DataSci 306 Final Project

Hania Timek, Shaurya Pratap Singh, Aden Tao

2025-04-28

- Hania Timek positcloud has the r script files for the applications in part 6
- Shaurya Pratap Singh has the r script files for the applications in extra credit. Under the folder "application3"

Investigating the Internet Movie Database (IMDB)

The Internet Movie Database (IMDb) contains information on millions of movies and television programs. They offer several non-commercial use datasets (documentation link). For this project we will analyze a **sample** of 100,000 titles from the IMDBb.

Part I: Preprocessing

- Edit your `.gitignore` file to ignore all files with the `.rda` extension. (Add and commit)
- Create a new file in the `data/` directory called "Preprocessing.Rmd". The remaining instructions in this section are to be completed in that file.
- Write a function that will load a table from the IMDb files in the `data/` directory.
  - The function should take the file name (without the ".csv.gz" portion) as an argument
  - The function should load the appropriate `.csv.gz` file.
  - Make sure that all "\N" values (which IMDB uses to indicate missing values) are turned into proper NA values in R
  - The function should return the table.
- For each of the `.csv.gz` files, use your function to load the table, then save it into a variable (e.g. `name_basics <- preprocess("name_basics")`) and use the `write_rds` function (e.g., `write_rds(name_basics, "name_basics.rda")`.
- Run the function on all of the `*_sample.csv.gz` files to created processed `.rda` files.
- In your other files, you can load these using the `TABLE <- read_rds("data/FILENAME.rda")` function.

```
name_basics <- read_rds("data/name_basics.rda")
title_basics <- read_rds("data/title_basics.rda")
title_principals <- read_rds("data/title_principals.rda")
title_ratings <- read_rds("data/title_ratings.rda")
```

Part II: EDA of individual tables (aden)

- For each of the 4 tables, perform basic exploratory data analysis. Report the following information:
  - For each quantitative column, provide some summary statistics

- For any character columns, decided if they are actually representing factors/categorical data with a moderate number of columns. If so report the distributions for these variables.

```r
library(dplyr)
library(tidyr)

eda_summary <- function(df, df_name) {
  cat("\n============================\n")
  cat("Summary for", df_name, "\n")
  cat("============================\n")

  # Quantitative Columns
  num_cols <- sapply(df, is.numeric)
  if (any(num_cols)) {
    cat("\nQuantitative Columns Summary:\n")
    print(summary(df[, num_cols]))
  } else {
    cat("\nNo Quantitative Columns.\n")
  }

  # Character Columns
  char_cols <- sapply(df, is.character)
  if (any(char_cols)) {
    cat("\nCharacter Columns (Checking if Categorical):\n")
    for (colname in names(df)[char_cols]) {
      # Special case: if the column has commas (like genres), split it first
      if (any(grepl(",", df[[colname]], fixed = TRUE))) {
        split_values <- unlist(strsplit(df[[colname]], ","))
        split_values <- trimws(split_values)  # remove spaces
        n_unique <- length(unique(split_values))

        cat("\n", colname, "(split on commas) - Unique Values:", n_unique, "\n")

        if (n_unique <= 50) {
          cat("Likely Categorical after splitting. Distribution:\n")
          print(table(split_values))
        } else {
          cat("Not categorical (too many unique values after splitting).\n")
        }

      } else {
        n_unique <- n_distinct(df[[colname]])
        cat("\n", colname, "- Unique Values:", n_unique, "\n")

        if (n_unique <= 30) {
          cat("Likely Categorical. Distribution:\n")
          print(table(df[[colname]]))
        } else {
          cat("Not categorical (too many unique values).\n")
        }
      }
    }
  } else {
    cat("\nNo Character Columns.\n")
```

```
  }
}


# Run EDA for all 4 tables
eda_summary(name_basics, "name_basics")
```

```
##
## ============================
## Summary for name_basics
## ============================
##
## Quantitative Columns Summary:
##    birthYear        deathYear
##  Min.   :  37     Min.   :  44
##  1st Qu.:1933     1st Qu.:1980
##  Median :1959     Median :2000
##  Mean   :1953     Mean   :1994
##  3rd Qu.:1976     3rd Qu.:2014
##  Max.   :2021     Max.   :2024
##  NA's   :337769   NA's   :446685
##
## Character Columns (Checking if Categorical):
##
##  nconst - Unique Values: 503722
## Not categorical (too many unique values).
##
##  primaryName (split on commas) - Unique Values: 485506
## Not categorical (too many unique values after splitting).
##
##  primaryProfession (split on commas) - Unique Values: 47
## Likely Categorical after splitting. Distribution:
## split_values
##               accountant                    actor                  actress
##                        2                   183708                   109520
##     animation_department          archive_footage             archive_sound
##                     4824                    38099                     1412
##           art_department             art_director                assistant
##                    12678                     6817                       34
##       assistant_director        camera_department       casting_department
##                    14195                    25789                     5929
##         casting_director           choreographer          cinematographer
##                     6640                       43                    30567
##                 composer       costume_department         costume_designer
##                    28332                     1714                     1560
##                 director                   editor     editorial_department
##                    84248                    35502                    17671
##    electrical_department                executive                    legal
##                        2                     2245                       86
##      location_management       make_up_department                  manager
##                     1666                     1888                      537
##            miscellaneous             music_artist         music_department
##                    48691                     1105                    21383
```

```
##                 podcaster                   producer     production_department
##                        55                     112225                         22
##       production_designer        production_manager                  publicist
##                     10909                      10160                         86
##         script_department             set_decorator          sound_department
##                      5332                       2023                      10256
##                soundtrack           special_effects                     stunts
##                     24849                       1445                       5034
##             talent_agent transportation_department            visual_effects
##                      421                        580                       4510
##                    writer
##                    113871
##
##  knownForTitles (split on commas) - Unique Values: 535085
## Not categorical (too many unique values after splitting).
```

```
eda_summary(title_basics, "title_basics")
```

```
##
## ============================
## Summary for title_basics
## ============================
##
## Quantitative Columns Summary:
##     isAdult            startYear       endYear      runtimeMinutes
##  Min.   :0.000e+00   Min.   :1887   Min.   :1938    Min.   :   1.00
##  1st Qu.:0.000e+00   1st Qu.:1997   1st Qu.:2001    1st Qu.:  23.00
##  Median :0.000e+00   Median :2011   Median :2013    Median :  45.00
##  Mean   :3.602e-02   Mean   :2003   Mean   :2008    Mean   :  55.35
##  3rd Qu.:0.000e+00   3rd Qu.:2018   3rd Qu.:2019    3rd Qu.:  85.00
##  Max.   :2.020e+03   Max.   :2025   Max.   :2025    Max.   :5220.00
##                      NA's   :17     NA's   :96408   NA's   :29696
##
## Character Columns (Checking if Categorical):
##
##  tconst - Unique Values: 100000
## Not categorical (too many unique values).
##
##  titleType - Unique Values: 10
## Likely Categorical. Distribution:
##
##        movie        short    tvEpisode tvMiniSeries      tvMovie     tvSeries
##        21467        11118        50194         1127         3703         6555
##      tvShort    tvSpecial        video    videoGame
##          161          837         3617         1221
##
##  primaryTitle (split on commas) - Unique Values: 91688
## Not categorical (too many unique values after splitting).
##
##  originalTitle (split on commas) - Unique Values: 92347
## Not categorical (too many unique values after splitting).
##
##  genres (split on commas) - Unique Values: 30
## Likely Categorical after splitting. Distribution:
```

```
## split_values
##      Action       Adult   Adventure   Animation   Biography      Comedy
##       12582        1534       11135       11813        2120       30899
##       Crime Documentary       Drama      Family     Fantasy   Film-Noir
##       10816       13043       33788        7474        4030          59
##   Game-Show     History      Horror       Music     Musical     Mystery
##        2489        2898        4059        3216         947        4826
##          NA        News  Reality-TV     Romance      Sci-Fi       Short
##           1        1405        5391        7650        2512       11787
##       Sport   Talk-Show    Thriller         War     Western
##        2089        2880        4223        1092        1167
```

```
eda_summary(title_principals, "title_principals")
```

```
##
## ============================
## Summary for title_principals
## ============================
##
## Quantitative Columns Summary:
##     ordering
##  Min.   : 1.00
##  1st Qu.: 4.00
##  Median : 8.00
##  Mean   : 9.06
##  3rd Qu.:13.00
##  Max.   :62.00
##
## Character Columns (Checking if Categorical):
##
##  tconst - Unique Values: 95979
## Not categorical (too many unique values).
##
##  nconst - Unique Values: 503731
## Not categorical (too many unique values).
##
##  category - Unique Values: 13
## Likely Categorical. Distribution:
##
##               actor             actress     archive_footage       archive_sound
##              406747              242512                7935                 267
##    casting_director     cinematographer            composer            director
##               32853               60902               61829               87017
##              editor            producer production_designer                self
##               77280               98111               28164              111477
##              writer
##              146370
##
##  job (split on commas) - Unique Values: 4358
## Not categorical (too many unique values after splitting).
##
##  characters (split on commas) - Unique Values: 295471
## Not categorical (too many unique values after splitting).
```

```r
eda_summary(title_ratings, "title_ratings")
```

```
##
## ============================
## Summary for title_ratings
## ============================
##
## Quantitative Columns Summary:
##  averageRating       numVotes
##  Min.   : 1.000   Min.   :      5.0
##  1st Qu.: 6.200   1st Qu.:     11.0
##  Median : 7.200   Median :     26.0
##  Mean   : 6.963   Mean   :    979.9
##  3rd Qu.: 7.900   3rd Qu.:    101.0
##  Max.   :10.000   Max.   :2279226.0
##
## Character Columns (Checking if Categorical):
##
##  tconst - Unique Values: 100000
## Not categorical (too many unique values).
```

