MovieLens project

Kjersti Framnes

2023-08-09

Introduction to project

As part of the HarvardX PH125.9x Data Science: Capstone course, I've undertaken a project involving the MovieLens dataset. In this report, I'll outline the project's key goals, approach, and my findings.

Recommendation systems leverage user ratings to generate customized item suggestions for users. These systems are beneficial to companies like Amazon, who collect vast amounts of data from users rating their products. By predicting a user's potential rating for an item, these systems can make targeted recommendations that align with a user's interests and preferences. This methodology can also be applied to movie recommendations, which is the focus of this project. Recommendation systems are a staple in machine learning, as evidenced by the success of Netflix's powerful recommendation engine. The Netflix Prize, a competition to develop the best algorithm for predicting user ratings for films based solely on past ratings, underscores the importance of recommendation algorithms in product recommendations.

Project Objectives

The primary goal of this project is to design and train a machine learning algorithm capable of predicting user movie ratings on a scale of 0.5 to 5 stars. I will use a subset of the data (the edx dataset provided by the course staff) to train the algorithm and evaluate its performance on a validation set.

I will assess the algorithm's performance using the Root Mean Square Error (RMSE). RMSE is a commonly used metric for evaluating the accuracy of predictive models by measuring the differences between the observed values and the values predicted by the model. A lower RMSE indicates better model performance. RMSE is calculated by taking the square root of the average of the squared differences between the observed and predicted values. Since each error's impact on the RMSE is proportional to the size of the squared error, RMSE is sensitive to outliers and larger errors have a disproportionately large effect.

In this project, I will develop four models and compare their performance using the RMSE.

‘

Set up:

```
############################################################
# 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")
if(!require(gt)) install.packages("gt", repos = "http://cran.us.r-project.org")
if(!require(ggplot2)) install.packages("ggplot2", repos = "http://cran.us.r-project.org")
if(!require(tidyr)) install.packages("tidyr", repos = "http://cran.us.r-project.org")
if(!require(scales)) install.packages("scales", repos = "http://cran.us.r-project.org")
if(!require(knitr)) install.packages("knitr", repos = "http://cran.us.r-project.org")
if(!require(ggpubr)) install.packages("ggpubr", repos = "http://cran.us.r-project.org")
if(!require(gridExtra)) install.packages("gridExtra", repos = "http://cran.us.r-project.org")



library(tidyverse)
library(caret)
library(gt)
library(ggplot2)
library(tidyr)
library(scales)
library(knitr)
library(ggpubr)
library(gridExtra)

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

options(timeout = 120)

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
# 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)
edx <- rbind(edx, removed)

rm(dl, ratings, movies, test_index, temp, movielens, removed)
knitr::opts_chunk$set(echo = FALSE,warning=FALSE)
```

‘

Create a train and test set:

After creating the validation set called "final_holdout_test" we need to partition further to obtain a train and test dataset. When I have experimented with different models to obtain the lowest RMSE I will run the same models using the unseen validation dataset ("final_holdout_test). This in order to avoid over fitting our model and verifying the best model.

```
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
# set.seed(1) # if using R 3.5 or earlier
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
edx_train <- edx[-test_index,]
edx_temp <- edx[test_index,]

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

# Add rows removed from final hold-out test set back into edx set
removed <- anti_join(edx_temp, edx_test)
edx_train <- rbind(edx_train, removed)

rm(dl, ratings, movies, test_index, temp, movielens, removed, edx_temp)
```

‘

Exploratory analyis of data

Information about the dataset:

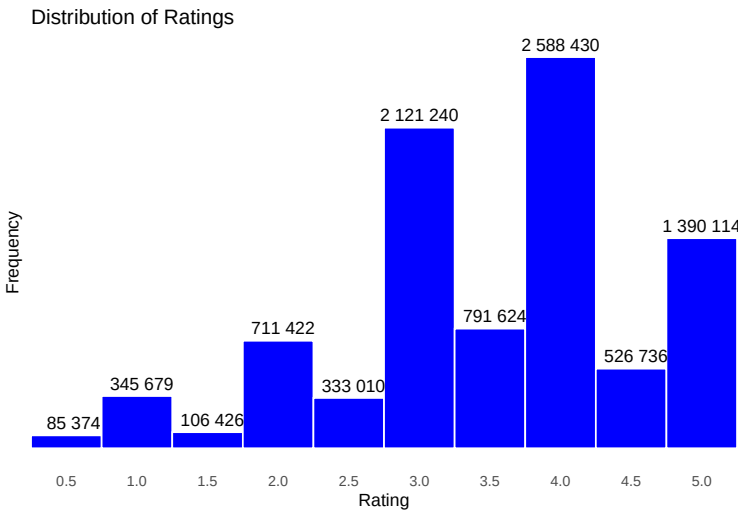
```
## 'data.frame':    9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : int  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|Ad
```

