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  1657261 88.6    2694621  144         NA  2694621 144.0
## Vcells 2963728 22.7    8388608   64      16384  4746168  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`:

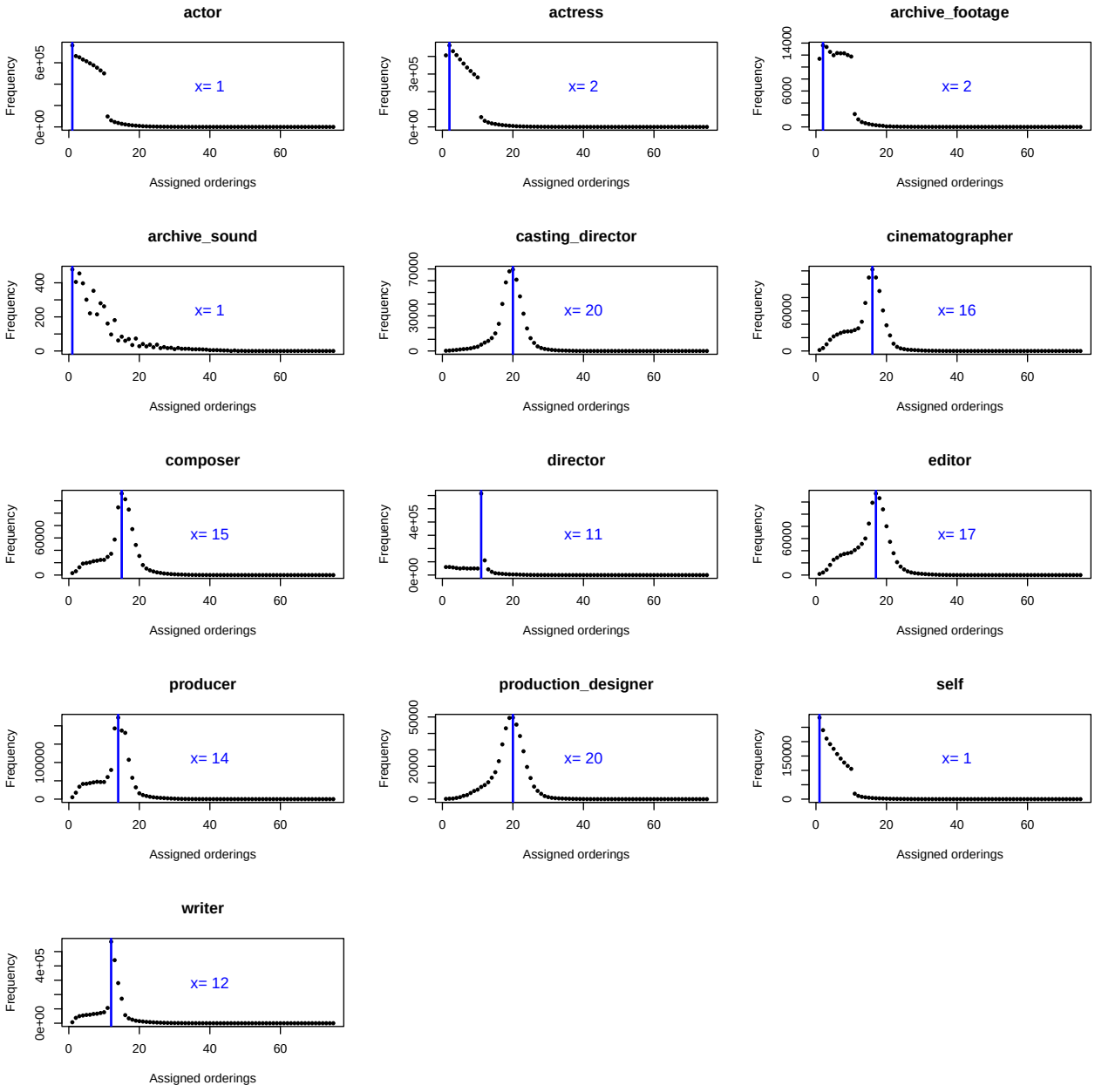
- **tconst**: There are no missing values. However, the number of unique film ID's does not equal the number of rows — this is because upon close inspection, we can see that there are duplicated ID's across several rows. This is different from the other three datasets, where there is one unique film 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)
```

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

```
# Here, it says TRUE. That is good.

