Lab-R Character Manipulation

Michael Robinson

2023-09-24

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"

```
library(readr)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
# Reading data from website.
url <- "https://raw.githubusercontent.com/fivethirtyeight/data/master/college-majors/recent-grads.csv"

# Load the dataset from the url
college_majors <- read_csv(url)
```

```
## Rows: 173 Columns: 21
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (2): Major, Major_category
## dbl (19): Rank, Major_code, Total, Men, Women, ShareWomen, Sample_size, Empl...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
# Filter majors that contain "DATA" or "STATISTICS"
filtered_majors <- college_majors %>%
  filter(grepl("DATA|STATISTICS", Major, ignore.case = TRUE))

# Show the majors that meet the specified criteria.
filtered_majors
```

```
## # A tibble: 3 x 21
##    Rank Major_code Major Total   Men Women Major_category ShareWomen Sample_size
##   <dbl>      <dbl> <chr> <dbl> <dbl> <dbl> <chr>               <dbl>       <dbl>
## 1    25       6212 MANA~ 18713 13496  5217 Business            0.279         278
## 2    47       3702 STAT~  6251  2960  3291 Computers & M~      0.526          37
## 3    54       2101 COMP~  4168  3046  1122 Computers & M~      0.269          43
## # i 12 more variables: Employed <dbl>, Full_time <dbl>, Part_time <dbl>,
## #   Full_time_year_round <dbl>, Unemployed <dbl>, Unemployment_rate <dbl>,
## #   Median <dbl>, P25th <dbl>, P75th <dbl>, College_jobs <dbl>,
## #   Non_college_jobs <dbl>, Low_wage_jobs <dbl>
```

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”)

```r
# Create a string vector
fruit_data <- c('[1] "bell pepper"  "bilberry"     "blackberry"   "blood orange"
[5] "blueberry"    "cantaloupe"   "chili pepper" "cloudberry"
[9] "elderberry"   "lime"         "lychee"       "mulberry"
[13] "olive"        "salal berry"')

# Eliminate line identifiers and square brackets.
new_fruit_vector <- gsub("\\[\\d+\\] |\\n", "", fruit_data)

# Segment the string using double quotes as separators.
new_fruit_vector <- unlist(strsplit(new_fruit_vector, '\\"'))

# Filtering for only letters
new_fruit_vector <- new_fruit_vector[grep("[a-z]", new_fruit_vector)]

# Print the result
print(new_fruit_vector)
```

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

```
comp_data <- c("bell pepper", "bilberry", "blackberry", "blood orange", "blueberry", "cantaloupe", "chi
# Print the result

comp_data
```

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

3. Describe, in words, what these expressions will match:

1. `(.)\1\1`: Matches three consecutive characters that are all the same.
2. `"(.)(.)\\2\\1"`: Matches any four character string where the first and last characters are the same, and the middle two characters are also the same.
3. `(..)\1`: Matches any four character string where the first two characters are the same as the last two characters.
4. `"(.).\\1.\\1"`: Matches any five character string that the first and the third character are the same, and the third and fifth character are the same.
5. `"(.)(.)(.).*\\3\\2\\1"`: Matches a string that begins with three characters in any order, followed by any number of characters, and ends with those same three characters in reverse order.

4. Construct regular expressions to match words that:

a.) Start and end with the same character. "ˆ(.).\1$" *b.) Contain a repeated pair of letters (e.g. "church" contains "ch" repeated twice.)* ".(.)\1.\1." c.) Contain one letter repeated in at least three places (e.g. "eleven" contains three "e"s.) ".(.)\1\1\1."