Summary of each column in the dataset:

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:9000055     Length:9000055
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

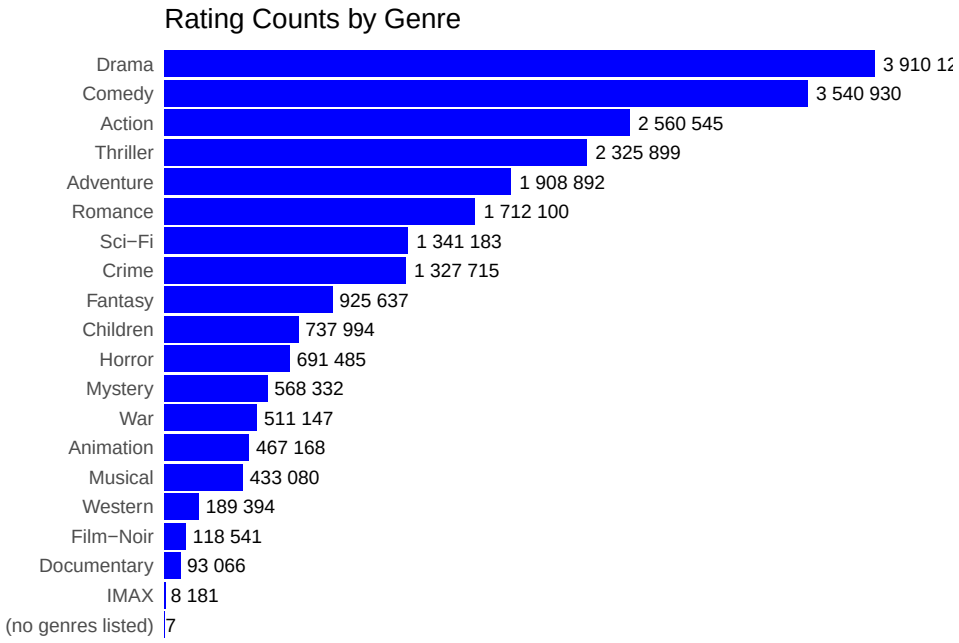
‘

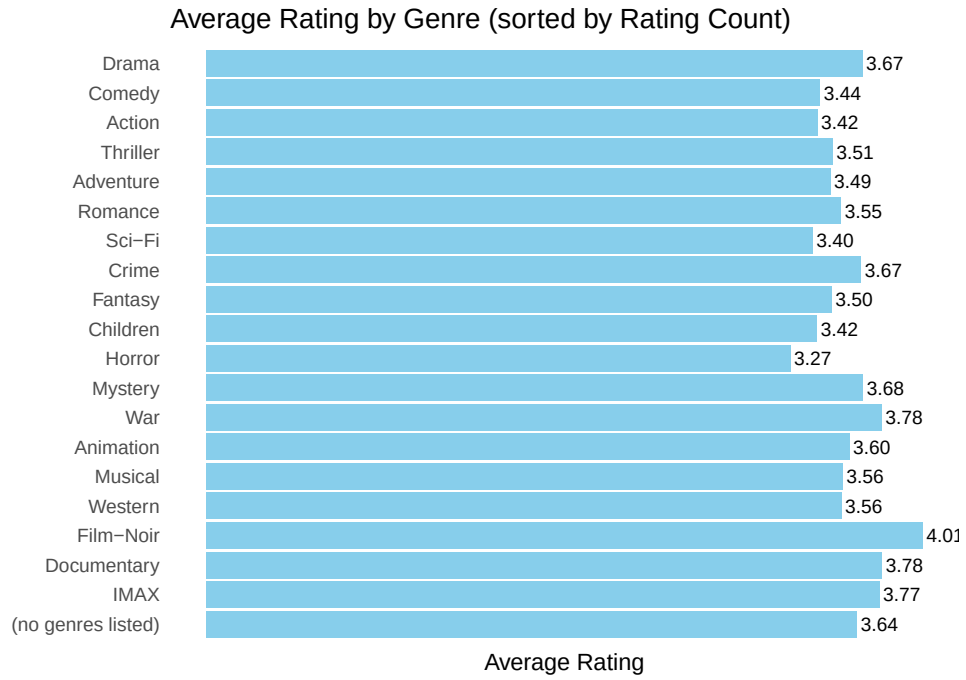
Lets see how the ratings looks like:



We see that three ratings are more populare in the dataset.

Lets have a look at the different genres in the dataset. Important to note that a movie can have more than one genre. But if I split the genres into single genres we can see that the genre with the most rating is "Drama". And looking at the averge rating for each single genre we see that the genre "Film Noir" (low number of ratings) has the highest average rating and the genre "Horror" has the lowest average rating.



