Open APIs and Web Scraping

Evan Bowman

2023-03-28

Using APIs

1. Pick a website of your choice (not discussed in class) that requires a free API key and has recent data. You may use an R package for the website or use {httr} to interact directly with the API.

I will be using Spotifyr to run analysis on different aspects of music.

2. Identify the link to instructions to obtain a key for the API.

The link to help create an API for Spotify is https://developer.Spotify.com/my-applications/#!/applications

3. Use your API with {keyring} to download a data set with multiple variables. Do not include your key in your file.

The access token was generated by inputting my Spotify Client ID and Password. This code chunk was not included in the final knit due to privacy reasons.

4. Convert elements of interest into a tibble

```
my_artists <- get_my_top_artists_or_tracks(type = 'artists',
                            time_range = 'medium_term',
                            limit = 50) %>%
    select(name, genres, popularity) %>%
    rowwise %>%
    mutate(genres = paste(.data$genres, collapse = ', ')) %>%
    ungroup %>%
    tibble()
head(my_artists)
```

```
## # A tibble: 6 x 3
##   name                genres                                              popul~1
##   <chr>               <chr>                                                 <int>
## 1 The Backseat Lovers indie pop, modern rock, slc indie                        69
## 2 Sam Fender          modern rock, north east england indie                    69
## 3 Built To Spill      alternative rock, anti-folk, idaho indie, indie r~       52
## 4 Mac Miller          hip hop, pittsburgh rap, rap                             85
## 5 Pavement            alternative rock, anti-folk, art rock, dream pop,~       59
## 6 The Strokes         alternative rock, garage rock, modern rock, perma~       78
## # ... with abbreviated variable name 1: popularity
```

5. State a question of interest.

How mainstream is my music taste: that is, what is the distribution of popularity of my 50 most listened to artist?

Given the specificity that Spotify returned for genres, I will attempt to mutate the data to give it more overarching genre tags. I recognize for some of these tags, I had bias in how I forced a genre on a particular tag.

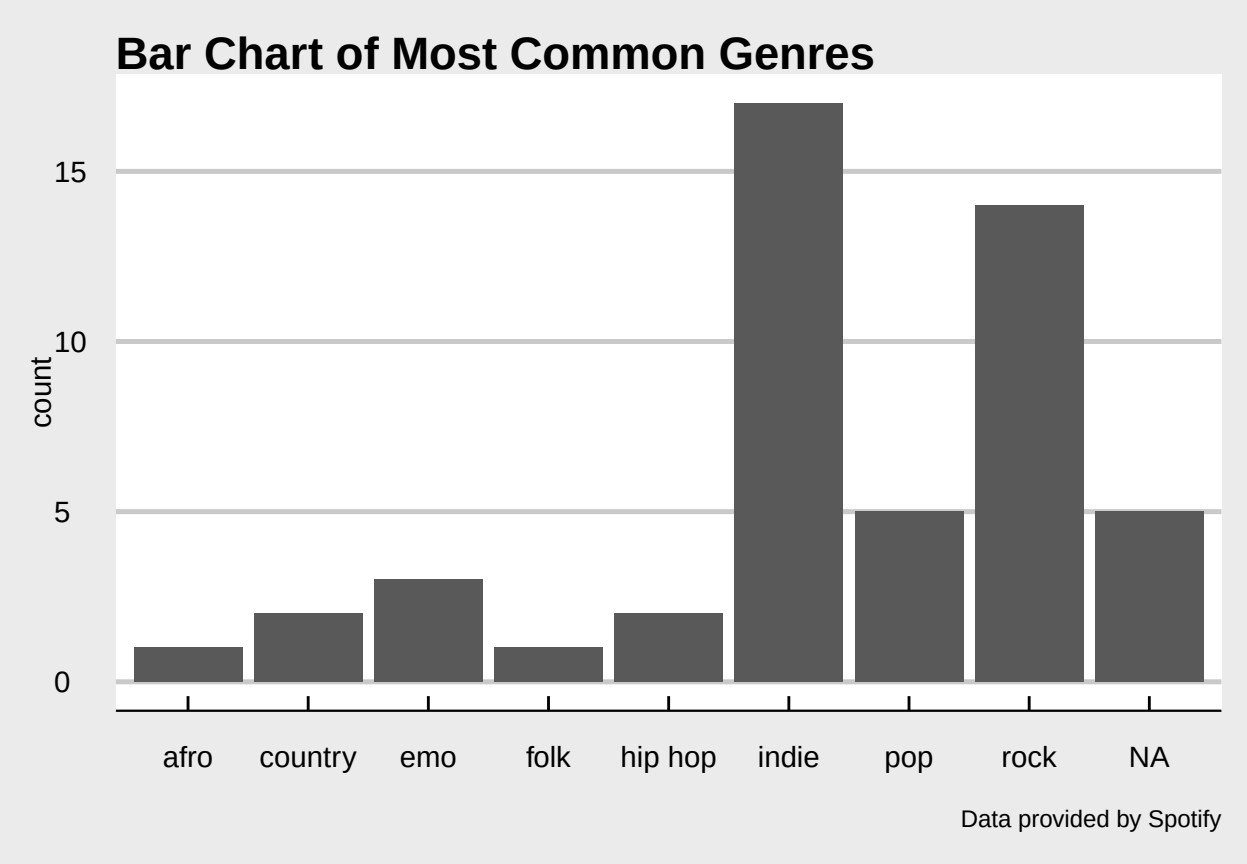
```
genres <- my_artists %>%
  separate(genres, into = "genre", sep = ",") %>%
  mutate(genre = case_when(
    str_detect(genre, "rock") ~ "rock",
    str_detect(genre, "emo") ~ "emo",
    str_detect(genre, "indie") ~ "indie",
    str_detect(genre, "country") ~ "country",
    str_detect(genre, "hip hop") ~ "hip hop",
    str_detect(genre, "afro") ~ "afro",
    str_detect(genre, "soul") ~ "soul",
    str_detect(genre, "pop") ~ "pop",
    str_detect(genre, "blues") ~ "blues",
    str_detect(genre, "americana") ~ "folk",
    str_detect(genre, "wave") ~ "rock",
    str_detect(genre, "standards") ~ "jazz",
    str_detect(genre, "drumming") ~ "rock",
    str_detect(genre, "punk") ~ "punk",
    str_detect(genre, "neo") ~ "folk"
    )
  )

head(genres)
```

```
## # A tibble: 6 x 3
##   name                genre   popularity
##   <chr>               <chr>        <int>
## 1 The Backseat Lovers indie           69
## 2 Sam Fender          rock            69
## 3 Built To Spill      rock            52
## 4 Mac Miller          hip hop         85
## 5 Pavement            rock            59
## 6 The Strokes         rock            78
```

