Strings: HW

Ziling Zhen

11_strings_part2.qmd: Exercises #1-17

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.1     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.1
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 becom

Attaching package: 'rvest'


The following object is masked from 'package:readr':

    guess_encoding


Rows: 350 Columns: 24
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (10): track_id, title, artist, genre, album_id, album_name, album_releas...
dbl (14): popularity, danceability, valence, energy, key, loudness, mode, sp...

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.

# A tibble: 10 x 6
   title             artist album_release_date album_name subgenre playlist_name
```

```
   <chr>              <chr>  <chr>              <chr>      <chr>    <chr>
 1 Hear Me Now        Alok   2016-01-01         Hear Me N~ indie p~ Chillout & R~
 2 Run the World (G~  Beyon~ 2011-06-24         4          post-te~ post-teen al~
 3 Formation          Beyon~ 2016-04-23         Lemonade   hip pop  Feeling Acco~
 4 7/11               Beyon~ 2014-11-24         BEYONCÉ [~ hip pop  Feeling Acco~
 5 My Oh My (feat. ~  Camil~ 2019-12-06         Romance    latin p~ 2020 Hits & ~
 6 It's Automatic     Frees~ 2013-11-28         It's Auto~ latin h~ 80's Freesty~
 7 Poetic Justice     Kendr~ 2012               good kid,~ hip hop  Hip Hop Cont~
 8 A.D.H.D            Kendr~ 2011-07-02         Section.80 souther~ Hip-Hop 'n R~
 9 Ya Estuvo          Kid F~ 1990-01-01         Hispanic ~ latin h~ HIP-HOP: Lat~
10 Runnin (with A$A~  Mike ~ 2018-11-16         Creed II:~ gangste~ RAP Gangsta
```

1. Identify the input type and output type for each of these examples:

```
#1
str_view(spot_smaller$subgenre, "pop")
```

```
[1] | indie <pop>timism
[2] | post-teen <pop>
[3] | hip <pop>
[4] | hip <pop>
[5] | latin <pop>
```

```
typeof(str_view(spot_smaller$subgenre, "pop"))
```

```
[1] "character"
```

```
class(str_view(spot_smaller$subgenre, "pop"))
```

```
[1] "stringr_view"
```

```
#2
#str_view(spot_smaller$subgenre, "pop", match = NA)
#str_view(spot_smaller$subgenre, "pop", html = TRUE)

#3
str_subset(spot_smaller$subgenre, "pop")
```

```
[1] "indie poptimism" "post-teen pop"   "hip pop"         "hip pop"
[5] "latin pop"
```

```
#4
str_detect(spot_smaller$subgenre, "pop")
```

```
 [1]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE
```

1) input: a vector of 10 strings, output: view of the strings with "pop"
2) input: a vector of 10 strings, output: view of the strings with "pop" and the ones without
3) input: a vector of 10 strings, output: 5 vectors with "pop"
4) input: a vector of 10 strings, output: trues or falses for if our string contains "pop"

2. Use str_detect to print the rows of the spot_smaller tibble containing songs that have "pop" in the subgenre. (i.e. make a new tibble with fewer rows)

```
spot_even_smaller <- spot_smaller |>
  mutate(sub_pop = str_detect(subgenre, "pop"))

spot_even_smaller |>
  filter(sub_pop)
```

```
# A tibble: 5 x 7
  title      artist album_release_date album_name subgenre playlist_name sub_pop
  <chr>      <chr>  <chr>              <chr>      <chr>    <chr>         <lgl>
1 Hear Me N~ Alok   2016-01-01         Hear Me N~ indie p~ Chillout & R~ TRUE
2 Run the W~ Beyon~ 2011-06-24         4          post-te~ post-teen al~ TRUE
3 Formation  Beyon~ 2016-04-23         Lemonade   hip pop  Feeling Acco~ TRUE
4 7/11       Beyon~ 2014-11-24         BEYONCÉ [~ hip pop  Feeling Acco~ TRUE
5 My Oh My ~ Camil~ 2019-12-06         Romance    latin p~ 2020 Hits & ~ TRUE
```

3. Find the mean song title length for songs with "pop" in the subgenre and songs without "pop" in the subgenre.

```
spot_even_smaller |>
  mutate(title_length = str_length(title)) |>
  group_by(sub_pop) |>
  summarize(mean_title_length = mean(title_length))
```

```
# A tibble: 2 x 2
  sub_pop mean_title_length
  <lgl>               <dbl>
1 FALSE                18.6
2 TRUE                 13.6
```

```r
spot_even_smaller |>
  mutate(title_length = str_length(title)) |>
  group_by(sub_pop) |>
  summarize(mean_title_length = mean(title_length)) |>
  mutate(sub_pop = ifelse(sub_pop,"Genre with pop", "Genre without pop"))
```

```
# A tibble: 2 x 2
  sub_pop           mean_title_length
  <chr>                         <dbl>
1 Genre without pop              18.6
2 Genre with pop                 13.6
```

Producing a table like this would be great:

A tibble: 2 × 2

sub_pop mean_title_length 1 FALSE 18.6 2 TRUE 13.6

Producing a table like this would be SUPER great (hint: ifelse()):

A tibble: 2 × 2

sub_pop mean_title_length 1 Genre with pop 13.6 2 Genre without pop 18.6

4. In the bigspotify dataset, find the proportion of songs which contain "love" in the title (track_name) by playlist_genre.

```r
bigspotify <- readr::read_csv('https://raw.githubusercontent.com/rfordatascience/tidytuesday/
```

```
Rows: 32833 Columns: 23
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (10): track_id, track_name, track_artist, track_album_id, track_album_na...
dbl (13): track_popularity, danceability, energy, key, loudness, mode, speec...

