Data 607 - Homework 3

Cameron Smith

2020-09-10

```
library(stringr)
```

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"

The below code loads data driectly from fivethirtyeight's Github repository. For redundancy a backup file is located in my Github at the following link: [https://raw.githubusercontent.com/cwestsmith/cuny-msds/master/datasets/majors-list.csv]

```
# Load data
df_orig <- read.csv("https://raw.githubusercontent.com/fivethirtyeight/data/master/college-majors/majors

# Filter data
df_orig$Major %>% str_subset("DATA|STATISTICS")
```

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

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")

The below code transforms the data and verifies the result:

```
# First create a list based on the 'clean' data
clean_data <- c("bell pepper", "bilberry", "blackberry", "blood orange", "blueberry", "cantaloupe", "chi

# Create a second list based on the raw / untransformed data
untransformed_data <- ’[1] "bell pepper"  "bilberry"     "blackberry"   "blood orange"

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

  [9] "elderberry"   "lime"         "lychee"       "mulberry"
  [13] "olive"        "salal berry"’

# Use regex to transform the data into a format identical to the original clean data
transformed_data <- unlist(str_extract_all(untransformed_data,’[[:alpha:]]+\\s[[:alpha:]]+|[[:alpha:]]+

# View transformed data
transformed_data
```

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

```
# Compare the clean data and the transformed data
identical(clean_data, transformed_data)
```

```
## [1] TRUE
```

The two exercises below are taken from R for Data Science, 14.3.5.1 in the on-line version:

Question 3

Describe, in words, what these expressions will match:

Part 1: (.)\1\1

This expression will search for any character, followed by that same character two more times. It should be noted however that when converted to a string for testing in R an additional backslash needs to be added to indicate the grouping to be used, otherwise it would evaluate simply as a character followed by a literal backslash followed by a one followed by a literal backslash followed by another one.

```
testdata <- c("a\1\1","z\1", "aaa", ’"abba"’, "abab",’"abaza"’,’"abcrandomtextcba"’, "abcrandomtextcba"

# Illustration showing incorrect regex format
str_subset(testdata,"(.)\1\1")
```

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

```
# Illustration showing correct regex format
str_subset(testdata,"(.)\\1\\1")
```

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

Part 2: “(.)(.)\2\1”

If evaluated using literal quotes, as indicated in the rubric, then this would be evaluated as a quote, followed by any two characters, followed by the second character again, followed by the first character again, followed by another literal quote (or in other words, 2 characters followed by the same two characters in reversed order, wrapped in quotes). The double backslashes serve as escape characters so that the last two parts are evaluated as groupings rather than literal characters. See example in code block above for what would happen otherwise.

```
str_subset(testdata,’"(.)(.)\\2\\1"’)
```

```
## [1] "\"abba\""
```

Part 3: (..)\1

This expression will search for any two characters, followed by those same two characters again. Similar to in part 1, it should be noticed that an extra backslash needed to be added to the expression when using it as a string in R to avoid the grouping being interpreted as a literal backslash, and a literal 1.

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

```
## [1] "abab"
```

Part 4: “(.).\1.\1”

If evaluated using literal quotes, as indicated in the rubric, then this would be evaluated as a quote, followed by any character, followed by any character, followed by the first character again, followed by any character, followed by the first character again, followed by another literal quote.

```
str_subset(testdata,’"(.).\\1.\\1"’)
```

```
## [1] "\"abaza\""
```

Part 5: "(.)(.)(.).*\3\2\1"

If evaluated using literal quotes, as indicated in the rubric, then this would be evaluated as a quote, followed by any 3 characters, followed by a set of 0 or more characters, followed by the first 3 characters again in reversed order, followed by another literal quote.

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

```
## [1] "\"abcrandomtextcba\""
```

Question 4

Construct regular expressions to match words that:

Start and end with the same character. Contain a repeated pair of letters (e.g. “church” contains “ch” repeated twice.) Contain one letter repeated in at least three places (e.g. “eleven” contains three “e”s.)

This question is answered within the comments of the code block below:

```
testdata<-c("abba","aba","abc","church", "eleven")

# Solution to part 1
# Start and end with the same character
#str_subset(testdata,"^(.)((.*\\1$)|\\1?$)")
str_subset(testdata,"^(.).*\\1$")
```

```
## [1] "abba" "aba"
```

```
# Solution to part 2
# Contain a repeated pair of letters
#str_subset(testdata,"([A-Za-z][A-Za-z]).*\\1")
str_subset(testdata,"([A-Za-z][A-Za-z]).*\\1")
```

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

```
# Solution to part 3
# Contain one letter repeated in at least three places
str_subset(testdata,"([a-z]).*\\1.*\\1.*")
```

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

-End of Homework 3-