DataSci 306 Final Project

Varsha Rajesh, Neha Sancheti, Elizabeth Szwajnos

2025-04-28

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.

Part II: EDA of individual tables

- 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.
  - 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)
- For the `titles_basics` table
  - use two different variables to group and explore how `runtimeMinutes` varies for these different groups. Produce appropriate summaries.
  - How many titles are known for name that is different than the original release name?
  - Graph the conditional distributions of release year based on the previous results. Comment on any trends you observe.
- 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?
- For the names table,
  - Count the number of titles each person is known for and plot this distribution.
  - investigate the age of cast members

    * Group the data into living and deceased cast members.
    * For deceased cast members, provide a graph that shows the distribution of ages.
    * Do the same for living cast members.

- Find all the actors with first names "Tom", "Thomas", "Thom" or "Tomas". How many are there?
- How many titles use alliteration (i.e., all words in the title start with the same letter)?

```r
# Load required libraries
library(readr)
library(dplyr)
library(ggplot2)
library(stringr)
library(tidyr)

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")

names(name_basics)
```

```
## [1] "nconst"            "primaryName"       "birthYear"
## [4] "deathYear"         "primaryProfession" "knownForTitles"
```

```r
names(title_basics)
```

```
## [1] "tconst"         "titleType"      "primaryTitle"   "originalTitle"
## [5] "isAdult"        "startYear"      "endYear"        "runtimeMinutes"
## [9] "genres"
```

```r
names(title_principals)
```

```
## [1] "tconst"     "ordering"   "nconst"     "category"   "job"
## [6] "characters"
```

```r
names(title_ratings)
```

```
## [1] "tconst"        "averageRating" "numVotes"
```

```r
# For each quantitative column, provide some summary statistics
name_basics %>% select(where(is.numeric)) %>% 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
```

```r
title_basics %>% select(where(is.numeric)) %>% summary()
```

```
##     isAdult           startYear       endYear      runtimeMinutes
##  Min.   :   0.000   Min.   :1887   Min.   :1938   Min.   :   1.00
##  1st Qu.:   0.000   1st Qu.:1997   1st Qu.:2001   1st Qu.:  23.00
##  Median :   0.000   Median :2011   Median :2013   Median :  45.00
##  Mean   :   0.036   Mean   :2003   Mean   :2008   Mean   :  55.35
##  3rd Qu.:   0.000   3rd Qu.:2018   3rd Qu.:2019   3rd Qu.:  85.00
##  Max.   :2020.000   Max.   :2025   Max.   :2025   Max.   :5220.00
```

```
##                     NA's   :17     NA's   :96408   NA's   :29696
```

```r
title_principals %>% select(where(is.numeric)) %>% summary()
```

```
##     ordering
##  Min.   : 1.00
##  1st Qu.: 4.00
##  Median : 8.00
##  Mean   : 9.06
##  3rd Qu.:13.00
##  Max.   :62.00
```

```r
title_ratings %>% select(where(is.numeric)) %>% 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
```

```r
# For any character columns, decide if they represent categorical data

summarize_categorical <- function(df, cols) {

  value_summary <- df %>%
    select(all_of(cols)) %>%
    pivot_longer(cols = everything(), names_to = "column", values_to = "value") %>%
    filter(!is.na(value)) %>%
    group_by(column, value) %>%
    summarise(count = n(), .groups = "drop") %>%
    group_by(column) %>%
    mutate(
      percent = round(100 * count / sum(count), 2)
    ) %>%
    arrange(column, desc(count))

  # Summary of total unique values
  unique_summary <- df %>%
    select(all_of(cols)) %>%
    summarise_all(~n_distinct(., na.rm = TRUE)) %>%
    pivot_longer(cols = everything(), names_to = "column", values_to = "total_unique_vals")

  list(
    value_summary = value_summary,
    unique_summary = unique_summary
  )
}


# name_basics
name_basics %>%
  select(where(is.character)) %>%
  summarise_all(~ n_distinct(.)) %>%
  pivot_longer(cols = everything(), names_to = "column", values_to = "unique_vals") %>%
```

```r
  filter(unique_vals < 50)
```

```
## # A tibble: 0 x 2
## # i 2 variables: column <chr>, unique_vals <int>
```

```r
# title_basics
title_basics_cats <- title_basics %>%
  select(where(is.character)) %>%
  summarise_all(~ n_distinct(.)) %>%
  pivot_longer(cols = everything(), names_to = "column", values_to = "unique_vals") %>%
  filter(unique_vals < 50) %>%
  pull(column)

title_basics_summary <- summarize_categorical(title_basics, title_basics_cats)

# View summaries
title_basics_summary$value_summary
```

```
## # A tibble: 10 x 4
## # Groups:   column [1]
##    column    value        count percent
##    <chr>     <chr>        <int>   <dbl>
##  1 titleType tvEpisode    50194   50.2
##  2 titleType movie        21467   21.5
##  3 titleType short        11118   11.1
##  4 titleType tvSeries      6555    6.56
##  5 titleType tvMovie       3703    3.7
##  6 titleType video         3617    3.62
##  7 titleType videoGame     1221    1.22
##  8 titleType tvMiniSeries  1127    1.13
##  9 titleType tvSpecial      837    0.84
## 10 titleType tvShort        161    0.16
```

```r
title_basics_summary$unique_summary
```

```
## # A tibble: 1 x 2
##   column    total_unique_vals
##   <chr>                 <int>
## 1 titleType                10
```

```r
# title_principals
title_principals_cats <- title_principals %>%
  select(where(is.character)) %>%
  summarise_all(~ n_distinct(.)) %>%
  pivot_longer(cols = everything(), names_to = "column", values_to = "unique_vals") %>%
  filter(unique_vals < 50) %>%
  pull(column)

title_principals_summary <- summarize_categorical(title_principals, title_principals_cats)

# View summaries
title_principals_summary$value_summary
```

```
## # A tibble: 13 x 4
## # Groups:   column [1]
##    column   value                count percent
```

```
##    <chr>    <chr>                <int>   <dbl>
##  1 category actor               406747   29.9
##  2 category actress             242512   17.8
##  3 category writer              146370   10.8
##  4 category self                111477    8.19
##  5 category producer             98111    7.21
##  6 category director             87017    6.39
##  7 category editor               77280    5.68
##  8 category composer             61829    4.54
##  9 category cinematographer      60902    4.47
## 10 category casting_director     32853    2.41
## 11 category production_designer  28164    2.07
## 12 category archive_footage       7935    0.58
## 13 category archive_sound          267    0.02
```

```r
title_principals_summary$unique_summary
```

```
## # A tibble: 1 x 2
##   column   total_unique_vals
##   <chr>                <int>
## 1 category                13
```

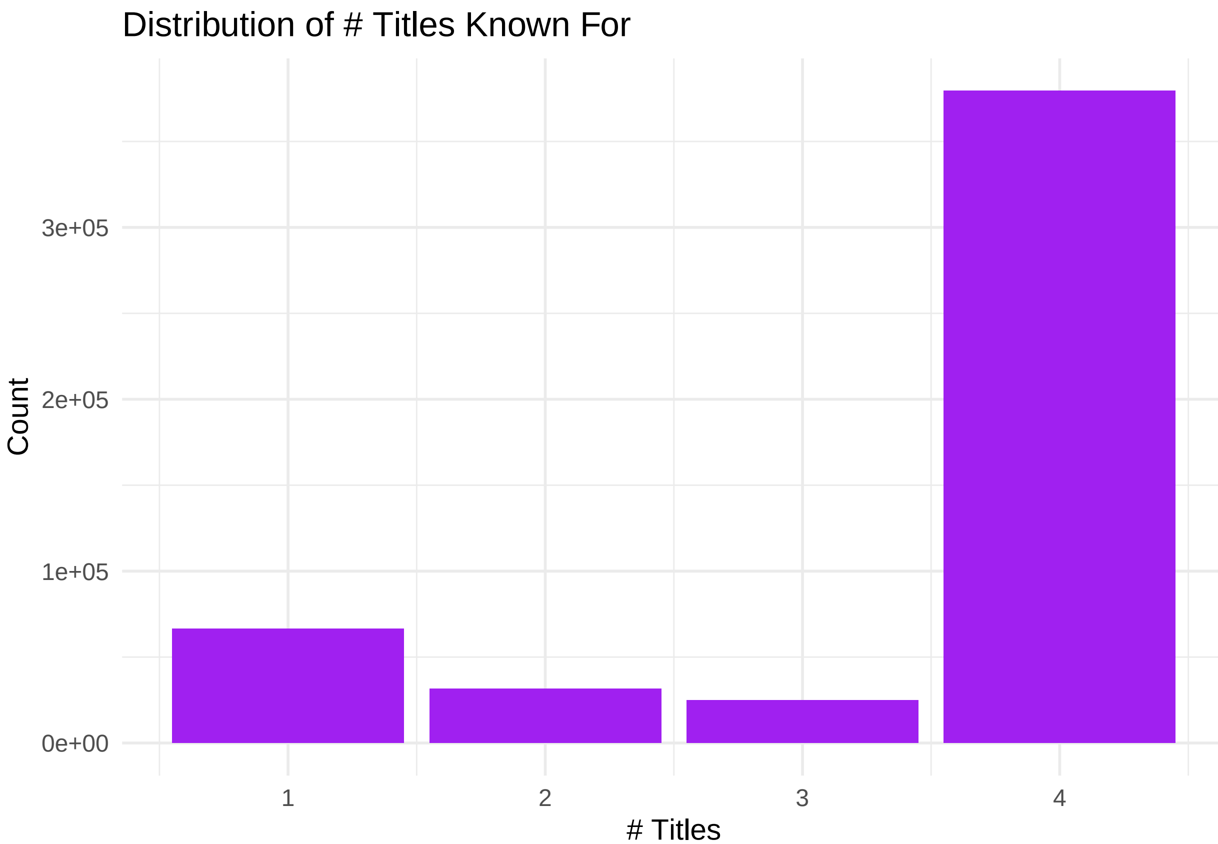
```r
# title_ratings
title_ratings %>%
  select(where(is.character)) %>%
  summarise_all(~ n_distinct(.)) %>%
  pivot_longer(cols = everything(), names_to = "column", values_to = "unique_vals") %>%
  filter(unique_vals < 50)
```

```
## # A tibble: 0 x 2
## # i 2 variables: column <chr>, unique_vals <int>
```

