STSCI 4100: IDMb Analysis Project

Tony Oh (do256), William Rhee (wr86)

2025-05-16

TABLE OF CONTENTS

[1] INTRODUCTION:

IMDb documents reviews of released movies and films in non-commercial datasets — these datasets contain vast amounts of metadata regarding viewer ratings, genres, release years, and more. In this project, we want to understand if these factors dictate any trends in film popularity and genre success.

In this project, we will be using four datasets from IMDb:

1. **title.basics.tsv**: A dataset providing all the basic details about a broad range of films from 1892 to 2026.
2. **title.crew.tsv**: A dataset providing the ID's specifically of the director and writers for the movies from Dataset 1.
3. **title.principals.tsv**: A dataset providing details about other crew (editors, producers, etc.) for the movies from Dataset 1.
4. **title.ratings.tsv**: A dataset that contains the rating data from IMDb's users for the movies from Dataset 1,

Our analysis focuses on a **wide range of films** — from shorts, TV series, movies, and even video content — in the period **1892 to 2026**. We will be answering **four questions** to explore trends in ratings, creative roles, and genres across the whole IMDb dataset.

1. **Ratings**: Does the number of votes per film affect the ratings?
2. **Creative role (writer)**: Does having a writer affect the ratings?
3. **Actor roles**: Does having certain actors have an affect on ratings?
4. **Genres**: Which genres get the highest ratings?

That being said, let's start with data exploration.

---

[2] DATA EXPLORATION:

There are four points in this section:

A. Importing the Datasets.
B. A Quick Glance.
C. Specific Breakdowns.
D. Combining the Datasets.

[A] Importing the datasets.

Firstly, to reproduce this project, clone our Git repository found at https://github.com/thetunr/stsci4100. Then, open this file (`analysis.Rmd`) and set your working directory to this project's root directory.

We provide two methods in importing and saving the datasets.

1. Download the exact RDS files we worked on. This method will be most accurate with the results we found in our analysis. To access these files, download the four RDS files from https://drive.google.com/drive/folders/1IQp6n6-yVzPhkcDjmhk52B3E0fm8emWi?usp=sharing. Once the download is complete, place the four RDS files into the folder `./rds` of the project's root directory.

- **IMPORTANT NOTE**: Note that this set of datasets is a result of subsetting the rows of each dataset in `import_save.Rmd` to match the `tconst` values of `ratings` since `ratings` is the limiting dataset in number of rows. That is, we deemed `ratings` to hold information about films that is necessary for all of our analysis. Thus, we removed rows in `basics`, `crew`, and `principals` that represented films that were not represented in `ratings`. A majority of films in `basics`, `crew`, and `principals` that **did not have ratings** were removed according to the films available in `ratings`. More detail about how this choice was executed can be found in `import_save.Rmd`.

2. Download the updated IMDb dataset from the IMDb source. This method may not align with the results we found in our analysis. To use this method of retrieving the datasets, follow the instructions detailed in `import_save.Rmd`.

Once you have the RDS files set up in the `./rds` directory, run the code chunk below to read in the RDS data.

```
library(data.table)
library(mgcv)
```

```
## Loading required package: nlme
```

```
## This is mgcv 1.9-1. For overview type 'help("mgcv-package")'.
```

```
gc() # Clear unused memory
```

```
##           used (Mb) gc trigger (Mb) limit (Mb) max used  (Mb)
## Ncells 1658084 88.6    2695778  144         NA  2695778 144.0
## Vcells 2973532 22.7    8388608   64      16384  4755661  36.3
```

```
# SET WORKING DIRECTORY TO PROJECT ROOT IF NOT YET DONE SO
```

```
rds_dir <- "rds/"
basics <- readRDS(paste0(rds_dir, "basics.rds"));
crew <- readRDS(paste0(rds_dir, "crew.rds"));
principals <- readRDS(paste0(rds_dir, "principals.rds"));
ratings <- readRDS(paste0(rds_dir, "ratings.rds"));

list.of.datasets <- list(basics, crew, principals, ratings)
attr(list.of.datasets, "names") <- c("basics", "crew", "principals", "ratings")
```

[B] A Quick Glance.

```
## [1] "In the basics dataset, there are 9 columns."
## [1] "Its columns are: tconst, titleType, primaryTitle, originalTitle, isAdult, startYear, endYear, ru
##
## [1] "In the crew dataset, there are 3 columns."
## [1] "Its columns are: tconst, directors, writers"
##
## [1] "In the principals dataset, there are 6 columns."
## [1] "Its columns are: tconst, ordering, nconst, category, job, characters"
##
## [1] "In the ratings dataset, there are 3 columns."
## [1] "Its columns are: tconst, averageRating, numVotes"
```

```
## [1] "There are 1568336 rows in the basics dataset."
## [1] "There are 1568333 rows in the crew dataset."
## [1] "There are 21693507 rows in the principals dataset."
## [1] "There are 1568336 rows in the ratings dataset."
```

As a quick reference, here are the datasets we will be looking at:

Dataset	Observations	Variables
`basics`	1,568,336	9
`crew`	1,568,333	3
`principals`	21,693,507	6
`ratings`	1,568,336	3

[C] Specific Breakdowns.

[C1] The basics dataset

The **basics** dataset:

- **tconst**: The unique ID for a specific film.
- **titleType**: The kind of film — it can be a short, movie, tvShort, and more.
- **primaryTitle**: The final title of the film.
- **originalTitle**: The title of the film (before it was changed).
- **isAdult**: The indicator variable — 1 if it's an adult film, 0 otherwise.
- **startYear**: The film's release year.
- **endYear**: The film's ending year, IF the film was a TV show (NA otherwise).
- **runtimeMinutes**: The film's runtime (in minutes).
- **genres**: The film's list of genres.

Here are the takeaways from our analysis of `basis`:

- **tconst**: There are no missing values. There are exactly 1,568,336 ID's (i.e. individual films), one for every row.
- **titleType**: There are no missing values. There are 10 unique kinds of films in this dataset, ranging from shorts to video games.
- **primaryTitle**: There are no missing values. There are 1,158,575 primaryTitles in the dataset ($<$ 1,568,336). This should be looked into.
- **originalTitle**: There are no missing values. There are 1,176,721 originalTitles in the dataset ($<$ 1,568,336 and $>$ 1,158,575). This should be looked into.
- **isAdult**: There are no missing values. Approximately 1.56% of all the films in the dataset are adult films.
- **startYear**: There are missing values! Ignoring missing values, the earliest release date was 1874, while the latest release date was 2025.

- **endYear**: There are missing values! Ignoring missing values, the earliest end date was 1932, while the latest end date was 2030.
- **runtimeMinutes**: There are missing values! Ignoring missing values, the shortest film is 0 minutes long, while the longest film is 3,692,080 minutes (roughly 61,534 hours) long. The average length of films was 58.39153 minutes.
- **genres**: There are missing values!

[C2] The crew dataset

The **crew** dataset:

- **tconst**: The unique ID for a specific film.
- **directors**: The film's list of directors.
- **writers**: The film's list of writers.

Here are the takeaways from our analysis of `crew`:

- **tconst**: There are no missing values. There are exactly 1,568,333 ID's (i.e. individual films), one for every row. Note that this value is minorly less than the number of values in `basics` and `ratings`.
- **directors**: There are some missing values. This means not every movie in our data will have a specified director.
- **writers**: There are some missing values. This means not every movie in our data will have a specified writer.

[C3] The principals dataset

The **principals** dataset:

- **tconst**: The unique ID for a specific film.
- **ordering**: The order in credits.
- **nconst**: The unique ID for a specific person such as an actor, director, and more.
- **category**: The role of a specific person such as an actor, director, and more.
- **job**: The job title such as a producer, editor, and more.
- **characters**: The character that a specific person has played as.

Here are the takeaways from our analysis of `principals`:

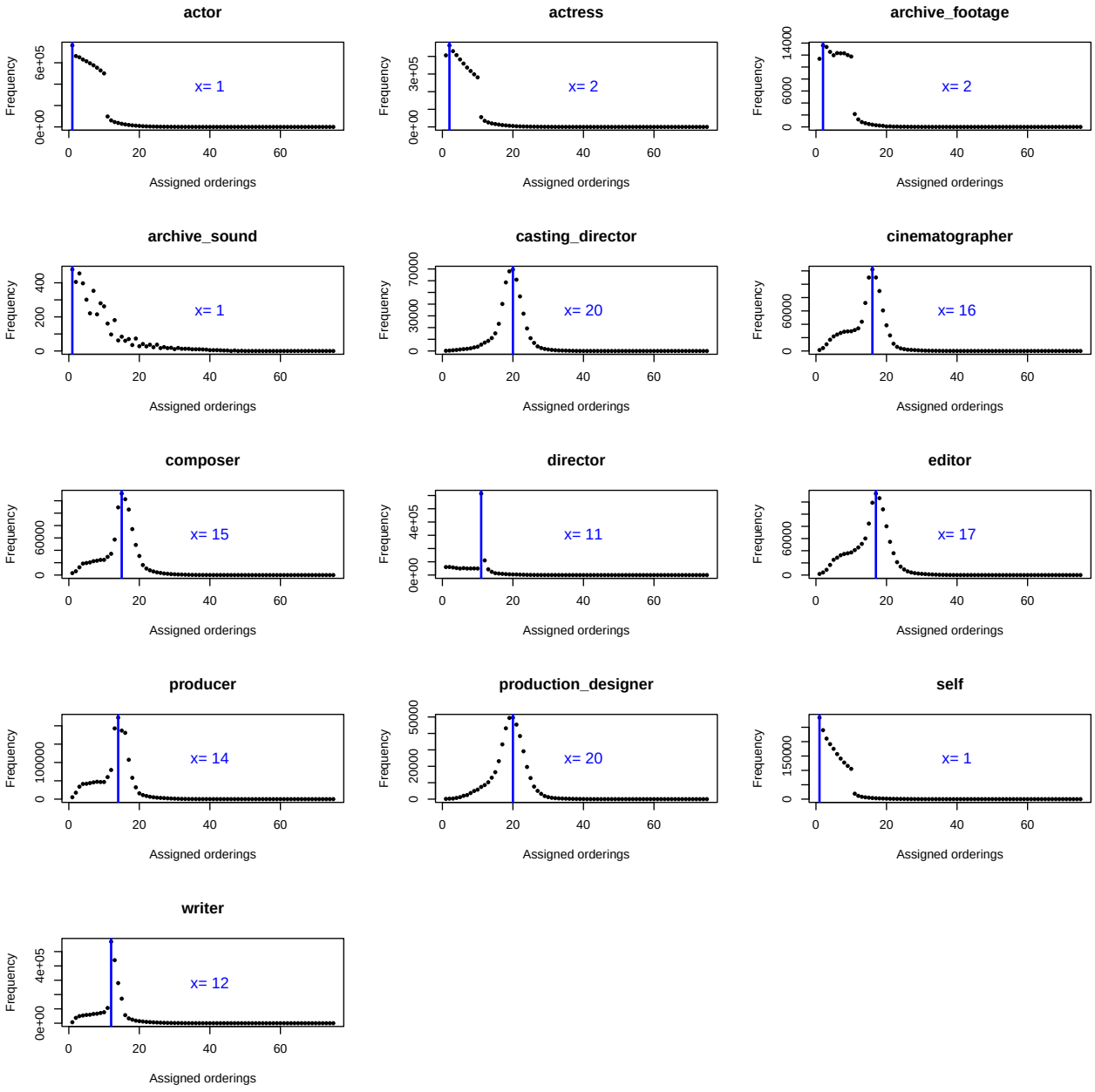
- **tconst**: There are no missing values. However, the number of unique film ID's does not equal the number of rows — this is because upon close inspection, we can see that there are duplicated ID's across several rows. This is different from the other three datasets, where there is one unique film ID for each row.
- **ordering**: There are no missing values. The ordering from most to least important ranges from 1 to 75.
- **nconst**: There are no missing values.
- **category**: There are no missing values. There are 13 unique categories in the dataset, from director to casting director.
- **job**: There are missing values — this is because of the duplicate values from the first column. There are 30,167 unique values within this column.
- **characters**: There are missing values — this is because of the duplicate values from the first column. There are 2,507,211 unique values within this column.

