HW 6

Lance Ding

March 2023

Instructions

Write this homework acting as if I don't know what I asked you. For example, don't simply list question numbers for the headings. If you gave this document to someone else who didn't know the assignment, they should be able to understand what you did by reading the headings, code, and accompanying text.

Look to my HW1 and RMarkdown Organization examples for how to write good headings and organize your assignment.

This HW is worth 10 total points. It leans heavily on activities from the Adventures in R Course created by Dr. Kelly Bodwin of California Polytechnic State University.

1. Change the author and date fields in the header above to your name and the date.
2. Make sure to load any packages you may need right at the start. Do *NOT* include the `learnr` package, ever, unless you are writing an interactive Tutorial (which you won't do in this) - this will cause problems.
3. Ensure that no chunks have the `include = FALSE` or `echo = FALSE` option, as I want to be able to see *all* your code and output.
4. Brief but descriptive headings and document organization (answers under headings, text near relevant code, brief explanatory text as indicated below, etc.) (1 pt)
5. For practice with `stringr` as well as more practice with data cleaning in a real world example, I'd like you to work through homework problems 1-4 from Dr. Martin van der Linden's HW5 assignment, which is posted on Canvas alongside this. The data `raw_ab.csv` is also posted alongside the assignment. (4.5 pts)
    - Q2 is a bit tricky because it deals with parentheses, which (like `.`) are special characters in regex. SEE *R for Data Science* section 14.3.1 for a reminder on how to deal with such characters.
6. For more practice with regular expressions, let's decode the secret message in Puzzle 4.1 from Adventures in R. (4.5 pts)

    HINTS AND CLARIFICATIONS:

    - You should do Warm-Up Exercises 1, 3, 4, and 5.
    - Warm-up 4: to identify and highlight any punctuation, you may use the regex character class `"[:punct:]"`

- Warm-up 5: First identify (using code, not manually!) the max length of an element of the scrambled message. Then recall (or perhaps learn for the first time) you can use the notation `<VECTOR>[<LOGICAL CRITERION>]` to pull out only the elements of a vector that meet the stated criterion. For example, if you have a vector `x <- c(1:10)`, what does `x[x %% 2 == 0]` return? Try it!
- Question 2: try `str_sub()`
- Question 6: Unfortunately for some reason I think the message doesn't actually include anything that ends with b plus a punctuation mark. But I still want you to attempt this question as written. To check your work, feel free to simply create a separate length-1 character vector of your own that ends in b with a punctuation mark and see if you can write code to make the indicated change.

  Now, this one is trickier than it looks at first, but you can do it. For example, if you had the word "dumb!" this wants you to change it to "dumy!" But if you have "dumb." it wants you to change it to "dumy." Hard-coding options for each possible punctuation mark is not something I want you to do. Instead, review the section on grouping and backreferences in *R for Data Science* Ch. 14.3.5. Can you figure out how you could use this in your `replacement =` argument to specify that you want it to replace the b with the y but keep any of the subsequent punctuation? You'll need to tell it to *refer back* to the punctuation in your `pattern =` argument.
- Question 7: Don't worry about replacing with the correct case (that is, you need to identify instances of both "k" and "K", but just replace them all with either "v" or "V" - don't worry about matching them up to the case of the original k).
- Don't forget to print the decoded message in an easily human-readable way using `str_c()` and tell me what movie it's from!

7. OPTIONAL OPTIONAL OPTIONAL. THIS IS NOT REQUIRED FOR THE ASSIGNMENT. ONLY DO IT IF YOU'RE CRAVING MORE PRACTICE.

For yet more practice with `stringr`, let's figure out some patterns in Billboard Hot 100 songs. Answer questions 6 and 7 from Lab 3 from Adventures in R. Ignore anything in the Instructions that counters how I tell you to structure a document.

```
HINTS AND CLARIFICATIONS:

* Question 6: the basic path you should take to solve it is as follows. Pull out the column ‘title‘ into

    If you’ve done this correctly, this should produce a list where each element is a character vector w

    You should then take *this* vector, and turn it into a ‘tibble()‘ with one column (the words themsel

    If you’ve done this correctly, the second-highest word should be "your" with 955 instances.

* Question 7: here you should probably begin by splitting the ‘artist‘ column into a main and featured s

    You could then drop any songs with no featued artist(s) since we don’t need to worry about them any

    If you’ve done this correctly, the second-most common featued artist should be "DRAKE" with 40 credi
```

