Homework 6

Stats 20 Lec 1

Winter 2022

General Guidelines

Please use R Markdown for your submission. Include the following files:

- Your .Rmd file.
- The compiled/knitted HTML document.

Name your .Rmd file with the convention `123456789_stats20_hw0.Rmd`, where `123456789` is replaced with your UID and `hw0` is updated to the actual homework number. Include your first and last name and UID in your exam as well. When you knit to HTML, the HTML file will inherit the same naming convention.

The knitted document should be clear, well-formatted, and contain all relevant R code, output, and explanations. R code style should follow the Tidyverse style guide: https://style.tidyverse.org/.

Any and all course material, including these homework questions, may not be posted online or shared with anyone at any time without explicit written permission by the instructor (Michael Tsiang), even after the quarter is over. Failure to comply is a breach of academic integrity.

Note: All questions on this homework should be done using only functions or syntax discussed in Chapters 1–9 of the lecture notes. No credit will be given for use of outside functions.

Basic Questions

Collaboration on basic questions must adhere to Level 0 collaboration described in the Stats 20 Collaboration Policy.

Question 1

The objective of this question is to complete the application to the Flesch reading ease formula from Chapter 9 and give practice with writing functions involving string manipulation.

(a)

Write a function called `reading_ease()` that will compute the Flesch reading ease score for an input text passage. The Flesch reading ease formula and rules for counting sentences, words, and syllables must adhere to the descriptions given in Section 5 of the Chapter 9 notes.

Your function should be able to compute the reading ease score for an entire text passage if it is inputted as a single character value (i.e., all sentences in a single string) or as a character vector (i.e., sentences may be split into separate strings).

Hint 1: You may use any and all code (particularly the helper functions) provided in the Chapter 9 notes to help in writing your `reading_ease()` function.

Hint 2: The `collapse` argument in the `paste()` function may be useful to accommodate character vectors of length greater than 1.

(b)

Verify that your `reading_ease()` function from (a) works on the `waffles` passage from the Chapter 9 notes (you may source the `waffles.R` file in your Rmd file). Verify that you compute the same Flesch reading ease score for the following vector:

```r
waffles_vec <-
  c("We need to remember what's important in life: friends, waffles, work.",
    "Or waffles, friends, work.",
    "Doesn't matter, but work is third."
  )
```

Note: Code should *never* be copy-pasted from PDFs. Certain characters become non-standard characters when embedded into a PDF. To make sure you and the grader have no issues when knitting your Rmd file, you should *always* retype code provided in PDFs.

Question 2

The objective of this question is to give practice with writing functions involving string manipulation and vectorization.

Write a function called `my_nchar()` that inputs a character vector `x` and outputs the number of characters contained in each entry of the vector without the `nchar()` function. The output of `my_nchar(x)` and `nchar(x)` should be identical for any character vector `x`.

Note: You do not need to consider the optional arguments in the `nchar()` function.

Question 3

The objective of this question is to give practice with writing functions involving string manipulation and vectorization.

Write a function called `my_strrep()` that inputs a character vector `x` and an integer vector `times` that repeats the entries of `x` a given number of times without the `strrep()` function. The entries of the `times` argument specify the number of times to repeat the corresponding entries of `x`, recycling as necessary. The output of `my_strrep(x, times)` and `strrep(x, times)` should be identical for any character vector `x` and integer vector `times`.

Intermediate Questions

Collaboration on intermediate questions must adhere to Level 1 collaboration described in the Stats 20 Collaboration Policy.

Question 4

The objective of this question is to give further practice with writing functions involving string manipulation and vectorization.

Without the `%in%` operator, write an infix operator called `%is_in%` that inputs a vector `x` on the left and a vector `table` on the right and returns one logical value for each element in `x` representing if there exists an element in the `table` vector with that same value. The output of `x %is_in% table` and `x %in% table` should be identical for any vectors `x` and `table`.

Question 5

Without the `regexpr()` and `gregexpr()` functions, write a function called `my_gregexpr()` that inputs a character string `pattern` and a character vector `text` and returns a list of the same length as `text`, each component of which is an integer vector giving the starting positions of every (disjoint) match of `pattern` in the corresponding entry in `text` or `-1` if there is no match. The output values from `my_gregexpr(pattern, text)` and `gregexpr(pattern, text)` should be identical for any character string `pattern` and character vector `text`.

Hint: Your `my_gregexpr()` function only needs to account for literal patterns, i.e., it does not need to work for regular expressions.