A side-note

We know that there are 13 unique categories within the **category** column. We also know that the ordering ranges from 1 to 75, where 1 is most important and 75 is least important.

So as a SIDE-NOTE, let's make a DataFrame (and plots) that count how many times each job **category** appears in a specific **ordering**.

```
##                category ordering
## 1                 actor        1
## 2               actress        2
## 3       archive_footage        2
## 4         archive_sound        1
## 5      casting_director       20
## 6       cinematographer       16
## 7              composer       15
## 8              director       11
## 9                editor       17
## 10             producer       14
## 11 production_designer       20
## 12                 self        1
## 13               writer       12
```



As we can see, the code matches what we'd expect if we eyeball what the code is doing to `my.table`.

[C4] The ratings dataset

The **ratings** dataset:

- **tconst**: The unique id for a specific film. There are exactly 1,568,336 ID's (i.e. individual films), one for every row.
- **averageRating**: The average rating, which was given to reviewers on a 1-10 scale.
- **numVotes**: The number of reviewers.

Here are the takeaways from our analysis of 'ratings":

- **tconst**: There are no missing values. There are exactly 1,568,336 ID's (i.e. individual films), one for every row.
- **averageRating**: There are no missing values. The worst review ever given was 1/10, while the best were 10/10. As a fun fact, the average rating across movies from the 1890's all the way to present day is roughly 6.95.
- **numVotes**: There are no missing values. Of all 1,568,336 films, 37,066 films have received the lowest number of reviews (only 5 reviews). Of all 1,568,336 films, only one film (i.e. the film in the 84,878th row) has the largest number of reviews (3,042,484 reviews).

D. Combining the Datasets.

Let's combine the four datasets:

1. The basics dataset.
2. The crew dataset.
3. The principals dataset.
4. The ratings dataset.

```
# COMBINING DATASETS:
# [1] Combining basics, crew, and ratings into one data table.
full_dt <- basics[crew, on = "tconst"]
full_dt <- full_dt[ratings, on = "tconst"]
# View(full_dt) # Uncomment to view data table

# [2] Checking the dimensions (rows, col).
dim(basics)      # This is correct - it should be (1,568,336 rows, 9 columns).
```

```
## [1] 1568336       9
```

```
dim(crew)        # This is correct - it should be (1,568,333 rows, 3 columns).
```

```
## [1] 1568333       3
```

```
dim(ratings)     # This is correct - it should be (1,568,336 rows, 3 columns).
```

```
## [1] 1568336       3
```

```
dim(full_dt)  # This is interesting - we expected (1,568,333 rows, 13 columns).
```

```
## [1] 1568336      13
```

```
# This is because crew is a limiting dataset as it has less rows than others.
```

As we can see:

1. We are able to merge `basics` and `ratings` based on the first column. This is because all of their values within the first column match.

2. However, our final dataset contains 1,568,336 rows. It has the right number of columns, but it seems that we are **not** being bottlenecked by the number of rows in crews. This may be a problem later when we try to do analysis on the crew data but some are missing from `full_dt`.

Let's run diagnostics to see why this is happening.

```
# DIAGNOSTICS: [PART 1]
# [1] Check matching of basics and ratings
# Check: Are all tconst values in ratings represented in basics?
all(ratings$tconst %in% basics$tconst) # TRUE. That is good.
# Check: Do all tconst values in basics have ratings data?
all(basics$tconst %in% ratings$tconst) # TRUE. That is good.

# [2] Check matching of basics and crew
# Check: Are all tconst values in crew represented in basics?
all(crew$tconst %in% basics$tconst) # TRUE. That is good.
# Check: Do all tconst values in basics have crew data?
all(basics$tconst %in% crew$tconst) # FALSE. That may be an issue.

# [3] Check matching of basics and crew
# Check: Are all tconst values in principals represented in basics?
all(principals$tconst %in% basics$tconst) # TRUE. That is good.
# Check: Do all tconst values in basics have principals data?
all(basics$tconst %in% principals$tconst) # FALSE. That may be an issue.
```

Here, we found out that both `crew` and `principals` lack `tconst` values compared to `basics`. That means we may have to trim all datasets to match that of `crew` when doing analysis that requires all datasets.

However, we have checked that `basics` and `ratings` match in its `tconst` values. This is due to the fact that we initially subsetted all datasets based on `ratings`. `basics` had `tconst` values for every film and was simply subsetted by `ratings`.

Since we have found a mismatch between `crew`, `principals` and the two other datasets, we will subset those datasets once again to provide consistency throughout our data.

```
# DIAGNOSTICS: [PART 2]
# [1] Finding bottlenecking dataset
length(unique(basics$tconst))     # 1568336. This is expected.
```

```
## [1] 1568336
```

```
length(unique(crew$tconst))       # 1568333. This could be a bottleneck.
```

```
## [1] 1568333
```

```
length(unique(principals$tconst)) # 1542357. This seems like the real bottleneck.
```

```
## [1] 1542357
```

```
length(unique(ratings$tconst))    # 1568336. This is expected.
```

```
## [1] 1568336
```

```
# [2] Using principals bottleneck on other datasets
principals_tconsts <- unique(principals$tconst)
basics <- basics[tconst %in% principals_tconsts]
crew <- crew[tconst %in% principals_tconsts]
ratings <- ratings[tconst %in% principals_tconsts]
```

```
# [3] Check matching
length(unique(basics$tconst))     # Output: 1542357
```

```
## [1] 1542357
```

```
length(unique(crew$tconst))       # Output: 1542357
```

```
## [1] 1542357
```

```
length(unique(principals$tconst)) # Output: 1542357
```

```
## [1] 1542357
```

```
length(unique(ratings$tconst))    # Output: 1542357
```

```
## [1] 1542357
```

Now, all the datasets have the same set of unique `tconst` values.

However, by the nature of the `principals` dataset, we can't store its multiple rows for each `tconst` value into one row in our final data table.

```
# DIAGNOSTICS: [PART 3]
# [1] Check whether the datasets can be merged
length(basics$tconst)     # 1542357. This is expected.
```

```
## [1] 1542357
```

```
length(crew$tconst)       # 1542357. This is expected.
```

```
## [1] 1542357
```

```
length(principals$tconst) # 21693507. This needs to be fixed.
```

```
## [1] 21693507
```

```
length(ratings$tconst)    # 1542357. This is expected.
```

```
## [1] 1542357
```

```
# [2] Check one row of principals
principals_first_tconst <- principals[1, tconst]
principals[principals$tconst == principals_first_tconst]
```

```
##       tconst ordering    nconst        category                   job
##       <char>    <int>    <char>          <char>                <char>
## 1: tt0000001        1 nm1588970            self                  <NA>
## 2: tt0000001        2 nm0005690        director                  <NA>
## 3: tt0000001        3 nm0005690        producer              producer
## 4: tt0000001        4 nm0374658 cinematographer director of photography
##    characters
##        <char>
## 1:   ["Self"]
## 2:       <NA>
## 3:       <NA>
## 4:       <NA>
```

```
# For this film, it seems that there are four rows associated with it.

# [3] Modify principals to include data for each tconst in one row
# Use ordering as order of each individual. Index value represents ordering.
```

```r
principals <- principals[, .(
  nconst = list(nconst),
  category = list(category),
  job = list(job),
  characters = list(characters)
  ), by = tconst]
principals$characters[[1]][1]
```

```
## [1] "[\"Self\"]"
```

```r
# [4] We see that the characters column has an issue with "[\" and "\]".
# We get rid of those symbols.
principals[, characters := lapply(characters, function(x) {
  gsub('^\\["|"]$', '', x)
})]
```

Now, we can safely merge all datasets.

```r
# DIAGNOSTICS: [PART 4]
full_dt <- basics[crew, on = "tconst"]
full_dt <- full_dt[principals, on = "tconst"]
full_dt <- full_dt[ratings, on = "tconst"]
nrow(full_dt) # Output: 1,542,357. This is good.
```

```
## [1] 1542357
```

```r
# saveRDS(full_dt, "full_dt.rds") # Run this line to save this cleaned df
```

TODO: Delete this later

---

[3] DATA ANALYSIS:

Now that we have gone through the datasets and obtained a final data table `full_dt` to work with, we'll tackle **five questions** starting with the most intuitive one first.

QUESTION 1:

Our first question is: **does the number of votes affect the ratings?**

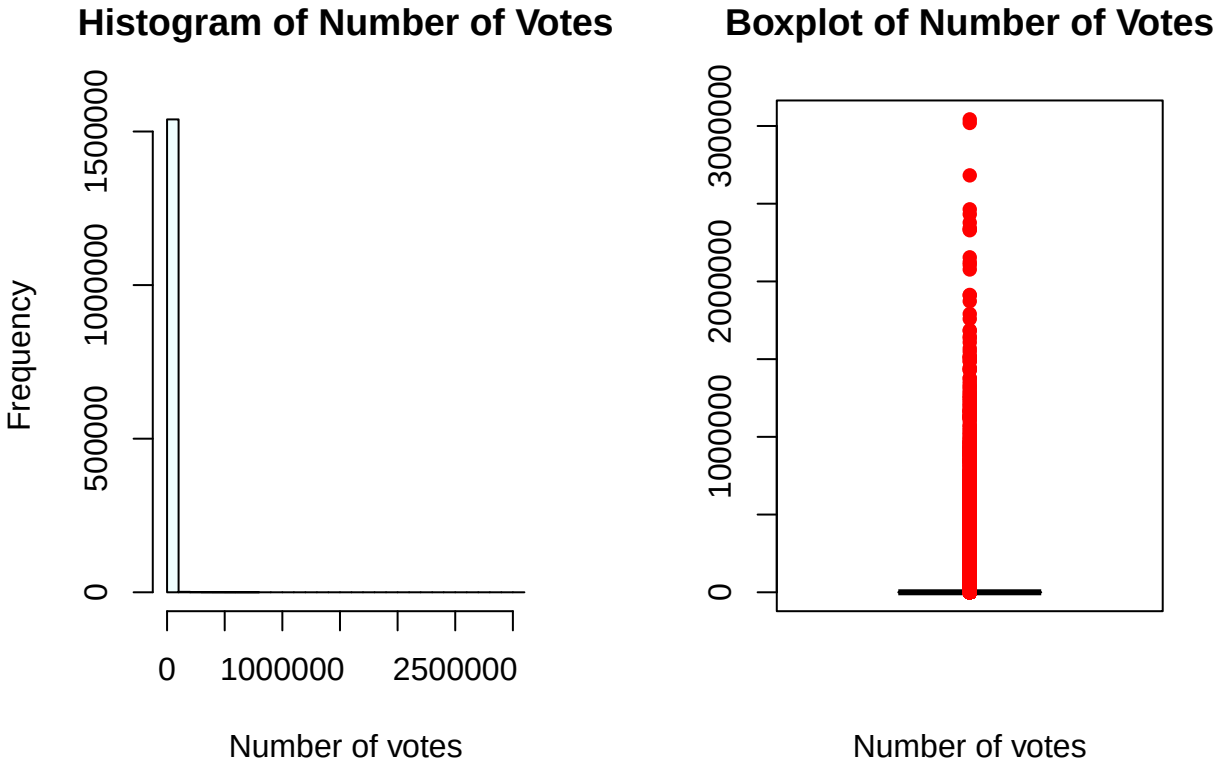
In the real world, many people want to know how good a movie is before watching it. To do so, they go online to check the movie's reviews. But here's the thing everyone suspects:

- Films that have very few votes (i.e. unpopular films) may not have the most reliable ratings, as their reviews don't contain much information.
- Films that have lots of votes (i.e. popular films) may suggest that the movie has larger positive reception or audience appeal.