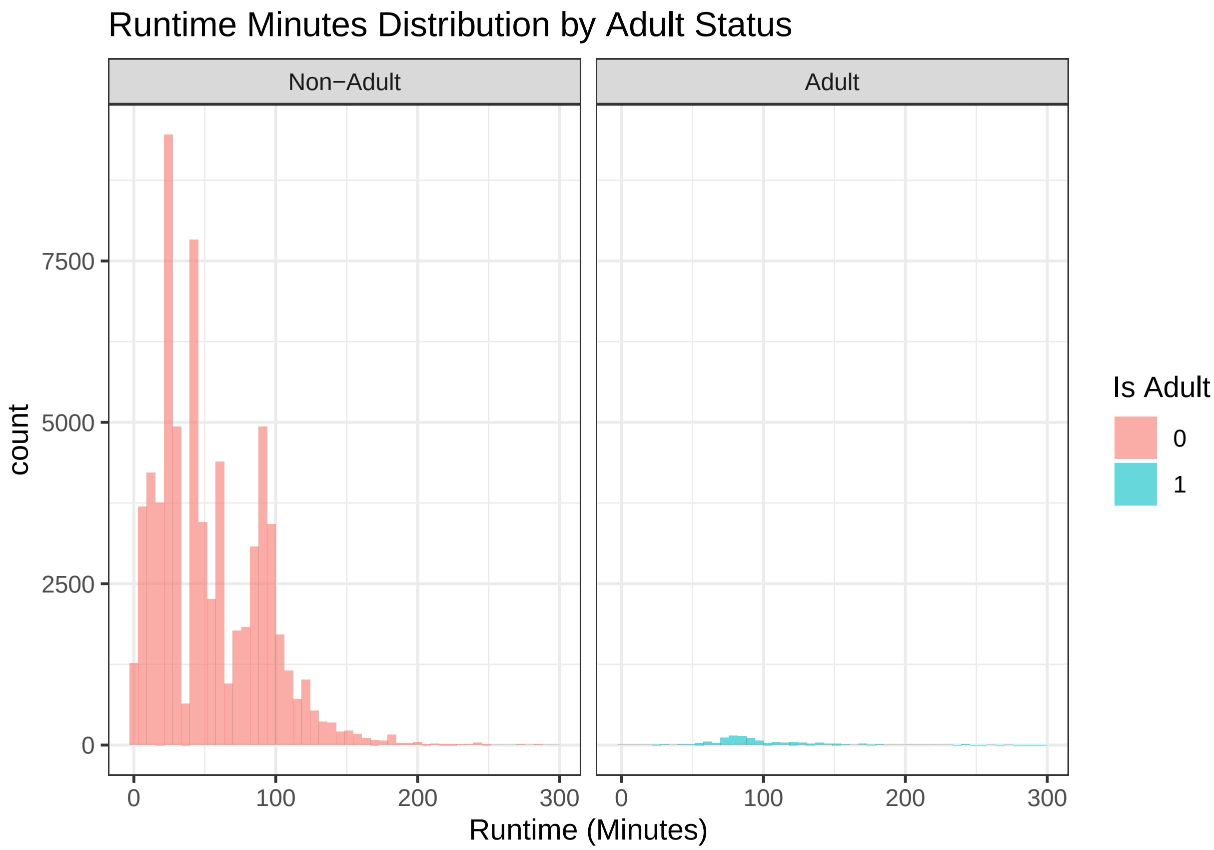
In title_basics, titleType is categorical. In title_principals, category is categorical. For each, distribution within the categorical variable is presented.

```r
# Plot 1: Distribution of Number of Titles Known For (name_basics)
name_basics %>%
  mutate(n_titles = str_count(knownForTitles, "," ) + 1) %>%
  ggplot(aes(x = n_titles)) +
  geom_bar(fill = "purple") +
  scale_x_continuous(breaks = 1:10) +
  labs(title = "Distribution of # Titles Known For", x = "# Titles", y = "Count") +
  theme_minimal()
```

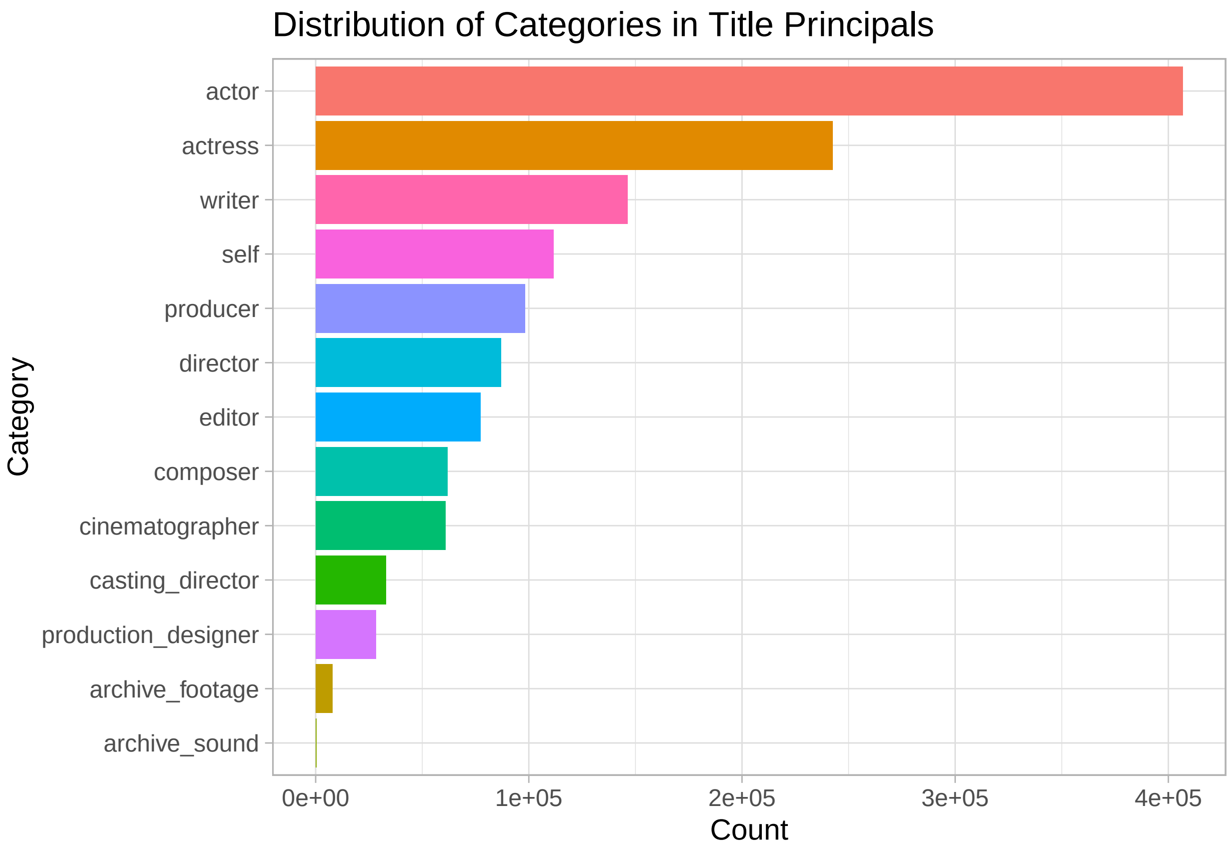
```
## Warning: Removed 1408 rows containing non-finite outside the scale range
## (`stat_count()`).
```




```
# Plot 2: Runtime Distribution Faceted by Adult Status (title_basics)
title_basics %>%
  filter(!is.na(runtimeMinutes), runtimeMinutes < 300) %>%  # limit extreme outliers
  ggplot(aes(x = runtimeMinutes, fill = as.factor(isAdult))) +
  geom_histogram(bins = 50, alpha = 0.6, position = "identity") +
  facet_wrap(~isAdult, labeller = labeller(isAdult = c(`0` = "Non-Adult", `1` = "Adult"))) +
  labs(title = "Runtime Minutes Distribution by Adult Status", x = "Runtime (Minutes)", fill = "Is Adul
  theme_bw()
```

