HarvardX Professional Certificate in Data Science PH125.9x: Capstone Project for MovieLens

Tan Wei Lun

2025-06-19

1. Introduction

This project focuses on developing a movie recommendation system using the MovieLens 10M dataset. The main objective is to train a machine learning algorithm using the inputs in one subset to predict movie ratings in the validation set. Model performance is assessed using Root Mean Square Error (RMSE). A series of models are built incrementally, beginning with a simple baseline and progressing to a regularized model that accounts for both movie and user effects. The final model is evaluated on a separate hold-out test set to assess its predictive accuracy.

2. Data Preparation

The MovieLens 10M dataset was split into two parts: the edx set and the final_holdout_test set. All model development, including training, tuning, and validation, was carried out using the edx set. The final_holdout_test set was kept separate and untouched during this process to ensure a fair and unbiased evaluation of the final model.

RMSE (Root Mean Square Error) was used as the main metric to assess prediction accuracy. To support model development, the edx data was further divided into training and test subsets, and cross-validation was applied where necessary. This allowed for thorough testing and refinement of the algorithm without relying on the final test set, which was reserved solely for the final performance evaluation.

Create edx and final_holdout_test sets:

```
# 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"
```

```r
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")
```

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

```r
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)`
```

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

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

3. Exploratory Data Analysis (EDA)

We begin by performing an exploratory data analysis (EDA) to better understand the structure and key features of the edx dataset. This includes looking at the dataset's size, how ratings are distributed, the number of unique users and movies, rating patterns by movie and user, genre frequencies, and the use of half-star versus whole-star ratings. These insights help identify important trends and potential challenges, providing a solid foundation for the modeling process that follows.

3.1 Dimensions of the edx Dataset

The first step is to examine the size of the edx dataset by checking its dimensions, that is, the number of rows (observations) and columns (variables). This is done using the dim() function, which gives a basic overview of the dataset's overall scale.

```
dim(edx)
```

```
## [1] 9000055       6
```

3.2 Frequency of All Ratings

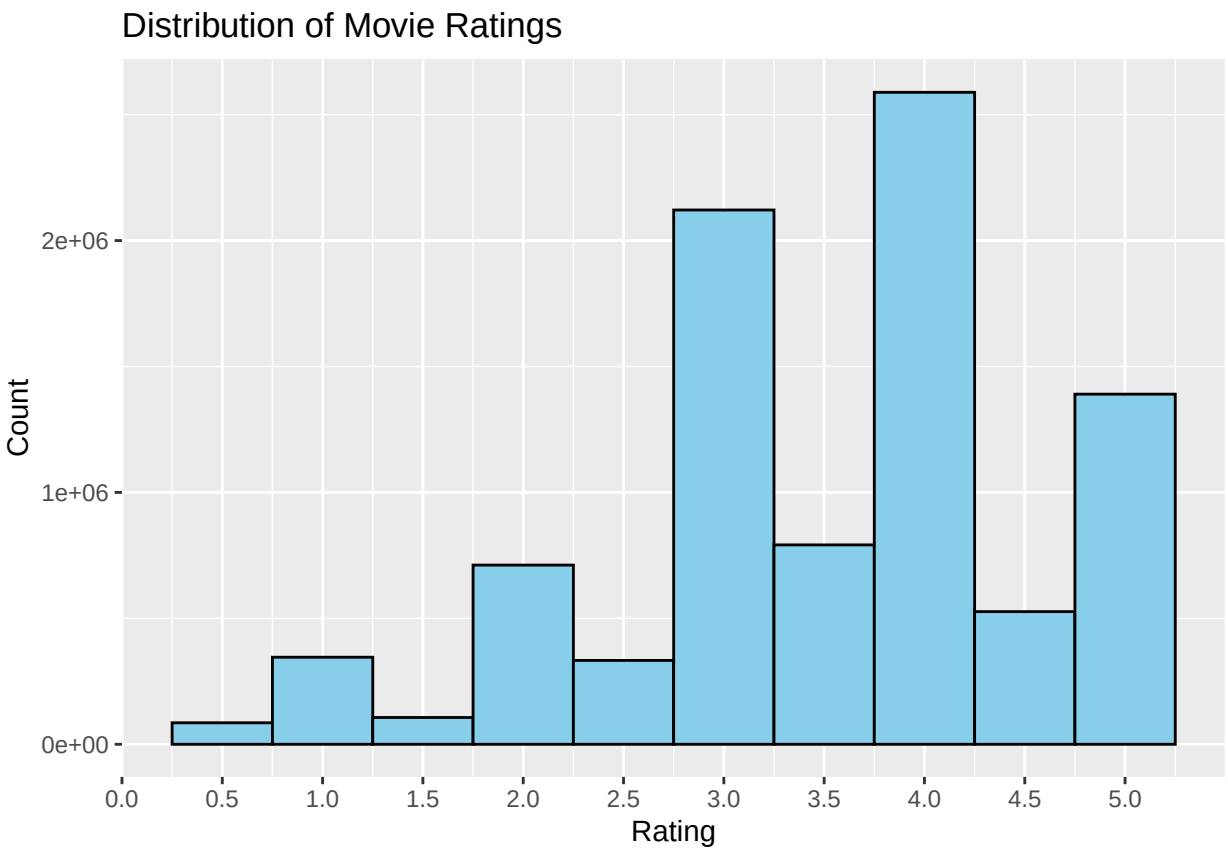
To examine how the ratings are distributed, We count how often each rating appears and sort the results in ascending order to see how the frequencies change across the scale.

```
edx %>%
  filter(rating >= 0 & rating <= 5) %>%
  count(rating) %>%
  arrange(rating)
```

```
##    rating       n
## 1     0.5   85374
## 2     1.0  345679
## 3     1.5  106426
## 4     2.0  711422
## 5     2.5  333010
## 6     3.0 2121240
## 7     3.5  791624
## 8     4.0 2588430
## 9     4.5  526736
## 10    5.0 1390114
```