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.
```

```
bigspotify
```

```
# A tibble: 32,833 x 23
   track_id              track_name track_artist track_popularity track_album_id
   <chr>                 <chr>      <chr>                   <dbl> <chr>
 1 6f807x0ima9a1j3VPbc7~ I Don't C~ Ed Sheeran                 66 2oCs0DGTsRO98~
 2 0r7CVbZTWZgbTCYdfa2P~ Memories ~ Maroon 5                   67 63rPSO264uRjW~
 3 1z1Hg7Vb0AhHDiEmnDE7~ All the T~ Zara Larsson               70 1HoSmj2eLcsrR~
 4 75FpbthrwQmzHlBJLuGd~ Call You ~ The Chainsm~              60 1nqYsOef1yKKu~
 5 1e8PAfcKUYoKkxPhrHqw~ Someone Y~ Lewis Capal~              69 7m7vv9wlQ4i0L~
 6 7fvUMiyapMsRRxr07cU8~ Beautiful~ Ed Sheeran                67 2yiy9cd2QktrN~
 7 2OAylPUDDfwRGfe0lYql~ Never Rea~ Katy Perry                62 7INHYSeusaFly~
 8 6b1RNvAcJjQH73eZO4BL~ Post Malo~ Sam Feldt                 69 6703SRPsLkS4b~
 9 7bF6tCO3gFb8INrEDcjN~ Tough Lov~ Avicii                    68 7CvAfGvq4RlIw~
10 1IXGILkPm0tOCNeq00kC~ If I Can'~ Shawn Mendes             67 4QxzbfSsVryEQ~
# i 32,823 more rows
# i 18 more variables: track_album_name <chr>, track_album_release_date <chr>,
#   playlist_name <chr>, playlist_id <chr>, playlist_genre <chr>,
#   playlist_subgenre <chr>, danceability <dbl>, energy <dbl>, key <dbl>,
#   loudness <dbl>, mode <dbl>, speechiness <dbl>, acousticness <dbl>,
#   instrumentalness <dbl>, liveness <dbl>, valence <dbl>, tempo <dbl>,
#   duration_ms <dbl>
```

```
bigspotify |>
  mutate(haslove = str_detect(track_name, "love")) |>
  group_by(playlist_genre) |>
  summarise(prop_love = mean(haslove, na.rm = TRUE))
```

```
# A tibble: 6 x 2
  playlist_genre prop_love
  <chr>              <dbl>
1 edm              0.00165
2 latin            0.000776
3 pop              0.00109
4 r&b              0.000552
5 rap              0.000348
6 rock             0.000808
```

Matching patterns with regular expressions

^abc string starts with abc abc$ string ends with abc . any character [abc] a or b or c [^abc] anything EXCEPT a or b or c

5. Given the corpus of common words in stringr::words, create regular expressions that find all words that:

- Start with "y".
- End with "x"
- Are exactly three letters long.
- Have seven letters or more.
- Start with a vowel.
- End with ed, but not with eed.
- Words where q is not followed by u. (are there any in `words`?)

```
# Try using str_view() or str_subset()

# For example, to find words with "tion" at any point, I could use:
#str_view(words, "tion")
str_subset(words, "tion")
```

```
 [1] "condition" "function"  "mention"   "motion"    "nation"    "position"
 [7] "question"  "relation"  "section"   "station"
```

```
#Start with "y"
str_subset(words, "^y")
```

```
[1] "year"      "yes"       "yesterday" "yet"       "you"       "young"
```

```
#End with "x"
str_subset(words, "x$")
```

```
[1] "box" "sex" "six" "tax"
```

```
#Are exactly three letters long
str_subset(words, "^[A-Za-z]{3}$")
```

```
  [1] "act" "add" "age" "ago" "air" "all" "and" "any" "arm" "art" "ask" "bad"
 [13] "bag" "bar" "bed" "bet" "big" "bit" "box" "boy" "bus" "but" "buy" "can"
 [25] "car" "cat" "cup" "cut" "dad" "day" "die" "dog" "dry" "due" "eat" "egg"
 [37] "end" "eye" "far" "few" "fit" "fly" "for" "fun" "gas" "get" "god" "guy"
 [49] "hit" "hot" "how" "job" "key" "kid" "lad" "law" "lay" "leg" "let" "lie"
 [61] "lot" "low" "man" "may" "mrs" "new" "non" "not" "now" "odd" "off" "old"
 [73] "one" "out" "own" "pay" "per" "put" "red" "rid" "run" "say" "see" "set"
 [85] "sex" "she" "sir" "sit" "six" "son" "sun" "tax" "tea" "ten" "the" "tie"
 [97] "too" "top" "try" "two" "use" "war" "way" "wee" "who" "why" "win" "yes"
[109] "yet" "you"
```

```
#OR
#str_subset(words, "^...$")

# Have seven letters or more.
str_subset(words, "^.......")
```

```
  [1] "absolute"    "account"     "achieve"     "address"     "advertise"
  [6] "afternoon"   "against"     "already"     "alright"     "although"
 [11] "america"     "another"     "apparent"    "appoint"     "approach"
 [16] "appropriate" "arrange"     "associate"   "authority"   "available"
 [21] "balance"     "because"     "believe"     "benefit"     "between"
 [26] "brilliant"   "britain"     "brother"     "business"    "certain"
 [31] "chairman"    "character"   "Christmas"   "colleague"   "collect"
 [36] "college"     "comment"     "committee"   "community"   "company"
 [41] "compare"     "complete"    "compute"     "concern"     "condition"
 [46] "consider"    "consult"     "contact"     "continue"    "contract"
 [51] "control"     "converse"    "correct"     "council"     "country"
 [56] "current"     "decision"    "definite"    "department"  "describe"
 [61] "develop"     "difference"  "difficult"   "discuss"     "district"
 [66] "document"    "economy"     "educate"     "electric"    "encourage"
 [71] "english"     "environment" "especial"    "evening"     "evidence"
 [76] "example"     "exercise"    "expense"     "experience"  "explain"
 [81] "express"     "finance"     "fortune"     "forward"     "function"
 [86] "further"     "general"     "germany"     "goodbye"     "history"
 [91] "holiday"     "hospital"    "however"     "hundred"     "husband"
 [96] "identify"    "imagine"     "important"   "improve"     "include"
[101] "increase"    "individual"  "industry"    "instead"     "interest"
[106] "introduce"   "involve"     "kitchen"     "language"    "machine"
[111] "meaning"     "measure"     "mention"     "million"     "minister"
[116] "morning"     "necessary"   "obvious"     "occasion"    "operate"
[121] "opportunity" "organize"    "original"    "otherwise"   "paragraph"
```

```
[126] "particular"  "pension"     "percent"     "perfect"     "perhaps"
[131] "photograph"  "picture"     "politic"     "position"    "positive"
[136] "possible"    "practise"    "prepare"     "present"     "pressure"
[141] "presume"     "previous"    "private"     "probable"    "problem"
[146] "proceed"     "process"     "produce"     "product"     "programme"
[151] "project"     "propose"     "protect"     "provide"     "purpose"
[156] "quality"     "quarter"     "question"    "realise"     "receive"
[161] "recognize"   "recommend"   "relation"    "remember"    "represent"
[166] "require"     "research"    "resource"    "respect"     "responsible"
[171] "saturday"    "science"     "scotland"    "secretary"   "section"
[176] "separate"    "serious"     "service"     "similar"     "situate"
[181] "society"     "special"     "specific"    "standard"    "station"
[186] "straight"    "strategy"    "structure"   "student"     "subject"
[191] "succeed"     "suggest"     "support"     "suppose"     "surprise"
[196] "telephone"   "television"  "terrible"    "therefore"   "thirteen"
[201] "thousand"    "through"     "thursday"    "together"    "tomorrow"
[206] "tonight"     "traffic"     "transport"   "trouble"     "tuesday"
[211] "understand"  "university"  "various"     "village"     "wednesday"
[216] "welcome"     "whether"     "without"     "yesterday"
```