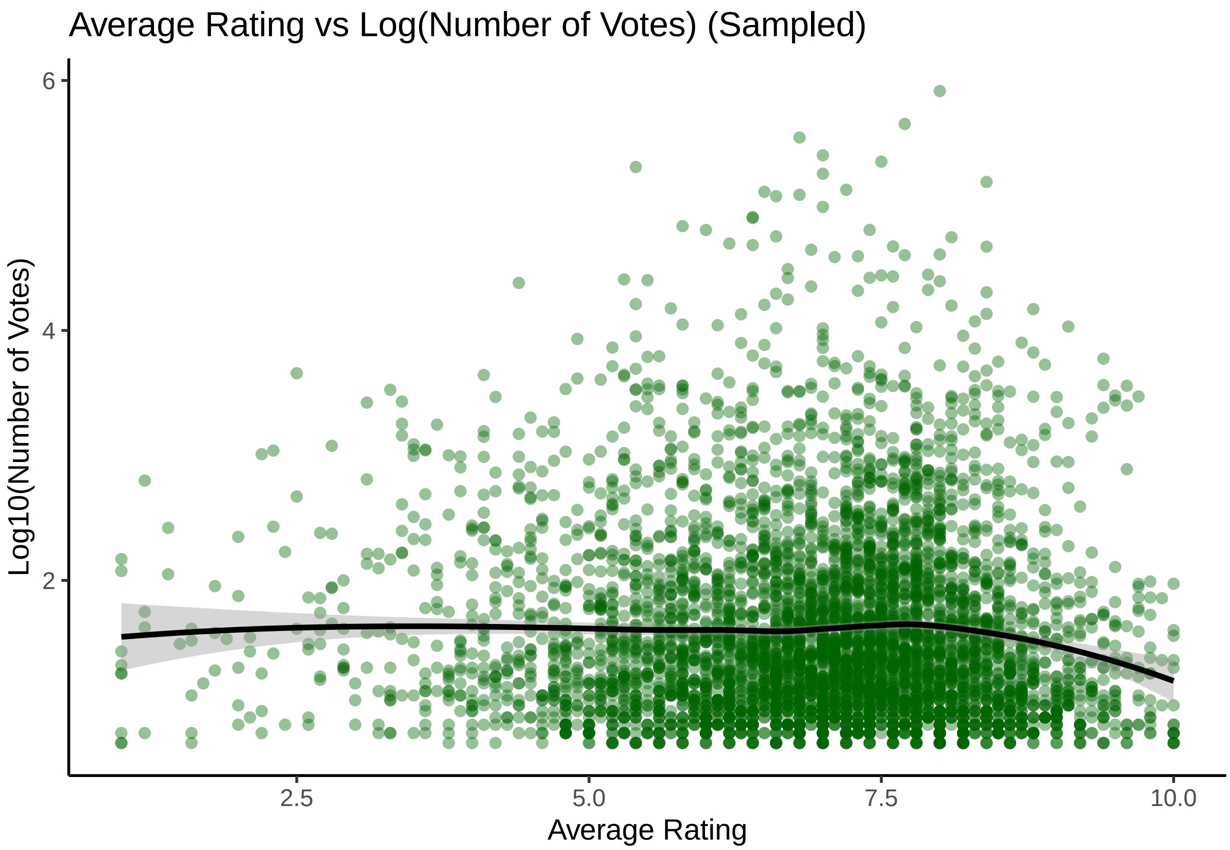



```r
# Plot 3: Categories in Title Principals (title_principals)
title_principals %>%
  count(category) %>%
  ggplot(aes(x = reorder(category, n), y = n, fill = category)) +
  geom_col(show.legend = FALSE) +
  coord_flip() +
  labs(title = "Distribution of Categories in Title Principals", x = "Category", y = "Count") +
  theme_light()
```




```r
# Plot 4 : Smoothed Relationship between Rating and Votes (sampled)
set.seed(123)  # for reproducibility
title_ratings_sampled <- title_ratings %>% sample_n(5000)

ggplot(title_ratings_sampled, aes(x = averageRating, y = log10(numVotes))) +
  geom_point(alpha = 0.4, color = "darkgreen") +
  stat_smooth(method = "loess", se = TRUE, color = "black") +
  labs(title = "Average Rating vs Log(Number of Votes) (Sampled)", x = "Average Rating", y = "Log10(Num
  theme_classic()
```

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



Plot 1: This bar chart, titled "Distribution of # Titles Known For," illustrates how many individuals in the IMDB dataset are primarily known for a specific number of film or television titles. The x-axis represents the count of "# Titles," ranging from 1 to 4, while the y-axis displays the corresponding "Count" of individuals. The most striking observation is the overwhelmingly large bar at "# Titles = 4," very large in comparison to the counts for individuals known for 1, 2, or 3 titles. This suggests that a substantial majority of the individuals in this dataset are primarily associated with exactly four titles.

Plot 2: The runtime distribution differs greatly between non-adult and adult titles in the IMDB dataset. Non-adult titles show a wider range of runtimes, peaking around typical film and episode lengths, while adult titles are predominantly much shorter. Adult content also has a considerably lower overall count compared to non-adult content in this sample.

Plot 3: The bar chart, "Distribution of Categories in Title Principals," highlights the frequency of different roles credited in film and television productions within the IMDB dataset. "actor" and "actress" are the most prevalent categories, indicating the large number of individuals credited in these on-screen roles. Following these, "writer" represents the next most common role, showing the significant number of individuals involved in script creation. The subsequent categories, such as "self," "producer," and "director," show decreasing counts, underscoring more specialized roles with fewer individuals typically involved per production, while categories like "archive_footage" have the lowest representation.

Plot 4: The scatter plot of sampled IMDB data reveals a nuanced relationship between average rating and the logarithm of the number of votes. Generally, movies with slightly higher average ratings tend to receive more votes, but this trend plateaus and slightly declines for the highest-rated films. Notably, movies with similar average ratings exhibit a wide range in the number of votes, indicating that factors beyond just the rating significantly influence a film's popularity on IMDB.

```r
# For the titles_basics table: Group by two variables and explore runtimeMinutes
title_basics %>%
  filter(!is.na(runtimeMinutes)) %>%
```

```r
  group_by(titleType, isAdult) %>%
  summarise(mean_runtime = mean(runtimeMinutes, na.rm = TRUE), .groups = "drop")
```

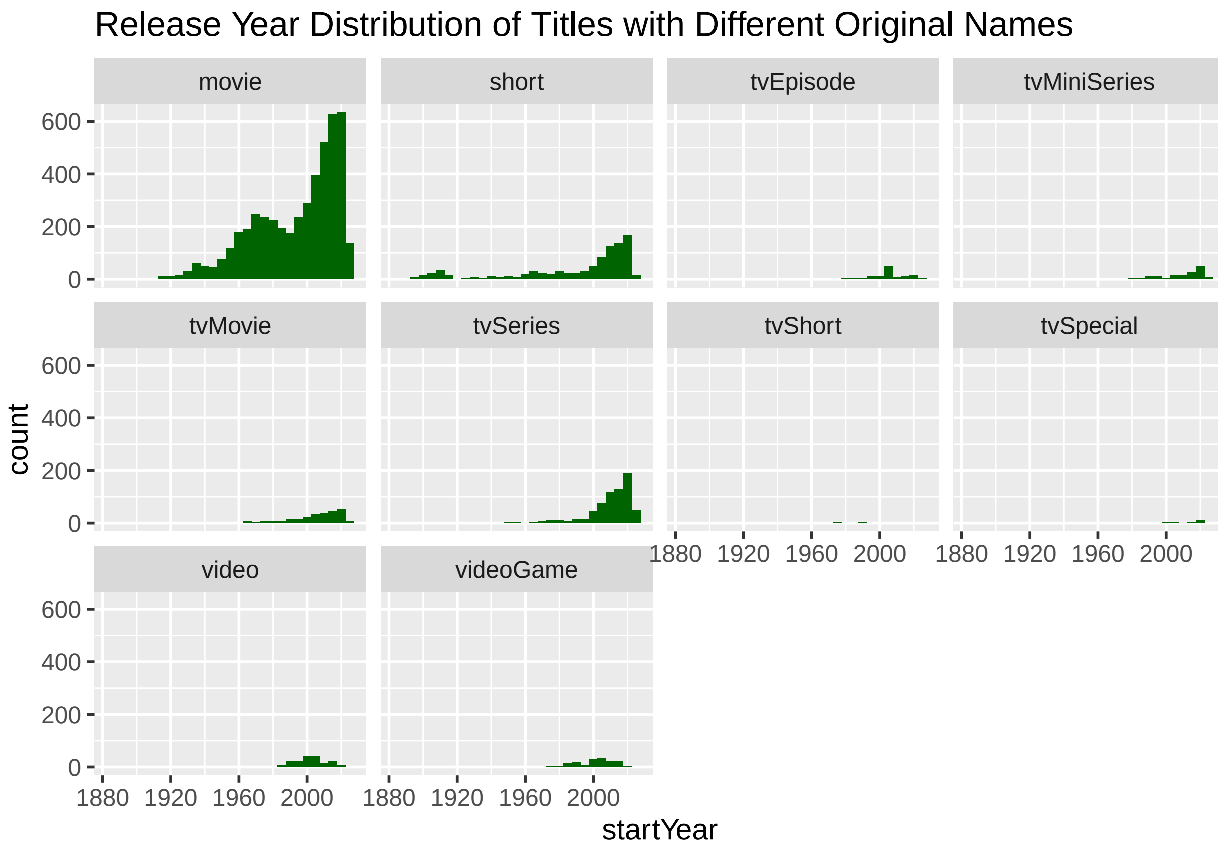
```
## # A tibble: 18 x 3
##    titleType    isAdult mean_runtime
##    <chr>          <dbl>        <dbl>
##  1 movie              0         94.3
##  2 movie              1         77.2
##  3 short              0         14.2
##  4 short              1         12
##  5 tvEpisode          0         38.9
##  6 tvEpisode          1         49.4
##  7 tvMiniSeries       0        111.
##  8 tvMiniSeries       1         93.7
##  9 tvMovie            0         81.1
## 10 tvMovie            1         89
## 11 tvSeries           0         49.1
## 12 tvSeries           1         35.3
## 13 tvShort            0         12.4
## 14 tvSpecial          0         89.8
## 15 tvSpecial          1         71
## 16 video              0         54.5
## 17 video              1        113.
## 18 videoGame          0        158.
```

```r
# How many titles are known for a name that is different than the original release name?
title_basics %>%
  filter(primaryTitle != originalTitle) %>%
  summarise(n_diff = n())
```

