MovieLens Capstone Project

Professional Certificate PH125.9x

Robert Gravelle

2023-12-05

Contents

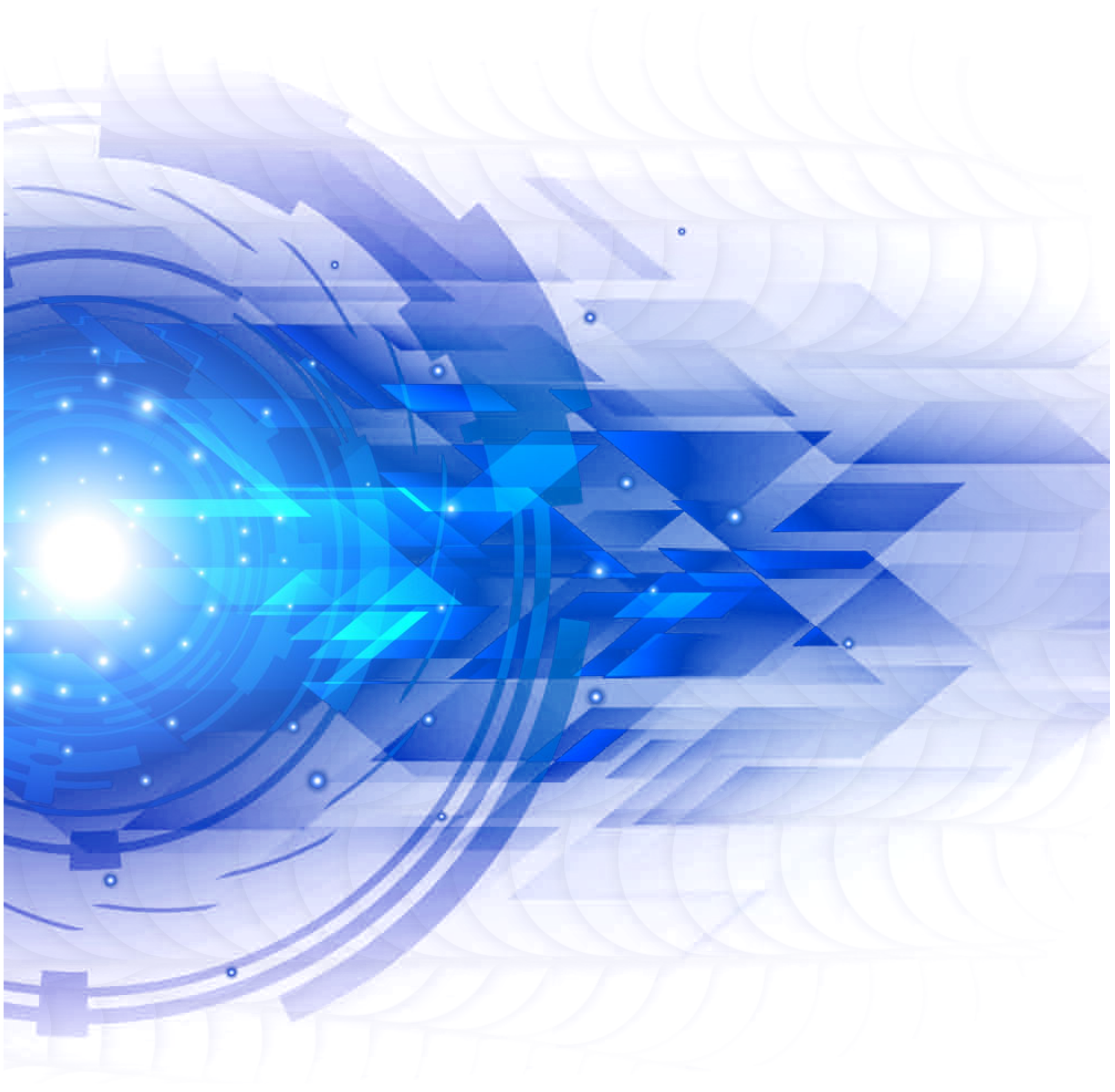


Figure 1: Background Image

MovieLens

Introduction

This report delves into the analysis of MovieLens data, encompassing the training of a recommendation model and assessing its performance. Initially, we embark on exploring and potentially refining the dataset, subjecting it to inspection for a comprehensive evaluation of potential training strategies. Subsequently, the focus shifts to constructing a machine learning model dedicated to suggesting movies to users.

Dataset Information

The 10M MovieLens dataset is a rich repository of cinematic insights, containing 10 million user-generated ratings across a spectrum of 10,000 movies. This expansive collection captures the diverse perspectives of 72,000 randomly selected users, offering a nuanced glimpse into the tapestry of audience preferences. This dataset serves as a valuable foundation for in-depth analyses and the development of robust recommendation models within the realm of movie suggestions.

Initial Project Configuration

In this project I initiated the loading process of the MovieLens 10M dataset, meticulously partitioning it into two subsets: *edx* and *final_holdout_test*. These subsets, constituting 10% of the entire MovieLens data, serve a specific purpose—solely reserved for conclusive validation towards the project's conclusion. In preparing this dataset, key elements used are: userId, movieId, rating, timestamp, title, and genre. Together, they form the foundational components of this dynamic dataset, prepared for detailed exploration and validation..

Goal

This dataset serves as a portal for exploration, providing valuable insights into the intricate process of crafting an effective recommendation algorithm. The quest for a machine learning algorithm led me to a robust and meticulously developed algorithm to rigorously test against the dataset of the final_holdout_test set. The meticulous process ensures the algorithm's robustness and efficacy in pursuing an efficient program of movie recommendations.

Capstone course Goals

The Capstone requirements stated four things the report needed to have: introduction/overview/executive summary which required the report to describe the dataset and summarize the goals and steps of the project. (Introduction) methods/analysis where the report is to express the techniques and the data cleaning, exploratin and visualizations and the chosen modeling approach. (Analysis) results section to present the modeling results and limitations and what would be done differently.(Results) conclusion section to give a brief summary of the report and the limitations and future changes to the report or methods. (Conclusion)

Summary

The analysis revealed interesting properties and correlations of the various features. Among other things, older films tended to be rated higher than newer ones, and some genres were generally rated slightly higher or lower. Above all, however, the films and the users play a decisive role in developing a model. Unfortunately, only an RMSE of about 0.879 was possible with this method. For further interesting model trainings neither the time nor the available computing power was sufficient, for example other training approaches like KNN or Decision Tree could be tried.

Additional Notes

One thing that I always do when I program, compose reports, or anything that gets published on the web I always perform a plagiarism check. One of the websites that I use is https://www.check-plagiarism.com/.

The reults of the scan report is PLAGIARISM SCAN REPORT Date December 06, 2023 Unique Content 100% Word Count 5248 Plagiarized Content 0%

Code provided by EdX in the Capstone directons for the project

```r
# Create edx and final_holdout_test sets

# Note: this process could take a couple of minutes

if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http.cran.us.r-project.org")

library(tidyverse)
library(caret)

# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

options(timeout = 250)

dl <- "ml-10M100K.zip"
if(!file.exists(dl))
  download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings_file <- "ml-10M100K/ratings.dat"
if(!file.exists(ratings_file))
  unzip(dl, ratings_file)

movies_file <- "ml-10M100K/movies.dat"
if(!file.exists(movies_file))
  unzip(dl, movies_file)

ratings <- as.data.frame(str_split(read_lines(ratings_file), fixed("::"), simplify = TRUE),
                         stringsAsFactors = FALSE)
colnames(ratings) <- c("userId", "movieId", "rating", "timestamp")
ratings <- ratings %>%
  mutate(userId = as.integer(userId),
         movieId = as.integer(movieId),
         rating = as.numeric(rating),
         timestamp = as.integer(timestamp))

movies <- as.data.frame(str_split(read_lines(movies_file), fixed("::"), simplify = TRUE),
                        stringsAsFactors = FALSE)
colnames(movies) <- c("movieId", "title", "genres")
movies <- movies %>%
  mutate(movieId = as.integer(movieId))

movielens <- left_join(ratings, movies, by = "movieId")

# Final hold-out test set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```r
# set.seed(1) # if using R 3.5 or earlier
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

# Make sure userId and movieId in final hold-out test set are also in edx set
final_holdout_test <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from final hold-out test set back into edx set
removed <- anti_join(temp, final_holdout_test)
```

```
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, genres)`
```