```r
#OR
#str_subset(words, "^.{7}")

#Start with a vowel.
str_subset(words, "^[aeiou]")
```

```
  [1] "a"           "able"        "about"       "absolute"    "accept"
  [6] "account"     "achieve"     "across"      "act"         "active"
 [11] "actual"      "add"         "address"     "admit"       "advertise"
 [16] "affect"      "afford"      "after"       "afternoon"   "again"
 [21] "against"     "age"         "agent"       "ago"         "agree"
 [26] "air"         "all"         "allow"       "almost"      "along"
 [31] "already"     "alright"     "also"        "although"    "always"
 [36] "america"     "amount"      "and"         "another"     "answer"
 [41] "any"         "apart"       "apparent"    "appear"      "apply"
 [46] "appoint"     "approach"    "appropriate" "area"        "argue"
 [51] "arm"         "around"      "arrange"     "art"         "as"
 [56] "ask"         "associate"   "assume"      "at"          "attend"
 [61] "authority"   "available"   "aware"       "away"        "awful"
 [66] "each"        "early"       "east"        "easy"        "eat"
 [71] "economy"     "educate"     "effect"      "egg"         "eight"
 [76] "either"      "elect"       "electric"    "eleven"      "else"
```

```
 [81] "employ"      "encourage"   "end"         "engine"      "english"
 [86] "enjoy"       "enough"      "enter"       "environment" "equal"
 [91] "especial"    "europe"      "even"        "evening"     "ever"
 [96] "every"       "evidence"    "exact"       "example"     "except"
[101] "excuse"      "exercise"    "exist"       "expect"      "expense"
[106] "experience"  "explain"     "express"     "extra"       "eye"
[111] "idea"        "identify"    "if"          "imagine"     "important"
[116] "improve"     "in"          "include"     "income"      "increase"
[121] "indeed"      "individual"  "industry"    "inform"      "inside"
[126] "instead"     "insure"      "interest"    "into"        "introduce"
[131] "invest"      "involve"     "issue"       "it"          "item"
[136] "obvious"     "occasion"    "odd"         "of"          "off"
[141] "offer"       "office"      "often"       "okay"        "old"
[146] "on"          "once"        "one"         "only"        "open"
[151] "operate"     "opportunity" "oppose"      "or"          "order"
[156] "organize"    "original"    "other"       "otherwise"   "ought"
[161] "out"         "over"        "own"         "under"       "understand"
[166] "union"       "unit"        "unite"       "university"  "unless"
[171] "until"       "up"          "upon"        "use"         "usual"
```

```r
#End with ed, but not with eed
str_subset(words, "[^e]ed$")
```

```
[1] "bed"     "hundred" "red"
```

```r
#Words where q is not followed by u
str_subset(words, "q[^u]")
```

```
character(0)
```

More useful regular expressions:

\d - any number \s - any space, tab, etc \b - any boundary: space, ., etc.

Here are the regular expression special characters that require an escape character (a preceding): ^ $. ? * | + ( ) [{

For any characters with special properties, use to "escape" its special meaning … but is itself a special character … so we need two \! (e.g. \$, \., etc.)

```
#str_view(spot_smaller$title, "$")
#str_view(spot_smaller$title, "\\$")
```

6. In bigspotify, how many track_names include a $? Be sure you print the track_names you find and make sure the dollar sign is not just in a featured artist!

```
bigspotify |>
  filter(str_detect(track_name, "\\$")) |>
  #filter(!str_detect(track_name,"\\(.*\\$.*\\)")) |>
  filter(!str_detect(track_name, "(feat|with).*\\$")) |>
  select(track_name, track_artist) |>
  print(n = Inf)
```

```
# A tibble: 25 x 2
   track_name                                          track_artist
   <chr>                                               <chr>
 1 Wing$                                               Macklemore & Ryan Lewis
 2 $Dreams                                             Max Frost
 3 $ave Dat Money (feat. Fetty Wap & Rich Homie Quan) Lil Dicky
 4 NO TRU$T                                            NUGAT
 5 A$AP Forever                                        A$AP Rocky
 6 M'$ (feat. Lil Wayne)                               A$AP Rocky
 7 Sie wollen meine Loui$ (Don Dollar)                 Kulturerbe Achim
 8 Foe Tha Love Of $                                   Bone Thugs-N-Harmony
 9 A$AP                                                Dillom
10 $$$ - Remix                                         Saramalacara
11 Fre$h                                               Lil Whigga
12 $ENHOR                                              FBC
13 $20 Fine                                            Jimi Hendrix
14 A$IAN BOY                                           Chriilz
15 ¿Cuánto E$?                                         Jhay Cortez
16 $. A. N. T. E. R. Í. A.                             Doble Porcion
17 A$IAN BOY                                           Chriilz
18 Bernice Burgo$                                      Nino Khayyam
19 $100 (feat. Polo Donatello)                         Mibbs
20 M'$                                                 A$AP Rocky
21 Dat $tick                                           Rich Brian
22 $ave Dat Money (feat. Fetty Wap & Rich Homie Quan) Lil Dicky
23 Love$ick                                            Mura Masa
24 A$IAN BOY                                           Chriilz
25 CA$H                                                Olly James
```

7. In bigspotify, how many track_names include a dollar amount (a $ followed by a number).

```
bigspotify |>
  filter(str_detect(track_name, "\\$\\d")) |>
  filter(!str_detect(track_name, "(feat|with).*\\$")) |>
  select(track_name, track_artist) |>
  print(n = Inf)
```

```
# A tibble: 2 x 2
  track_name                  track_artist
  <chr>                       <chr>
1 $20 Fine                    Jimi Hendrix
2 $100 (feat. Polo Donatello) Mibbs
```

OR

```
bigspotify |>
    filter(str_detect(track_name, "\\$\\d")) |>
    select(track_name, track_artist)
```

```
# A tibble: 2 x 2
  track_name                  track_artist
  <chr>                       <chr>
1 $20 Fine                    Jimi Hendrix
2 $100 (feat. Polo Donatello) Mibbs
```

Repetition

? 0 or 1 times + 1 or more * 0 or more {n} exactly n times {n,} n or more times {,m} at most m times {n,m} between n and m times ? or

```
str_subset(spot_smaller$album_name, "[A-Z]{2,}")
```

```
[1] "BEYONCÉ [Platinum Edition]" "Creed II: The Album"
```

```
str_subset(spot_smaller$album_release_date, "\\d{4}-\\d{2}")
```

```
[1] "2016-01-01" "2011-06-24" "2016-04-23" "2014-11-24" "2019-12-06"
[6] "2013-11-28" "2011-07-02" "1990-01-01" "2018-11-16"
```

Use at least 1 repetition symbol when solving 8-10 below

8. Modify the first regular expression above to also pick up "m.A.A.d" (in addition to "BEYONC" and "II"). That is, pick up strings where there might be a period between capital letters.