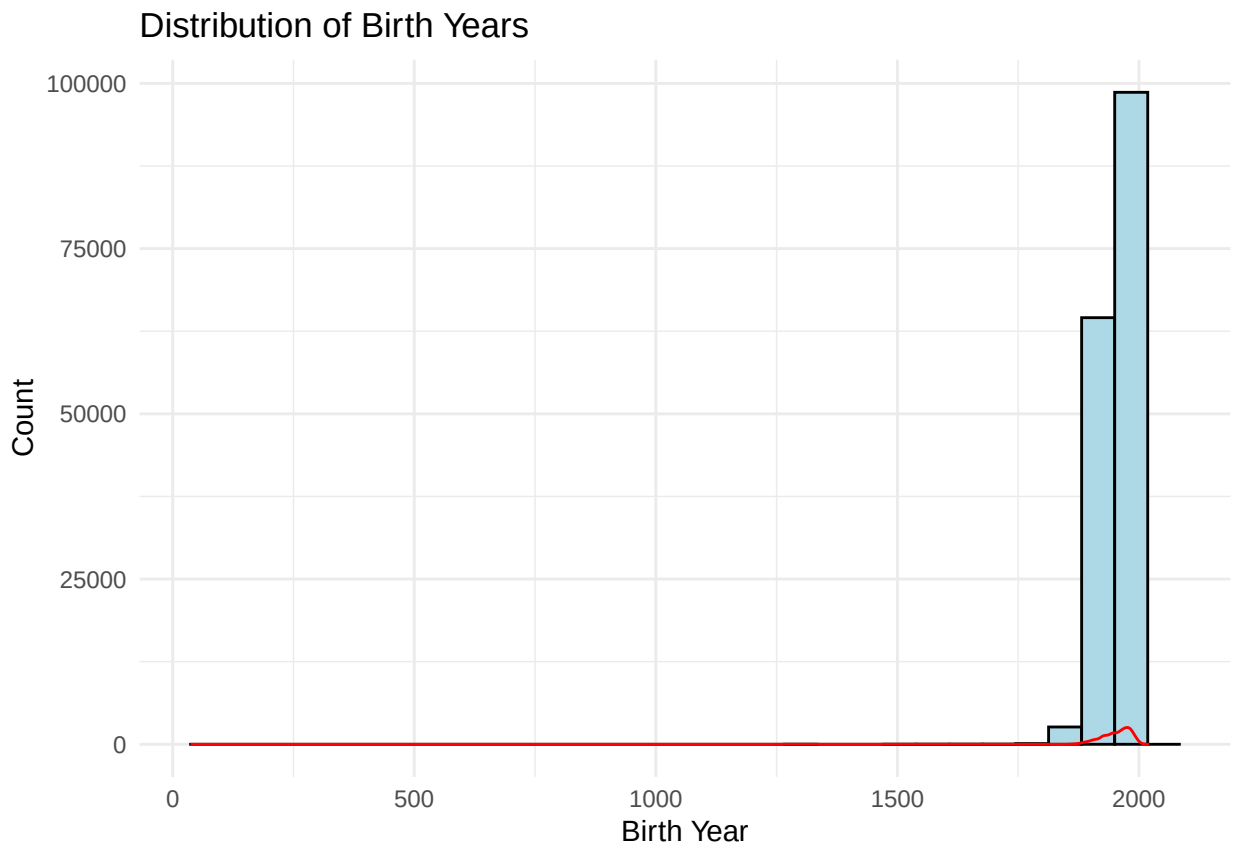
- Provide a plot for each table. Across all of the plots, try to show off the most possible different ggplot features (`geoms_` functions, `stat_` functions, coordinate systems, facets, use of several variables, annotations)

```r
ggplot(name_basics, aes(x = birthYear)) +
  geom_histogram(bins = 30, fill = "lightblue", color = "black") +
  geom_density(aes(y = ..count..), color = "red") +
  labs(title = "Distribution of Birth Years", x = "Birth Year", y = "Count") +
  theme_minimal()
```

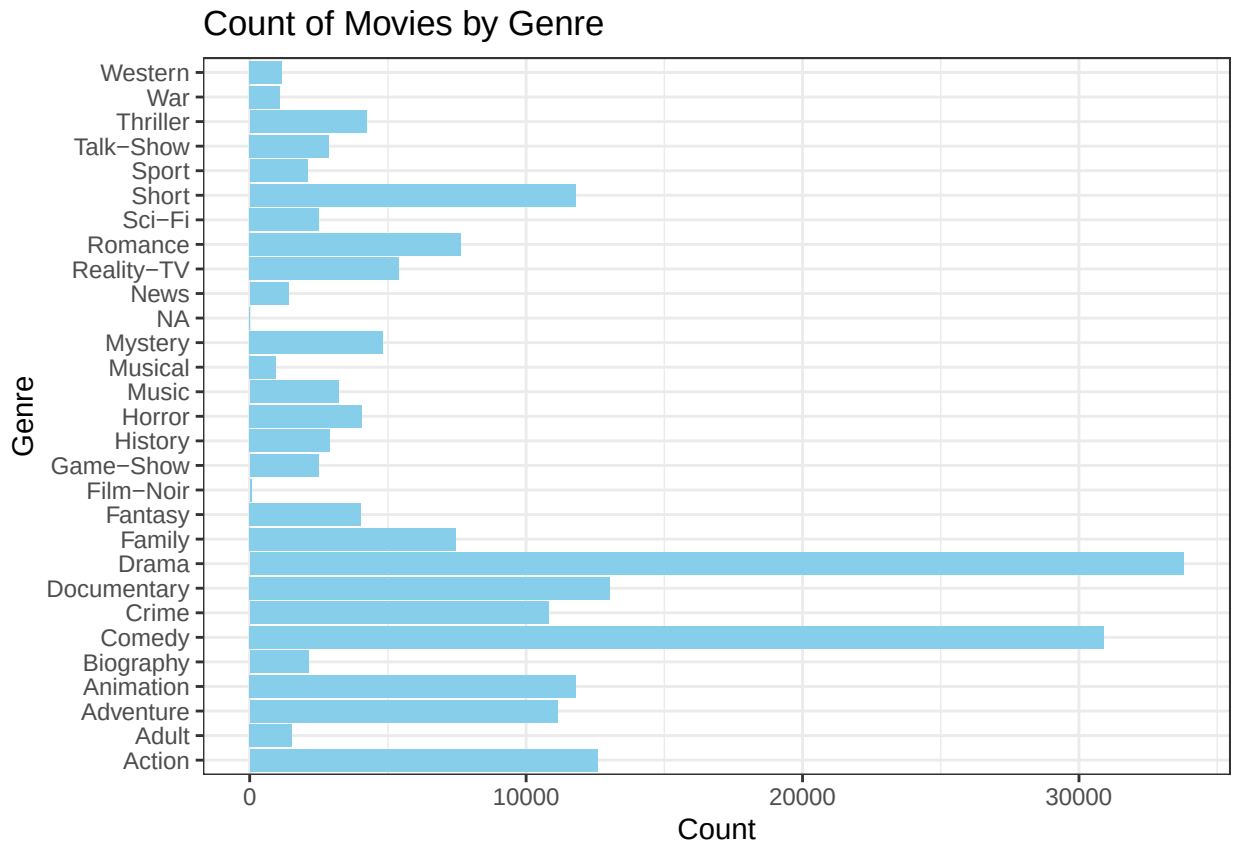
```
## Warning: The dot-dot notation (`..count..`) was deprecated in ggplot2 3.4.0.
## i Please use `after_stat(count)` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
## Warning: Removed 337769 rows containing non-finite outside the scale range
## (`stat_bin()`).
```

```
## Warning: Removed 337769 rows containing non-finite outside the scale range
## (`stat_density()`).
```




```
title_basics_long <- title_basics %>%
  tidyr::separate_rows(genres, sep = ",") %>%
  filter(!is.na(genres))

ggplot(title_basics_long, aes(x = genres)) +
  geom_bar(fill = "skyblue") +
  coord_flip() +
  labs(title = "Count of Movies by Genre", x = "Genre", y = "Count") +
  theme_bw()
```

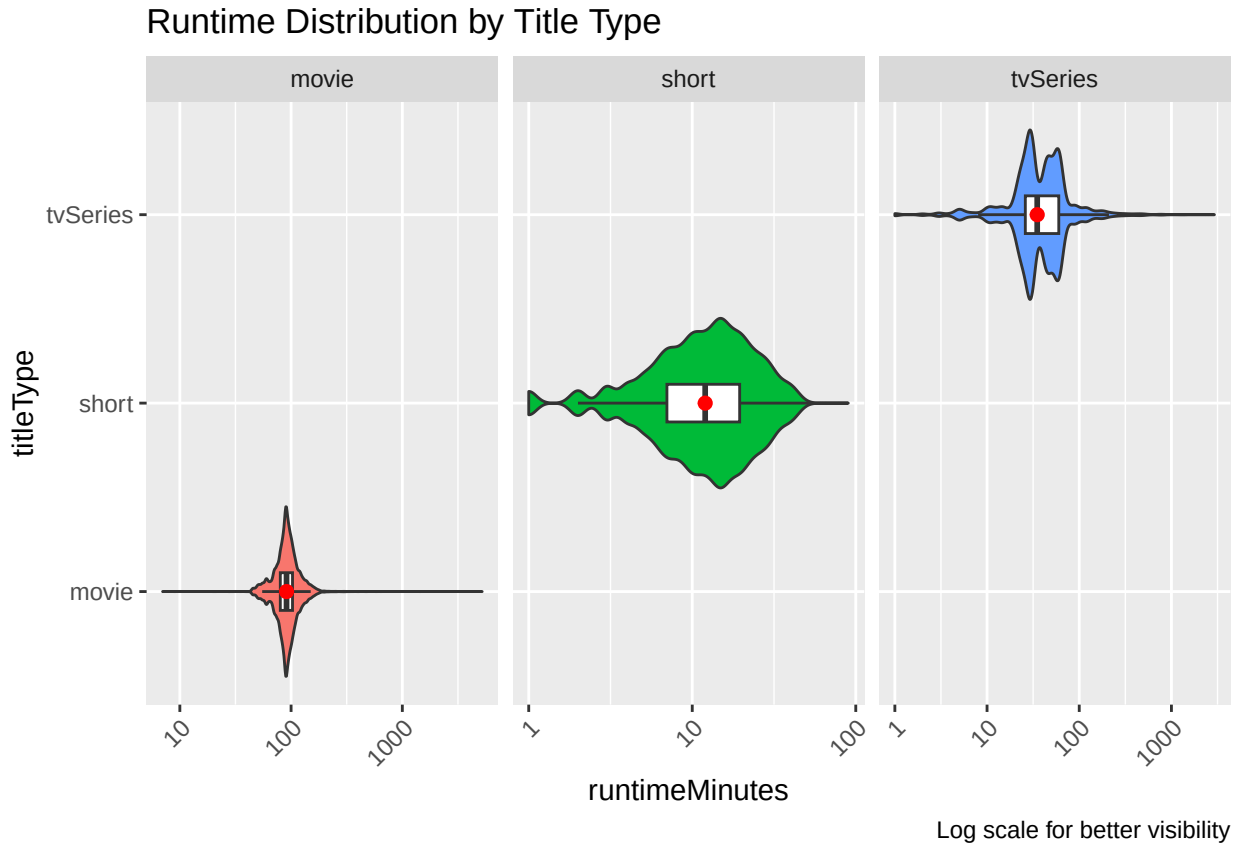



```
title_plot <- title_basics %>%
  filter(titleType %in% c("movie", "tvSeries", "short")) %>%
  mutate(runtimeMinutes = as.numeric(runtimeMinutes)) %>%
  ggplot(aes(titleType, runtimeMinutes)) +
  geom_violin(aes(fill = titleType), show.legend = FALSE) +
  geom_boxplot(width = 0.2, outlier.shape = NA) +
  stat_summary(fun = "median", geom = "point", size = 2, color = "red") +
  scale_y_log10() +
  facet_wrap(~titleType, scales = "free_x") +
  labs(title = "Runtime Distribution by Title Type",
       caption = "Log scale for better visibility") +
  coord_flip() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

title_plot
```

```
## Warning: Removed 6562 rows containing non-finite outside the scale range
## (`stat_ydensity()`).
```

```
## Warning: Removed 6562 rows containing non-finite outside the scale range
## (`stat_boxplot()`).
```

```
## Warning: Removed 6562 rows containing non-finite outside the scale range
## (`stat_summary()`).
```