#

Analysis

Data Inspection and preprocessing

Let's begin by examining the structure and contents of the Edx dataset in closer detail.

	userId	movieId	rating	timestamp	title	genres
1	1	122	5	838985046	Boomerang (1992)	Comedy|Romance
2	1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller
4	1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller
5	1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi
6	1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi

This dataset comprises various columns, each providing crucial information:

- **userId**: Uniquely identifies the user who rated the movie.
- **movieId**: Serves as a distinctive identifier for the rated movie.
- **rating**: Indicates the user-assigned rating for the movie, with the rating scale yet to be determined.
- **timestamp**: Presents the timestamp in UNIX format.
- **title**: Encompasses the movie's title, inclusive of its release year.
- **genres**: Enumerates a set of genres linked to the movie, with multiple genres separated by '|'.

Within the realms of the edx dataset, a multitude of captivating explorations await:

1. **Exceptional Cinematic Appeal:**
   - Identify movies that soar above the average rating, potentially fueled by compelling narratives or impeccable production.
2. **Genre Elevation:**
   - Scrutinize variations in ratings across different genres or their combinations, unveiling those that consistently receive higher acclaim.
3. **User Rating Dynamics:**
   - Delve into the potential bias within user ratings, discerning whether there's a prevalent inclination towards loftier or lower ratings overall.
4. **Genre Affinity Impact:**
   - Investigate how individual user genre preferences influence their ratings. For instance, a user inclined towards Horror and Action may tend to rate movies outside these genres lower.
5. **Temporal Rating Odyssey:**
   - Unearth the correlation between user ratings and movie release years, potentially revealing evolving patterns or trends over time.
6. **Popularity Quotient:**
   - Navigate the landscape of user ratings concerning the popularity spectrum, distinguishing between indie and blockbuster films. This exploration might unveil distinct rating patterns associated with movies of varying popularity levels.

Before delving into our analysis, it's imperative to meticulously inspect the dataset for any missing values or anomalies in unique data names that might hint at outlier information.

The dataset *edx* contains 0 missing values.

Lets discover the genres in the dataset

Genres
Comedy
Romance
Action
Crime
Thriller
Drama
Sci-Fi
Adventure
Children
Fantasy
War
Animation
Musical
Western
Mystery
Film-Noir
Horror
Documentary
IMAX
(no genres listed)

We notice a unusual genre named in the list of *(no genres listed)*.

	userId	movieId	rating	timestamp	title	genres
1025055	7701	8606	5.0	1190806786	Pull My Daisy (1958)	(no genres listed)
1453345	10680	8606	4.5	1171170472	Pull My Daisy (1958)	(no genres listed)
4066835	29097	8606	2.0	1089648625	Pull My Daisy (1958)	(no genres listed)
6456906	46142	8606	3.5	1226518191	Pull My Daisy (1958)	(no genres listed)
8046611	57696	8606	4.5	1230588636	Pull My Daisy (1958)	(no genres listed)
8988750	64411	8606	3.5	1096732843	Pull My Daisy (1958)	(no genres listed)
9404670	67385	8606	2.5	1188277325	Pull My Daisy (1958)	(no genres listed)

Upon delving deeper, it came to light that a solitary movie, "Pull My Daisy," lacks a designated genre. As per IMDB, this 1958 film falls under the category of "Short." To refine our dataset, let's extract and categorize the listed genres, allocating the initial three genres of each movie into distinct columns.

In discovery we see there are 20 unique genres in the EdX dataset.

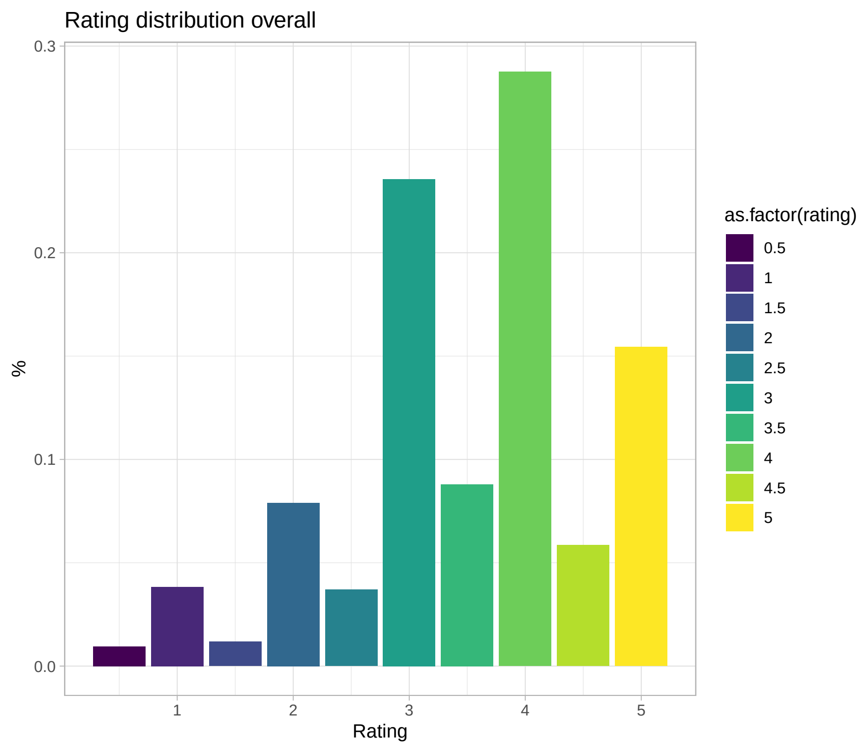
Subsequently, we transform the UNIX timestamps into a more user-friendly date/time format. For added convenience, we extract the year when each movie was rated. Furthermore, we isolate the release year embedded in the movie titles, setting the stage for potential future exploration.

Now some columns are added for further inspection, lets see the table again.

	userId	movieId	rating	timestamp	title	genres	main_genre	side1_genre	side2_genre	date	yearrated	rele
1	1	122	5	838985046	Boomerang (1992)	Comedy|Romance	Comedy	Romance	NA	1996-08-02 11:24:06	1996	
2	1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller	Action	Crime	Thriller	1996-08-02 10:58:45	1996	
4	1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller	Action	Drama	Sci-Fi	1996-08-02 10:57:01	1996	
5	1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi	Action	Adventure	Sci-Fi	1996-08-02 10:56:32	1996	
6	1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi	Action	Adventure	Drama	1996-08-02 10:56:32	1996	
7	1	355	5	838984474	Flintstones, The (1994)	Children|Comedy|Fantasy	Children	Comedy	Fantasy	1996-08-02 11:14:34	1996	