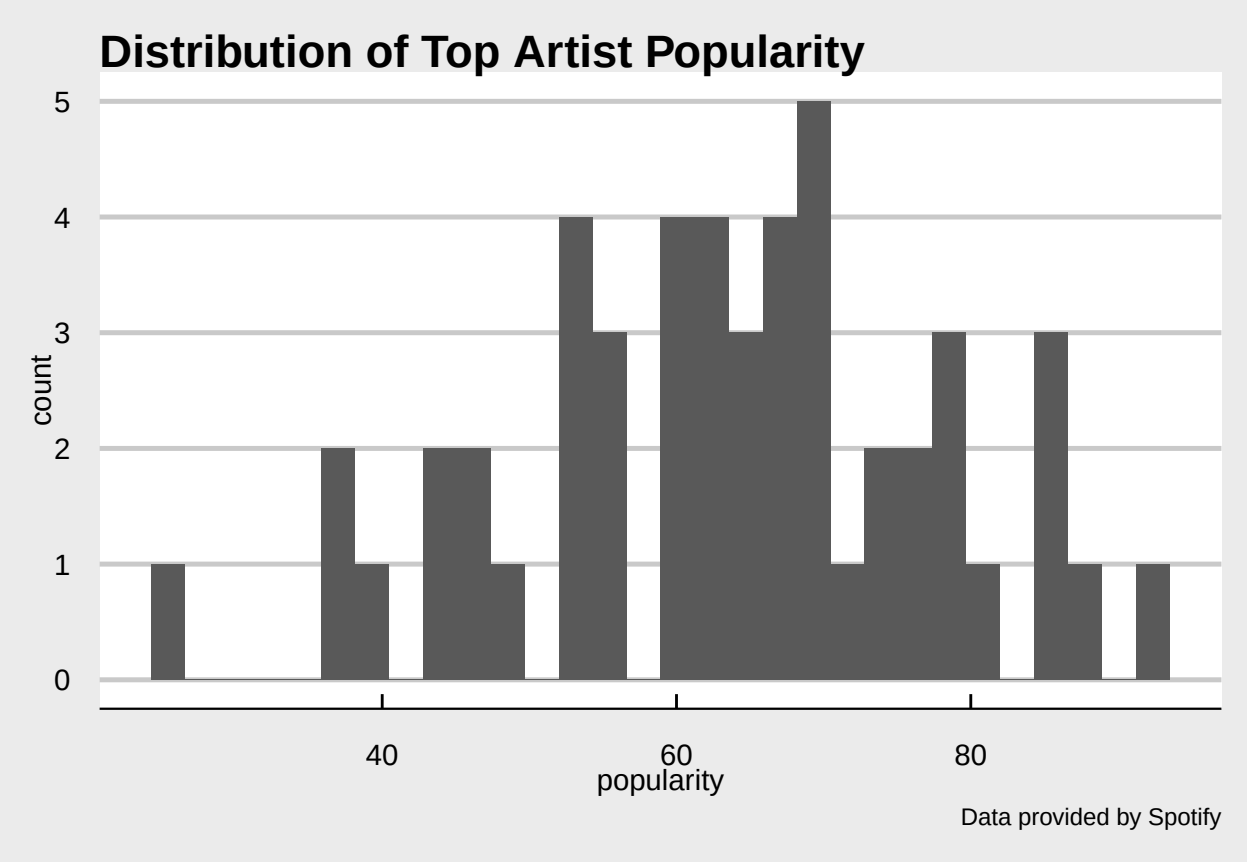
6. Create an appropriate plot with proper labels and theme to analyze your question of interest.

```
# Bar plot of most common genres
genres %>%
  ggplot(aes(x = genre)) +
  geom_bar() +
  xlab("") +
  ggthemes::theme_economist_white() +
  ggtitle("Bar Chart of Most Common Genres") +
  labs(caption = "Data provided by Spotify")
```




```
# Distribution of top artist popularity
my_artists %>%
  ggplot(aes(x = popularity)) +
  geom_histogram()+
  ggthemes::theme_economist_white() +
  ggtitle("Distribution of Top Artist Popularity") +
  labs(caption = "Data provided by Spotify")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



7. Interpret the plot to answer your question.

From the bar plot returned above, I definitely listen to more indie and rock artists. Like I said previously, there is some variability within these categories with the way I coded the genre tags (i.e. classic rock, alternative rock, etc.).

From the histogram of popularity, I would say there is a decent amount of variability in my top artists popularity. From Spotify's website: popularity is determined by streams and songs saved to a user's library. With this knowledge, I would say that my music taste is not mainstream due to the high concentration of artists with popularity in between 50 and 75. It was interesting to see some of my top returned artists with abysmal popularity. However, I investigated this further using:

```
my_artists %>%
  filter(popularity < 40)
```

```
## # A tibble: 3 x 3
##   name           genres            popularity
##   <chr>          <chr>                  <int>
## 1 The Weeks      mississippi indie         37
## 2 Duncan Fellows austindie                 36
## 3 Lo Noom        mississippi indie         25
```

It turns out that these artists with poor popularity are actually bands that are very local bands from my hometown. This makes sense as they have never really gathered a large following outside of the state.

8. Read the site guidance for use of their data, e.g., terms and conditions, API documentation, or FAQ (if they exist)) and, in a few sentences, comment on the severity or strictness of the guidance or the lack of guidance. Discuss how the guidance could enable or restrain unethical uses of the data.

I feel the terms and conditions of Spotify's Developer Program is relatively on par for the music industry. Upon looking at the first page of the terms and conditions, emphasis is placed on protecting both users and artists by requiring users have control of their data and can disconnect their accounts from any developer projects at any time. There is also transparency requirements such as not allowing the creation of bots that can increase popularity through automatic streaming. I think this is also important based on how artists are compensated on the platform (i.e. paid per stream). I think these policies are pretty thorough in how they protect both users, artists, and the company.

IMDB List of Best Foreign Films

IMDB has a list of the Top 100 Best Foreign Films. This question requires scraping the following elements from the webpage and creating a tibble with all of these elements.

- Number
- Title
- Year
- MPAA Rating
- Length in minutes
- Genre
- Star Rating
- Metascore Rating
- Gross Receipts
- Votes
- Country

Required Questions

1. Scrape the following elements, convert the scraped data into a tibble, and add logical variables to identify which rows belong to which variable. You can use various approaches to accomplish this. The following steps are suggested to guide your work.

a. Download the entire web page and scrape the required elements and save the resulting html object into a variable.

```
html_obj <- read_html("https://www.imdb.com/list/ls062615147/")
```

b. Create a reference set of individually scraped elements

- Create a vector of the variable names of interest called `my_vars`
- Create a vector of the CSS tags for the variable of interest called `my_css`.
  - These are the tags I used so you can check what you get for each variable.
    - .genre

```
genre <- html_nodes(html_obj,
                    css = ".genre") %>%
  html_text()
head(genre)
```

```
## [1] "\nCrime, Drama            "
## [2] "\nDrama, Romance            "
## [3] "\nAction, Adventure, Drama            "
## [4] "\nComedy, Drama, Thriller            "
## [5] "\nAction, Crime, Thriller            "
## [6] "\nDrama, Romance, Thriller            "
```

- .lister-item-header a

```
title <- html_nodes(html_obj,
                    css = ".lister-item-header a") %>%
  html_text()
head(title)
```

```
## [1] "City of God"             "A Short Film About Love"
## [3] "Battle Royale"           "Wild Tales"
## [5] "Run Lola Run"            "The Handmaiden"
```

- .text-primary

```
rankings <- html_nodes(html_obj,
           css = ".text-primary") %>%
  html_text()
head(rankings)
```

```
## [1] "1." "2." "3." "4." "5." "6."
```

- .ipl-rating-star.small .ipl-rating-star__rating

```
stars <- html_nodes(html_obj,
           css = ".ipl-rating-star.small .ipl-rating-star__rating") %>%
  html_text()
head(stars)
```