```
str_subset(spot_smaller$album_name, "[A-Z?.A-Z]{2,}")
```

```
[1] "BEYONCÉ [Platinum Edition]"      "good kid, m.A.A.d city (Deluxe)"
[3] "Creed II: The Album"
```

```
str_subset(spot_smaller$album_name, "[A-Z?.]{2,}")
```

```
[1] "BEYONCÉ [Platinum Edition]"      "good kid, m.A.A.d city (Deluxe)"
[3] "Creed II: The Album"
```

9. Create some strings that satisfy these regular expressions and explain.

- "^.*$" this can be any expression
- "\{.+\}" this contains curly brackets with 1 or more things in it

```
str_detect("", "^.*$")
```

```
[1] TRUE
```

```
str_detect("m,y", "^.*$")
```

```
[1] TRUE
```

```
str_detect("97^7hf", "^.*$")
```

```
[1] TRUE
```

```
str_detect(")9nfud%{}", "^.*$")
```

```
[1] TRUE
```

```
str_detect("n 09", "^.*$")
```

```
[1] TRUE
```

```
str_detect("{tfvb0987}", "\\{.+\\}")
```

```
[1] TRUE
```

```
str_detect("{ziling zhen}", "\\{.+\\}")
```

```
[1] TRUE
```

```
str_detect("{Z}", "\\{.+\\}")
```

```
[1] TRUE
```

```
str_detect("{ }", "\\{.+\\}")
```

```
[1] TRUE
```

10. Create regular expressions to find all `stringr::words` that:

    - Start with three consonants.
    - Have two or more vowel-consonant pairs in a row.

```
#consonant
str_subset(words, "^[^aeiouy]{3,}")
```

```
 [1] "Christ"    "Christmas" "mrs"       "scheme"    "school"    "straight"
 [7] "strategy"  "street"    "strike"    "strong"    "structure" "three"
[13] "through"   "throw"
```

OR

```
#consonant
words |>
  as_tibble() |>
  filter(str_detect(value, "^[^aeiouy]{3,}"))
```

```
# A tibble: 14 x 1
   value
   <chr>
 1 Christ
 2 Christmas
 3 mrs
 4 scheme
 5 school
 6 straight
 7 strategy
 8 street
 9 strike
10 strong
11 structure
12 three
13 through
14 throw
```

```
#vowel-consonant pairs
str_subset(words, "([aeiouy][^aeiouy]){2,}")
```

```
  [1] "absolute"    "agent"       "along"       "america"     "another"
  [6] "apart"       "apparent"    "authority"   "available"   "aware"
 [11] "balance"     "basis"       "become"      "before"      "begin"
 [16] "behind"      "benefit"     "business"    "character"   "closes"
 [21] "community"   "consider"    "cover"       "debate"      "decide"
 [26] "decision"    "definite"    "department"  "depend"      "design"
 [31] "develop"     "difference"  "difficult"   "direct"      "divide"
 [36] "document"    "during"      "economy"     "educate"     "elect"
 [41] "electric"    "eleven"      "encourage"   "environment" "europe"
 [46] "even"        "evening"     "ever"        "every"       "evidence"
 [51] "exact"       "example"     "exercise"    "exist"       "family"
 [56] "figure"      "final"       "finance"     "finish"      "future"
 [61] "general"     "govern"      "holiday"     "honest"      "hospital"
 [66] "however"     "identify"    "imagine"     "individual"  "interest"
 [71] "introduce"   "item"        "jesus"       "level"       "likely"
 [76] "limit"       "local"       "major"       "manage"      "meaning"
 [81] "measure"     "minister"    "minus"       "minute"      "moment"
 [86] "music"       "nature"      "necessary"   "never"       "notice"
 [91] "open"        "operate"     "opportunity" "organize"    "original"
 [96] "over"        "paper"       "paragraph"   "parent"      "particular"
[101] "photograph"  "police"      "policy"      "politic"     "position"
```

```
[106] "positive"    "power"       "prepare"     "present"     "presume"
[111] "private"     "probable"    "process"     "produce"     "product"
[116] "project"     "proper"      "propose"     "protect"     "provide"
[121] "quality"     "realise"     "reason"      "recent"      "recognize"
[126] "recommend"   "record"      "reduce"      "refer"       "regard"
[131] "relation"    "remember"    "report"      "represent"   "result"
[136] "return"      "saturday"    "second"      "secretary"   "secure"
[141] "separate"    "seven"       "similar"     "specific"    "strategy"
[146] "student"     "stupid"      "telephone"   "television"  "therefore"
[151] "thousand"    "together"    "tomorrow"    "tonight"     "total"
[156] "toward"      "travel"      "unit"        "unite"       "university"
[161] "upon"        "visit"       "water"       "woman"
```

Useful functions for handling patterns

str_extract() : extract a string that matches a pattern str_count() : count how many times a pattern occurs within a string

```
str_extract(spot_smaller$album_release_date, "\\d{4}-\\d{2}")
```

```
 [1] "2016-01" "2011-06" "2016-04" "2014-11" "2019-12" "2013-11" NA
 [8] "2011-07" "1990-01" "2018-11"
```

```
spot_smaller |>
  select(album_release_date) |>
  mutate(year_month = str_extract(album_release_date, "\\d{4}-\\d{2}"))
```

```
# A tibble: 10 x 2
   album_release_date year_month
   <chr>              <chr>
 1 2016-01-01         2016-01
 2 2011-06-24         2011-06
 3 2016-04-23         2016-04
 4 2014-11-24         2014-11
 5 2019-12-06         2019-12
 6 2013-11-28         2013-11
 7 2012               <NA>
 8 2011-07-02         2011-07
 9 1990-01-01         1990-01
10 2018-11-16         2018-11
```

```
spot_smaller |>
  select(artist) |>
  mutate(n_vowels = str_count(artist, "[aeiou]"))
```

```
# A tibble: 10 x 2
   artist            n_vowels
   <chr>                <int>
 1 Alok                     1
 2 Beyoncé                  2
 3 Beyoncé                  2
 4 Beyoncé                  2
 5 Camila Cabello           6
 6 Freestyle                3
 7 Kendrick Lamar           4
 8 Kendrick Lamar           4
 9 Kid Frost                2
10 Mike WiLL Made-It        5
```

11. In the spot_smaller dataset, how many words are in each title? (hint \b)