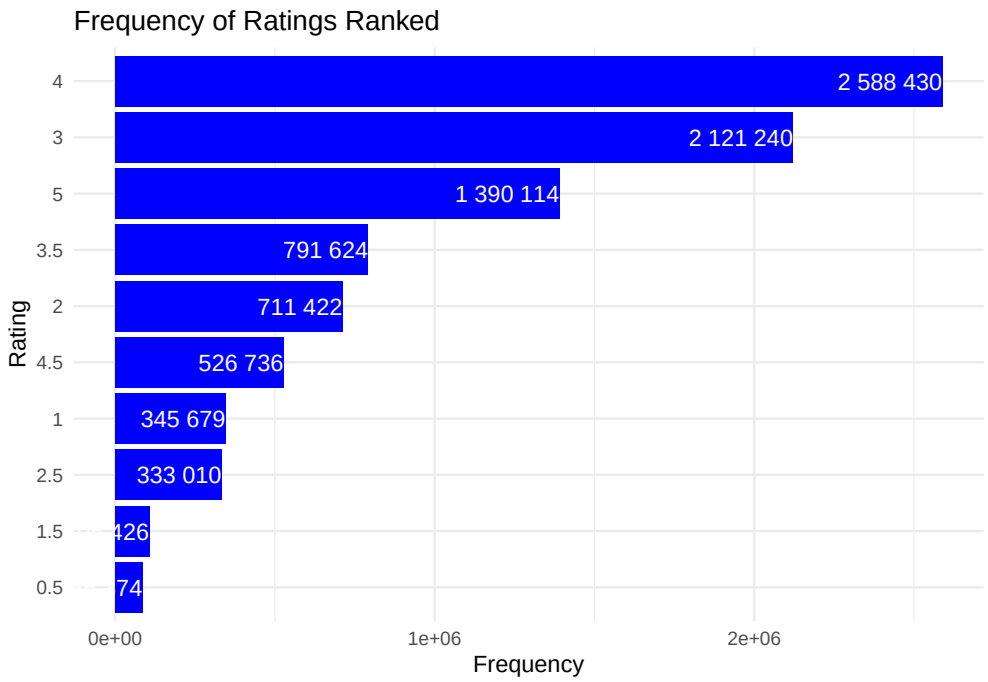


Most rated movies:

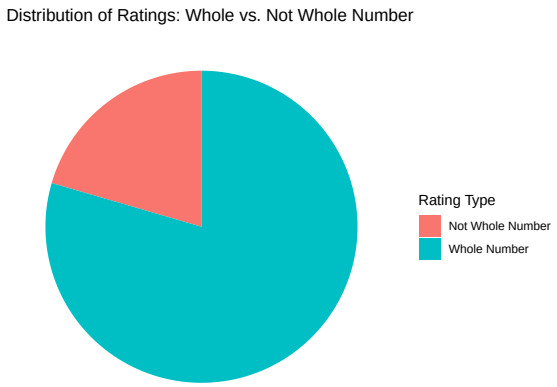
Table 1: Top 10 Most Rated Movies with Average Ratings

title	rating_count	average_rating
Pulp Fiction (1994)	31362	4.154789
Forrest Gump (1994)	31079	4.012822
Silence of the Lambs, The (1991)	30382	4.204101
Jurassic Park (1993)	29360	3.663522
Shawshank Redemption, The (1994)	28015	4.455131
Braveheart (1995)	26212	4.081852
Fugitive, The (1993)	25998	4.009155
Terminator 2: Judgment Day (1991)	25984	3.927859
Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)	25672	4.221311
Apollo 13 (1995)	24284	3.885789

In this graph we can clearly see what rating are the most populare;