```
## [1] "8.6" "8.2" "7.6" "8.1" "7.7" "8.1"
```

- .text-muted.unbold

```
year <- html_nodes(html_obj,
           css = ".text-muted.unbold") %>%
  html_text()
head(year)
```

```
## [1] "IMDb user rating (average)" "Number of votes"
## [3] "Release year or range"      "(2002)"
## [5] "(1988)"                     "(2000)"
```

- .ghost~ .text-muted+ span

```
gross <- html_nodes(html_obj,
          css = ".ghost~ .text-muted+ span") %>%
  html_text()
head(gross)
```

```
## [1] "|"      "|"      "$7.56M" "$3.11M" "$7.27M" "$2.01M"
```

- .mode-detail .list-description p

```
 country <- html_nodes(html_obj,
          css = ".mode-detail .list-description p") %>%
  html_text()
head(country)
```

```
## [1] "From Brazil"      "From Poland"      "From Japan"       "From Argentina"
## [5] "From Germany"     "From South Korea"
```

- .text-muted+ span:nth-child(2)

```
votes <- html_nodes(html_obj,
          css = ".text-muted+ span:nth-child(2)") %>%
  html_text()
head(votes)
```

```
## [1] ""        ""        ""        "765,611" "24,445"  "186,990"
```

- .list-description p

```
ratings <- html_nodes(html_obj,
          css = ".certificate") %>%
  html_text()
head(ratings)
```

```
## [1] "R"         "Not Rated" "Not Rated" "R"         "R"         "Not Rated"
```

- .metascore

```
metascore <- html_nodes(html_obj,
          css = ".metascore") %>%
  html_text()
head(metascore)
```

```
## [1] "79        " "81        " "77        " "77        " "84        "
## [6] "89        "
```

- .runtime

```
runtime <- html_nodes(html_obj,
           css = ".runtime") %>%
  html_text()
head(runtime)
```

```
## [1] "130 min" "87 min"  "114 min" "122 min" "80 min"  "145 min"
```

```
my_vars <- c("Number", "Title", "Year", "MPAA Rating", "Length in minutes", "Genre", "Star Rating", "Me

my_css <- c(".genre, .lister-item-header a, .text-primary, .ipl-rating-star.small .ipl-rating-star__rati
```

- Use `map()` to scrape each element in `my_css` into a list called `movie_list`.
- Assign names for the list elements in `movie_list` using the values from `my_vars`.
- Use `map_dbl()` to create a named vector of the length of each item in the list and sum the total.
- Your output should look *similar* to the following.
- Note that some elements have more than 100 elements and some have less. The ones with more are due to the CSS identifying elements that may not have been visible in Selector Gadget. You will remove them later.
- The ones with fewer than 100 elements are where some movies are missing a value but we don't know for which movies we have values.

c. Use a {stringr} function to collapse `my_css` into a single value called `css_values` containing all of the css elements separated by ",".

```
my_css <- str_flatten(my_css)
```

d. Use `css_values` to scrape all of the elements at once into a single html object.

```
movie_obj <- html_nodes(html_obj,
                        css = my_css)
```

e. Create a new variable with the text from each element.

```
movie_text <- html_text(movie_obj)

head(movie_text)
```

```
## [1] "|"                          "|"
## [3] "IMDb user rating (average)" ""
## [5] "Number of votes"            ""
```

- Confirm that the length of the vector is the same as the sum of the lengths of the elements from `movie_list`

```
length(movie_text) # Matches the length of the map function table found in the homework rubric
```

```
## [1] 1011
```

- Use head() to check the first 12 values.

```
head(movie_text, 12)
```

```
##  [1] "|"                          "|"
##  [3] "IMDb user rating (average)" ""
##  [5] "Number of votes"            ""
##  [7] "Release year or range"      ""
##  [9] "1."                         "City of God"
## [11] "(2002)"                     "R"
```

- Remove any values prior to the number of the first movie (1) which should be followed by "City of God"

```
tibble(text = movie_text) %>%
  slice(-(1:8))
```

```
## # A tibble: 1,003 x 1
##    text
##    <chr>
##  1 "1."
##  2 "City of God"
##  3 "(2002)"
##  4 "R"
##  5 "130 min"
##  6 "\nCrime, Drama            "
##  7 "8.6"
##  8 "79        "
##  9 "765,611"
## 10 "$7.56M"
## # ... with 993 more rows
```

f. Create a tibble from the text and use a {stringr} function to remove any extra white space in the text or on either end.

```
str_squish(movie_text) %>%
  tibble() %>%
  slice(-(1:8))
```

```
## # A tibble: 1,003 x 1
##    .
##    <chr>
##  1 1.
##  2 City of God
##  3 (2002)
##  4 R
```

```
##  5 130 min
##  6 Crime, Drama
##  7 8.6
##  8 79
##  9 765,611
## 10 $7.56M
## # ... with 993 more rows
```

g. Create logical variables to uniquely identify the rows for each variable. Discard any non-movie-related data prior to the first row with movie data.