```
spot_smaller |>
  select(title) |>
  mutate(n_words = str_count(title, "\\b[^ ]+\\b"))
```

```
# A tibble: 10 x 2
   title                                                n_words
   <chr>                                                  <int>
 1 Hear Me Now                                                3
 2 Run the World (Girls)                                      4
 3 Formation                                                  1
 4 7/11                                                       1
 5 My Oh My (feat. DaBaby)                                    5
 6 It's Automatic                                             2
 7 Poetic Justice                                             2
 8 A.D.H.D                                                    1
 9 Ya Estuvo                                                  2
10 Runnin (with A$AP Rocky, A$AP Ferg & Nicki Minaj)          8
```

```
str_subset(spot_smaller$title, "\\b[^ ]+\\b")
```

```
 [1] "Hear Me Now"
 [2] "Run the World (Girls)"
 [3] "Formation"
 [4] "7/11"
 [5] "My Oh My (feat. DaBaby)"
 [6] "It's Automatic"
 [7] "Poetic Justice"
 [8] "A.D.H.D"
 [9] "Ya Estuvo"
[10] "Runnin (with A$AP Rocky, A$AP Ferg & Nicki Minaj)"
```

12. In the spot_smaller dataset, extract the first word from every title. Show how you would print out these words as a vector and how you would create a new column on the spot_smaller tibble. That is, produce this:

```
# [1] "Hear"      "Run"       "Formation" "7/11"      "My"        "It's"
# [7] "Poetic"    "A.D.H.D"   "Ya"        "Runnin"
```

Then this:

```
# A tibble: 10 × 2
#   title                                                 first_word
#   <chr>                                                 <chr>
# 1 Hear Me Now                                           Hear
# 2 Run the World (Girls)                                 Run
# 3 Formation                                             Formation
# 4 7/11                                                  7/11
# 5 My Oh My (feat. DaBaby)                               My
# 6 It's Automatic                                        It's
# 7 Poetic Justice                                        Poetic
# 8 A.D.H.D                                               A.D.H.D
# 9 Ya Estuvo                                             Ya
#10 Runnin (with A$AP Rocky, A$AP Ferg & Nicki Minaj) Runnin
```

```
# "^[^ ]+" all non spaces until first space
spot_smaller |>
  select(title) |>
  mutate(first_word = str_extract(title, "^[^ ]+"))
```

```
# A tibble: 10 x 2
   title                                                 first_word
   <chr>                                                 <chr>
```

```
 1 Hear Me Now                                          Hear
 2 Run the World (Girls)                                Run
 3 Formation                                            Formation
 4 7/11                                                 7/11
 5 My Oh My (feat. DaBaby)                              My
 6 It's Automatic                                       It's
 7 Poetic Justice                                       Poetic
 8 A.D.H.D                                              A.D.H.D
 9 Ya Estuvo                                            Ya
10 Runnin (with A$AP Rocky, A$AP Ferg & Nicki Minaj) Runnin
```

```
spot_smaller |>
  select(title) |>
  mutate(first_word = str_extract(title, "^[^ ]+")) |>
  pull(first_word)
```

```
 [1] "Hear"      "Run"       "Formation" "7/11"      "My"        "It's"
 [7] "Poetic"    "A.D.H.D"   "Ya"        "Runnin"
```

OR

```
spot_smaller |>
  select(title) |>
  mutate(first_word = str_extract(title, "\\b[^ ]+\\b"))
```

```
# A tibble: 10 x 2
   title                                          first_word
   <chr>                                          <chr>
 1 Hear Me Now                                    Hear
 2 Run the World (Girls)                          Run
 3 Formation                                      Formation
 4 7/11                                           7/11
 5 My Oh My (feat. DaBaby)                        My
 6 It's Automatic                                 It's
 7 Poetic Justice                                 Poetic
 8 A.D.H.D                                        A.D.H.D
 9 Ya Estuvo                                      Ya
10 Runnin (with A$AP Rocky, A$AP Ferg & Nicki Minaj) Runnin
```

```
spot_smaller |>
  select(title) |>
  mutate(first_word = str_extract(title, "\\b[^ ]+\\b")) |>
  pull(first_word)
```

```
 [1] "Hear"      "Run"       "Formation" "7/11"      "My"        "It's"
 [7] "Poetic"    "A.D.H.D"   "Ya"        "Runnin"
```

OR

```
str_extract(spot_smaller$title, "^[^ ]+\\b")
```

```
 [1] "Hear"      "Run"       "Formation" "7/11"      "My"        "It's"
 [7] "Poetic"    "A.D.H.D"   "Ya"        "Runnin"
```

13. Which decades are popular for playlist_names? Using the bigspotify dataset, try doing each of these steps one at a time!

- filter the bigspotify dataset to only include playlists that include something like “80’s” or “00’s” in their title.
- create a new column that extracts the decade
- use count to find how many playlists include each decade
- what if you include both “80’s” and “80s”?
- how can you count “80’s” and “80s” together in your final tibble?

```
bigspotify |>
  filter(str_detect(playlist_name, "[0-9]{2}('?)s")) |> # or \\d\\d('?)s
  mutate(decade = str_extract(playlist_name, "[0-9]{2}('?)s")) |>
  select(playlist_name, decade) |>
  filter(decade != "08's") |>
  mutate(decade = str_replace(decade, "'", "")) |>
  count(decade)
```

```
# A tibble: 6 x 2
  decade     n
  <chr>  <int>
1 00s       45
2 10s      281
3 50s      100
4 70s      442
5 80s      682
6 90s     1013
```

Grouping and backreferences

```
# find all fruits with repeated pair of letters.
fruit = stringr::fruit
fruit
```

```
 [1] "apple"            "apricot"          "avocado"
 [4] "banana"           "bell pepper"      "bilberry"
 [7] "blackberry"       "blackcurrant"     "blood orange"
[10] "blueberry"        "boysenberry"      "breadfruit"
[13] "canary melon"     "cantaloupe"       "cherimoya"
[16] "cherry"           "chili pepper"     "clementine"
[19] "cloudberry"       "coconut"          "cranberry"
[22] "cucumber"         "currant"          "damson"
[25] "date"             "dragonfruit"      "durian"
[28] "eggplant"         "elderberry"       "feijoa"
[31] "fig"              "goji berry"       "gooseberry"
[34] "grape"            "grapefruit"       "guava"
[37] "honeydew"         "huckleberry"      "jackfruit"
[40] "jambul"           "jujube"           "kiwi fruit"
[43] "kumquat"          "lemon"            "lime"
[46] "loquat"           "lychee"           "mandarine"
[49] "mango"            "mulberry"         "nectarine"
[52] "nut"              "olive"            "orange"
[55] "pamelo"           "papaya"           "passionfruit"
[58] "peach"            "pear"             "persimmon"
[61] "physalis"         "pineapple"        "plum"
[64] "pomegranate"      "pomelo"           "purple mangosteen"
[67] "quince"           "raisin"           "rambutan"
[70] "raspberry"        "redcurrant"       "rock melon"
[73] "salal berry"      "satsuma"          "star fruit"
[76] "strawberry"       "tamarillo"        "tangerine"
[79] "ugli fruit"       "watermelon"
```