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

```r
# Graph conditional distributions of release year based on titles with different original names
ggplot(
  title_basics %>% filter(primaryTitle != originalTitle),
  aes(x = startYear)
) +
  geom_histogram(binwidth = 5, fill = "darkgreen") +
  labs(title = "Release Year Distribution of Titles with Different Original Names") +
  facet_wrap(~ titleType)
```

```
## Warning: Removed 1 row containing non-finite outside the scale range
## (`stat_bin()`).
```



A prominent trend across several title types, particularly "movie," "short," "tvEpisode," and "tvSeries," is a noticeable increase in the number of releases in more recent years, generally after the year 2000. This suggests a significant growth in content production across these formats in the 21st century. For "movie," there's an upward trend in releases starting around the mid-20th century, with a significant growth in the late 20th and early 21st centuries. "Short" films also show a concentration of releases in more recent times, although the overall volume is much lower than movies. Likewise, "tvEpisode" and "tvSeries" counts appear to increase notably in the later part of the 20th and the beginning of the 21st century.

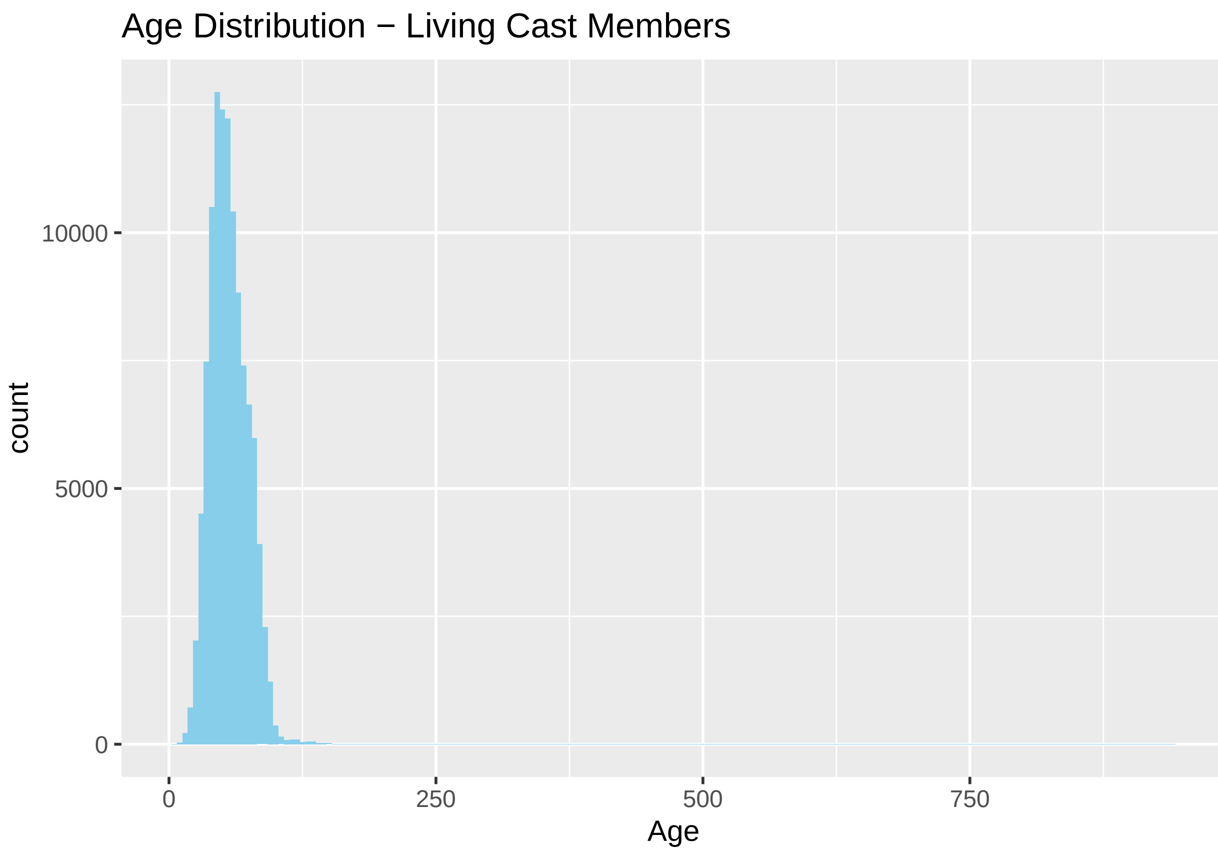
In contrast, title types like "tvMovie," "tvMiniSeries," "tvShort," "tvSpecial," "video," and "videoGame" show relatively lower counts overall, and their release year distributions don't show the same increasing trend towards the present day as seen in movies and more regular television formats. This could indicate different production patterns or cataloging practices for these types of titles within the IMDB database.

```r
# For the ratings: cut into three groups and analyze numVotes
title_ratings$rating_group <- cut(title_ratings$averageRating, breaks = 3)

title_ratings %>%
  group_by(rating_group) %>%
  summarise(
    mean_votes = mean(numVotes, na.rm = TRUE),
    median_votes = median(numVotes, na.rm = TRUE)
  )
```

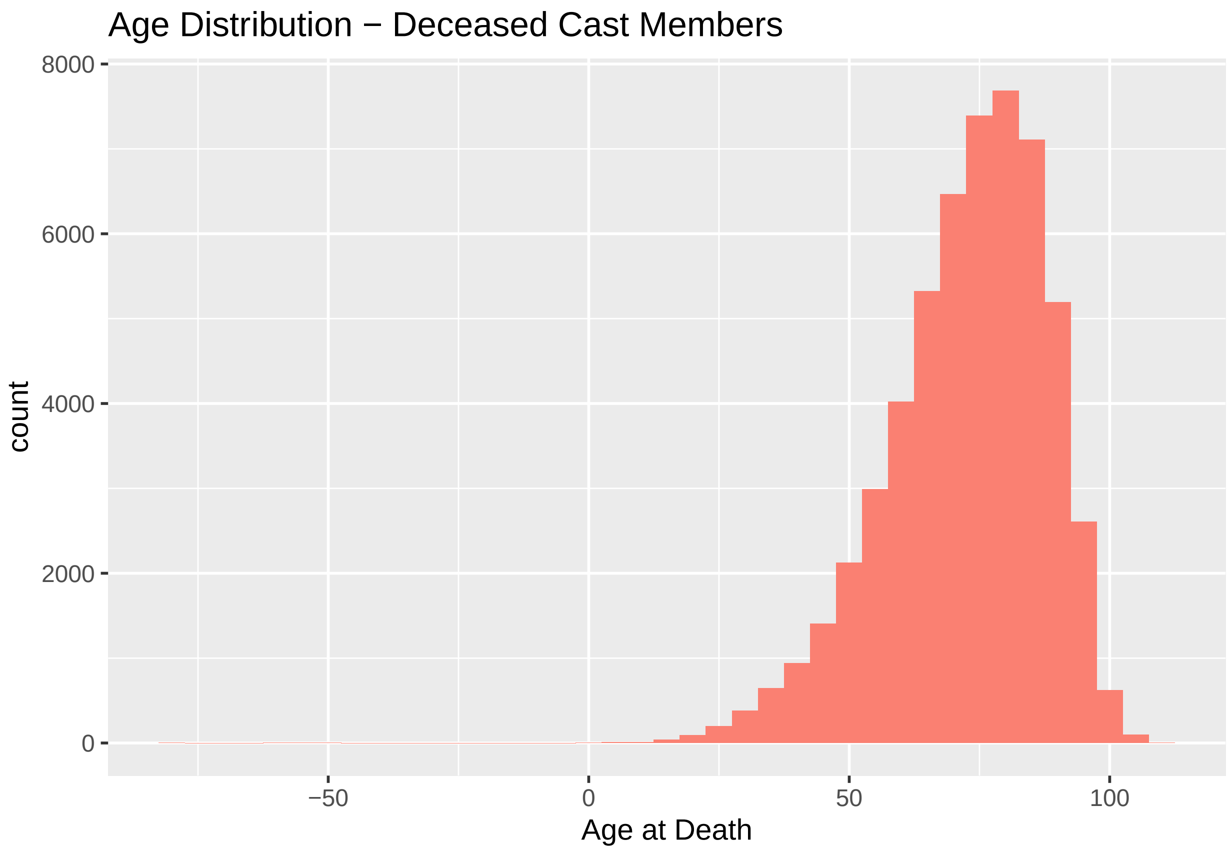
```
## # A tibble: 3 x 3
##   rating_group mean_votes median_votes
##   <fct>             <dbl>        <dbl>
## 1 (0.991,4]          339.           29
## 2 (4,7]              886.           27
## 3 (7,10]            1099.           25
```