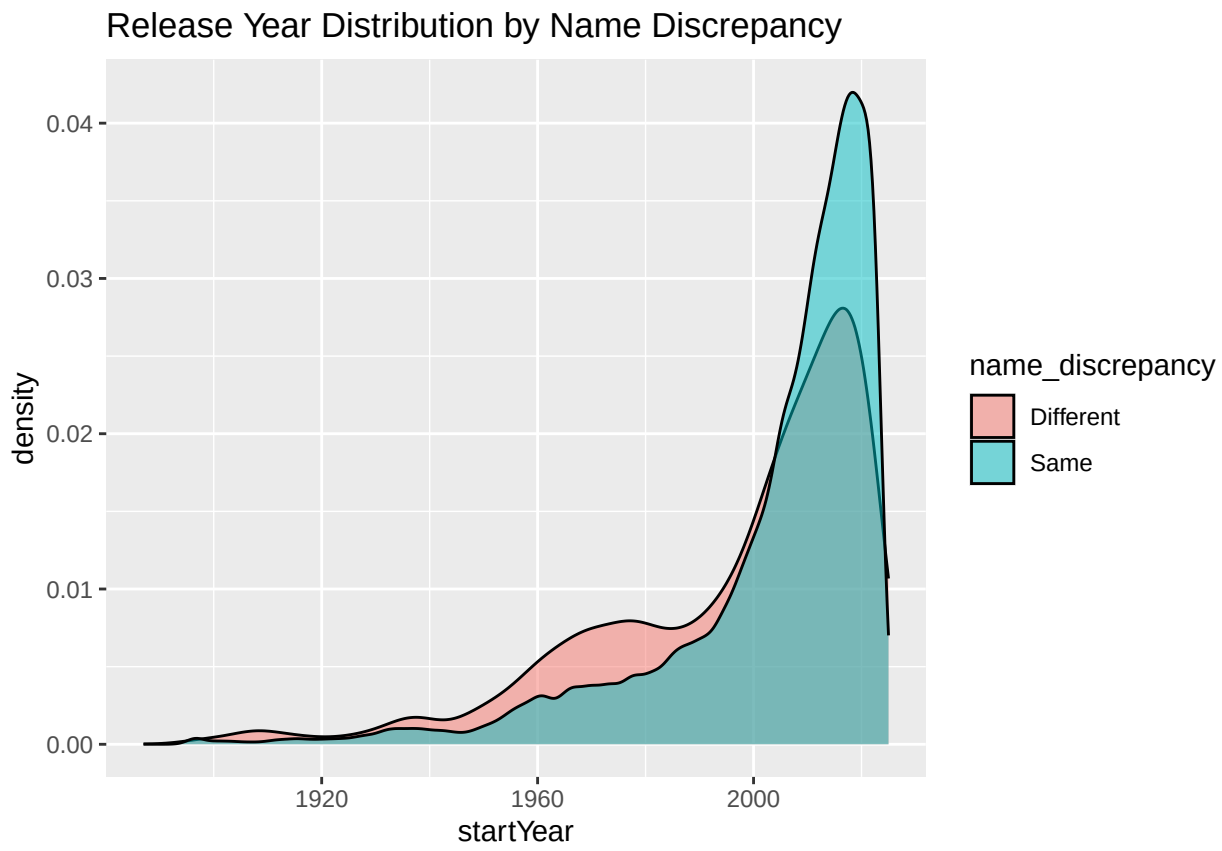



```
title_basics %>%
  filter(primaryTitle != originalTitle) %>%
  count()
```

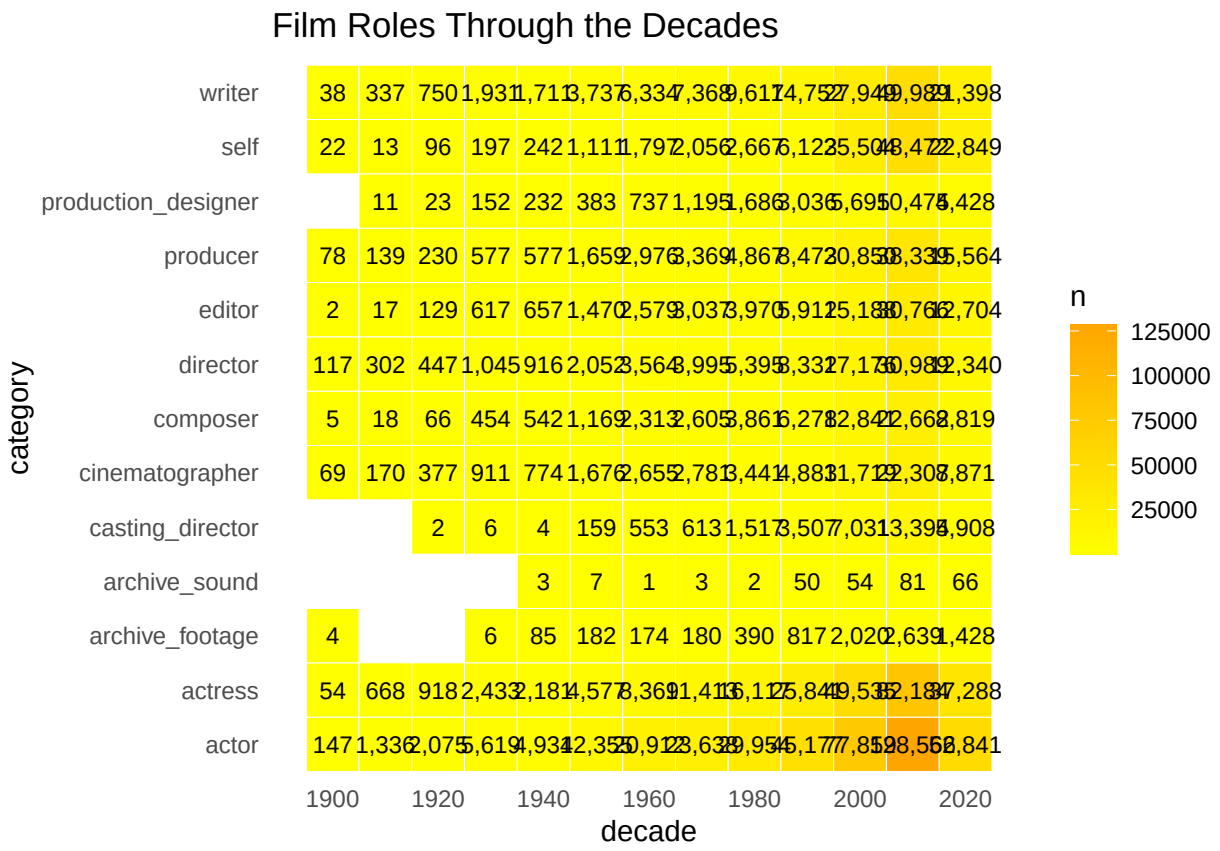
```
## # A tibble: 1 x 1
##       n
##   <int>
## 1  7244
```

```
title_basics %>%
  mutate(name_discrepancy = ifelse(primaryTitle != originalTitle, "Different", "Same")) %>%
  ggplot(aes(x = startYear, fill = name_discrepancy)) +
  geom_density(alpha = 0.5) +
  labs(title = "Release Year Distribution by Name Discrepancy")
```

```
## Warning: Removed 17 rows containing non-finite outside the scale range
## (`stat_density()`).
```



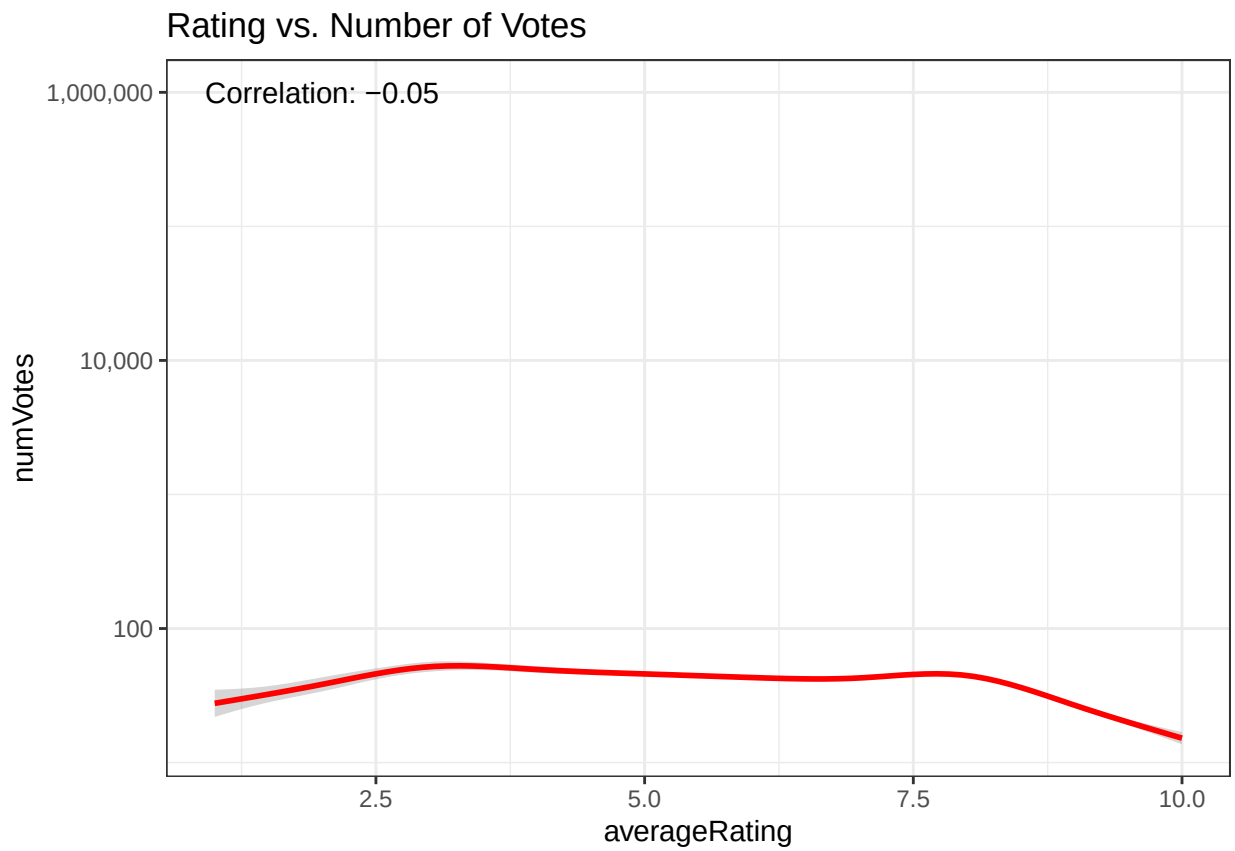

```
principals_plot <- title_principals %>%
  left_join(title_basics %>% select(tconst, startYear), by = "tconst") %>%
  filter(startYear >= 1900, startYear <= 2023) %>%
  mutate(decade = floor(startYear/10)*10) %>%
  count(decade, category) %>%
  ggplot(aes(decade, category)) +
  geom_tile(aes(fill = n), color = "white") +
  geom_text(aes(label = scales::comma(n)), color = "black", size = 3) +
  scale_fill_gradient(low = "yellow", high = "orange") +
  scale_x_continuous(breaks = seq(1900, 2020, 20)) +
  labs(title = "Film Roles Through the Decades",) +
  theme_minimal() +
  theme(panel.grid = element_blank())

principals_plot
```




```
ratings_plot <- title_ratings %>%
  ggplot(aes(averageRating, numVotes)) +
  geom_hex(bins = 50) +
  geom_smooth(method = "gam", color = "red") +
  annotate("text", x = 2, y = 1e6,
           label = paste("Correlation:",
                         round(cor(title_ratings$averageRating,
                                   log(title_ratings$numVotes)), 2))) +  # Explicit reference
  scale_y_log10(labels = scales::comma) +
  scale_fill_viridis_c(option = "magma") +
  labs(title = "Rating vs. Number of Votes") +
  theme_bw()

ratings_plot
```

```
## Warning: Computation failed in `stat_binhex()`.
## Caused by error in `compute_group()`:
## ! The package "hexbin" is required for `stat_bin_hex()`.
```

```
## `geom_smooth()` using formula = 'y ~ s(x, bs = "cs")'
```