# Check: Do all tconst values in basics have ratings data?
all(basics$tconst %in% ratings$tconst)
```

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

```
# Here, it says 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)
```

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

```
# Here, it says TRUE. That is good.

# Check: Do all tconst values in basics have crew data?
all(basics$tconst %in% crew$tconst)
```

```
## [1] FALSE
```

```
# Here, it says 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)
```

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

```
# Here, it says TRUE. That is good.

# Check: Do all tconst values in basics have principals data?
all(basics$tconst %in% principals$tconst)
```

```
## [1] FALSE
```

```
# Here, it says 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.

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

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

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

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

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

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

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

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

```r
# [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
```

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

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

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

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

```r
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.

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

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

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

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

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

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

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

```
## [[1]]
## [1] "[\"Self\"]" NA           NA           NA
##
## [[2]]
## [1] NA NA
##
## [[3]]
## [1] NA NA NA NA NA NA
##
## [[4]]
## [1] NA NA
##
## [[5]]
## [1] "[\"Blacksmith\"]" "[\"Assistant\"]"  NA
##
## [[6]]
## [1] NA NA NA NA NA NA NA
```

```
# [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

```r
# library(data.table)
# gc()
# setwd("~/Documents/06.sp25/04.stsci4100/stsci4100")
# full_dt <- readRDS("full_dt.rds")
```

---

[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.

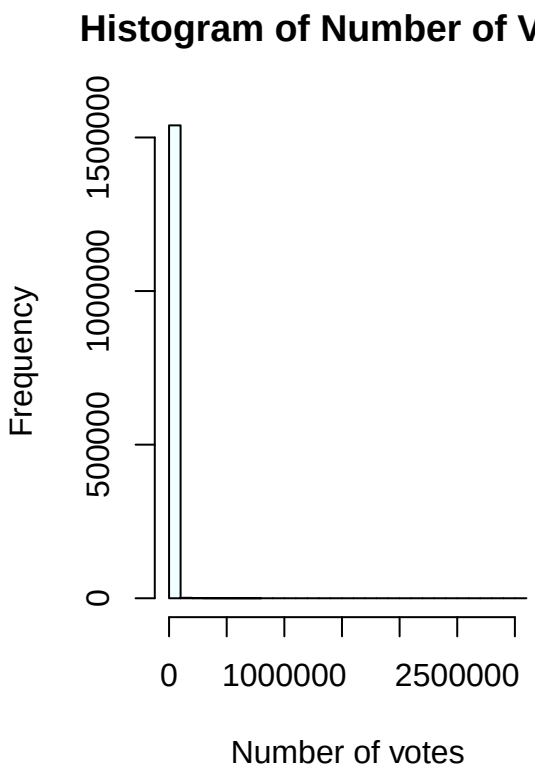
Specifically, we'll investigate whether or not films that have more votes tend to have higher or lower ratings (i.e. whether popularity is correlated with perceived quality).

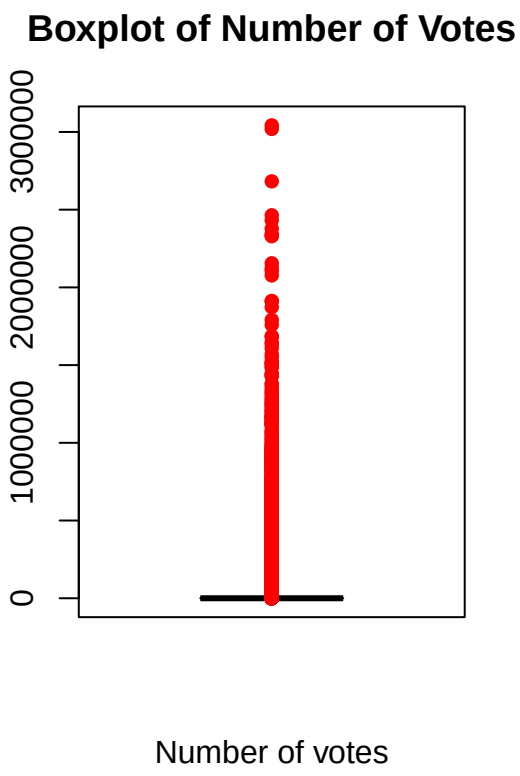
This will provide a good opening introduction, and hopefully provide some intuition, to our analysis.

```
# [1] Summary of our data's distribution of numVotes
summary(full_dt$numVotes)
```

