Assignment 3

Alice Ding

2023-02-06

Question 1

Using the 173 majors listed in fivethirtyeight.com's College Majors dataset [https://fivethirtyeight.com/features/the-economic-guide-to-picking-a-college-major/], provide code that identifies the majors that contain either "DATA" or "STATISTICS"

```
majors <- read.csv("https://raw.githubusercontent.com/fivethirtyeight/data/master/college-majors/majors-
to_find <- c("DATA", "STATISTICS")
matches <- unique(grep(paste(to_find, collapse="|"), majors$Major, value=TRUE))
matches
```

```
## [1] "MANAGEMENT INFORMATION SYSTEMS AND STATISTICS"
## [2] "COMPUTER PROGRAMMING AND DATA PROCESSING"
## [3] "STATISTICS AND DECISION SCIENCE"
```

There are three majors that contain either "DATA" or "STATISTICS".

Question 2

Write code that transforms the data below:

[1] "bell pepper" "bilberry" "blackberry" "blood orange"

[5] "blueberry" "cantaloupe" "chili pepper" "cloudberry"

[9] "elderberry" "lime" "lychee" "mulberry"

[13] "olive" "salal berry"

Into a format like this:

c("bell pepper", "bilberry", "blackberry", "blood orange", "blueberry", "cantaloupe", "chili pepper", "cloudberry", "elderberry", "lime", "lychee", "mulberry", "olive", "salal berry")

```
fruits <- '[1] "bell pepper" "bilberry"   "blackberry"  "blood orange"
[5] "blueberry"  "cantaloupe"  "chili pepper" "cloudberry"
[9] "elderberry"  "lime"     "lychee"    "mulberry"
[13] "olive"    "salal berry"'
fruits
```

```
## [1] "[1] \"bell pepper\" \"bilberry\"   \"blackberry\"  \"blood orange\"\n[5] \"blueberry\"  \"canta
```

First, I would remove the [X] numbers.

```
remove_brackets <- gsub(pattern = "\\[[0-9]+\\]", replacement = "", fruits)
remove_brackets
```

```
## [1] " \"bell pepper\" \"bilberry\"   \"blackberry\"  \"blood orange\"\n \"blueberry\"  \"cantaloupe\
```

Next, I would clean up the spaces and add commas.

```
clean_spaces <- gsub(pattern = "\\s+", replacement = " ", remove_brackets)
add_commas <- gsub(pattern = "\" ", replacement = "\", ", clean_spaces)
trim_whitespace <- trimws(add_commas)
trim_whitespace
```

```
## [1] "\"bell pepper\", \"bilberry\", \"blackberry\", \"blood orange\", \"blueberry\", \"cantaloupe\",
```

Then, I would add the "c(" to the front and ")" at the end.

```
add_p <- paste0("c(", trim_whitespace, ")")
add_p
```

```
## [1] "c(\"bell pepper\", \"bilberry\", \"blackberry\", \"blood orange\", \"blueberry\", \"cantaloupe\
```

Now, I would remove the slashes.

```
remove_slashes <- gsub(pattern = "\\\\", replacement = "", add_p)
remove_slashes
```

```
## [1] "c(\"bell pepper\", \"bilberry\", \"blackberry\", \"blood orange\", \"blueberry\", \"cantaloupe\
```

It looks like these backslashes just denote " " so they can't be removed.

Question 3

Describe, in words, what these expressions will match:

Expression 1

`(.)\1\1`

This expression would likely not match anything, however if reformatted to `(.)\\1\\1`, it would find three consecutive identical characters.

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.4.0      v purrr   1.0.1
## v tibble  3.1.8      v dplyr   1.0.10
## v tidyr   1.3.0      v stringr 1.5.0
## v readr   2.1.3      v forcats 0.5.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
q3_words <- c("aaa", "aa", "a", "abcdefg", "mmmm", "babba", "baba", "banana", "abacada", "racecar", "TA
str_subset(q3_words, "(.)\1\1")
```

```
## character(0)
```

```
str_subset(q3_words, "(.)\\1\\1")
```

```
## [1] "aaa"  "mmmm"
```

```
str_match(q3_words, "(.)\\1\\1")
```

```
##       [,1]  [,2]
##  [1,] "aaa" "a"
##  [2,] NA    NA
##  [3,] NA    NA
##  [4,] NA    NA
##  [5,] "mmm" "m"
##  [6,] NA    NA
##  [7,] NA    NA
##  [8,] NA    NA
##  [9,] NA    NA
## [10,] NA    NA
## [11,] NA    NA
```

We can see here both "aaa" and "mmmm" have at least 3 characters in a row that are the same.

Expression 2

```
"(.)(.)\\2\\1"
```

This expression would find any two characters that are then repeated in reverse order of the original character pair.

