Harvard Data Science Professional Certificate: Capstone Project-MovieLens

Bhathiya Maneendra Pilanawithana

2023-04-11

1. Introduction

The MovieLens dataset is a database with multiple interconnected tables. It contains a vast amount of data, often spanning several years, with thousands of movies, millions of ratings, and thousands of users, providing a significant sample size for analysis. The Movies table in the MovieLens dataset provides comprehensive metadata about movies, including movie titles, genres, and release dates. This information can be used for data exploration, visualization, and analysis of movie trends, such as popular genres, movie release patterns, and changes in movie genres over time. The movie genres can also be used for content-based filtering, where movies are recommended based on similarity in genre. The Users table in the dataset includes demographic information about users, such as age, gender, and occupation. The Ratings table in the dataset contains user ratings for movies, represented as numeric values or ratings on a scale with a time-stamp on review submission.

The project description of the Capstone project contains an initial downloading and connection of tables which provides two datasets one for training and one for testing. The dataset as the following information.

1. Movie ID - Column Name: movieId
2. Movie Title - Column Name: title

   Apart from the title of the movie, "title" column contains the release year.
3. Genres - Column Name: genre

   Genres of the a movie are pipe-separated and are selected from the following genres.

   Action, Adventure, Animation, Children, Comedy, Crime, Documentary, Drama, Fantasy, Film-Noir, Horror, Musical, Mystery, Romance, Sci-Fi, Thriller, War, Western
4. User ID - Column Name: userID
5. Rating - Column Name: rating

   Ratings of the movie submitted by the each user. Rating is a number ranging from 0 to 5. Only the integers or .5s are allowed as ratings. Highest rated is a 5 and a least rated is a 0.
6. Timestamp - Column Name: timestamp

   Unix timestamp of user review submission.

Each row of the dataframe is a movie review submission by a user. each row consists of a userId, movieId, title, timestamp, rating, genre. No missing values were detected with in both training and test dataframes. To check for missing values the following code can be used.

```r
na_sum_edx <- sum(is.na(edx %>%
                          select(userId, movieId, rating, timestamp)))
na_sum_holdout <- sum(is.na(final_holdout_test %>%
                              select(userId, movieId, rating, timestamp)))
```

1.1 Objective and Methodology

The objective of the project is to develop a rating prediction model using the four available variables, which are movieId, userId, genres, years_to_rate. The given dataset is too large to deploy any Machine Learning technique in a personal computer. Therefore, a prediction model is developed based on average rating. The development of the prediction model is described in detail in section 2.3.

2. Analysis

First, a data pre-processing step is carried out to extract information from dataset.

2.1 Data Pre-processing

The review submission time-stamp is given as a UNIX epoch. For intuition, further analysis and to stratify the time-stamp is converted to year. After that the time-stamp column will be dropped as it does not have any further information. The review submission year information should also be available in the test set for prediction. Decoding the UNIX time-stamp to append the review submission year to both training set and the test set can be achieved by the following code

```r
edx <- edx %>%
  mutate(rating_year = as.POSIXlt(timestamp, origin = "1970-01-01")$year+1900) %>%
  select(-timestamp)

final_holdout_test <- final_holdout_test %>%
  mutate(rating_year = as.POSIXlt(timestamp, origin = "1970-01-01")$year+1900) %>%
  select(-timestamp)
```

The release year of a movie may also deem useful for a rating prediction. The release year of a movies is stated in the title column and it is last 4 characters of the movie title. Using the release year of the movie and the rating year, a new variable named "years_to_rate" can be calculated, which describes the difference of years from release to user review submission. The following code can be used to extract the release year information, calculate the years_to_rate and append both training and testing set.

```r
edx <- edx %>%
  mutate(release_year = as.integer(substr(title, str_length(title) - 4,
                                          str_length(title) - 1))) %>%
  mutate(years_to_rate = rating_year - release_year)

final_holdout_test <- final_holdout_test %>%
  mutate(release_year = as.integer(substr(title, str_length(title) - 4,
                                          str_length(title) - 1)))  %>%
  mutate(years_to_rate = rating_year - release_year)
```

The pre-processing step is required to be done on both testing and training sets to make sure all the required columns are available on the test set as well. After this, the testing test is not used for any data analysis action until the calculation of testing RMSE.