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

```
# [2] Visualizing the distribution of the numVotes column
par(mfrow = c(1, 2))
hist(full_dt$numVotes,
     xlab = "Number of votes",
     ylab = "Frequency",
     main = "Histogram of Number of Votes",
     col = "azure1")
boxplot(full_dt$numVotes,
        xlab = "Number of votes",
        main = "Boxplot of Number of Votes",
        col = "black",
        pars = list(outpch = 16, outcol = "red"))
```





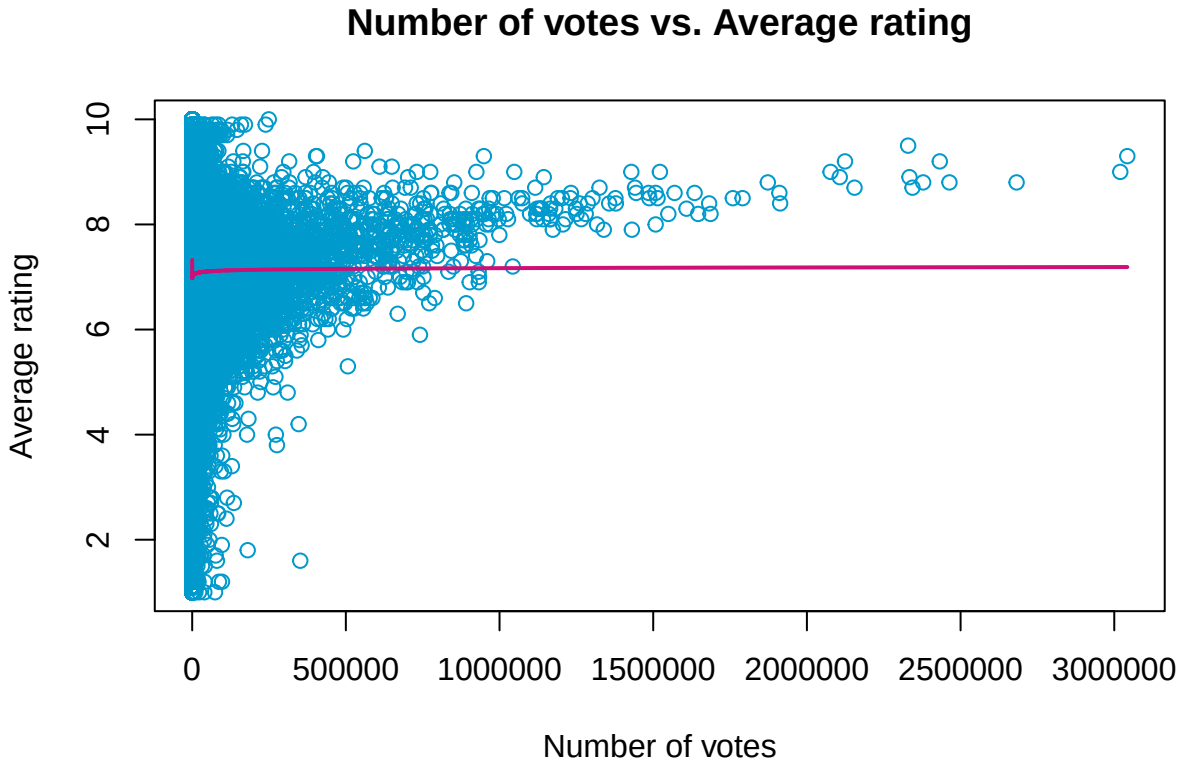
Outliers are colored red, and we see that the box plot deems most of the visible points as outliers. The information so far suggests a heavy right skew.

