
## [2,]    4   12   20   28   36   44
## [3,]    9   21   33   45   57   69
## [4,]   16   32   48   64   80   96
```

```r
v2 %*% m2
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]   30   70  110  150  190  230
```

Exercise 6

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:

```
l1 = list(p = c(2,4), c(10,20,90), c(55, 555, 5))
l2 = list(c(1,2), c(3,4,5))
c(l1, l2)
```

```
## $p
## [1] 2 4
##
## [[2]]
## [1] 10 20 90
##
## [[3]]
## [1]  55 555   5
##
## [[4]]
## [1] 1 2
##
## [[5]]
## [1] 3 4 5
```

Special Values

Watch the "Special Values" lecture video.

Exercise 7

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. It returns NA. It can beoverrided by setting na.rm = TRUE.

```
mean(c(1,2,NA))
```

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

```
mean(c(1,2,NA), na.rm=TRUE)
```

```
## [1] 1.5
```

2. It returns NaN or Inf.

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

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

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

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

Making Comparisons

Watch the "Making Comparisons" lecture video.

Exercise 8

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

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

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

```r
50 < '6'
```

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

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

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

YOUR ANSWER GOES HERE:

> For these, implicit coercion tells us that data in R will automatically change classes if coerced with different data types in this order: logical -> integer -> numeric -> complex -> character.
>
> In the case of `3 == "3"`, 3 (numeric) became "3" (character) to check equality which is TRUE.
>
> In the case of `50 < '6'`, 50 (numeric) became "50" (character) to check if "50" is less than "6" and becaue R uses dictionary ordering and "5" < "6" is TRUE it returns TRUE.
>
> In the case of `isTRUE("TRUE")`, the value inside the isTRUE(), "TRUE" (character), is not the logical argument, TRUE, so it returns FALSE.

Exercise 9

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 code is not correct because it is checking if the value in each index of the vector is the same as the value in the same index of the other vector. For example, in the code above, 1 is in the first vector and appears twice in the second vector. However, since the 1 is in the first index of the left vector and in the right vector 1 is in the second and fourth index, it does not accurately recognize that 1 appeared in both vectors. The right vector returned true for the fourth indexed 1 only because it was longer than the left vector.
2. Examples of `%in%`

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

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

```
5 %in% c(3, 4, 7, 8)
```

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

```
5 %in% c(55, 15)
```

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