STSCI 4100x

Tony Oh (do256), William Rhee (wr86)

2025-03-25

App Ideas: - Filter recommendation system: basic, but we could try to implement AI. - Rating estimator: basic regression analysis. - Success estimator: I was thinking we could take rating to be a high heuristic value while number of votes and help determine success. If there's many votes, the more reliable the ratings can be, so we can deem more extreme ratings with many votes as more correct in terms of success. Then, we can estimate the level of success a movie would have based on their input of title, is adult, start year, end year, endtime in minutes, and genres. Basically a regression with heuristics for success.

Possible questions: 1. Does the number of votes affect the ratings? - Graph: x = numVotes, y = rating - See if normal distribution 2. Does having a writer affect the ratings? 3. Does having certain actors have an affect on ratings? Or is this just an effect of hiring good actors for good movies? 4. How does runtime affect ratings? 5. Which genres get the highest ratings?

TODO: - Combine datasets with unique tconst: basics, crew, ratings (excluding principals). - Figure out what principals dataset is.

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.

```
library(readr)

basics <- read.delim("title.basics.tsv", sep = "\t", header = TRUE,
                     na.strings = "\\N", nrows = 200000)
crew <- read.delim("title.crew.tsv", sep = "\t", header = TRUE,
                   na.strings = "\\N", nrows = 200000)
principals <- read.delim("title.principals.tsv", sep = "\t", header = TRUE,
                         na.strings = "\\N", nrows = 200000)
ratings <- read.delim("title.ratings.tsv", sep = "\t", header = TRUE,
                      na.strings = "\\N", nrows = 200000)

#head(basics)
#head(crew)
#head(principals)
#head(ratings)

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

[B] A Quick Glance.

```
# Let's view our datasets.
#View(basics)
#View(crew)
#View(principals)
#View(ratings)

# How many columns are in our datasets?
for (i in names(list.of.datasets)){
  column.names <- paste(colnames(list.of.datasets[[i]]), collapse = ", ")
  print(paste("In the", i, "dataset, there are", ncol(list.of.datasets[[i]]), "columns."))
  print(paste("Its columns are:", column.names))
  cat("\n")
}
```

```
## [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"
```

```
# How many rows are in our datasets?
for (i in 1:4) print(paste("There are", nrow(list.of.datasets[[i]]), "rows in the", names(list.of.datas
```

```
## [1] "There are 200000 rows in the basics dataset."
## [1] "There are 200000 rows in the crew dataset."
## [1] "There are 200000 rows in the principals dataset."
## [1] "There are 200000 rows in the ratings dataset."
```

[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 is a quick exploration of the **basics** dataset's columns:

```
anyNA(basics$tconst)                                # Investigating the column: tconst.
length(unique(basics$tconst)) == nrow(basics)

anyNA(basics$titleType)                             # Investigating the column: titleType.
unique(basics$titleType)

anyNA(basics$primaryTitle)                          # Investigating the column: primaryTitle.
length(unique(basics$primaryTitle))

anyNA(basics$originalTitle)                         # Investigating the column: originalTitle.
length(unique(basics$originalTitle))

anyNA(basics$isAdult)                               # Investigating the column: isAdult.
mean(basics$isAdult)

anyNA(basics$startYear)                             # Investigating the column: startYear.
min(basics$startYear, na.rm = TRUE)
max(basics$startYear, na.rm = TRUE)

anyNA(basics$endYear)                               # Investigating the column: endYear.
min(basics$endYear, na.rm = TRUE)
max(basics$endYear, na.rm = TRUE)

anyNA(basics$runtimeMinutes)                        # Investigating the column: runtimeMinutes.
min(basics$runtimeMinutes, na.rm = TRUE)
max(basics$runtimeMinutes, na.rm = TRUE)
mean(basics$runtimeMinutes, na.rm = TRUE)
```

```r
anyNA(basics$genres)
```

Here are the takeaways from the given code:

- **tconst**: There are no missing values. There are exactly 200,000 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 179081 primaryTitles in the dataset ($<$ 200,000). This should be looked into.
- **originalTitle**: There are no missing values. There are 182034 originalTitles in the dataset ($<$ 200,000 and $>$ 179081). This should be looked into.
- **isAdult**: There are no missing values. About 7% of all the films in the dataset are adult films. (OMG)
- **startYear**: There are missing values! Ignoring missing values, the earliest release date was 1892, while the most recent release date was 2025.
- **endYear**: There are missing values! Ignoring missing values, the earliest end date was 1945, while the most recent end date was 2026.
- **runtimeMinutes**: There are missing values! Ignoring missing values, the shortest film is a minute long, while the longest film is 1,620 minutes (27 hours) long. The average film length is around 75 minutes (1.25 hours).
- **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 is a quick exploration of the **crew** dataset's columns:

```r
anyNA(crew$tconst)                                  # Investigating the column: tconst.
length(unique(crew$tconst)) == nrow(crew)

anyNA(crew$directors)                               # Investigating the column: directors.
anyNA(crew$writers)                                 # Investigating the column: writers.
```

Here are the takeaways from the given code:

- **tconst**: There are no missing values. There are exactly 200,000 id's (i.e. individual films), one for every row.
- **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, where 1 is the most important and 35 is least important.
- **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 is a quick exploration of the **principals** dataset's columns:

```
anyNA(principals$tconst)                                    # Investigating the column: tconst.
length(unique(principals$tconst)) == nrow(principals)

anyNA(principals$ordering)                                  # Investigating the column: ordering.
unique(principals$ordering)

anyNA(principals$nconst)                                    # Investigating the column: nconst.

anyNA(principals$category)                                  # Investigating the column: category.
unique(principals$category)

anyNA(principals$job)                                       # Investigating the column: job.
# unique(principals$job)                                    # Do not run. Leave this commented out.