The histogram is create to visualize the rating distribution.

```
edx %>%
  ggplot(aes(x = rating)) +
  geom_histogram(binwidth = 0.5, fill = "skyblue", color = "black") +
  scale_x_continuous(breaks = seq(0, 5, 0.5)) +
  labs(title = "Distribution of Movie Ratings",
       x = "Rating",
       y = "Count")
```



3.3 Number of Unique Movies and Users

The analysis continues by counting the number of unique movieId and userId values to understand the diversity of movies and users in the dataset. This is done using the n_distinct() function.

```
n_distinct(edx$movieId)
```

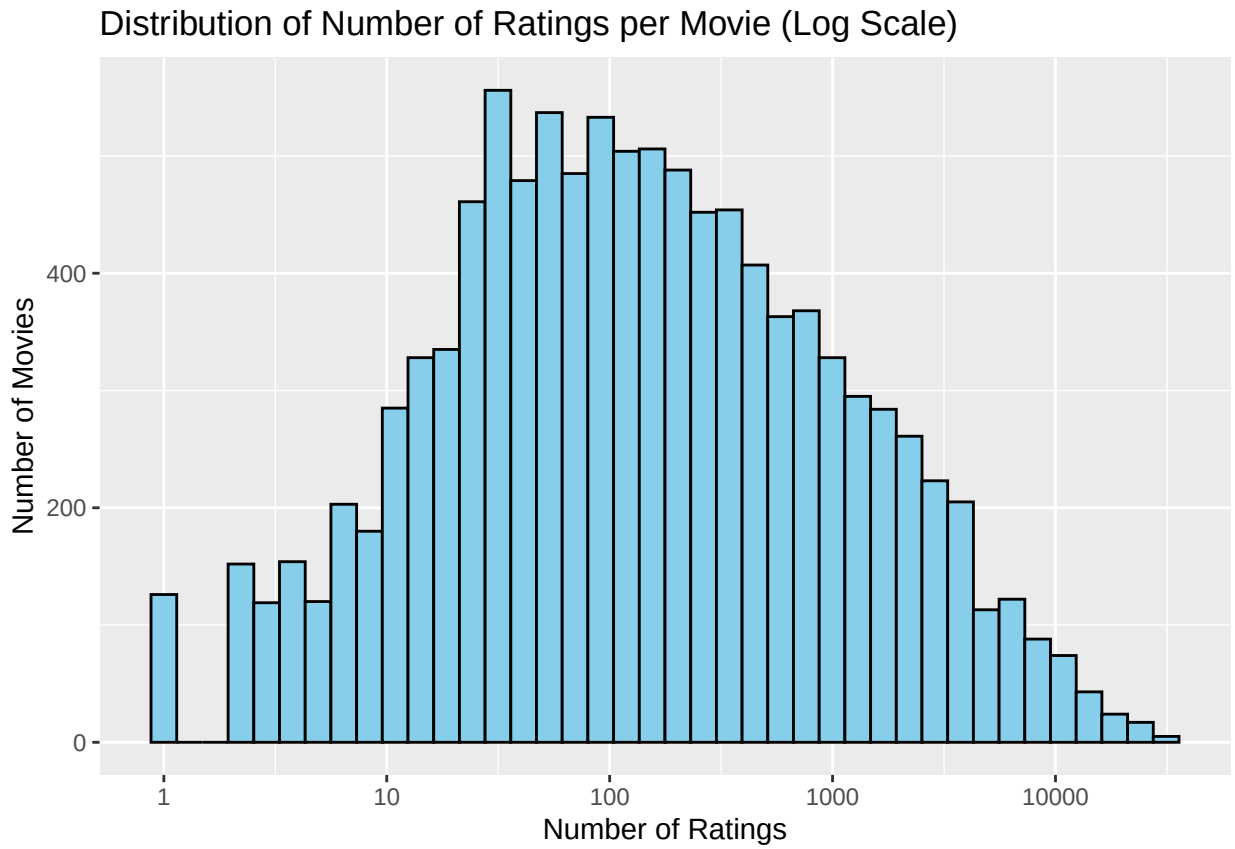
```
## [1] 10677
```

```
n_distinct(edx$userId)
```