To submit this assignment:

Ideally, knit straight to PDF by changing `html_document` to `pdf_document` in line 5 above. This should work as long as you properly installed LaTeX in Tutorial 0.1. Otherwise:

1. Knit to HTML. An HTML document should open automatically in another RStudio window.

2. Click "Open in Browser" in that HTML document. It should open as a webpage in your default browser (e.g. Chrome).
3. Click Ctrl+P/Command+P, but instead of printing a hard copy on your printer click "Save as PDF."
4. Save and upload that document to Canvas.

——BEGIN ANSWER BELOW——

Load necessary packages

```
raw_ab <- as_tibble(read.csv("./Datasets/raw_ab.csv"))
# raw_ab <- read.csv('./Datasets/raw_ab.csv')
```

Dr. Martin van der Linden HW5 : Exercises with `stringr`

We will be working with Dr. Martin van der Linden's HW5 assignment. Specifically, we will clean the data in `raw_ab.csv` - a dataset containing abortion data from both the **Guttmacher Institue** and the **CDC**.

Removing Missing Values from `raw_ab`

We will start by removing all rows that either:

- Does not contain any number
- is `NA`

We will accomplish that by using `str_detect()` in conjunction with regex to keep only rows that have digits in them. Since rows containing the value `NA` do not have any digits, this operation also removes rows with `NA`. We then store this new dataframe as `ab1`.

```
ab1 <- raw_ab[str_detect(raw_ab$CDC, "\\d"), ] %>%
    drop_na("CDC")
ab1
```

```
## # A tibble: 2,306 x 5
##        X year  CDC    AGI      state
##    <int> <chr> <chr>  <chr>    <chr>
##  1    30 1972  1,156  ""       AL
##  2    32 1974  3,392  "5,080"  AL
##  3    33 1975  2,747  "6,020"  AL
##  4    34 1976  7,278  "8,350"  AL
##  5    35 1977  11,042 "11,680" AL
##  6    36 1978  13,260 "15,120" AL
##  7    37 1979  15,449 "17,590" AL
##  8    38 1980  17,920 "20,780" AL
##  9    39 1981  13,485 "19,840" AL
## 10    40 1982  16,081 "19,930" AL
## # ... with 2,296 more rows
```

Removing Parentheses from the CDC and AGI columns of ab1

Some of the values in the CDC and AGI columns in ab1 contain parentheses. If we are to transform the columns into numerical data for computation, we must remove these parentheses. We use the string escape character with the regex escape character as well as the "or" (|) to find all parentheses, which we then replace with empty strings. This effectively removes all parentheses.

```
ab2 <- ab1
ab2$CDC <- ab2$CDC %>%
    str_replace_all("\\(|\\)", "")
ab2$AGI <- ab2$AGI %>%
    str_replace_all("\\(|\\)", "")
ab2
```

```
## # A tibble: 2,306 x 5
##        X year  CDC    AGI      state
##    <int> <chr> <chr>  <chr>    <chr>
##  1    30 1972  1,156  ""       AL
##  2    32 1974  3,392  "5,080"  AL
##  3    33 1975  2,747  "6,020"  AL
##  4    34 1976  7,278  "8,350"  AL
##  5    35 1977  11,042 "11,680" AL
##  6    36 1978  13,260 "15,120" AL
##  7    37 1979  15,449 "17,590" AL
##  8    38 1980  17,920 "20,780" AL
##  9    39 1981  13,485 "19,840" AL
## 10    40 1982  16,081 "19,930" AL
## # ... with 2,296 more rows
```