anyNA(principals$characters)                                # Investigating the column: characters.
# unique(principals$characters)                             # Do not run. Leave this commented out.
```

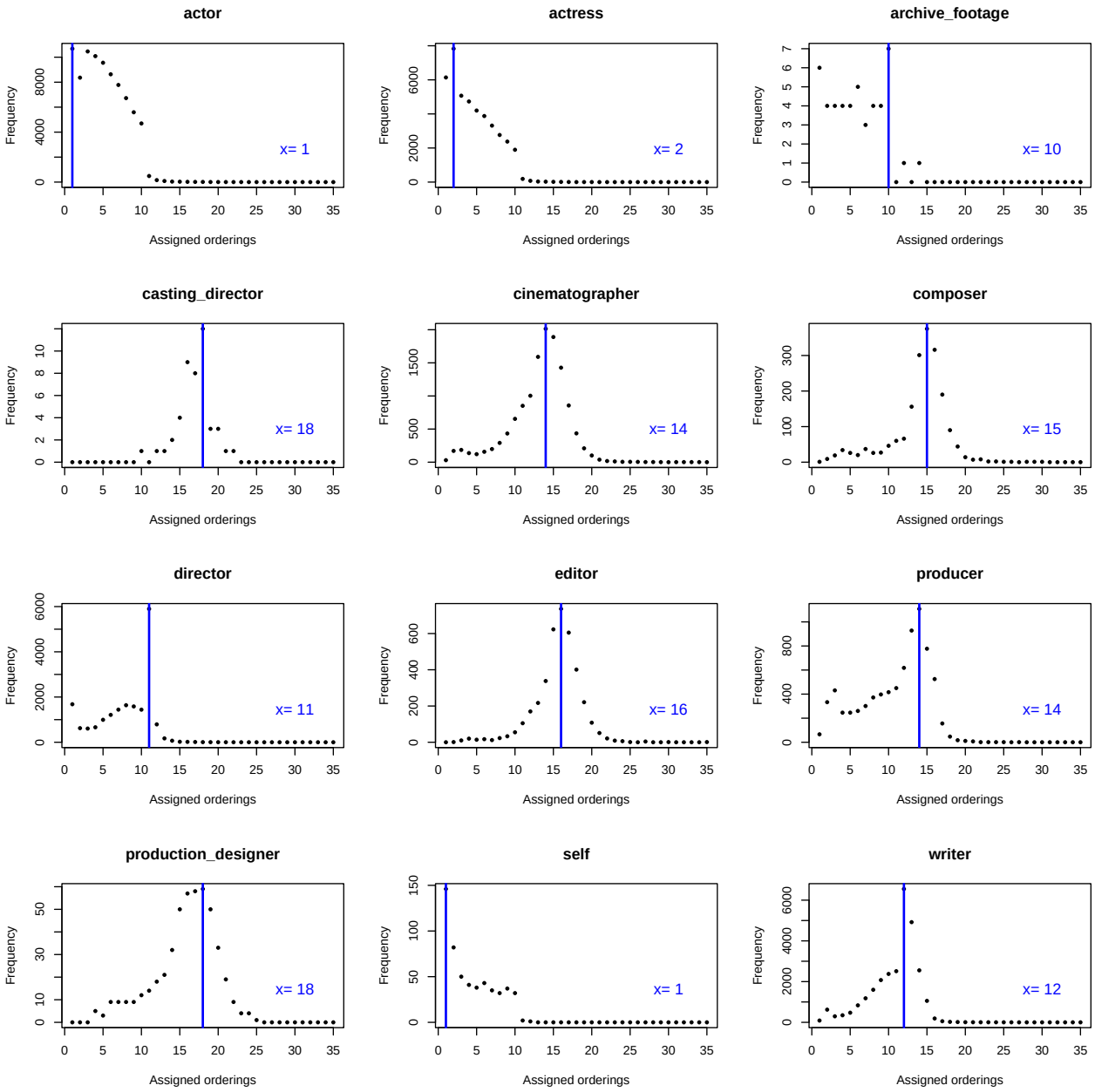
Here are the takeaways from the given code:

- **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 35.
- **nconst**: There are no missing values.
- **category**: There are no missing values. There are 12 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 too many unique values within this column for R to run through without crashing (several times).
- **characters**: There are missing values — this is because of the duplicate values from the first column. There are too many unique values within this column for R to run through without crashing (several times).

A side-note

We know that there are 12 unique categories within the **category** column. We also know that the ordering ranges from 1 to 35, where 1 is most important and 35 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       10
## 4    casting_director       18
## 5     cinematographer       14
## 6            composer       15
## 7            director       11
## 8              editor       16
## 9            producer       14
## 10 production_designer       18
## 11               self        1
## 12             writer       12
```

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

- In addition, we can see that the ordering for the category **archive_footage** has an unclear assignment, where it can be assigned to an ordering of either 1 or 10.
- This is because the category was equally assigned to those orderings an equal number of times.

[C4] The ratings dataset

The **ratings** dataset:

- **tconst**: The unique id for a specific film. There are exactly 200,000 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 is a quick exploration of the **ratings** dataset's columns:

```
anyNA(ratings$tconst)                                    # Investigating the column: tconst.
length(unique(ratings$tconst)) == nrow(ratings)

anyNA(ratings$averageRating)                             # Investigating the column: averageRating.
unique(ratings$averageRating)
min(ratings$averageRating)
max(ratings$averageRating)
mean(ratings$averageRating)

anyNA(ratings$numVotes)                                  # Investigating the column: numVotes.
min(ratings$numVotes)
length(which(ratings$numVotes == min(ratings$numVotes)))
max(ratings$numVotes)
length(which(ratings$numVotes == max(ratings$numVotes)))
which(ratings$numVotes == max(ratings$numVotes))
```

Here are the takeaways from the given code:

- **tconst**: There are no missing values. There are exactly 200,000 id's (i.e. individual films), one for every row.
- **averageRating**: There are no missing values.
- **numVotes**: 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 6/10. Of all 200000 films, 513 films have received the lowest number of reviews (only 5 reviews). Of all 200000 films, only one film (i.e. the film in the 84776th row) has the largest number of reviews (3026798 reviews).

D. Combining the Datasets.

Excluding the principals dataset, let's combine the three datasets that share the same first column **tconst** (i.e. the same unique ID values):

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

```
# COMBINING DATASETS:
# [1] Combining all three datasets into one final dataset.
my.dataset <- merge(basics, crew, by = "tconst")
my.dataset <- merge(my.dataset, ratings, by = "tconst")
#View(my.dataset)
```

```r
# [2] Checking the dimensions (rows, col).
dim(basics)                                    # This is correct - it should be (200000 rows, 9 col
```

```
## [1] 200000      9
```

```r
dim(crew)                                      # This is correct - it should be (200000 rows, 3 col
```

```
## [1] 200000      3
```

```r
dim(ratings)                                   # This is correct - it should be (200000 rows, 3 col
```

```
## [1] 200000      3
```

```r
dim(my.dataset)                                # This is WRONG - it should be (200000 rows, 13 colu
```

```
## [1] 139468     13
```

As we can see:

1. We are able to merge the basics dataset with the crew dataset based on the first column. This is because all of their values within the first column match.
2. However, our final dataset does **not** contain 200,000 rows. It has the right number of columns, but this is still an issue we need to fix.

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

```r
# DIAGNOSTICS: [PART 1]
# [1] Comparing the first column of each dataset.
all(basics$tconst == crew$tconst)                                  # Check: are the first colum
all(basics$tconst == ratings$tconst)                               # Check: are the first colum

# Here, we found out that the datasets basics and crew have the exact same first column - both their ro
# Here, we found out that the dataset ratings does not match - either by a difference in their rows (i.

# [2] Comparing the first column's rows (order).
mismatched.rows <- which(basics$tconst != ratings$tconst)          # A vector of all the rows t
c(min(mismatched.rows), max(mismatched.rows), length(mismatched.rows))  # A quick exploration of mis
all(mismatched.rows == 178:200000)                                 # It seems that mismatched.r

all(basics$tconst[1:177] == ratings$tconst[1:177])                 # Double check: does the fir
all(basics$tconst[178:200000] == ratings$tconst[178:200000])       # Double check: does the fir
```

Here, we found out that rows 178 to 200,000 of the first column between the basic and ratings dataset do not match. But here's the weird thing:

- The merge() function doesn't seem to care whether the ROWS of our datasets match.
- Why? Because the number of mismatched rows is 199,823 rows. If the merge() function cared about this number, then the final dataset will only have 200,000 - 199,823 = 177 rows.
- In reality, our final dataset ended up with 139602 rows. That is only 200,000 - 139,602 = 60,398 rows short (not 177).
- Therefore, our diagnosis is **not over** yet.

