System Recommendation Movie

Yuliana Naddaf

2024-06-09

The following project presents a detailed analysis of the performance of various movie recommendation models using the MovieLens 10M dataset.

Linear Regression Model (Mean Baseline):

*** The average of the scores in the training set is calculated.**

*** Predictions are made using this average for the validation set.**

*** The model is evaluated using metrics such as MAE, MSE and RMSE.**

*** This approach establishes a baseline for comparing other more complex models.**

Linear Regression Model with Film Bias:

*** The bias is calculated for each movie in the training set.**

*** This bias is incorporated into a linear regression model to make predictions.**

*** The model is evaluated using the same evaluation metrics.**

Linear Regression Model with Movie and User Bias:

*** The bias is calculated for both movies and users in the training set.**

*** These biases are incorporated into a linear regression model to make predictions.**

*** The model is evaluated using the same evaluation metrics.**

Linear Regression Model with Regularized Bias:

*** The code begins by installing and loading the necessary libraries. It then defines three primary functions: compute_biases, predict_ratings, and evaluate_model. The compute_biases function calculates the average rating and biases for movies and users, returning these values in a list. The predict_ratings function uses these biases to predict ratings for the validation dataset. The evaluate_model function integrates these two functions, computing the biases and using them to predict ratings, ultimately calculating and returning the RMSE.**

*** In the main execution block, the structure and missing values of the training and validacion datasets are checked. A range of lambda values is defined, and for each lambda, the RMSE is calculated using the evaluate_model function. The lambda with the lowest RMSE is selected as the best lambda. Final biases are computed using this best lambda, and these biases are then used to predict ratings for the validation dataset. The model is evaluated by calculating MAE, MSE, and RMSE, and the results are printed.**

*** Overall, the code efficiently implements a regularization model by iteratively finding the best lambda, computing biases, predicting ratings, and evaluating the model's performance.**

Factorization Matrix:

*** The training and test sets are converted into a format compatible with the factorization matrix.**

*** The factorization matrix is used to make predictions.**

*** The model is evaluated using the same evaluation metrics.**

Each model is evaluated on both the validation set and the final test set (final_holdout_test). The evaluation results are compiled in a table that shows the MAE, MSE, and RMSE metrics for each model in both data sets. This approach allows the performance of different movie recommendation models to be compared in terms of predictive accuracy.

In conclusion the results show that more advanced models, such as those that incorporate both movie and user biases, along with regularization or matrix factorization, significantly outperform the simple baseline model that uses only the average of the ratings. These advanced models achieve significant reductions in Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE) compared to the base model, indicating an improvement in recommendation accuracy. of movies.

Furthermore, matrix factorization emerges as the most effective technique, with the lowest values of MAE, MSE, and RMSE in both data sets, suggesting that it more accurately captures the complex interactions between users and movies in the data set. MovieLens 10M.

```
options(repos = c(CRAN = "https://cloud.r-project.org/"))

# 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")
```

In summary, the results suggest that more complex models, which consider both movie and user bias, along with regularization or matrix factorization, tend to provide better movie recommendation accuracy compared to a simple baseline approach.

```
## Le chargement a nécessité le package : tidyverse

## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ---------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

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

```
## Le chargement a nécessité le package : caret
## Le chargement a nécessité le package : lattice
##
## Attachement du package : 'caret'
##
## L'objet suivant est masqué depuis 'package:purrr':
##
##     lift
```

```
install.packages("tibble")
```

```
## Warning: le package 'tibble' est en cours d'utilisation et ne sera pas installé
```

```
install.packages("stringr")
```

```
## Warning: le package 'stringr' est en cours d'utilisation et ne sera pas
## installé
```

```
library(stringr)
library(tidyverse)
library(caret)
library(tibble)


# MovieLens 10M dataset:
```

```r
# 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))

head(ratings)
```

```
##   userId movieId rating timestamp
## 1      1     122      5 838985046
## 2      1     185      5 838983525
## 3      1     231      5 838983392
## 4      1     292      5 838983421
## 5      1     316      5 838983392
## 6      1     329      5 838983392
```

```r
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))

head(movies)
```

```
##   movieId                               title
## 1       1                    Toy Story (1995)
## 2       2                      Jumanji (1995)
## 3       3             Grumpier Old Men (1995)
## 4       4            Waiting to Exhale (1995)
## 5       5 Father of the Bride Part II (1995)
## 6       6                           Heat (1995)
##                                        genres
```

```
## 1 Adventure|Animation|Children|Comedy|Fantasy
## 2                  Adventure|Children|Fantasy
## 3                              Comedy|Romance
## 4                        Comedy|Drama|Romance
## 5                                      Comedy
## 6                       Action|Crime|Thriller
```

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

head(movielens)
```

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046               Boomerang (1992)
## 2      1     185      5 838983525                Net, The (1995)
## 3      1     231      5 838983392           Dumb & Dumber (1994)
## 4      1     292      5 838983421                Outbreak (1995)
## 5      1     316      5 838983392                Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
##                          genres
## 1                Comedy|Romance
## 2          Action|Crime|Thriller
## 3                        Comedy
## 4  Action|Drama|Sci-Fi|Thriller
## 5       Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
```

```
# Final hold-out test set will be 10% of MovieLens data
set.seed(1)
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")

head(final_holdout_test)
```

```
##   userId movieId rating  timestamp
## 1      1     588    5.0  838983339
## 2      2    1210    4.0  868245644
## 3      2    1544    3.0  868245920
## 4      3     151    4.5 1133571026
## 5      3    1288    3.0 1133571035
## 6      3    5299    3.0 1164885617
##                                                  title
## 1                                      Aladdin (1992)
## 2       Star Wars: Episode VI - Return of the Jedi (1983)
## 3 Lost World: Jurassic Park, The (Jurassic Park 2) (1997)
## 4                                      Rob Roy (1995)
## 5                          This Is Spinal Tap (1984)
## 6                  My Big Fat Greek Wedding (2002)
```

```
##                                       genres
## 1 Adventure|Animation|Children|Comedy|Musical
## 2                      Action|Adventure|Sci-Fi
## 3     Action|Adventure|Horror|Sci-Fi|Thriller
## 4                    Action|Drama|Romance|War
## 5                              Comedy|Musical
## 6                              Comedy|Romance
```

```
# 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)`
```

```
edx <- rbind(edx, removed)

