HW4

Gwynnie Hayes

```r
library(tidyverse)
library(tidyverse)
library(rvest)
library(httr)

#spotify <- read_csv("Data/spotify.csv")
spotify <- read_csv("https://joeroith.github.io/264_spring_2025/Data/spotify.csv")

spot_smaller <- spotify |>
  select(
    title,
    artist,
    album_release_date,
    album_name,
    subgenre,
    playlist_name
  )

spot_smaller <- spot_smaller[c(5, 32, 49, 52, 83, 175, 219, 231, 246, 265), ]
spot_smaller
```

```
# 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
```

Strings Part 2

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

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

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

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

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

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

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

```r
str_view(spot_smaller$subgenre, "pop", match = NA)
```

```
 [1] | indie <pop>timism
 [2] | post-teen <pop>
 [3] | hip <pop>
 [4] | hip <pop>
 [5] | latin <pop>
 [6] | latin hip hop
 [7] | hip hop
 [8] | southern hip hop
 [9] | latin hip hop
[10] | gangster rap
```

```r
str_view(spot_smaller$subgenre, "pop", html = TRUE)
```

```
file:////private/var/folders/mr/t2grrvgn50v_2prqk9lzyyt40000gn/T/RtmpIvp7Bx/filebf9e47c0f243/
```

```
indie poptimism
post-teen pop
hip pop
hip pop
latin pop
```

```r
str_subset(spot_smaller$subgenre, "pop")
```

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

```r
str_detect(spot_smaller$subgenre, "pop")
```

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

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)

```r
spot_smaller |>
  filter(str_detect(subgenre, "pop"))
```

```
# A tibble: 5 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 (Gi~ 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. D~ Camil~ 2019-12-06         Romance    latin p~ 2020 Hits & ~
```

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

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

```r
spot_smaller |>
  mutate(title_length = str_length(title),
         sub_pop = ifelse(str_detect(subgenre, "pop"), "Genre with Pop", "Genre without Pop")
  group_by(sub_pop) |>
  summarise(mean_title_length = mean(title_length))
```

```
# A tibble: 2 x 2
  sub_pop           mean_title_length
  <chr>                         <dbl>
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.
```

```r
bigspotify |>
  mutate(title = str_to_lower(track_name),
         love = str_detect(title, "love"))|>
  group_by(playlist_genre) |>
  summarize(prop_love = mean(love))
```

```
# A tibble: 6 x 2
  playlist_genre prop_love
  <chr>              <dbl>
1 edm               0.0399
2 latin            NA
3 pop               0.0481
4 r&b               0.0639
5 rap              NA
6 rock              0.0450
```

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

```r
# Guess the output!

str_view(spot_smaller$artist, "^K")
```

```
[7] │ <K>endrick Lamar
[8] │ <K>endrick Lamar
[9] │ <K>id Frost
```

```r
#starts with K
str_view(spot_smaller$album_release_date, "01$")
```

```
[1] │ 2016-01-<01>
[9] │ 1990-01-<01>
```

```r
#released on first day of the month
str_view(spot_smaller$title, "^.. ")
```

```
[5] │ <My >Oh My (feat. DaBaby)
[9] │ <Ya >Estuvo
```

```r
str_view(spot_smaller$artist, "[^A-Za-z ]")
```

```
 [2] │ Beyonc<é>
 [3] │ Beyonc<é>
 [4] │ Beyonc<é>
[10] │ Mike WiLL Made<->It
```

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`?)

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

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

```
[181] │ condi<tion>
[347] │ func<tion>
[516] │ men<tion>
[536] │ mo<tion>
[543] │ na<tion>
[631] │ posi<tion>
[667] │ ques<tion>
[695] │ rela<tion>
[732] │ sec<tion>
[804] │ sta<tion>
```

```r
str_subset(words, "tion")
```