In summary: the fact that only the first 177 rows in the first column match between the basics and ratings dataset is **NOT** the reason why our final merged dataset didn't end up with 200,000 rows.

```r
# DIAGNOSTICS: [PART 2]
# [1] Comparing the first column's values (order does not matter).
length(unique(basics$tconst))                                  # Check: Do we have 200,000 uni
length(unique(crew$tconst))                                    # Check: Do we have 200,000 uni
length(unique(ratings$tconst))                                 # Check: Do we have 200,000 uni
```

```r
# Here, we found out that all three datasets have exactly 200,000 unique ID values. That's good.

all(unique(basics$tconst) == unique(crew$tconst))                    # Check: Are the first column's
all(unique(basics$tconst) == unique(ratings$tconst))                 # Check: Are the first column's

# Here, we found out that the datasets basics and crew have the exact same first column - both their ro
# Here, we found out that the datasets basics and ratings do NOT have the exact same first column - one
```

Here's the big takeaway from this diagnosis:

- All three datasets have exactly 200,000 unique ID values — however, the **ratings** dataset does NOT cover ALL the same films.
- Instead, the basics and crew datasets talk about the exact same 200,000 films.
- Instead, the ratings dataset is a loner that brings up movies that aren't shared with those two datasets. (Most of them are in common, but not all.)

So according to our final merged dataset's dimensions, we can say that out of all 200,000 films listed in each dataset, only 139602 of them are shared between all three datasets.

- The remaining 60398 films are films that the ratings dataset doesn't have with the basics and crew dataset.
- The ratings dataset talks about 60398 other films instead.

So what does this mean moving forward?

- It means that before we analyze any data, we'll have to make the decision to only analyze the films all three datasets share (i.e. only 139602 out of 200,000 films).
- If we choose to bluntly include ALL movies in all three datasets, we'll just end up with a final merged DataFrame that contains NA's for the films that don't have reviews.
- If considering the reviews of a movie are important, then we should stick with just analyzing the 139,602 out of 200,000 films (because we won't have reviews for the rest).
- If considering as many films as possible is important, then we should include all 200,000 films (even if they obviously won't have reviews attached to them).

As a final check, here is the last diagnosis to confirm these findings about the differences between the basics/crew dataset and the ratings dataset.

```r
difference.1 <- setdiff(basics$tconst, crew$tconst)                  # A vector that lists all the elemen
length(difference.1)                                                 # This should be zero because both b

difference.2 <- setdiff(basics$tconst, ratings$tconst)               # A vector that lists all the elemen
length(difference.2)                                                 # This should be 60398 because basic

difference.3 <- setdiff(ratings$tconst, basics$tconst)               # A vector that lists all the elemen
length(difference.3)                                                 # This should be 60398 because ratin

shared.amount1 <-intersect(basics$tconst, crew$tconst)               # A vector that lists all the elemen
length(shared.amount1)                                               # This should be 200,000. (It is!)

shared.amount2 <-intersect(basics$tconst, ratings$tconst)            # A vector that lists all the elemen
length(shared.amount2)                                               # This should be 139602. (It is!)
```

---

[3] DATA ANALYSIS:

Now that we have gone through the datasets and obtained a final dataset **my.dataset** 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?**

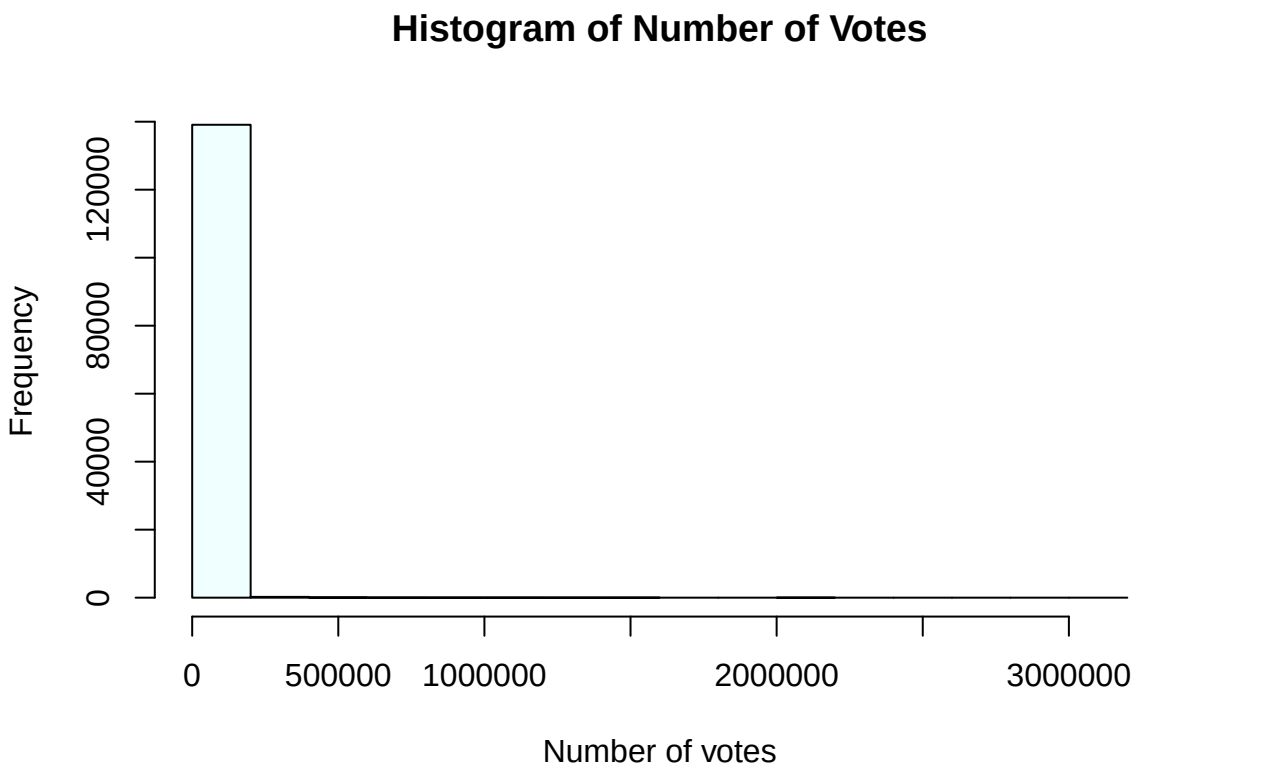
In the real world, many people want to know how good a movie is before watching it. To do so, they go online to check the movie's reviews. But here's the thing everyone suspects:

- 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.

