STAT570 HW2

Miray Çınar

2023-11-20

15.4.6 Grouping and capturing

First install the required packages and then check the data:

```
# install.packages("tidyverse")
# install.packages("babynames")

library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.4.4     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.0
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(babynames)

head(fruit)
```

```
## [1] "apple"       "apricot"     "avocado"     "banana"      "bell pepper"
## [6] "bilberry"
```

```
head(words)
```

```
## [1] "a"        "able"     "about"    "absolute" "accept"   "account"
```

```
head(sentences)
```

```
## [1] "The birch canoe slid on the smooth planks."
## [2] "Glue the sheet to the dark blue background."
## [3] "It's easy to tell the depth of a well."
## [4] "These days a chicken leg is a rare dish."
## [5] "Rice is often served in round bowls."
## [6] "The juice of lemons makes fine punch."
```

Grouping: In regular expressions, parentheses () are used to create groups. These groups serve multiple purposes:

Logical Grouping: They help define a section of the pattern to be treated as a single unit for applying quantifiers or alternation (|). Capturing Groups: They capture the part of the string that matches the pattern within the parentheses. This captured content can be referenced later using backreferences. Capturing and Back References: The capturing groups allow you to retrieve or reference the matched substrings within the same regex pattern.

Backreferences: These references, like \1, \2, etc., refer back to the contents of the corresponding capturing group within the same regex pattern. For instance, \1 refers to the first capturing group's match, \2 refers to the second one, and so on.

For example, the following pattern finds all fruits that have a repeated pair of letters in it:

```
str_view(fruit, "(..)\\1")
```

```
##  [4] | b<anan>a
## [20] | <coco>nut
## [22] | <cucu>mber
## [41] | <juju>be
## [56] | <papa>ya
## [73] | s<alal> berry
```

Extra example:

In this one we see that it search for pattern that fruits that have the same consecutive letters:

```
str_view(fruit, "(.)\\1")
```

```
##  [1] | a<pp>le
##  [5] | be<ll> pe<pp>er
##  [6] | bilbe<rr>y
##  [7] | blackbe<rr>y
##  [8] | blackcu<rr>ant
##  [9] | bl<oo>d orange
## [10] | bluebe<rr>y
## [11] | boysenbe<rr>y
## [16] | che<rr>y
## [17] | chili pe<pp>er
## [19] | cloudbe<rr>y
## [21] | cranbe<rr>y
## [23] | cu<rr>ant
## [28] | e<gg>plant
## [29] | elderbe<rr>y
## [32] | goji be<rr>y
## [33] | g<oo>sebe<rr>y
## [38] | hucklebe<rr>y
## [47] | lych<ee>
## [50] | mulbe<rr>y
## ... and 9 more
```

This regex pattern finds the words that start and end with the same pair of letters.

```
str_view(words, "^(..).*\\1$")
```

```
## [152] | <church>
## [217] | <decide>
## [617] | <photograph>
## [699] | <require>
## [739] | <sense>
```

So, how does this work?

^(..): The pattern starts by matching any two characters at the beginning of the word and captures them in the first group.

.*: This matches any number of characters in between the start and end pairs.

\1$: Ensures that the string ends with the same pair of characters that were captured at the beginning.

Extra example:

```
str_view(fruit, "^(..).*\\1$")
```

```
## [51] | <nectarine>
```

We can also use back references in str_replace().

First look into the dataset:

```
head(sentences)
```

```
## [1] "The birch canoe slid on the smooth planks."
## [2] "Glue the sheet to the dark blue background."
## [3] "It's easy to tell the depth of a well."
## [4] "These days a chicken leg is a rare dish."
## [5] "Rice is often served in round bowls."
## [6] "The juice of lemons makes fine punch."
```

```
sentences |>
  str_replace("(\\w+) (\\w+) (\\w+)", "\\1 \\3 \\2") |>
  str_view()
```

```
##  [1] | The canoe birch slid on the smooth planks.
##  [2] | Glue sheet the to the dark blue background.
##  [3] | It's to easy tell the depth of a well.
##  [4] | These a days chicken leg is a rare dish.
##  [5] | Rice often is served in round bowls.
##  [6] | The of juice lemons makes fine punch.
##  [7] | The was box thrown beside the parked truck.
##  [8] | The were hogs fed chopped corn and garbage.
##  [9] | Four of hours steady work faced us.
## [10] | A size large in stockings is hard to sell.
## [11] | The was boy there when the sun rose.
## [12] | A is rod used to catch pink salmon.
## [13] | The of source the huge river is the clear spring.
```

```
## [14] | Kick ball the straight and follow through.
## [15] | Help woman the get back to her feet.
## [16] | A of pot tea helps to pass the evening.
## [17] | Smoky lack fires flame and heat.
## [18] | The cushion soft broke the man's fall.
## [19] | The breeze salt came across from the sea.
## [20] | The at girl the booth sold fifty bonds.
## ... and 700 more
```

The output would show the sentences where the second and third words have swapped positions, effectively altering the sentence structure while maintaining the first word unchanged.