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.

```
# [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 plot this.

TODO:

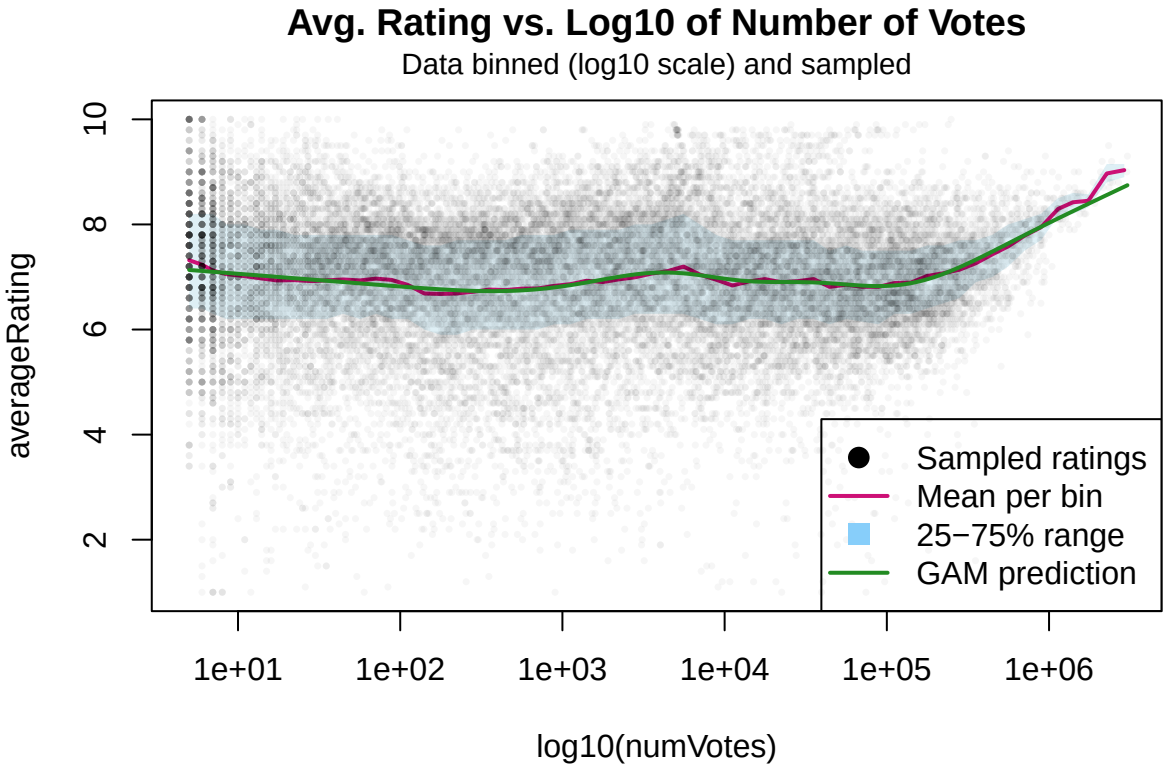
```
# [7] Fit GAM
gam_model <- gam(averageRating ~ s(x_log), data = q1_dt)
# Predict
log_x_pred <- seq(log10(min(q1_dt$numVotes)), log10(max(q1_dt$numVotes)),
                  length.out = 500)
x_pred <- 10^log_x_pred
y_pred <- predict(gam_model, newdata = data.frame(x_log = log_x_pred))
```

TODO:

```
# [8]
# Plot
plot(NA,
     xlim = range(q1_stats$x_bin),
     ylim = range(q1_dt_sampled$averageRating, na.rm = TRUE),
     log = "x",
     xlab = "log10(numVotes)",
     ylab = "averageRating",
     main = "Avg. Rating vs. Log10 of Number of Votes"
)
mtext("Data binned (log10 scale) and sampled", side = 3, line = 0.5, cex = 0.9)

# 25-75% IQR Shading
polygon(c(q1_stats$x_bin, rev(q1_stats$x_bin)),
        c(q1_stats$p25, rev(q1_stats$p75)),
        col = adjustcolor("lightblue", alpha.f = 0.4), border = NA)
# Mean Line
lines(q1_stats$x_bin, q1_stats$mean_rating, col = "deeppink3", lwd = 2)
# Sampled points
points(q1_dt_sampled$numVotes, q1_dt_sampled$averageRating,
       pch = 16, col = rgb(0, 0, 0, 10, maxColorValue = 255), cex = 0.5)
# Plot GAM prediction line
lines(x_pred, y_pred, col = "forestgreen", lwd = 2, lty = 1)