Higher rated titles rated more often than lower rated titles.

```r
# For the names table: investigate age of cast members (Living vs Deceased)

# Add status and age
name_basics <- name_basics %>%
  mutate(
    birthYear = as.numeric(birthYear),
    deathYear = as.numeric(deathYear),
    age = deathYear - birthYear,
    status = ifelse(is.na(deathYear), "Living", "Deceased")
  )

# Plot: Living cast members' age distribution
ggplot(filter(name_basics, status == "Living", !is.na(birthYear)), aes(x = 2025 - birthYear)) +
  geom_histogram(binwidth = 5, fill = "skyblue") +
  labs(title = "Age Distribution - Living Cast Members", x = "Age")
```




```r
# Plot: Deceased cast members' age distribution
ggplot(filter(name_basics, status == "Deceased", !is.na(age)), aes(x = age)) +
  geom_histogram(binwidth = 5, fill = "salmon") +
  labs(title = "Age Distribution - Deceased Cast Members", x = "Age at Death")
```



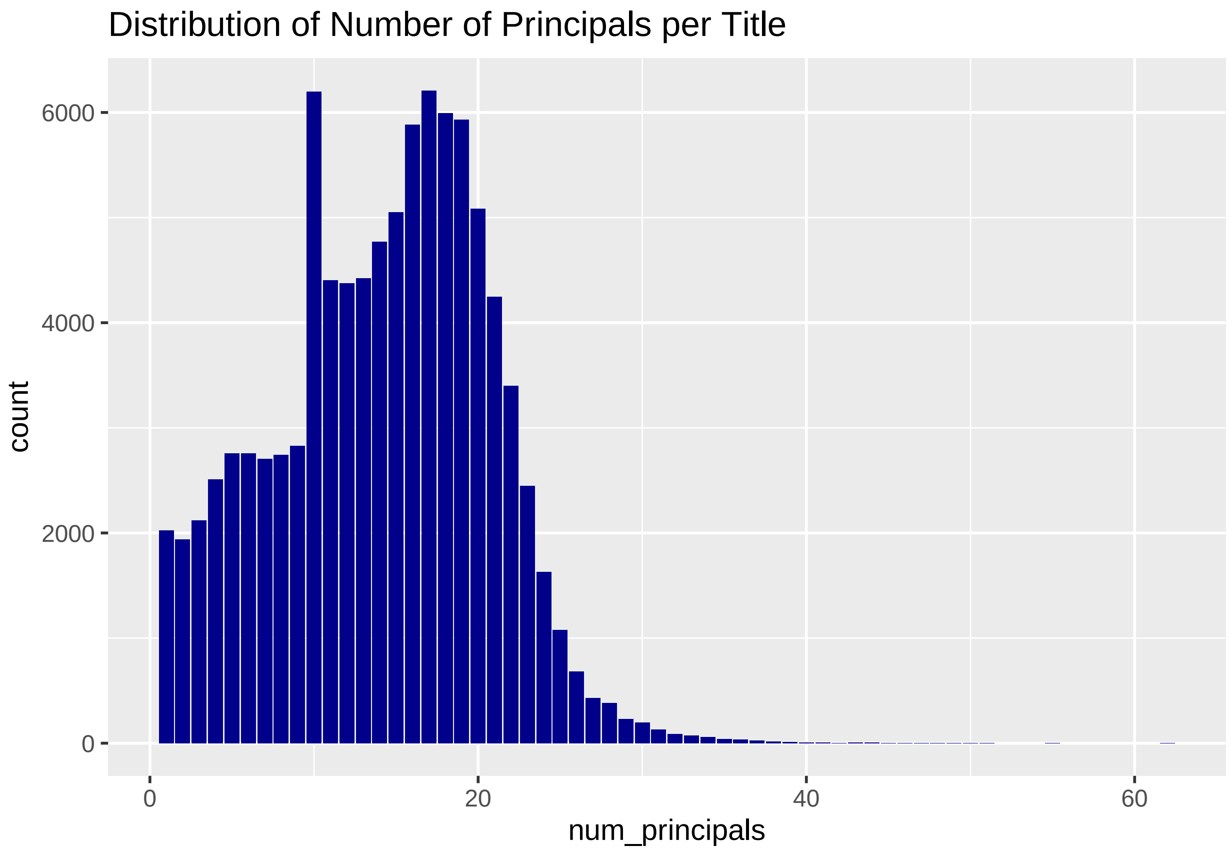

```r
# Find all actors with first names "Tom", "Thomas", "Thom" or "Tomas"

name_basics %>%
  filter(str_detect(primaryName, "^Thom|^Tom|^Tomas|^Thomas")) %>%
  summarise(count = n())
```

```
## # A tibble: 1 x 1
##   count
##   <int>
## 1  4331
```

```r
# How many titles use alliteration
is_alliterative <- function(title) {
  words <- unlist(str_split(title, "\\s+"))
  letters <- str_sub(words, 1, 1)
  length(unique(letters)) == 1
}

title_basics %>%
  filter(!is.na(primaryTitle)) %>%
  mutate(alliterative = sapply(primaryTitle, is_alliterative)) %>%
  summarise(total_alliterative = sum(alliterative))
```

```
## # A tibble: 1 x 1
##   total_alliterative
##                <int>
## 1              16351
```

```r
# Distribution of number of principals per title
title_principals %>%
  group_by(tconst) %>%
  summarise(num_principals = n()) %>%
  ggplot(aes(x = num_principals)) +
  geom_bar(fill = "darkblue") +
  labs(title = "Distribution of Number of Principals per Title")
```