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

```
# splits the dataset edx into two parts: a training set that contains 80% of the data and
# a validation set that contains the remaining 20%.

set.seed(1)
train_index <- createDataPartition(y = edx$rating, times = 1, p = 0.8, list = FALSE)
training <- edx[train_index,]
validacion <- edx[-train_index,]


# LINEAR REGRESION MODEL
# Mean Baseline
# Baseline model predicts the average of the ratings from the training set for all data in the
# validation set, evaluates its performance using various metrics, and stores and displays the
# results.

install.packages("Metrics")
```

```
## Installation du package dans 'C:/Users/nadda/AppData/Local/R/win-library/4.2'
## (car 'lib' n'est pas spécifié)

## le package 'Metrics' a été décompressé et les sommes MD5 ont été vérifiées avec succés
##
## Les packages binaires téléchargés sont dans
##  C:\Users\nadda\AppData\Local\Temp\RtmpaETIyL\downloaded_packages
```

```
library(Metrics)
```

```
##
## Attachement du package : 'Metrics'

## Les objets suivants sont masqués depuis 'package:caret':
##
##     precision, recall
```

```r
# Calculate the average of the ratings in the training set
mean_rating <- mean(training$rating)

# Create a prediction vector using the calculated average
predictions <- rep(mean_rating, nrow(validacion))

# Evaluate the model and store the results in a tibble
valuation <- tibble(
  Models = "Mean baseline model (validation)",
  MAE = mae(validacion$rating, predictions),
  MSE = mse(validacion$rating, predictions),
  RMSE = rmse(validacion$rating, predictions)
)

# Print the evaluations
print(valuation)
```

```
## # A tibble: 1 x 4
##   Models                              MAE   MSE  RMSE
##   <chr>                             <dbl> <dbl> <dbl>
## 1 Mean baseline model (validation) 0.856  1.13  1.06
```

```r
# Final Evaluation with the final_holdout_test
# Calculate the average of the ratings in the full dataset
mean_rating <- mean(edx$rating)

# Create a prediction vector using the calculated average
predictions <- rep(mean_rating, nrow(final_holdout_test))

# Evaluate the model and store the results in a data frame
valuation <- data.frame(
  Models = "Mean baseline model (final_holdout_test)",
  MAE = mae(final_holdout_test$rating, predictions),
  MSE = mse(final_holdout_test$rating, predictions),
  RMSE = rmse(final_holdout_test$rating, predictions),
  stringsAsFactors = FALSE
)

# Print the evaluations
print(valuation)
```

```
##                                     Models       MAE     MSE     RMSE
## 1 Mean baseline model (final_holdout_test) 0.8560101 1.12498 1.060651
```

```r
# MOVIE BIAS  Caracteristica efect s_movie
# Bias per movie
# The code calculates a rating prediction model that incorporates the bias of each movie
# and evaluates its performance using error metrics.

# Calculate the bias for each movie
bias_movie <- training %>%
  group_by(movieId) %>%
```

```
  summarize(s_movie = mean(rating - mean_rating),
            s_movie_isolated = mean(rating))

# Show the first 20 rows of the movie biases
head(bias_movie, 20)
```

```
## # A tibble: 20 x 3
##    movieId s_movie s_movie_isolated
##      <int>   <dbl>            <dbl>
##  1       1  0.412              3.92
##  2       2 -0.323              3.19
##  3       3 -0.358              3.15
##  4       4 -0.651              2.86
##  5       5 -0.425              3.09
##  6       6  0.299              3.81
##  7       7 -0.147              3.37
##  8       8 -0.379              3.13
##  9       9 -0.525              2.99
## 10      10 -0.0828             3.43
## 11      11  0.164              3.68
## 12      12 -0.928              2.58
## 13      13 -0.345              3.17
## 14      14 -0.0638             3.45
## 15      15 -0.782              2.73
## 16      16  0.243              3.76
## 17      17  0.468              3.98
## 18      18 -0.176              3.34
## 19      19 -0.945              2.57
## 20      20 -0.655              2.86
```

```
# Build the linear model (mean + bias movie)
predicted_ratings <- validacion %>%
  left_join(bias_movie, by = "movieId") %>%
  mutate(prediction = mean_rating + s_movie)

# Replace NA with the average value
predicted_ratings$prediction[is.na(predicted_ratings$prediction)] <- mean_rating

# Evaluate the model and store the results in a table
valuation <- bind_rows(valuation,
             tibble(Models = "Model linear mean + s_movie (validacion)",
             MAE = mae(validacion$rating, predicted_ratings$prediction),
             MSE = mse(validacion$rating, predicted_ratings$prediction),
             RMSE = rmse(validacion$rating, predicted_ratings$prediction)))

# Print the evaluations
print(valuation)
```

```
##                                     Models       MAE       MSE      RMSE
## 1 Mean baseline model (final_holdout_test) 0.8560101 1.1249798 1.0606506
## 2 Model linear mean + s_movie (validacion) 0.7381239 0.8913586 0.9441179
```

```r
# Final Evaluation with the final_holdout_test
# Calculate the bias for each movie
bias_movie <- edx %>%
  group_by(movieId) %>%
  summarize(s_movie = mean(rating - mean_rating),
            s_movie_isolated = mean(rating))

# Show the first 20 rows of the movie biases
head(bias_movie, 20)
```

```
## # A tibble: 20 x 3
##    movieId s_movie s_movie_isolated
##      <int>   <dbl>            <dbl>
##  1       1  0.414              3.93
##  2       2 -0.310              3.20
##  3       3 -0.362              3.15
##  4       4 -0.632              2.88
##  5       5 -0.434              3.08
##  6       6  0.299              3.81
##  7       7 -0.147              3.37
##  8       8 -0.395              3.12
##  9       9 -0.517              3.00
## 10      10 -0.0878             3.42
## 11      11  0.165              3.68
## 12      12 -0.936              2.58
## 13      13 -0.328              3.18
## 14      14 -0.0661             3.45
## 15      15 -0.793              2.72
## 16      16  0.239              3.75
## 17      17  0.466              3.98
## 18      18 -0.180              3.33
## 19      19 -0.937              2.58
## 20      20 -0.651              2.86
```

```r
# Construct the linear model (mean + movie bias)
predicted_ratings <- final_holdout_test %>%
  left_join(bias_movie, by = "movieId") %>%
  mutate(prediction = mean_rating + s_movie)

# Replace NA with the average value
predicted_ratings$prediction[is.na(predicted_ratings$prediction)] <- mean_rating

# Evaluate the model and store the results in a table
valuation <- bind_rows(valuation,
                       tibble(Models = "Model linear mean + s_movie (final_holdout_test)",
                       MAE = mae(final_holdout_test$rating, predicted_ratings$prediction),
                       MSE = mse(final_holdout_test$rating, predicted_ratings$prediction),
                       RMSE = rmse(final_holdout_test$rating, predicted_ratings$prediction)))

# Print the evaluations
print(valuation)
```

```
##                                          Models       MAE       MSE
```

```
## 1         Mean baseline model (final_holdout_test) 0.8560101 1.1249798
## 2         Model linear mean + s_movie (validacion) 0.7381239 0.8913586
## 3 Model linear mean + s_movie (final_holdout_test) 0.7381881 0.8905784
##        RMSE
## 1 1.0606506
## 2 0.9441179
## 3 0.9437046
```

```
# Bias per user
# The code calculates a rating prediction model that incorporates both movie and user biases,
# evaluates its performance using error metrics, and updates the evaluation results.
# Calculate the bias of each user

bias_user <- training %>%
  left_join(bias_movie, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(s_usuario = mean(rating - mean_rating - s_movie),
            s_usuario_isolated = mean(rating))

# Show the first 20 rows of user biases
head(bias_user, 20)
```

```
## # A tibble: 20 x 3
##    userId s_usuario s_usuario_isolated
##     <int>     <dbl>              <dbl>
##  1      1   1.70                5
##  2      2  -0.212               3.29
##  3      3   0.357               4
##  4      4   0.624               4
##  5      5   0.00235             3.8
##  6      6   0.266               3.79
##  7      7   0.0321              3.90
##  8      8   0.196               3.39
##  9      9   0.0547              3.88
## 10     10   0.0624              3.77
## 11     11   0.609               4.22
## 12     12   0.143               3.49
## 13     13  -0.0227              3.22
## 14     14  -0.00922             3.40
## 15     16   0.215               3.8
## 16     17   0.410               4
## 17     18   0.0996              3.44
## 18     19   0.0242              3.68
## 19     22  -0.101               3.41
## 20     23   0.371               4.1
```

```
# Build the linear model (mean + bias of movie  + bias of user)
predicted_ratings_with2bias <- validacion %>%
  left_join(bias_movie, by = 'movieId') %>%
  left_join(bias_user, by = 'userId') %>%
  mutate(prediction = mean_rating + s_movie + s_usuario)

# Replace NA with the average value
```

```r
predicted_ratings_with2bias$prediction[is.na(predicted_ratings_with2bias$prediction)] <- mean_rating


# Evaluate the model and store the results in a tibble
valuation <- bind_rows(valuation,
                       tibble(Models = "Model linear mean/s_movie/s_usuario (validation)",
                       MAE = mae(validacion$rating, predicted_ratings_with2bias$prediction),
                       MSE = mse(validacion$rating, predicted_ratings_with2bias$prediction),
                       RMSE = rmse(validacion$rating, predicted_ratings_with2bias$prediction)))

# Print the evaluations
print(valuation)
```

```
##                                             Models       MAE       MSE
## 1         Mean baseline model (final_holdout_test) 0.8560101 1.1249798
## 2         Model linear mean + s_movie (validacion) 0.7381239 0.8913586
## 3 Model linear mean + s_movie (final_holdout_test) 0.7381881 0.8905784
## 4 Model linear mean/s_movie/s_usuario (validation) 0.6687190 0.7486483
##        RMSE
## 1 1.0606506
## 2 0.9441179
## 3 0.9437046
## 4 0.8652446
```

```r
# Final Evaluation with the final_holdout_test
bias_user <- edx %>%
  left_join(bias_movie, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(s_usuario = mean(rating - mean_rating - s_movie),
            s_usuario_isolated = mean(rating))

# Show the first 20 rows of user biases
head(bias_user, 20)
```

```
## # A tibble: 20 x 3
##    userId s_usuario s_usuario_isolated
##     <int>     <dbl>              <dbl>
##  1      1   1.71                5
##  2      2  -0.385               3.17
##  3      3   0.306               3.98
##  4      4   0.661               4.06
##  5      5   0.0190              3.84
##  6      6   0.322               3.90
##  7      7   0.0715              3.92
##  8      8   0.199               3.39
##  9      9   0.0477              3.85
## 10     10   0.0778              3.81
## 11     11   0.661               4.25
## 12     12   0.111               3.42
## 13     13   0.00454             3.24
## 14     14  -0.0245              3.37
## 15     16   0.186               3.74
## 16     17   0.310               3.95
```

```
## 17     18   0.123                3.47
## 18     19   0.0305               3.70
## 19     22  -0.105                3.42
## 20     23   0.424                4.21
```

```r
# Construct the linear model (mean + bias of movie + bias of user)
predicted_ratings_with2bias <- final_holdout_test %>%
  left_join(bias_movie, by = 'movieId') %>%
  left_join(bias_user, by = 'userId') %>%
  mutate(prediction = mean_rating + s_movie + s_usuario)

# Replace NA with the average value
predicted_ratings_with2bias$prediction[is.na(predicted_ratings_with2bias$prediction)] <- mean_rating

# Evaluate the model and store the results in a tibble
valuation <- bind_rows(valuation,
                       tibble(Models = "Model linear mean/s_movie/s_usuario (final_hold_out)",
                       MAE = mae(final_holdout_test$rating, predicted_ratings_with2bias$prediction),
                       MSE = mse(final_holdout_test$rating, predicted_ratings_with2bias$prediction),
                       RMSE = rmse(final_holdout_test$rating, predicted_ratings_with2bias$prediction)))

# Print the evaluations
print(valuation)
```

```
##                                                Models       MAE       MSE
## 1             Mean baseline model (final_holdout_test) 0.8560101 1.1249798
## 2             Model linear mean + s_movie (validacion) 0.7381239 0.8913586
## 3     Model linear mean + s_movie (final_holdout_test) 0.7381881 0.8905784
## 4     Model linear mean/s_movie/s_usuario (validation) 0.6687190 0.7486483
## 5 Model linear mean/s_movie/s_usuario (final_hold_out) 0.6694863 0.7491472
##        RMSE
## 1 1.0606506
## 2 0.9441179
## 3 0.9437046
## 4 0.8652446
## 5 0.8655329
```

```r
#MODELING WITH REGULARIZATION

#Regularization is a technique to improve the performance and generalizability of machine learning
#models by adding a penalty to the model's complexity. In recommender systems, regularization
#helps prevent overfitting by penalizing the complexity of user and item biases, leading to more
#accurate and robust recommendations.

install.packages("ggthemes")
```

```
## Installation du package dans 'C:/Users/nadda/AppData/Local/R/win-library/4.2'
## (car 'lib' n'est pas spécifié)

## le package 'ggthemes' a été décompressé et les sommes MD5 ont été vérifiées avec succés
##
## Les packages binaires téléchargés sont dans
##  C:\Users\nadda\AppData\Local\Temp\RtmpaETIyL\downloaded_packages
```

```r
install.packages("recosystem")
```

```
## Installation du package dans 'C:/Users/nadda/AppData/Local/R/win-library/4.2'
## (car 'lib' n'est pas spécifié)

## le package 'recosystem' a été décompressé et les sommes MD5 ont été vérifiées avec succés

## Warning: impossible de supprimer l'installation précédente du package
## 'recosystem'

## Warning in file.copy(savedcopy, lib, recursive = TRUE): problème lors de la
## copie de
## C:\Users\nadda\AppData\Local\R\win-library\4.2\00LOCK\recosystem\libs\x64\recosystem.dll
## vers
## C:\Users\nadda\AppData\Local\R\win-library\4.2\recosystem\libs\x64\recosystem.dll:
## Permission denied

## Warning: 'recosystem' restauré

##
## Les packages binaires téléchargés sont dans
##  C:\Users\nadda\AppData\Local\Temp\RtmpaETIyL\downloaded_packages
```

```r
install.packages("dplyr")
```

```
## Warning: le package 'dplyr' est en cours d'utilisation et ne sera pas installé
```

```r
install.packages("cowplot")
```

```
## Installation du package dans 'C:/Users/nadda/AppData/Local/R/win-library/4.2'
## (car 'lib' n'est pas spécifié)

## le package 'cowplot' a été décompressé et les sommes MD5 ont été vérifiées avec succés
##
## Les packages binaires téléchargés sont dans
##  C:\Users\nadda\AppData\Local\Temp\RtmpaETIyL\downloaded_packages
```

```r
install.packages("data.table")
```

```
## Installation du package dans 'C:/Users/nadda/AppData/Local/R/win-library/4.2'
## (car 'lib' n'est pas spécifié)

## le package 'data.table' a été décompressé et les sommes MD5 ont été vérifiées avec succés

## Warning: impossible de supprimer l'installation précédente du package
## 'data.table'
```

```
## Warning in file.copy(savedcopy, lib, recursive = TRUE): problème lors de la
## copie de
## C:\Users\nadda\AppData\Local\R\win-library\4.2\00LOCK\data.table\libs\x64\data_table.dll
## vers
## C:\Users\nadda\AppData\Local\R\win-library\4.2\data.table\libs\x64\data_table.dll:
## Permission denied

## Warning: 'data.table' restauré

##
## Les packages binaires téléchargés sont dans
##  C:\Users\nadda\AppData\Local\Temp\RtmpaETIyL\downloaded_packages
```

```
install.packages("Metrics")
```

```
## Warning: le package 'Metrics' est en cours d'utilisation et ne sera pas
## installé
```

```
# Loading the required libraries
library(ggthemes)
library(recosystem)
library(data.table)
```

```
##
## Attachement du package : 'data.table'

## Les objets suivants sont masqués depuis 'package:lubridate':
##
##     hour, isoweek, mday, minute, month, quarter, second, wday, week,
##     yday, year

## Les objets suivants sont masqués depuis 'package:dplyr':
##
##     between, first, last

## L'objet suivant est masqué depuis 'package:purrr':
##
##     transpose
```

```
library(dplyr)
library(cowplot)
```

```
##
## Attachement du package : 'cowplot'

## L'objet suivant est masqué depuis 'package:ggthemes':
##
##     theme_map

## L'objet suivant est masqué depuis 'package:lubridate':
##
##     stamp
```

```r
library(Metrics)

#compute_biases: This function calculates the average rating, movie biases, and user biases
#for a given dataset and lambda value.
compute_biases <- function(training, lambda) {
  avg_rating <- mean(training$rating)

  movie_b <- training %>%
    group_by(movieId) %>%
    summarize(s_movie = sum(rating - avg_rating) / (n() + lambda), .groups = 'drop')

  user_b <- training %>%
    left_join(movie_b, by = "movieId") %>%
    group_by(userId) %>%
    summarize(s_usuario = sum(rating - avg_rating - s_movie) / (n() + lambda), .groups = 'drop')

  return(list(movie_b = movie_b, user_b = user_b, avg_rating = avg_rating))
}

#This function predicts ratings based on the provided biases.
predict_ratings <- function(validacion, biases) {
  predictions <- validacion %>%
    left_join(biases$movie_b, by = "movieId") %>%
    left_join(biases$user_b, by = "userId") %>%
    mutate(pred = biases$avg_rating + s_movie + s_usuario) %>%
    pull(pred)

  predictions[is.na(predictions)] <- biases$avg_rating

  return(predictions)
}

#This function evaluates the model by computing the biases, predicting ratings,
#and calculating the RMSE for a given lambda value.
evaluate_model <- function(lambda, training, validacion) {
  biases <- compute_biases(training, lambda)
  predictions <- predict_ratings(validacion, biases)
  rmse_value <- rmse(validacion$rating, predictions)
  return(rmse_value)
}

# Ensure there are no missing values in the datasets
print("Checking for missing values in training dataset:")
```

```
## [1] "Checking for missing values in training dataset:"
```

```r
print(sum(is.na(training)))
```