```
str_view(fruit, "(..)\\1", match = TRUE)
```

```
 [4] | b<anan>a
[20] | <coco>nut
[22] | <cucu>mber
[41] | <juju>be
```

```
[56] | <papa>ya
[73] | s<alal> berry
```

```
# why does the code below add "pepper" and even "nectarine"?
str_view(fruit, "(..)(.*)\\1", match = TRUE)
```

```
 [4] | b<anan>a
 [5] | bell <peppe>r
[17] | chili <peppe>r
[20] | <coco>nut
[22] | <cucu>mber
[29] | eld<erber>ry
[41] | <juju>be
[51] | <nectarine>
[56] | <papa>ya
[73] | s<alal> berry
```

Tips with backreference: - You must use () around the the thing you want to reference. - To backreference multiple times, use \1 again. - The number refers to which spot you are referencing… e.g. \2 references the second set of ()

```
x1 <- c("abxyba", "abccba", "xyaayx", "abxyab", "abcabc")
str_subset(x1, "(.)(.)(..)\\2\\1")
```

```
[1] "abxyba" "abccba" "xyaayx"
```

```
str_subset(x1, "(.)(.)(..)\\1\\2")
```

```
[1] "abxyab"
```

```
str_subset(x1, "(.)(.)(.)\\1\\2\\3")
```

```
[1] "abcabc"
```

14. Describe to your groupmates what these expressions will match, and provide a word or expression as an example:

- (.)\1\1 » any string with the same 3 characters, ex. 333, bbb, lll

```
str_detect("o328fjknttt890bjfe", "(.)\\1\\1")
```

```
[1] TRUE
```

```
str_detect("ppp", "(.)\\1\\1")
```

```
[1] TRUE
```

- "(.)(.)(.).*\3\2\1" » strings 3 characters where they appear in reverse order later in the word

```
str_detect("cbef789dfgnfdj987", "(.)(.)(.).*\\3\\2\\1")
```

```
[1] TRUE
```

```
str_detect("iuencm14lkwejrc41mfhiuwer", "(.)(.)(.).*\\3\\2\\1")
```

```
[1] TRUE
```

```
str_detect("bbyybb", "(.)(.)(.).*\\3\\2\\1")
```

```
[1] TRUE
```

Which words in `stringr::words` match each expression?

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

```
character(0)
```

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

```
[1] "paragraph"
```

15. Construct a regular expression to match words in `stringr::words` that contain a repeated pair of letters (e.g. "church" contains "ch" repeated twice) but *not* match repeated pairs of numbers (e.g. 507-786-3861).

```
str_subset(words, "([a-z]{2}).*\\1")
```

```
 [1] "appropriate" "church"      "condition"   "decide"      "environment"
 [6] "london"      "paragraph"   "particular"  "photograph"  "prepare"
[11] "pressure"    "remember"    "represent"   "require"     "sense"
[16] "therefore"   "understand"  "whether"
```

```
str_detect("507-786-7861", "([a-z]{2}).*\\1")
```

```
[1] FALSE
```

16. Reformat the album_release_date variable in spot_smaller so that it is MM-DD-YYYY instead of YYYY-MM-DD. (Hint: str_replace().)

```
spot_smaller |>
  mutate(release_date = str_replace(album_release_date, "(\\d{4})-(\\d{2})-(\\d{2})", "\\2-\
  select(album_release_date, release_date)
```

```
# A tibble: 10 x 2
   album_release_date release_date
   <chr>              <chr>
 1 2016-01-01         01-01-2016
 2 2011-06-24         06-24-2011
 3 2016-04-23         04-23-2016
 4 2014-11-24         11-24-2014
 5 2019-12-06         12-06-2019
 6 2013-11-28         11-28-2013
 7 2012               2012
 8 2011-07-02         07-02-2011
 9 1990-01-01         01-01-1990
10 2018-11-16         11-16-2018
```

17. BEFORE RUNNING IT, explain to your partner(s) what the following R chunk will do:

```
sentences %>%
  str_replace("([^ ]+) ([^ ]+) ([^ ]+)", "\\1 \\3 \\2") %>%
  head(5)
```

```
[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."
```

This is going to swap the second and third word in our sentences.

12_strings_part3.qmd : On Your Own #1-9.

On Your Own - Extra practice with strings and regular expressions

1. Describe the equivalents of ?, +, * in {m,n} form.

`?` makes a pattern optional (i.e. it matches 0 or 1 times), so in {m,n} form it would be represented as {0, 1}

`+` lets a pattern repeat (i.e. it matches at least once), so in {m,n} form it would be represented {1, }

`*` lets a pattern be optional or repeats, so in {m,n} form it would be represented as {0, }

2. Describe, in words, what the expression "(.)(.)\2\1" will match, and provide a word or expression as an example.

   The expression "(.)(.)\2\1" would match any 2 character, digit, or special character and then those 2 characters, digit, or special character in reverse order, with nothing in between. The \2 is saying take look for whatever is in our second set of parentheses and \1 is saying whatever is in our first set of parentheses.

```
# an & na
str_detect("anna", "(.)(.)\\2\\1")
```

```
[1] TRUE
```

```
# %0 & 0%
str_detect("yhbfvh%00%", "(.)(.)\\2\\1")
```

```
[1] TRUE
```

```
#  il & li
str_detect("millionaire", "(.)(.)\\2\\1")
```

```
[1] TRUE
```

3. Produce an R string which the regular expression represented by "\..\..\.." matches. In other words, find a string `y` below that produces a TRUE in `str_detect`.

> This regular expression is looking for a . represented by \. followed by any character, digit, or special character, then another . represented by \. followed by any character, digit, or special character and then another . represented by \. followed by any character, digit, or special character, thus giving it the form ".z.z.z" where z can be any character, digit, or special character. Note: the string doesn't have to be by itself.