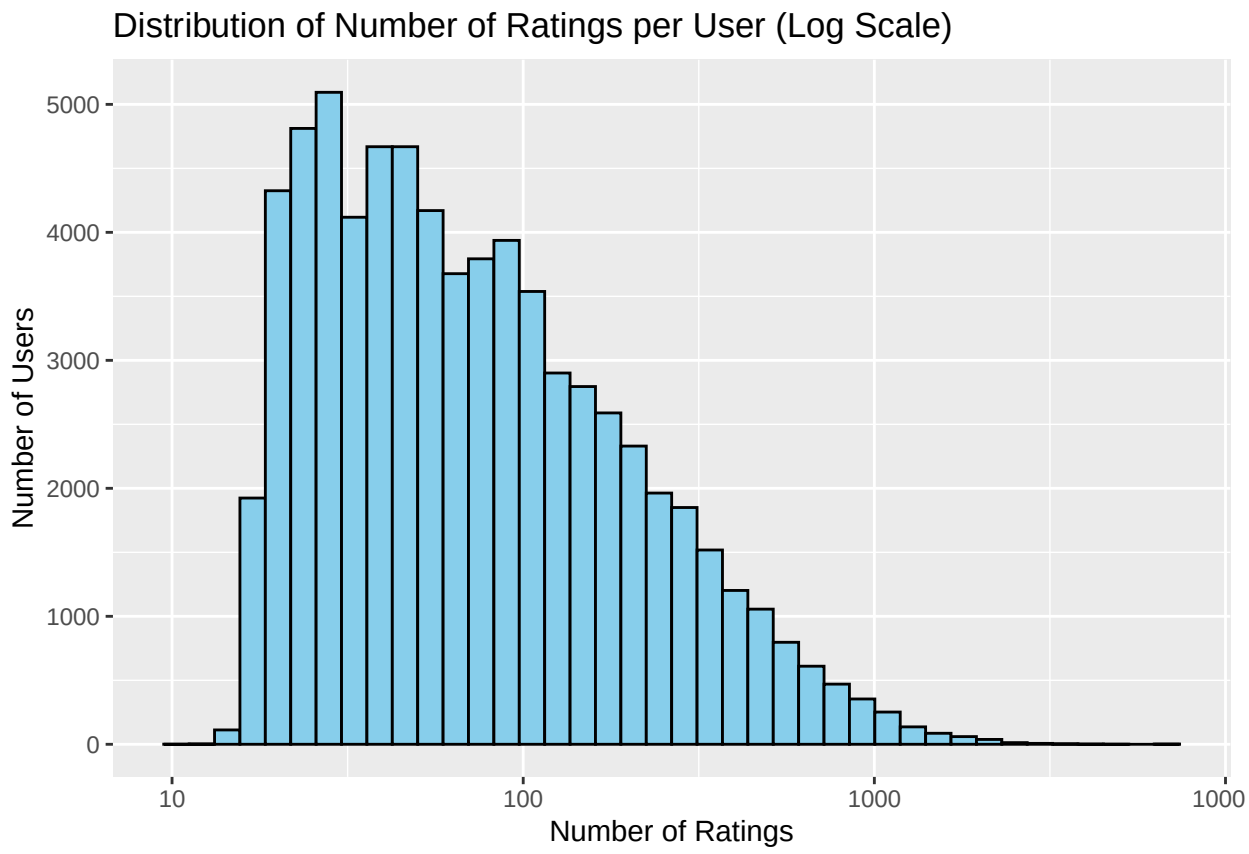
```
## [1] 69878
```

The histograms of the number of ratings per movie and per user are generated on a log scale to further explore rating behavior.

```
# Ratings Per Movie (Log Scale)
edx %>%
  count(movieId) %>%
  ggplot(aes(x = n)) +
  geom_histogram(bins = 40, fill = "skyblue", color = "black") +
  scale_x_log10() +
  labs(title = "Distribution of Number of Ratings per Movie (Log Scale)",
       x = "Number of Ratings",
       y = "Number of Movies")
```




```r
# Ratings Per User (Log Scale)
edx %>%
  count(userId) %>%
  ggplot(aes(x = n)) +
  geom_histogram(bins = 40, fill = "skyblue", color = "black") +
  scale_x_log10() +
  labs(title = "Distribution of Number of Ratings per User (Log Scale)",
       x = "Number of Ratings",
       y = "Number of Users")
```



3.4 Number of Ratings for All Genres

The frequency of each genre is counted and sorted in descending order to analyze genre popularity.

