Lab 5: Working with Text and Strings

Alex Ptacek

Overview

In this lab you will practice perform a series of exercises that use text and string manipulation to either analyze data with text, manipulate data containing strings, apply regular expressions, or handle data files with unusual formats or text strings.

Problems

Problem 1. Using the 173 majors listed in fivethirtyeight.com's College Majors dataset, provide code that identifies the majors that contain either "DATA" or "STATISTICS", case insensitive. You can find this dataset on R by installing the package `fivethirtyeight` and using the `major` column in either `college_recent_grades`, `college_new_grads`, or `college_all_ages`.

```
library(tidyverse)
library(fivethirtyeight)
```

```
college_df <- tibble(college_all_ages)

str_view(college_df$major, regex("data|statistics", ignore_case = TRUE))
```

```
 [20] | Computer Programming And <Data> Processing
 [93] | <Statistics> And Decision Science
[170] | Management Information Systems And <Statistics>
```

Problem 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")
```

As your starting point take the string defined in the following code chunk:

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

```
food <- regex(r"(
                      [a-z]+   #any word
                      \ ?      #optional space
                      [a-z]*   #optional second word
                       )",
                      comments = TRUE)

str_extract_all(messyString, food, simplify = TRUE)
```

```
     [,1]          [,2]       [,3]         [,4]           [,5]
[1,] "bell pepper" "bilberry" "blackberry" "blood orange" "blueberry"
     [,6]         [,7]           [,8]         [,9]         [,10]  [,11]
[1,] "cantaloupe" "chili pepper" "cloudberry" "elderberry" "lime" "lychee"
     [,12]      [,13]   [,14]
[1,] "mulberry" "olive" "salal berry"
```

Hint: There are many different ways to solve this problem, but if you use `str_extract_all` a helpful flag that returns a character vector instead of a list is `simplify=TRUE`. Then you can apply other tools from `stringr` if needed.

Problem 3 Describe, in words, what these regular expressions will match. Read carefully to see if each entry is a regular expression or a string that defines a regular expression.

- `^.*$`
- `"\\{.+\\}"`
- `\d{4}-\d{2}-\d{2}`

- `"\\\\{4}"`
- `"(..)\\1"`

1. Matches a string that starts and ends with any amount (0-1+) of any character.
2. Matches a string that has an opening brace, followed by at least one of any character, then has a closing brace.
3. Matches a string that has a 4 digit number, then a dash, then a two digit number, then a dash, then a 2 digit number.
4. Matches a string that has 4 backslashes in a row.
5. Matches a string that repeats a pair of any characters.

Problem 4. Construct regular expressions to match words that:

- Start with "y". `"^y"`
- Have seven letters or more. [a-z] `".{7,}"`
- Contain a vowel-consonant pair `"[aeiou][^aeiou]"`
- Contain at least two vowel-consonant pairs in a row. `"([aeiou][^aeiou]){2,}"`
- Contain the same vowel-consonant pair repeated twice in a row. `"([aeiou][^aeiou])\\1"`

For each example, verify that they work by running them on the `stringr::words` dataset and show the first 10 results (hint: combine `str_detect` and logical subsetting).

```
words_df <- tibble(words)

#Start with "y'
filter(words_df, str_detect(words, "^y")) |>
  head(n = 10)
```

```
# A tibble: 6 x 1
  words
  <chr>
1 year
2 yes
3 yesterday
4 yet
5 you
6 young
```

```
#Have seven letters or more
filter(words_df, str_detect(words, ".{7,}")) |>
  head(n = 10)
```

```
# A tibble: 10 x 1
   words
   <chr>
 1 absolute
 2 account
 3 achieve
 4 address
 5 advertise
 6 afternoon
 7 against
 8 already
 9 alright
10 although
```

```
#Contain a vowel-consonant pair
filter(words_df, str_detect(words, "[aeiou][^aeiou]")) |>
  head(n = 10)
```

```
# A tibble: 10 x 1
   words
   <chr>
 1 able
 2 about
 3 absolute
 4 accept
 5 account
 6 achieve
 7 across
 8 act
 9 active
10 actual
```

```
#Contain at least two vowel-consonant pair in a row
filter(words_df, str_detect(words, "([aeiou][^aeiou]){2,}")) |>
  head(n = 10)
```

```
# A tibble: 10 x 1
   words
   <chr>
 1 absolute
 2 agent
```

```
 3 along
 4 america
 5 another
 6 apart
 7 apparent
 8 authority
 9 available
10 aware
```

```
#Contain the same vowel-consonant pair repeated twice in a row
filter(words_df, str_detect(words, "([aeiou][^aeiou])\\1")) |>
  head(n = 10)
```

```
# A tibble: 1 x 1
  words
  <chr>
1 remember
```

Problem 5 Consider the `gss_cat` data-frame discussed in Chapter 16 of R4DS (provided as part of the `forcats` package):

- Create a new variable that describes whether the party-id of a survey respondent is "strong" if they are a strong republican or strong democrat, "weak" if they are a not strong democrat, not strong republican, or independent of any type, and "other" for the rest.

```
gss_cat |>
  group_by(partyid) |>
  count() |> arrange(desc(n))
```

```
# A tibble: 10 x 2
# Groups:   partyid [10]
   partyid               n
   <fct>             <int>
 1 Independent        4119
 2 Not str democrat   3690
 3 Strong democrat    3490
 4 Not str republican 3032
 5 Ind,near dem       2499
 6 Strong republican  2314
 7 Ind,near rep       1791
```

```
 8 Other party          393
 9 No answer            154
10 Don't know             1
```

```
strong = c("Strong democrat", "Strong republican")
weak = c("Independent", "Not str democrat", "Not str republican",
         "Ind,near dem", "Ind,near rep")

gss_cat <- gss_cat |>
  mutate(sentiment = fct_collapse(partyid,
                                  "strong" = strong,
                                  "weak" = weak,
                                  other_level = "other"))
```

- Calculate the mean hours of TV watched by each of the groups "strong", "weak", and "other" and display it with a dot-plot (geom_point). Sort the levels in the dot-plot so that the group appears in order of most mean TV hours watched.

```
gss_cat |>
  group_by(sentiment) |>
  summarise(mean = mean(tvhours, na.rm = TRUE)) |>
  ggplot(aes(x = mean, y = fct_reorder(sentiment, mean))) +
  geom_point(size = 4, color = 'salmon')
```