STAT 33B Workbook 3

Yuanrui Zhu (3034615728)

Sep 17, 2020

This workbook is due **Sep 17, 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.

Three Ways to Subset

Watch the "Three Ways to Subset" lecture video.

Exercise 1

Create a variable `count` that contains the integers from 1 to 100 (inclusive).

The `as.character()` function coerces its argument into a character vector. Coerce `count` into a character vector and assign the result to a variable called `fizzy`. Now you have congruent vectors `count` and `fizzy`.

Use subset assignment to replace every number in `fizzy` that's:

- Divisible by 3 with `"Fizz"`
- Divisible by 5 with `"Buzz"`
- Divisible by 15 with `"FizzBuzz"`

Leave all other numbers in `fizzy` as-is.

Print out the final version of `fizzy`. It should begin:

```
 [1] "1"        "2"        "Fizz"     "4"        "Buzz"     "Fizz"
 [7] "7"        "8"        "Fizz"     "Buzz"     "11"       "Fizz"
[13] "13"       "14"       "FizzBuzz" "16"       "17"       "Fizz"
```

Hint 1: Take advantage of the fact that `count` *and* `fizzy` *are congruent.*

Hint 2: The modulo operator `%%` *returns the remainder after dividing its first argument by its second argument. You can use the modulo operator to test whether a number is divisible by some other number (that is, the remainder is zero after divsion).*

YOUR ANSWER GOES HERE:

```r
count = seq(1, 100)
fizzy = as.character(count)
fizzy = replace(fizzy, c(seq(3, 100, 3)), "Fizz")
fizzy = replace(fizzy, c(seq(5, 100, 5)), "Buzz")
fizzy = replace(fizzy, c(seq(15, 100, 15)), "FizzBuzz")
count
```

```
##   [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
##  [19]  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
##  [37]  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
##  [55]  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
##  [73]  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
##  [91]  91  92  93  94  95  96  97  98  99 100
```

```r
fizzy
```

```
##   [1] "1"        "2"        "Fizz"     "4"        "Buzz"     "Fizz"    
##   [7] "7"        "8"        "Fizz"     "Buzz"     "11"       "Fizz"    
##  [13] "13"       "14"       "FizzBuzz" "16"       "17"       "Fizz"    
##  [19] "19"       "Buzz"     "Fizz"     "22"       "23"       "Fizz"    
##  [25] "Buzz"     "26"       "Fizz"     "28"       "29"       "FizzBuzz"
##  [31] "31"       "32"       "Fizz"     "34"       "Buzz"     "Fizz"    
##  [37] "37"       "38"       "Fizz"     "Buzz"     "41"       "Fizz"    
##  [43] "43"       "44"       "FizzBuzz" "46"       "47"       "Fizz"    
##  [49] "49"       "Buzz"     "Fizz"     "52"       "53"       "Fizz"    
##  [55] "Buzz"     "56"       "Fizz"     "58"       "59"       "FizzBuzz"
##  [61] "61"       "62"       "Fizz"     "64"       "Buzz"     "Fizz"    
##  [67] "67"       "68"       "Fizz"     "Buzz"     "71"       "Fizz"    
##  [73] "73"       "74"       "FizzBuzz" "76"       "77"       "Fizz"    
##  [79] "79"       "Buzz"     "Fizz"     "82"       "83"       "Fizz"    
##  [85] "Buzz"     "86"       "Fizz"     "88"       "89"       "FizzBuzz"
##  [91] "91"       "92"       "Fizz"     "94"       "Buzz"     "Fizz"    
##  [97] "97"       "98"       "Fizz"     "Buzz"
```

Logic

Watch the "Logic" lecture video.

Exercise 2

Suppose you conduct a survey and store the results in the following congruent vectors:

```r
# Q: What's your favorite color?
color = c("red", "blue", "blue", "green", "yellow", "green")
color = factor(color)

# Q: Name a dessert you like?
sweet = c("egg tart", "brownie", "ice cream", "ice cream", "fruit", "egg tart")
sweet = factor(sweet)

# Q: Name a desert (not dessert) you like?
dry = c("Kalahari", "Atacama", "Taklamakan", "Sonoran", "Atacama", "Atacama")
dry = factor(dry)

# Q: How old are you?
age = c(23, 15, 92, 21, 28, 45)

# Q: How many UFOs have you seen since 2010?
ufo = c(0, 3, 122, 0, 0, 1)
```

Use the vectors above, comparison operators, and logical operators to compute a logical vector that corresponds to each of the following conditions.

1. People who have seen a UFO.
2. People who have seen a UFO but aren't over 50 years old.
3. People who didn't choose ice cream.
4. People who like both ice cream and the color green.
5. People who like the color red or the color green.

YOUR ANSWER GOES HERE:

```r
#people who have seen a ufo
ufo > 0
```

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

```r
#people who have seen a ufo but aren't over 50 years old
ufo > 0 & age < 50
```

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

```r
#people who didn't choose ice cream
sweet != "ice cream"
```

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

```
#people who like both ice ream and the color green
sweet == "ice cream" & color == "green"
```

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

```
#people who like color red or color green
color == "red" | color == "green"
```

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

Exercise 3

In the expression `(x < 5) == TRUE`, explain why `== TRUE` is redundant.

