R Character Manipulation and Date Processing

Jawaid Hakim

2022-09-19

Contents

1 Assignment

1.1 Search for majors containing "DATA" or "STATISTICS" - 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"

Load data from GitHub *raw* file.

```
ds <- read.csv("https://raw.githubusercontent.com/fivethirtyeight/data/2d2ff3e9457549d51f8e571c52099bfe9
```

Let's take a look at the unique values of *Major* and *Major_Category* columns. There are 174 unique majors and 16 major categories.

```
glimpse(as.factor(ds$Major))
```

```
##  Factor w/ 174 levels "ACCOUNTING","ACTUARIAL SCIENCE",..: 70 6 5 7 67 151 164 117 68 132 ...
```

```
glimpse(as.factor(ds$Major_Category))
```

```
##  Factor w/ 16 levels "Agriculture & Natural Resources",..: 1 1 1 1 1 1 1 1 1 1 ...
```

Let's assume the search terms may appear in either Major or Major Category columns and we want to consider both.

Filter data using *grepl* regular expressions, one per column. The search terms are found in 3 majors:

- COMPUTER PROGRAMMING AND DATA PROCESSING
- MANAGEMENT INFORMATION SYSTEMS AND STATISTICS
- STATISTICS AND DECISION SCIENCE

```
ds %>%
    filter(
            grepl("DATA|STATISTICS", ignore.case = TRUE, Major) |
            grepl("DATA|STATISTICS", ignore.case = TRUE, Major_Category)) %>%
    arrange(Major)
```

```
##   FOD1P                                         Major          Major_Category
## 1  2101      COMPUTER PROGRAMMING AND DATA PROCESSING Computers & Mathematics
## 2  6212 MANAGEMENT INFORMATION SYSTEMS AND STATISTICS                Business
## 3  3702               STATISTICS AND DECISION SCIENCE Computers & Mathematics
```

1.2 Transform data - 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”)

First we generate the input data as a string.

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

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

Extract fruit names from the input data by removing square brackets and digits and trimming.

```
s <- str_trim(str_remove_all(input_data, "[\n\\[[0-9]+\\]]"))
s
```

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

Replace space delimiters with commas.

```
s <- str_replace_all(s, '\\"[ ]+\\"', '\\", \\"')
s
```

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

Finally,generate the output string.

```
s1 <- noquote(s)              # remove quotes
out_str <- 'c('               # init output
for (s in s1[[1]]) {
     out_str <- paste(out_str, sprintf(', %s', s), sep = '')
}
out_str <- paste(out_str, ")", sep = '')           # add closing paren
out_str <- str_replace(out_str, "\\(, ", "\\(")    # remove extra leading comma
out_str <- noquote(out_str)
out_str
```

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

1.3 Describe Data - 3

1. (.)\1\1

I mistakenly thought this regex would match strings where the first character was repeated twice. The professor pointed out that unlike regex in other languages, the single backslash escape character is literally the character \001. As we can see, this regex does not match aaa but does match a\1\1. A single match group is recognized.

```
str_match('aaa', '(.)\1\1')
```

```
##      [,1] [,2]
## [1,] NA   NA
```

```
str_match('a\1\1', '(.)\1\1')
```

```
##      [,1]       [,2]
## [1,] "a\001\001" "a"
```

1. (.)(.)\\2\\1

Matches string of length 4: any 2 characters followed by same characters in reverse order.

```
str_match('abba', '(.)(.)\\2\\1')
```

```
##      [,1]   [,2] [,3]
## [1,] "abba" "a"  "b"
```

1. (..)\1 :

Matches strings of length 3: any 2 characters followed by the character \001. A single match group - of length 2 - is recognized.

```
str_match('ab\1', '(..)\1')
```

```
##      [,1]     [,2]
## [1,] "ab\001" "ab"
```

1. (.).\\1.\\1 :

Matches strings of length 5: any 2 characters, followed by the first character, followed by any character, followed by the first character.

```
str_match('abada',  '(.).\\1.\\1')
```

```
##      [,1]    [,2]
## [1,] "abada" "a"
```

1. (.)(.)(.).*\\3\\2\\1 :

Matches any 3 characters, followed by 0 or more characters, followed by the first 3 characters in reverse order.

```
str_match('abc123456cba',  '(.)(.)(.).*\\3\\2\\1')
```

```
##      [,1]           [,2] [,3] [,4]
## [1,] "abc123456cba" "a"  "b"  "c"
```

1.4 Construct regular expressions to match words - 4

1. Start and end with same character: ^(\\w)\\w*\\1$

```
str_match('ecommerce', '^(\\w)\\w*\\1$')
```

```
##      [,1]        [,2]
## [1,] "ecommerce" "e"
```

1. Contain a repeated pair of letters (e.g. "church" contains "ch" repeated twice.): ^\\w*(\\w)(\\w).*\\1\\2\\w*$

```
str_match('church', '^\\w*(\\w)(\\w).*\\1\\2\\w*$')
```

```
##      [,1]     [,2] [,3]
## [1,] "church" "c"  "h"
```

1. Contain one letter repeated in at least three places (e.g. "eleven" contains three "e"s.): ^\\w*(\\w)\\w*\\1\\w*\\1\\w*$

```
str_match('eleven', '^\\w*(\\w)\\w*\\1\\w*\\1\\w*$')
```

```
##      [,1]     [,2]
## [1,] "eleven" "e"
```