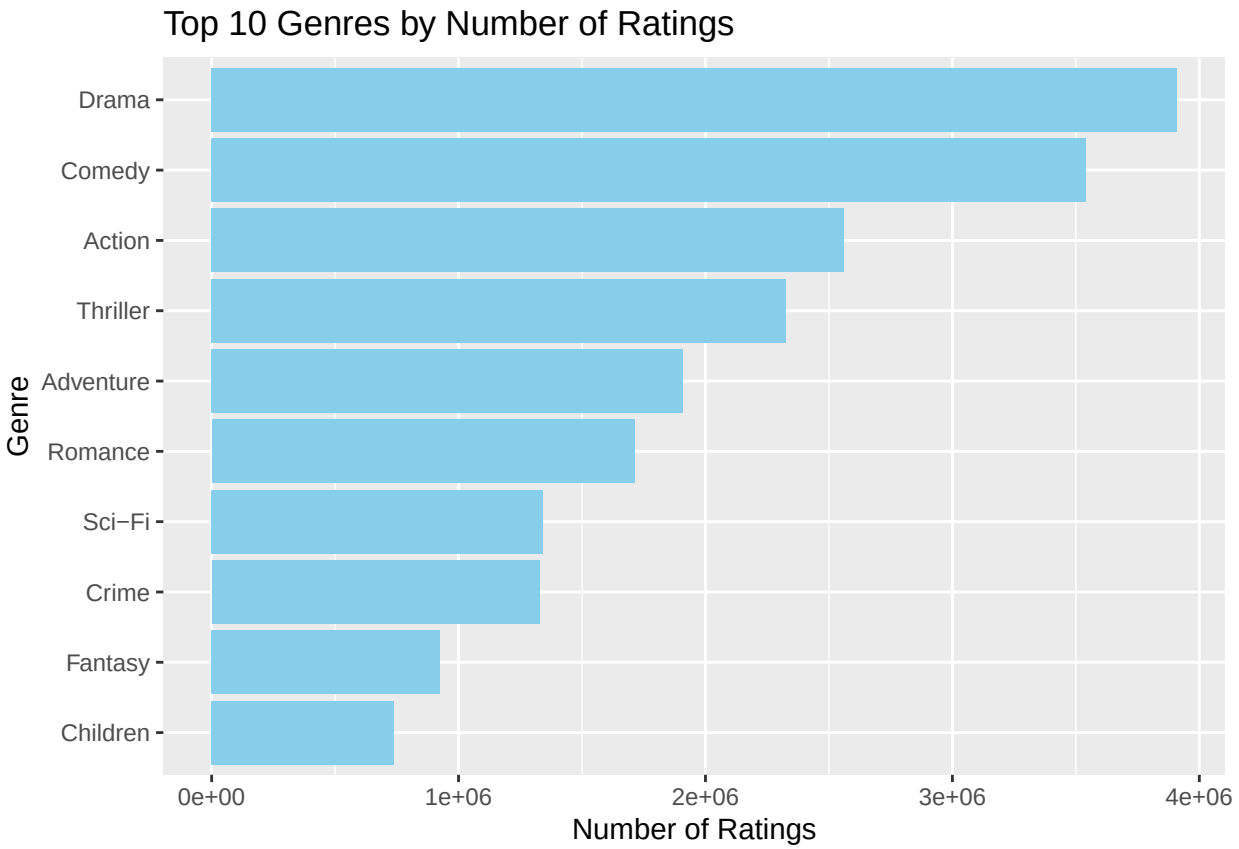
```
edx %>%
  separate_rows(genres, sep = "\\|") %>%
  count(genres) %>%
  arrange(desc(n))
```

```
## # A tibble: 20 x 2
##    genres                 n
##    <chr>              <int>
##  1 Drama            3910127
##  2 Comedy           3540930
##  3 Action           2560545
##  4 Thriller         2325899
##  5 Adventure        1908892
##  6 Romance          1712100
##  7 Sci-Fi           1341183
##  8 Crime            1327715
##  9 Fantasy           925637
## 10 Children          737994
## 11 Horror            691485
## 12 Mystery           568332
```

```
## 13 War                 511147
## 14 Animation           467168
## 15 Musical             433080
## 16 Western             189394
## 17 Film-Noir           118541
## 18 Documentary          93066
## 19 IMAX                  8181
## 20 (no genres listed)       7
```