Specifically, we'll investigate whether or not films that have more votes tend to have higher or lower ratings (i.e. whether popularity is correlated with perceived quality).

This will provide a good opening introduction, and hopefully provide some intuition, to our analysis. Let us first look at the distribution of `numVotes`.

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       5      12      26    1040     102 3042484
```



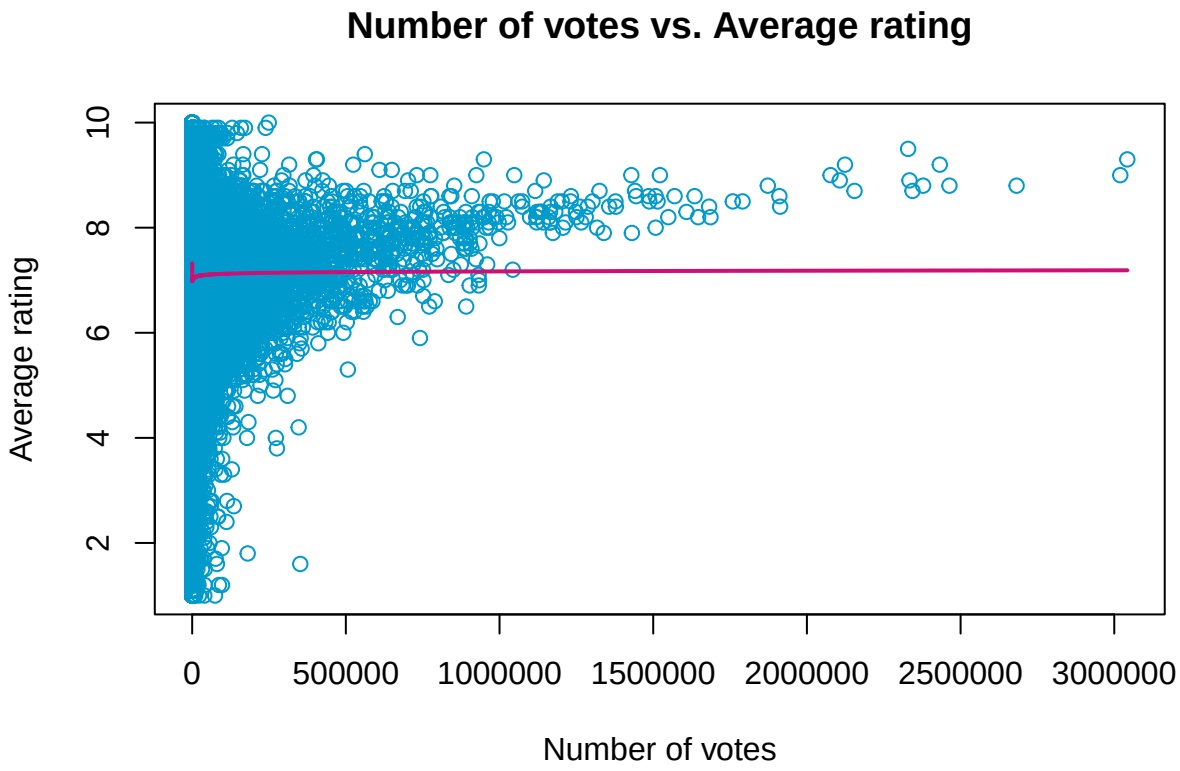
Outliers are colored red, and we see that the box plot deems most of the visible points as outliers. The information so far suggests a heavy right skew.

As we can see, the number of votes in our dataset is extremely right-skewed, where:

- Most movies have very few votes.
- Few movies have lots of votes.

- So far, the data doesn't seem to follow a strictly linear pattern. The data seems to follow some sort of nonlinear pattern, however. Let us fit a local regression model — a non-parametric statistical method — to visualize the trajectory when we logarithmize `numVotes` (x).

```
# [3] Plot numVotes (x) against averageRating (y)
par(mfrow = c(1, 1))
x <- full_dt$numVotes
y <- full_dt$averageRating
plot(x = x,
     y = y,
     xlab = "Number of votes",
     ylab = "Average rating",
     main = "Number of votes vs. Average rating",
     col = "deepskyblue3")

# [4] Fit local regression model
# Fitting a LOWESS model with default arguments
# (Computationally more efficient than LOESS)
my.lowess <- lowess(log10(x), y)
# LOWESS curve
lines(10^my.lowess$x, my.lowess$y, col = "deeppink3", lwd = 2)
```




```
# [5] Check the strength of the linear relationship between these variables.
cor(log10(x), y)
```

```
## [1] -0.05594476
```

The local regression model doesn't seem to give us a helpful fitted line on the trend of `averageRating` for each film. Let us try plotting it again but also bin the logarithm of the x-axis and sample a number of values rather than relying on the multitude of values heavily focused on the left side of the graph.

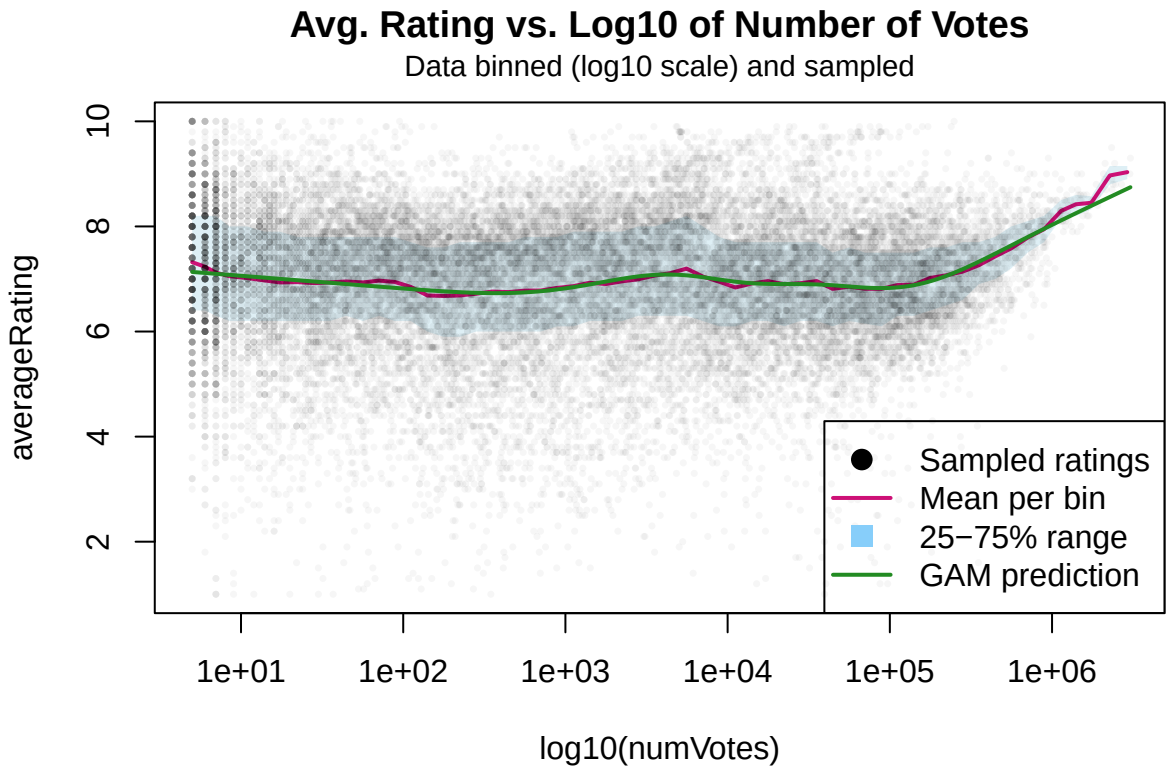
```r
# [6] Bin numVotes and sample a number of averageRatings for each bin
q1_dt <- full_dt[, .(numVotes, averageRating)]