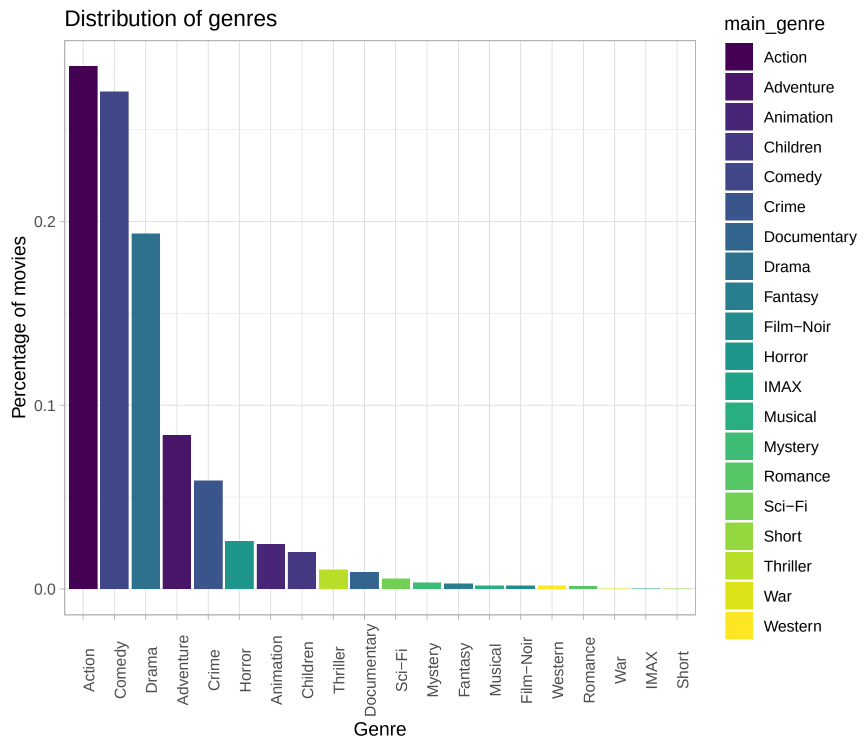
Distributions and Plots

\begin}center} We need to see the distribution of Ratings \end{center}

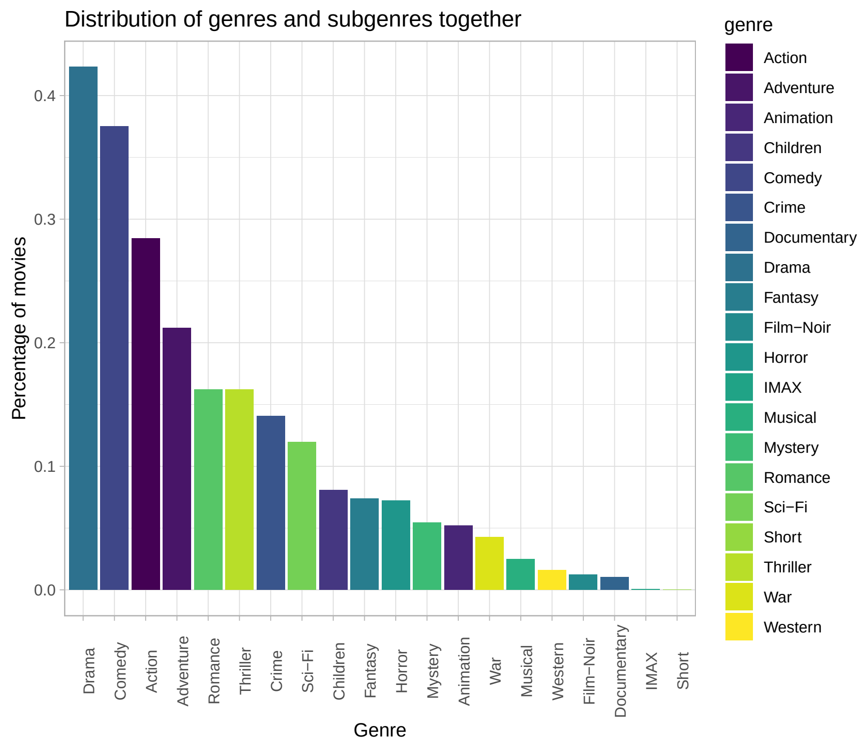


The majority of ratings are whole numbers; Additionally, the distribution of half-point ratings mirrors the pattern and distribution of the observed ratings in whole-number ratings.

Now lets view the the distribution of the genres.

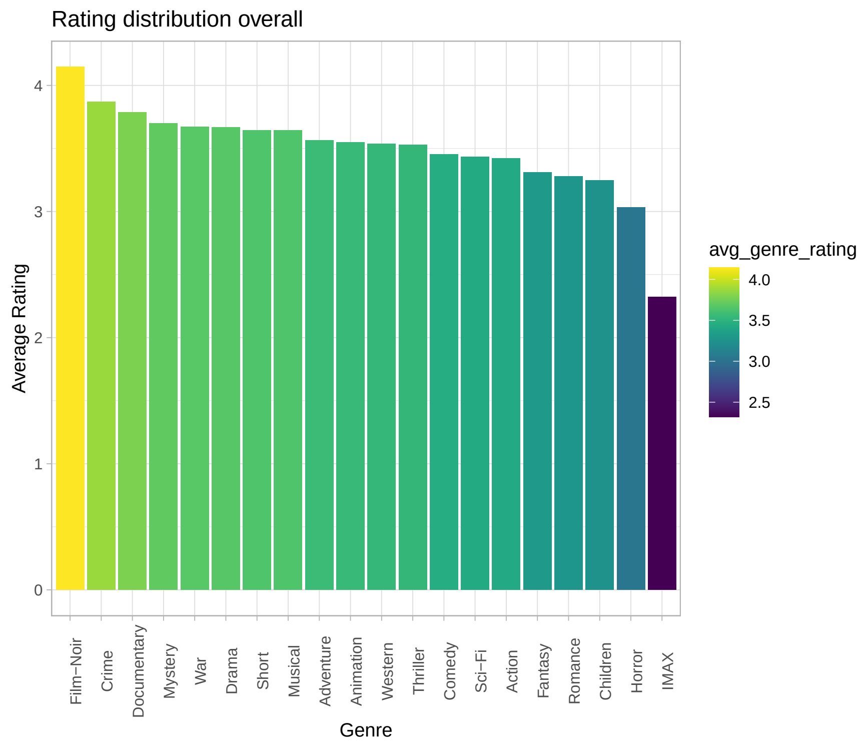


The prevalent genres predominantly include Action, Comedy, and Drama. Yet, delving into the realm of sub-genres unveils a more nuanced distribution. While Drama, Comedy, and Action still claim the top spots, their positions undergo subtle shifts. Consequently, Drama emerges as a frequently paired side genre, harmonizing seamlessly with genres such as Romance, Thriller, and others.

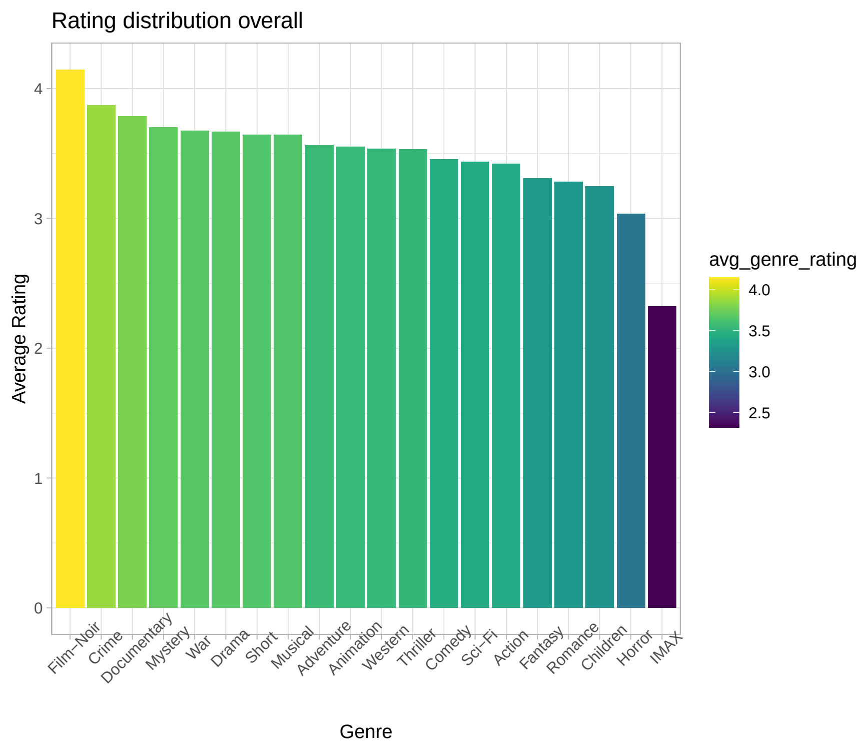


Genres

Lets now see the Genres graphed against the average rating.