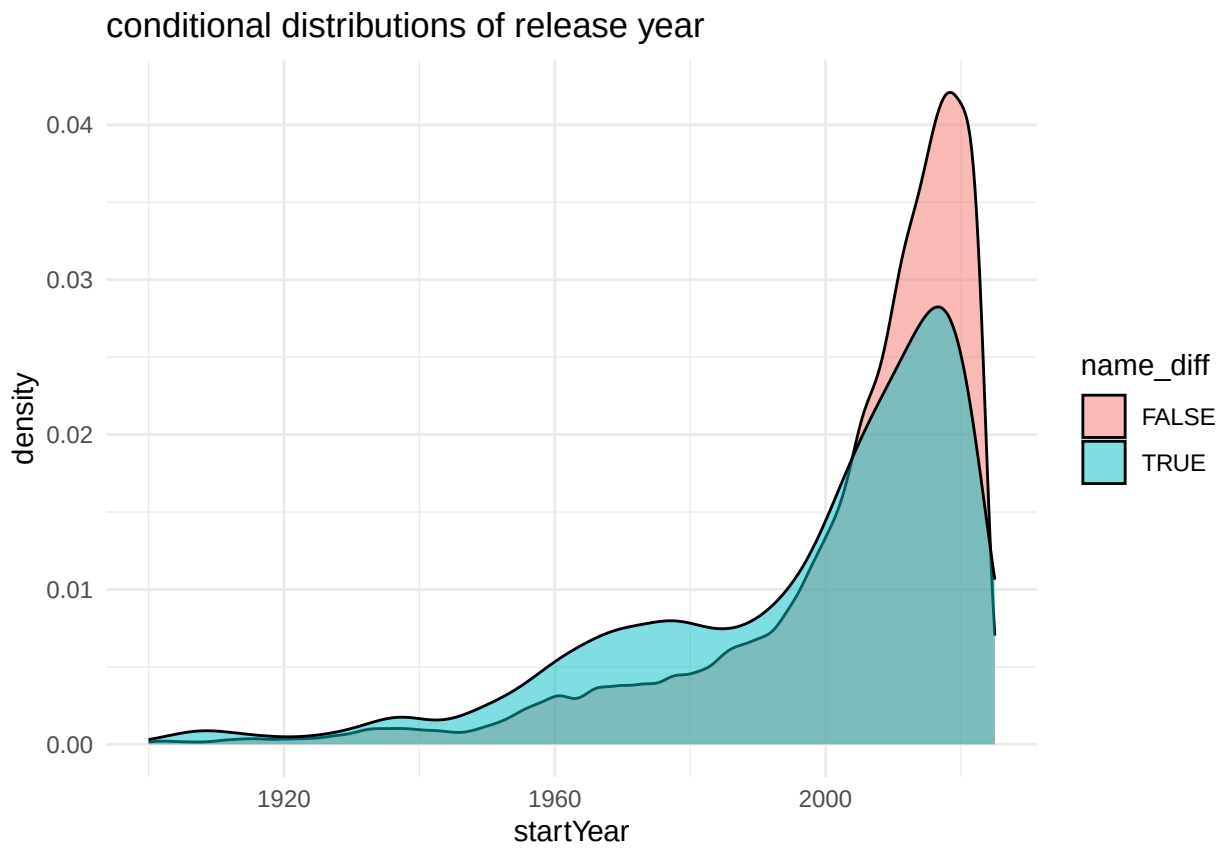
* How many titles are known for name that is different than the original release name?

```
count <- sum(title_basics$primaryTitle != title_basics$originalTitle, na.rm = TRUE)
count
```

```
## [1] 7244
```

- Graph the conditional distributions of release year based on the previous results.

```
title_basics %>%
  mutate(
    name_diff = primaryTitle != originalTitle,
    startYear = as.numeric(startYear)
  ) %>%
  filter(!is.na(startYear), startYear >= 1900) %>%
  ggplot(aes(startYear, fill = name_diff)) +
  geom_density(alpha = 0.5) +
  labs(title = "conditional distributions of release year") +
  theme_minimal()
```



Comment on any trends you observe.

Titles with different names are more common in recent decades and this suggests increasing rebranding or localization efforts in modern media.

- For the ratings, use the `cut` function to break the data into three groups based on the average ratings. Are higher rated titles rated more often or less often than lower rated titles?

```
ratings_group <- title_ratings %>%
  mutate(
    rating_group = cut(
      averageRating,
      breaks = c(0, 5, 7, 10),
      labels = c("Low (0-5)", "Medium (5-7)", "High (7-10)")
    )
  ) %>%
  group_by(rating_group) %>%
  summarise(median_votes = median(numVotes, na.rm = TRUE))

ratings_group
```

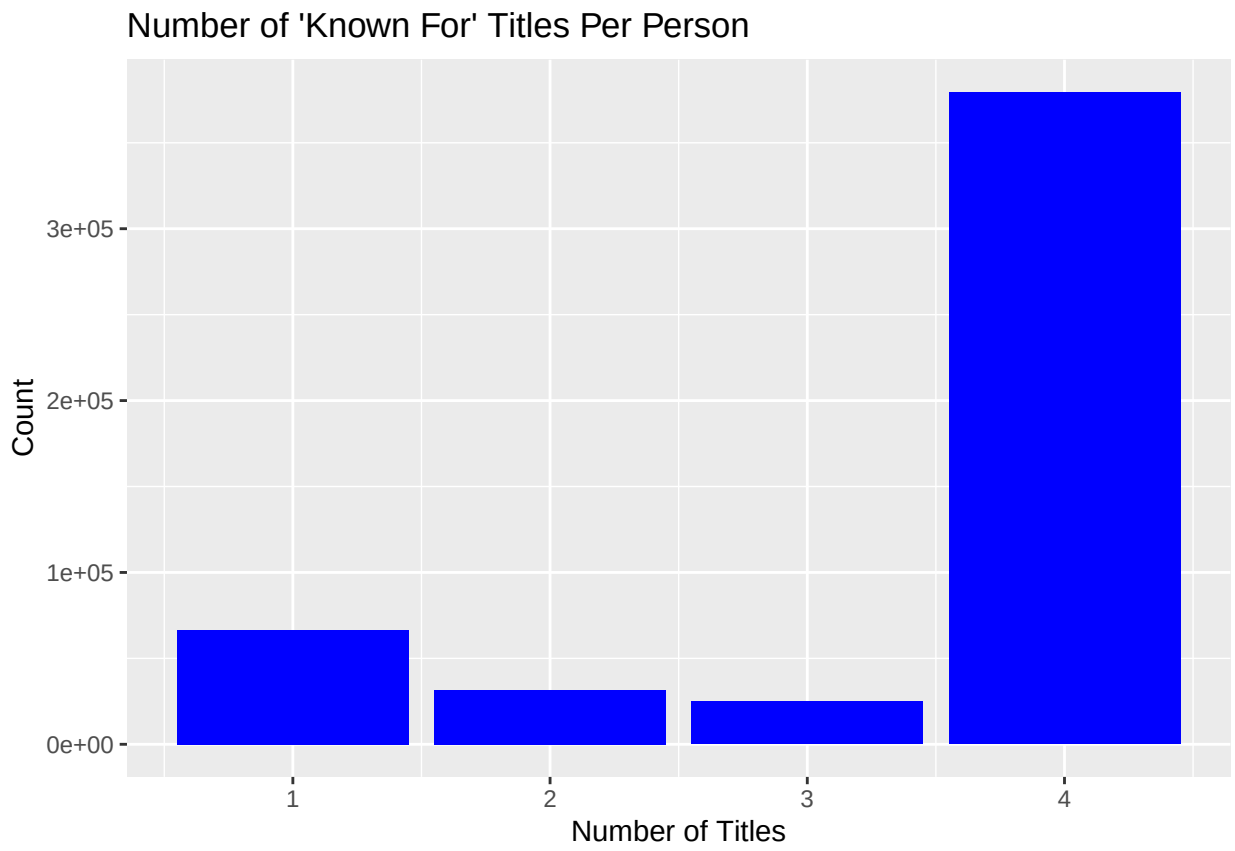
```
## # A tibble: 3 x 2
##   rating_group median_votes
##   <fct>               <dbl>
## 1 Low (0-5)              28
## 2 Medium (5-7)           26
## 3 High (7-10)            25
```

- For the names table,
  - Count the number of titles each person is known for and plot this distribution.

```
name_counts <- name_basics %>%
  mutate(num_titles = lengths(strsplit(knownForTitles, ","))) %>%
  filter(!is.na(knownForTitles))

(name_counts)
```

```
## # A tibble: 502,314 x 7
##    nconst    primaryName    birthYear deathYear primaryProfession knownForTitles
##    <chr>     <chr>              <dbl>     <dbl> <chr>             <chr>
##  1 nm0000001 Fred Astaire        1899      1987 actor,miscellane~ tt0072308,tt0~
##  2 nm0000002 Lauren Bacall       1924      2014 actress,soundtra~ tt0037382,tt0~
##  3 nm0000003 Brigitte Bard~      1934        NA actress,music_de~ tt0057345,tt0~
##  4 nm0000004 John Belushi        1949      1982 actor,writer,mus~ tt0072562,tt0~
##  5 nm0000005 Ingmar Bergman      1918      2007 writer,director,~ tt0050986,tt0~
##  6 nm0000006 Ingrid Bergman      1915      1982 actress,producer~ tt0034583,tt0~
##  7 nm0000007 Humphrey Boga~      1899      1957 actor,producer,m~ tt0034583,tt0~
##  8 nm0000008 Marlon Brando       1924      2004 actor,director,w~ tt0078788,tt0~
##  9 nm0000009 Richard Burton      1925      1984 actor,producer,d~ tt0061184,tt0~
## 10 nm0000010 James Cagney        1899      1986 actor,director,p~ tt0029870,tt0~
## # i 502,304 more rows
## # i 1 more variable: num_titles <int>
```

```
name_counts %>%
  ggplot(aes(num_titles)) +
  geom_bar(fill = "blue") +
  labs(
    title = "Number of 'Known For' Titles Per Person",
    x = "Number of Titles",
    y = "Count"
  )
```



- investigate the age of cast members
  - Group the data into living and deceased cast members.