# Logarithmize
q1_dt$x_log <- log10(q1_dt$numVotes)
# Bin log10(numVotes)
bin_width <- 0.1
q1_dt[, bin := floor(x_log / bin_width) * bin_width]
# Sample 500 (max) per bin
q1_dt_sampled <- q1_dt[, {
  idx <- sample(.N, min(.N, 500))
  .SD[idx]
}, by = bin]
# IQR ribbon
q1_stats <- q1_dt[, .(
  p25 = quantile(averageRating, 0.25),
  p75 = quantile(averageRating, 0.75),
  mean_rating = mean(averageRating),
  x_bin = 10^mean(x_log)
), by = bin][order(x_bin)]
```

Let us fit a Generalized Additive Model (GAM) to capture the non-linearity between `numVotes` and `averageRating` — in our case, modeling how averageRating changes with the logarithmic number of votes `log10(numVotes)`.

We hope that this approach gives us a more stable and interpretable fit than LOWESS when dealing with our data.

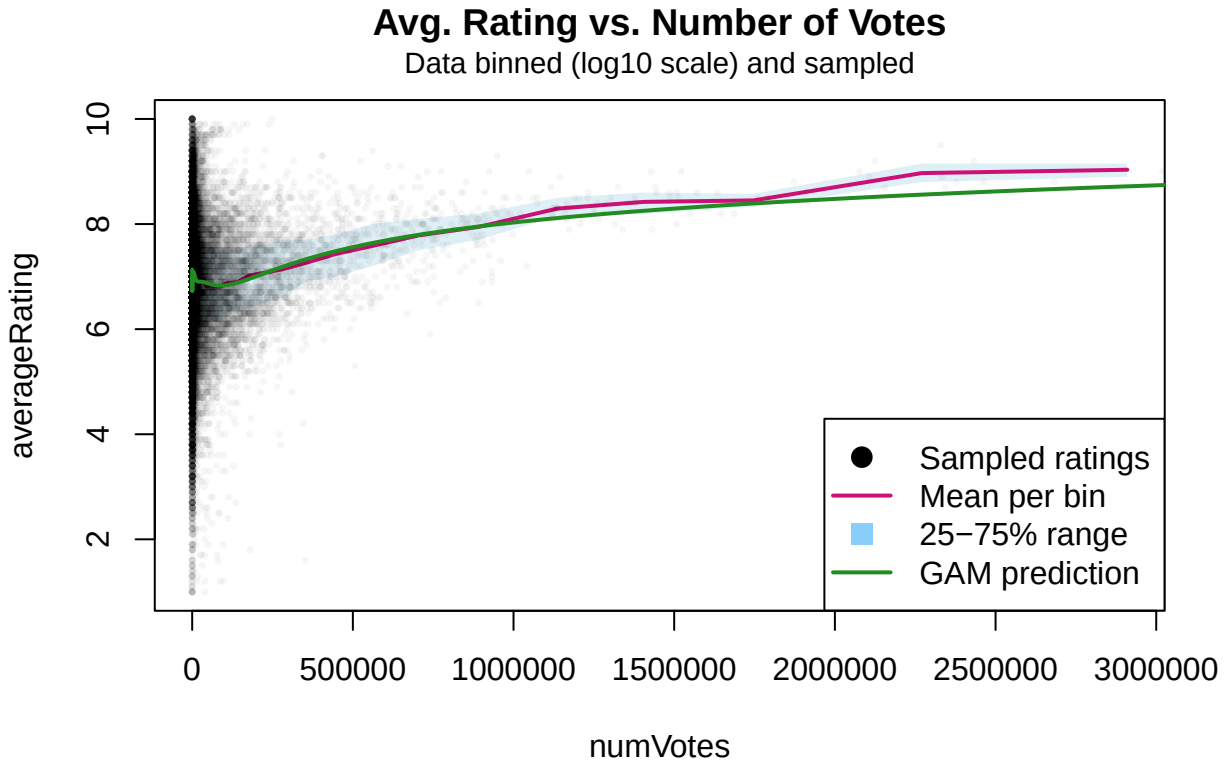
```r
# [7] Fit GAM
gam1 <- gam(averageRating ~ s(x_log), data = q1_dt)
# Predict
log_x_pred <- seq(log10(min(q1_dt$numVotes)), log10(max(q1_dt$numVotes)),
                  length.out = 500)
x_pred <- 10^log_x_pred
y_pred <- predict(gam1, newdata = data.frame(x_log = log_x_pred))
```

Let us plot this first with the x-axis as the logarithmic number of votes.



This plot shows strong evidence that as the number of votes increases, a film's `averageRating` fluctuates until it has a steep increase for its rating if the film gets an extremely high number of votes.

Let us now plot this with the x-axis as the original number of votes and see if the same trend appears.



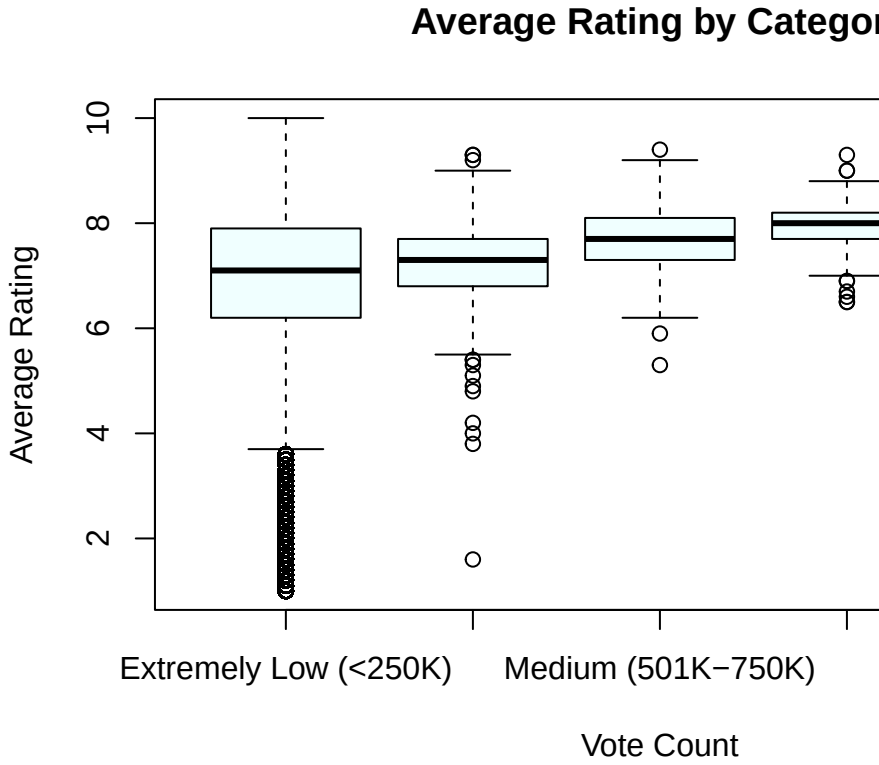
Again, we see a similar trend where a majority of the films with less than 500,000 votes fluctuate heavily

in `averageRating`, but as the number of votes increases steeply, the average rating values also gradually increase. Note that this plot follows the same GAM as before on the logarithmic `numVotes` variable.

Let us look at the summary of our GAM:

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## averageRating ~ s(x_log)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  6.95017    0.00111    6260   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##           edf Ref.df    F p-value
## s(x_log) 8.59  8.923 1289  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.0074   Deviance explained = 0.74%
## GCV = 1.9009  Scale est. = 1.9009    n = 1542357
```

The GAM holds a p-value less than 2e-16. Seeing our plots, we see a stable horizontal line for its prediction line until higher values of `numVotes`. Thus, we have good evidence from the extremely low p-value that this trend is true. Additionally, it seems that the average rating for a film with not too extremely large number of votes is about 6.95017.



Lastly, here is a boxplot in support of our analysis.

The apperance of this box plot follows the same idea that higher values of vote counts indicate higher average ratings, only if extremely high.

Here's what we can take away from this analysis:

1. The plotted graph of average ratings against the number of votes exhibits some nonlinear pattern.
    - Specifically, films that have less votes tend to have a wider spread of average ratings, roughly between 2 to 8.
    - Specifically, films that have more votes tend to have a smaller spread of average ratings.
2. As the number of votes increases, the average ratings stays stable at around 7-8 before increasing as a film gets a much larger number of votes.
3. Movies with extremely high votes (at least a million) have narrow/tight distributions with a much higher average rating than the others. This suggests that the number of votes a film receives can act as an indicator of the film's resulting ratings, only if the number of votes (`numVotes`) is extremely high.

QUESTION 2:

The next question we want to answer is: **does having a writer affect the ratings?**

Let's take a step back to understand why we're even asking this question. Upon a quick glance at the dataset, we can see that its column **writers** is filled with a mixture of filled and missing values. In other words, not every film has a specified writer.

As a result, it naturally follows to ask how this would affect the film's ratings. Here's our approach to investigating this:

```
q2_dt <- full_dt

# [1] Define a binary column called haswriter.
#  Binary column: 1 if there's a writer, 0 otherwise.
q2_dt$haswriter <- as.numeric(!is.na(q2_dt$writers))

# [2] Obtain a summary statistic and mean for the column haswriter.
summary(q2_dt$haswriter)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  1.0000  1.0000  0.7758  1.0000  1.0000
```

```
mean(q2_dt$haswriter)
```

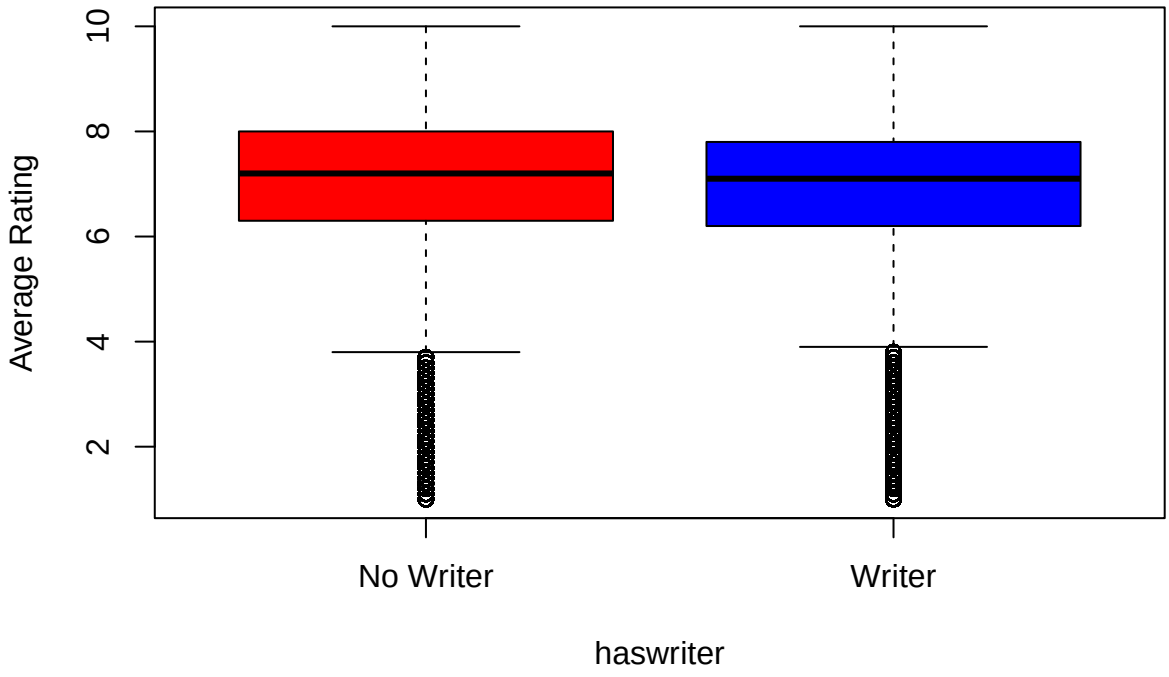
```
## [1] 0.7757672
```

Here, the min and max are obviously 0 and 1. The mean of the column `haswriter` is 0.7758. This means that roughly 77.6% of our films in the dataset have a specified writer.

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00    6.20    7.10    6.95    7.90   10.00
```

```
## [1] 6.950173
```

The worst films have an average rating of 1/10. The best films have an average rating of 10/10. The **average** average rating, as mentioned before, is 6.95/10.




```
##
##  Welch Two Sample t-test
##
## data:  averageRating by haswriter
## t = 34.832, df = 541488, p-value < 2.2e-16
## alternative hypothesis: true difference in means between group 0 and group 1 is not equal to 0
## 95 percent confidence interval:
##  0.09002716 0.10076283
## sample estimates:
## mean in group 0 mean in group 1
##        7.024177        6.928782
```

Here's our takeaway from our analysis:

1. In our comparison of films that have specified and unspecified writers, we can see that their boxplots are identical. Their distribution's span are relatively the same with identical means around 7.0.
2. There visually doesn't seem to be much of a difference between their means.
3. Just to be safe, we conducted a two-sample t-test. This is where things got interesting:
    - The p-value is less than 2.2e-16, which is less than an assumed significance level of alpha = 0.05.
    - Technically, this means we reject the null and conclude there is enough evidence to suggest there is a statistically significant difference between these two means.
    - But this obviously does not align with our findings from the dataset.
    - In addition, the t-test's results shows us that the mean for "No Writer" is 7.024177 while the mean for "Writer" is 6.928782. Those means are incredibly close and identical in the **practical** sense. We simply do not care whether a film has a rating that is 0.095395/10 higher than another.
4. To get around this caveat, we asked ourselves two questions:
    - How could a film not have writers? They do; it's just that some films in the dataset have no specified writers because of data incompleteness, especially for really old or lesser-known films where the writers may have not been well documented.
    - Do all audiences necessarily care about who the writers are for an upcoming film? Unless it's a well-known writer who is mentioned in the trailers, probably not! They likely pay more attention

to the director(s) and actors.
- This is an example where the context of the situation need to be considered, not just a statistical test's dry facts.

All in all, having a writer or not in the given dataset does not seem to affect the ratings of a film.

QUESTION 3:

Our third question is: **"Does having certain actors have an affect on ratings? Or is this just an effect of hiring good actors for good movies?"**

To answer this question, we need to realize that we only care about the following columns within the dataset `full_dt`:

- category: specifically rows that only contain actor information ("actor", "actress"). Unfortunately, not every movie has a specified actor/actress.
- tconst: The movie that has actor information.
- averageRating: gives us the average rating of movies with actor information.

Let's first filter out our dataset `full_dt`. We only want the rows where the **category** column contains actor information.