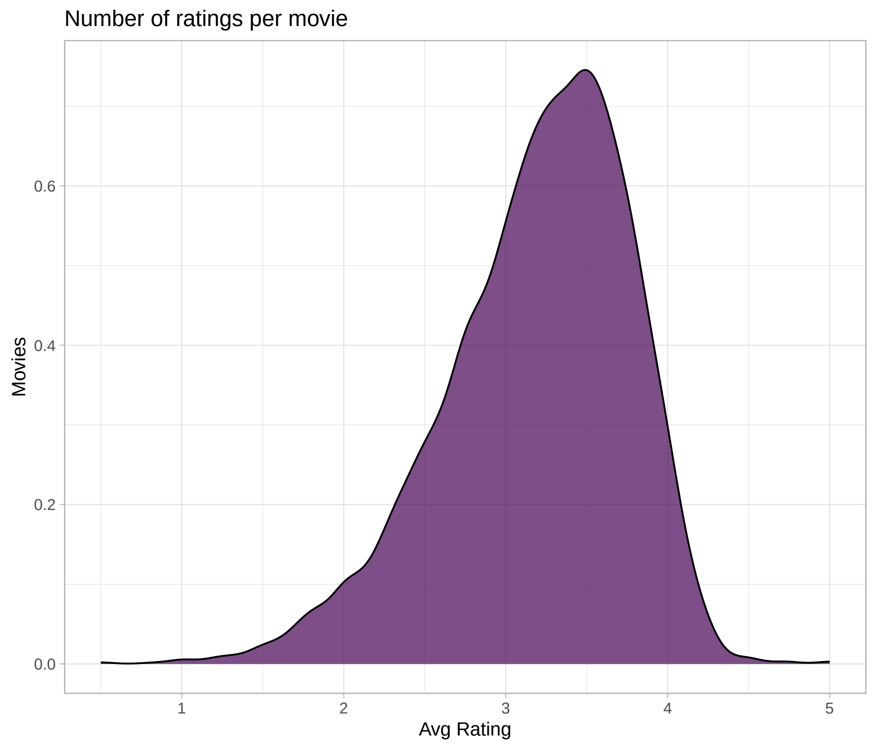


The ratings for various movie genres uncover a predilection for "intellectual" genres such as Film-Noir, Crime, and Drama. These genres consistently receive higher ratings compared to genres typically associated with entertainment, such as Action, Fantasy, and Horror. This pattern persists when examining the average ratings of genre combinations, particularly those with more than 50,000 ratings.



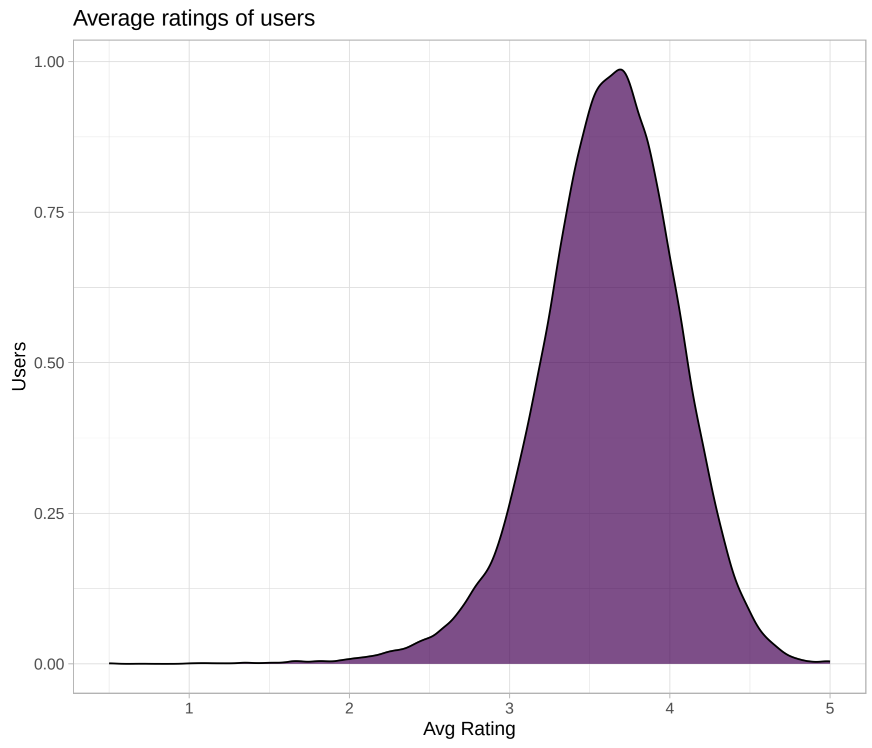
Considering genre combinations, a similar pattern emerges, although some combinations featuring entertainment genres, particularly those involving Action (e.g., Action|Drama|War, Action|Adventure), tend to rank higher on the list.

Rating distribution of movies



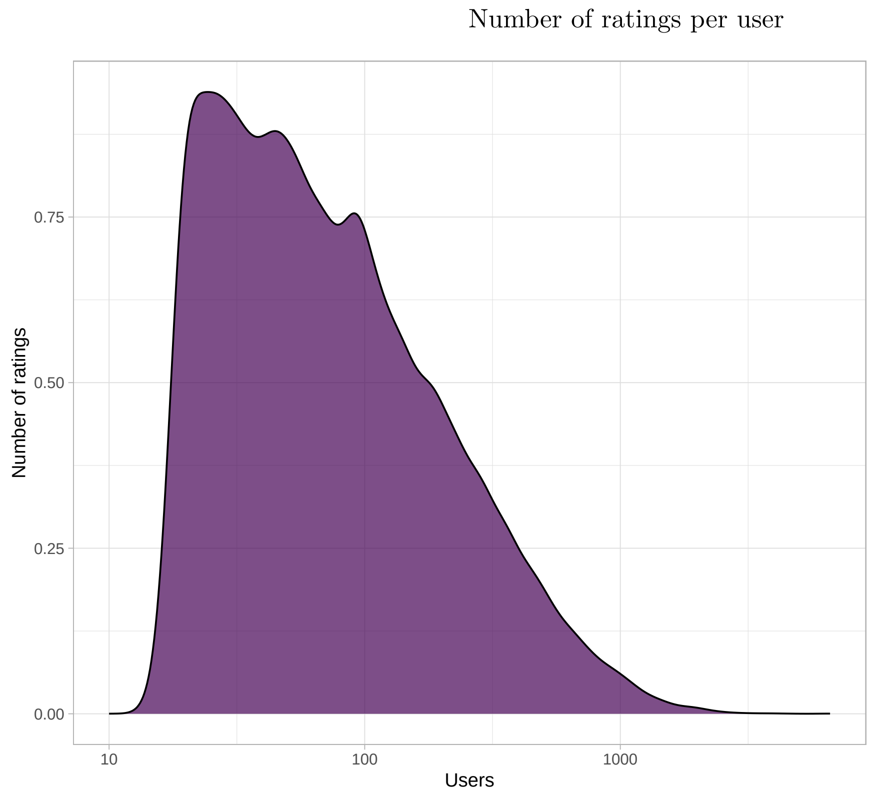
Only a scant number of movies achieve an average rating surpassing approximately 4.2. The majority of films fall within the range of 2 and 4.5 in terms of average ratings.