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

```r
str_view(words, "^y")
```

```
[975] │ <y>ear
[976] │ <y>es
[977] │ <y>esterday
[978] │ <y>et
[979] │ <y>ou
[980] │ <y>oung
```

```r
str_view(words, "$x")
str_view(words, "^...$")
```

```
  [9] │ <act>
 [12] │ <add>
 [22] │ <age>
 [24] │ <ago>
 [26] │ <air>
 [27] │ <all>
 [38] │ <and>
 [41] │ <any>
 [51] │ <arm>
 [54] │ <art>
 [56] │ <ask>
 [68] │ <bad>
 [69] │ <bag>
 [73] │ <bar>
 [82] │ <bed>
 [89] │ <bet>
 [91] │ <big>
 [94] │ <bit>
[108] │ <box>
[109] │ <boy>
... and 90 more
```

```r
str_view(words, "^.......")
```

```
 [4] │ <absolut>e
 [6] │ <account>
 [7] │ <achieve>
[13] │ <address>
[15] │ <adverti>se
[19] │ <afterno>on
[21] │ <against>
[31] │ <already>
[32] │ <alright>
[34] │ <althoug>h
[36] │ <america>
[39] │ <another>
[43] │ <apparen>t
[46] │ <appoint>
[47] │ <approac>h
[48] │ <appropr>iate
[53] │ <arrange>
[57] │ <associa>te
[61] │ <authori>ty
[62] │ <availab>le
... and 199 more
```

```r
str_view(words, "^[aeiouy]")
```

```
 [1] │ <a>
 [2] │ <a>ble
 [3] │ <a>bout
 [4] │ <a>bsolute
 [5] │ <a>ccept
 [6] │ <a>ccount
 [7] │ <a>chieve
 [8] │ <a>cross
 [9] │ <a>ct
[10] │ <a>ctive
[11] │ <a>ctual
[12] │ <a>dd
[13] │ <a>ddress
[14] │ <a>dmit
[15] │ <a>dvertise
[16] │ <a>ffect
[17] │ <a>fford
[18] │ <a>fter
[19] │ <a>fternoon
[20] │ <a>gain
... and 161 more
```

```r
str_view(words, "[^e]ed$")
```

```
 [82] │ <bed>
[410] │ hund<red>
[690] │ <red>
```

```r
str_view(words, "q[^u]")
```

More useful regular expressions:

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

```r
str_view(spot_smaller$album_name, "\\d")
```

```
[2] │ <4>
[8] │ Section.<8><0>
```

```r
str_view(spot_smaller$album_name, "\\s")
```

```
 [1] │ Hear< >Me< >Now
 [4] │ BEYONCÉ< >[Platinum< >Edition]
 [6] │ It's< >Automatic
 [7] │ good< >kid,< >m.A.A.d< >city< >(Deluxe)
 [9] │ Hispanic< >Causing< >Panic
[10] │ Creed< >II:< >The< >Album
```

```r
str_view(spot_smaller$album_name, "\\b")
```

```
 [1] │ <>Hear<> <>Me<> <>Now<>
 [2] │ <>4<>
 [3] │ <>Lemonade<>
 [4] │ <>BEYONCÉ<> [<>Platinum<> <>Edition<>]
 [5] │ <>Romance<>
 [6] │ <>It<>'<>s<> <>Automatic<>
 [7] │ <>good<> <>kid<>, <>m<>.<>A<>.<>A<>.<>d<> <>city<> (<>Deluxe<>)
 [8] │ <>Section<>.<>80<>
 [9] │ <>Hispanic<> <>Causing<> <>Panic<>
[10] │ <>Creed<> <>II<>: <>The<> <>Album<>
```

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.)

```r
str_view(spot_smaller$title, "$")
```

```
 [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)<>
```

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

```
[10] | Runnin (with A<$>AP Rocky, A<$>AP Ferg & Nicki Minaj)
```

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!

```r
bigspotify |>
  filter(str_detect(track_name, "\\$")) |>
  select(track_name, track_artist) |>
  filter(!str_detect(track_name, "feat.")) |>
  filter(!str_detect(track_name, "with"))
```

```
# A tibble: 21 x 2
   track_name                         track_artist
   <chr>                              <chr>
 1 Wing$                              Macklemore & Ryan Lewis
 2 $Dreams                            Max Frost
 3 NO TRU$T                           NUGAT
 4 A$AP Forever                       A$AP Rocky
 5 Sie wollen meine Loui$ (Don Dollar) Kulturerbe Achim
 6 Foe Tha Love Of $                  Bone Thugs-N-Harmony
 7 A$AP                               Dillom
 8 $$$ - Remix                        Saramalacara
 9 Fre$h                              Lil Whigga
10 $ENHOR                             FBC
# i 11 more rows
```

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

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

```
# A tibble: 2 x 23
  track_id               track_name track_artist track_popularity track_album_id
  <chr>                  <chr>      <chr>                   <dbl> <chr>
1 7pse475uICmWRY5hEkvPvI $20 Fine   Jimi Hendrix               44 0EfHWQeb3T1UJ~
2 1ivrsBwgQe5KwjMHu8xje4 $100 (fea~ Mibbs                      20 6DjgLGR3L3LjG~
# 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>
```