# Legend
legend("bottomright",
       legend = c("Sampled ratings", "Mean per bin", "25-75% range",
                  "GAM prediction"),
       col = c("black", "deeppink3", rgb(135, 206, 250, maxColorValue = 255),
               "forestgreen"),
       pch = c(16, NA, 15, NA), lwd = c(NA, 2, NA, 2),
       lty = c(NA, 1, NA, 1), pt.cex = 1.5)
```

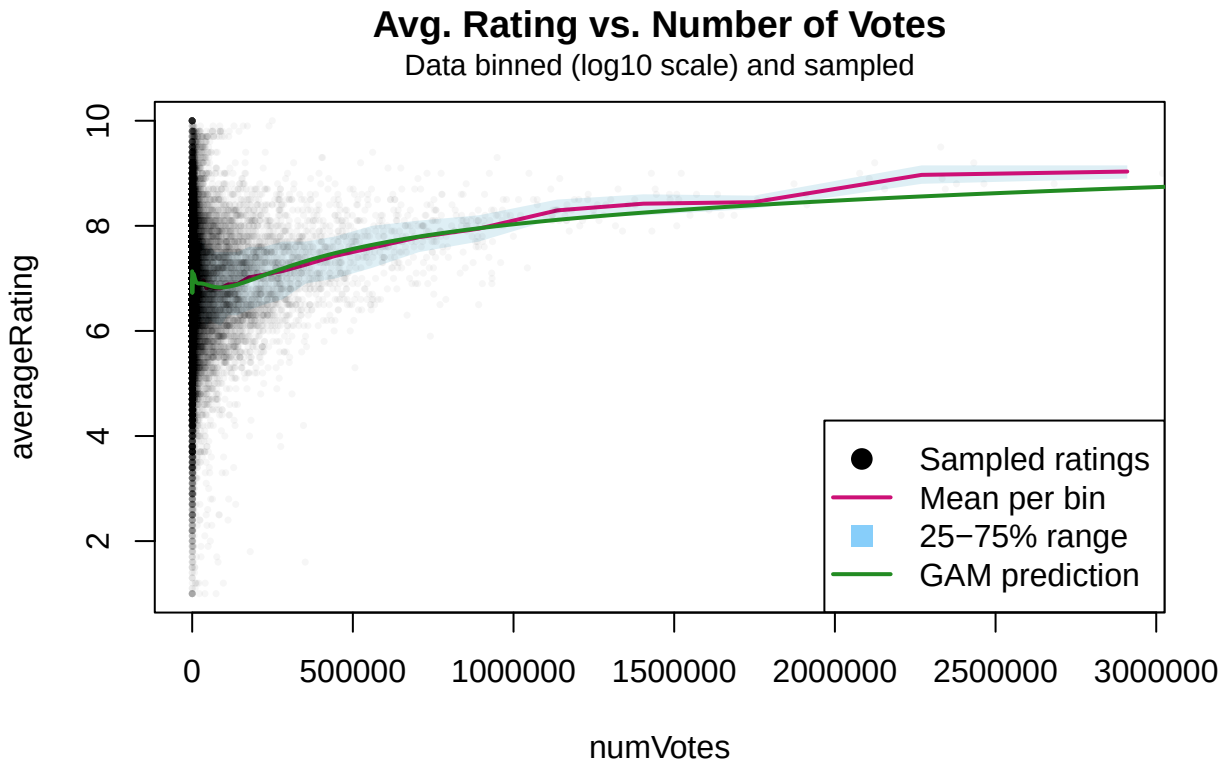


TODO:

```r
# [9]
# Plot
plot(NA,
     xlim = range(q1_stats$x_bin),
     ylim = range(q1_dt_sampled$averageRating, na.rm = TRUE),
     xlab = "numVotes",
     ylab = "averageRating",
     main = "Avg. Rating vs. Number of Votes"
)
mtext("Data binned (log10 scale) and sampled", side = 3, line = 0.5, cex = 0.9)