YOUR ANSWER GOES HERE:

Because the ‘x < 5’ automatically judge whether the argument is true or false, there is no need for explicit statement.

Logical Summaries

Watch the “Logical Summaries” lecture video.

No exercises for this section. You’re halfway finished!

Subset vs. Extract

Watch the “Subset vs. Extract” lecture video.

Exercise 4

A **recursive** list is a list with elements that are also lists.

Here’s an example of a recursive list:

```
mylist = list(list(1i, 2, 3i), list(c("hello", "hi"), 42))
```

Use the recursive list above to answer the following:

1. What’s the first element? What’s the second element?
2. Use the extraction operator `[[` to get the value `3i`.
3. What does `mylist[[c(1, 3)]]` do? What does the index `c(1, 3)` mean here? Experiment with using other vectors in `[[` to get elements from the recursive list. Then explain what the extraction operator `[[` does for recursive lists when the index is a vector.

YOUR ANSWER GOES HERE:

The first element is “list(1i, 2, 3i)”, and the second element is “list(c(”hello“,”hi), 42)"

```
mylist[[1]][[3]]
```

```
## [1] 0+3i
```

```
mylist[[c(1, 3)]]
```

```
## [1] 0+3i
```

```
#this means getting the third argument of the first argument.
```

Exercise 5

For the list `cool_list = list("Hope", "springs", "eternal")`, why is `cool_list[1]` the same as `cool_list[1][1][1][1][1]`? Is this property unique to `cool_list`, or is it a property of all lists? Explain your answer.

YOUR ANSWER GOES HERE:

```
cool_list = list("Hope", "springs", "eternal")
class(cool_list[1][1])
```

```
## [1] "list"
```

```
#the class of cool_list[1] is a list, and so does the cases when random numbers of '[1]' are added afte
```

Subsets of Data Frames

Watch the "Subsets of Data Frames" lecture video.

Exercise 6

For the dogs data, compute:

1. The subset that contains rows 10-20 of the height, weight, and longevity columns.
2. The mean and median of the longevity column (ignoring missing values).
3. The number of dog breeds whose average weight is greater than 42. *Note: the `weight` column is the average weight of each row's breed.*
4. The subset of large dogs that require daily grooming.

YOUR ANSWER GOES HERE:

```
dogs = readRDS("dogs.rds")
dogs[10:20, c("height", "weight", "longevity")]
```

```
##    height weight longevity
## 10  14.50   22.0     12.53
## 11  21.75   47.5     12.58
## 12  10.50   15.0     13.92
## 13  10.25     NA     11.42
## 14     NA   24.0     12.63
## 15  13.00   15.5     11.81
## 16   5.00    5.5     16.50
## 17  10.50     NA     11.05
## 18  20.00     NA     12.87
## 19  19.50   45.0     12.54
## 20  10.50     NA     12.80
```

```
mean(!is.na(dogs$longevity))
```

```
## [1] 0.7848837
```

```
median(!is.na(dogs$longevity))
```

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

```
sum(dogs$breed > 42)
```

```
## [1] 172
```

```
subset(dogs, (size == 'large' & grooming == 'daily'))
```

```
##               breed   group datadog popularity_all popularity lifetime_cost
## 44           Briard herding    2.71            125         79         19673
## 62  Giant Schnauzer working    2.38             95         70         26686
## 67     Afghan Hound   hound    2.08             88         66         24077
## 75           Borzoi   hound    1.89            102         71         16176
## 79 Alaskan Malamute working    1.82             58         47         21986
## 86    Saint Bernard working    1.42             49         43         20022
##    intelligence_rank longevity ailments price food_cost grooming   kids
## 44                30     11.17        1   650       466    daily   high
## 62                28     10.00        1   810      1349    daily medium
## 67                80     11.92        0   890       710    daily   high
## 75                76      9.08        0   675       466    daily medium
## 79                50     10.67        2  1210       710    daily medium
## 86                65      7.78        3   875      1217    daily   high
##    megarank_kids megarank  size weight height
## 44            44       33 large     NA   24.5
## 62            62       67 large   77.5   25.5
## 67            67       60 large   55.0   26.0
## 75            75       82 large   82.5   28.0
## 79            79       83 large   80.0   24.0
## 86            86       81 large  155.0   26.5
```

Exercise 7

The `sort()` function sorts the elements of a vector. For instance:

```
x = c(4, 5, 1)
sort(x)
```

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

The `order()` function is a more flexible alternative to `sort()`. Instead of returning the sorted vector, the `order()` function returns the index that sorts the vector. To actually sort the vector, you have to pass this index to the subset operator `[`:

```
x = c(4, 5, 1)
x[order(x)]
```

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

The advantage of `order()` over `sort()` is that you can use `order()` to sort one vector based on the elements of some other congruent vector.

Use the `order()` function to sort the rows of the dogs data set based on height. What are the 3 tallest breeds of dog?

YOUR ANSWER GOES HERE:

```
dogs$height[order(dogs$height, decreasing = TRUE)][1:3]
```

```
## [1] 32 30 30
```