```
# [1] Create a new dataset called actors.dataset. It's a filtered version of full_dt, only containing t
actors.dataset <- full_dt[grepl("actor|actress", principals$category), ]
dim(actors.dataset)  # Contains 1,226,885 rows and 17 columns.
```

```
## [1] 1226885      17
```

```
# [2] Clean actors.dataset to ONLY have entries that say "actor" and "actresses."
library(data.table)   # Load data.table library - used to handle large datasets.
setDT(actors.dataset) # Convert to data.table.

actors.dataset[, c("nconst", "category", "job", "characters"):={    # Filter each column using mapply()

  actor.index <- lapply(category, function(cats) cats %in% c("actor", "actress"))
  list(
    mapply(function(x, index) x[index], nconst, actor.index, SIMPLIFY = FALSE),
    mapply(function(x, index) x[index], category, actor.index, SIMPLIFY = FALSE),
    mapply(function(x, index) x[index], job, actor.index, SIMPLIFY = FALSE),
    mapply(function(x, index) x[index], characters, actor.index, SIMPLIFY = FALSE)
  )
}]
```

Here is what we should take away from this dataset:

1. Our filtered dataset `actors.dataset` contains 1,226,885 observations and 17 variables.
2. In other words, out of all given 1,542,357 films, only 1,226,885 of them contain actor information while the rest didn't.
3. A film can have **more than one** actor specified within the **category** column.
4. The dataset does **not always** specify what role the actor played — we can't tell if this is an actor with a big role or not.

That being said, let's conduct our analysis for question 3. Our analysis is broken down into several observations, as listed below.

Observation 1. Extreme Ratings

The following is our analysis for question 3:

```
# [1] Setting up our data.
library(data.table)                                                    # Load data.table libra
setDT(actors.dataset)                                                  # Convert to data.table
long <- actors.dataset[, .(nconst = unlist(nconst),                    # Use unlist() function
                           averageRating = rep(averageRating, lengths(nconst)),
                           numVotes = rep(numVotes, lengths(nconst)))]
```

```
# [2] Compute each actor's mean rating and number of films.
actor.results <- long[, .(                                                    # A DataFrame w/4 colum
  weighted.average.rating = sum(averageRating * numVotes) / sum(numVotes),
  total.votes = sum(numVotes),
  number.of.films = .N),
  by = nconst][order(-weighted.average.rating)]                               # Using order(-average.

head(actor.results)                                                           # Displays head of Data
```

```
##        nconst weighted.average.rating total.votes number.of.films
##        <char>                   <num>       <int>           <int>
## 1: nm11000153                      10          23               1
## 2:  nm2436198                      10          11               1
## 3:  nm1449677                      10          11               1
## 4:  nm1451968                      10          11               1
## 5:  nm1451969                      10          11               1
## 6:  nm1450118                      10          11               1
```

Here, we defined a DataFrame that displays a list of actors who starred in films ordered from highest to least average weighted ratings.

As we can see, actors who starred in films with the top ratings (like 10/10) were actors who, **at least according to our dataset**, starred in only a **single film**.

- This clearly doesn't tell us very much.
- Why? Because an actor who only starred in only one film that got a 10/10 review will obviously have a weighted average film rating of 10/10. That clearly doesn't say much in our analysis.
- If an actor who say starred in 1000 films still ended up getting a top weighted rating of 10/10, then that would mean significantly more.

Caveat 2. Cautionaries

To get our feet even further wet, let's check out some specific actors.

```
# [1] Bobby Connelly (1909-1922); ID: "nm0175050"
actor.results[which(actor.results$nconst == "nm0175050")]
```

```
##       nconst weighted.average.rating total.votes number.of.films
##       <char>                   <num>       <int>           <int>
## 1: nm0175050                6.464286         910              13
```

```
# [2] Jerold T. Hevener (1873-1947); ID: "nm0381936"
actor.results[which(actor.results$nconst == "nm0381936")]
```

```
##       nconst weighted.average.rating total.votes number.of.films
##       <char>                   <num>       <int>           <int>
## 1: nm0381936                7.045506         178              10
```

Here, we arbitrarily picked out two actors — Bobby Connelly (1909-1922) and Jerold T. Hevener (1873-1947).

Bobby Connelly was a famous child-actor who unfortunately passed away young. *__According to the dataset__, he starred in 13 films, which seems reasonable given his short career.

However, there seems to be some mistake in the dataset. **According to the dataset**, Jerold T. Hevener was an actor who starred in 10 films. In actuality, he starred in around 36 films.

This quick investigation tells us that we should approach these numbers with caution — the dataset isn't guaranteed to give us back the accurate number of films a specific actor has starred in.

Observation 3. Converging Ratings

In the previous section, we investigated two arbitrary actors. But now, let's investigate the actor who has starred in the most number of films.

```
maximum.films <- max(actor.results$number.of.films)
index <- which(actor.results$number.of.films == maximum.films)
actor.results[index]
```

```
##       nconst weighted.average.rating total.votes number.of.films
##       <char>                   <num>       <int>           <int>
## 1: nm0048389                7.964585    12753444            8230
```

Here, the actor who starred in the most number number of films is **Dee Bradley Baker** (ID: "nm0048389") an extremely well-known voice actor who voices in many films and TV cartoons ever since the early 2000's. According to this dataset, all the 8,230 films he voiced in have an average weighted rating of 7.964585 out of 10 based on 12,753,444 votes. That is a relatively high rating with a large sample voting size.

But here's the catch: does this one actor's situation imply that if you're an actor who starred in **many** films that have **large voting size**, then your films will have an overall higher weighted average rating?

To investigate this, let's do a **Google Search** and **deliberately** investigate other extremely well-known actors who also were in many films.

```
# Famous Actors:
actor.results[which(actor.results$nconst == "nm0000138")]     # [1] Leonardo DiCaprio; ID: "nm0000138"
```

```
##       nconst weighted.average.rating total.votes number.of.films
##       <char>                   <num>       <int>           <int>
## 1: nm0000138                7.998845    18761495              82
```

```
actor.results[which(actor.results$nconst == "nm0000151")]     # [2] Morgan Freeman; ID: "nm0000151"
```

```
##       nconst weighted.average.rating total.votes number.of.films
##       <char>                   <num>       <int>           <int>
## 1: nm0000151                7.914311    19659977             240
```

```
actor.results[which(actor.results$nconst == "nm0000158")]     # [3] Tom Hanks; ID: "nm0000158"
```

```
##       nconst weighted.average.rating total.votes number.of.films
##       <char>                   <num>       <int>           <int>
## 1: nm0000158                  7.7756    19522793             210
```

```
actor.results[which(actor.results$nconst == "nm0000375")]     # [4] Robert Downey Jr; ID: "nm0000375"
```

```
##       nconst weighted.average.rating total.votes number.of.films
##       <char>                   <num>       <int>           <int>
## 1: nm0000375                7.713361    22081110             186
```

```
actor.results[which(actor.results$nconst == "nm0424060")]     # [5] Scarlett Johansson; ID: "nm0424060"
```

```
##       nconst weighted.average.rating total.votes number.of.films
##       <char>                   <num>       <int>           <int>
## 1: nm0424060                7.619085    25317458              94
```

```
# Average of the Entire Dataset:
mean(actors.dataset$averageRating)
```

```
## [1] 6.928019
```

As we can see, these chosen famous actors have weighted average ratings between 7.6 and 8, based off of **millions** of votes and **hundreds** of films.

Their weighted average ratings are **slightly higher** than the dataset's global average, which is around 6.928019 out of 10. This suggests that these high-profile actors do tend to appear in **well-received, widely watched** films.

Now here's the thing — as humans, our intuition generally expects high-profile actors to have high ratings such as 9/10 or 8/10. But as we can see, these famous actors' ratings are not necessarily the very highest in the dataset.

In fact, we've already seen actors who had ratings of 10/10. These are specifically actors who were featured in **a smaller number of films** and received **fewer votes**.

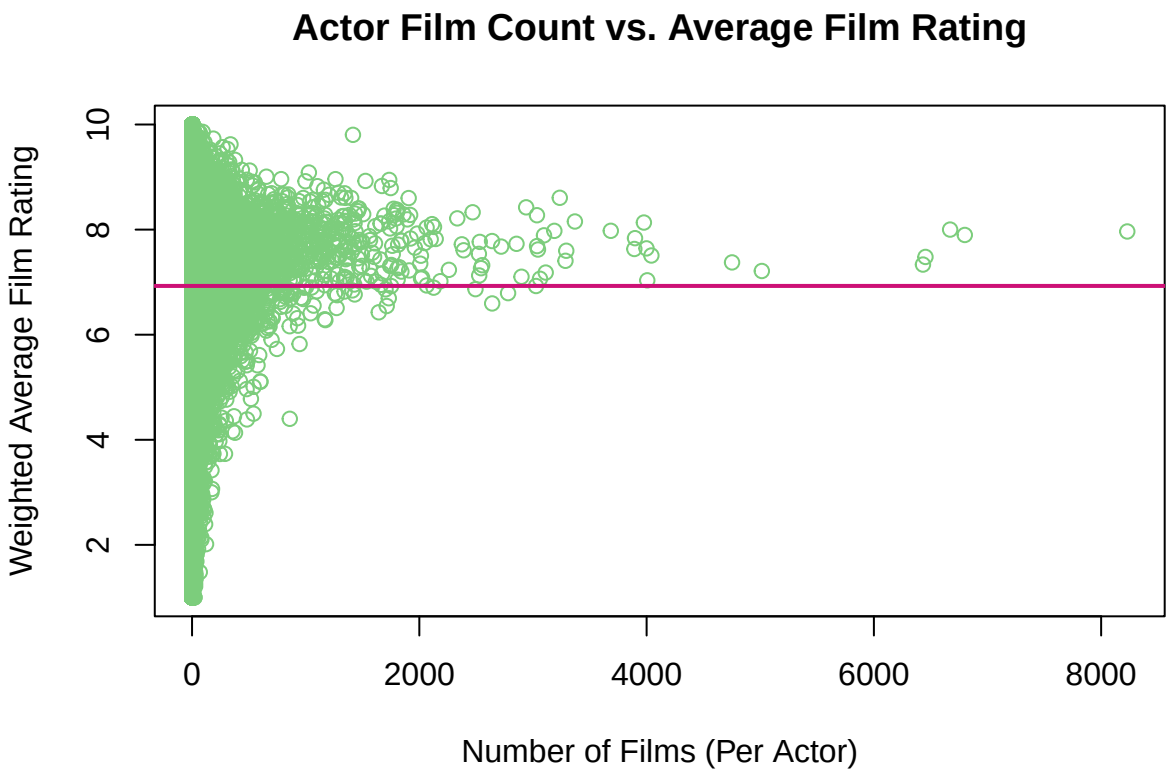
This is a classic example of the **Law of Large Numbers**: as the number of films increases, average ratings tend to converge towards the mean.

Colloquially speaking:

- If you're an actor that was in a just **one** film reviewed by say only **10 people**, and consequently got a perfect 10/10 rating, then your weighted average rating will be a 10/10.
- If you're an actor that was in a just **one** film reviewed by say only **10 people**, and consequently got a bad 1/10 rating, then your weighted average rating will be a 1/10.
- Conversely, if you're an actor that was featured in **many films** that were reviewed by **millions of people**, then your resulting weighted average rating will of course likely not be a perfect 10/10 — it's going to be **MUCH LOWER** and **CLOSER** to the entire dataset's mean rating.

Here is a supporting plot that visualizes the Law of Large Numbers:

```
plot(actor.results$number.of.films,
     actor.results$weighted.average.rating,
     main = "Actor Film Count vs. Average Film Rating",
     xlab = "Number of Films (Per Actor)",
     ylab = "Weighted Average Film Rating",
     col = "palegreen3")
abline(h = mean(actors.dataset$averageRating), col = "deeppink3", lwd = 2)
```



Here, we can see that:

- Actors who were featured in extremely small amounts of films tended to have extremely high weighted ratings.
- But as we said earlier, this could potentially be large actors who actually featured in more films than the dataset says, OR small actors whose few films have done well.
- Actors at the right end of the plot — large actors who acted in many films — had much lower ratings than what's shown on the left side of the plot. Their ratings also tend to be roughly around the average ratings of the entire dataset (roughly around 6).
- Notice how **similar** this plot looks to the plot from a previous question. This goes back to the **Law of Large Numbers** idea we addressed earlier.

Observation 4. Large Ratings and Large Voting Size

This part of the analysis does **not** prove that specific actors **cause** their films to have high ratings. Instead, this part of the analysis will hint at how frequently top movies have top actors.

Note: **Just** for this observation, we will (for now) focus on full-length **movies**, excluding TV show episodes.