# 25-75% IQR Shading
polygon(c(q1_stats$x_bin, rev(q1_stats$x_bin)),
        c(q1_stats$p25, rev(q1_stats$p75)),
        col = adjustcolor("lightblue", alpha.f = 0.4), border = NA)
# Mean Line
lines(q1_stats$x_bin, q1_stats$mean_rating, col = "deeppink3", lwd = 2)
# Sampled points
points(q1_dt_sampled$numVotes, q1_dt_sampled$averageRating,
       pch = 16, col = rgb(0, 0, 0, 10, maxColorValue = 255), cex = 0.5)
# Plot GAM prediction line
lines(x_pred, y_pred, col = "forestgreen", lwd = 2, lty = 1)
# Legend
legend("bottomright",
       legend = c("Sampled ratings", "Mean per bin", "25-75% range",
                  "GAM prediction"),
       col = c("black", "deeppink3", rgb(135, 206, 250, maxColorValue = 255),
               "forestgreen"),
```

```
       pch = c(16, NA, 15, NA), lwd = c(NA, 2, NA, 2),
       lty = c(NA, 1, NA, 1), pt.cex = 1.5)
```



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

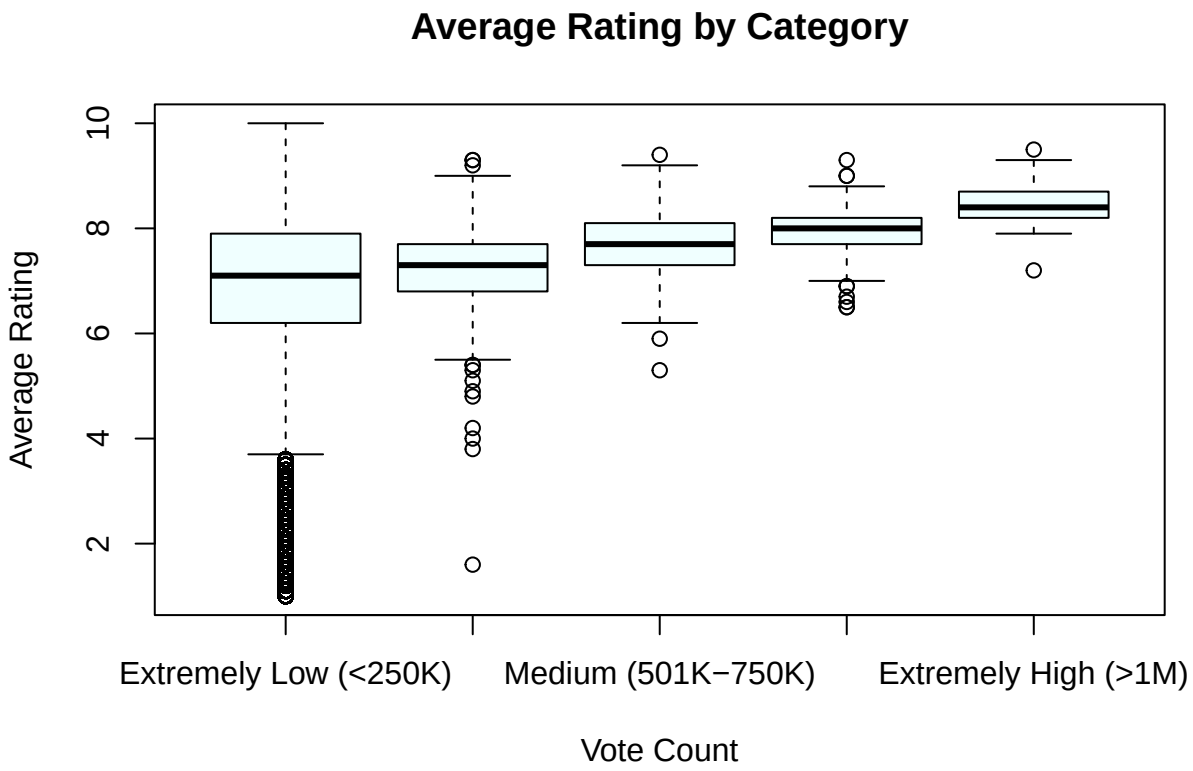
```
q1_dt$numVote_category <- NA # Create a new column in q1_dt