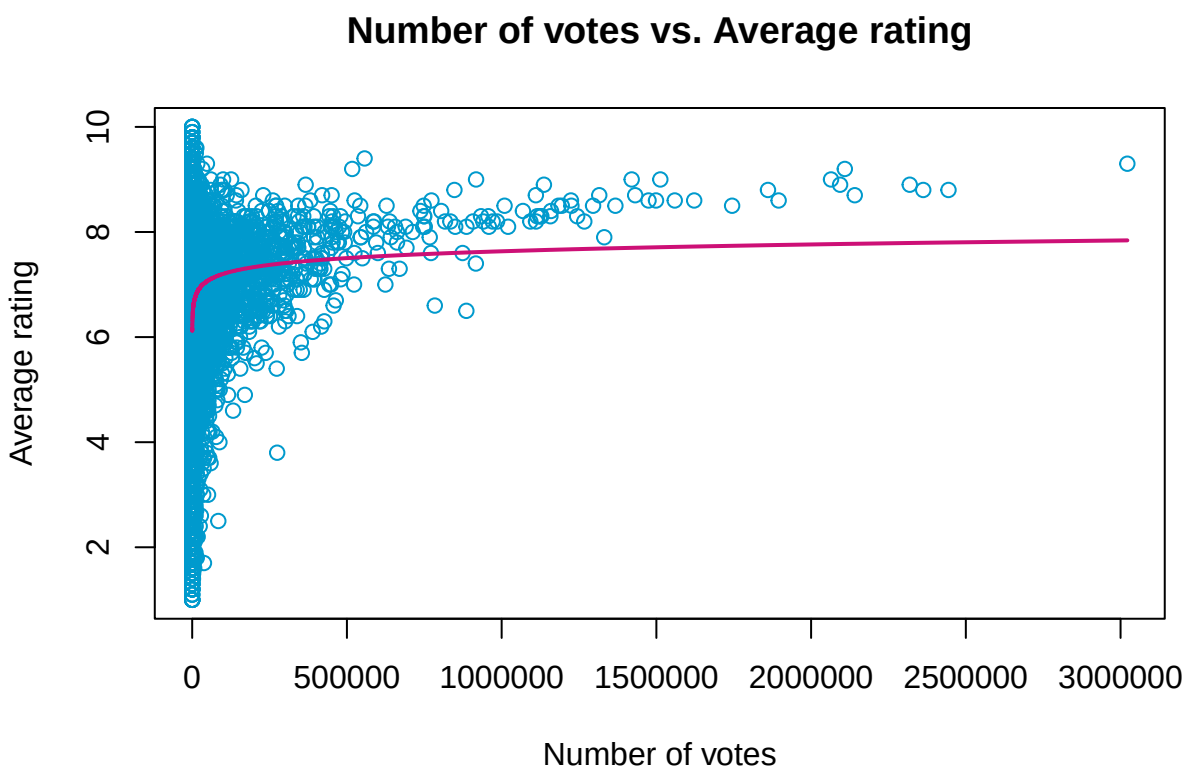
```
# [1] First, let's visualize the distribution of the numVotes column.
hist(my.dataset$numVotes,
     xlab = "Number of votes",
     ylab = "Frequency",
     main = "Histogram of Number of Votes",
     col = "azure1")
```




```
summary(my.dataset$numVotes)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       5      27      88    2913     349 3021924
```

```r
# [2] Second, let's plot the numVotes column (x) against the averageRating column (y).
x <- my.dataset$numVotes
y <- my.dataset$averageRating
plot(x = x,
     y = y,
     xlab = "Number of votes",
     ylab = "Average rating",
     main = "Number of votes vs. Average rating",
     col = "deepskyblue3")

# So far, the data doesn't seem to follow a strictly linear pattern.
# The data seems to follow some sort of nonlinear pattern.
# So, let's fit a local regression model - a non-parametric statistical method - to visualize the traje

# [3] Lastly, fit a local regression model.
my.lowess <- lowess(log10(x), y)                                    # Fitting a LOWESS mode
lines(10^my.lowess$x,                                               # Adds LOWESS curve.
      my.lowess$y,
      col = "deeppink3",
      lwd = 2)
```




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

```
## [1] 0.1132438
```

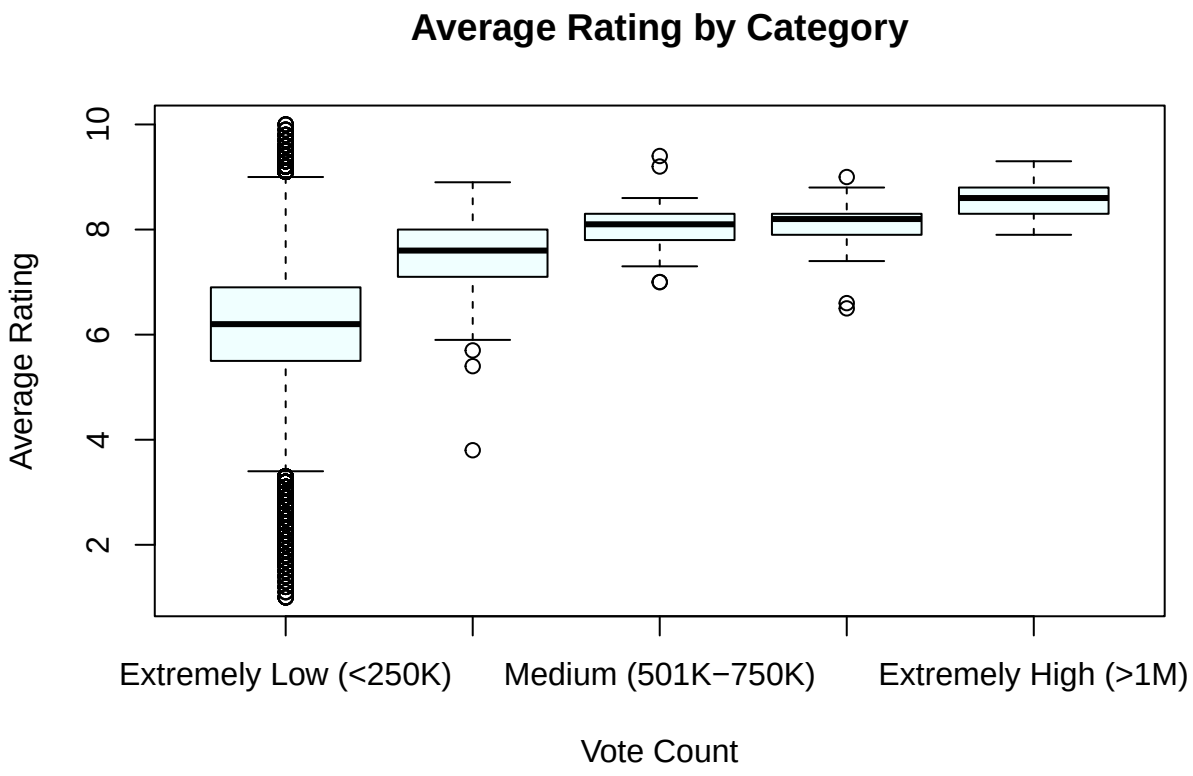
As we can see, the number of votes in our dataset is extremely right-skewed, where:

- Most movies have very few votes. (Seems to be realistic.)
- Few movies have lots of votes. (Seems to be realistic; barely in this case.)
- Since the data is skewed, we performed a log transformation on the number of votes before fitting a

local regression curve.