```
## Rank and movie for future mutate
imdb_rank_movie <- html_nodes (html_obj,
                      css = ".lister-item-header a , .text-primary")
rank_movie_text <- html_text(imdb_rank_movie)

tibble(text = rank_movie_text) %>%
  mutate(
    rownum = row_number(),
    iseven = rownum %% 2 == 0,
    movie = rep(1:100, each = 2)) -> rank_movie_text

rank_movie_text %>%
  select(-rownum) %>%
  pivot_wider(names_from = iseven, values_from = text) %>%
  select(-movie, rank = "FALSE", movie = "TRUE") %>%
  mutate(rank = parse_number(rank)) -> rank_movie

tibble(text = movie_text) %>%
  slice(-(1:8)) %>%
  mutate(is_movie_rank = str_detect(text, "^\\d+\\.$"),
         movie_num = cumsum(is_movie_rank),
         is_name = text %in% rank_movie$movie,
         is_year = str_detect(text, "\\(\\d+\\)"),
         is_genre = str_detect(text, "^\\n+\\D"),
         is_run_time = str_detect(text, "^\\d{2,3} min$"),
         is_country = str_detect(text, "(From)"),
         is_metascore = text %in% metascore,
         is_gross = str_detect(text, "\\$"),
         is_stars = text %in% stars,
         is_vote = str_detect(text, "\\d{3,}$"),
         is_mpaa = text %in% ratings) -> test
head(test)
```

```
## # A tibble: 6 x 13
##   text   is_mo~1 movie~2 is_name is_year is_ge~3 is_ru~4 is_co~5 is_me~6 is_gr~7
##   <chr>  <lgl>     <int> <lgl>   <lgl>   <lgl>   <lgl>   <lgl>   <lgl>   <lgl>
## 1 "1."   TRUE          1 FALSE   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE
## 2 "City~ FALSE         1 TRUE    FALSE   FALSE   FALSE   FALSE   FALSE   FALSE
## 3 "(200~ FALSE         1 FALSE   TRUE    FALSE   FALSE   FALSE   FALSE   FALSE
## 4 "R"    FALSE         1 FALSE   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE
## 5 "130 ~ FALSE         1 FALSE   FALSE   FALSE   TRUE    FALSE   FALSE   FALSE
## 6 "\nCr~ FALSE         1 FALSE   FALSE   TRUE    FALSE   FALSE   FALSE   FALSE
```

```
## # ... with 3 more variables: is_stars <lgl>, is_vote <lgl>, is_mpaa <lgl>, and
## #   abbreviated variable names 1: is_movie_rank, 2: movie_num, 3: is_genre,
## #   4: is_run_time, 5: is_country, 6: is_metascore, 7: is_gross
```

3. Use {dplyr} and {tidyr} functions to tidy/reshape the tibble, without the logical variables, and save into a new tibble.

```
test%>%
  mutate(key = case_when(
    is_movie_rank ~ "rank",
    is_name ~ "name",
    is_year ~ "year",
    is_genre ~ "genre",
    is_run_time ~ "run_time",
    is_country ~ "country",
    is_metascore ~ "metacritic",
    is_gross ~ "gross",
    is_stars ~ "stars",
    is_vote ~ "votes",
    is_mpaa ~ "rating")) %>%
  select(key, text, movie_num) %>%
  pivot_wider(names_from = key, values_from = text) -> movie_df

movie_df%>%
  select(-1, -13) -> movie_df

movie_df[7,3] = "(2016)"
```

4. Use {readr} parse functions and {stringr} functions to clean the data and save back to the tibble.

- Eliminate any extra variables and characters.
- Ensure all apparent numbers are numeric.
- Use a {readr} function to convert `country` into a factor.
- Show the first 6 rows of the tibble and visually check the class for each variable.

```
movie_df %>%
  mutate(genre = str_replace(genre, "\\n", ""),
         genre = str_squish(genre),
         rank = parse_number(rank),
         year = parse_number(year),
         run_time = parse_number(run_time),
         stars = parse_number(stars),
         metacritic = parse_number(metacritic),
         votes = parse_number(votes),
         gross = parse_number(gross),
         country = str_replace_all(country, "From", ""),
         country = parse_factor(country)) %>%
  rename("gross_(M)" = gross) -> movie_df