# Category 1: Extremely Low
q1_dt$numVote_category[q1_dt$numVotes <= 250000] <- "Extremely Low (<250K)"
# Category 2: Low
q1_dt$numVote_category[q1_dt$numVotes > 250000
                       & q1_dt$numVotes <= 500000] <- "Low (251K-500K)"
# Category 3: Medium
q1_dt$numVote_category[q1_dt$numVotes > 500000
                       & q1_dt$numVotes <= 750000] <- "Medium (501K-750K)"
# Category 4: High
q1_dt$numVote_category[q1_dt$numVotes > 750000
                       & q1_dt$numVotes <= 1000000] <- "High (751K-1M)"
# Category 5: Extremely High
q1_dt$numVote_category[q1_dt$numVotes > 1000000] <- "Extremely High (>1M)"

q1_dt$numVote_category <- factor(q1_dt$numVote_category,
                                 levels = c("Extremely Low (<250K)",
                                            "Low (251K-500K)",
                                            "Medium (501K-750K)",
                                            "High (751K-1M)",
                                            "Extremely High (>1M)"))

boxplot(averageRating ~ numVote_category,
        data = q1_dt,
```

```
        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.
full_dt$haswriter <- as.numeric(!is.na(full_dt$writers))                    #  Binary column: 1 if there'

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

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

```
mean(full_dt$haswriter)
```

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

```
# 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(full_dt$averageRating)
```

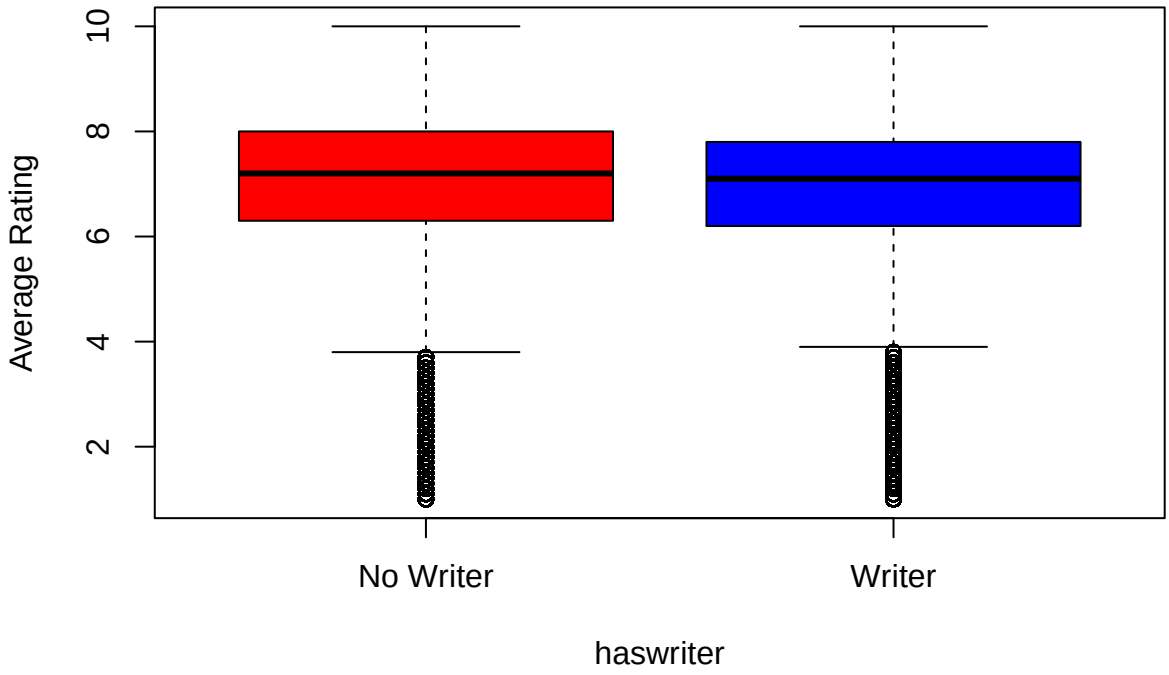
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00    6.20    7.10    6.95    7.90   10.00
```

```
mean(full_dt$averageRating)
```

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

```
# 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 = full_dt,
        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 = full_dt)
```

```
##
##  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 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.