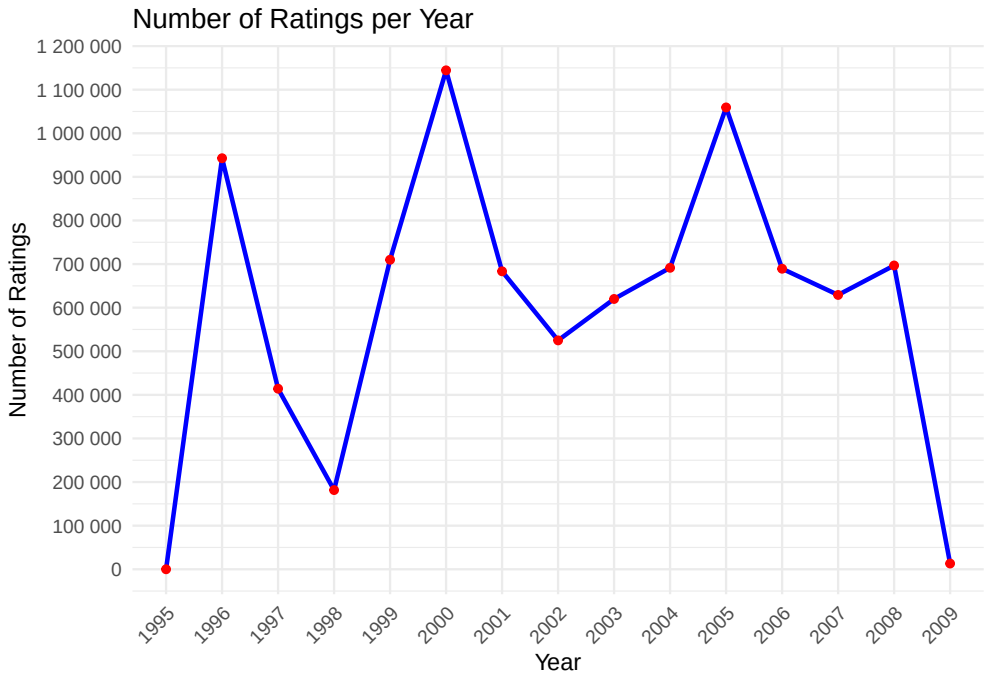


If we look at ratings given as whole numbers or half numbers we clearly see that the critics are more fab of giving ratings as whole numbers. Maybe this means they are more certain when they give ratings.



Now I also wanted to look at the ratings given over years. Here I had to convert the timestamp into a more workable format; as.Date(as.POSIXct(edx$timestamp, origin="1970-01-01", tz="UTC"))

By looking at the time series grapg we can see that in the years 1996, 2000 and 2005 most ratings were given.


Number of Ratings per Year
Number of Ratings
1 200 000
1 100 000
1 000 000
900 000
800 000
700 000
600 000
500 000
400 000
300 000
200 000
100 000
0
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
Year

‘

Modelling

Introduction: We need to build a function that we will use with the different models:

```
# Define a function 'RMSE' to calculate the Root Mean Square Error between true ratings and predicted r
RMSE <- function(true_ratings, predicted_ratings) {
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

Model 1: Average rating

Firstly we need to calculate the mean to have a benchmark;

```
#Creating the first model
Model1_rmse <- RMSE(edx_test$rating,mu)
model_results <- tibble(Model = "1. Average rating model", RMSE = Model1_rmse)
```

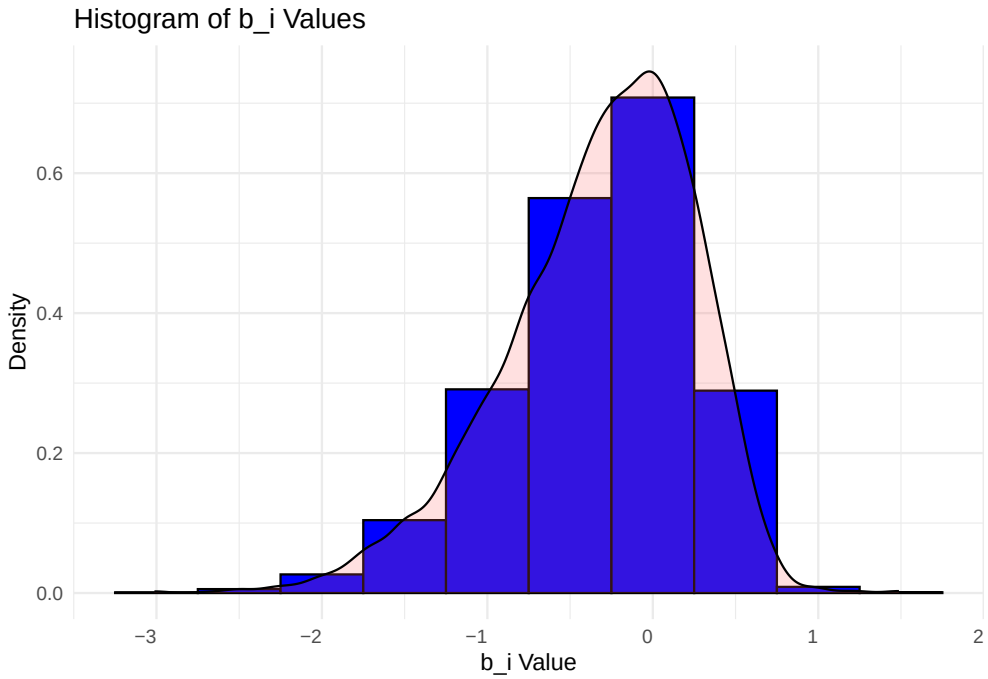
Model Results

Performance Metrics

Model Name	RMSE
1. Average rating model	1.0601

Model 2: Movie effect

The inherent quality of movies can cause variations in their ratings. To account for this, we introduce a "bias" term, b_i, to our model, representing the average deviation of ratings for a specific movie from the overall average rating (μ). While these deviations are traditionally called "effects" in statistics, they're termed "bias" in contexts like the Netflix challenge. Instead of using the computationally intensive lm() function to estimate these bias terms for each movie, we can directly compute them by subtracting the overall average rating from the average rating of each movie. This approach offers a more efficient way to capture the inherent differences in movie ratings.