And here is a boxplot in support of our scatterplot with the fitted local regression:

```
my.dataset$votecategories <- NA
my.dataset$votecategories[my.dataset$numVotes <= 250000] <- "Extremely Low (<250K)"
my.dataset$votecategories[my.dataset$numVotes > 250000 & my.dataset$numVotes <= 500000] <- "Low (251K-5
my.dataset$votecategories[my.dataset$numVotes > 500000 & my.dataset$numVotes <= 750000] <- "Medium (501
my.dataset$votecategories[my.dataset$numVotes > 750000 & my.dataset$numVotes <= 1000000] <- "High (751K
my.dataset$votecategories[my.dataset$numVotes > 1000000] <- "Extremely High (>1M)"

my.dataset$votecategories <- factor(my.dataset$votecategories, levels = c("Extremely Low (<250K)",
                                                                           "Low (251K-500K)",
                                                                           "Medium (501K-750K)",
                                                                           "High (751K-1M)",
                                                                           "Extremely High (>1M)"))


boxplot(averageRating ~ votecategories,
        data = my.dataset,
        main = "Average Rating by Category",
        xlab = "Vote Count",
        ylab = "Average Rating",
        col = "azure1")
```



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.
   - Specifically, the Pearson correlation coefficient is close to zero at around 0.1142457 — this confirms there is no/an extremely weak **linear** relationship, as we visually saw.

2. As the number of votes increases, the average ratings tend to increase with more votes before stabilizing around 7-8.
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.

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:

```
# [1] Define a binary column called haswriter.
my.dataset$haswriter <- as.numeric(!is.na(my.dataset$writers))                  #  Binary column: 1 if

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

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  1.0000  1.0000  0.9003  1.0000  1.0000
```

```
mean(my.dataset$haswriter)
```

```
## [1] 0.9003069
```

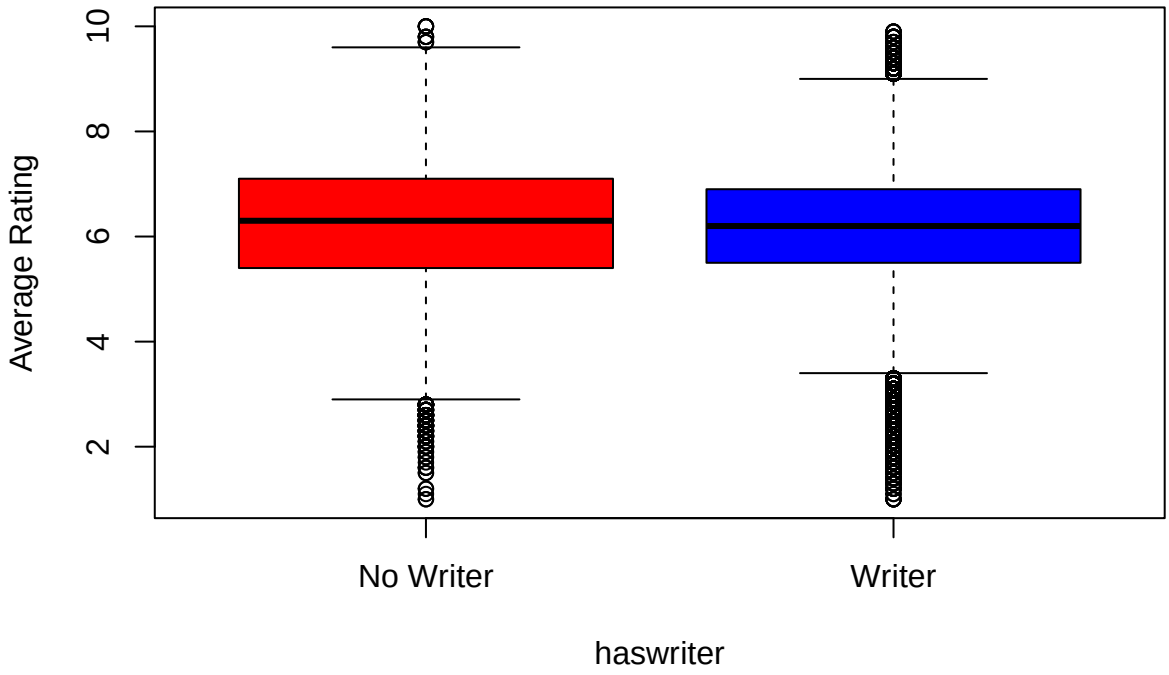
```
# Here, the min and max are obviously 0 and 1.
# Here, the mean of the column haswriter is 0.9000014. This means that roughly 90% of our films in the

# [3] Obtain a summary statistic and mean for the column averageRating.
summary(my.dataset$averageRating)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   5.500   6.200   6.157   6.900  10.000
```

```
mean(my.dataset$averageRating)
```

```
## [1] 6.157458
```

```
# Here, the worst films have an average rating of 1/10. The best films have an average rating of 10/10.
# Here, the average rating is 6.2/10.

# [4] Plot a boxplot of these summary statistics.
boxplot(averageRating ~ haswriter, data = my.dataset,
        col = c("red", "blue"),
        names = c("No Writer", "Writer"),
        ylab = "Average Rating",
        main = "Films without vs. with Specified Writers")
```




```
# Here, there doesn't seem to be much of a difference.
# Let's conduct a t-test to see if the difference is statistically significant.

# [5] Conduct the t-test.
t.test(averageRating ~ haswriter, data = my.dataset)
```

```
##
##  Welch Two Sample t-test
##
## data:  averageRating by haswriter
## t = 8.2853, df = 16385, 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:
##  0.07170255 0.11614242
## sample estimates:
## mean in group 0 mean in group 1
##        6.242017        6.148094
```

Here's our takeaway from our analysis:

1. In our comparison of films that have specified and unspecified writers, we cam see that their boxplots are identical. Their distribution's span are relatively the same with identical means around 6.
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 3.045e-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 6.239083 while the mean for "Writer" is 6.146662. Those means are incredibly close and identical.
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'll need to use the **principals** dataset because it contains actor information within the column **category**.

```
# [1] Define a new DataFrame that's the same as principals, EXCEPT it only contains the rows that say "
unique(principals$category)
```

```
##  [1] "self"                "director"            "producer"
##  [4] "cinematographer"     "composer"            "writer"
##  [7] "editor"              "actor"               "actress"
## [10] "production_designer" "archive_footage"     "casting_director"
```

```
actors.dataset <- principals[which(principals$category %in% c("actor", "actress")), ]

# [2] Merge the new DataFrame with my.dataset's "tconst" and "averageRating" column.
problem3 <- merge(actors.dataset,                                                    # New Data
                  my.dataset[, c("tconst", "averageRating")],                        # 7 column
                  by = "tconst")

dim(problem3)
```

```
## [1] 64602     7
```

Before we move on, let's quickly introduce a caveat regarding our new dataset **problem3** — it contains 64,974 rows, while the original dataset it was filtered out of (**principals**) contains 200,000 rows.