```
str_detect("tyu iny^.^.0.E..  ... .....", "\\..\\..\\..")
```

```
[1] TRUE
```

```
str_detect("ziling.....zhen", "\\..\\..\\..")
```

```
[1] TRUE
```

```
str_detect(".a.b.c", "\\..\\..\\..")
```

```
[1] TRUE
```

4. Solve with `str_subset()`, using the words from `stringr::words`:

- Find all words that start or end with x.

```
str_subset(words, "^x|x$")
```

```
[1] "box" "sex" "six" "tax"
```

- Find all words that start with a vowel and end with a consonant.

```
str_subset(words, "^[aieou].*[^aieou]$")
```

```
  [1] "about"       "accept"      "account"     "across"      "act"
  [6] "actual"      "add"         "address"     "admit"       "affect"
 [11] "afford"      "after"       "afternoon"   "again"       "against"
 [16] "agent"       "air"         "all"         "allow"       "almost"
 [21] "along"       "already"     "alright"     "although"    "always"
 [26] "amount"      "and"         "another"     "answer"      "any"
 [31] "apart"       "apparent"    "appear"      "apply"       "appoint"
 [36] "approach"    "arm"         "around"      "art"         "as"
 [41] "ask"         "at"          "attend"      "authority"   "away"
 [46] "awful"       "each"        "early"       "east"        "easy"
 [51] "eat"         "economy"     "effect"      "egg"         "eight"
 [56] "either"      "elect"       "electric"    "eleven"      "employ"
 [61] "end"         "english"     "enjoy"       "enough"      "enter"
 [66] "environment" "equal"       "especial"    "even"        "evening"
 [71] "ever"        "every"       "exact"       "except"      "exist"
 [76] "expect"      "explain"     "express"     "identify"    "if"
 [81] "important"   "in"          "indeed"      "individual"  "industry"
 [86] "inform"      "instead"     "interest"    "invest"      "it"
 [91] "item"        "obvious"     "occasion"    "odd"         "of"
 [96] "off"         "offer"       "often"       "okay"        "old"
[101] "on"          "only"        "open"        "opportunity" "or"
[106] "order"       "original"    "other"       "ought"       "out"
[111] "over"        "own"         "under"       "understand"  "union"
[116] "unit"        "university"  "unless"      "until"       "up"
[121] "upon"        "usual"
```

- Find all words that start and end with the same letter

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

```
 [1] "america"    "area"       "dad"        "dead"       "depend"
 [6] "educate"    "else"       "encourage"  "engine"     "europe"
[11] "evidence"   "example"    "excuse"     "exercise"   "expense"
[16] "experience" "eye"        "health"     "high"       "knock"
[21] "level"      "local"      "nation"     "non"        "rather"
[26] "refer"      "remember"   "serious"    "stairs"     "test"
[31] "tonight"    "transport"  "treat"      "trust"      "window"
[36] "yesterday"
```

5. What words in `stringr::words` have the highest number of vowels? What words have the highest proportion of vowels? (Hint: what is the denominator?) Figure this out using the tidyverse and piping, starting with `as_tibble(words) |>`.

```r
# most vowels
as_tibble(words) |>
  mutate(n_vowels = str_count(value, "[aieou]")) |>
  arrange(desc(n_vowels))
```

```
# A tibble: 980 x 2
   value       n_vowels
   <chr>          <int>
 1 appropriate        5
 2 associate          5
 3 available          5
 4 colleague          5
 5 encourage          5
 6 experience         5
 7 individual         5
 8 television         5
 9 absolute           4
10 achieve            4
# i 970 more rows
```

```r
# proportion of vowels
as_tibble(words) |>
  mutate(n_vowels = str_count(value, "[aieou]"),
         word_length = str_length(value),
         prop_vowels = n_vowels/word_length) |>
  arrange(desc(prop_vowels))
```

```
# A tibble: 980 x 4
   value  n_vowels word_length prop_vowels
   <chr>     <int>       <int>       <dbl>
 1 a             1           1       1
 2 area          3           4       0.75
 3 idea          3           4       0.75
 4 age           2           3       0.667
 5 ago           2           3       0.667
 6 air           2           3       0.667
 7 die           2           3       0.667
 8 due           2           3       0.667
 9 eat           2           3       0.667
10 europe        4           6       0.667
# i 970 more rows
```

6. From the Harvard sentences data, use `str_extract` to produce a tibble with 3 columns: the sentence, the first word in the sentence, and the first word ending in "ed" (NA if there isn't one).

```
as_tibble(sentences) |>
  mutate(first_word = str_extract(value, "^[^ ]+\\b"),
         first_word_ed = str_extract(value, "\\b\\w*ed\\b"))
```

```
# A tibble: 720 x 3
   value                                        first_word first_word_ed
   <chr>                                        <chr>      <chr>
 1 The birch canoe slid on the smooth planks.   The        <NA>
 2 Glue the sheet to the dark blue background.  Glue       <NA>
 3 It's easy to tell the depth of a well.       It's       <NA>
 4 These days a chicken leg is a rare dish.     These      <NA>
 5 Rice is often served in round bowls.         Rice       served
 6 The juice of lemons makes fine punch.        The        <NA>
 7 The box was thrown beside the parked truck.  The        parked
 8 The hogs were fed chopped corn and garbage.  The        fed
 9 Four hours of steady work faced us.          Four       faced
10 A large size in stockings is hard to sell.   A          <NA>
# i 710 more rows
```

```
# str_view(sentences, "\\b\\w*ed\\b")
# "^[^ ]+\\b" first word
```

7. Find and output all contractions (words with apostrophes) in the Harvard sentences, assuming no sentence has multiple contractions.

```
as_tibble(sentences) |>
  mutate(contractions = str_extract(value, "\\w*'\\w\\b")) |>
  filter(contractions != is.na(NA))
```

```
# A tibble: 18 x 2
   value                                        contractions
   <chr>                                        <chr>
 1 It's easy to tell the depth of a well.       It's
 2 The soft cushion broke the man's fall.       man's
 3 Open the crate but don't break the glass.    don't
 4 Add the store's account to the last cent.    store's
 5 The beam dropped down on the workman's head. workman's
```

```
 6 Let's all join as we sing the last chorus.   Let's
 7 The copper bowl shone in the sun's rays.     sun's
 8 A child's wit saved the day for us.          child's
 9 A ripe plum is fit for a king's palate.      king's
10 It's a dense crowd in two distinct ways.     It's
11 We don't get much money but we have fun.     don't
12 Ripe pears are fit for a queen's table.      queen's
13 Cheap clothes are flashy but don't last.     don't
14 The facts don't always show who is right.    don't
15 Pack the kits and don't forget the salt.     don't
16 We don't like to admit our small faults.     don't
17 Dig deep in the earth for pirate's gold.     pirate's
18 She saw a cat in the neighbor's house.       neighbor's
```

This prints out all the words with apostrophes in the Harvard sentences, note that words like store's, mans's, king's, queen's, etc are showing up as well.

8. *Carefully* explain what the code below does, both line by line and in general terms.