head(movie_df)
```

```
## # A tibble: 6 x 11
```

```
##    rank name      year rating run_t~1 genre stars metac~2  votes gross~3 country
##   <dbl> <chr>    <dbl> <chr>    <dbl> <chr> <dbl>   <dbl>  <dbl>   <dbl> <fct>
## 1     1 City of~  2002 R          130 Crim~   8.6      79 765611    7.56 Brazil
## 2     2 A Short~  1988 Not R~      87 Dram~   8.2      NA  24445   NA    Poland
## 3     3 Battle ~  2000 Not R~     114 Acti~   7.6      81 186990   NA    Japan
## 4     4 Wild Ta~  2014 R          122 Come~   8.1      77 203102    3.11 Argent~
## 5     5 Run Lol~  1998 R           80 Acti~   7.7      77 201604    7.27 Germany
## 6     6 The Han~  2016 Not R~     145 Dram~   8.1      84 154242    2.01 South ~
## # ... with abbreviated variable names 1: run_time, 2: metacritic,
## #   3: `gross_(M)`
```

5.

- a. Create a plot of the length of a film and its gross, color coded by rating, where you *filter out any MPAA category with less than 4 films.* Add a linear smoother *for each rating.*

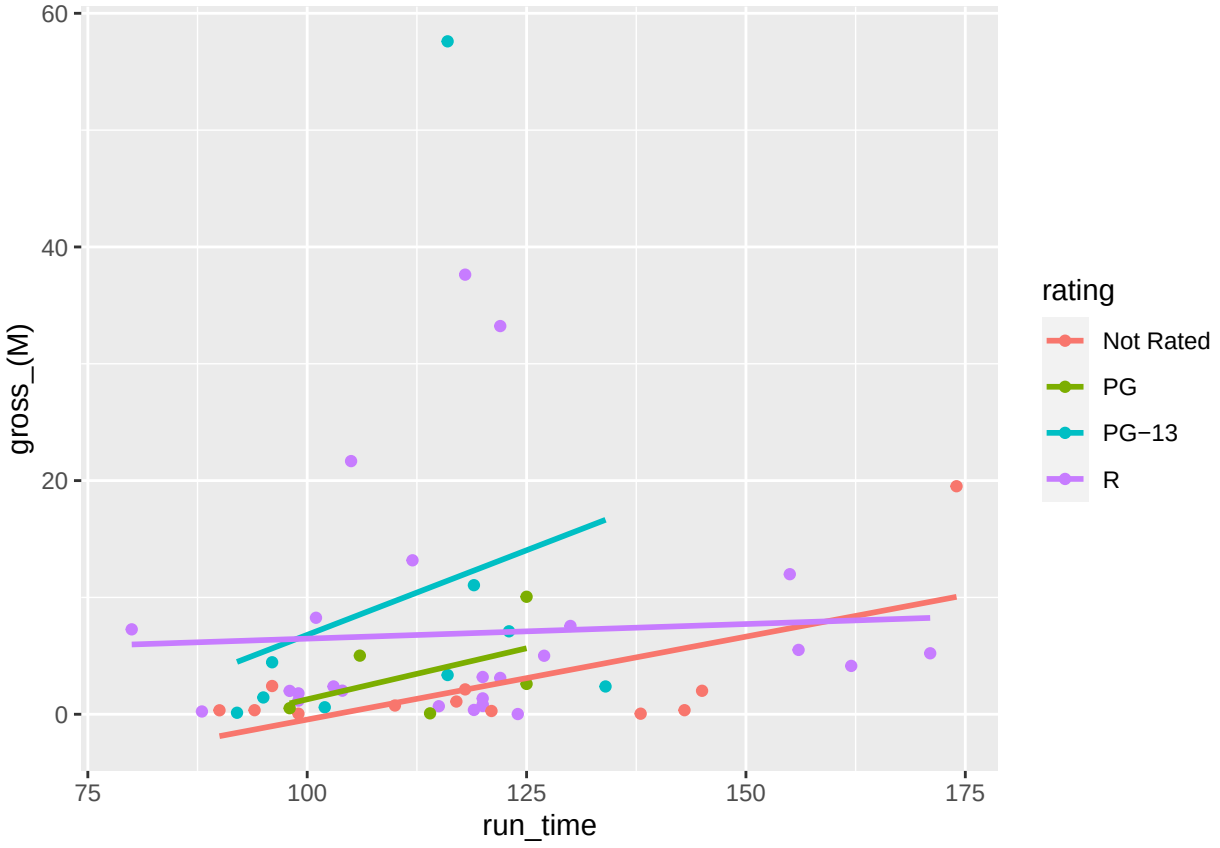
```
movie_df %>% group_by(rating) %>%
 count() %>%
 filter(n < 4)
```

```
## # A tibble: 7 x 2
## # Groups:   rating [7]
##   rating      n
##   <chr>   <int>
## 1 G           3
## 2 GP          1
## 3 Passed      1
## 4 TV-14       1
## 5 TV-MA       3
## 6 TV-PG       3
## 7 Unrated     1
```

```
 # Will only include Not Rated, PG, PG-13, and R

movie_df %>%
  filter(rating == "Not Rated" | rating == "PG" | rating == "PG-13" | rating == "R") %>%
  ggplot(aes(run_time, `gross_(M)`, color = rating)) +
  geom_point() +
  geom_smooth(method = "lm", se = F)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



- b. Interpret the plot to answer the question: **for which MPAA ratings (if any) is there a positive or negative relationship between length of a film and its gross revenue?**