Removing Thousand Separators in the CDC and AGI columns of ab2

The values in the CDCD and AGI columns, in their raw form, are formatted such that every thousand, or 3 digits starting from the right, there is a , acting as a separator. Normally, this would increase human readability of large numbers, but in order to do numerical operations on these numbers we have to remove the commas. We do an operation similar to what we did with the parenthesis - finding commas with regex and replacing them with empty strings.

```
ab3 <- ab2
ab3$CDC <- ab3$CDC %>%
    str_replace_all(",", "")
ab3$AGI <- ab3$AGI %>%
    str_replace_all(",", "")
ab3
```

```
## # A tibble: 2,306 x 5
##        X year  CDC   AGI     state
##    <int> <chr> <chr> <chr>   <chr>
##  1    30 1972  1156  ""      AL
##  2    32 1974  3392  "5080"  AL
##  3    33 1975  2747  "6020"  AL
##  4    34 1976  7278  "8350"  AL
##  5    35 1977  11042 "11680" AL
```

```
##  6    36 1978  13260 "15120" AL
##  7    37 1979  15449 "17590" AL
##  8    38 1980  17920 "20780" AL
##  9    39 1981  13485 "19840" AL
## 10    40 1982  16081 "19930" AL
## # ... with 2,296 more rows
```

Converting ab3 to Numeric

We are now ready to convert our numeric values to numeric types in R, which allows for computations. Using `select()`, we first drop the column `X` - we will not need this column. We will then use array indexing to grab the columns we want, then use `as.numeric()` in conjunction with `apply()` to execute this conversion.

```
ab4 <- ab3 %>%
    select("year":"state")
ab4[1:3] <- apply(ab4[, 1:3], 2, function(x) as.numeric(x))
ab4
```

```
## # A tibble: 2,306 x 4
##     year   CDC   AGI state
##    <dbl> <dbl> <dbl> <chr>
##  1  1972  1156    NA AL
##  2  1974  3392  5080 AL
##  3  1975  2747  6020 AL
##  4  1976  7278  8350 AL
##  5  1977 11042 11680 AL
##  6  1978 13260 15120 AL
##  7  1979 15449 17590 AL
##  8  1980 17920 20780 AL
##  9  1981 13485 19840 AL
## 10  1982 16081 19930 AL
## # ... with 2,296 more rows
```

Decoding the Secret Message in `Puzzle 4.1`

Apparently there is a secret message hidden in this puzzle. Let's load it and see what it is about.

```
message <- read.csv("https://www.dropbox.com/s/lgpn3vmksk3ssdo/scrambled_message.txt?dl=1",
    stringsAsFactors = FALSE)$Word
head(message)
```

```
## [1] "              Koila!"      "                In   "
## [3] "               kiew,"      "                    a"
## [5] "           humble   "      "kaudevillianugh?aoghajdbn"
```

It looks like we have a very messy character vector. We will first do the Warm-up exercises, then try to decode the message.

Warm-up Exercises

Number of Characters

The first warm-up wants us to find the number of characters there are in the message. We will assume that this means removing the spaces with `str_trim()` then summing the lengths (`str_length()`) of the remaining strings.

```
message %>%
    str_trim() %>%
    str_length() %>%
    sum()
```

```
## [1] 892
```

Number of Words

According to the challenge, a "word" is a set of characters with no white space - in other words, any non-whitespace element in the `message` vector is a "word". We will use `str_trim()` to remove the whitespace then use `length()` to find the number of elements in the vector.

```
message %>%
    str_trim() %>%
    length()
```

```
## [1] 127
```

Show Punctuation Marks

The next warm-up exercise wants us to show all the punctuation marks in the scrambled message. To view all matches and highlight punctuation, we employ `str_view_all()`, and use the regex character class `[:punct:]` to match all punctuation.

```
str_view_all(message, "[:punct:]")
```

```
## Warning: `str_view()` was deprecated in stringr 1.5.0.
## i Please use `str_view_all()` instead.
```

```
##  [1] |               Koila<!>
##  [2] |                In
##  [3] |                kiew<,>
##  [4] |                    a
##  [5] |            humble
##  [6] | kaudevillianugh<?>aoghajdbn
##  [7] |          kezeran<,>
##  [8] |              casz
##  [9] |          kicariouslb
## [10] |                as
## [11] |                 bozh
## [12] |            kiczim
## [13] |            ughhh<!>and
```

```
## [14] |              killain
## [15] |                bb
## [16] |               zhe
## [17] |         kicissizudes
## [18] |             ughhh<!>of
## [19] |                faze<.>
## [20] |          Thisughhhh<!>
## ... and 107 more
```