A bar plot is generated to display the top 10 most frequent genres. This is done by selecting the top 10 genres based on count and reordering them by frequency, allowing us to highlight the most prominent trends in genre preferences.

```
edx %>%
  separate_rows(genres, sep = "\\|") %>%
  count(genres) %>%
  top_n(10, n) %>%
  ggplot(aes(x = reorder(genres, n), y = n)) +
  geom_col(fill = "skyblue") +
  coord_flip() +
  labs(title = "Top 10 Genres by Number of Ratings",
       x = "Genre",
       y = "Number of Ratings")
```



3.5 Half-Star vs Whole-Star Ratings

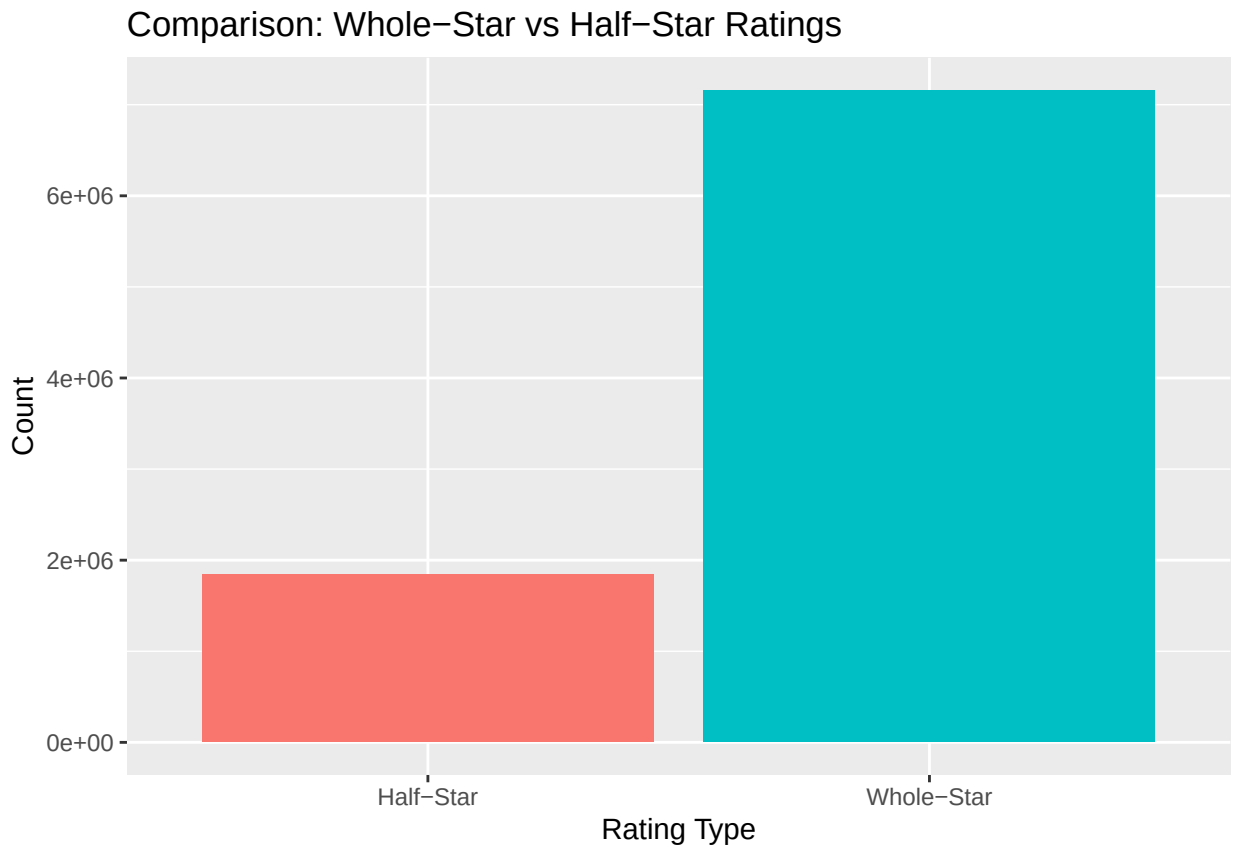
The granularity of the ratings is examined by classifying each rating as either a "Whole-Star" or a "Half-Star," and then calculating the total count for each category to determine how frequently users assign whole-star versus half-star ratings.

```
edx %>%
  mutate(rating_type = ifelse(rating %% 1 == 0, "Whole-Star", "Half-Star")) %>%
  count(rating_type)
```

```
##   rating_type       n
## 1   Half-Star 1843170
## 2  Whole-Star 7156885
```