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

```r
print("Checking for missing values in validation dataset:")
```

```
## [1] "Checking for missing values in validation dataset:"
```

```r
print(sum(is.na(validacion)))
```

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

```r
# Range of lambda values to test
lambda_values <- seq(0, 15, 0.20)

# Calculate RMSE for each lambda with debugging
rmse_values <- sapply(lambda_values, evaluate_model, training = training, validacion = validacion)

# Create a tibble to store lambda and RMSE values
rmse_results <- tibble(Lambda = lambda_values, RMSE = rmse_values)
print(rmse_results)
```

```
## # A tibble: 76 x 2
##    Lambda  RMSE
##     <dbl> <dbl>
##  1    0   0.867
##  2    0.2 0.866
##  3    0.4 0.866
##  4    0.6 0.866
##  5    0.8 0.866
##  6    1   0.866
##  7    1.2 0.866
##  8    1.4 0.866
##  9    1.6 0.866
## 10    1.8 0.866
## # i 66 more rows
```

```r
# Select the best lambda
best_lambda <- lambda_values[which.min(rmse_values)]
print(best_lambda)
```

```
## [1] 4.8
```

```r
# Compute biases with the best lambda
final_biases <- compute_biases(training, best_lambda)

# Predict ratings using the final biases
final_predictions <- predict_ratings(validacion, final_biases)

# Evaluate the model
valuation <- tibble(
  Models = "Linear model with regularized bias (validacion)",
  MAE = mae(validacion$rating, final_predictions),
  MSE = mse(validacion$rating, final_predictions),
```

```
  RMSE = rmse(validacion$rating, final_predictions)
)

print(valuation)
```

```
## # A tibble: 1 x 4
##   Models                                            MAE   MSE  RMSE
##   <chr>                                           <dbl> <dbl> <dbl>
## 1 Linear model with regularized bias (validacion) 0.670 0.750 0.866
```

```
# EVALUATION WITH FINAL_HOLDOUT_TEST same procedure but with the edx set and final_holdout_test

compute_biases <- function(edx, lambda) {
  avg_rating <- mean(edx$rating)

  movie_b <- edx %>%
    group_by(movieId) %>%
    summarize(s_movie = sum(rating - avg_rating) / (n() + lambda), .groups = 'drop')

  user_b <- edx %>%
    left_join(movie_b, by = "movieId") %>%
    group_by(userId) %>%
    summarize(s_usuario = sum(rating - avg_rating - s_movie) / (n() + lambda), .groups = 'drop')

  return(list(movie_b = movie_b, user_b = user_b, avg_rating = avg_rating))
}

predict_ratings <- function(final_holdout_test, biases) {
  predictions <- final_holdout_test %>%
    left_join(biases$movie_b, by = "movieId") %>%
    left_join(biases$user_b, by = "userId") %>%
    mutate(pred = biases$avg_rating + s_movie + s_usuario) %>%
    pull(pred)

  predictions[is.na(predictions)] <- biases$avg_rating

  return(predictions)
}

evaluate_model <- function(lambda, edx, final_holdout_test) {
  biases <- compute_biases(edx, lambda)
  predictions <- predict_ratings(final_holdout_test, biases)
  rmse_value <- rmse(final_holdout_test$rating, predictions)
  return(rmse_value)
}


# Ensure there are no missing values in the datasets
print("Checking for missing values in edx:")
```

```
## [1] "Checking for missing values in edx:"
```

```r
print(sum(is.na(edx)))
```

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

```r
print("Checking for missing values in final_holdout_test:")
```

```
## [1] "Checking for missing values in final_holdout_test:"
```

```r
print(sum(is.na(final_holdout_test)))
```