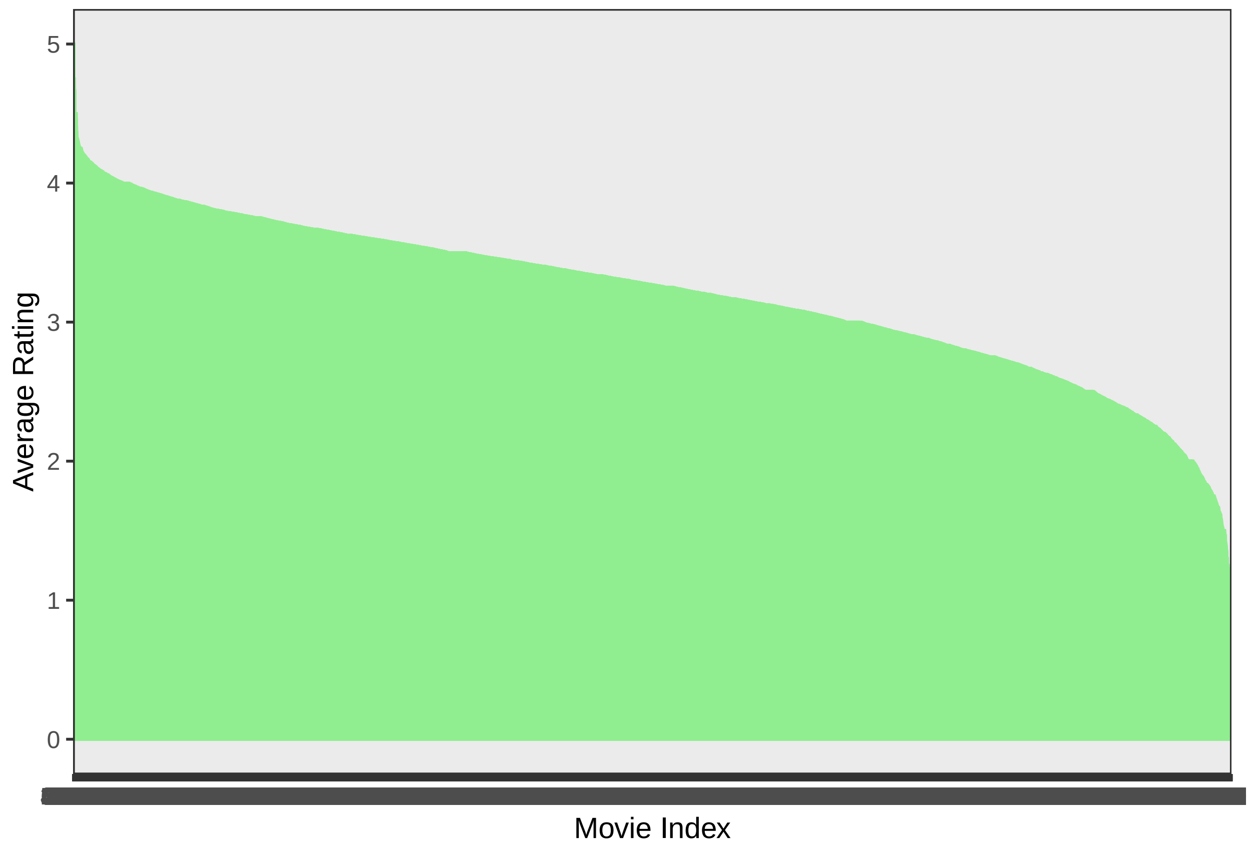
2.2 Data Exploration and Visualization

Since the training set is too large to plot individual variables, summary statistics generated using group_by() and summarize() functions, are used. Average rating and standard deviation of user ratings per each movie can be calculated using the following code.