A bar plot is used to compare the "Whole-Star" and "Half-Star" categories.

```
edx %>%
  mutate(rating_type = ifelse(rating %% 1 == 0, "Whole-Star", "Half-Star")) %>%
  count(rating_type) %>%
  ggplot(aes(x = rating_type, y = n, fill = rating_type)) +
  geom_col() +
  labs(title = "Comparison: Whole-Star vs Half-Star Ratings",
       x = "Rating Type", y = "Count") +
  theme(legend.position = "none")
```



4. Model Development and Evaluation

First, the edx dataset is split into edx_train and edx_test. The edx_train set is used to train models during development, while edx_test is used to evaluate their RMSE performance. This separation allows for safe experimentation without using the final_holdout_test set, which is reserved exclusively for final model evaluation.

The movie rating predictions will be compared to the true ratings in the train models and final model. The RMSE function is defined as

```
RMSE <- function(true_ratings, predicted_ratings) {
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

4.1 Dataset Split for Model Testing

This R code starts by setting a random seed to ensure the reproducibility of results. It then uses the createDataPartition() function from the caret package to randomly select 10% of the edx dataset, stratified by the rating variable, and stores the corresponding indices as the test set. The dataset is subsequently divided into edx_train (90%) and edx_test (10%). To ensure the model can generate valid predictions, edx_test is further filtered to retain only rows where both movieId and userId exist in the training set. This step is critical for building an effective recommendation system, as it ensures the test set includes only users and movies the model has previously encountered.

```
set.seed(1, sample.kind = "Rounding")
```

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

```
test_index <- createDataPartition(edx$rating, times = 1, p = 0.1, list = FALSE)
edx_train <- edx[-test_index, ]
edx_test <- edx[test_index, ]

# Keep only users and movies in test that are also in train
edx_test <- edx_test %>%
  semi_join(edx_train, by = "movieId") %>%
  semi_join(edx_train, by = "userId")
```

4.2 Baseline Model: Predict Global Average

The simplest model predicts ratings using the overall mean rating from the training set.

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

This model yields an RMSE of naive_rmse:

```
naive_rmse <- RMSE(edx_test$rating, mu_hat)
naive_rmse
```

```
## [1] 1.060054
```

The RMSE for this model is 1.0601, which is expected to be high. Nevertheless, it will serve as a benchmark for comparison with the subsequent models to be developed.

4.3 Movie Effect Model

The movie effect model improves upon the baseline by incorporating a movie-specific bias (b_i), calculated as the average deviation of a movie's ratings from the global mean. The prediction is mu_hat + b_i:

```
movie_avgs <- edx_train %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu_hat))

predicted_ratings <- edx_test %>%
  left_join(movie_avgs, by = "movieId") %>%
  mutate(pred = mu_hat + b_i) %>%
  pull(pred)
```

The RMSE for this model is movie_effect_rmse:

```
movie_effect_rmse <- RMSE(edx_test$rating, predicted_ratings)
movie_effect_rmse
```

```
## [1] 0.9429615
```

The RMSE for this model is 0.943, reflecting the impact of movie-specific variations. We can see the value is lower than baseline model, however, still far from our initial goal.

4.4 Movie + User Effect Model

This model extends the movie effect by adding a user-specific bias (b_u), capturing individual user rating tendencies. The prediction becomes mu_hat + b_i + b_u:

```
user_avgs <- edx_train %>%
  left_join(movie_avgs, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu_hat - b_i))

predicted_ratings <- edx_test %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  mutate(pred = mu_hat + b_i + b_u) %>%
  pull(pred)
```

The RMSE for this model is movie_user_effect_rmse:

```
movie_user_effect_rmse <- RMSE(edx_test$rating, predicted_ratings)
movie_user_effect_rmse
```