Average rating distribution of users



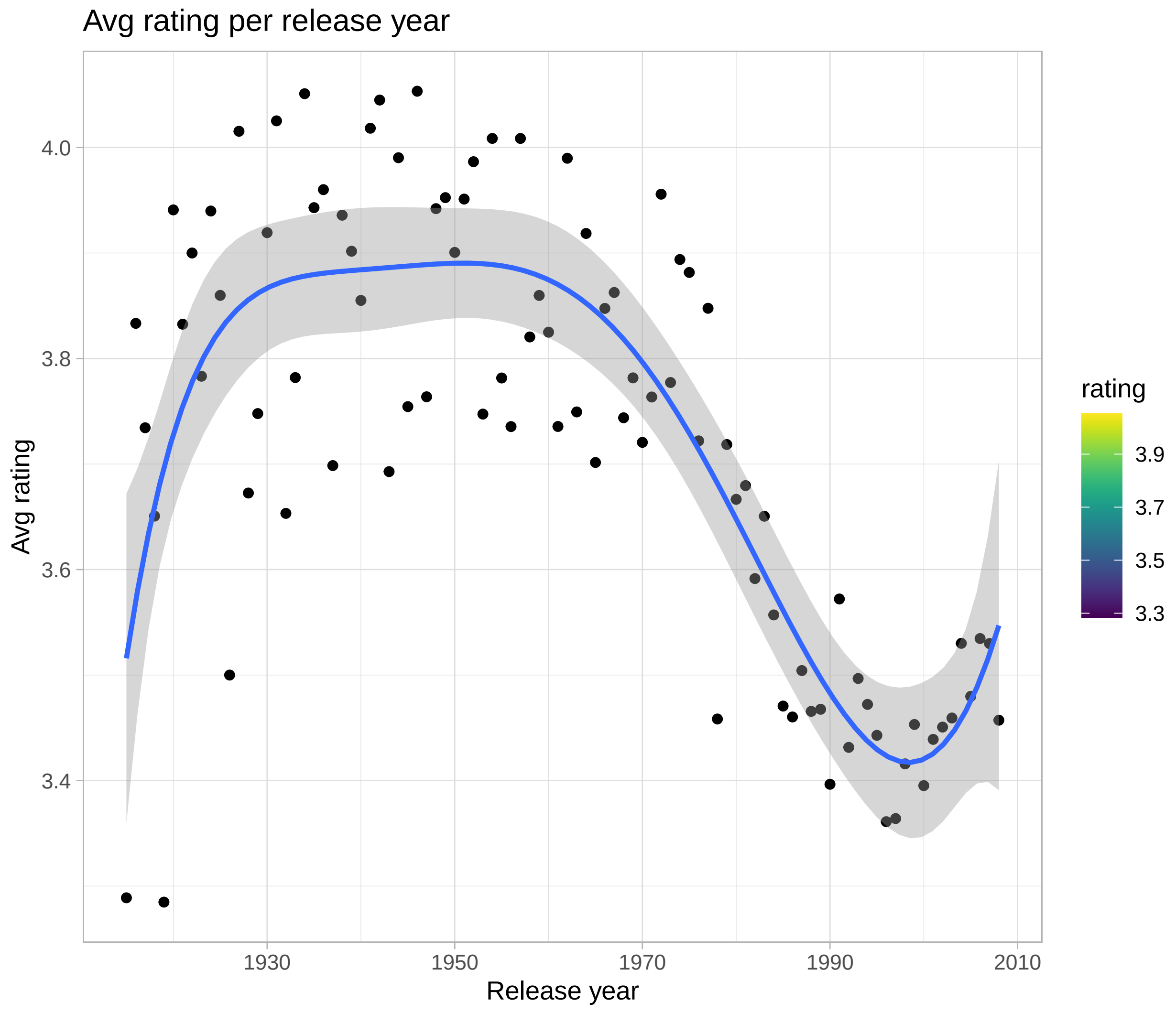
User Ratings

The majority of users tend to rate movies within the range of 3.5 to 4.2, surpassing the average ratings observed in the overall movie rating distribution.



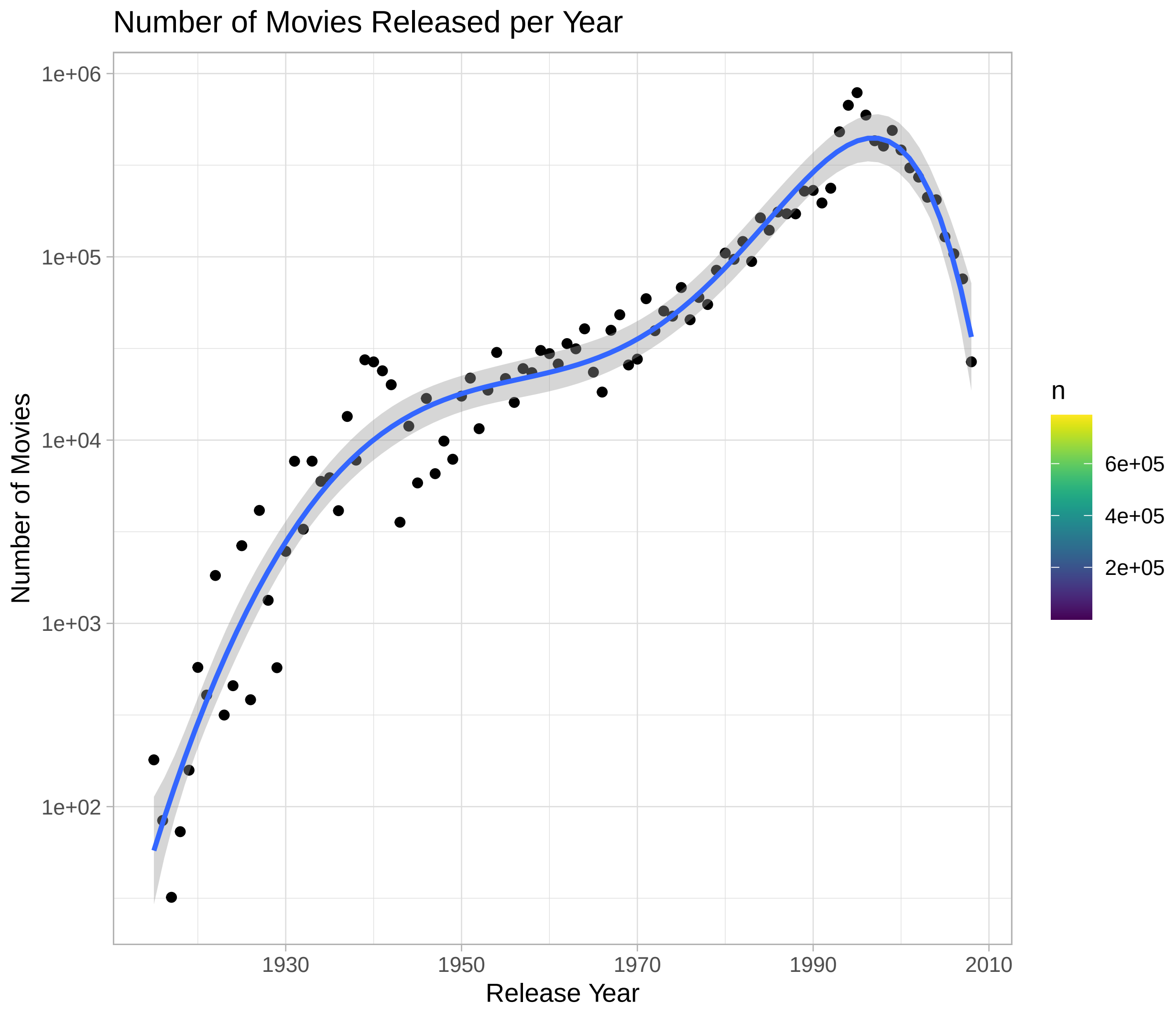
Trendline Graphs for Ratingsand Release Dates