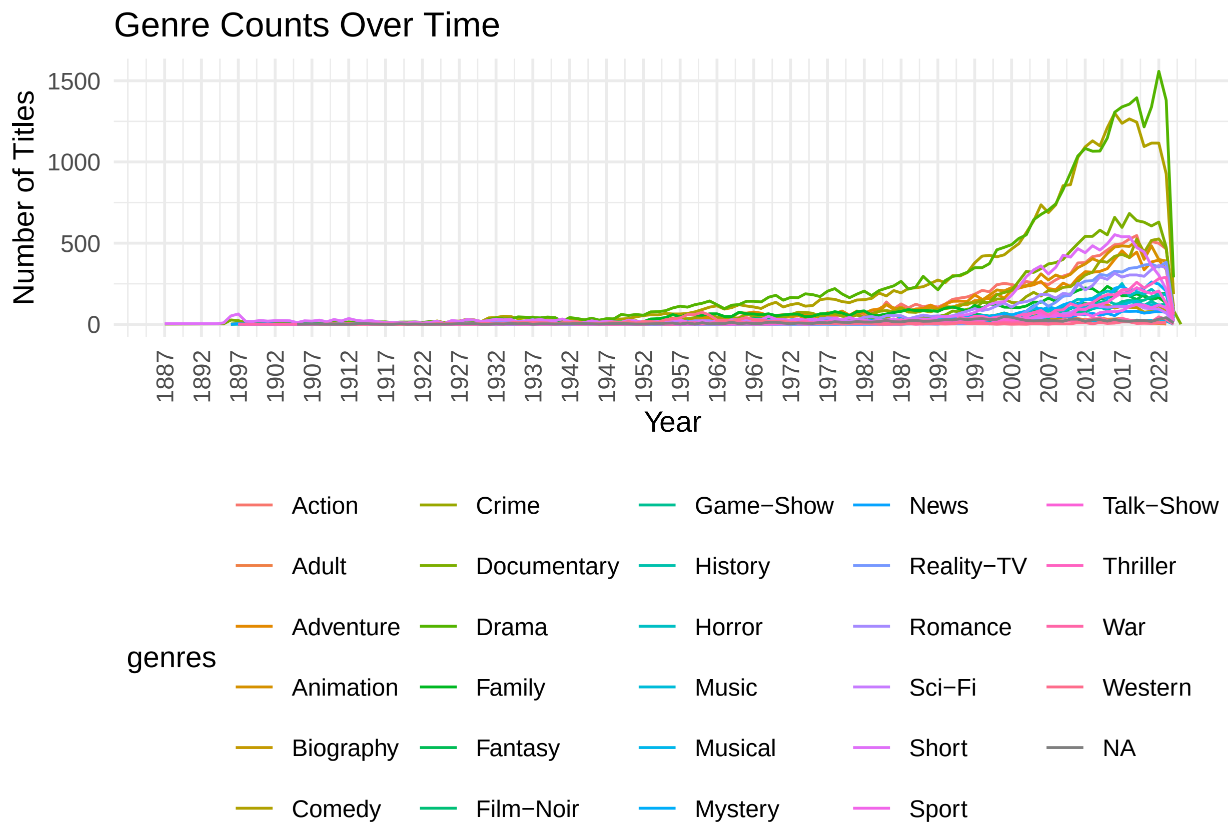


Part III: Pivoting

- 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.
- 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).
- 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)

```r
 library(dplyr)
library(ggplot2)

title_genres <- title_basics |>
  separate_rows(genres, sep = ",") |>
  mutate(startYear = as.numeric(startYear)) |>
  filter(!is.na(startYear))

genre_count_per_year <- title_genres |>
  group_by(startYear, genres) |>
```

```r
  summarise(count = n(), .groups = "drop")

ggplot(genre_count_per_year, aes(x = startYear, y = count, color = genres)) +
  geom_line() +
  labs(title = "Genre Counts Over Time", x = "Year", y = "Number of Titles") +
  theme_minimal() +
  theme(
    legend.position = "bottom",
    axis.text.x = element_text(angle = 90, vjust = 0.5, hjust = 1)  # <-- rotate x labels
  ) +
  scale_x_continuous(breaks = seq(min(genre_count_per_year$startYear), max(genre_count_per_year$startYe
```




```r
genre_matrix <- model.matrix(~ genres - 1, data = title_genres)
genre_df <- as.data.frame(genre_matrix)
cor_matrix <- cor(genre_df)
```

Part IV: Joining Tables

- Join the table with one title-genre per row from the previous section with the ratings table.
  - What is the highest rated genre? What is the lowest rated genre?
  - 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.
- 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.
- 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.

    - Find the proportions of genres for the the titles that include the top 10 rated principals.
    - Graph ratings against years. What trends do you see?
- 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.
-

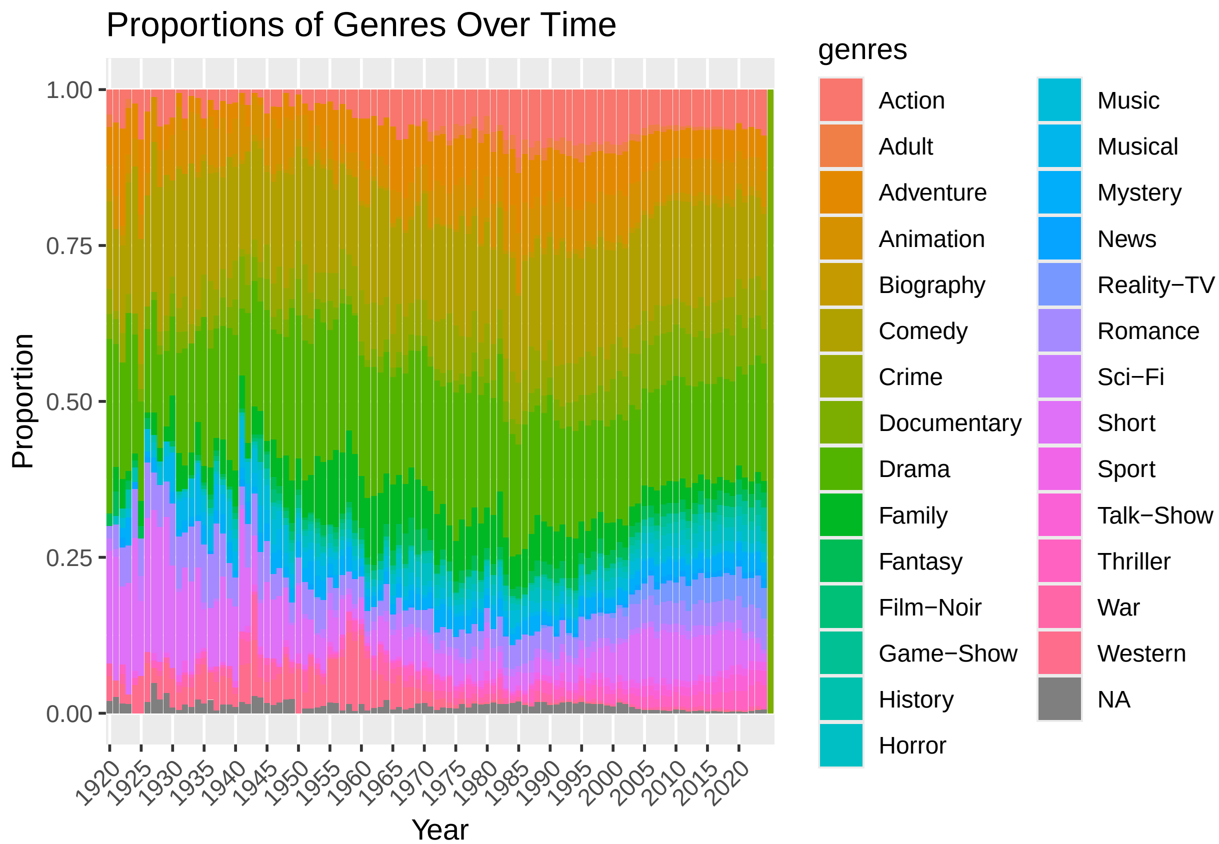
```r
genre_ratings <- title_genres |>
  inner_join(title_ratings, by = "tconst")

genre_summary <- genre_ratings |>
  group_by(genres) |>
  summarise(mean_rating = mean(averageRating, na.rm = TRUE)) |>
  arrange(desc(mean_rating))

head(genre_summary, 1)
```

```
## # A tibble: 1 x 2
##   genres  mean_rating
##   <chr>         <dbl>
## 1 History        7.33
```