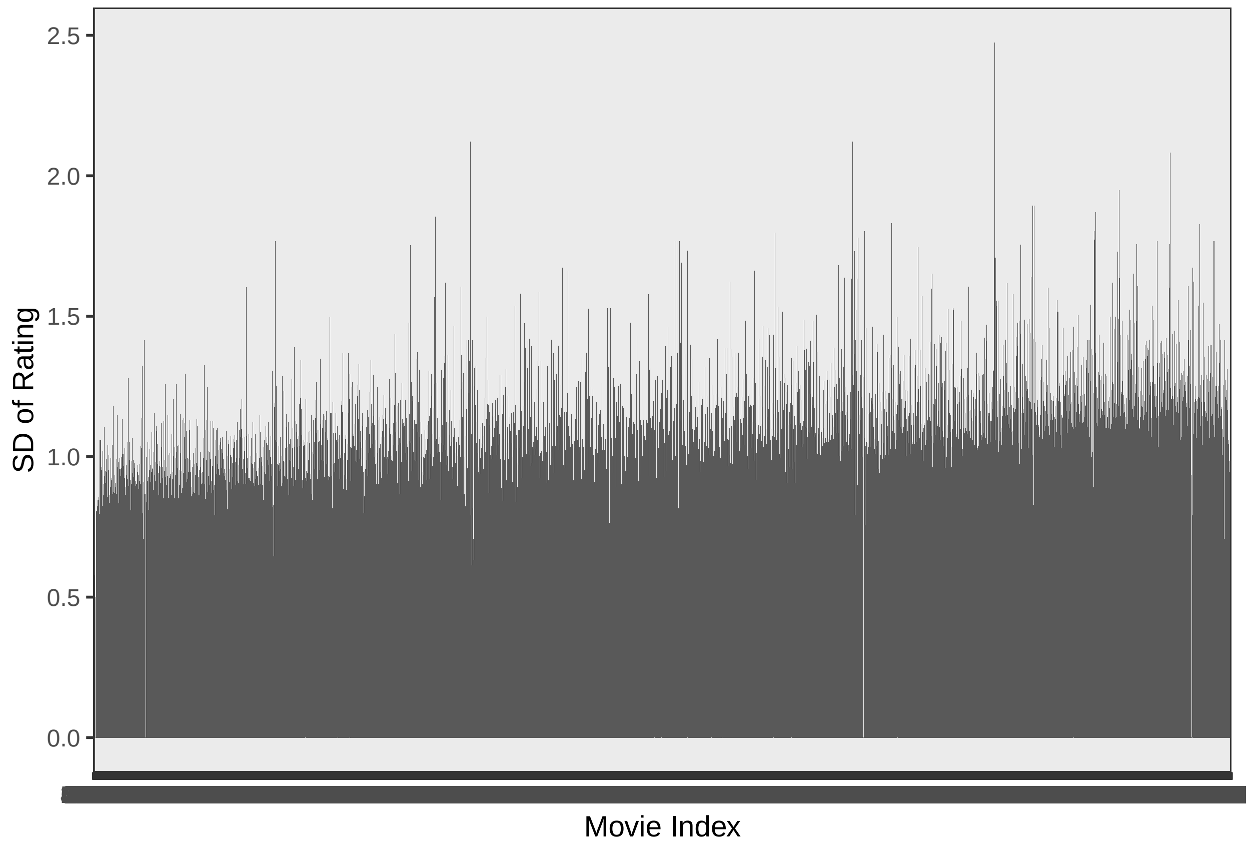
```
summary_by_movieId <- edx %>%
  group_by(movieId) %>%
  summarise(mean_rating = mean(rating), sd_rating = sd(rating))
```

The average rating and standard deviation of rating per movie can be plotted using the following code.

```
summary_by_movieId %>%
  ggplot(aes(x = reorder(as.character(movieId), -mean_rating), y = mean_rating)) +
  geom_bar(stat="identity", color = "palegreen2") +
  theme(axis.text.x = element_blank()) +
  labs( x = "Movie Index", y = "Average Rating") +
  theme_bw()
```