- So why was there a reduction in the number of rows? It's mainly because multiple rows within the principals dataset can tell information for the same, single movie.
- Unlike the main dataset my.dataset, we're no longer dealing with one row for one movie.
- Consequently, not every row within the original principals dataset is meant for an actor — it can be meant for a director, composer, producer, etc.
- So if we only pull out the rows that contain "actor" or "actress," then we are significantly dealing with only a subset of the principals dataset's 200,000 rows.

In addition, the same film can have **more than one** actor. So in our new dataset **problem3**, we can see rows with repeated "nm000000" values, as those rows are discussing multiple actors within the same film.

Besides our new dataset **problem3**, there are **other caveats** to recognize:

1. The original principals dataset does not always specify an actor for every movie.
2. Some actors played relatively small roles such as "sneezing man."
3. Worse, the principals dataset does not always even specify what role the actor played at all — we can't tell if this is an actor with a big role or not.

The following is our analysis for question 3. Note that we will do a **similar** analysis for question 4 in terms of the steps taken.

```
# [3] Compute each actor's number of films and average rating.
actors <- unique(problem3$nconst)                                                    # A vector of al
average.ratings <- numeric()                                                         # Initialize an
number.of.films <- numeric()                                                         # Initialize an

for (i in actors) {
  actor.indices <- which(problem3$nconst == i)                                       # A vector of th
  actor.ratings <- problem3$averageRating[actor.indices]                             # A vector of th
```

```
  average.ratings <- c(average.ratings, mean(actor.ratings, na.rm = TRUE))             # Computes the a
  number.of.films <- c(number.of.films, length(actor.ratings))                         # Computes the n
}

actor.results <- data.frame(nconst = actors,                                           # A DataFrame of
                            average.ratings = average.ratings,
                            number.of.films = number.of.films)
actor.results <- actor.results[order(actor.results$average.ratings, decreasing = TRUE), ]   # Reorganiz

head(actor.results)                                                                    # Displayin
```

```
##         nconst average.ratings number.of.films
## 1111 nm0367362               9               1
## 1295 nm0712121               9               1
## 4594 nm0696106               9               1
## 4595 nm0187006               9               1
## 4596 nm0861401               9               1
## 4629 nm0072594               9               1
```

As we can see, when we rearranged our DataFrame to show us the actors who starred in films that got the highest to least average ratings, we ended up seeing actors who only starred in one film at the top. This tells us something important:

- Many actors who starred in films with high ratings (like 9/10) were actors who, **at least according to our dataset**, starred in only one film.
- That's not that telling. Why? Here's how we think of it: an actor who only starred in one film that got 10/10 reviews will obviously have an average film rating of 10/10.
- That clearly doesn't say much in our analysis.
- On the other hand, an actor who starred in say 1000 films that all got 10/10 reviews will significantly mean more to our analysis.
- Yes, it could be true that if an actor starred in films that have an overall high rating, then the actor had some influence on the rating.
- However, if the actor has only appeared in one film (and that film has a high rating), then it's harder to reach those same conclusions.

So, let's see what happens if we focus on actors who starred in, say, at least 5 films.

```
# [1] Filter the DataFrame to only have actors who starred in at least 5 films.
filtered.actor.results <- actor.results[actor.results$number.of.films >= 5, ]          # This is a ne

# [2] Extract the top 10 actors - they are within the first 10 rows.
filtered.actor.results[1:10,]                                                          # Extract the
```

```
##          nconst average.ratings number.of.films
## 2414  nm0175050        7.620000               5
## 11886 nm0323479        7.400000               5
## 1110  nm0381936        7.383333               6
## 10679 nm0006471        7.380000               5
## 4480  nm0822623        7.366667               6
## 1412  nm0206750        7.340000               5
## 6394  nm0731247        7.333333              12
## 5071  nm0348162        7.300000               5
## 1112  nm0000858        7.263333              30
## 10580 nm0226158        7.250000               8
```

Now let's check out the actors with a Google search.

In IMDb's records, the first actor in our DataFrame — "nm0175050" — is **Bobby Connelly** (1909-1922). Here's the thing:

- He was a child actor during the early 1900's who unfortunately passed young at 13.
- So, the fact that the number of films the dataset claims he starred in is small (5) isn't unreasonable.
- The small number of films he starred in though explains why all his films' have a high final average rating of 7.62.

In IMDb's records, the third actor in our DataFrame — "nm0381936" — is **Jerold T. Hevener** (1873-1947). Here's the thing:

- He starred in 36 films, not six.
- Based on only six films, his films' average rating is 7.383333 out of 10.
- The fact that the given dataset does not accurately tell us the correct number of films he had in his career is one thing.
- But the fact that the number of films he starred in was only six — and he was not a child actor who passed young — in this dataset is still considerably small.

So, let's see what happens if we focus on actors who starred in, say, at least 20 films (this is larger than 5 films).

```
# [1] Filter the DataFrame to only have actors who starred in at least 20 films.
filtered.actor.results2 <- actor.results[actor.results$number.of.films >= 20, ]          # This is a

# [2] Extract the top 10 actors - they are within the first 10 rows.
filtered.actor.results2[1:10,]                                                          # Extract the
```

```
##          nconst average.ratings number.of.films
## 1112  nm0000858        7.263333              30
## 3498  nm0074788        7.220000              20
## 4034  nm0042317        7.018182              22
## 9456  nm0375609        7.014286              21
## 11513 nm0404257        6.918182              22
## 8493  nm0396012        6.908696              23
## 8771  nm0295992        6.885714              21
## 1286  nm0324553        6.876000              25
## 11452 nm0177371        6.857576              33
## 318   nm0832458        6.838462              26
```

Here, the first actor "nm0000858" is **John Barrymore** (1882-1942).

- He starred in at least 60 films, not 30.
- According to IMDb's documentary, this is because there are records of lost films that people believe he may have been featured in.
- Nevertheless, those 30 films have an average rating of 7.263333 / 10.

What's the point we're trying to get to?

1. The number of films in our dataset are not entirely accurate — many are smaller than they are because of **lost records** and poor documentation (most are early 1900's films).
2. The inaccurate number of films affects what their computed average ratings are.
3. Therefore, it's possible that there are actors within our dataset who may have either higher or lower final average ratings than our computations suggest.