Trend lines average rating per release year



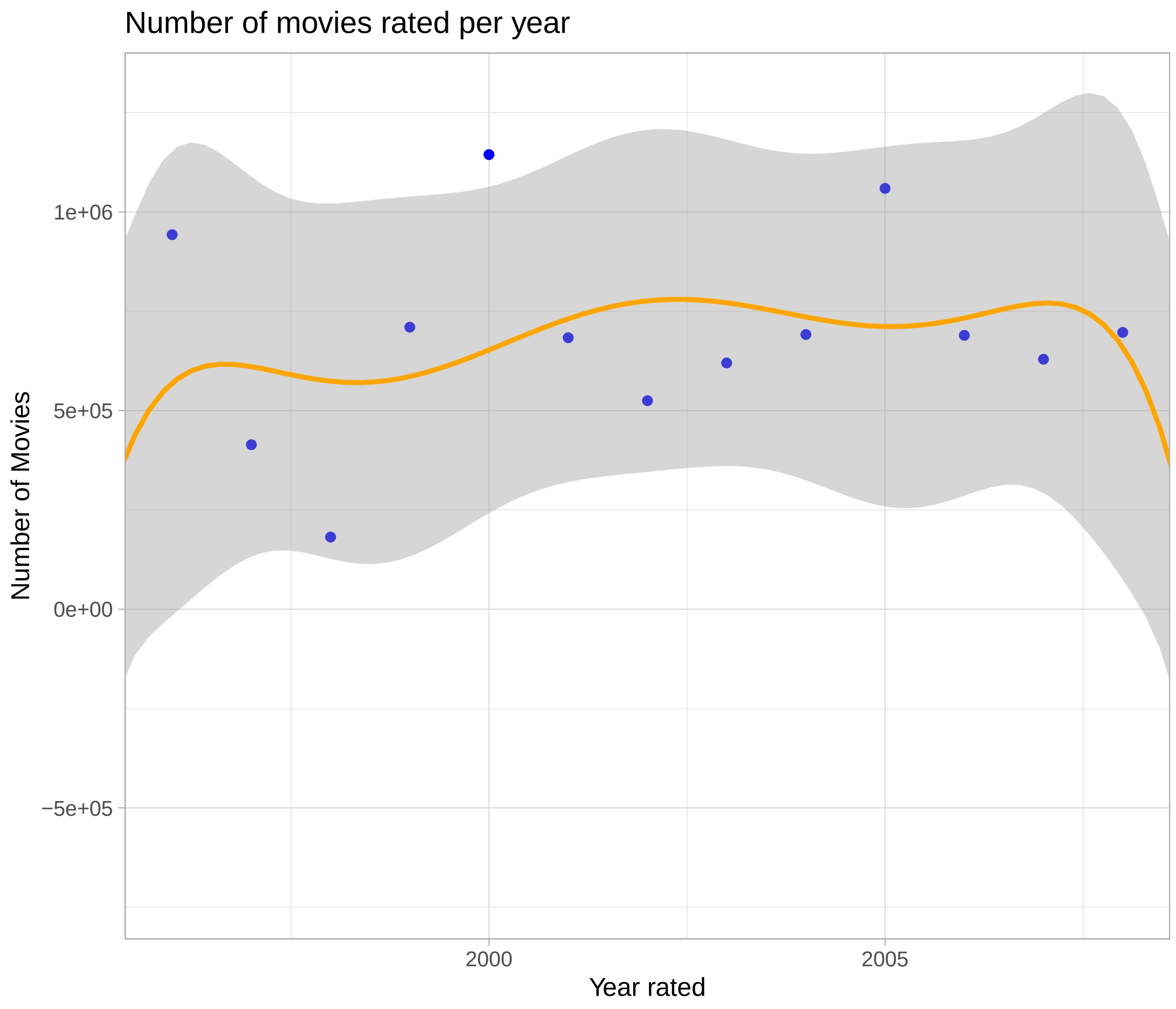
Films from the pre-1970 era tend to receive higher ratings compared to those released in the last two decades. This phenomenon is a well-known effect, where only the exceptional works withstand the test of time.

Number of released movies per year



The landscape of movie releases followed a relatively steady trajectory before 1970, underwent a substantial increase around 1990, and nearly reached an explosive surge in the subsequent years. This surge can be attributed to various factors such as technological advancements, changes in production dynamics, and shifts in the market landscape, including the widespread availability of movie theaters, the popularity of movie rentals, and the influence of television. However, starting from the late 1990s, there was a notable downturn in the number of movie releases, marking a reversal from the rapid expansion observed a decade earlier.

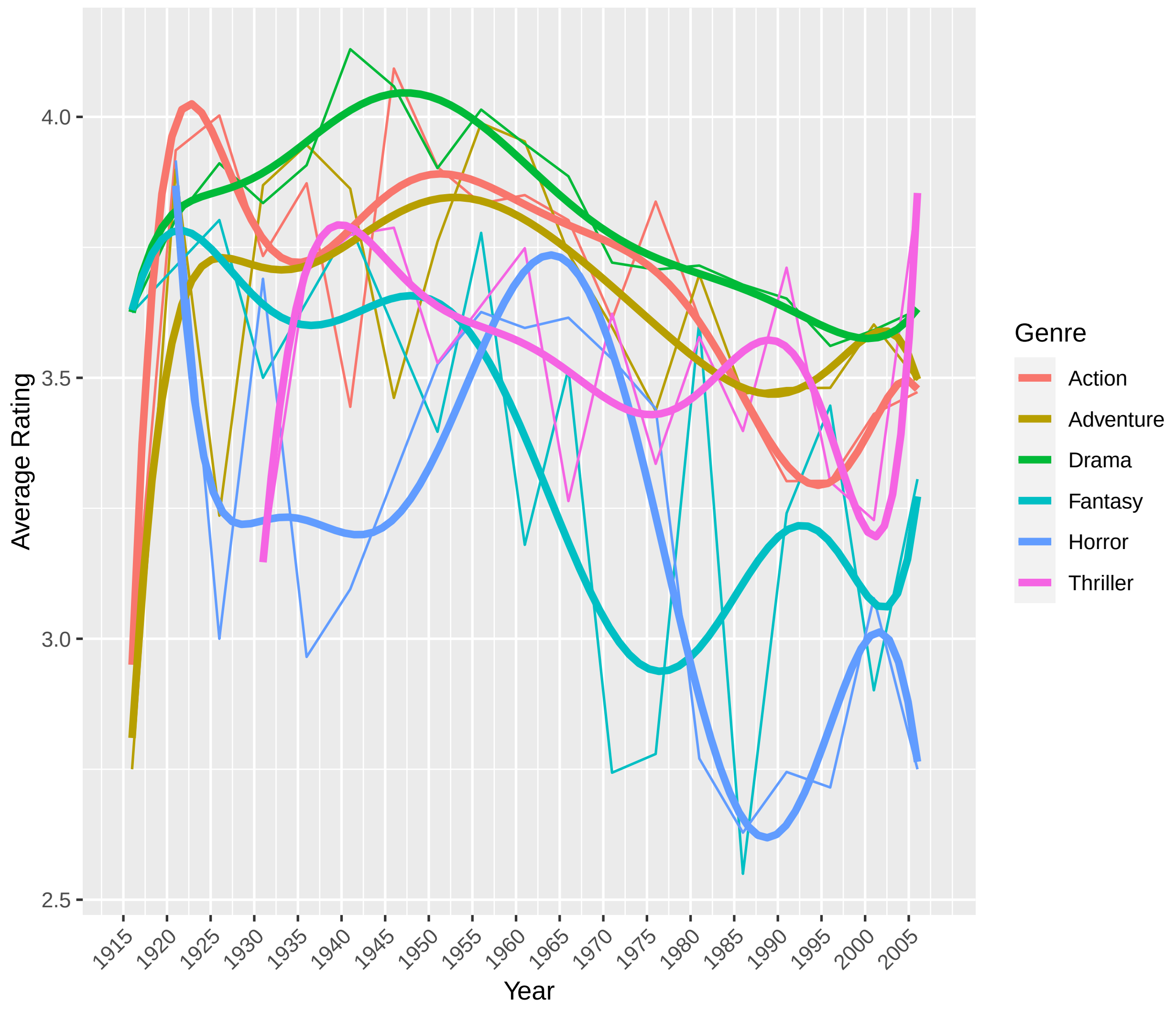
Ratings distribution over the years, focusing on the period from 1996 to 2008, with data outside this range registered as zero.



Movie ratings per year are stable with some outlier

Get the average rating of movie per genre but in 5 year bins.