The regex pattern "(\w+) (\w+) (\w+)" is used to match three words in a sentence. Each (\w+) captures a word (a sequence of alphanumeric characters).

"\1 \3 \2": This is the replacement pattern. \1, \2, and \3 refer to the first, second, and third captured groups from the regex pattern. By rearranging the order of the back references (\1, \3, \2), it switches the positions of the second and third words while keeping the first word unchanged.

Extra example:

```
tail(sentences)
```

```
## [1] "Small children came to see him."
## [2] "The grass and bushes were wet with dew."
## [3] "The blind man counted his old coins."
## [4] "A severe storm tore down the barn."
## [5] "She called his name many times."
## [6] "When you hear the bell, come quickly."
```

```
exsentences <- c("Small children came to see him.", "The grass and bushes were wet with dew.","The blin
```

```
switched_sentences <- exsentences %>%
  str_replace("(\\w+) (\\w+) (\\w+)", "\\1 \\3 \\2")

switched_sentences
```

```
## [1] "Small came children to see him."
## [2] "The and grass bushes were wet with dew."
## [3] "The man blind counted his old coins."
```

If you want to extract the matches for each group you can use str_match(). But str_match() returns a matrix, so it's not particularly easy to work with8:

```
sentences |>
  str_match("the (\\w+) (\\w+)") |>
  head()
```

```
##      [,1]                [,2]     [,3]
## [1,] "the smooth planks" "smooth" "planks"
## [2,] "the sheet to"      "sheet"  "to"
## [3,] "the depth of"      "depth"  "of"
## [4,] NA                  NA       NA
## [5,] NA                  NA       NA
## [6,] NA                  NA       NA
```

You could convert to a tibble and name the columns:

```
sentences |>
  str_match("the (\\w+) (\\w+)") |>
  as_tibble(.name_repair = "minimal") |>
  set_names("match", "word1", "word2")
```

```
## # A tibble: 720 x 3
##    match             word1  word2
##    <chr>             <chr>  <chr>
##  1 the smooth planks smooth planks
##  2 the sheet to      sheet  to
##  3 the depth of      depth  of
##  4 <NA>              <NA>   <NA>
##  5 <NA>              <NA>   <NA>
##  6 <NA>              <NA>   <NA>
##  7 the parked truck  parked truck
##  8 <NA>              <NA>   <NA>
##  9 <NA>              <NA>   <NA>
## 10 <NA>              <NA>   <NA>
## # i 710 more rows
```

str_match() function is applied to each sentence in the dataset based on the provided regex pattern. as_tibble(.name_repair = "minimal"):

The output of str_match() (which is a matrix) is converted to a tibble (data frame). .name_repair = "minimal" is an argument used in as_tibble() to control the way column names are repaired in case they are invalid or duplicated. set_names("match", "word1", "word2"):

The column names of the resulting tibble are explicitly set using set_names().

Finally we can use parantheses without creating matching groups.

```
x <- c("a gray cat", "a grey dog")
str_match(x, "gr(e|a)y")
```

```
##      [,1]   [,2]
## [1,] "gray" "a"
## [2,] "grey" "e"
```

The regex pattern matches "gr" followed by either "e" or "a" and then followed by "y". This pattern captures either "e" or "a" as part of the match.

Within the regex pattern, (e|a) is a capturing group. | represents an OR operator, allowing the pattern to match either "e" or "a". The overall pattern matches "grey" or "gray" where the letter between "gr" and "y" can be either "e" or "a".

Non-Capturing Group (?:):

The use of (?:) within the regex pattern creates a non-capturing group. In this case, the non-capturing group (?:e|a) matches either "e" or "a" without capturing this part of the match separately. It's used for grouping without creating a capture group explicitly.

```
str_match(x, "gr(?:e|a)y")
```

```
##      [,1]
## [1,] "gray"
## [2,] "grey"
```

The crucial aspect here is the (?:) used in the regex pattern. It signifies a non-capturing group, meaning the pattern inside it (e|a) will match either "e" or "a" but won't be captured as a separate group in the output.

Extra example:

```r
gardrobe <- c("tshirt is blue.", "skirt is yellow", "scarf is red")

# Search for matches
matches <- str_match(gardrobe, "(skirt|scarf) is (?:\\w+)")
matches
```

```
##      [,1]              [,2]
## [1,] NA                NA
## [2,] "skirt is yellow" "skirt"
## [3,] "scarf is red"    "scarf"
```

(skirt|scarf) captures either "skirt" or "scarf". is matches the word "is". (?:\w+) is a non-capturing group (?:) matching any additional word (\w+ represents one or more word characters). The non-capturing group (?:\w+) ensures that the additional word is matched without being captured separately.