```r
# Predict ratings using a baseline and bias for each movie
prediction <- mu + edx_test %>%
  left_join(avgs, by='movieId') %>%  # Join 'edx_test' with 'avgs' on 'movieId'
  pull(b_i)                          # Extract bias values

# Calculate the RMSE between predicted ratings and actual ratings from 'edx_test'
Model2_rmse <- RMSE(prediction, edx_test$rating)
```

Model Results

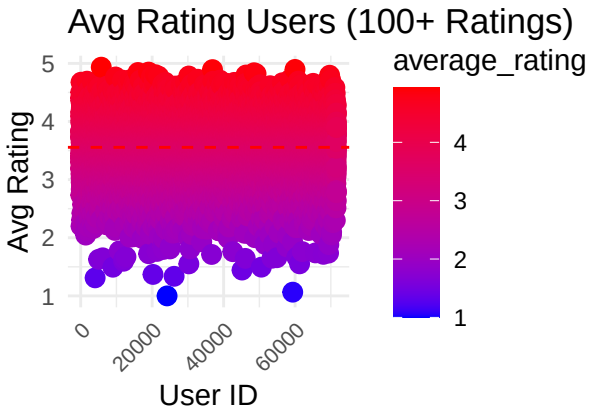
Performance Metrics

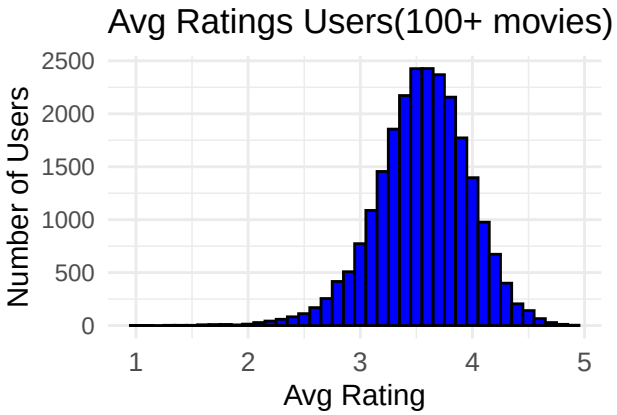
Model Name	RMSE
1. Average rating model	1.0601
2. Movie effect model	0.9417

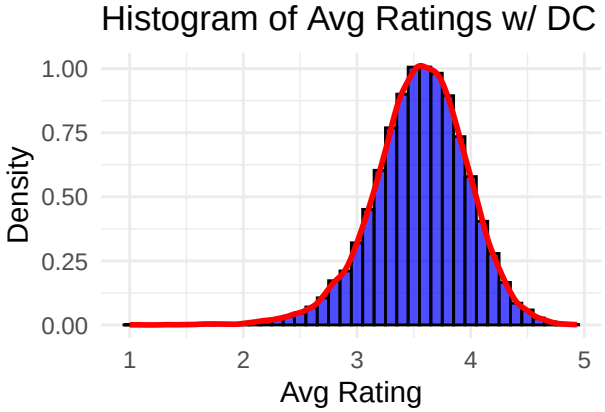
Model 3: Movie and user effect

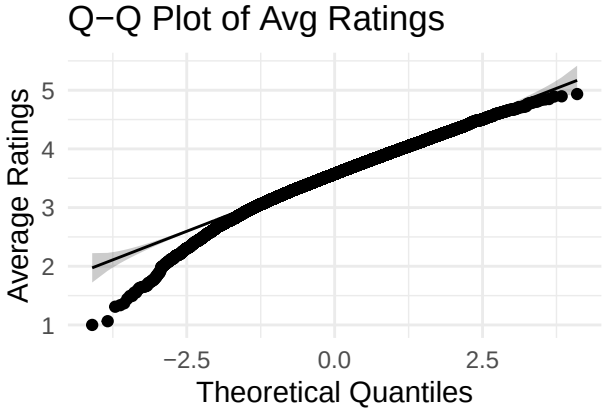
We want to reduce RSME further. While movies have their inherent appeal or lack thereof, users also bring their unique biases to the table. By acknowledging both these aspects, the model aims to make more accurate predictions on how a user might rate a particular movie.

Looking at this we can see that there are some extreme values. Although user avg rating seems to be close to normally distributed we can see from the historgams and the Q-Q plot that we have longer negative / left side tail indicating there are more extreme negative values and we can not say the user avg rating is normally distrubuted.









The User-Movie Rating Model predicts a user's movie rating based on two main factors:

Movie Effect (b_i): Reflects the inherent quality or appeal of a movie. User Effect (b_u): Represents an individual user's tendency to rate movies more or less favorably. The mathematical representation of the model is:

The equation $y_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$

$y_{u,i}$ is the predicted rating of user u for movie i.

μ denotes the average rating across all movies.

b_i is the movie effect.

b_u is the user effect.

$\epsilon_{u,i}$ represents random errors centered at 0.

In essence, the model combines the overall movie appeal and the individual biases of users to forecast ratings. By incorporating both these elements, it aims to achieve more precise predictions on how a specific user might rate a given movie.