Here's one last case to look at:

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

c(actor.results$nconst[indices[1]], actor.results$nconst[indices[2]])
```

```
## [1] "nm0115524" NA
```

```
c(actor.results$average.ratings[indices[1]], actor.results$average.ratings[indices[2]])
```

```
## [1] 5.688554       NA
```

Here, we have found two actors who, at least in our dataset, have starred in the largest number of films: **Kate Bruce** (1860-1946; nm0115524) and **Arthur V. Johnson(1876-1916)**.

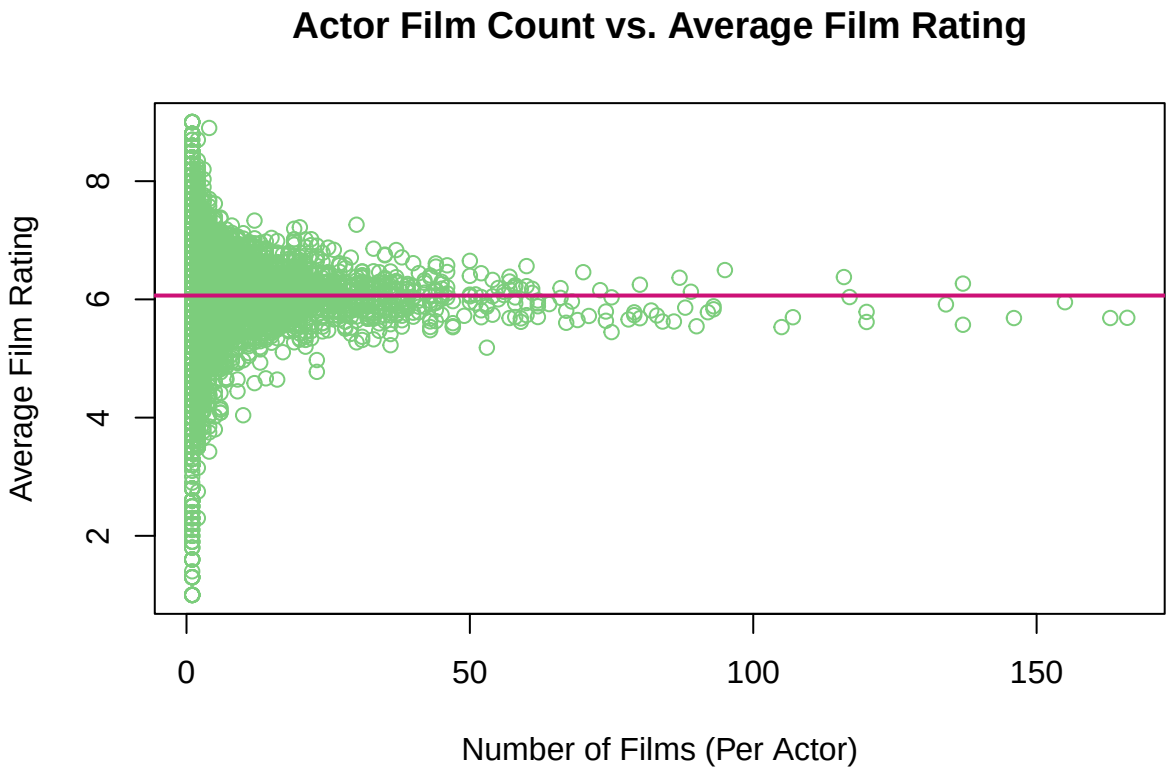
- Both of these actors were actually **famous** during the early 19th Century.
- These actors starred in 174 films.
- However, their average film ratings are quite low at 5.71092 and 5.70000.

This clearly reinforces our point earlier about sample size — comparing smaller actors who have less films with higher final average ratings against larger actors who have more films with smaller final average ratings doesn't seem to make sense.

In short, it's **hard** to identify a pattern and see if having certain actors have an affect on ratings due to data incompleteness. We **can't confuse causation with correlation**, but it's hard to **find a pattern** when some actors should have a larger associated number of films, and when those who do have lots of films end up with lower ratings.

As a final summary, here is a plot that visualizes the cases we've seen:

```
plot(actor.results$number.of.films,
     actor.results$average.ratings,
     main = "Actor Film Count vs. Average Film Rating",
     xlab = "Number of Films (Per Actor)",
     ylab = "Average Film Rating",
     col = "palegreen3")
abline(h = mean(problem3$averageRating), col = "deeppink3", lwd = 2)
```



Here, we can see that:

- Actors who, at least in our dataset, were featured in small amounts of films tended to have high average ratings.
- But as our analysis shows, these could potentially be large actors who actually featured in more films than the dataset says, OR small actors whose few films have done well.
- And the 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.

QUESTION 4:

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

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

list.of.genres <- list()                                              # Instantiates an empt
for (i in 1:unique.length){                                           # Uses a for-loop to u
  extracted.vector <- strsplit(unique(my.dataset$genres)[i], split = ",")[[1]]
  list.of.genres[[i]] <- extracted.vector                             # Here, list.of.genres
}                                                                     # ex: The character "D

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.

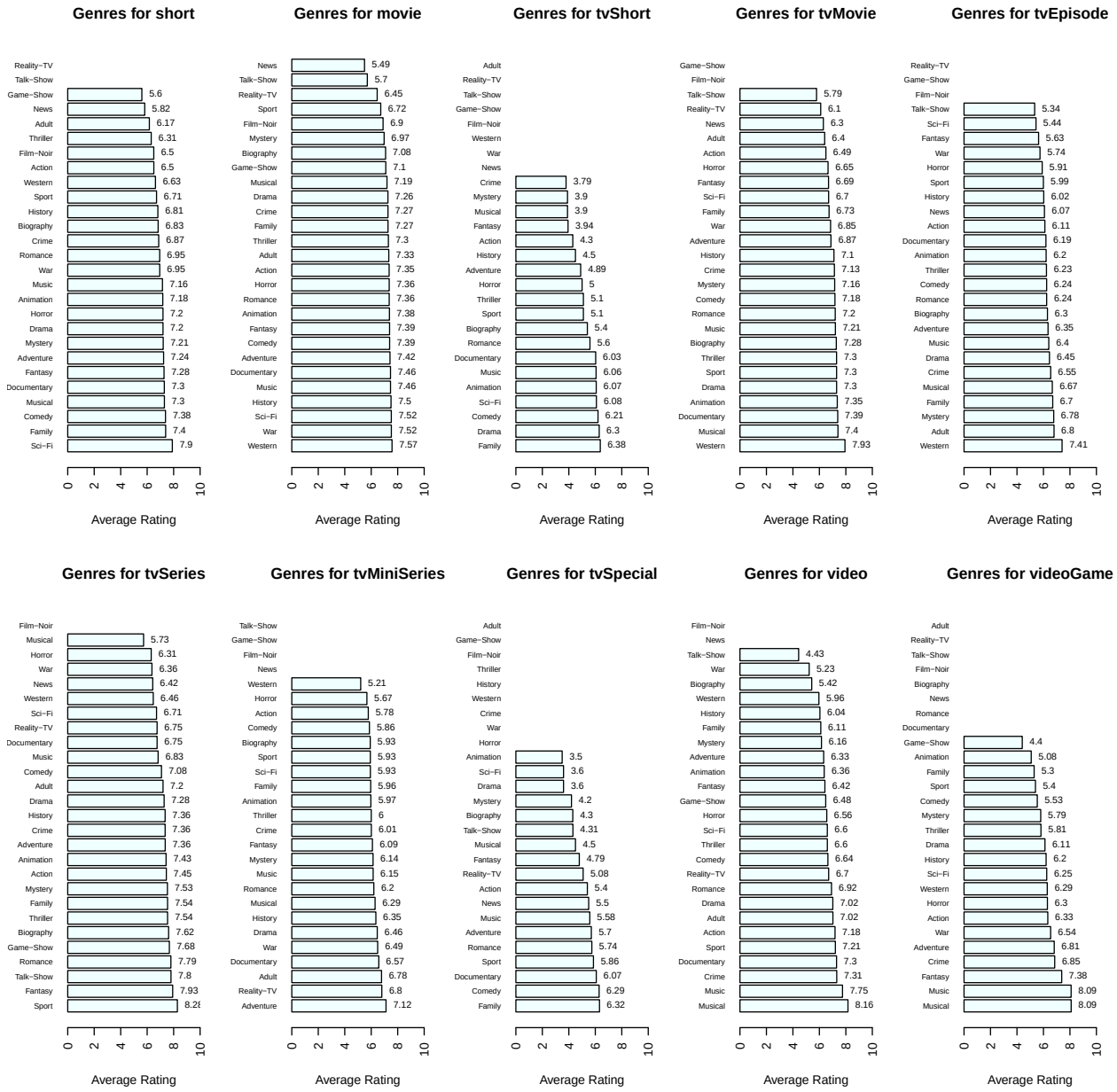
```
# Checking how many rows (i.e. films) just say "Short" and NA:
number.of.shorts <- length(which(my.dataset[,9] == "Short"))          # Number of rows in dataset
number.of.NA <- sum(is.na(my.dataset[,9]))                            # Number of rows in dataset