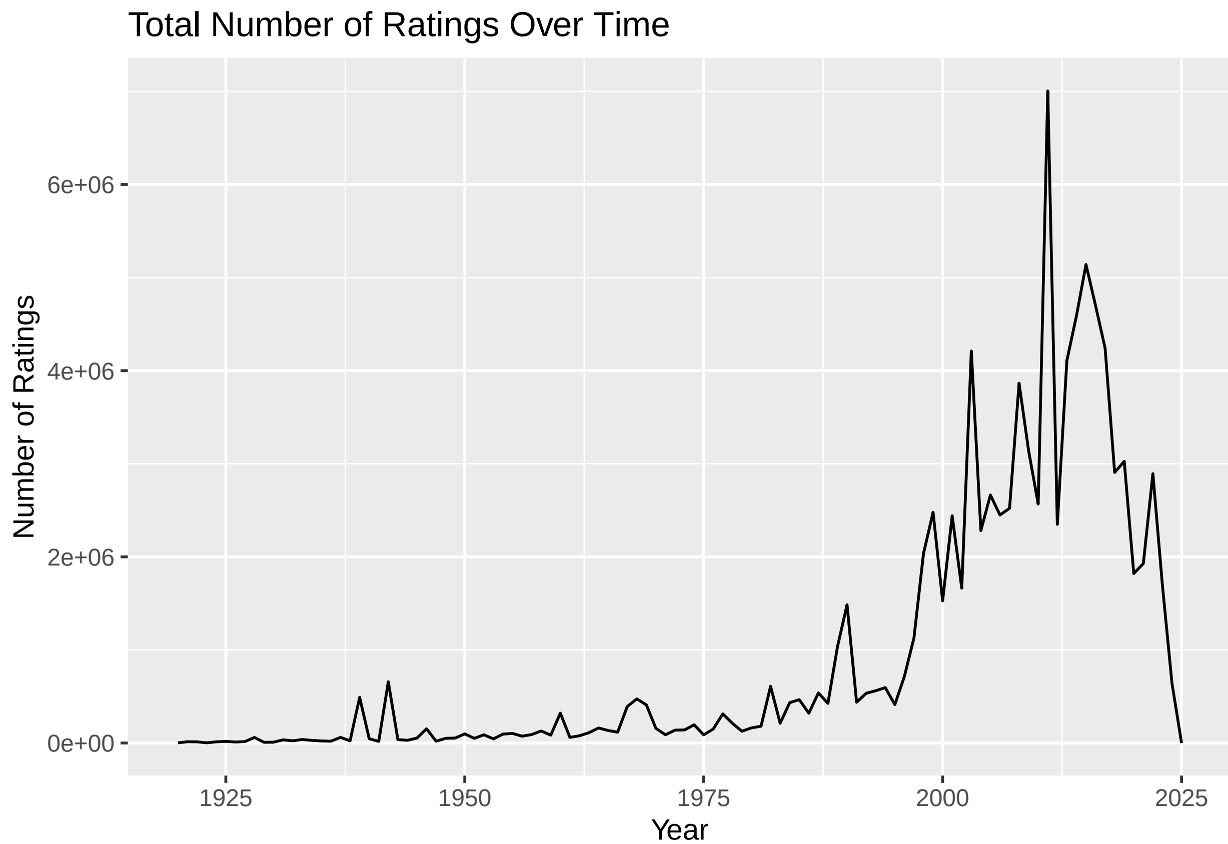
```r
tail(genre_summary, 1)
```

```
## # A tibble: 1 x 2
##   genres mean_rating
##   <chr>        <dbl>
## 1 Horror        6.16
```

```r
genre_ratings_clean <- genre_ratings |>
  filter(!is.na(startYear), startYear >= 1920, startYear <= 2025)

years_to_show <- seq(1920, 2020, by = 5)

plot1 <- ggplot(genre_ratings_clean, aes(x = factor(startYear), fill = genres)) +
  geom_bar(position = "fill") +
  scale_x_discrete(breaks = years_to_show) +
  labs(title = "Proportions of Genres Over Time",
       x = "Year", y = "Proportion") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
plot1
```




```
title_with_ratings <- title_basics |>
  inner_join(title_ratings, by = "tconst") |>
  filter(!is.na(startYear), startYear >= 1920, startYear <= 2025)

ratings_per_year <- title_with_ratings |>
  group_by(startYear) |>
  summarise(total_votes = sum(numVotes, na.rm = TRUE)) |>
  arrange(desc(total_votes))

plot2 <- ggplot(ratings_per_year, aes(x = startYear, y = total_votes)) +
  geom_line() +
  labs(title = "Total Number of Ratings Over Time",
       x = "Year", y = "Number of Ratings")
plot2
```




```r
numeric_summary <- ratings_per_year |>
  summarise(
    earliest_year = min(startYear),
    latest_year = max(startYear),
    peak_year = startYear[which.max(total_votes)],
    peak_votes = max(total_votes),
    median_votes = median(total_votes),
    total_all_years = sum(total_votes)
  )

print(numeric_summary)
```

```
## # A tibble: 1 x 6
##   earliest_year latest_year peak_year peak_votes median_votes total_all_years
##           <dbl>       <dbl>     <dbl>      <dbl>        <dbl>           <dbl>
## 1          1920        2025      2011    7004647       171101        97848202
```

```r
title_principals_filtered <- title_principals |>
  filter(category %in% c("actor", "actress")) |>
  select(tconst, nconst)

title_ratings_filtered <- title_ratings |>
  filter(numVotes >= 100) |>
  select(tconst, averageRating)

name_ratings_small <- title_principals_filtered |>
  inner_join(title_ratings_filtered, by = "tconst") |>
  inner_join(name_basics |> select(nconst, primaryName), by = "nconst") |>
```

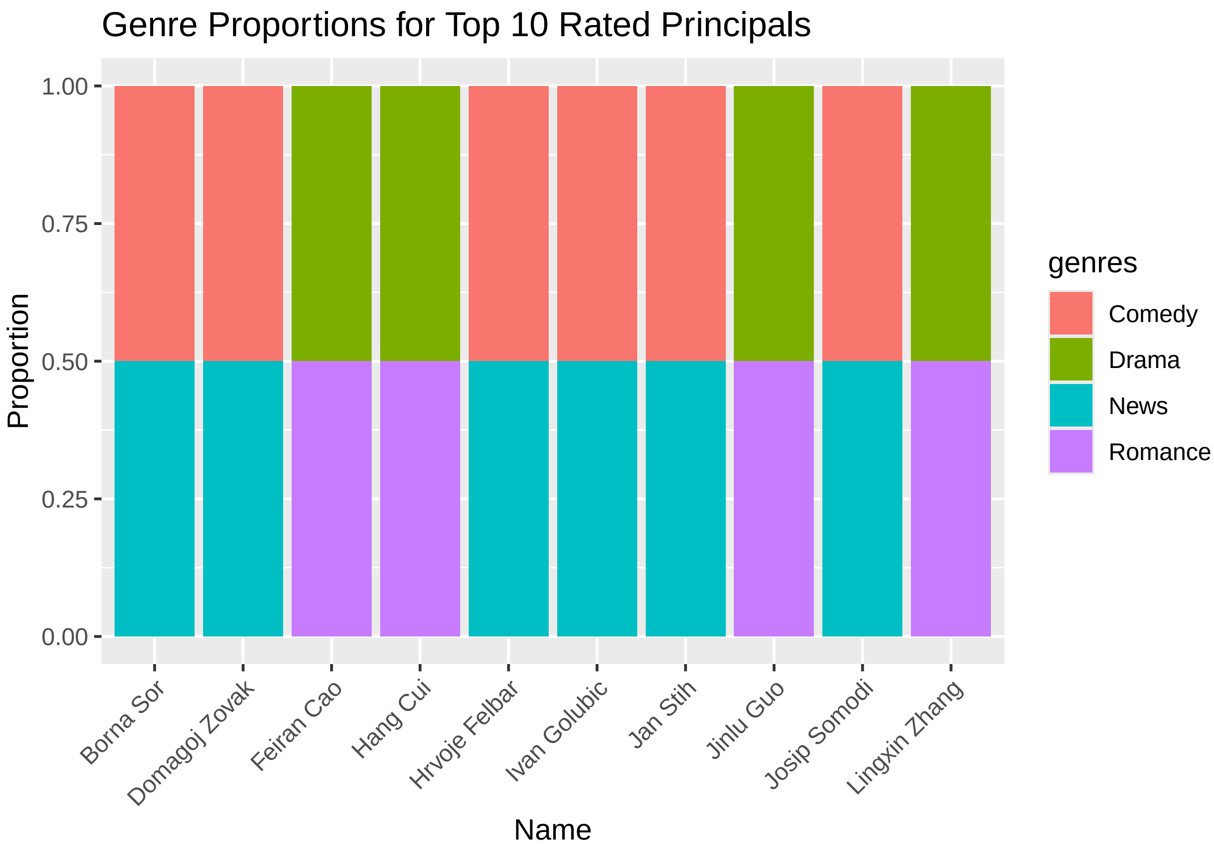
```r
  select(nconst, tconst, primaryName, averageRating)

top_people <- name_ratings_small |>
  group_by(primaryName) |>
  summarise(
    median_rating = median(averageRating, na.rm = TRUE),
    count = n()
  ) |>
  filter(count >= 3) |>
  arrange(desc(median_rating)) |>
  slice_head(n = 10)

top10_names <- top_people$primaryName
top10_names
```

```
##  [1] "Borna Sor"     "Domagoj Zovak" "Feiran Cao"    "Hang Cui"
##  [5] "Hrvoje Felbar" "Ivan Golubic"  "Jan Stih"      "Jinlu Guo"
##  [9] "Josip Somodi"  "Lingxin Zhang"
```

```r
top10_data_small <- name_ratings_small |>
  filter(primaryName %in% top10_names)

small_title_genres <- title_genres |>
  filter(tconst %in% top10_data_small$tconst)

top10_data <- top10_data_small |>
  inner_join(small_title_genres, by = "tconst")
```