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

```r
# Range of lambda values to test
lambda_values <- seq(0, 15, 0.20)

# Calculate RMSE for each lambda with debugging
rmse_values <- sapply(lambda_values, evaluate_model, edx = edx, final_holdout_test = final_holdout_test)

# Create a tibble to store lambda and RMSE values
rmse_results <- tibble(Lambda = lambda_values, RMSE = rmse_values)
print(rmse_results)
```

```
## # A tibble: 76 x 2
##    Lambda  RMSE
##     <dbl> <dbl>
##  1    0   0.866
##  2    0.2 0.865
##  3    0.4 0.865
##  4    0.6 0.865
##  5    0.8 0.865
##  6    1   0.865
##  7    1.2 0.865
##  8    1.4 0.865
##  9    1.6 0.865
## 10    1.8 0.865
## # i 66 more rows
```

```r
# Select the best lambda
best_lambda <- lambda_values[which.min(rmse_values)]
print(best_lambda)
```

```
## [1] 5.4
```

```r
# Compute biases with the best lambda
final_biases <- compute_biases(edx, best_lambda)

# Predict ratings using the final biases
final_predictions <- predict_ratings(final_holdout_test, final_biases)
```

```r
# Evaluate the model
valuation <- tibble(
  Models = "Linear model with regularized bias (final_holdout_test)",
  MAE = mae(final_holdout_test$rating, final_predictions),
  MSE = mse(final_holdout_test$rating, final_predictions),
  RMSE = rmse(final_holdout_test$rating, final_predictions)
)

print(valuation)
```

```
## # A tibble: 1 x 4
##   Models                                                    MAE   MSE  RMSE
##   <chr>                                                   <dbl> <dbl> <dbl>
## 1 Linear model with regularized bias (final_holdout_test) 0.670 0.748 0.865
```

```r
#MATRIX DE FACTORIZATION
# Matrix factorization involves decomposing a large matrix into a product of two or more smaller
# matrices. In the context of recommender systems, this large matrix is typically the user-item
# interaction matrix, where each entry represents a user's rating for a particular item.

# The training and validacion sets need to be converted into recosystem input format

library(recosystem)
library(dplyr)
library(Metrics)

set.seed(1)

# Convert training and validacion sets into recosystem input format
entrenamiento <- with(training, data_memory(user_index = userId, item_index = movieId, rating = rating))
prueba <- with(validacion, data_memory(user_index = userId, item_index = movieId, rating = rating))

# Create the model object
create_model <- Reco()

# Define tuning options
tune_combination <- list(dim = c(10, 15, 20, 25), lrate = c(0.1, 0.2, 0.3), nthread = 4, niter = 10)

# Tune the model
best_tune <- create_model$tune(entrenamiento, opts = tune_combination)

# Train the model with the best tuning parameters
training_options <- c(best_tune$min, nthread = 4, niter = 20)
create_model$train(entrenamiento, opts = training_options)
```

```
## iter      tr_rmse          obj
##    0       0.9862   9.9224e+06
##    1       0.8771   8.0554e+06
##    2       0.8451   7.4934e+06
##    3       0.8235   7.1495e+06
##    4       0.8071   6.9037e+06
##    5       0.7947   6.7262e+06
##    6       0.7844   6.5892e+06
```

```
##    7       0.7754   6.4785e+06
##    8       0.7679   6.3888e+06
##    9       0.7612   6.3140e+06
##   10       0.7555   6.2481e+06
##   11       0.7505   6.1943e+06
##   12       0.7459   6.1478e+06
##   13       0.7420   6.1069e+06
##   14       0.7384   6.0720e+06
##   15       0.7352   6.0400e+06
##   16       0.7322   6.0108e+06
##   17       0.7296   5.9857e+06
##   18       0.7271   5.9626e+06
##   19       0.7248   5.9431e+06
```

```r
# Make predictions
predictions <- create_model$predict(prueba, out_memory())

# Calculate evaluation metrics
mae_value <- mae(validacion$rating, predictions)
mse_value <- mse(validacion$rating, predictions)
rmse_value <- rmse(validacion$rating, predictions)

# Add the results to the evaluation tibble
valuation <- valuation %>%
  add_row(Models = "Matrix of factorization (validacion)",
          MAE = mae_value,
          MSE = mse_value,
          RMSE = rmse_value)

# Print the evaluation results
print(valuation)
```

```
## # A tibble: 2 x 4
##   Models                                                  MAE   MSE  RMSE
##   <chr>                                                 <dbl> <dbl> <dbl>
## 1 Linear model with regularized bias (final_holdout_test) 0.670 0.748 0.865
## 2 Matrix of factorization (validacion)                    0.610 0.628 0.792
```

```r
#Final Evaluation with the final_holdout_test set

set.seed(1)

# Convert training and validacion sets into recosystem input format
entrenamiento <- with(edx, data_memory(user_index = userId, item_index = movieId, rating = rating))
prueba <- with(final_holdout_test, data_memory(user_index = userId, item_index = movieId, rating = rati

# Create the model object
create_model <- Reco()

# Define tuning options
tune_combination <- list(dim = c(10, 15, 20, 25), lrate = c(0.1, 0.2, 0.3), nthread = 4, niter = 10)

# Tune the model
```

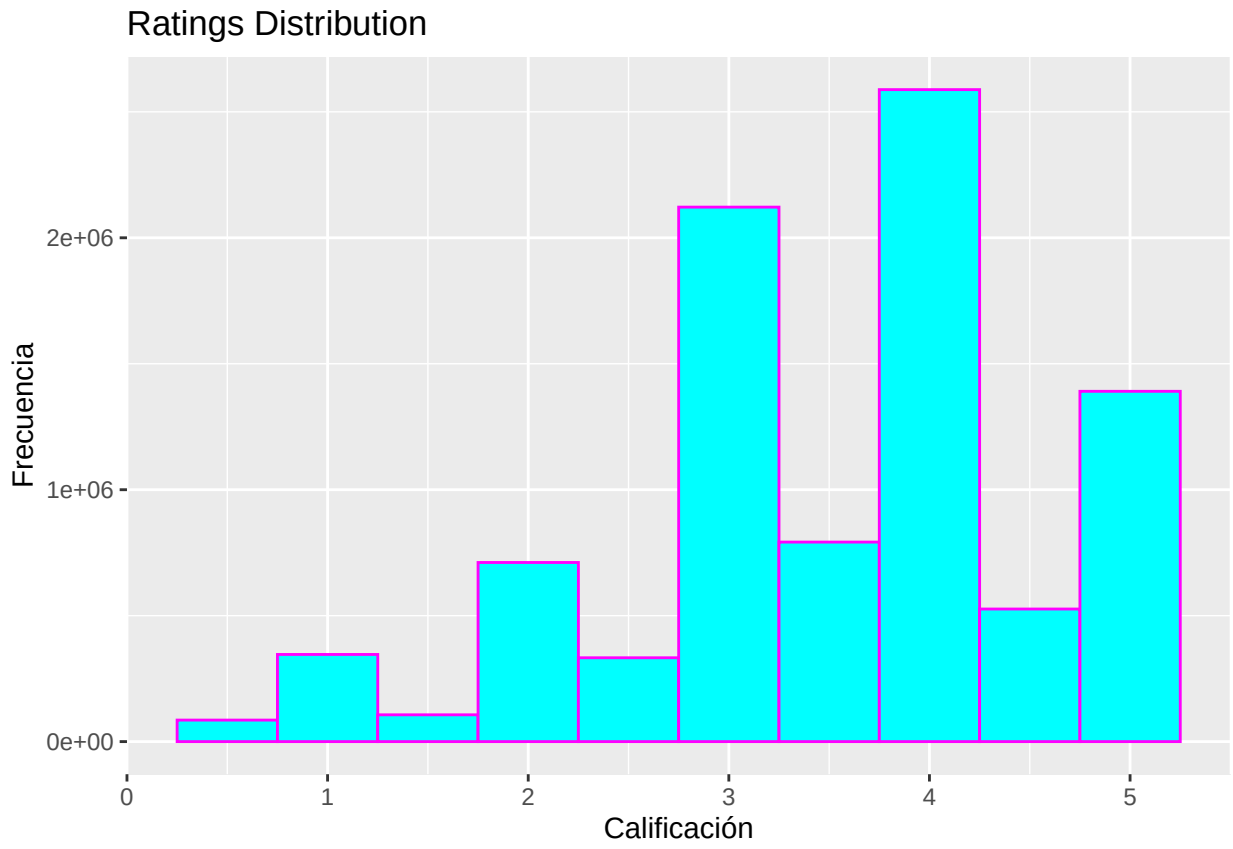
```
best_tune <- create_model$tune(entrenamiento, opts = tune_combination)

# Train the model with the best tuning parameters
training_options <- c(best_tune$min, nthread = 4, niter = 20)
create_model$train(entrenamiento, opts = training_options)
```

```
## iter      tr_rmse          obj
##    0       0.9690   1.1928e+07
##    1       0.8714   9.8660e+06
##    2       0.8385   9.1714e+06
##    3       0.8172   8.7525e+06
##    4       0.8022   8.4798e+06
##    5       0.7910   8.2883e+06
##    6       0.7816   8.1429e+06
##    7       0.7735   8.0208e+06
##    8       0.7669   7.9229e+06
##    9       0.7614   7.8455e+06
##   10       0.7565   7.7798e+06
##   11       0.7522   7.7228e+06
##   12       0.7484   7.6787e+06
##   13       0.7450   7.6339e+06
##   14       0.7420   7.5980e+06
##   15       0.7392   7.5655e+06
##   16       0.7367   7.5359e+06
##   17       0.7345   7.5120e+06
##   18       0.7322   7.4856e+06
##   19       0.7303   7.4644e+06
```

```
# Make predictions
predictions <- create_model$predict(prueba, out_memory())

# Calculate evaluation metrics
mae_value <- mae(final_holdout_test$rating, predictions)
mse_value <- mse(final_holdout_test$rating, predictions)
rmse_value <- rmse(final_holdout_test$rating, predictions)

# Add the results to the evaluation tibble
valuation <- valuation %>%
  add_row(Models = "Matrix of factorization (final_holdout_test)",
          MAE = mae_value,
          MSE = mse_value,
          RMSE = rmse_value)

# Print the evaluation results
print(valuation)
```