```
## [1] 0.8646844
```

The RMSE for this model is 0.8647, showing a noticeable improvement over the previous model. However, we will now explore the concept of regularization to further enhance its performance.

4.5 Regularized Movie + User Effect Model

We now introduce the concept of regularization to address overfitting, particularly for movies or users with very few ratings. This approach involves adding a penalty parameter, lambda (lambda), to reduce the influence of extreme bias estimates. By penalizing large deviations in the movie and user bias terms, the model becomes more robust and generalizes better. To determine the optimal lambda value, a range of values from 0 to 10 (in increments of 0.25) is tested, with the goal of minimizing the RMSE and improving the overall performance of the recommendation system.

The tuning process begins by defining a sequence of lambda values ranging from 0 to 10 in increments of 0.25. For each lambda, the model first computes the global mean rating from the training set. It then calculates movie-specific biases (b_i) by grouping ratings by movieId, subtracting the global mean, summing the differences, and applying regularization by dividing by the number of ratings plus the lambda value.

Next, user-specific biases (b_u) are determined by joining the computed movie biases, grouping by userId, subtracting both the global mean and movie bias, summing the residuals, and again applying regularization using the same formula.

With these bias terms, predictions are generated for the test set by joining both movie and user biases and adding them to the global mean. The Root Mean Squared Error (RMSE) is then calculated by comparing these predicted ratings to the actual ratings in the test set.

The optimal lambda value (best_lambda) is selected as the one that yields the lowest RMSE across all tested values, ensuring the best balance between model complexity and prediction accuracy.

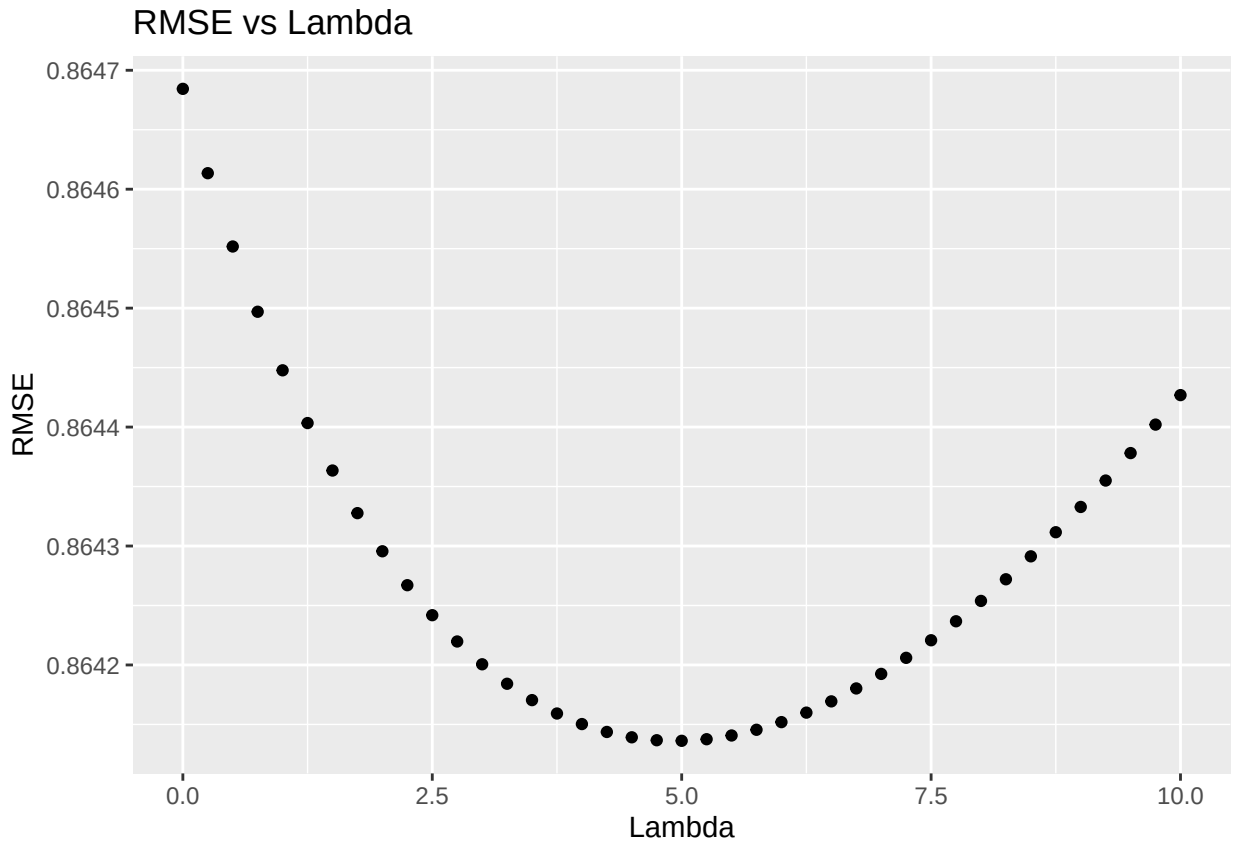
```
lambdas <- seq(0, 10, 0.25)

rmses <- sapply(lambdas, function(lambda) {
  mu <- mean(edx_train$rating)

  b_i <- edx_train %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu) / (n() + lambda))