```
str_subset(q3_words, "(.)(.)\\2\\1")
```

```
## [1] "mmmm"         "babba"        "TATTARRATTAT"
```

```
str_match(q3_words, "(.)(.)\\2\\1")
```

```
##       [,1]   [,2] [,3]
##  [1,] NA     NA   NA
##  [2,] NA     NA   NA
##  [3,] NA     NA   NA
##  [4,] NA     NA   NA
##  [5,] "mmmm" "m"  "m"
##  [6,] "abba" "a"  "b"
##  [7,] NA     NA   NA
##  [8,] NA     NA   NA
##  [9,] NA     NA   NA
## [10,] NA     NA   NA
## [11,] "ATTA" "A"  "T"
```

We can see in this example that "mmmm" and "babba" both work here, however "babba" only works because "abba" is part of the string.

Expression 3

`(..)\1`

This expression is similar to expression 1 where it would return nothing due to the backslashes. If reformatted to `(..)\\1` though, it would find any two characters that are then followed by the same two characters.

```
str_subset(q3_words, "(..)\1")
```

```
## character(0)
```

```
str_match(q3_words, "(..)\1")
```

```
##       [,1] [,2]
##  [1,] NA   NA
##  [2,] NA   NA
##  [3,] NA   NA
##  [4,] NA   NA
##  [5,] NA   NA
##  [6,] NA   NA
##  [7,] NA   NA
##  [8,] NA   NA
##  [9,] NA   NA
## [10,] NA   NA
## [11,] NA   NA
```

```
str_subset(q3_words, "(..)\\1")
```

```
## [1] "mmmm"   "baba"   "banana"
```

```
str_match(q3_words, "(..)\\1")
```

```
##       [,1]   [,2]
##  [1,] NA     NA
##  [2,] NA     NA
##  [3,] NA     NA
##  [4,] NA     NA
##  [5,] "mmmm" "mm"
##  [6,] NA     NA
##  [7,] "baba" "ba"
##  [8,] "anan" "an"
##  [9,] NA     NA
## [10,] NA     NA
## [11,] NA     NA
```

We can see that the original expression did not find anything, however the adjusted one locates “mmmm” and “baba” as strings that follow the expression.

Expression 4

`"(.).\\1.\\1"`

This expression would find strings that have one character followed by any character, then the first character, any character, then the first one again.

```
str_subset(q3_words, "(.).\\1.\\1")
```

```
## [1] "banana"  "abacada"
```

```
str_match(q3_words, "(.).\\1.\\1")
```

```
##       [,1]    [,2]
##  [1,] NA      NA
##  [2,] NA      NA
##  [3,] NA      NA
##  [4,] NA      NA
##  [5,] NA      NA
##  [6,] NA      NA
##  [7,] NA      NA
##  [8,] "anana" "a"
##  [9,] "abaca" "a"
## [10,] NA      NA
## [11,] NA      NA
```

Here, we see that "banana" follows this expression as "anana" is a substring of this word. "abacada" also works as it's the character "a" switching off with other letters.

Expression 5

`"(.)(.)(.).*\\3\\2\\1"`

This expression would return find characters that have character 1, character 2, character 3, then either no or any number of characters, then the same three characters as before in reverse order.

```
str_subset(q3_words, "(.)(.)(.).*\\3\\2\\1")
```

```
## [1] "racecar"      "TATTARRATTAT"
```

```
str_match(q3_words, "(.)(.)(.).*\\3\\2\\1")
```

```
##       [,1]           [,2] [,3] [,4]
##  [1,] NA             NA   NA   NA
##  [2,] NA             NA   NA   NA
##  [3,] NA             NA   NA   NA
##  [4,] NA             NA   NA   NA
##  [5,] NA             NA   NA   NA
##  [6,] NA             NA   NA   NA
##  [7,] NA             NA   NA   NA
##  [8,] NA             NA   NA   NA
##  [9,] NA             NA   NA   NA
## [10,] "racecar"      "r"  "a"  "c"
## [11,] "TATTARRATTAT" "T"  "A"  "T"
```

We can see that words such as palindromes (words that are the same spelled forwards as backwards) fall in this category.

Question 4

Construct regular expressions to match words that:

Expression 1

Start and end with the same character.

`^(.).*\\1$`

```
q4_words <- c("bob", "church", "eleven", "momo")
str_subset(q4_words, "^(.).*\\1$")
```

```
## [1] "bob"
```

```
str_match(q4_words, "^(.).*\\1$")
```

```
##      [,1]  [,2]
## [1,] "bob" "b"
## [2,] NA    NA
## [3,] NA    NA
## [4,] NA    NA
```

Expression 2

Contain a repeated pair of letters (e.g. “church” contains “ch” repeated twice.)

`(..).*\\1`

```
str_subset(q4_words, "(..).*\\1")
```

```
## [1] "church" "momo"
```

```
str_match(q4_words, "(..).*\\1")
```

```
##      [,1]     [,2]
## [1,] NA       NA
## [2,] "church" "ch"
## [3,] NA       NA
## [4,] "momo"   "mo"
```

Expression 3

Contain one letter repeated in at least three places (e.g. “eleven” contains three “e”s.)

`(.).+\\1.+\\1.*`

```
str_subset(q4_words, "(.).+\\1.+\\1.+")
```

```
## [1] "eleven"
```

```
str_match(q4_words, "(.).+\\1.+\\1.+")
```

```
##      [,1]     [,2]
## [1,] NA       NA
## [2,] NA       NA
## [3,] "eleven" "e"
## [4,] NA       NA
```