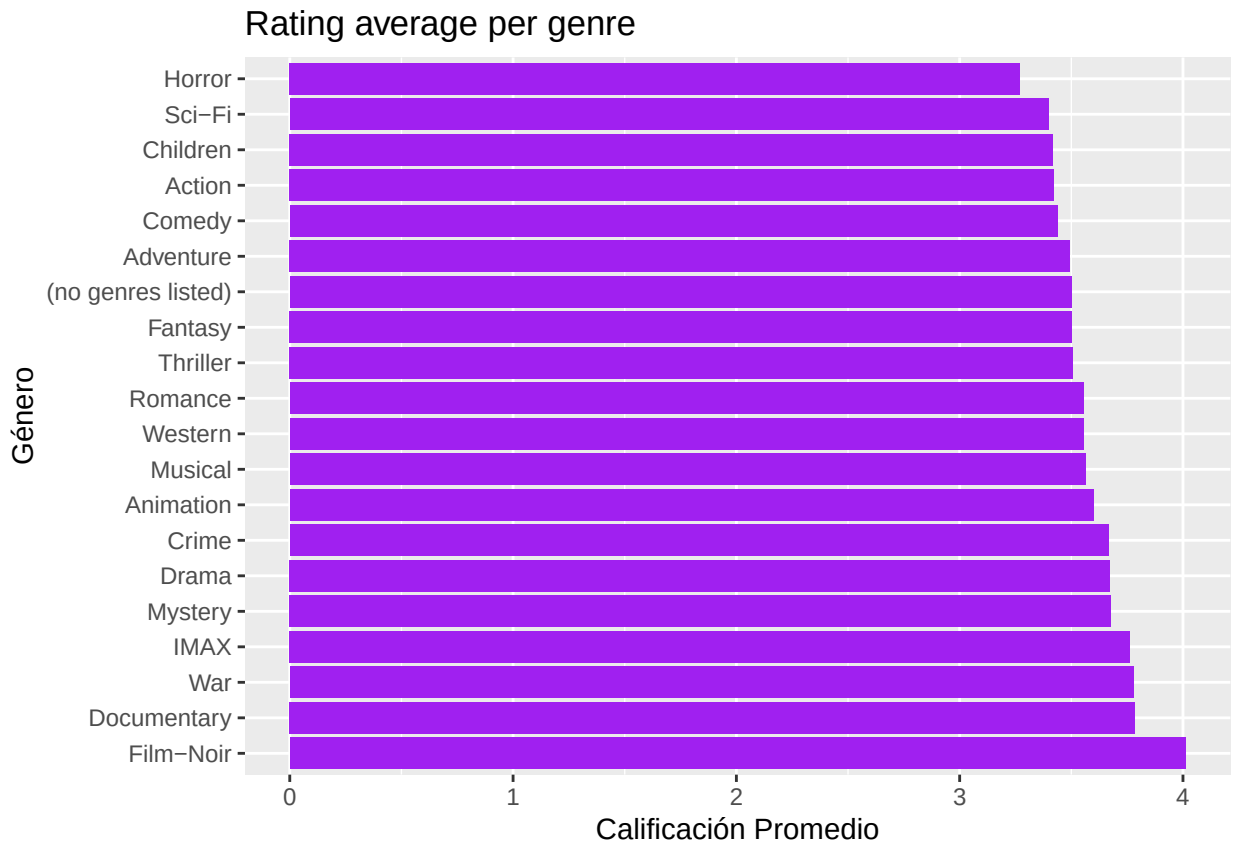
```
## # A tibble: 3 x 4
##   Models                                                MAE   MSE  RMSE
##   <chr>                                               <dbl> <dbl> <dbl>
## 1 Linear model with regularized bias (final_holdout_test) 0.670 0.748 0.865
## 2 Matrix of factorization (validacion)                0.610 0.628 0.792
## 3 Matrix of factorization (final_holdout_test)        0.605 0.617 0.785
```

```r
# Exploratory Data Analysis EDA
# Ratings Distribution
# The plot shows that most movie ratings are high, with 4 being the most frequent, followed by 3
# and 5.Lower ratings (1 and 2) are much less common, indicating that users generally rate movies posit


library(ggplot2)

ggplot(edx, aes(x = rating)) +
  geom_histogram(binwidth = 0.5, fill = "cyan", color = "magenta") +
  ggtitle("Ratings Distribution") +
  xlab("Calificación") +
  ylab("Frecuencia")
```




```r
# Rating Average per genre
# The plot shows the average movie rating for different genres. Horror, Sci-Fi, and Children
# genres have the highest average ratings, indicating they are well-received by viewers.
# On the other hand, Film-Noir, Documentary, and War genres have the lowest average ratings,
# suggesting they are less popular among the audience. Overall, the plot highlights significant
# differences in viewer preferences across various movie genres.

edx_genres <- edx %>%
  separate_rows(genres, sep = "\\|")

average_rating_per_genre <- edx_genres %>%
```

```r
  group_by(genres) %>%
  summarize(avg_rating = mean(rating))

ggplot(average_rating_per_genre, aes(x = reorder(genres, -avg_rating), y = avg_rating)) +
  geom_bar(stat = "identity", fill = "purple") +
  coord_flip() +
  ggtitle("Rating average per genre ") +
  xlab("Género") +
  ylab("Calificación Promedio")
```




```r
# Rating per year

library(dplyr)
library(stringr)

# This plot highlights how the average ratings of movies have changed over time, showing that
# starting from the mid-1970s, there is a noticeable decline in average ratings, dropping
# below 3.8 compared to earlier decades.

# Extract the year from the movie titles
edx <- edx %>%
  mutate(year = str_extract(title, "\\(\\d{4}\\)")) %>%
  mutate(year = as.numeric(str_replace_all(year, "[()]", "")))

# Have a look if the years were extracted correctly
```

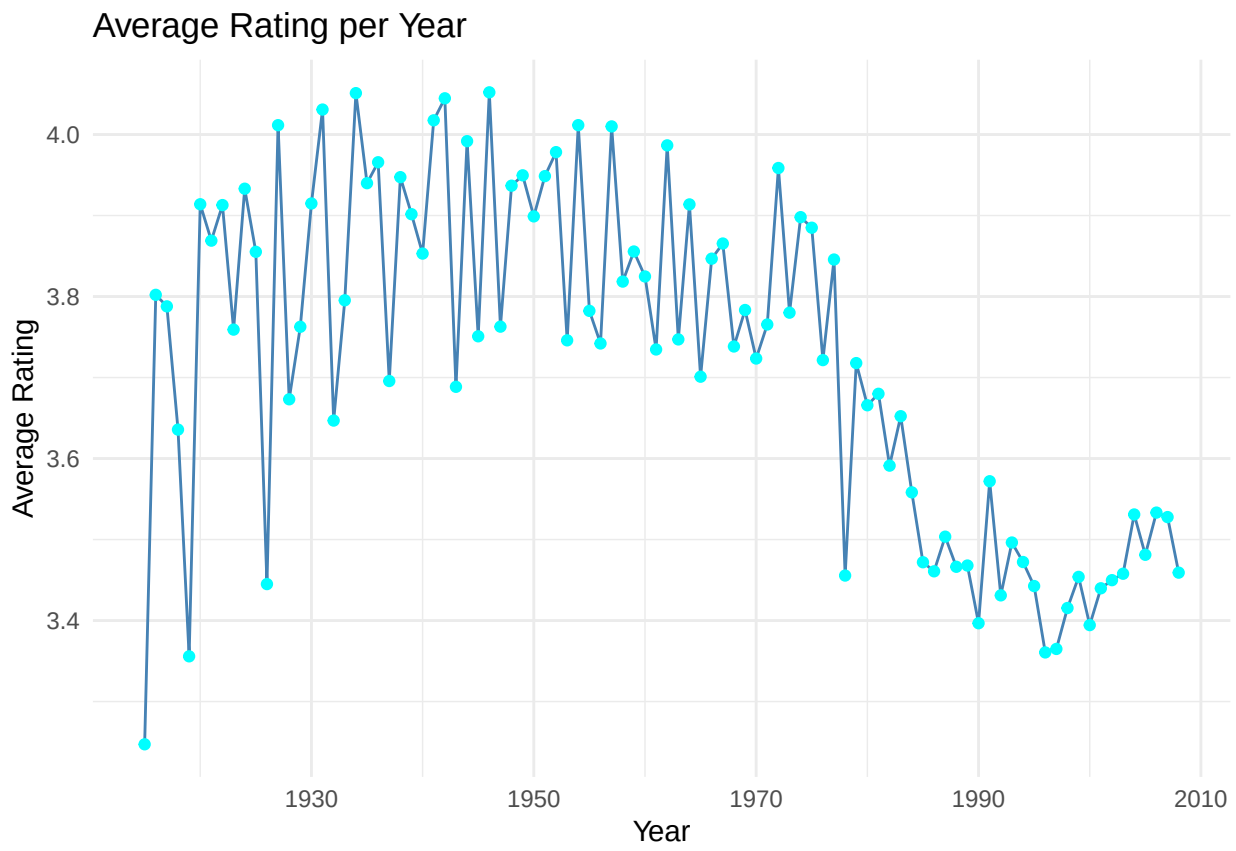
```r
head(edx)
```

```
##   userId movieId rating timestamp                          title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 3      1     231      5 838983392          Dumb & Dumber (1994)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
##                          genres year
## 1                Comedy|Romance 1992
## 2         Action|Crime|Thriller 1995
## 3                        Comedy 1994
## 4  Action|Drama|Sci-Fi|Thriller 1995
## 5       Action|Adventure|Sci-Fi 1994
## 6 Action|Adventure|Drama|Sci-Fi 1994
```