```
# [1] Find the top 10 movies that have BOTH the highest rating AND highest number of votes.
full_movies <- actors.dataset[titleType == "movie"]                              # Filter to only actual
top_movies <- full_movies[order(-averageRating, -numVotes)]                      # Sorting full dataset
top_movies <- top_movies[numVotes > 50000]                                       # I CHOSE at least 50,0

top10_movies <- head(top_movies, 10)                                             # Grab the top 10 films
top10_movies[, .(tconst, nconst, primaryTitle, averageRating, numVotes)]
```

```
##        tconst                                                          nconst
##        <char>                                                          <list>
##  1: tt0111161 nm0000209,nm0000151,nm0348409,nm0006669,nm0000317,nm0004743,...
##  2: tt0068646 nm0000008,nm0000199,nm0001001,nm0000473,nm0144710,nm0000380,...
##  3: tt0468569 nm0000288,nm0005132,nm0001173,nm0000323,nm0350454,nm0000198,...
##  4: tt0167260 nm0000704,nm0001557,nm0005212,nm0089217,nm0032370,nm0040012,...
##  5: tt0108052 nm0000553,nm0000146,nm0001426,nm0328751,nm0755974,nm0001110,...
##  6: tt0071562 nm0000199,nm0000134,nm0000380,nm0000473,nm0001030,nm0001735,...
##  7: tt0050083 nm0000020,nm0002011,nm0000842,nm0275835,nm0550855,nm0001430,...
##  8: tt0110912 nm0000237,nm0000235,nm0000168,nm0000246,nm0000619,nm0001625,...
##  9: tt0120737 nm0000704,nm0005212,nm0089217,nm0000293,nm0397102,nm0032370,...
## 10: tt1375666 nm0000138,nm0330687,nm0680983,nm0913822,nm0362766,nm2438307,...
##                                        primaryTitle averageRating numVotes
##                                              <char>         <num>    <int>
##  1:                          The Shawshank Redemption           9.3  3042484
##  2:                                   The Godfather           9.2  2123901
##  3:                                 The Dark Knight           9.0  3019519
##  4:     The Lord of the Rings: The Return of the King           9.0  2076994
##  5:                                 Schindler's List           9.0  1521787
##  6:                            The Godfather Part II           9.0  1428736
##  7:                                    12 Angry Men           9.0   924651
##  8:                                    Pulp Fiction           8.9  2333671
##  9: The Lord of the Rings: The Fellowship of the Ring           8.9  2106913
## 10:                                       Inception           8.8  2682224
```

```
# View(top10_movies)                                                             # Uncomment to open the

# [2] Find the actors that were featured in those top 10 films.
list.of.actors <- list()
```

```r
for (i in 1:10) {
  list.of.actors[[i]] <- unlist(top10_movies$nconst[i])  # This flattens the inner list
}
names(list.of.actors) <- paste("Movie", 1:10)
# list.of.actors                                                    # Uncomment to see the

# [3] Define a new list to hold actor stats per movie.
actor.stats.by.movie <- list()                                      # Instantiating a new l
for (i in 1:10) {                                                   # Repeat for each movie
  actors.ids <- list.of.actors[[i]]                                 # Grab the actors withi
  stats <- actor.results[nconst %in% actors.ids]                    # Go back to the actor.
  actor.stats.by.movie[[i]] <- stats                                # Update the list.
}
names(actor.stats.by.movie) <- paste("Movie", 1:10)
# actor.stats.by.movie                                              # Uncomment to see the

# [4] Define a final summarizing table.
movie <- top10_movies$primaryTitle
number.of.actors <- sapply(actor.stats.by.movie, nrow)
avg.weighted.rating <- sapply(actor.stats.by.movie, function(df) mean(df$weighted.average.rating))
avg.number.of.films <- sapply(actor.stats.by.movie, function(df) mean(df$number.of.films))
top.actor.rating <- sapply(actor.stats.by.movie, function(df) max(df$weighted.average.rating))

summarizing.table <- data.frame(
  movie = movie,
  number.of.actors = number.of.actors,
  avg.weighted.rating = avg.weighted.rating,
  avg.number.of.films = avg.number.of.films,
  top.actor.rating = top.actor.rating
)

print(summarizing.table)
```

```
##                                                movie number.of.actors
## Movie 1                          The Shawshank Redemption               10
## Movie 2                                     The Godfather               10
## Movie 3                                   The Dark Knight               10
## Movie 4      The Lord of the Rings: The Return of the King               10
## Movie 5                                   Schindler's List               10
## Movie 6                                The Godfather Part II               10
## Movie 7                                     12 Angry Men               10
## Movie 8                                     Pulp Fiction               10
## Movie 9  The Lord of the Rings: The Fellowship of the Ring               10
## Movie 10                                       Inception               10
##          avg.weighted.rating avg.number.of.films top.actor.rating
## Movie 1             8.519878               338.1         9.127888
## Movie 2             8.629481                95.9         9.194541
## Movie 3             8.194301               107.2         8.895936
## Movie 4             8.478570               149.6         8.995000
## Movie 5             8.235334                83.6         8.999648
## Movie 6             8.387191                69.6         8.928679
## Movie 7             8.495751               208.0         8.997379
## Movie 8             8.054948               413.7         8.900000
## Movie 9             8.405614               152.7         8.946819
```

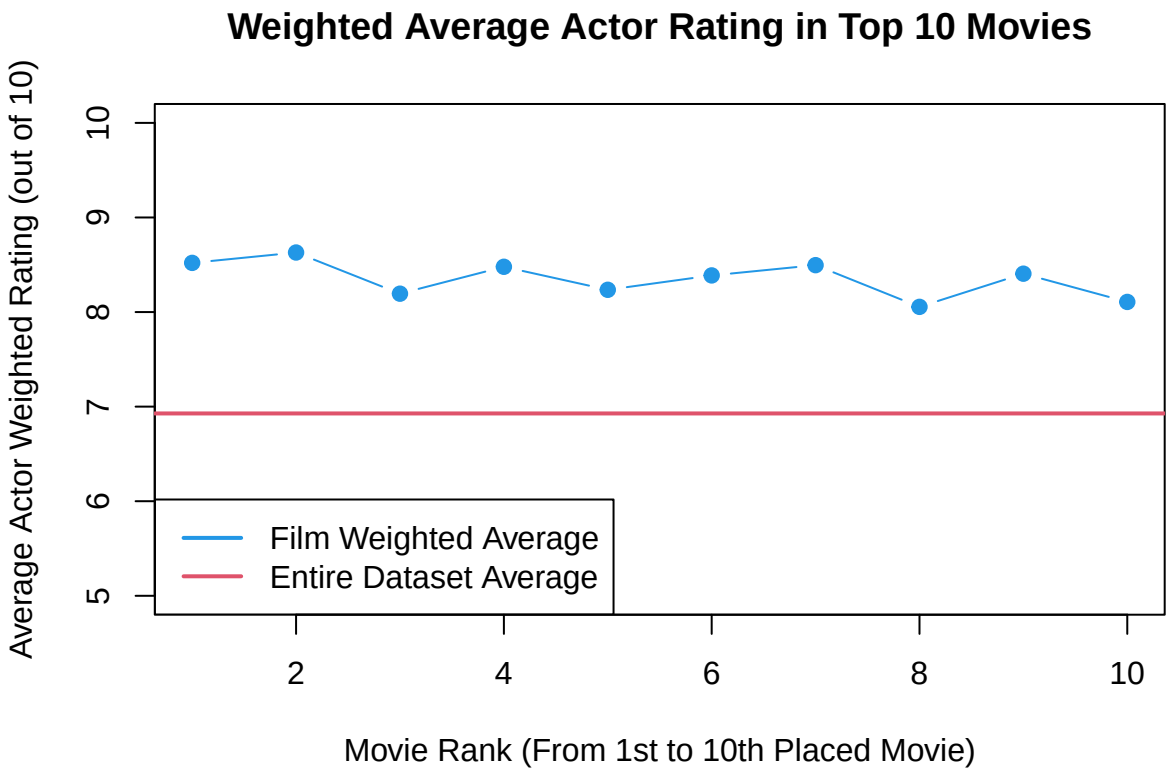
```
## Movie 10            8.106939               110.1         8.628751
```

This is a table that shows the average weighted rating and the average number of films for **all** the actors within **each** movie. It also shows the highest actor rating received among all the actors within **each** movie.

Why do we care about this table? From here, we can compare our table to to the entire dataset's mean, and visually make a plot that shows whether top movies tend to feature top-rated actors.

Here is the plot:

```
plot(summarizing.table$avg.weighted.rating,
     type = "b",
     pch = 19,
     col = 4,
     main = "Weighted Average Actor Rating in Top 10 Movies",
     xlab = "Movie Rank (From 1st to 10th Placed Movie)",
     ylab = "Average Actor Weighted Rating (out of 10)",
     ylim = c(5, 10))
abline(h = mean(actors.dataset$averageRating), col = 2, lwd = 2)
legend("bottomleft", legend = c("Film Weighted Average","Entire Dataset Average"), col = c(4, 2), lwd =
```



As we can see, the top 10 **highest-rated**, **popular** movies (excluding episodes) often feature actors with **above-average weighted ratings**. This is indicated by the blue plots consistently placed above the red line.

This plot suggests that great movies do attract talented or well-received actors, but it does NOT guarantee that those actors bring high ratings on their own.

Conclusion

In conclusion, our analysis shows that while well-known actors tend to star in **well-received** films with **strong** ratings, this does not necessarily mean that their presence **causes** those ratings to be high.

In other words:

- We do not find clear evidence that certain actors cause higher movie ratings.
- However, we do find **correlation**: top-rated movies tend to include actors who also have strong average film ratings.

Specifically, we found that:

- The **extremely high** average ratings, such as a perfect 10/10, mainly come from actors who appeared in only one or two highly-rated, niche films.
- The weighted average ratings for **major actors** — Leonardo DiCaprio, Morgan Freeman, etc. — fall between 7.6 and 8.0, which is solidly above average but not necessarily that extremely high.
- This distinction is a result of the **Law of Large Numbers**, where actors with many films tend to regress toward the dataset's overall average. On the other hand, actors with just a handful of films with smaller vote counts can only appear to have perfect ratings.
- Excluding episodes, the **top 10 highest-rated popular movies** often feature actors with above-average weighted ratings.

At the end of the day, the evidence supports the idea that great actors are more likely to have been selected into great films, rather than the idea that actors are **causing** higher ratings. In other words, good movies pick good actors to star (and not always the other way around).

In actuality, movie quality is likely a result of many factors working in combination with each other **beyond** only the actors, number of votes, and average rating. The director, production value, script, and other factors could be at play.

If we want to look more into the **causes** of highly-rated and highly-popular movies, we'll need to look more into **causal inference**, which is another huge topic worth worth looking into in **future investigations**.

QUESTION 4:

Lastly, we'll finally ask: **which genres get the highest ratings?**

```
# [1] Define our variables.
unique.length <- length(unique(full_dt$genres))                                  # Defines the length o
#anyNA(full_dt$genres)                                                           # Remember there are m

list.of.genres <- strsplit(unique(full_dt$genres), split = ",")                  # Split all unique gen
pure.genres <- unique(unlist(list.of.genres))                                    # Unlist and get uniqu
pure.genres <- pure.genres[!pure.genres %in% c("Short", NA)]                     # Remove unwanted entr