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

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

```
 [4] │ <BEYONC>É [Platinum Edition]
[10] │ Creed <II>: The Album
```

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

```
 [1] │ <2016-01>-01
 [2] │ <2011-06>-24
 [3] │ <2016-04>-23
 [4] │ <2014-11>-24
 [5] │ <2019-12>-06
 [6] │ <2013-11>-28
 [8] │ <2011-07>-02
 [9] │ <1990-01>-01
[10] │ <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 "A.A" (in addition to "BEYONC" and "II"). That is, pick up strings where there might be a period between capital letters.

```r
str_view(spot_smaller$album_name, "[A-Z]{1}.?[A-Z]{1}")
```

```
 [4] │ <BEY><ONC>É [Platinum Edition]
 [7] │ good kid, m.<A.A>.d city (Deluxe)
[10] │ Creed <II>: The Album
```

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

- "^.*$"
- "\{.+\}"

```r
str_view("aria is being a little silly goofy", "^.*$")
```

```
[1] │ <aria is being a little silly goofy>
```

```r
str_view("aria dont think toooo hard {here}!", "\\{.+\\}")
```

```
[1] │ aria dont think toooo hard <{here}>!
```

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

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

```r
str_view(words, "^[^aeiou]{3}")
```

```
[150] │ <Chr>ist
[151] │ <Chr>istmas
[249] │ <dry>
[328] │ <fly>
[538] │ <mrs>
[724] │ <sch>eme
[725] │ <sch>ool
[811] │ <str>aight
[812] │ <str>ategy
[813] │ <str>eet
[814] │ <str>ike
[815] │ <str>ong
[816] │ <str>ucture
[836] │ <sys>tem
[868] │ <thr>ee
[869] │ <thr>ough
[870] │ <thr>ow
[895] │ <try>
[901] │ <typ>e
[952] │ <why>
```

```r
str_view(words, "([aeiou][^aeiou]){2}")
```

```
  [4] │ abs<olut>e
 [23] │ <agen>t
 [30] │ <alon>g
 [36] │ <amer>ica
 [39] │ <anot>her
 [42] │ <apar>t
 [43] │ app<aren>t
 [61] │ auth<orit>y
 [62] │ ava<ilab>le
 [63] │ <awar>e
 [64] │ <away>
 [70] │ b<alan>ce
 [75] │ b<asis>
 [81] │ b<ecom>e
 [83] │ b<efor>e
 [84] │ b<egin>
 [85] │ b<ehin>d
 [87] │ b<enef>it
[119] │ b<usin>ess
[143] │ ch<arac>ter
... and 149 more
```

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

```r
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"
```

```r
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)

```
str_view(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>)
```

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

```
 [1] 3 4 1 1 5 2 2 1 2 8
```

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
```

```r
str_view(spot_smaller$title, "^\\w+")
```

```
 [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)
```

```
spot_smaller |>
  select(title) |>
  mutate(first_word = str_extract(spot_smaller$title, "^\\w+"))
```

```
# 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
 5 My Oh My (feat. DaBaby)                              My
 6 It's Automatic                                       It
 7 Poetic Justice                                       Poetic
 8 A.D.H.D                                              A
 9 Ya Estuvo                                            Ya
10 Runnin (with A$AP Rocky, A$AP Ferg & Nicki Minaj) 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 |>
  select(playlist_name) |>
  filter(str_detect(playlist_name, "[\\d]0[']?s")) |>
  mutate(playlist_name = str_replace(playlist_name, "'s", "s"),
         decade = str_extract(playlist_name, "[\\d]0s")) |>
  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"       
```

```r
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
```

```r
# 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 ()

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

```
[1] │ <abxyba>
[2] │ <abccba>
[3] │ <xyaayx>
```

```r
str_view(x1, "(.)(.)(..)\\1\\2")
```

```
[4] │ <abxyab>
```

```r
str_view(x1, "(.)(.)(.)\\1\\2\\3")
```

```
[5] │ <abcabc>
```

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

- (.)\1\1 no words match
- "(.)(.)(.).*\3\2\1" paragraph matches

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

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

```
[598] │ <paragrap>h
```

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).

```r
str_view(words, "(..)(..)\\1")
```

```
[152] │ <church>
[217] │ <decide>
[638] │ p<repare>
[858] │ the<refore>
```

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().)

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

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

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 switching the 2nd and 3rd words in the sentence.