Note: You do not need to add attributes to each component of the output list of `my_gregexpr()` to match `gregexpr()`. This will mean `identical(my_gregexpr(pattern, text), gregexpr(pattern, text))` will always return `FALSE`.

Advanced Questions

Collaboration on advanced questions must adhere to Level 1 collaboration described in the Stats 20 Collaboration Policy.

Note: Advanced Questions are intended for further enrichment and a deeper challenge, so they will not count against your grade if they are not completed or attempted.

Question 6

(a)

While `gsub()` is a wonderful function, it has many limitations. Most notably, the `gsub()` function is not vectorized over patterns *or* replacement strings. Consider the example `text` below:

Good TIMEOFDAY STUDENT, how are you doing? STUDENT said you stopped by my office yesterday TIMEOFDAY. Goodbye STUDENT

What if we wanted to replace the first instance of **STUDENT** with **Ethan** and the second instance with **Raymond**? And the first **TIMEOFDAY** with **afternoon** and the second with **morning**?

We can try with `gsub()`:

```r
output <- gsub("STUDENT", c("Ethan", "Raymond"), text)
```

```
Warning in gsub("STUDENT", c("Ethan", "Raymond"), text): argument 'replacement'
has length > 1 and only the first element will be used
```

Good TIMEOFDAY Ethan, how are you doing? Ethan said you stopped by my office yesterday TIMEOFDAY. Goodbye Ethan

This clearly does not work, so we will have to write our own function to do the job.

Read through the following description:

```
FUNCTION: pargsub()
INPUTS:   text, a length-1 character vector.
          patterns, a character vector of string patterns to be replaced.
          replacements, a list of the same length as the patterns vector.
            Each element of replacements is a character vector of at least length-1.
Output:   The input text with each instance of each pattern replaced by the corresponding
          element of the corresponding replacement vector. If there are more instances of
          a particular pattern than elements in that pattern's replacement vector, the
          replacement vector will be recycled.
```

Using the provided description, write the `pargsub()` function (the `par` stands for "parallel").

Example:

Good TIMEOFDAY STUDENT, how are you doing? STUDENT said you stopped by my office yesterday TIMEOFDAY. Goodbye STUDENT

```r
output <- pargsub(text,
                  c("STUDENT", "TIMEOFDAY"),
                  list(c("Ethan", "Raymond"), c("afternoon", "morning")))
```

Good afternoon Ethan, how are you doing? Raymond said you stopped by my office yesterday morning. Goodbye Ethan

Hint 1: Try to avoid loops in favor of vectorized functions. Consider what the output should be for the following expression:

```r
pargsub("red, green, blue", c("red", "green", "blue"), list("green", "blue", "red"))
```

Hint 2: It may be helpful to use your `my_gregexpr()` function in your definition of `pargsub()`.

(b)

Load the contents of the `madlib.RData` file into your workspace and use your `pargsub()` function to fill out the madlib object `text`.

The definition/format of a madlib: https://en.wikipedia.org/wiki/Mad_Libs#Format

The provided madlib was found at:
https://www.madtakes.com/

The word lists were found at:
https://www.momswhothink.com/list-of-nouns/
https://www.momswhothink.com/list-of-verbs/
https://www.momswhothink.com/list-of-adjectives/
https://helpingwithmath.com/printables/tables_charts/cha0301_numbers_words01.htm
http://www.manythings.org/vocabulary/lists/e/words.php?f=body

In order to randomize the selection of words, you will need two functions:

1. **`sample()`**: This will be used to generate a (pseudo)random sample from a vector. For this assignment, the usage will look like this:

   ```r
   x <- 1:10
   sample(x)
   ```

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

   The basic behavior of `sample()` will shuffle (i.e., permute) the values in the input vector into a random order.

2. **`set.seed()`**: This function is required in order to make your work *reproducible*. If you set the system random seed to a value before doing something "random" you can get the same "random" result again by setting your seed to the same value again. See the example below:

   ```r
   set.seed(123)
   sample(x)
   ```

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

   ```r
   sample(x)
   ```

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

   ```r
   set.seed(123)
   sample(x)
   ```

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

   In addition, anyone else who runs your code can get results identical to yours by setting the same random seed.

Try setting a few different random seeds and randomizing the elements of each of the vectors in the `replacements` list you pass to your `pargsub()` function until you get a madlib you think is funny. Include the seed and your completed madlib in your file.