# Here, list.of.genres is a list of all the genres, each entry being a vector.
# ex: The character "Documentary, Short" will turn into the vector ("Documentary, "Short")

vector.of.genres <- unlist(list.of.genres)                                       # Unpacks a list list.
pure.genres <- unique(vector.of.genres)                                          # Defines a vector pur

# [2] Remove the entries NA and "Short". Those are NOT genres.
pure.genres <- pure.genres[!pure.genres %in% c("Short", NA)]                     # These are all the of
```

For this analysis, we will **assume** that we're working with **pure genres**.

- We have defined a vector **pure.genres** that contains all 27 unique, pure genres in our dataset.
- A **pure** genre is the opposite of a combined genre. For instance, our vector's entries say words such as "Action" or "Comedy" rather than something such as "Action Comedy."

It's important to note that our vector **pure.genres** does not include the words "Short" and NA:

- Long story short: NA is a missing value, while "Short" is not a genre. (It's a format.)
- In addition, when you quickly look at the column "genres" within our dataset, you will briefly see plenty of rows that contain a genre that says "Short" (and nothing more).
- This is clearly not helpful, as it tells us nothing about what the genre of the film was.
- So for our analysis, we made the decision to **ignore** films whose corresponding genre says just "Short" or NA.

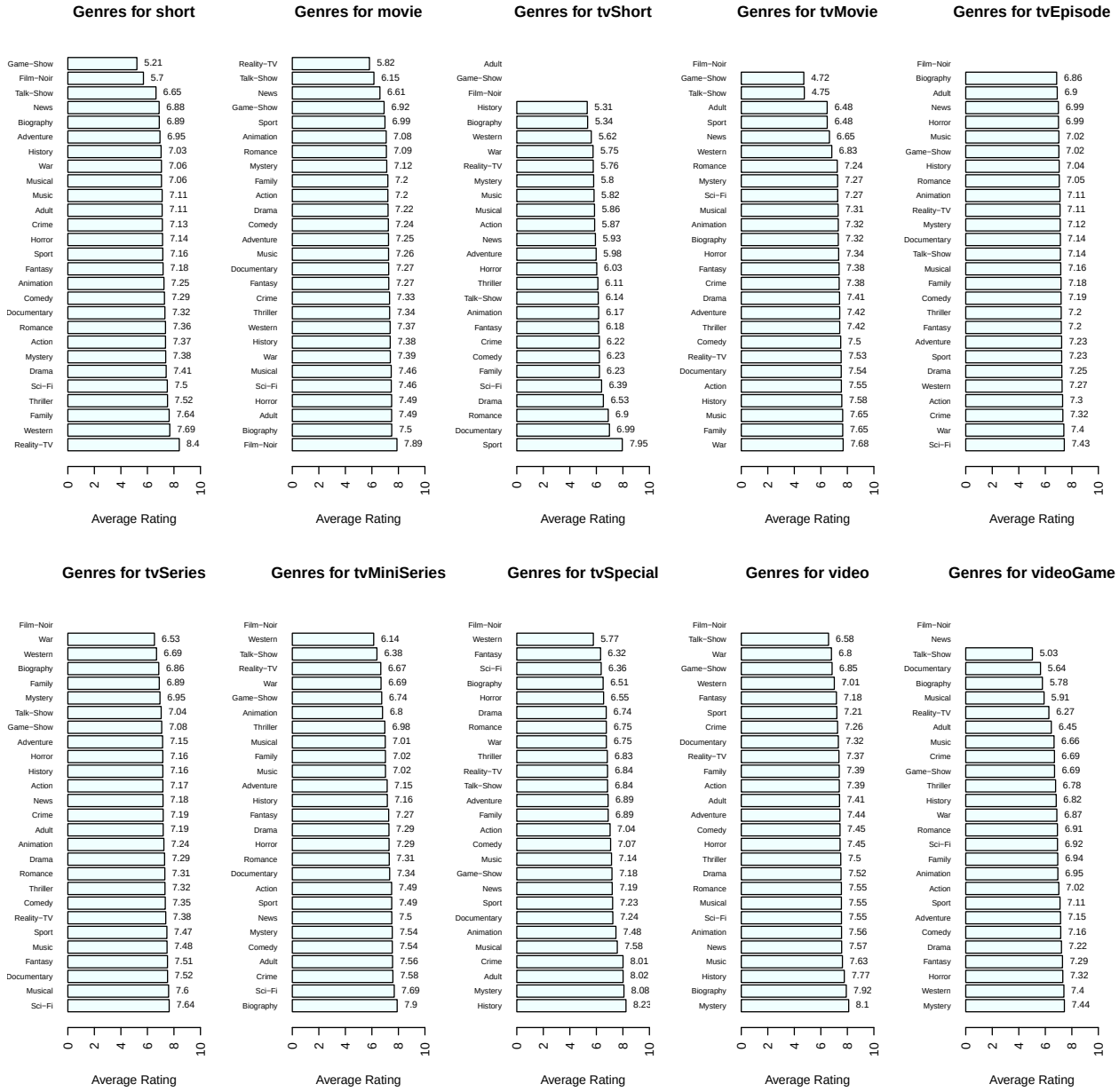
```
# Checking how many rows (i.e. films) just say "Short" and NA:
number.of.shorts <- length(which(full_dt[,9] == "Short"))                  # Number of rows in dataset
number.of.NA <- sum(is.na(full_dt[,9]))                                    # Number of rows in dataset


# Here are the proportions, where the total number of rows is 1542357:
number.of.shorts / 1542357                                                 # Proportion of "Short" row
number.of.NA / 1542357                                                     # Proportion of missing val
(number.of.shorts + number.of.NA) / 1542357                                # Proportion of both "Short
```

As we can see, there are 26,904 rows that just say "Short" and 21,464 rows that contain missing values, making up 1.74% and 1.39% of the dataset's total rows. In total, they make up around 3% of the dataset's total rows.

- That is **really, really small**.
- So, we'll proceed to ignore those "Short" and missing value rows for our analysis.

```
par(mfrow = c(2, 5))

for (i in unique(full_dt$titleType)){                                      # [1] For each kind o
  number.of.ratings <- c()                                                 # Initialize empty ve
  average.ratings <- c()                                                   # Initialize empty ve
```

```
  movie.indices <- which(full_dt$titleType == i)

  for (j in pure.genres){                                                    # Update both empty v
    indices <- which(grepl(j, full_dt$genres[movie.indices]))
    how.many.ratings <- length(indices)

    ratings <- full_dt$averageRating[indices]
    number.of.votes <- full_dt$numVotes[indices]
    weighted.average <- sum(ratings * number.of.votes) / sum(number.of.votes)

    average.ratings <- append(average.ratings, weighted.average)
    number.of.ratings <- append(number.of.ratings, how.many.ratings)
  }

  ratings.df <- data.frame(genre = pure.genres,                              # [2] Make a barplot
                           number = number.of.ratings,
                           average = average.ratings)
  ratings.df <- ratings.df[order(ratings.df$average, decreasing = TRUE), ]

  my.barplot <- barplot(ratings.df$average, main = paste("Genres for", i),
          names.arg = ratings.df$genre,
          xlab = "Average Rating",
          las = 2,
          col = "azure1",
          cex.names = 0.6,
          horiz = TRUE,
          xlim = c(0, 10))

  text(x = ratings.df$average,
       y = my.barplot,
       labels = round(ratings.df$average, 2),
       pos = 4,
       cex = 0.7,
       col = "black")
}
```



Here is what we should take away from the plot.

1. Each plot displays the least to most ranked genres within each kind of film — shorts, movies, video games, etc.
2. All these plots have genres with weighted average ratings roughly between 5 to 7.
3. This range of average ratings is clearly a lot narrower than we probably intuitively expected to see. Ratings can go from 1-10, but these go from 5-7.

Back then, we simply calculated a regular **average** using the mean() function, which incorrectly acts as if all films have received the same number of votes.

- Consequently, we got (some admittedly) bizarre results.
- For instance, three of our box plots used to suggest that adult genres have been the most highly rated genres for tvMovies, tvSeries, and tvMiniSeries.

So instead, we computed a **weighted average** rating for each genre.

- This is crucial because upon looking at the column **numVotes**, we can see that not every film has had the same number of votes. In other words, not every film has had the same number of reviewers.
- Also, not all box plots have a plotted bar for every genre. For instance, the lower right box plot tells us that there weren't any videogames under the "News" genre (which makes sense). In these kinds of cases, where some genres have zero matched films, the code made sure to automatically exclude those genres from the weighted average calculation.
- The number of votes clearly influences how each genre's computed average rating will turn out.
- By computing a weighted average, we are giving less influence to small genres that have skewed/inflated ratings and more influence to larger genres.

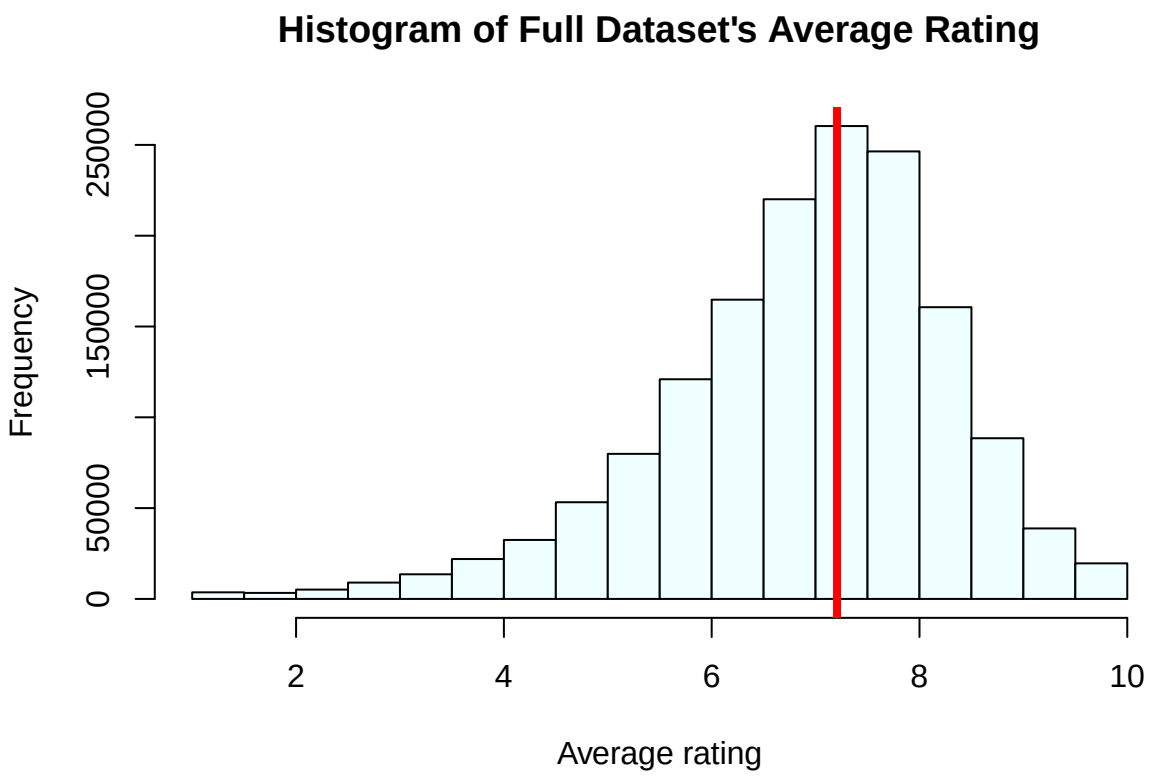
Now, what could explain the narrow range of a rating of 5-7?

To start, let's check out the distribution of the averageRating column from our dataset.