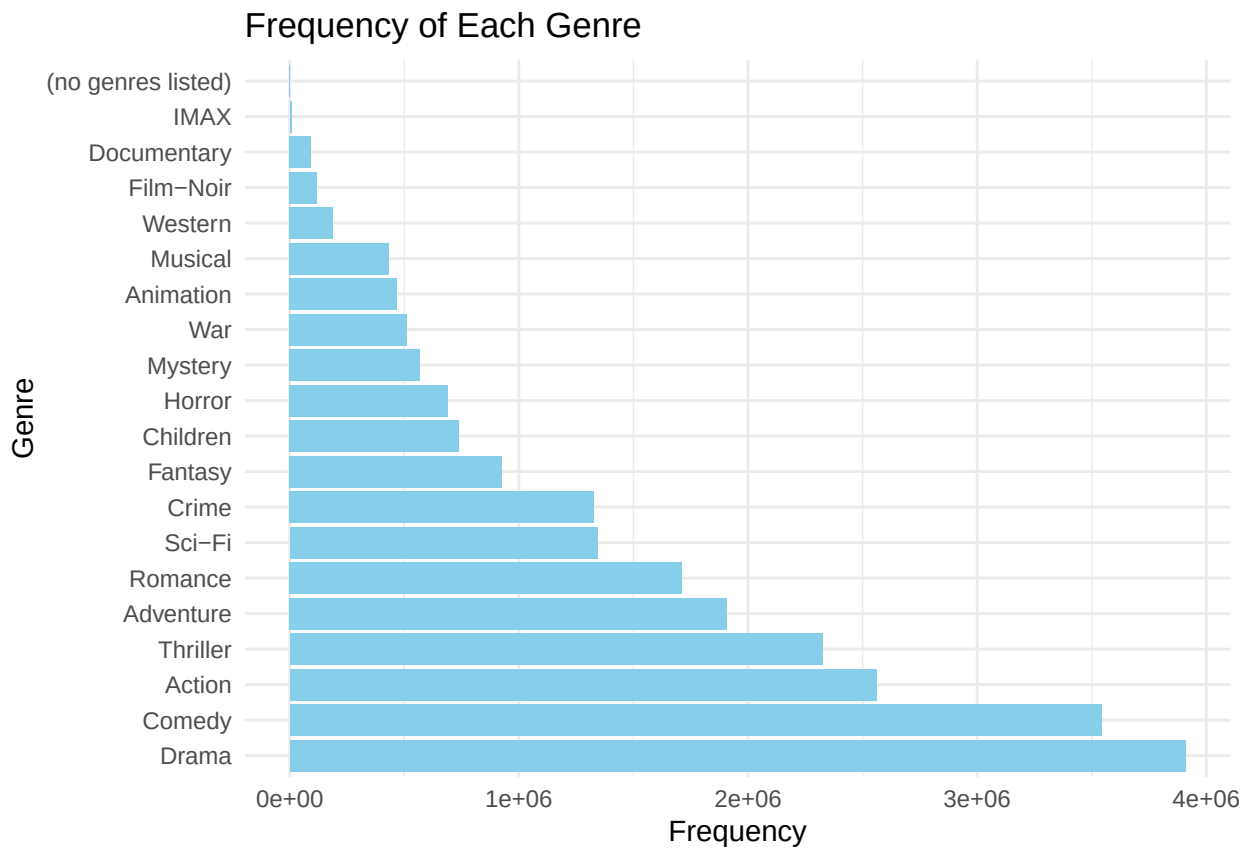
```r
# Calculate the average rating per year
average_rating_per_year <- edx %>%
  group_by(year) %>%
  summarize(avg_rating = mean(rating, na.rm = TRUE), count = n())

# Check the results
head(average_rating_per_year)
```

```
## # A tibble: 6 x 3
##    year avg_rating count
##   <dbl>      <dbl> <int>
## 1  1915       3.25   180
## 2  1916       3.80    96
## 3  1917       3.79    33
## 4  1918       3.64    70
## 5  1919       3.36   163
## 6  1920       3.91   575
```

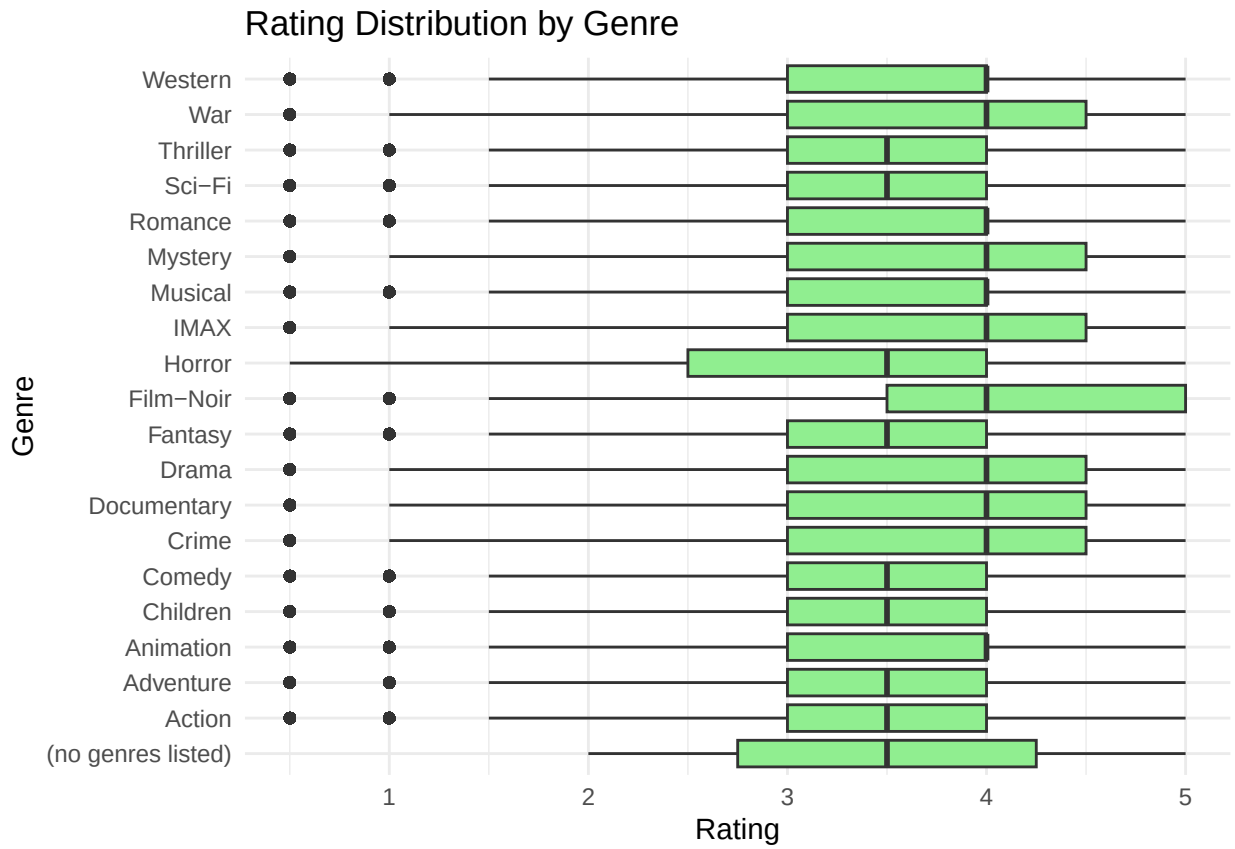
```r
library(ggplot2)

# Create the plot of average rating per year
ggplot(average_rating_per_year, aes(x = year, y = avg_rating)) +
  geom_line(color = "steelblue") +
  geom_point(color = "cyan") +
  ggtitle("Average Rating per Year") +
  xlab("Year") +
  ylab("Average Rating") +
  theme_minimal()
```



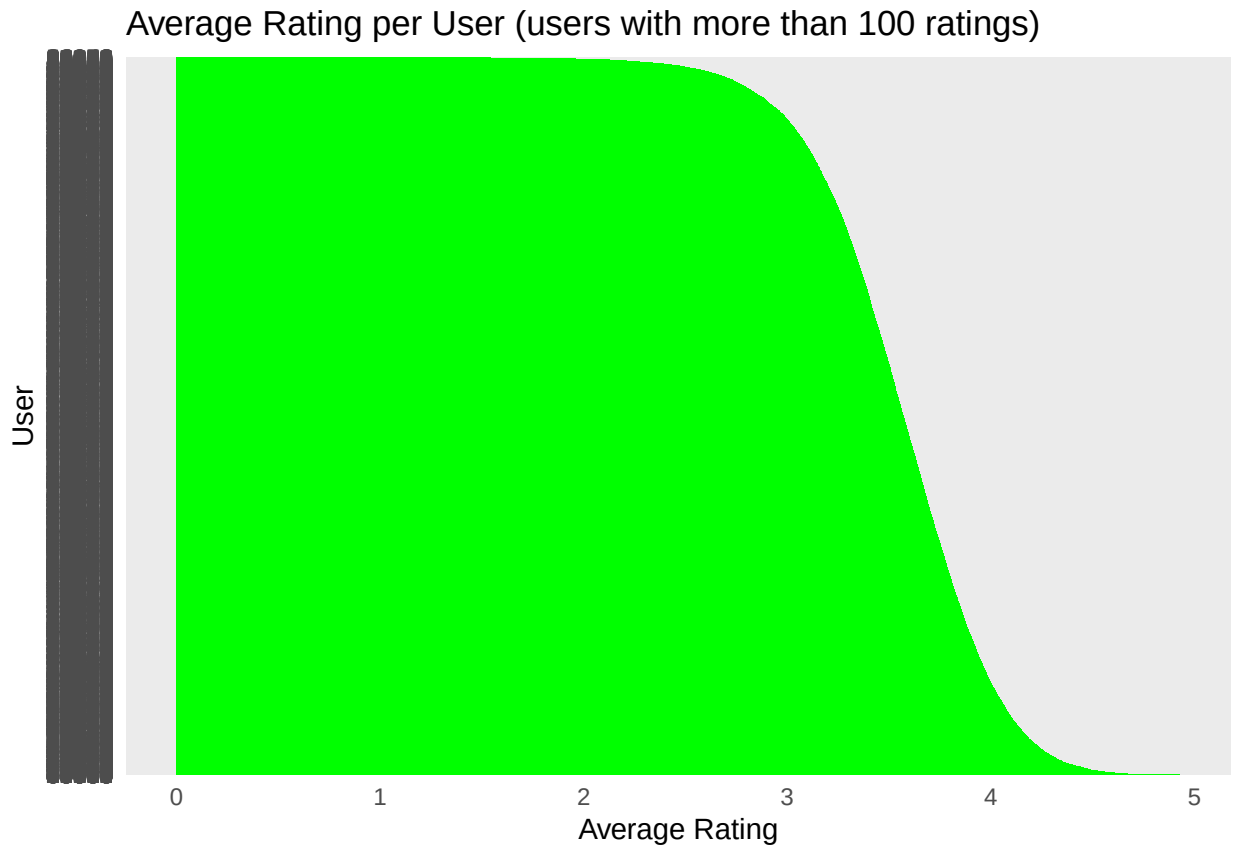

```
# Genre Distribution
# Count the frequency of each genre
# This plot highlights the distribution of movie genres in the dataset, showing which genres
# are most and least frequent based on user ratings.

genre_count <- edx %>%
  separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>%
  summarize(count = n()) %>%
  arrange(desc(count))

# Create the bar chart
ggplot(genre_count, aes(x = reorder(genres, -count), y = count)) +
  geom_bar(stat = "identity", fill = "skyblue") +
  coord_flip() +
  ggtitle("Frequency of Each Genre") +
  xlab("Genre") +
  ylab("Frequency") +
  theme_minimal()
```