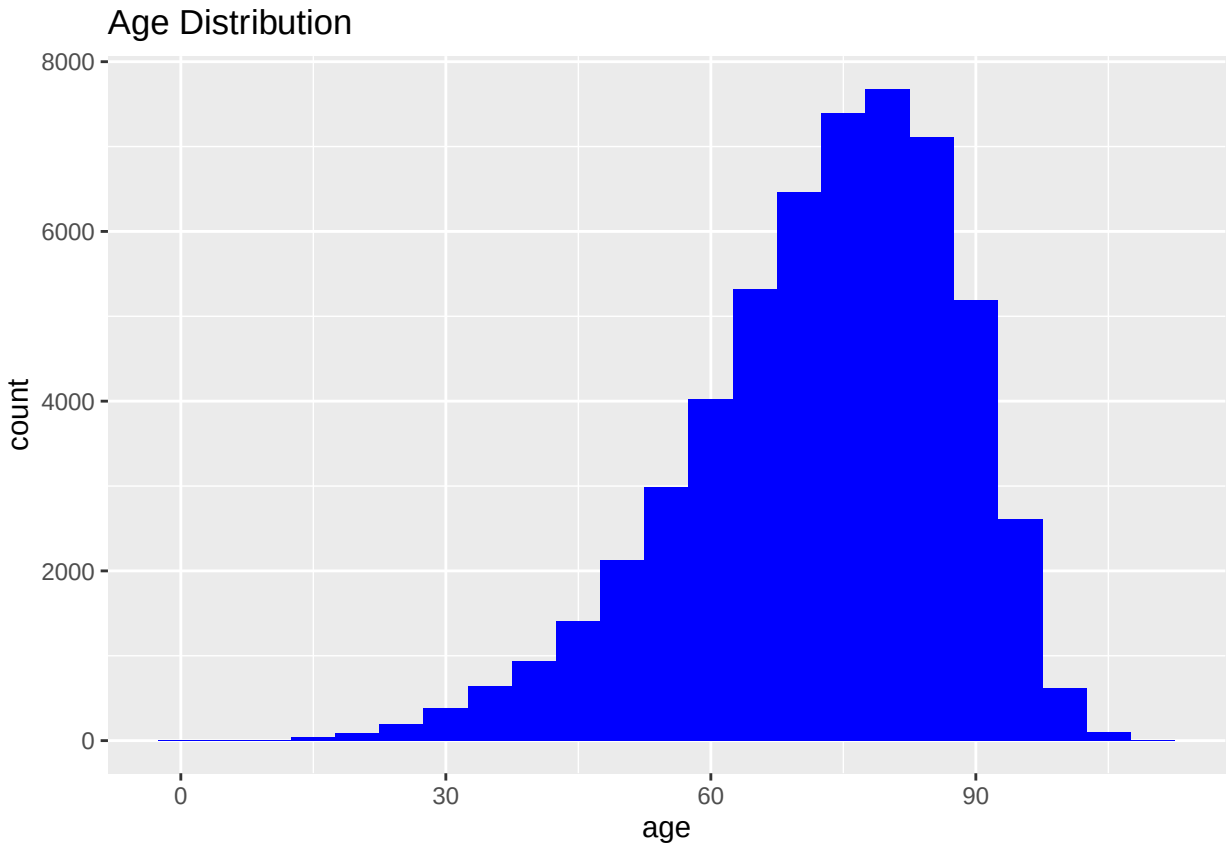
```
name_ages <- name_basics %>%
  mutate(
    status = ifelse(is.na(deathYear), "Living", "Deceased"),
    age = ifelse(
      status == "Deceased",
      deathYear - birthYear,
      2025 - birthYear
    )
  ) %>%
  filter(age > 0, age < 120)

name_ages
```

```
## # A tibble: 165,661 x 8
##    nconst    primaryName    birthYear deathYear primaryProfession knownForTitles
##    <chr>     <chr>              <dbl>     <dbl> <chr>             <chr>
##  1 nm0000001 Fred Astaire        1899      1987 actor,miscellane~ tt0072308,tt0~
##  2 nm0000002 Lauren Bacall       1924      2014 actress,soundtra~ tt0037382,tt0~
##  3 nm0000003 Brigitte Bard~      1934        NA actress,music_de~ tt0057345,tt0~
##  4 nm0000004 John Belushi        1949      1982 actor,writer,mus~ tt0072562,tt0~
##  5 nm0000005 Ingmar Bergman      1918      2007 writer,director,~ tt0050986,tt0~
```

```
##  6 nm0000006 Ingrid Bergman      1915      1982 actress,producer~ tt0034583,tt0~
##  7 nm0000007 Humphrey Boga~      1899      1957 actor,producer,m~ tt0034583,tt0~
##  8 nm0000008 Marlon Brando       1924      2004 actor,director,w~ tt0078788,tt0~
##  9 nm0000009 Richard Burton      1925      1984 actor,producer,d~ tt0061184,tt0~
## 10 nm0000010 James Cagney        1899      1986 actor,director,p~ tt0029870,tt0~
## # i 165,651 more rows
## # i 2 more variables: status <chr>, age <dbl>
```

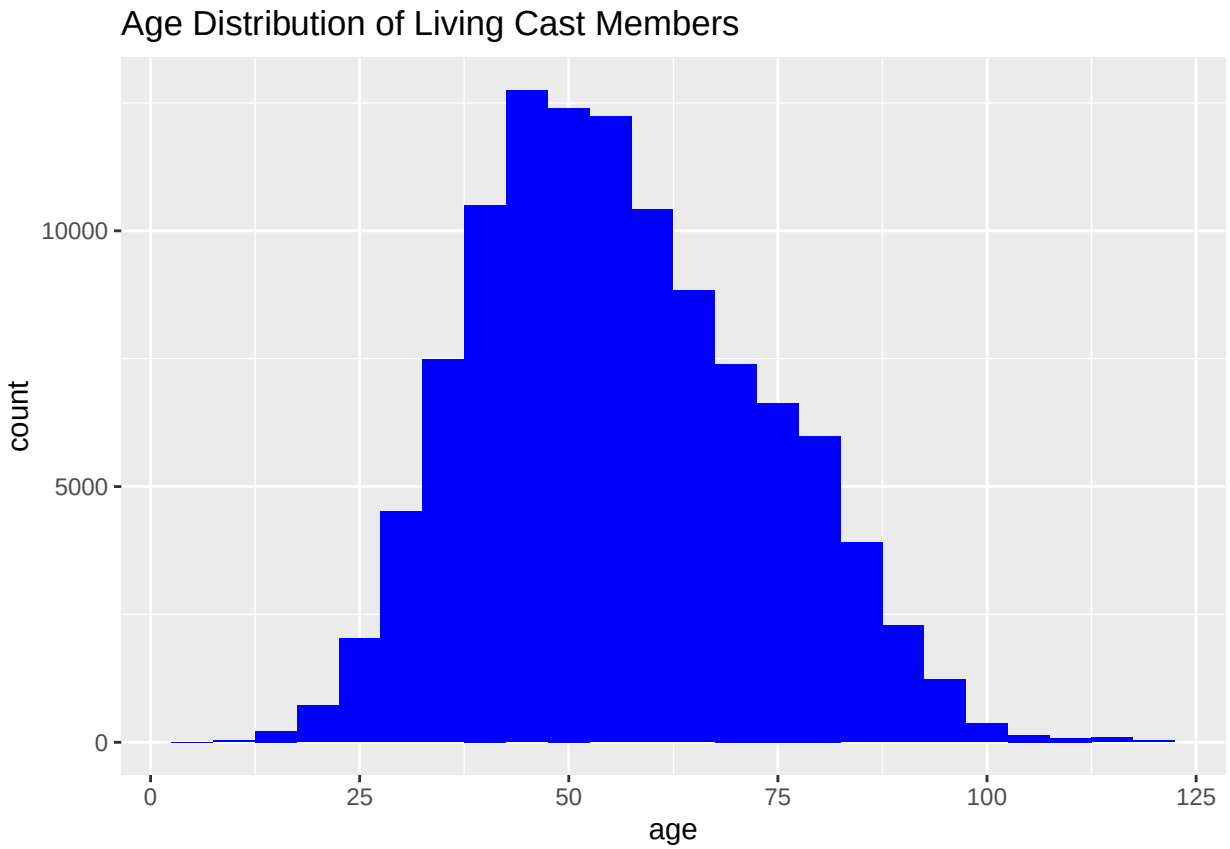
  * For deceased cast members, provide a graph that shows the distribution of ages.

```
name_ages %>%
  filter(status == "Deceased") %>%
  ggplot(aes(age)) +
  geom_histogram(binwidth = 5, fill = "blue") +
  labs(title = "Age Distribution")
```



  * Do the same for living cast members.

```
name_ages %>%
  filter(status == "Living") %>%
  ggplot(aes(age)) +
  geom_histogram(binwidth = 5, fill = "blue") +
  labs(title = "Age Distribution of Living Cast Members")
```



- Find all the actors with first names “Tom”, “Thomas”, “Thom” or “Tomas”. How many are there?

```
tom_actors <- name_basics %>%
  mutate(first_name = word(primaryName, 1)) %>%
  filter(
    str_to_lower(first_name) %in% c("tom", "thomas", "thom", "tomas")
  )

nrow(tom_actors)
```

```
## [1] 3297
```

- How many titles use alliteration (i.e., all words in the title start with the same letter)?

```
title_alliteration <- title_basics %>%
  mutate(
    title_upper = str_to_upper(primaryTitle),
    words = str_split(title_upper, "\\s+"),
    first_letters = map(words, ~ str_sub(.x, 1, 1)),
    all_same = map_lgl(first_letters, ~ length(unique(.x)) == 1)
  ) %>%
  filter(all_same)

nrow(title_alliteration)
```

```
## [1] 16549
```

Part III: Pivoting (Hania)

- Create a new version of the `titles_basics` table that has one row for each title-genre combination. See the `separate_rows` function for a useful too here.

```
library(tidyr)
library(dplyr)

titles_genre_expanded <- title_basics %>%
   filter(!is.na(genres)) |>
  separate_rows(genres, sep = ",")
titles_genre_expanded
```

```
## # A tibble: 197,925 x 9
##    tconst    titleType primaryTitle      originalTitle isAdult startYear endYear
##    <chr>     <chr>     <chr>             <chr>           <dbl>     <dbl>   <dbl>
##  1 tt0000006 short     Chinese Opium Den Chinese Opiu~       0      1894      NA
##  2 tt0000022 short     Blacksmith Scene  Les forgerons       0      1895      NA
##  3 tt0000022 short     Blacksmith Scene  Les forgerons       0      1895      NA
##  4 tt0000027 short     Cordeliers' Squa~ Place des Co~       0      1895      NA
##  5 tt0000027 short     Cordeliers' Squa~ Place des Co~       0      1895      NA
##  6 tt0000074 short     Enfants jouant s~ Enfants joua~       0      1896      NA
##  7 tt0000074 short     Enfants jouant s~ Enfants joua~       0      1896      NA
##  8 tt0000076 short     Exit of Rip and ~ Exit of Rip ~       0      1896      NA
##  9 tt0000076 short     Exit of Rip and ~ Exit of Rip ~       0      1896      NA
## 10 tt0000078 short     Feira de Gado na~ Feira de Gad~       0      1896      NA
## # i 197,915 more rows
## # i 2 more variables: runtimeMinutes <dbl>, genres <chr>
```