Printing Longest Word in Capitals

The final recommended warm-up exercise wants us to print the longest "word" in the scrambled message in all caps. To do this, we must first identify the longest word in the message. We can do that with a composition of `str_length()` and `max()`. Now that we know the max length, we can use this length as a condition to match the string with this length in `message` by using array indexing `[]`. Finally, we use `str_to_upper()` to capitalize the "word".

```
str_to_upper(message[str_length(message) == max(str_length(message))])
```

```
## [1] "KAUDEVILLIANUGH?AOGHAJDBN"
```

Decoding the Message

We will now follow the steps on the challenge page to decode the message. We first remove any whitespace before and after each word using `str_trim()`.

```
message <- message %>%
    str_trim()
# message
```

Then we drop all characters off the end of each word that is more than 16 characters long. To do so, we first locate all strings longer than 16 characters, then we replace those strings with their respective first 16 characters.

```
message[str_length(message) > 16] <- str_sub(message[str_length(message) >
    16], 1, 16)

# Debug
max(str_length(message))
```

```
## [1] 16
```

```
# message
```

The next step wants us to drop all words that contain the character sequence `ugh` followed with any number of h's then punctuation. We achieve the matching by using the regex repetition character `+` and the punctuation character class `[:punct:]`. We then replace the matches with empty strings `""`.

```
message <- message %>%
    str_replace_all("ugh+[:punct:]", "")
# message
```

Following that, we are to replace all instances of exactly 2 `a`s with exactly 2 `e`s. We will do so by finding all elements of `message` that contain exactly 2 `a`s with `str_count()`. We then replace all the `a`s in those strings with `e`s.

I forgot that `str_count()` existed at all before looking at the cheat sheet when checking this. I left my original solution with regex, which should also work.

```
message[str_count(message, "a") == 2] <- message[str_count(message,
    "a") == 2] %>%
    str_replace_all("a", "e")

# message[str_detect(message, '^.*a.*a.*$')] <-
# message[str_detect(message, '^.*a.*a.*$')] %>%
# str_replace_all('a', 'e')

# message
```

The next step asks us to replace all `z`s with `t`s. This is simple - we can use `str_replace_all()` to quickly replace every `z` with a `t`.

```
message <- message %>%
    str_replace_all("z", "t")
# message
```

We then want to change all words that end in a `b` to a `y`. That means even words with punctuation like `dumb!` should be changed to `dumy!`. We use a **look around** to achieve the matching of only the `b` regardless of if there is punctuation behind it, then use `str_replace()` to replace the `b` with a `y`. This solution bypasses the potential need for **backreferencing**.

Also, I couldn't quite figure out how to backreference one string from another string e.g. I don't know how to reference (match1) in the third argument of `str_replace()` - `str_replace(message, "(match1)", "not sure what to put here"))`

```
message <- str_replace(message, "b(?=($|[:punct:]$))", "y")
# message
```

Similarly, we now replace all words that start with a `k` to a `v`. We will detect both capitalization cases via **hard coding**, which is traditionally considered bad practice but since we are only dealing with one case we do not need a general solution. Also, we will not retain the capitalization of the `k` - we will replace all matches with a lowercase `v`.

```
message <- message %>%
    str_replace("^(K|k)", "v")
# message
```

Finally, we combine `message` via `str_c()`.

```r
options(width = 60)
message %>%
    str_c(" ", collapse = "")
```

```
## [1] "voila! In view, a humble veudevillien veteran, cast vicariously as both victim and villain by th
```

A quick google search tells us that this quote is from *V for Vendetta*, said by **Alan Moore**.