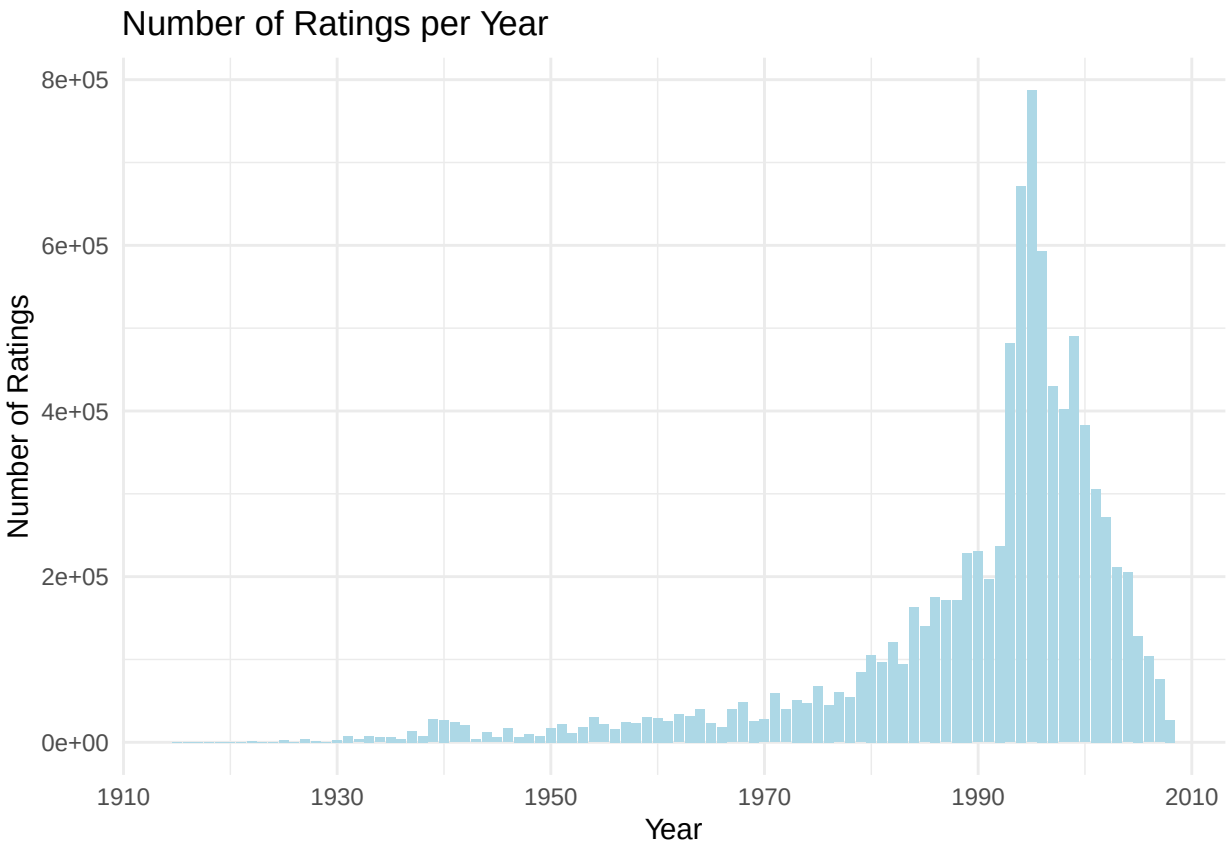

```
# Rating Distribution by Genre
# The box plots provide a visual summary of the central tendency, dispersion, and skewness of the ratin
# for each genre. For example, genres like "IMAX" and "Film-Noir" show a wider range of ratings with
# some outliers, indicating more variability in how these movies are rated. Genres like "Action" and
# "Comedy" have a more concentrated range of ratings with fewer outliers, indicating more consistent
# ratings within these genres.

# The median rating for most genres appears to be around 3 to 4, showing that the central tendency of
# ratings is relatively high across genres.

# Create a boxplot of rating distribution by genre
ggplot(edx_genres, aes(x = genres, y = rating)) +
  geom_boxplot(fill = "lightgreen") +
  coord_flip() +
  ggtitle("Rating Distribution by Genre") +
  xlab("Genre") +
  ylab("Rating") +
  theme_minimal()
```




```
# Average Rating per User
# This plot highlights that users who rate many movies tend to give ratings that are generally
# higher, clustering around the 3-4 range, indicating a tendency towards positive ratings.

# Calculate the average rating per user
average_rating_per_user <- edx %>%
  group_by(userId) %>%
  summarize(avg_rating = mean(rating), count = n()) %>%
  arrange(desc(count)) %>%
  filter(count > 100)  # Filter for users with more than 100 ratings

# Create the bar chart
ggplot(average_rating_per_user, aes(x = reorder(userId, -avg_rating), y = avg_rating)) +
  geom_bar(stat = "identity", fill = "green") +
  coord_flip() +
  ggtitle("Average Rating per User (users with more than 100 ratings)") +
  xlab("User") +
  ylab("Average Rating") +
  theme_minimal()
```




```
# Number of Ratings per Year
# This plot helps to understand the temporal distribution of movie ratings, showing which
# periods have the highest and lowest number of ratings and providing insights into trends
# in movie popularity over time.

# Calculate the number of ratings per year
ratings_per_year <- edx %>%
  group_by(year) %>%
  summarize(count = n())

# Create the bar chart
ggplot(ratings_per_year, aes(x = year, y = count)) +
  geom_bar(stat = "identity", fill = "lightblue") +
  ggtitle("Number of Ratings per Year") +
  xlab("Year") +
  ylab("Number of Ratings") +
  theme_minimal()
```




```
# EDA Eploratory Data Analysis
install.packages("skimr")
```

```
## Installation du package dans 'C:/Users/nadda/AppData/Local/R/win-library/4.2'
## (car 'lib' n'est pas spécifié)
```

```
## le package 'skimr' a été décompressé et les sommes MD5 ont été vérifiées avec succés
##
## Les packages binaires téléchargés sont dans
##  C:\Users\nadda\AppData\Local\Temp\RtmpaETIyL\downloaded_packages
```

```
library(skimr)

# Summary of edx data
summary(edx)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18122   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35743   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35869   Mean   : 4120   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53602   3rd Qu.: 3624   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres              year
```

```
##  Length:9000061     Length:9000061     Min.   :1915
##  Class :character   Class :character   1st Qu.:1987
##  Mode  :character   Mode  :character   Median :1994
##                                        Mean   :1990
##                                        3rd Qu.:1998
##                                        Max.   :2008
```

```r
# This skim(edx) provides a comprehensive overview of the edx dataset, including the distribution
# and completeness of the data across different variables. It highlights key statistics such as
# the mean, median, and percentiles, helping to understand the data's structure and characteristics.
# Summary using skimr for more details
skim(edx)
```

Table 1: Data summary

Name	edx
Number of rows	9000061
Number of columns	7
Column type frequency:	
character	2
numeric	5
Group variables	None

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
title	0	1	8	160	0	10676	0
genres	0	1	3	60	0	797	0