Strings Part 3

1. Describe the equivalents of ?, +, * in {m,n} form. {0, 1} {1,} {0,}
2. Describe, in words, what the expression "(.)(.)\2\1" will match, and provide a word or expression as an example.

the expression will match any word that has a repeating phrase that is opposite.

```
sword <- c("finniy", "finally")

str_view(sword, "(.)(.)\\2\\1")
```

```
[1] | f<inni>y
```

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`.

```
y <- "ad.s.s.s.s.s"

str_detect(y,"\\..\\..\\..")
```

```
[1] TRUE
```

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

- Find all words that start or end with x.
- Find all words that start with a vowel and end with a consonant.
- Find all words that start and end with the same letter

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

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

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

```
  [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"
```

```r
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
as_tibble(words) |>
  mutate(num_vowels = str_count(words, "[aeiou]"), num_letters = str_count(words), prop_vow =
  arrange(desc(prop_vow))
```

```
# A tibble: 980 x 4
   value  num_vowels num_letters prop_vow
   <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).

```r
as_tibble(sentences) |>
  mutate(first_word = str_extract(sentences, "^\\w+"),
         first_ed = str_extract(sentences, "\\w+ed"))
```

```
# A tibble: 720 x 3
   value                                       first_word first_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         <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
```

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

```r
as_tibble(sentences) |>
  mutate(contraction = str_extract(sentences, "\\w+'\\w+"))
```

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

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

```r
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
```

this is words that start with any letter upper or lower case then any characters then ends in any lower case character, then reorders the first and last letter of the words so abc would become cba, then its joining with words tibble and printing the output

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")

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)
```

str_trim gets rid of unnecessary white space at the beginning and end of a string

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.

```
as_tibble(top_reviews) |>
  mutate(value = str_to_lower(value)) |>
  filter(str_detect(value, "cr\\w* | emotion\\w*"))
```

```
# A tibble: 5 x 1
  value
  <chr>
1 in a country with an ever increasing hispanic and mexican population, a film ~
2 a wonderful return to form for pixar, who again deliver the emotional and cre~
3 the film has a galloping rhythm, and the animation is scrupulous and ravishin~
4 funny, irreverent and eye-popping. it will also make you want to cry at least~
5 every plot point and thematic implication slots into place, but the pleasures~
```

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.

```r
as_tibble(user_reviews) |>
  mutate(i_count = str_count(user_reviews, "[iI]"))
```

```
# A tibble: 20 x 2
   value                                                               i_count
   <chr>                                                                 <int>
 1 "I loved the plot twist at the end, and the designs of some of the c~      10
 2 "Captivatingly beautiful. Will tug at your heartstrings."                   5
 3 "Such A Beautiful Film, This Story Is Disney/Pixar At Their Finest! ~      16
 4 "I find myself as a grown man watching this every once and while whe~      10
 5 "This was a good and funny musical that everyone can in joy."               3
 6 "Definitely the best movie of 2017 and what makes it so are the beau~      38
 7 "Great movie excited about the second movie"                                3
 8 "flawless film. just stunning."                                             2
 9 "All Time Classic. Just masterfully done across the board. The music~      64
10 "I've seen this movie a hundred times, and every time it gets to me ~      16
11 "The film's visuals and narrative take you on a journey through cult~       5
12 "How can I not love this movie? Coco is my favorite movie in Pixar. ~      72
13 "What's not to love?  This tender story of the little boy who journe~      17
14 "Another enjoyable movie from Pixar.  Heartwarming and great for the~       5
15 "It's one of the best movies pixar has made in these late years. No ~      24
16 "Very colorful much like Encanto and had a fun way of dealing with d~       4
17 "Mild spoilers: \n\nI've heard people make fun of the fact that ther~      21
18 "So good, visually stunning."                                               2
19 "Excellent coulorful movie explaining the Mexican culture of the Day~       4
20 "A very good and well made pixar film. The musics are fire and the p~       6
```

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".

```r
as_tibble(user_reviews) |>
  mutate(comma_count = str_count(user_reviews, ",")) |>
  summarize(avg_commas = mean(comma_count))
```

```
# A tibble: 1 x 1
  avg_commas
       <dbl>
1          2
```

```r
as_tibble(top_reviews) |>
  mutate(comma_count = str_count(top_reviews, ",")) |>
  summarize(avg_commas = mean(comma_count))
```

```
# A tibble: 1 x 1
  avg_commas
       <dbl>
1       1.35
```