STAT 33A Workbook 12

CJ HINES (3034590053)

Nov 19, 2020

This workbook is due **Nov 19, 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.

If you have any last-minute trouble knitting, **DON'T PANIC**. Submit your Rmd file on time and follow up in office hours or on Piazza to sort out the PDF.

Tidy Data

Watch the "Tidy Data" lecture video.

No exercises for this section.

Columns into Rows

Watch the "Columns into Rows" lecture video.

No exercises for this section.

Rows into Columns

Watch the "Rows into Columns" lecture video.

String Processing

Watch the "String Processing" lecture video.

Exercise 1

Visit the stringr documentation.

How does the `str_sub` function work? Give 3 examples, including 1 that shows how to use `str_sub` to reassign part of a string.

YOUR ANSWER GOES HERE:

> The `str_sub` function extracts substrings from a character vector.

```
library(stringr)
words = c("The", "Maple", "Leafs", "haven't", "won", "a", "cup", "since", "1967.")

str_sub(words, 2, 4)
```

```
## [1] "he"  "apl" "eaf" "ave" "on"  ""    "up"  "inc" "967"
```

```
str_sub(words, 1, 2)
```

```
## [1] "Th" "Ma" "Le" "ha" "wo" "a"  "cu" "si" "19"
```

```
str_sub(words, 2, 2) = "woo"
words
```

```
## [1] "Twooe"     "Mwoople"   "Lwooafs"   "hwooven't" "wwoon"     "awoo"
## [7] "cwoop"     "swoonce"   "1woo67."
```

Regular Expressions

Watch the "Regular Expressions" lecture video.

If you use RStudio, the RegExplain RStudio addin makes it easier to learn how to use regular expressions. You can find information about how to install RegExplain on the documentation page for stringr.

If you decide not to install RegExplain, there are many online regular expression testers that can be helpful when learning. For example, I like https://regex101.com.

Exercise 2

The lecture video "Printing Output" described how R interprets backslashes in strings as escape sequences. For instance, the string `"\t"` is a tab character.

Because backslashes have a special meaning, to put a literal backslash in a string, you have to write the backslash twice: `"\\"`.

Regular expressions also use backslash as a special character. In a regular expression, putting a backslash in front of a metacharacter causes it to be interpreted literally. For instance, the pattern `"\."` matches a single, literal dot (rather than being a wildcard).

R's rule for backslashes interacts with regex's rule for backslashes in an unfortunate way. Regex patterns in R are just strings, and R interprets `"\."` (or backslash followed by any other character) as an escape sequence. Since we need a literal backslash, we have to write `"\\."` in R. Then the regular expression system sees this as `"\."`, and searches for a literal `"."`.

This interaction is especially bad if you want to search for a literal backslash with a regular expression. You'd need to write `"\\\\"`!

As a remedy, R version 4.0 introduced *raw strings*. In a raw string, R always interprets backslashes literally (rather than as escape sequences).

1. Raw strings are documented in `?Quotes`. Find the section in that file and read it.
2. Give an example of creating a raw string.
3. Write a vectorized function `has_backslash` with parameter `x` that returns `TRUE` if `x` contains a backslash and `FALSE` otherwise. In your function, use `str_detect` with a raw string for the pattern.

YOUR ANSWER GOES HERE:

```
r"(\\goodbye\world\)"
```

```
## [1] "\\\\goodbye\\world\\"
```

```
has_backslash = function(x) {
  word = as.character(x)
  str_detect(word, r"(\\)")
}

has_backslash("ok\\")
```

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

Exercise 3

Write a function `extract_phone` that extracts a phone number from a string. Your function should return the phone number as a string in the format `NNN-NNNN` without any other characters.

Test your function on the following strings:

```
ex1 = "Phone: 555-2920"
ex2 = "Hi! The number you've called is 555-3131!"
ex3 = "The phone for unit 4342 is 555-9753."
```

Hint 1: Use `str_match` to extract the number.

Hint 2: Take advantage of non-numeric characters that mark the boundaries of a phone number.

YOUR ANSWER GOES HERE:

```
nums = "[0-9]{3}[- .][0-9]{4}"
extract_phone = function(x) {
  woo = str_match(x, nums)
  woo
}

extract_phone(ex1)
```

```
##      [,1]
## [1,] "555-2920"
```

```
extract_phone(ex2)
```

```
##      [,1]
## [1,] "555-3131"
```

```
extract_phone(ex3)
```

```
##      [,1]
## [1,] "555-9753"
```