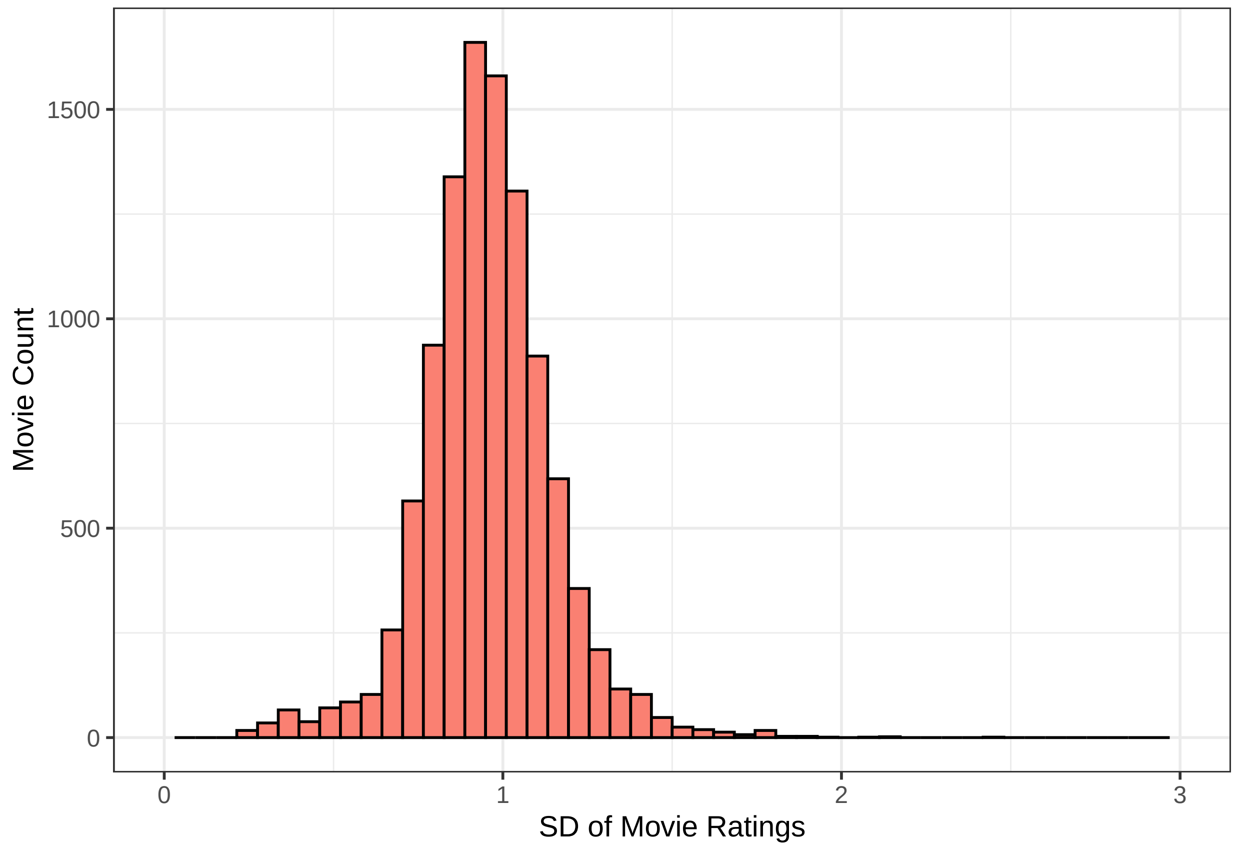


```
summary_by_movieId %>%
  ggplot(aes(x = reorder(as.character(movieId), -mean_rating), y = sd_rating)) +
  geom_bar(stat="identity") +
  theme(axis.text.x = element_blank()) +
  labs( x = "Movie Index", y = "SD of Rating") +
  theme_bw()
```



The average rating plot and standard deviation of the rating is ordered by the average rating. It can be observed that movies has considerable standard deviations. This can be further verified using the histogram of standard deviations of movie rating, which can be plotted using the following code.

```r
summary_by_movieId %>%
  ggplot(aes(x = sd_rating)) +
  geom_histogram(bins = 50, color = "black", fill = "salmon") +
  labs( x = "SD of Movie Ratings", y = "Movie Count") +
  xlim(0, 3) +
  theme_bw()
```



It can be observed that there are considerable variations for rating per movie. Therefore predicting the rating using only the average rating movies will incur considerable performance degradation.