```
par(mfrow = c(1, 1))
hist(full_dt$averageRating,
     col = "azure1",
     xlab = "Average rating",
     ylab = "Frequency",
     main = "Histogram of Full Dataset's Average Rating")
summary(full_dt$averageRating)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00    6.20    7.10    6.95    7.90   10.00
```

```
ratings <- full_dt$averageRating
number.of.votes <- full_dt$numVotes
weighted.average <- sum(ratings * number.of.votes) / sum(number.of.votes)
abline(v = weighted.average, col = "red", lwd = 4)
```



As we can see:

- The distribution of the **averageRating** column from our dataset is already indeed narrow, with a

center around 7.206151 (i.e. the weighted average of the entire dataset).
- In addition, the number of ratings used to compute each pure genre's average are all relatively large, with the smallest being only 88 ratings for the genre **Reality-TV**.

There are some possible reasons why the distribution of the full dataset's average rating is clustered between the 5-7 range, even though ratings can go from 1-10.

1. **Selection bias**: People generally review films that they're really interested in or chose to watch voluntarily. They may also review films that they personally find really controversial. Overall, people are likely to give reviews for films that stick out to them for any reason. This could affect the histogram and land its final focus around 5-7.
2. **IMDb anti-skew measures**: IMDb is said to employ anti-skew measures when collecting their survey data — they employ certain weights to ensure that meaningful reviews are counted more while meaningless, jokingly, deliberately, and excessively negative or positive reviews are counted less. This could affect the histogram and land its final focus around 5-7.

Some final notes to conclude this section with:

- One factor that **could** lead to a change in results is the fact that we had to omit films whose specified genres either said "Short" or "genre." If we knew what the genres of those films actually were, then their contribution could change the outcome of our computed weighted averages.
- In addition, the average ratings don't necessarily determine what's more popular — it's **also** important to consider other factors such as **total box office revenue**.

QUESTION 5:

TODO: Needs cleaning. Decide whether to include this or not. The final question we want to ask is: **does experience in creative roles improve film ratings?**

In earlier questions, we explored whether a film has a **writer** (Question 2), or how **vote count** impacts ratings (Question 1). But now we want to ask a deeper question:

> Does the *experience* of the writers, directors, or actors improve a film's average rating?

We define experience as the **number of films** a person has previously appeared in, in a specific role. Then, for each film, we compute the **median experience** of each group.

---

[1] Compute experience metrics for writers, directors, and actors

We start by reshaping the nested crew structure (`nconst`, `category`) to count how often each person appears in a given role. Then we compute the **median experience** per film, and merge it into our dataset.

```
q5_dt <- full_dt

# Flatten nested roles into long format
long_principals <- full_dt[, .(tconst, nconst, category)]
long_principals <- long_principals[
  , .(nconst = unlist(nconst),
      category = unlist(category)),
  by = tconst
]

# Count appearances per person per category
role_experience <- long_principals[, .N, by = .(nconst, category)]
setnames(role_experience, "N", "experience")

# Function to compute median experience per film for a role
get_role_experience_by_film <- function(role_name) {
  role_exp <- role_experience[category %in% role_name]
  role_map <- full_dt[, .(tconst, nconst, category)]
  role_map <- role_map[, .(
    nconst = unlist(nconst),
    category = unlist(category)
  ), by = tconst]
  role_map <- role_map[category %in% role_name]
  role_map <- merge(role_map, role_exp, by = "nconst", all.x = TRUE)
  role_exp_by_film <- role_map[, .(
    exp_median = median(experience, na.rm = TRUE)
  ), by = tconst]
  role_name <- if ("actor" %in% role_name) "actor" else role_name[1]
  setnames(role_exp_by_film, "exp_median", paste0(role_name, "_exp_median"))
  return(role_exp_by_film)
}

# Get experience for each group
writer_exp_by_film   <- get_role_experience_by_film(c("writer"))
director_exp_by_film <- get_role_experience_by_film(c("director"))
actor_exp_by_film    <- get_role_experience_by_film(c("actor", "actress"))
```

```r
# Merge into q5_dt
q5_dt <- merge(q5_dt, writer_exp_by_film,   by = "tconst", all.x = TRUE)
q5_dt <- merge(q5_dt, director_exp_by_film, by = "tconst", all.x = TRUE)
q5_dt <- merge(q5_dt, actor_exp_by_film,    by = "tconst", all.x = TRUE)

# Replace NAs with 0 for missing experience values
exp_cols <- grep("_exp_median$", names(q5_dt), value = TRUE)
for (col in exp_cols) {
  q5_dt[is.na(get(col)), (col) := 0]
}
```

---

[2] Fit GAMs to assess the role of experience

We now fit two Generalized Additive Models (GAMs):
- One with separate effects for each group
- Another with an interaction between writer and director experience

```r
library(mgcv)

# Additive GAM with median experience
gam_median <- gam(averageRating ~
                    s(writer_exp_median) +
                    s(director_exp_median) +
                    s(actor_exp_median),
                  data = q5_dt)

# Interaction GAM: writer × director
gam_median_inter <- gam(averageRating ~
                          te(writer_exp_median, director_exp_median),
                        data = q5_dt)

# View model summaries
summary(gam_median)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## averageRating ~ s(writer_exp_median) + s(director_exp_median) +
##     s(actor_exp_median)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 6.950173   0.001097    6335   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                           edf Ref.df      F p-value
## s(writer_exp_median)   8.908  8.997  799.2  <2e-16 ***
## s(director_exp_median) 8.950  8.999  184.8  <2e-16 ***
## s(actor_exp_median)    8.887  8.996 1232.5  <2e-16 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.0308   Deviance explained = 3.08%
## GCV = 1.8562  Scale est. = 1.8562    n = 1542357
```

```
summary(gam_median_inter)
```

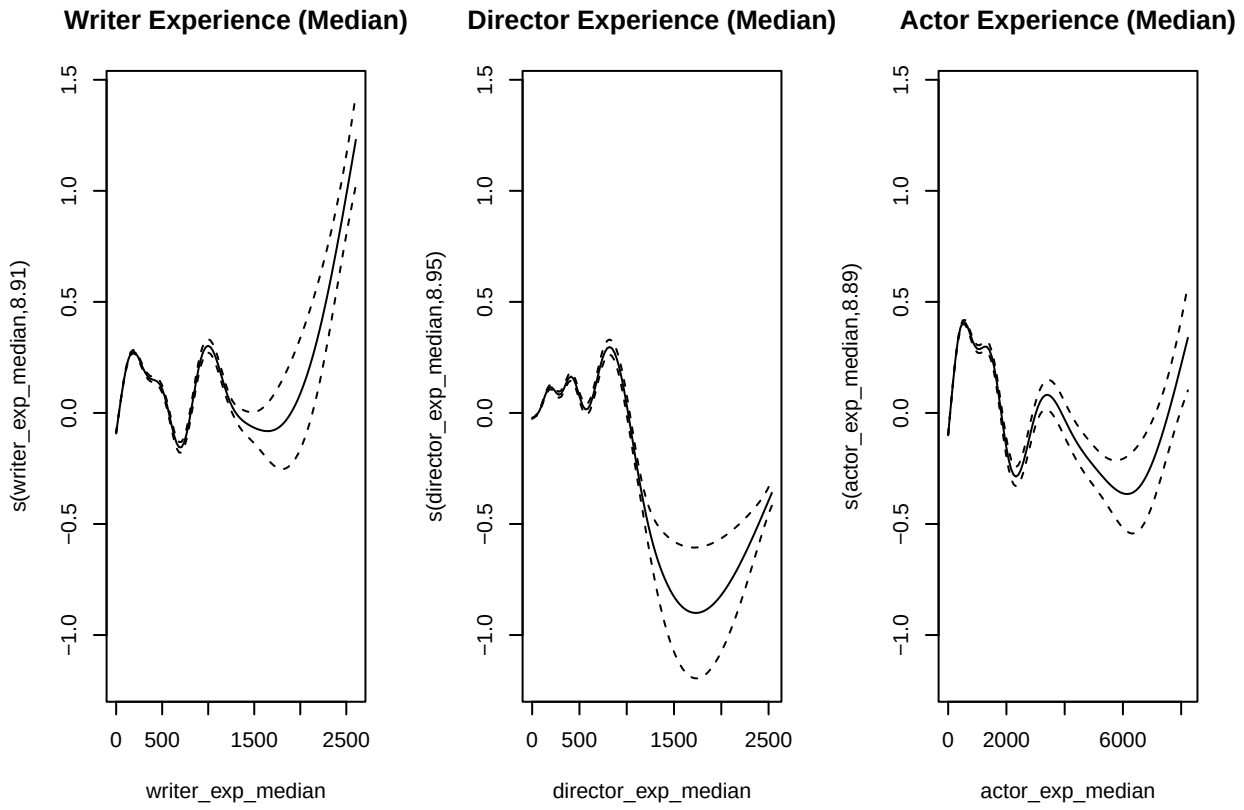
```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## averageRating ~ te(writer_exp_median, director_exp_median)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   6.9502     0.0011    6317   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                                            edf Ref.df    F p-value
## te(writer_exp_median,director_exp_median) 23.42  23.75 1682  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.0252   Deviance explained = 2.52%
## GCV = 1.8668  Scale est. = 1.8668    n = 1542357
```

---

[3] Visualize the effects of experience

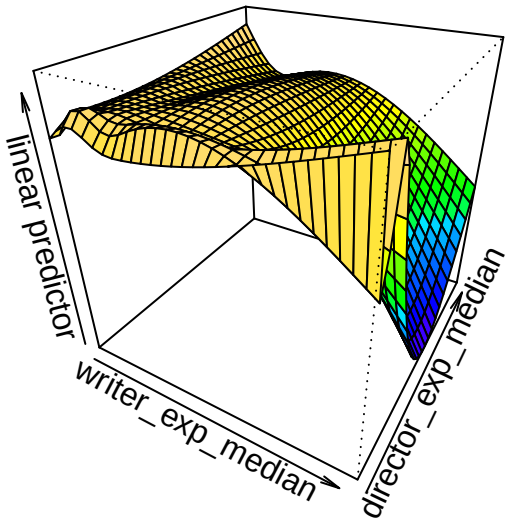
Each plot shows how the median experience in one creative group affects the predicted rating, while holding the others constant.

```
# 1D smooth plots
par(mfrow = c(1, 3))
plot(gam_median, select = 1, main = "Writer Experience (Median)")
plot(gam_median, select = 2, main = "Director Experience (Median)")
plot(gam_median, select = 3, main = "Actor Experience (Median)")
```




```
# 3D interaction plot (writer × director)
par(mfrow = c(1, 1))
vis.gam(gam_median_inter,
        view = c("writer_exp_median", "director_exp_median"),
        plot.type = "persp",
        color = "topo",
        theta = 30, phi = 30,
        main = "3D Surface: Writer × Director Median Experience")
```

3D Surface: Writer × Director Median Experience



Here is what we takeaway from this analysis:

- **Writer experience** shows a mild upward trend: films with more seasoned writers tend to have slightly higher average ratings.
- **Director experience** shows a nonlinear shape, where moderate experience is associated with better outcomes than either very low or very high.
- **Actor experience** has a mostly flat effect, suggesting their experience doesn't significantly drive perceived film quality.
- The **interaction plot** reveals that the combined experience of writers and directors can lead to more favorable ratings — especially when both are moderately experienced.

These trends are statistically significant (due to large sample size), but practically modest. Still, they provide a nuanced look at the **behind-the-scenes influence** on how films are received.

This analysis highlights that experience matters — especially for those who write and direct films — though not necessarily in a linear or extreme way.