There seems to be a positive correlation between run time and gross earning across all genres except R rated films. However, exemplified from the table below, there is very little observations for each rating. Thus, it is difficult to draw any concrete conclusions.

```
movie_df %>%
  filter(rating == "Not Rated" | rating == "PG" | rating == "PG-13" | rating == "R") %>%
  group_by(rating) %>%
  count()
```

```
## # A tibble: 4 x 2
## # Groups:   rating [4]
##   rating        n
##   <chr>     <int>
## 1 Not Rated    27
## 2 PG            6
## 3 PG-13        12
## 4 R            27
```

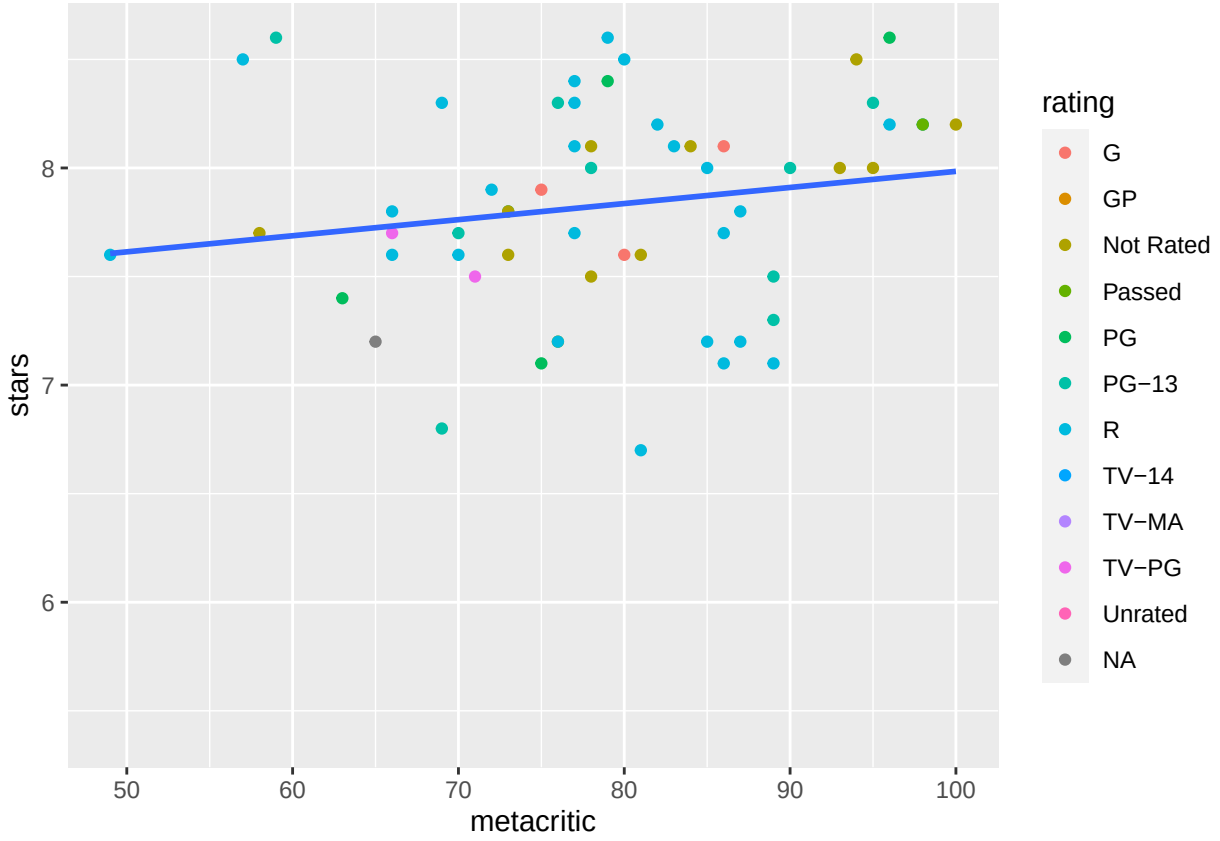
6.

- a. Create a plot of `stars` versus `metacritic score`, color coded by MPAA rating (i.e., for predicting stars rating based on meta-critic scores).

- b. Include a single Ordinary Least Squares smoothing line with no standard errors and interpret the plot.

```
movie_df %>%
  ggplot(aes(metacritic, stars)) +
  geom_point(aes(color = rating)) +
  geom_smooth(method = "lm", se = F)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



There seems to be a slight positive linear relationship between the metacritic score and the star rating of the films.

- c. Use a linear model to assess if there is there a meaningful relationship.
  - Show the summary of the output and interpret in terms of the p-value and the adjusted R-Squared.
  - Explain why you are surprised at the result (or not) based on the plot.