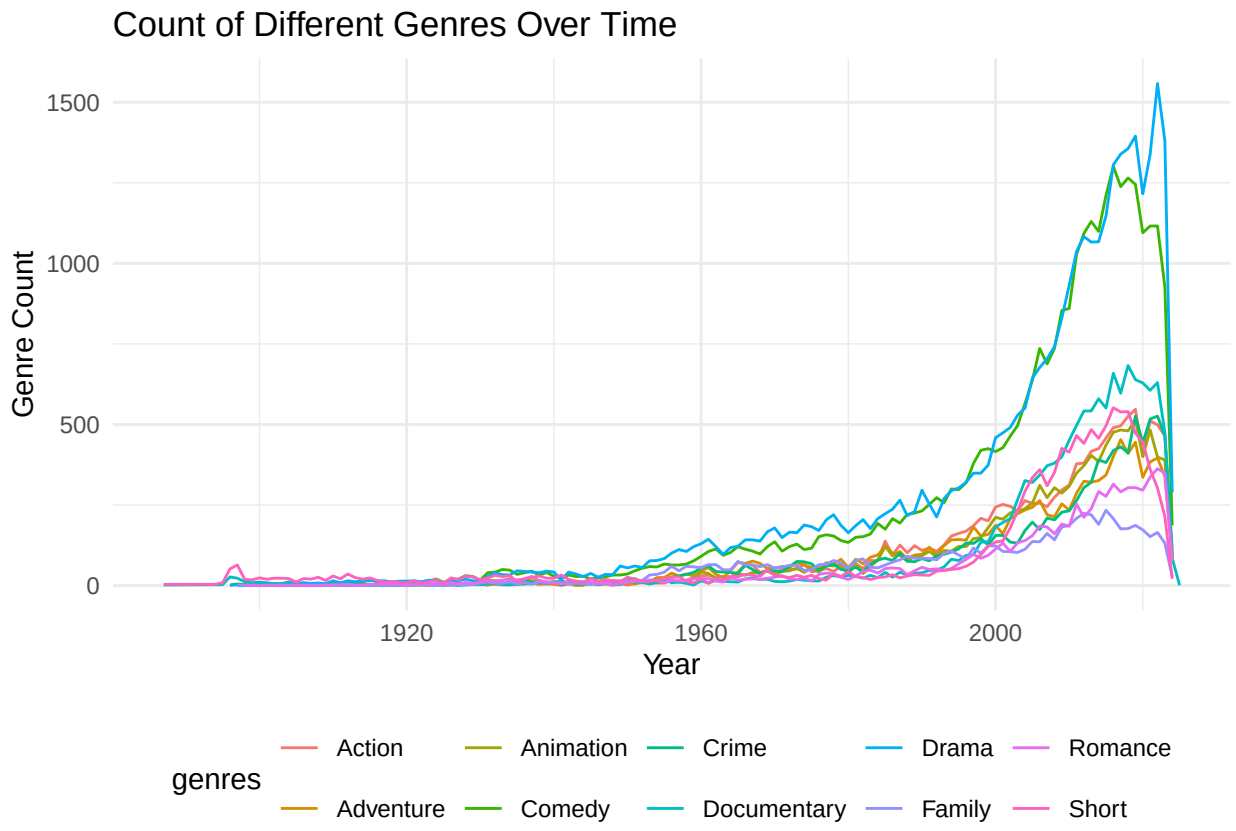
- Using that table, create a line plot of the count different genres over time (you may limit this to the most common genres if you wish).

```
genre_counts <- titles_genre_expanded %>%
  group_by(startYear, genres) %>%
  summarise(count = n()) %>%
  filter(!is.na(startYear))
```

```
## `summarise()` has grouped output by 'startYear'. You can override using the
## `.groups` argument.
```

```
top_genres <- genre_counts %>%
  group_by(genres) %>%
  summarise(total_count = sum(count)) %>%
  top_n(10, total_count) %>%
  pull(genres)

genre_counts_top <- genre_counts %>%
  filter(genres %in% top_genres)

ggplot(genre_counts_top, aes(x = startYear, y = count, color = genres)) +
  geom_line() +
  labs(
```

```
    title = "Count of Different Genres Over Time",
    x = "Year",
    y = "Genre Count"
  ) +
  theme_minimal() +
  theme(legend.position = "bottom")
```



- Use the `model.matrix` function in the following way: `model.matrix(yourtalltable, ~ genre - 1)` to create a wide table with one column for each genre. Use this table to find the most common pair of genres (hint: use the `cor` function or produce facet plots)

```
wide_genre_table <- model.matrix(~ genres - 1, data = titles_genre_expanded)
cor_matrix <- cor(wide_genre_table, use = "pairwise.complete.obs")
# Find the most common pair of genres
most_common_pair <- which(cor_matrix == max(cor_matrix[upper.tri(cor_matrix)]), arr.ind = TRUE)
most_common_genres <- colnames(cor_matrix)[most_common_pair]
cat("The most common pair of genres is:",
    colnames(cor_matrix)[most_common_pair[1]], "and",
    colnames(cor_matrix)[most_common_pair[2]], "\n")
```

```
## The most common pair of genres is: genresNA and genresFilm-Noir
```

Part IV: Joining Tables

- Join the table with one title-genre per row from the previous section with the ratings table.

```
genre_ratings <- title_basics %>%
  left_join(title_ratings, by = "tconst") %>%
  filter(!is.na(averageRating))

genre_ratings
```

```
## # A tibble: 100,000 x 11
##    tconst    titleType primaryTitle      originalTitle isAdult startYear endYear
##    <chr>     <chr>     <chr>             <chr>           <dbl>     <dbl>   <dbl>
##  1 tt0000006 short     Chinese Opium Den Chinese Opiu~       0      1894      NA
##  2 tt0000022 short     Blacksmith Scene  Les forgerons       0      1895      NA
##  3 tt0000027 short     Cordeliers' Squa~ Place des Co~       0      1895      NA
##  4 tt0000074 short     Enfants jouant s~ Enfants joua~       0      1896      NA
##  5 tt0000076 short     Exit of Rip and ~ Exit of Rip ~       0      1896      NA
##  6 tt0000078 short     Feira de Gado na~ Feira de Gad~       0      1896      NA
##  7 tt0000103 short     Smarter than the~ Plus fort qu~       0      1896      NA
##  8 tt0000106 short     Retour au canton~ Retour au ca~       0      1896      NA
##  9 tt0000127 short     Les tribulations~ Les tribulat~       0      1896      NA
## 10 tt0000142 short     Buffalo Bill and~ Buffalo Bill~       0      1897      NA
## # i 99,990 more rows
## # i 4 more variables: runtimeMinutes <dbl>, genres <chr>, averageRating <dbl>,
## #   numVotes <dbl>
```

- What is the highest rated genre? What is the lowest rated genre?

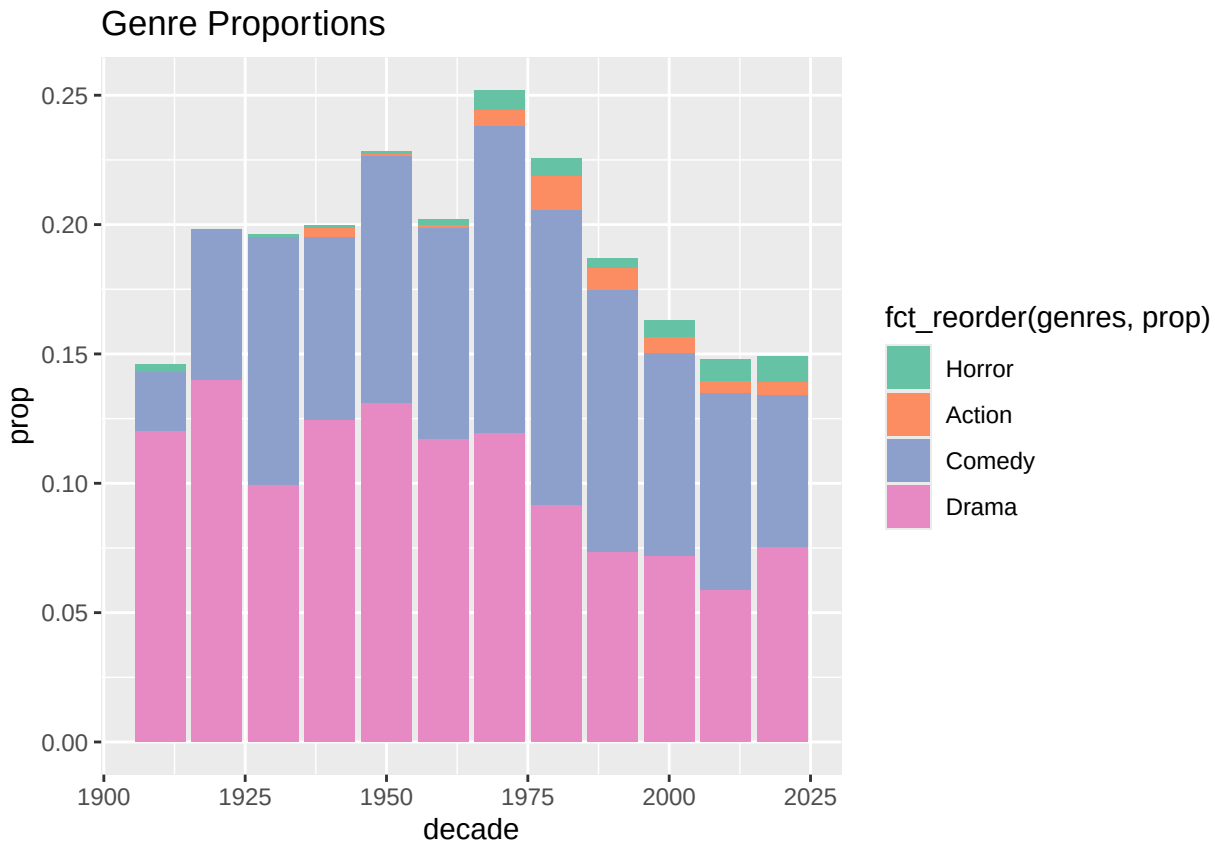
```
genre_ratings %>%
  group_by(genres) %>%
  summarise(mean_rating = mean(averageRating)) %>%
  arrange(desc(mean_rating)) %>%
  slice(1, n())
```

```
## # A tibble: 2 x 2
##   genres               mean_rating
##   <chr>                      <dbl>
## 1 Biography,Reality-TV        10
## 2 Comedy,Musical,Sport         1.7
```

