R Character Manipulation and Date Processing

Jawaid Hakim

2022-09-14

Contents


```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.3.6     v purrr   0.3.4
## v tibble  3.1.8     v dplyr   1.0.9
## v tidyr   1.2.0     v stringr 1.4.1
## v readr   2.1.2     v forcats 0.5.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

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/2d2ff3e9457549d51f8e571c52099bfe
```

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 using *str_extract_all* and flatten the result into a vector using *unlist*.

```
input_split <- unlist(str_extract_all(input_data, '"[a-zA-Z ]+"'))
input_split
```

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

All that remains to be done is removing those pesky double quotes using *str_remove*. And we are finished!

```
str_remove_all(input_split, '"')
```

```
##  [1] "bell pepper"  "bilberry"     "blackberry"   "blood orange" "blueberry"
##  [6] "cantaloupe"   "chili pepper" "cloudberry"   "elderberry"   "lime"
## [11] "lychee"       "mulberry"     "olive"        "salal berry"
```

Of course, we can *pipe* the two transformations for brevity and still get same results.

```
unlist(str_extract_all(input_data, '"[a-zA-Z ]+"')) %>% str_remove_all('"')
```

```
##  [1] "bell pepper"  "bilberry"     "blackberry"   "blood orange" "blueberry"
##  [6] "cantaloupe"   "chili pepper" "cloudberry"   "elderberry"   "lime"
## [11] "lychee"       "mulberry"     "olive"        "salal berry"
```

1.3 Describe Data - 3

1. (.)\1\1 : matches strings of length 3 where all characters are identical. Example, aaa, BBB, 111, etc.
2. (.)(.)\2\1 : *palindromes* of length 4. For example, abba, 1221, etc.
3. (..)\1 : matches strings of length 4 where the first 2 characters are repeated. For example, abab, 1212.
4. (.).\1.\1 : matches strings of length 5 where the 1^{st}, 3^{rd}, 5^{th} characters are the same. Example, abaka, 12141, etc.
5. (.)(.)(.).*\3\2\1 : matches strings of length 6 or greater where the first 3 characters are repeated at the end of the string in reverse order. For example, abccba, abc123cba.

1.4 Construct regular expressions to match words - 4

1. Start and end with same character: (.).*\1
2. Contain a repeated pair of letters (e.g. “church” contains “ch” repeated twice.): .*([a-zA-Z])([a-zA-Z]).*\1\2.*
3. Contain one letter repeated in at least three places (e.g. “eleven” contains three “e”s.): .*([a-zA-Z]).*\1.*\1.*