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

  predicted_ratings <- edx_test %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

  return(RMSE(edx_test$rating, predicted_ratings))
})

# Best lambda
best_lambda <- lambdas[which.min(rmses)]
best_lambda
```

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

A visualization of RMSE versus lambda is generated using a scatter plot. The qplot function is used to plot lambda values on the x-axis and their corresponding RMSE values on the y-axis. The plot includes a clear

title and labeled axes to illustrate the relationship between lambda and RMSE, making it easier to identify the optimal lambda value that minimizes prediction error.

```
ggplot(data = data.frame(lambda = lambdas, rmse = rmses), aes(x = lambda, y = rmse)) +
  geom_point() +
  labs(title = "RMSE vs Lambda", x = "Lambda", y = "RMSE")
```



The final model is built using the optimal value of best_lambda. It begins by calculating the global mean rating from the training set. Movie biases (b_i) are then computed by grouping the data by movieId, subtracting the global mean, summing the differences, and applying regularization by dividing by the number of ratings plus best_lambda. Next, user biases (b_u) are calculated by joining the movie biases, grouping by userId, subtracting both the global mean and movie bias, summing the differences, and dividing by the number of ratings plus best_lambda. Predictions for the test set are made by joining the movie and user biases with the test data, then adding these to the global mean. The model's performance is evaluated using RMSE (regularized_rmse) by comparing the predicted ratings with the actual ratings in the test set.

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

b_i <- edx_train %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu) / (n() + best_lambda))

b_u <- edx_train %>%
  left_join(b_i, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - mu - b_i) / (n() + best_lambda))
```

```r
predicted_ratings <- edx_test %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

regularized_rmse <- RMSE(edx_test$rating, predicted_ratings)
regularized_rmse
```

```
## [1] 0.8641362
```

The model demonstrates a slight improvement compared to the previous approach, achieving an RMSE of 0.8641.

The performance of the regularized model is compared with the previous models—Naive Mean, Movie Effect, and Movie + User Effect—using a summary table. This table is created as a tibble, with columns showing the names of the methods and their corresponding RMSE values, taken from the variables naive_rmse, movie_effect_rmse, movie_user_effect_rmse, and regularized_rmse. The table is then printed to clearly display how each model performs.

```r
rmse_results <- tibble(
  method = c("Naive Mean", "Movie Effect", "Movie + User Effect", "Regularized Movie + User"),
  RMSE = c(naive_rmse, movie_effect_rmse, movie_user_effect_rmse, regularized_rmse)
)

print(rmse_results)
```

```
## # A tibble: 4 x 2
##   method                     RMSE
##   <chr>                     <dbl>
## 1 Naive Mean                1.06
## 2 Movie Effect              0.943
## 3 Movie + User Effect       0.865
## 4 Regularized Movie + User 0.864
```

5. Final Evaluation on Holdout Set

In this section, we using the optimal best_lambda identified during the tuning phase, this section applies the regularized Movie + User Effect model to generate predictions for the final_holdout_test dataset.

```r
lambda <- best_lambda

# Global average
mu <- mean(edx$rating)

# Movie effect
b_i <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu) / (n() + lambda))

# User effect
```

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

# Predict final ratings
final_predictions <- final_holdout_test %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

# Replace any missing predictions with mu
final_predictions[is.na(final_predictions)] <- mu

# Final RMSE
final_rmse <- RMSE(final_holdout_test$rating, final_predictions)
final_rmse
```

```
## [1] 0.8648177
```

The regularized Movie + User Effect model was applied to the final_holdout_test set using the optimal best_lambda value. By combining the global mean with movie and user biases, the model predicted ratings for unseen data.

It achieved an RMSE of 0.8648, showing good accuracy and meeting the goal of the project.

6 Conclusion

This project built a movie recommendation system using the MovieLens 10M dataset. We started with simple models based on average ratings, and gradually improved performance by adding movie and user effects. To avoid overfitting, we applied regularization and selected the best parameter using cross-validation on the training data.

The final model, trained on the full edx set, achieved an RMSE of 0.8648 on the final_holdout_test set. This result meets the objective for the project. Overall, the model effectively captures key patterns in user and movie behavior. Future work could explore advanced methods like matrix factorization to further improve predictions.