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



Certain genres maintain a steady average rating throughout the years, while others, such as Horror or Fantasy, exhibit more significant fluctuations.

Modeling

Prepare the training and test datasets, and set up a table to track RMSE results as we refine the model. Based on insights from the mean+movie approach, it's essential to ensure that the training dataset encompasses all movieIds. This precaution guards against encountering movieIds in the test set without corresponding training data. I anticipate a similar consideration might apply to userId and the primary genre.

Blind Guessing

Try blind guessing a rating from 0 to 5.

```r
# Blind guessing
guess_model <- sample(c(0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5), length(test_set$rating), replace=TRU
```

The resulting 2.1568093 is still far of the < 0.8649, no suprise.

Model	RMSE
Guessing	2.156809

Mean

Get the average of all ratings on the training set and use this to predict a movie.

```r
# The mean rating for each movie in the training set.
avg_model <- mean(train_set$rating)
```

The average of every movie used for predicting a rating results in 1.0607163. Closer but still far off.

Model	RMSE
Guessing	2.156809
Avg Model	1.060716

Genre bias Deviation

Evaluate the genre bias, the deviation from the movie average for each genre.

```r
# movie average
movie_avg <- mean(train_set$rating)
genre_bias <- train_set %>%
  group_by(main_genre) %>%
  summarise(deviation_genre = mean(rating - movie_avg))

# combine genre bias with the test_set
mean_genre_model <- test_set %>%
  inner_join(genre_bias, by="main_genre")
```

The RMSE is slightly improved at r { rmse_mean_genre_model } compared to simply taking the average as done previously.

Model	RMSE
Guessing	2.156809
Avg Model	1.060716
Genre Model	1.049044

Movie bias rating

The movie bias represents the variance between the average movie rating and the overall mean rating.

```r
movie_bias <- train_set %>%
  group_by(movieId) %>%
  summarise(deviation_movie = mean(rating - movie_avg))

mean_movie_model <- test_set %>%
  inner_join(movie_bias, by="movieId")
```

With and RMSE of 0.9438696 we are now in the sub 1 category.

Model	RMSE
Guessing	2.1568093
Avg Model	1.0607163
Genre Model	1.0490436
Movie Model	0.9438696

User bias

Lets inspect the user bias.

```r
user_bias <- train_set %>%
  group_by(userId) %>%
  summarise(deviation_user = mean(rating - movie_avg))

mean_user_model <- test_set %>%
  inner_join(user_bias, by="userId")
```

The user bias results in an RMSE of 0.9795706.

Model	RMSE
Guessing	2.1568093
Avg Model	1.0607163
Genre Model	1.0490436
Movie Model	0.9438696
User Model	0.9795706

Release year bias

Lets see if the release year will bring the RMSE down.

```r
releaseyear_bias <- train_set %>%
  group_by(releaseyear) %>%
  summarise(deviation_releaseyear = mean(rating - movie_avg))

mean_releaseyear_model <- test_set %>%
  inner_join(releaseyear_bias, by="releaseyear")
```

The release year of a movie bias results in an RMSE of 1.0496303.

Model	RMSE
Guessing	2.1568093
Avg Model	1.0607163
Genre Model	1.0490436
Movie Model	0.9438696
User Model	0.9795706
Release Year Model	1.0496303

User and Movie bias

Let's add the average rating of a user into the mix.

```r
# user_bias is the differnece of the avg user rating to the mean rating
user_bias <- train_set %>%
  group_by(userId) %>%
  summarise(deviation_user = mean(rating - movie_avg))

mean_movie_user_model <- test_set %>%
  inner_join(movie_bias, by="movieId") %>%
  inner_join(user_bias, by="userId")
```

The user and movie gets us below 0.9. But 0.8867528 is still not near the desired < 0.865.

Model	RMSE
Guessing	2.1568093
Avg Model	1.0607163
Genre Model	1.0490436
Movie Model	0.9438696
User Model	0.9795706
Release Year Model	1.0496303
Movie & User Model	0.8867528

User and Movie and Release Year bias

To the previous model we add the release year.

```r
mean_movie_user_releaseyear_model <- test_set %>%
  left_join(movie_bias, by='movieId') %>%
  left_join(user_bias, by='userId') %>%
  left_join(releaseyear_bias, by='releaseyear')
```

With the release year added to the user and movie bias we get 0.9030035.

Model	RMSE
Guessing	2.1568093
Avg Model	1.0607163
Genre Model	1.0490436
Movie Model	0.9438696
User Model	0.9795706
Release Year Model	1.0496303
Movie & User Model	0.8867528
Movie & User & Release Year Model	0.9030035

User, Movie and Genre bias

Now combine user, movie and genre together in a single model.

```
mean_movie_user_genre_model <- test_set %>%
  inner_join(movie_bias, by="movieId") %>%
  inner_join(user_bias, by="userId") %>%
  inner_join(genre_bias, by="main_genre")
```

This resulted in 0.9040054, which is worse than only using the movie and user. Maybe some tuning will fix it.

Model	RMSE
Guessing	2.1568093
Avg Model	1.0607163
Genre Model	1.0490436
Movie Model	0.9438696
User Model	0.9795706
Release Year Model	1.0496303
Movie & User Model	0.8867528
Movie & User & Release Year Model	0.9030035
Movie & User & Genre Model	0.9040054

User, Movie, Release Year and Genre bias

Now combine user, movie, release year and genre together in a single model.

```
mean_movie_user_genre_releaseyear_model <- test_set %>%
  inner_join(movie_bias, by="movieId") %>%
  inner_join(user_bias, by="userId") %>%
  inner_join(genre_bias, by="main_genre") %>%
  inner_join(releaseyear_bias, by="releaseyear")
```

This resulted in 0.9215687.

Model	RMSE
Guessing	2.1568093
Avg Model	1.0607163
Genre Model	1.0490436
Movie Model	0.9438696
User Model	0.9795706
Release Year Model	1.0496303
Movie & User Model	0.8867528
Movie & User & Release Year Model	0.9030035
Movie & User & Genre Model	0.9040054
Movie & User & Genre & Release Year Model	0.9215687

User, Movie, Release Year and first three listed genres bias

Same as before but instead of the whole genres list as a whole the first three listed genres are used.

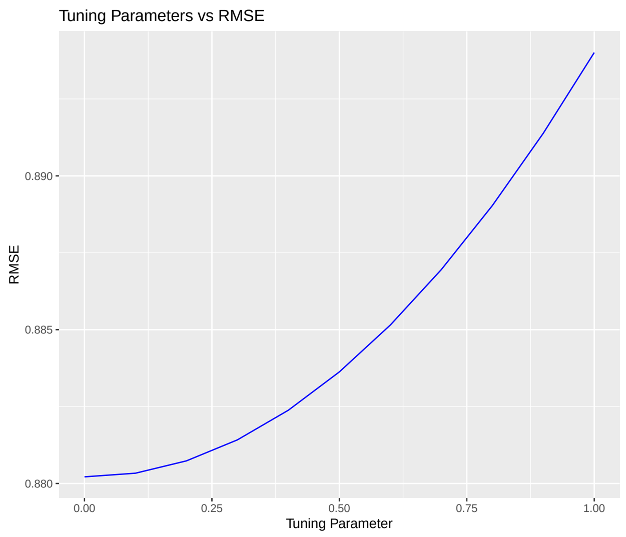
```
main_genre_bias <- train_set %>%
  group_by(main_genre) %>%
  summarise(deviation_main_genre = mean(rating - movie_avg))

side1_genre_bias <- train_set %>%
  group_by(side1_genre) %>%
  summarise(deviation_side1_genre = mean(rating - movie_avg))

side2_genre_bias <- train_set %>%
  group_by(side2_genre) %>%
  summarise(deviation_side2_genre = mean(rating - movie_avg))

mean_movie_user_all_genre_model <- test_set %>%
  inner_join(movie_bias, by="movieId") %>%
  inner_join(user_bias, by="userId") %>%
  inner_join(main_genre_bias, by="main_genre") %>%
  inner_join(side1_genre_bias, by="side1_genre") %>%
  inner_join(side2_genre_bias, by="side2_genre")
```