```
lm <- lm(stars ~ metacritic, data = movie_df)
summary(lm)
```

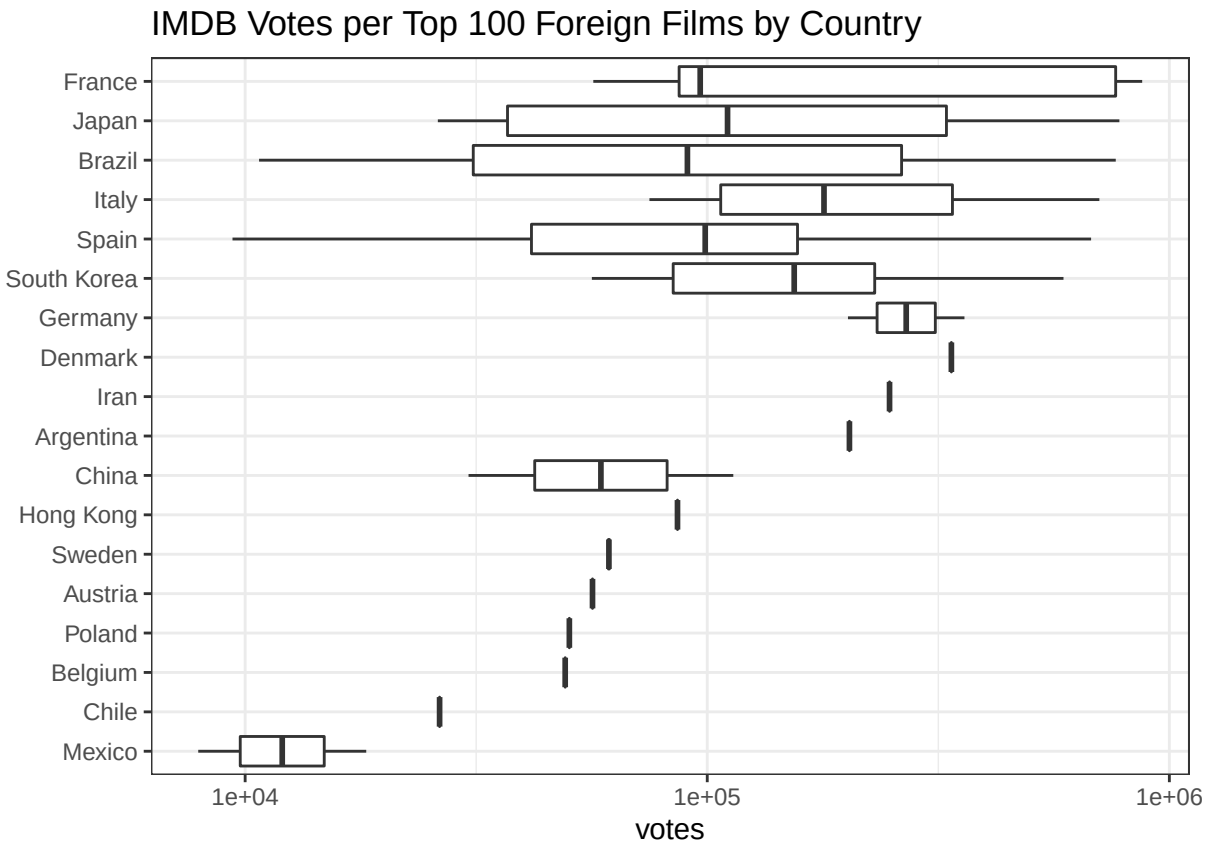
```
##
## Call:
## lm(formula = stars ~ metacritic, data = movie_df)
```

```
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.14352 -0.25645  0.05269  0.26200  0.91959
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 7.242975   0.437329  16.562   <2e-16 ***
## metacritic  0.007414   0.005480   1.353    0.181
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4697 on 57 degrees of freedom
##   (41 observations deleted due to missingness)
## Multiple R-squared:  0.03111,    Adjusted R-squared:  0.01411
## F-statistic:  1.83 on 1 and 57 DF,  p-value: 0.1814
```

From the linear model summary, with a p-value of .375, there is no linear relationship between the metacritic score and the star rating for these foreign films. It is a bit surprising given the topic of the list being top 100 foreign films. I wonder if this highlights the discrepancies in standards that film critics hold films to compared to the general public. However, with an adjusted R-squared value of .014, it is clear that additional variables account for a more accurate prediction of stars.

8. Create a plot that shows which country has the most votes in the

```
# Not sure why Japan would not reorder correctly when the remaining order of countries by max value is
movie_df %>%
  na.omit() %>%
  ggplot(aes(x = fct_reorder(country, votes, max), y = votes)) +
  geom_boxplot() +
  coord_flip() +
  ggtitle("IMDB Votes per Top 100 Foreign Films by Country") +
  xlab("") +
  scale_y_log10() +
  theme_bw()
```



The countries with higher counts of film are not very surprising to me as I feel I have heard of popular films coming from these countries through analysis of Cannes festivals etc. There seems to be quite a bit of diversity in votes across geographic regions.