# Here are the proportions, where the total number of rows is 139602:
number.of.shorts / 139602                                             # Proportion of "Short" rows
number.of.NA / 139602                                                 # Proportion of missing valu
(number.of.shorts + number.of.NA) / 139602                            # Proportion of both "Short"
```

As we can see, there are 2529 rows that just say "Short" and 4442 rows that contain missing values, making up 1.8% and 3.1% of the dataset's total rows. In total, they make up around 5% 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(my.dataset$titleType)){                              # [1] For each kind o
  number.of.ratings <- c()                                            # Initialize empty ve
  average.ratings <- c()                                              # Initialize empty ve
  movie.indices <- which(my.dataset$titleType == i)

  for (j in pure.genres){                                             # Update both empty v
```

```
    indices <- which(grepl(j, my.dataset$genres[movie.indices]))
    how.many.ratings <- length(indices)

    ratings <- my.dataset$averageRating[indices]
    number.of.votes <- my.dataset$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 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.

In addition, 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. + 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.

Before, 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 get bizarre results.
- For instance, three of our box plots suggested that adult genres have been the most highly rated genres for tvMovies, tvSeries, and tvMiniSeries.

Now, what could explain the narrow range of 5-7? And do these plots imply that people believe adult films are better than musicals or talk-shows for tv shows?

Here are two observations we made:

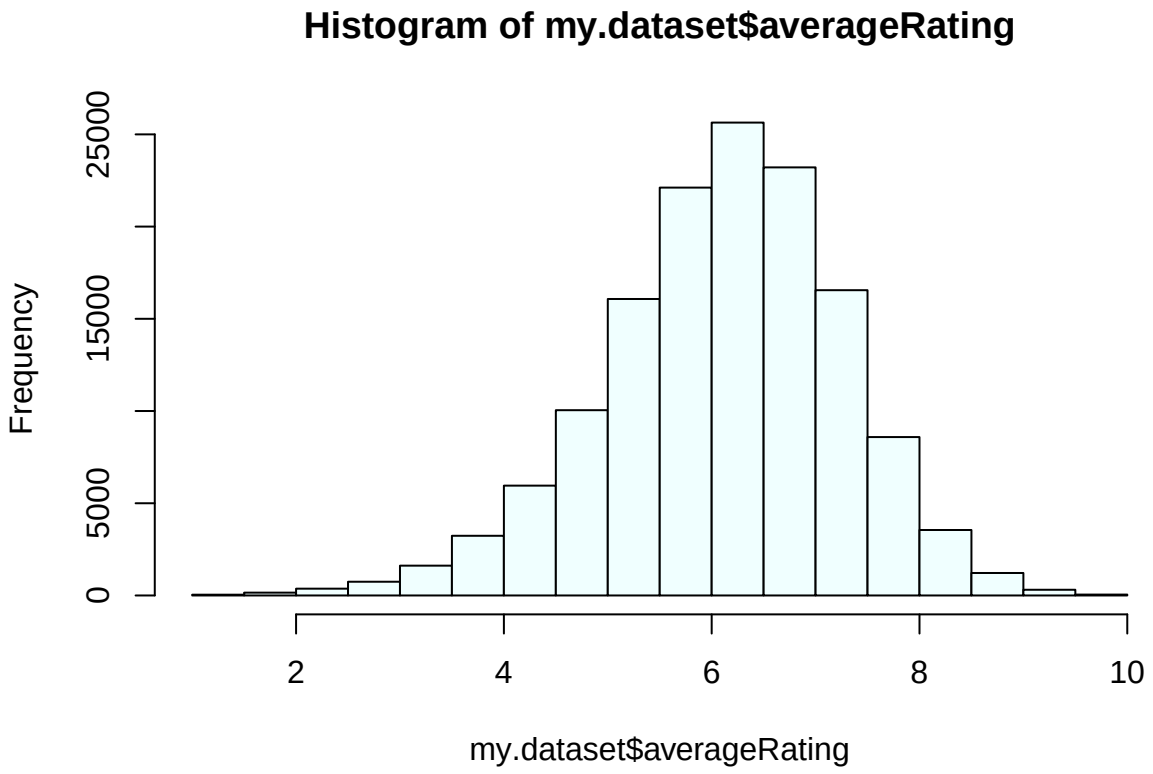
```
# [1] Check if any sample sizes (used to compute each pure genre's average) were small.
any(ratings.df$number < 50)
```

```
## [1] TRUE
```

```
min(ratings.df$number)
```

```
## [1] 0
```

```
# [2] Check the distribution of the averageRating column from our dataset.
hist(my.dataset$averageRating, col = "azure1")
```




```
summary(my.dataset$averageRating)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   5.500   6.200   6.157   6.900  10.000
```

As we can see:

- The distribution of the **averageRating** column from our dataset is already indeed narrow, with a center around 6.2 / 10 and large amounts of data between the 5.5 to 6.8 quartile range.
- 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**.

Some other external notes:

- 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.

---

[4] SUMMARY:

In short, we have learned a lot about our datasets from these four questions. However, these analyses don't cover absolutely everything we can learn from them.

If time permits, we can look further into:

1. Comparing movies over the years — movies from the early 19th Century to movies today.
2. Looking to see if there are any more open-source, updated datasets with all actors' films recorded.
3. Considering total box office revenue as part of a larger goal to better understand what factors dictate the success of a movie.
4. Looking into questions for **specific** kinds of film — investigate trends only within movies, or shorts, and so on rather than a whole dataset that deals with a blend of them all.

Overall, this project was a good exercise that blends **technical statistical** knowledge with **real circumstance**. Understanding statistical theory is important, but understanding when and how it should be used is just as, if not more, important.