```
# Calculate the average rating bias for each user in the 'edx_train' dataset
user_avg_rating_edx_train <- edx_train %>%
  left_join(avgs, by='movieId') %>%        # Join with 'avgs' to get movie biases (b_i)
  group_by(userId) %>%                     # Group by user
  summarize(b_u=mean(rating - mu - b_i))   # Calculate user bias (b_u) for each user

# Predict ratings for 'edx_test' using global average, movie bias, and user bias
```

```r
prediction_model3 <- edx_test %>%
  left_join(avgs, by='movieId') %>%                              # Join with 'avgs' to get movie biases (b_
  left_join(user_avg_rating_edx_train, by='userId') %>%          # Join with user biases (b_u)
  mutate(pred_m3 = mu + b_i + b_u) %>%                           # Compute the predicted ratings
  pull(pred_m3)                                                  # Extract the predicted values

# Calculate the RMSE between predicted ratings and actual ratings from 'edx_test' for Model 3
model3_rmse <-  RMSE(prediction_model3, edx_test$rating)
```

Model Results

Performance Metrics

Model Name	RMSE
1. Average rating model	1.0601
2. Movie effect model	0.9417
3. Movie and user effect model	0.8636

In our initial model, we found that movies with few ratings could have extreme bias terms, b_i , leading to higher RMSE in our predictions. We previously used confidence intervals to address this uncertainty, but for predictions, we need a single estimate. Regularization offers a solution by adding a penalty for large b_i values in the minimization process, reducing the influence of extreme bias estimates from small sample sizes. The regularization parameter, λ determines the penalty strength. We must find the optimal λ to minimize the RMSE. Regularization shrinks the b_i and b_u terms, especially for small rating counts, mitigating outliers' impact on our predictions.

Model 4: Regularized movie and user effect

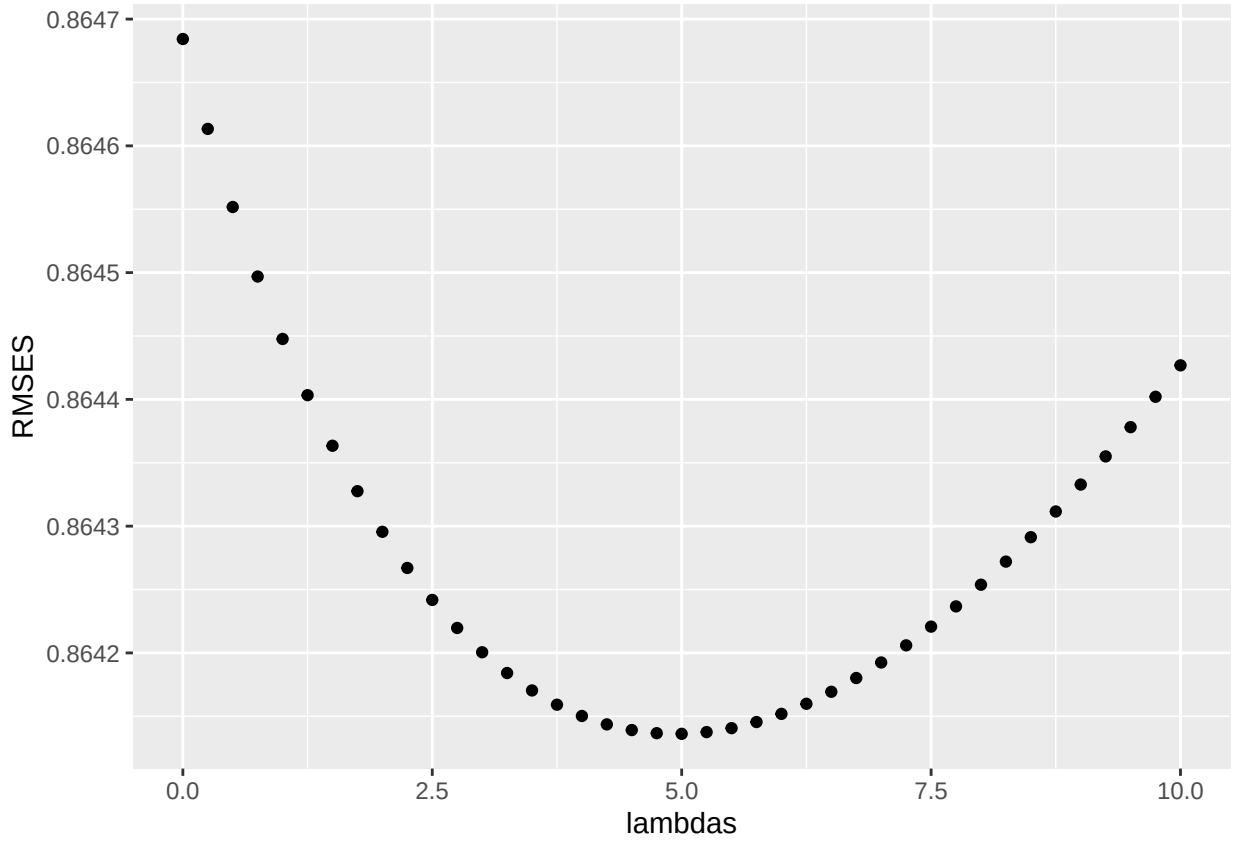
```r
# Initialize a sequence of lambda values from 0 to 10 in increments of 0.1
lambdas <- seq(0, 10, 0.25)