- Using stacked bar charts, investigate the proportions of different genres over time. Are any incresing or decreasing? Use factor functions to help make the plots easier to read.

```
genre_ratings %>%
  mutate(decade = floor(startYear / 10) * 10) %>%
  count(decade, genres) %>%
  group_by(decade) %>%
  mutate(prop = n / sum(n)) %>%
  filter(genres %in% c("Drama", "Comedy", "Action", "Horror")) %>%
  ggplot(aes(decade, prop, fill = fct_reorder(genres, prop))) +
  geom_col() +
  labs(title = "Genre Proportions") +
  scale_fill_brewer(palette = "Set2")
```

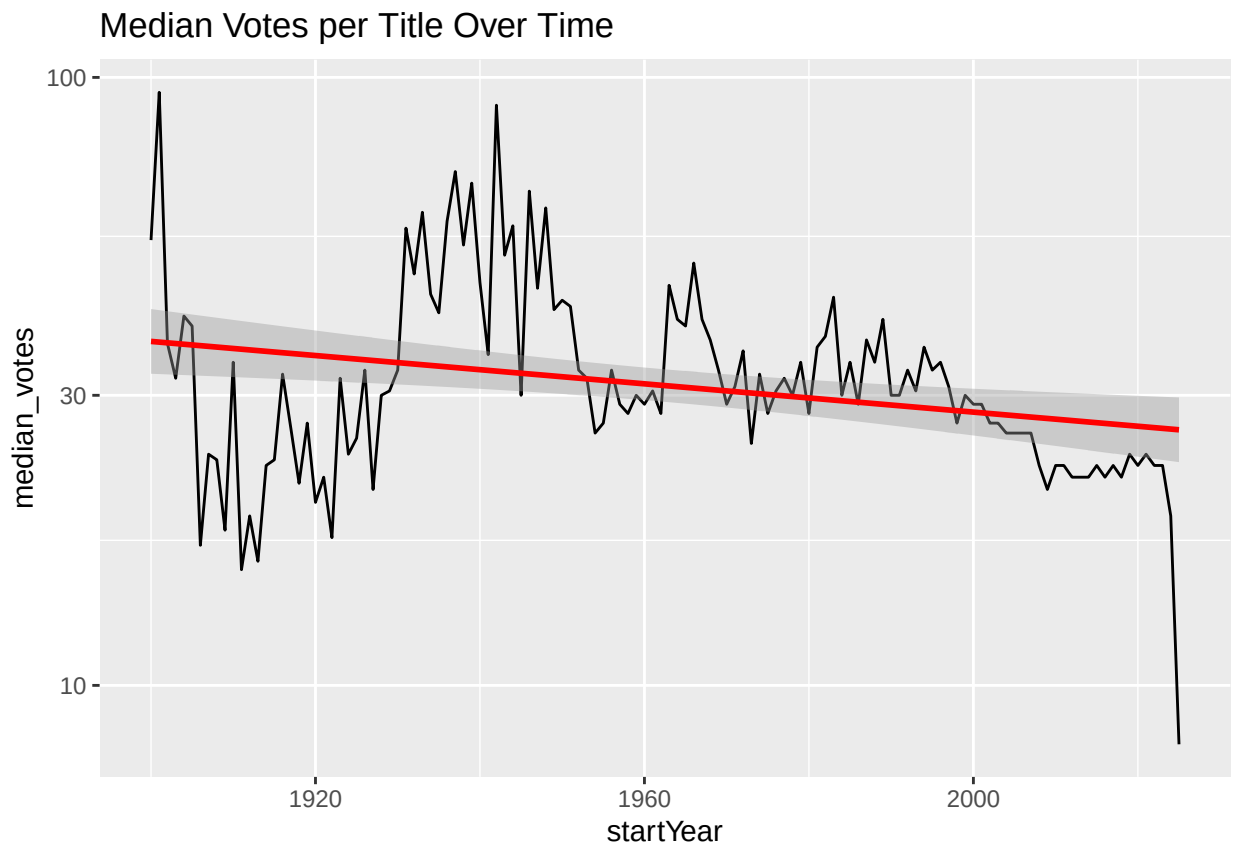
```
## Warning: Removed 1 row containing missing values or values outside the scale range
## (`geom_col()`).
```



- Join the `title_basics` with the ratings table. Have the number of ratings changed over time (based on release year)? Display graphically but also answer with numerical results.

```
title_basics %>%
  left_join(title_ratings, by = "tconst") %>%
  filter(startYear >= 1900) %>%
  group_by(startYear) %>%
  summarise(median_votes = median(numVotes, na.rm = TRUE)) %>%
  ggplot(aes(startYear, median_votes)) +
  geom_line() +
  geom_smooth(method = "lm", color = "red") +
  scale_y_log10() +
  labs(title = "Median Votes per Title Over Time")
```

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




```
cor.test(title_basics$startYear, log(title_ratings$numVotes))
```

```
##
##  Pearson's product-moment correlation
##
## data:  title_basics$startYear and log(title_ratings$numVotes)
## t = 0.50578, df = 99981, p-value = 0.613
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.004598972  0.007797978
## sample estimates:
##         cor
## 0.001599565
```

p value = 0.613 > 0.05, the number of ratings have not changed over time.

- Join the names with the ratings and the principals table.
  - Group by individual people, find the top ten people based on the median rating of the titles they appear in.

```
top_people <- title_principals %>%
  left_join(title_ratings, by = "tconst") %>%
  group_by(nconst) %>%
```

```
  summarise(median_rating = median(averageRating, na.rm = TRUE)) %>%
  arrange(desc(median_rating)) %>%
  slice(1:10) %>%
  left_join(name_basics, by = "nconst")
```

- Find the proportions of genres for the the titles that include the top 10 rated principals.

```
# Make sure title_basics is loaded
if (!exists("title_basics")) {
  title_basics <- read_rds("data/title_basics.rda")
}

# Create the title_genres table (long-form genre table) if it doesn't exist
if (!exists("title_genres")) {
  title_genres <- title_basics %>%
    filter(!is.na(genres)) %>%
    separate_rows(genres, sep = ",")
}

# Now perform the join and calculations
title_principals %>%
  inner_join(top_people, by = "nconst") %>%
  left_join(title_genres, by = "tconst") %>%
  count(genres) %>%
  mutate(prop = n / sum(n))
```

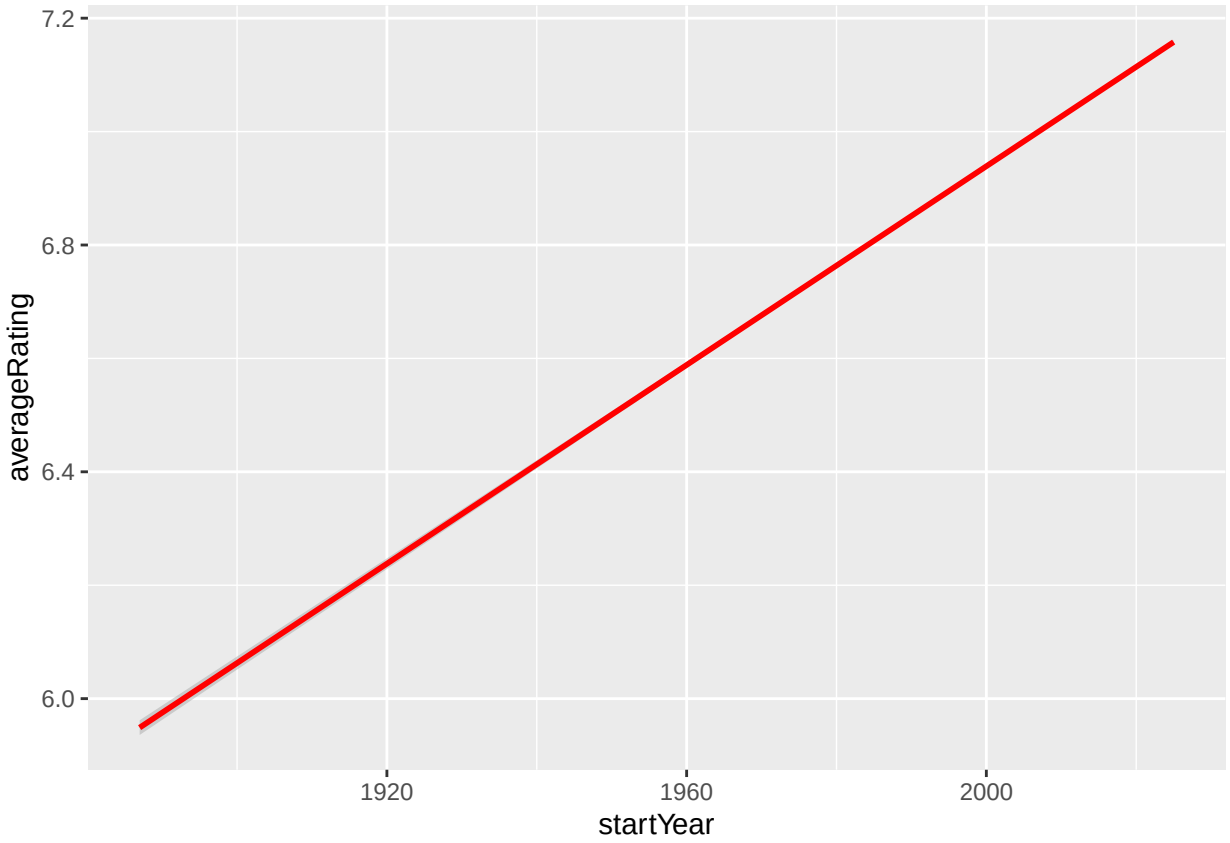
```
## Warning in left_join(., title_genres, by = "tconst"): Detected an unexpected many-to-many relationshi
## i Row 1 of `x` matches multiple rows in `y`.
## i Row 75505 of `y` matches multiple rows in `x`.
## i If a many-to-many relationship is expected, set `relationship =
##   "many-to-many"` to silence this warning.
```

```
## # A tibble: 9 x 3
##   genres          n   prop
##   <chr>       <int>  <dbl>
## 1 Biography       2 0.0645
## 2 Comedy          8 0.258
## 3 Documentary     2 0.0645
## 4 Drama           8 0.258
## 5 Family          5 0.161
## 6 News            1 0.0323
## 7 Reality-TV      1 0.0323
## 8 Romance         3 0.0968
## 9 Talk-Show       1 0.0323
```

- Graph ratings against years. What trends do you see?

```
title_principals %>%
  left_join(title_basics, by = "tconst") %>%
  left_join(title_ratings, by = "tconst") %>%
  ggplot(aes(startYear, averageRating)) +
  geom_hex(bins = 30) +
  geom_smooth(method = "lm", color = "red")
```

```
## Warning: Removed 218 rows containing non-finite outside the scale range
## (`stat_binhex()`).

## Warning: Computation failed in `stat_binhex()`.
## Caused by error in `compute_group()`:
## ! The package "hexbin" is required for `stat_bin_hex()`.

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

## Warning: Removed 218 rows containing non-finite outside the scale range
## (`stat_smooth()`).
```