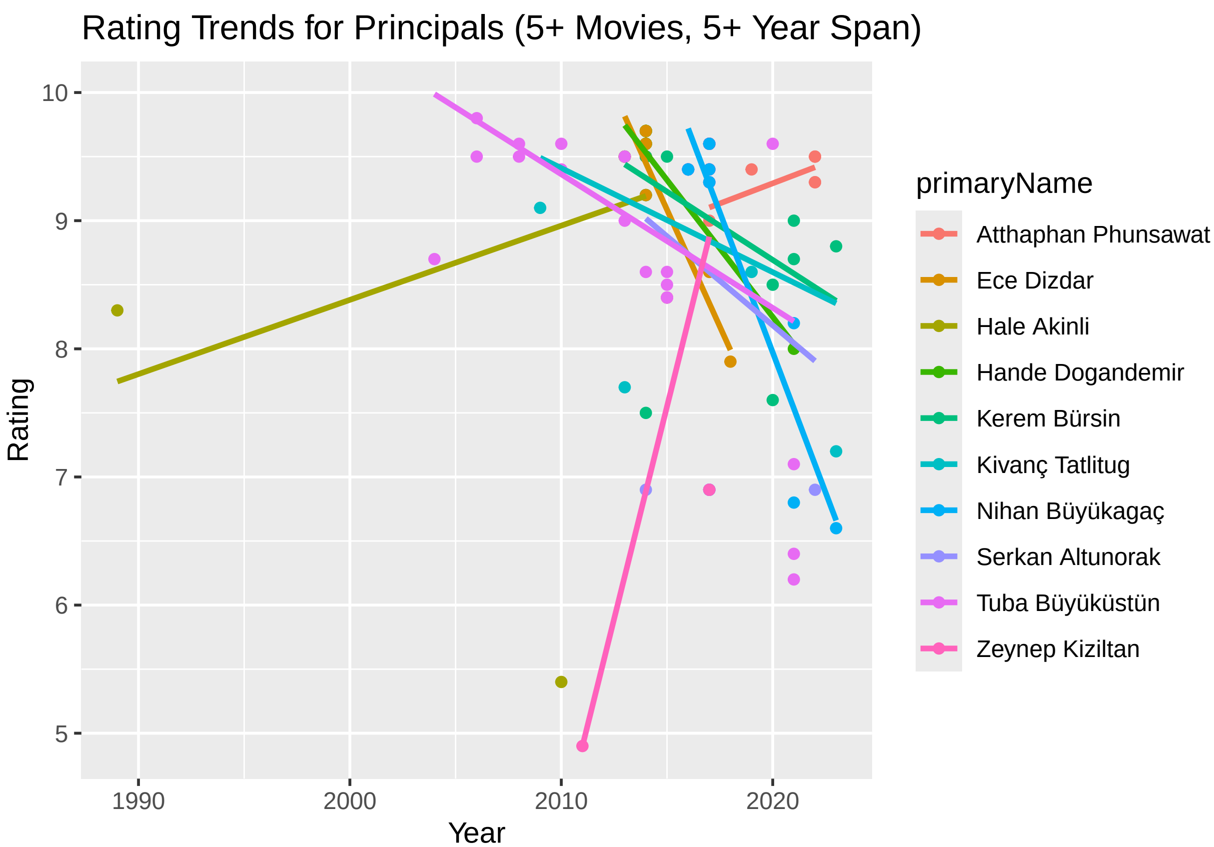
```
## Warning in inner_join(top10_data_small, small_title_genres, by = "tconst"): Detected an unexpected ma
## i Row 1 of `x` matches multiple rows in `y`.
## i Row 1 of `y` matches multiple rows in `x`.
## i If a many-to-many relationship is expected, set `relationship =
##   "many-to-many"` to silence this warning.
```

```r
plot3 <- ggplot(top10_data, aes(x = primaryName, fill = genres)) +
  geom_bar(position = "fill") +
  labs(
    title = "Genre Proportions for Top 10 Rated Principals",
    x = "Name", y = "Proportion"
  ) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

print(plot3)
```




```r
better_principals <- name_ratings_small |>
  inner_join(title_basics |> select(tconst, startYear), by = "tconst") |>
  filter(!is.na(startYear)) |>
  group_by(primaryName) |>
  summarise(
    median_rating = median(averageRating, na.rm = TRUE),
    n_movies = n(),
    career_span = max(startYear) - min(startYear)
  ) |>
  filter(
    n_movies >= 5,
    career_span >= 5
  ) |>
  arrange(desc(median_rating)) |>
  slice_head(n = 10)

trend_data <- name_ratings_small |>
  inner_join(title_basics |> select(tconst, startYear), by = "tconst") |>
  filter(primaryName %in% better_principals$primaryName)

ggplot(trend_data, aes(x = startYear, y = averageRating, color = primaryName)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +  # Linear trend line
  labs(
    title = "Rating Trends for Principals (5+ Movies, 5+ Year Span)",
    x = "Year", y = "Rating"
  )
```

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



Most people seem to experience a decrease in ratings. However, a few experienced an increase, mainly those with more years of experience.

```
known_for_titles <- name_basics |>
  select(nconst, primaryName, knownForTitles)

known_for_ratings <- known_for_titles |>
  inner_join(title_ratings |> filter(numVotes >= 100), by = c("knownForTitles" = "tconst")) |>
  select(nconst, primaryName, knownForTitles, averageRating)

median_ratings <- known_for_ratings |>
  group_by(primaryName) |>
  summarize(median_rating = median(averageRating, na.rm = TRUE), .groups = "drop")

highest_median_person <- median_ratings |>
  filter(median_rating == max(median_rating))

print(highest_median_person)
```

```
## # A tibble: 9 x 2
##   primaryName      median_rating
##   <chr>                    <dbl>
## 1 Anne Pearson               9.9
## 2 David Pisarra              9.9
## 3 Douglas C. Wicks           9.9
## 4 Hrvoje Felbar              9.9
## 5 Jule Lamm                  9.9
```

```
## 6 Mario Vukelic              9.9
## 7 Mato Filipovic             9.9
## 8 Ray Hoogenraad             9.9
## 9 The Darlings               9.9
```

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

```
profvis({
  analyze_genres(title_basics)
})
```

Running profvis on the function "anaylze_genres" which is responsible for part 3 in the project, we see that the function takes the longest in as.data.frame.matrix part which is responsible for for converting a matrix into a data frame in R. Since the data is so large, this takes the longest time. A way to fasten this performance is to use "data.table" instead of as.data.frame. This would fasten it as it avoids copying columns unnecessarily.

```
# Optimized version of analyze_genres
optimized_analyze_genres <- function(title_basics) {
  title_genres <- title_basics %>%
    as.data.table() %>%
    separate_rows(genres, sep = ",") %>%
    mutate(startYear = as.numeric(startYear)) %>%
    filter(!is.na(startYear)) %>%
    as.data.table()

  genre_count_per_year <- title_genres |>
    group_by(startYear, genres) |>
    summarise(count = n(), .groups = "drop")

  plot <- ggplot(genre_count_per_year, aes(x = startYear, y = count, color = genres)) +
    geom_line() +
    labs(title = "Genre Counts Over Time", x = "Year", y = "Number of Titles") +
    theme_minimal() +
    theme(
      legend.position = "bottom",
      axis.text.x = element_text(angle = 90, vjust = 0.5, hjust = 1)  # <-- rotate x labels
    ) +
    scale_x_continuous(breaks = seq(min(genre_count_per_year$startYear), max(genre_count_per_year$start

  print(plot)

  genre_matrix <- model.matrix(~ genres - 1, data = title_genres)
  genre_df <- as.data.table(genre_matrix)
```

```
  cor_matrix <- cor(genre_df)
}
```

```
profvis({
  optimized_analyze_genres(title_basics)
})
```

This optimization reduces the time for that to 1160s as opposed to the unoptimized function which takes around 1570s to run.

Looking into parts that can be fastened using parallelization, we look at the different functions that are run. In part 4, the task of joining the names with the ratings and the principals table and then grouping by individual people and finding the top ten people based on the median rating of the titles they appear in would benefit the most from parallelization.

```
library(future)
library(furrr)

find_top10_people <- function(title_principals, title_ratings, name_basics, min_titles = 3, top_n = 10)

  # Parallel setup
  plan(multisession)

  name_ratings <- title_principals |>
    inner_join(title_ratings, by = "tconst") |>
    inner_join(name_basics, by = "nconst")

  name_groups <- name_ratings |>
    group_split(primaryName)

  people_stats <- future_map_dfr(name_groups, function(df) {
    tibble(
      primaryName = unique(df$primaryName),
      median_rating = median(df$averageRating, na.rm = TRUE),
      count = n()
    )
  })

  top_people <- people_stats |>
    filter(count >= min_titles) |>
    arrange(desc(median_rating)) |>
    slice_head(n = top_n)

  top10_names <- top_people$primaryName

  top10_titles <- name_ratings |>
    filter(primaryName %in% top10_names) |>
    select(primaryName, tconst)

  return(top10_titles)
}
```

The time taken by the parallelized version is lesser than the general version which was the expected result.

One task we performed involved counting items in strings separated by commas.Two different functions that could perform this tasks are shown below.

```r
# Version 1: Using str_count
count_items_v1 <- function(df) {
  df |>
    mutate(genre_count = str_count(genres, ",") + 1) |>
    select(tconst, genre_count)
}

# Version 2: Using strsplit
count_items_v2 <- function(df) {
  df |>
    rowwise() |>
    mutate(genre_count = length(unlist(strsplit(genres, ",")))) |>
    ungroup() |>
    select(tconst, genre_count)
}

# Benchmarking the two functions
library(rbenchmark)

benchmark(
  strsplit_version = count_items_v1(title_basics),
  separate_rows_version = count_items_v2(title_basics)
)
```

```
##                    test replications elapsed relative user.self sys.self
## 2 separate_rows_version          100 117.724   60.248   116.730    0.167
## 1      strsplit_version          100   1.954    1.000     1.924    0.016
##   user.child sys.child
## 2          0         0
## 1          0         0
```

Based on the benchmark results, we see that the strsplit function is better as it is faster and takes up less memory generally.

Part VI: Shiny Applications

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.

###EMAILED BY VARAJESH@UMICH.EDU

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.

###EMAILED BY VARAJESH@UMICH.EDU

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.

###EMAILED BY VARAJESH@UMICH.EDU