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
userId	0	1	3.586890e+04	20583.75	1.0	18122	35743	53602	71567	
movieId	0	1	4.120230e+03	8939.16	1.0	648	1834	3624	65133	
rating	0	1	3.510000e+00	1.06	0.5	3	4	4	5	
timestamp	0	1	1.032587e+09	195963032.76	789652009.0	946763211.0	1035425747.0	1126667472.0	1231131303	
year	0	1	1.990220e+03	13.59	1915.0	1987	1994	1998	2008	

```r
# Distribution of ratings
ratings_summary <- edx %>%
  group_by(rating) %>%
  summarise(count = n()) %>%
  arrange(desc(count))

print(ratings_summary)
```

```
## # A tibble: 10 x 2
##    rating   count
##     <dbl>   <int>
##  1    4   2588021
##  2    3   2121638
##  3    5   1390541
##  4    3.5  792037
##  5    2    710998
##  6    4.5  526309
##  7    1    345935
##  8    2.5  332783
##  9    1.5  106379
## 10    0.5   85420
```

```
#  Number of ratings per movie:

# Count ratings per movie
ratings_per_movie <- edx %>%
  group_by(movieId) %>%
  summarise(count = n()) %>%
  arrange(desc(count))

# Summary statistics for ratings per movie
ratings_per_movie_summary <- ratings_per_movie %>%
  summarise(
    min = min(count),
    q1 = quantile(count, 0.25),
    median = median(count),
    mean = mean(count),
    q3 = quantile(count, 0.75),
    max = max(count)
  )

print(ratings_per_movie_summary)
```

```
## # A tibble: 1 x 6
##     min    q1 median  mean    q3   max
##   <int> <dbl>  <int> <dbl> <dbl> <int>
## 1     1    30    122  843.   563 31336
```

```
#  Number of ratings per user:

# Count ratings per user
ratings_per_user <- edx %>%
  group_by(userId) %>%
  summarise(count = n()) %>%
  arrange(desc(count))

# Summary statistics for ratings per user
ratings_per_user_summary <- ratings_per_user %>%
  summarise(
    min = min(count),
    q1 = quantile(count, 0.25),
```

```
    median = median(count),
    mean = mean(count),
    q3 = quantile(count, 0.75),
    max = max(count)
  )

print(ratings_per_user_summary)
```

```
## # A tibble: 1 x 6
##     min    q1 median  mean    q3   max
##   <int> <dbl>  <dbl> <dbl> <dbl> <int>
## 1    13    32     62  129.   140  6637
```

```
# Average ratings by genre:


# Separate genres into individual columns
edx_genres <- edx %>% separate_rows(genres, sep = "\\|")

# Average ratings by genre
avg_ratings_genres <- edx_genres %>%
  group_by(genres) %>%
  summarise(avg_rating = mean(rating), count = n()) %>%
  filter(count > 1000) %>%
  arrange(desc(avg_rating)) # Filter genres with less than 1000 ratings and sort by average rating

print(avg_ratings_genres)
```

```
## # A tibble: 19 x 3
##    genres      avg_rating   count
##    <chr>            <dbl>   <int>
##  1 Film-Noir         4.01  118394
##  2 Documentary       3.78   93252
##  3 War               3.78  511330
##  4 IMAX              3.76    8190
##  5 Mystery           3.68  567865
##  6 Drama             3.67 3909401
##  7 Crime             3.67 1326917
##  8 Animation         3.60  467220
##  9 Musical           3.56  432960
## 10 Western           3.56  189234
## 11 Romance           3.55 1712232
## 12 Thriller          3.51 2325349
## 13 Fantasy           3.50  925624
## 14 Adventure         3.49 1908692
## 15 Comedy            3.44 3541284
## 16 Action            3.42 2560649
## 17 Children          3.42  737851
## 18 Sci-Fi            3.40 1341750
## 19 Horror            3.27  691407
```

```r
# Movie Popularity and Rating Analysis

# Most rated movies
most_rated_movies <- edx %>%
  group_by(movieId, title) %>%
  summarise(count = n()) %>%
  arrange(desc(count)) %>%
  slice(1:10)
```

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

```r
print(most_rated_movies)
```

```
## # A tibble: 10,677 x 3
## # Groups:   movieId [10,677]
##    movieId title                                count
##      <int> <chr>                                <int>
##  1       1 Toy Story (1995)                     23826
##  2       2 Jumanji (1995)                       10717
##  3       3 Grumpier Old Men (1995)               7053
##  4       4 Waiting to Exhale (1995)              1579
##  5       5 Father of the Bride Part II (1995)    6415
##  6       6 Heat (1995)                          12385
##  7       7 Sabrina (1995)                        7273
##  8       8 Tom and Huck (1995)                    811
##  9       9 Sudden Death (1995)                   2280
## 10      10 GoldenEye (1995)                     15250
## # i 10,667 more rows
```

```r
# Least rated movies
least_rated_movies <- edx %>%
  group_by(movieId, title) %>%
  summarise(count = n()) %>%
  arrange(count) %>%
  slice(1:10)
```

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

```r
print(least_rated_movies)
```

```
## # A tibble: 10,677 x 3
## # Groups:   movieId [10,677]
##    movieId title                                count
##      <int> <chr>                                <int>
##  1       1 Toy Story (1995)                     23826
##  2       2 Jumanji (1995)                       10717
##  3       3 Grumpier Old Men (1995)               7053
##  4       4 Waiting to Exhale (1995)              1579
##  5       5 Father of the Bride Part II (1995)    6415
```

```
##  6       6 Heat (1995)                         12385
##  7       7 Sabrina (1995)                       7273
##  8       8 Tom and Huck (1995)                   811
##  9       9 Sudden Death (1995)                  2280
## 10      10 GoldenEye (1995)                    15250
## # i 10,667 more rows
```

```
# Average rating per movie
average_rating_per_movie <- edx %>%
  group_by(movieId, title) %>%
  summarise(avg_rating = mean(rating), count = n()) %>%
  arrange(desc(avg_rating))
```

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

```
print(average_rating_per_movie)
```

```
## # A tibble: 10,677 x 4
## # Groups:   movieId [10,677]
##    movieId title                                                avg_rating count
##      <int> <chr>                                                     <dbl> <int>
##  1    3226 Hellhounds on My Trail (1999)                             5         1
##  2   33264 Satan's Tango (Sátántangó) (1994)                         5         2
##  3   42783 Shadows of Forgotten Ancestors (1964)                     5         1
##  4   51209 Fighting Elegy (Kenka erejii) (1966)                      5         1
##  5   53355 Sun Alley (Sonnenallee) (1999)                            5         1
##  6   64275 Blue Light, The (Das Blaue Licht) (1932)                  5         1
##  7   65001 Constantine's Sword (2007)                                5         1
##  8   26048 Human Condition II, The (Ningen no joken II) (1959)       4.83      3
##  9    5194 Who's Singin' Over There? (a.k.a. Who Sings Over Th~      4.75      4
## 10   26073 Human Condition III, The (Ningen no joken III) (196~      4.75      4
## # i 10,667 more rows
```

```
# Average rating per user
average_rating_per_user <- edx %>%
  group_by(userId) %>%
  summarise(avg_rating = mean(rating), count = n()) %>%
  arrange(desc(avg_rating))

print(average_rating_per_user)
```

```
## # A tibble: 69,878 x 3
##    userId avg_rating count
##     <int>      <dbl> <int>
##  1      1          5    21
##  2   1686          5    21
##  3   7984          5    18
##  4  11884          5    19
##  5  13027          5    31
##  6  13513          5    19
##  7  13524          5    19
```

```
##  8  15575          5    28
##  9  18965          5    50
## 10  22045          5    22
## # i 69,868 more rows
```

```r
# Number of movies and ratings per genre
movies_per_genre <- edx_genres %>%
  group_by(genres) %>%
  summarise(num_movies = n_distinct(movieId), num_ratings = n()) %>%
  arrange(desc(num_ratings))

print(movies_per_genre)
```

```
## # A tibble: 20 x 3
##    genres             num_movies num_ratings
##    <chr>                   <int>       <int>
##  1 Drama                    5336     3909401
##  2 Comedy                   3703     3541284
##  3 Action                   1473     2560649
##  4 Thriller                 1705     2325349
##  5 Adventure                1025     1908692
##  6 Romance                  1685     1712232
##  7 Sci-Fi                    754     1341750
##  8 Crime                    1117     1326917
##  9 Fantasy                   543      925624
## 10 Children                  528      737851
## 11 Horror                   1013      691407
## 12 Mystery                   509      567865
## 13 War                       510      511330
## 14 Animation                 286      467220
## 15 Musical                   436      432960
## 16 Western                   275      189234
## 17 Film-Noir                 148      118394
## 18 Documentary               481       93252
## 19 IMAX                       29        8190
## 20 (no genres listed)          1           6
```

```r
# Popularity and average rating by genre over time
genre_trends <- edx_genres %>%
  mutate(year = format(as.POSIXct(timestamp, origin = "1970-01-01"), "%Y")) %>%
  group_by(genres, year) %>%
  summarise(count = n(), avg_rating = mean(rating)) %>%
  arrange(genres, year)
```

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

```r
print(genre_trends)
```

```
## # A tibble: 273 x 4
## # Groups:   genres [20]
##    genres             year   count avg_rating
```

```
##    <chr>              <chr>  <int>      <dbl>
##  1 (no genres listed) 2004       2       2.75
##  2 (no genres listed) 2007       3       4
##  3 (no genres listed) 2008       1       3.5
##  4 Action             1995       1       3
##  5 Action             1996  298067       3.44
##  6 Action             1997  116638       3.54
##  7 Action             1998   49472       3.44
##  8 Action             1999  175347       3.45
##  9 Action             2000  297280       3.47
## 10 Action             2001  181575       3.46
## # i 263 more rows
```

```r
# Trends in the number of ratings over time
# Ratings per month
ratings_per_month <- edx %>%
  mutate(month = format(as.POSIXct(timestamp, origin = "1970-01-01"), "%Y-%m")) %>%
  group_by(month) %>%
  summarise(count = n()) %>%
  arrange(month)

print(ratings_per_month)
```

```
## # A tibble: 157 x 2
##    month    count
##    <chr>    <int>
##  1 1995-01      2
##  2 1996-01     18
##  3 1996-02    690
##  4 1996-03   5503
##  5 1996-04  31648
##  6 1996-05 111140
##  7 1996-06 148606
##  8 1996-07 128038
##  9 1996-08 129006
## 10 1996-09  87712
## # i 147 more rows
```

```r
# Check for missing values
missing_values <- sapply(edx, function(x) sum(is.na(x)))

print(missing_values)
```

```
##    userId   movieId    rating timestamp     title    genres      year
##         0         0         0         0         0         0         0
```