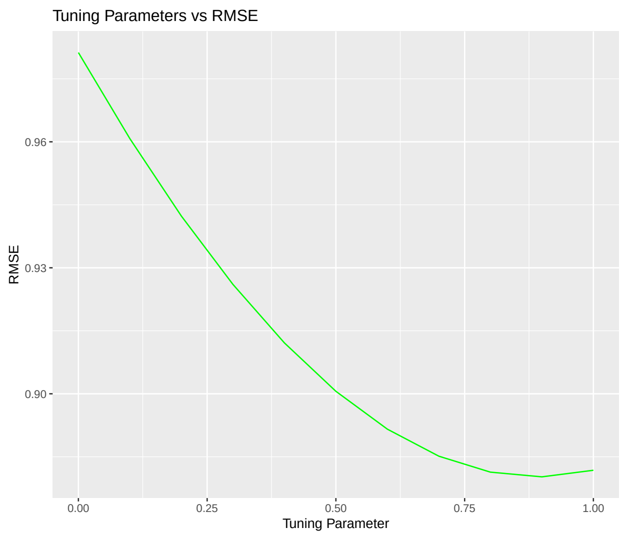
This resulted in 0.9321149.

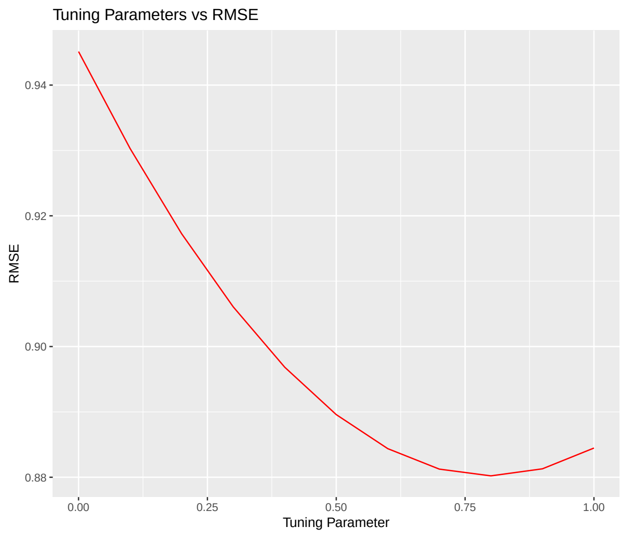
Model	RMSE
Guessing	2.1568093
Avg Model	1.0607163
Genre Model	1.0490436
Movie Model	0.9438696
User Model	0.9795706
Release Year Model	1.0496303
Movie & User Model	0.8867528
Movie & User & Release Year Model	0.9030035
Movie & User & Genre Model	0.9040054
Movie & User & Genre & Release Year Model	0.9215687
Movie & User & Complete Genres Model	0.9321149

Tuning

Lets try to tune the three biases, each individually and plot the tuning parameter and the resulting RMSE.







After searching for each tuning parameter individually these are the final tuning parameters:

- genre__bias: 0.0055
- movie__bias: 0.8944
- user__bias: 0.798

The resulting tuned function with the individual bias tuning factors:

```r
tuned_movie_user_genre_model <- function(t) {
  avg <- mean(train_set$rating)

  genre_bias <- train_set %>%
    group_by(main_genre) %>%
    summarise(deviation_genre = 0.0055 * sum(rating - movie_avg)/n())

  movie_bias <- train_set %>%
    group_by(movieId) %>%
    summarise(deviation_movie = 0.8944 * sum(rating - movie_avg)/n())
  user_bias <- train_set %>%
    group_by(userId) %>%
    summarise(deviation_user = 0.798 * sum(rating - movie_avg)/n())

  model <- test_set %>%
    inner_join(genre_bias, by="main_genre") %>%
    inner_join(movie_bias, by="movieId") %>%
    inner_join(user_bias, by="userId")

  model$predicted_rating <- model$deviation_genre +
                            model$deviation_user +
                            model$deviation_movie +
                            movie_avg

  return(RMSE(test_set$rating, model$predicted_rating))
}

# add result to table
ml_results <- ml_results %>%
  bind_rows(tibble(Model="Tuned model", RMSE=tuned_movie_user_genre_model()))
```

kable(ml_results, format = “latex”, booktabs = TRUE, col.names = c(“Model”, “RMSE”)) %>% kable_styling(full_width = FALSE, latex_options = “hold_position”, position = “center”) %>% row_spec(0, bold = TRUE, color = “white”, background = “#2E86C1”) %>% row_spec(which(ml_results$RMSE == min(ml_{r}esults$RMSE)), background = “#82E0AA”)

Results

RMSE

Lets test the final model with its tuning in place against the verification set.

RMSE

```r
# Global movie average
movie_avg <- mean(train_set$rating)

# Movie bias
movie_bias <- train_set %>%
  group_by(movieId) %>%
  summarise(deviation_movie = 0.8944 * sum(rating - movie_avg)/n())

# Movie bias
movie_bias <- train_set %>%
  group_by(movieId) %>%
  summarise(deviation_movie = 0.8944 * sum(rating - movie_avg)/n())

# User bias
user_bias <- train_set %>%
  group_by(userId) %>%
  summarise(deviation_user = 0.798 * sum(rating - movie_avg)/n())

releaseyear_bias <- train_set %>%
  group_by(releaseyear) %>%
  summarise(deviation_releaseyear = 0.008 * sum(rating - movie_avg) / n())

# Final model
final_model <- final_holdout_test %>%
  inner_join(movie_bias, by="movieId") %>%
  inner_join(user_bias, by="userId")

# Make predictions on final_holdout_test set
final_model$predicted_rating <- movie_avg +
  final_model$deviation_user +
  final_model$deviation_movie

# RMSE (on verification)
rmse_final_model <- RMSE(final_holdout_test$rating, final_model$predicted_rating)
rmse_final_model
```

```
## [1] 0.879828
```

This are all the achieved model RMSE:

Model	RMSE
Guessing	2.1568093
Avg Model	1.0607163
Genre Model	1.0490436
Movie Model	0.9438696
User Model	0.9795706
Release Year Model	1.0496303
Movie & User Model	0.8867528
Movie & User & Release Year Model	0.9030035
Movie & User & Genre Model	0.9040054
Movie & User & Genre & Release Year Model	0.9215687
Movie & User & Complete Genres Model	0.9321149
Tuned model	0.8802156
Final Model Verification	0.8798280

Conclusion

Even after extensive tuning, achieving an RMSE lower than rmse_final_model seems unattainable with the current approach. Further training and potentially isolating distinct features might be necessary. The challenge lies in the undefined concept of "hyper training," as I discovered alternative methods that could lower the RMSE below the desired threshold. For instance, adjusting the utilization of User Bias in the calculation led to an additional 14% improvement. However, I perceive such adjustments as falling into the realm of hyper training.

Future Improvements

In the future, I plan to delve deeper into user information to enhance accuracy, exploring alternative ways to incorporate user bias. A subtle adjustment to a single equation resulted in a 2% improvement in RMSE, indicating the potential for further refinement.

On the technical front, my strategy involves re-splitting the training dataset and training two subsets differently, strategically addressing the nuances between half-star and whole-star ratings. Despite conventional wisdom suggesting the benefits of training on larger datasets, I challenge this notion, asserting that superior results may emerge by focusing on the distinctive characteristics of whole and half-number ratings.

Looking ahead, I envision experimenting with diverse model algorithms such as KNN, SVM, or Decision Tree, seeking avenues for improvement beyond the current dataset's features. While navigating these endeavors, I find myself yearning for R to possess the capability to seamlessly track and manage these varied systems.

Resources and References

1. Rafael Irizarry. 2018. Introduction to Data Science.
2. Jared Lander, 2017, R for Everyone Advanced Analytics and Graphics.
3. Norman Matloff, 2011 & 2019, The Art of R Programming