# Use sapply to calculate the RMSE for each lambda value
RMSES <- sapply(lambdas, function(l){
  # Calculate the global mean rating from the edx_train dataset
  edx_train_mu <- mean(edx_train$rating)

  # Calculate the bias term b_i for each movie in the edx_train dataset
  b_i <- edx_train %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - edx_train_mu)/(n() + l))

  # Calculate the bias term b_u for each user in the edx_train dataset
  b_u <- edx_train %>%
    left_join(b_i, by='movieId') %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - edx_train_mu)/(n() +l))

  # Predict the ratings for the edx_test dataset based on the bias terms b_i and b_u
  predicted_ratings <- edx_test %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
```

```
    mutate(pred = edx_train_mu + b_i +  b_u) %>% .$pred

  # Return the RMSE between the predicted ratings and the actual ratings in the edx_test dataset
  return(RMSE(predicted_ratings, edx_test$rating))
})

# Find the lambda value that gives the lowest RMSE
lambda <- lambdas[which.min(RMSES)]
lambda
```

```
## [1] 5
```

The best lambda is 5.

If we look at the plot we can verify the optimal lambda:




```
edx_train_mu <- mean(edx_train$rating)

# Calculate the bias term b_i for each movie in the edx_train dataset
b_i <- edx_train %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - edx_train_mu)/(n()+lambda))

# Calculate the bias term b_u for each user in the edx_train dataset
b_u <-edx_train %>%
```

```
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - edx_train_mu)/(n()+lambda))

# Predict the ratings for the edx_test dataset based on the bias terms b_i and b_u
prediction_model4 <- edx_test %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(predictions = edx_train_mu + b_i + b_u) %>% .$predictions

model4_rmse <-RMSE(prediction_model4,edx_test$rating)
```

Model Results

Performance Metrics

Model Name	RMSE
1. Average rating model	1.0601
2. Movie effect model	0.9417
3. Movie and user effect model	0.8636
4. (Regularized) Movie and user effect model	0.8641

‘

Summmary:

Average Rating Model (RMSE: 1.0601)

This is a basic model that predicts the average rating for all movies. Its relatively high RMSE indicates that it doesn't account for variations in individual movie ratings or user preferences, and therefore may not be very accurate. ### Movie Effect Model (RMSE: 0.9417) This model introduces a factor for individual movies, presumably improving its accuracy compared to the simple average rating model. The RMSE has decreased, suggesting better performance in predicting movie ratings. ### Movie and User Effect Model (RMSE: 0.8636) This model incorporates effects from both movies and users. This means it accounts for variations in ratings by movie as well as variations in ratings by individual users. Its lower RMSE suggests that taking both movie and user effects into consideration leads to more accurate predictions. ### (Regularized) Movie and User Effect Model (RMSE: 0.8641) This model is a variation of the previous one but includes regularization, which is a technique used to avoid overfitting. The RMSE is slightly higher than the non-regularized version. This could indicate that while the model may generalize better to unseen data (a typical advantage of regularization), it might not perform as well on the training data or the specific test data provided.

In summary, as we move from the first to the third model, the RMSE decreases, indicating improvements in prediction accuracy. The fourth model, while having a slightly higher RMSE than the third, might be better suited for new or unseen data due to its regularization.

‘

Now we need to run all the models using the validation ("final_holdout_test") dataset.

Model 1: Average rating

Firstly we need to calculate the mean;

```
#Creating the first model
Model1_rmse_final <- RMSE(final_holdout_test$rating,mu_final)
```

Model Results

Performance Metrics

Model Name	RMSE	RMSE Final
1. Average rating model	1.0601	1.0612

Model 2: Movie effect with validation data