```
temp <- str_replace_all(words, "^([A-Za-z])(.*)([a-z])$", "\\3\\2\\1")
as_tibble(words) |>
  semi_join(as_tibble(temp)) |>
  print(n = Inf)
```

```
Joining with `by = join_by(value)`

# A tibble: 45 x 1
   value
   <chr>
 1 a
 2 america
 3 area
 4 dad
 5 dead
 6 deal
 7 dear
 8 depend
 9 dog
10 educate
11 else
12 encourage
```

```
13 engine
14 europe
15 evidence
16 example
17 excuse
18 exercise
19 expense
20 experience
21 eye
22 god
23 health
24 high
25 knock
26 lead
27 level
28 local
29 nation
30 no
31 non
32 on
33 rather
34 read
35 refer
36 remember
37 serious
38 stairs
39 test
40 tonight
41 transport
42 treat
43 trust
44 window
45 yesterday
```

The first line, assigns a vector to "temp" of our words in which we use str_replace to change them a little. In our str_replace we are saying the first set of parenthesis is the starting letter, the second set of parentheses is anything of any length (the letters in between the starting letter and the last letter) so this will go to our last set of parentheses which is the last letter, and for our replace we are saying last parentheses, second parentheses, then first parentheses, so we now have words in the form where it starts with the letter it ends with and ends with the letter it starts with. In line 2 we are taking a words and turning it into a tibble,

then keeping all the words that are also in our temp vector. We then print all these words. Now, we have a list of words in which if you swap the first and last letter they are still a word, including words that start and end with the same letter.

Coco and Rotten Tomatoes

We will check out the Rotten Tomatoes page for the 2017 movie Coco, scrape information from that page (we'll get into web scraping in a few weeks!), clean it up into a usable format, and answer some questions using strings and regular expressions.

```
# used to work
# coco <- read_html("https://www.rottentomatoes.com/m/coco_2017")

robotstxt::paths_allowed("https://www.rottentomatoes.com/m/coco_2017")
```

```
 www.rottentomatoes.com
```

```
[1] TRUE
```

```
library(polite)
coco <- "https://www.rottentomatoes.com/m/coco_2017" |>
  bow() |>
  scrape()

top_reviews <-
  "https://www.rottentomatoes.com/m/coco_2017/reviews?type=top_critics" |>
  bow() |>
  scrape()
top_reviews <- html_nodes(top_reviews, ".review-text")
top_reviews <- html_text(top_reviews)

user_reviews <-
  "https://www.rottentomatoes.com/m/coco_2017/reviews?type=user" |>
  bow() |>
  scrape()
user_reviews <- html_nodes(user_reviews, ".js-review-text")
user_reviews <- html_text(user_reviews)
```

9. `top_reviews` is a character vector containing the 20 most recent critic reviews (along with some other junk) for Coco, while `user_reviews` is a character vector with the 10 most recent user reviews.

a) Explain how the code below helps clean up both `user_reviews` and `top_reviews` before we start using them.

```
user_reviews <- str_trim(user_reviews)
top_reviews <- str_trim(top_reviews)
```

> We remove the whitespace from the start and end of the strings in user_review and topreviw

b) Print out the critic reviews where the reviewer mentions "emotion" or "cry". Think about various forms ("cried", "emotional", etc.) You may want to turn reviews to all lower case before searching for matches.

```
str_subset(str_to_lower(user_reviews), "cry|cried|emotional|sad|devasted")
```

```
[1] "such an emotional cultural movie. so good!"
[2] "made me cry many times. sensational!"
[3] "*coco* is a stunning masterpiece from pixar that beautifully captures the essence of mex
[4] "animations give way to ideas such as <i>coco</i> to be conveyed through a medium that is
```

c) In critic reviews, replace all instances where "Pixar" is used with its full name: "Pixar Animation Studios".

```
str_replace(top_reviews, "Pixar", "Pixar Animation Studios")
```

```
 [1] "A fine addition to the Pixar Animation Studios legacy… a very sweet film about family,
 [2] "An unexpectedly brilliant and dynamic story about lineage, connection, and self-discove
 [3] "In a country with an ever increasing Hispanic and Mexican population, a film like Coco
 [4] "I don't think there's any question that Coco is really great."
 [5] "Several times I found myself sobbing without knowing exactly why only to realize why th
 [6] "A wonderful return to form for Pixar Animation Studios, who again deliver the emotional
 [7] "The film has a galloping rhythm, and the animation is scrupulous and ravishing, from it
 [8] "On paper, the mythology scans as complicated and dark, but in the capable hands of Acad
 [9] "Pixar Animation Studios's latest project is a glittering return to non-franchise form a
[10] "Its victorious denouement offers everyone a different way to think about what it means
[11] "At worst it suggests that the brains trust at Pixar Animation Studios, after 22 years o
[12] "Funny, irreverent and eye-popping. It will also make you want to cry at least once but
[13] "This is a charming and very memorable film."
```

```
[14] "Despite the fact that it's so well told and really beautifully directed, it doesn't hav
[15] "... Coco is another triumph for Pixar Animation Studios..."
[16] "Funny and heart-tugging with some knockout tunes, the movie glows with warmth. And how
[17] "Not top-tier Pixar Animation Studios. But decent enough."
[18] "Pixar Animation Studios has raised the animation bar again, with its most musical - and
[19] "While the animation is Pixar Animation Studios perfect, I don't think the story grips t
[20] "Every plot point and thematic implication slots into place, but the pleasures of Coco a
```

d) Find out how many times each user uses "I" in their review. Remember that it could be used as upper or lower case, at the beginning, middle, or end of a sentence, etc.

```
str_count(str_to_lower(user_reviews), "\\b(i)\\b")
```

```
 [1] 0 0 0 3 0 0 3 1 4 2 0 0 0 1 0 0 1 2 0 0
```

```
str_count(str_to_lower(user_reviews), "[i]")
```

```
 [1]  3  4  5 43  2  2 15 31 37 33  1  3 15  5  3 64 27 20  4  7
```

e) Do critics or users have more complex reviews, as measured by average number of commas used? Be sure your code weeds out commas used in numbers, such as "12,345".

```
mean(str_count(str_to_lower(top_reviews), "[^\\d],"))
```

```
[1] 1.35
```

```
mean(str_count(str_to_lower(user_reviews), "[^\\d],"))
```

```
[1] 2.2
```

Users have a mean use of 2.2 commas and critics have a mean use of 1.35.