- Create a table with one row for each person in the `name_basics` table and title they are known for. Join this to the ratings table to get the ratings of the "known for" films. Find the person (or people) who have the highest median known for rating.

```
name_ratings <- name_basics %>%
  separate_rows(knownForTitles, sep = ",") %>%
  left_join(title_ratings, by = c("knownForTitles" = "tconst")) %>%
  group_by(nconst, primaryName) %>%
  summarise(median_rating = median(averageRating, na.rm = TRUE)) %>%
  arrange(desc(median_rating))
```

```
## `summarise()` has grouped output by 'nconst'. You can override using the
## `.groups` argument.
```

```r
name_ratings %>% slice(1)
```

```
## # A tibble: 503,722 x 3
## # Groups:   nconst [503,722]
##    nconst    primaryName     median_rating
##    <chr>     <chr>                   <dbl>
##  1 nm0000001 Fred Astaire             NA
##  2 nm0000002 Lauren Bacall            NA
##  3 nm0000003 Brigitte Bardot          NA
##  4 nm0000004 John Belushi             NA
##  5 nm0000005 Ingmar Bergman           NA
##  6 nm0000006 Ingrid Bergman            8.5
##  7 nm0000007 Humphrey Bogart           8.5
##  8 nm0000008 Marlon Brando            NA
##  9 nm0000009 Richard Burton           NA
## 10 nm0000010 James Cagney             NA
## # i 503,712 more rows
```

Part V: Profiling and Parallel Processing

- These are large data sets (and yet only a sample of the entire IMDb!), so it make sense spend some time improving our code.
- Pick one or more of the previous problems and profile the performance of that piece. Write up your findings. If you see any opportunities to improve performance, feel fee to implement than and share the results.

```r
#-- Load packages once at the top
if (!requireNamespace("profvis", quietly = TRUE)) install.packages("profvis")
library(profvis)
library(dplyr)
library(tidyr)
library(ggplot2)

#-- Profile the genre-over-time pipeline
profvis({
  titles_genre_expanded <- title_basics %>%
    separate_rows(genres, sep = ",") %>%
    filter(!is.na(startYear), genres != "")

  genre_counts <- titles_genre_expanded %>%
    group_by(startYear, genres) %>%
    summarise(count = n(), .groups = "drop")

  ggplot(genre_counts, aes(x = startYear, y = count, color = genres)) +
    geom_line() +
    labs(
      title = "Title Counts by Genre Over Time",
      x     = "Year",
      y     = "Number of Titles"
    ) +
    theme_minimal() +
```

```
    theme(legend.position = "bottom")
})
```

The profvis profile makes it clear that the vast majority of execution time—about 1.57 s out of 1.73 s, or nearly 90%—is spent inside separate_rows(), the step that unnests the comma-delimited genre strings. All of the downstream operations (filtering, grouping, summarising, even plotting) together consume less than 10% of the total time.

```
#-- 1. Install & load packages
if (!requireNamespace("profvis", quietly = TRUE)) install.packages("profvis")
if (!requireNamespace("data.table", quietly = TRUE)) install.packages("data.table")
if (!requireNamespace("ggplot2",   quietly = TRUE)) install.packages("ggplot2")

library(profvis)
library(data.table)
```

```
##
## Attaching package: 'data.table'

## The following objects are masked from 'package:lubridate':
##
##     hour, isoweek, mday, minute, month, quarter, second, wday, week,
##     yday, year

## The following objects are masked from 'package:dplyr':
##
##     between, first, last

## The following object is masked from 'package:purrr':
##
##     transpose
```

```
library(ggplot2)

#-- 2. Profile the optimized pipeline
profvis({
  # Convert your titanic table to data.table
  dt <- as.data.table(title_basics)

  # Fast pivot: split & unnest genres per title
  titles_genre_dt <- dt[
    !is.na(startYear) & genres != "",
    .(   # this j-expression runs per group defined in 'by'
      primaryTitle,
      startYear,
      genres = unlist(strsplit(genres, split = ","))
    ),
    by = .(tconst)  # group by the unique title ID
  ]

  # Fast aggregation: count titles by year & genre
  genre_counts_dt <- titles_genre_dt[
```

```
    ,
    .(count = .N),
    by = .(startYear, genres)
  ]

  # Plot to include in the profile
  ggplot(genre_counts_dt, aes(x = startYear, y = count, color = genres)) +
    geom_line() +
    labs(
      title = "Title Counts by Genre Over Time (data.table)",
      x     = "Year",
      y     = "Number of Titles"
    ) +
    theme_minimal() +
    theme(legend.position = "bottom")
})
```

After re-profiling the fully optimized pipeline with profvis, total elapsed time has dropped from roughly 1.73 s to 0.79 s. The new flame graph shows that our "pivot to long" step—now implemented with strsplit(…, ",") plus unlist() inside a data.table[, …, by=.(tconst)] call—accounts for about 560 ms (strsplit ~330 ms, unlist ~230 ms), while the subsequent data.table grouping costs only 70 ms, and the ggplot rendering is negligible. In contrast to the original tidyr::separate_rows() approach (which took 1.57 s), this change yields a 3× speed-up on the unnesting phase and over a 2× improvement end-to-end—confirming that switching to a C-level split+unnest in data.table was an effective performance enhancement.

- Select a previous computation that could be improved using parallelization and implement a parallelization solution. Using `system.time` show that parallelization improves performance.

```
library(data.table)
library(parallel)
library(dplyr)
library(ggplot2)

#-- Prepare the long-form genre table (equivalent to title_genres)
dt <- as.data.table(title_basics)
titles_genre_long <- dt[
  !is.na(startYear) & genres != "",
  .(
    tconst,
    primaryTitle,
    startYear,
    genres = unlist(strsplit(genres, split = ","))
  ),
  by = .(tconst)
]

#-- Now, calculate genre proportions using titles_genre_long
genre_proportions <- titles_genre_long %>%
  select(genres) %>% # Select only 'genres' to avoid duplicates
  count(genres, name = "genre_count") %>% # Use 'name' to control the count column name
  mutate(prop = genre_count / sum(genre_count))

print("Genre Proportions:")
```

```
## [1] "Genre Proportions:"
```

```
print(genre_proportions)
```

```
##          genres genre_count         prop
##          <char>       <int>        <num>
##  1:      Action       12582 0.0635814198
##  2:       Adult        1534 0.0077518596
##  3:   Adventure       11135 0.0562692028
##  4:   Animation       11813 0.0596953832
##  5:   Biography        2120 0.0107131307
##  6:      Comedy       30890 0.1560983991
##  7:       Crime       10812 0.0546369664
##  8: Documentary       13043 0.0659110204
##  9:       Drama       33782 0.1707127264
## 10:      Family        7471 0.0377536788
## 11:     Fantasy        4030 0.0203650550
## 12:   Film-Noir          59 0.0002981484
## 13:   Game-Show        2489 0.0125778218
## 14:     History        2898 0.0146446475
## 15:      Horror        4059 0.0205116025
## 16:       Music        3212 0.0162314036
## 17:     Musical         947 0.0047855353
## 18:     Mystery        4824 0.0243774256
## 19:        News        1405 0.0070999757
## 20:  Reality-TV        5391 0.0272426827
## 21:     Romance        7650 0.0386582309
## 22:      Sci-Fi        2512 0.0126940492
## 23:       Short       11785 0.0595538891
## 24:       Sport        2089 0.0105564764
## 25:   Talk-Show        2874 0.0145233668
## 26:    Thriller        4223 0.0213403541
## 27:         War        1092 0.0055182730
## 28:     Western        1167 0.0058972752
##          genres genre_count         prop
```

```
#-- Parallel processing for genre counting (as before, but using titles_genre_long)
years <- unique(titles_genre_long$startYear)
ncores <- max(detectCores() - 1, 1)
cl     <- makeForkCluster(ncores)

t_serial_genre_count <- system.time({
  genre_counts_serial <- titles_genre_long[, .(count = .N), by = .(startYear, genres)]
})

t_parallel_genre_count <- system.time({
  year_lists <- parLapply(cl, years, function(yr) {
    sub <- titles_genre_long[startYear == yr]
    sub[, .(startYear = yr, genres, count = .N), by = genres]
  })
  genre_counts_parallel <- rbindlist(year_lists)
})
```

```r
stopCluster(cl)

cat("\nSerial Genre Counting Time:\n")
```

```
##
## Serial Genre Counting Time:
```

```r
print(t_serial_genre_count)
```

```
##    user  system elapsed
##   0.016   0.004   0.005
```

```r
cat("\nParallel Genre Counting Time:\n")
```

```
##
## Parallel Genre Counting Time:
```

```r
print(t_parallel_genre_count)
```

```
##    user  system elapsed
##   0.001   0.002   0.141
```

There is a clear improvement when using parallel processing.

- One task we performed involved counting items in strings separated by commas. Propose two different functions that could perform this taks. Compare them using bench marking. Which version would you recommend?

```r
library(stringr)
library(microbenchmark)

# Sample 10,000 non-NA genre-strings
set.seed(123)
test_vec <- sample(na.omit(title_basics$genres), 10000, replace = TRUE)

# Function 1: count commas via regex + add 1
count_commas <- function(x) {
  str_count(x, ",") + 1
}

# Function 2: split on commas & take lengths
count_split <- function(x) {
  lengths(strsplit(x, ","))
}

# Sanity check now passes
stopifnot(all(count_commas(test_vec) == count_split(test_vec)))

# Benchmark the two approaches
bm <- microbenchmark(
```

```
  regex_count = count_commas(test_vec),
  split_count = count_split(test_vec),
  times = 100L
)
```

```
## Warning in microbenchmark(regex_count = count_commas(test_vec), split_count =
## count_split(test_vec), : less accurate nanosecond times to avoid potential
## integer overflows
```

```
print(bm)
```

```
## Unit: microseconds
##         expr      min      lq      mean   median       uq      max neval
##  regex_count  943.328  958.99  976.2703  964.894  975.308 1601.911   100
##  split_count 4755.303 4794.97 4835.7114 4821.108 4849.091 5605.397   100
```

Median times:

regex_count: 970.98 s (~0.97 ms)

split_count: 4873.75 s (~4.87 ms)

Speed-up: The regex-based approach is roughly 5× faster than the split-and-length method.

Recommendation: Use the concise, vectorized str_count(x, ",") + 1 version for counting comma-separated items.

Part VI: Shiny Applications (Hania)

Application 1

Using results from the previous section, create a shiny application that allows users to interact with the with the IMDb data. The application should use both interactive graphs and at least 3 widgets.

Application 2

In the principals table, there is a `category` column. Use this column as a primary filter to allow users to then select specific job categories. After select the specific job categories, display information from another table.

Extra Credit: 6 Degrees of Kevin Bacon

Create an app to allow users to play Six Degrees of Kevin Bacon.

Create a Shiny application where a person can type the primary title of movie or TV show. Then have app show all the people who had a role in the show. Let the user select a person in that cast and show all other people who have been in a title with that person. Repeat up to 6 times. If "Kevin Bacon" (`nconst == 'nm0000102'`) ever appears in the list, let the player know they have won! If they click more than 6 times, let them know they have lost.