```
# Calculate the average bias (b_i_final) for each movie from the 'edx' dataset
avgs_final <- edx %>%
  group_by(movieId) %>%
  summarize(b_i_final = mean(rating - mu_final))  # Compute bias by subtracting global average (mu_fina

# Predict ratings for 'final_holdout_test' using the global average (mu_final) and movie bias (b_i_fina
prediction_final <- mu_final + final_holdout_test %>%
  left_join(avgs_final, by='movieId') %>%         # Join with 'avgs_final' to get the movie biases
  pull(b_i_final)                                 # Extract the movie bias values

# Calculate the RMSE between the predicted ratings and actual ratings from 'final_holdout_test'
Model2_rmse_final <- RMSE(prediction_final, final_holdout_test$rating)
```

Model Results

Performance Metrics

Model Name	RMSE	RMSE Final
1. Average rating model	1.0601	1.0612
2. Movie effect model	0.9417	0.9439

Model 3: Movie and user effect with validation data

```
# Calculate the user-specific bias (b_u_final) from the 'edx' dataset
user_avg_rating_edx <- edx %>%
  left_join(avgs_final, by='movieId') %>%          # Join with 'avgs_final' to get movie biases (b_i_fi
  group_by(userId) %>%                             # Group by user
  summarize(b_u_final=mean(rating - mu_final - b_i_final))   # Compute user bias by subtracting global

# Predict ratings for 'final_holdout_test' using global average (mu_final), movie bias (b_i_final), and
```

```
prediction_model3_final <- final_holdout_test %>%
  left_join(avgs_final, by='movieId') %>%                          # Join with 'avgs_final' to get movi
  left_join(user_avg_rating_edx, by='userId') %>%                  # Join with user biases
  mutate(pred_m3_final = mu_final + b_i_final + b_u_final) %>%     # Compute the predicted ratings
  pull(pred_m3_final)                                              # Extract the predicted values

# Calculate the RMSE for Model 3 between predicted and actual ratings from 'final_holdout_test'
model3_rmse_final <-  RMSE(prediction_model3_final, final_holdout_test$rating)
model3_rmse_final  # Output the RMSE value
```

```
## [1] 0.8653488
```

Model Results

Performance Metrics

Model Name	RMSE	RMSE Final
1. Average rating model	1.0601	1.0612
2. Movie effect model	0.9417	0.9439
3. Movie and user effect model	0.8636	0.8653

Model 4: Regularized movie and user effect with validation data

```
edx_mu <- mean(edx$rating)

# Calculate the bias term b_i for each movie in the edx_train dataset
b_i_final <- edx %>%
  group_by(movieId) %>%
  summarize(b_i_final = sum(rating - edx_mu)/(n()+lambda))

# Calculate the bias term b_u for each user in the edx_train dataset
b_u_final <-edx %>%
  left_join(b_i_final, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u_final = sum(rating - b_i_final - edx_mu)/(n()+lambda))

# Predict the ratings for the edx_test dataset based on the bias terms b_i and b_u
prediction_model4_final <- final_holdout_test %>%
  left_join(b_i_final, by = "movieId") %>%
  left_join(b_u_final, by = "userId") %>%
  mutate(predictions = edx_mu + b_i_final + b_u_final) %>% .$predictions

model4_rmse_final <-RMSE(prediction_model4_final,final_holdout_test$rating)
```

Model Results

Performance Metrics

Model Name	RMSE	RMSE Final
1. Average rating model	1.0601	1.0612

2. Movie effect model	0.9417	0.9439
3. Movie and user effect model	0.8636	0.8653
4. (Regularized) Movie and user effect model	0.8641	0.8648

Summary and Conclusion:

In our evaluation of the models against the unseen dataset, we noticed a clear pattern of improved predictive accuracy. Starting from the Average Rating Model and moving to the Regularized Movie and User Effect Model, there's a consistent decrease in RMSE. This shows the benefits of adding more detailed features and regularization techniques.

However, an important observation to make is that each model displayed a slightly higher RMSE when tested on the unseen data compared to results from our train/test split. This indicates that, while our models are improving, they still face a slight challenge when adapting to completely new data.

n our pursuit to refine our predictive models, we've achieved significant improvements. However, by incorporating factors such as the movie's release date (indicating its age) and main genre, there's potential for further optimization, ensuring more consistent performance across varied datasets.

There are more modeling methods we can consider to potentially boost our predictions. We could look into deep learning, which, while powerful, can be resource-intensive and might require more than a standard Mac to run efficiently. Techniques like matrix factorization, specifically Singular Value Decomposition (SVD), could be applied – they've been popular for recommendation tasks. Ensemble methods, where we combine different models, are another avenue. However, these can sometimes be computationally demanding as well.