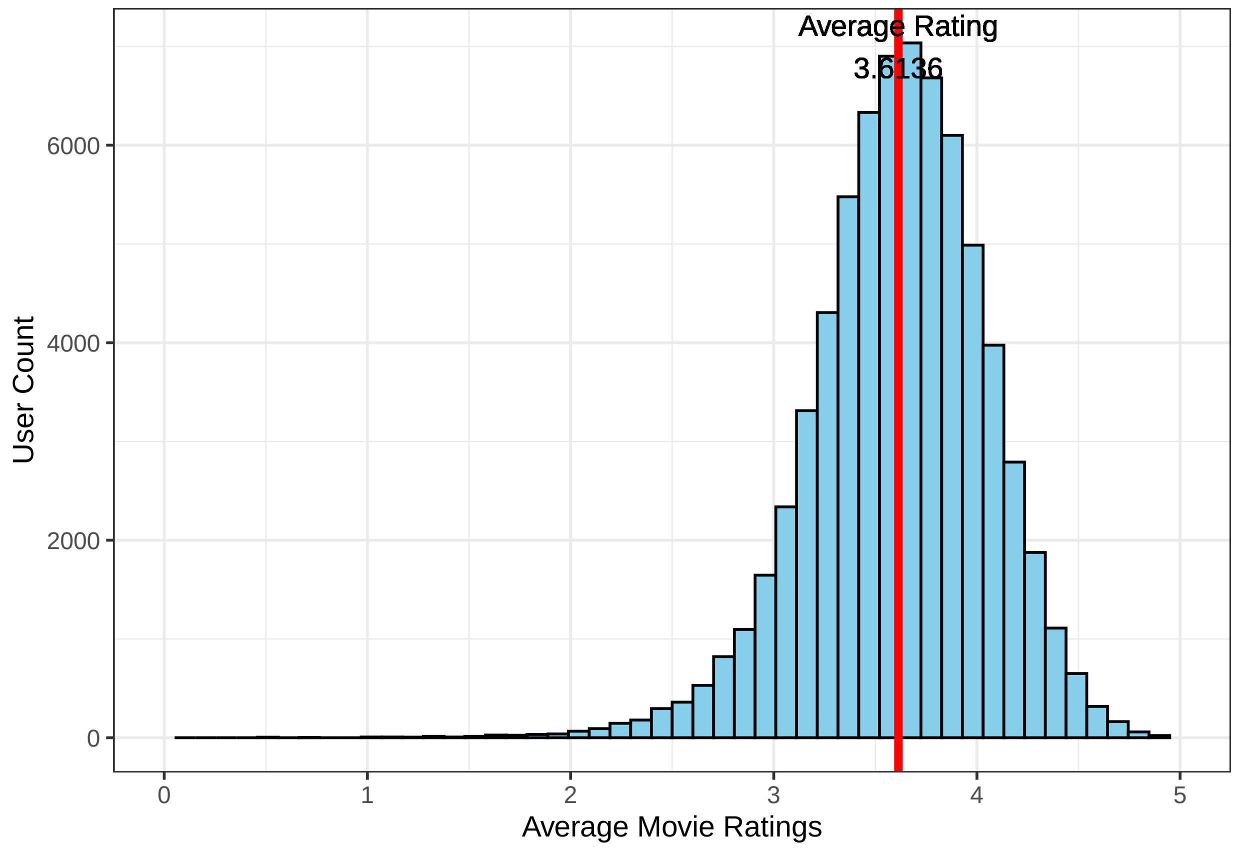
Next, the summary statistics per user can be calculated using the following code.

```
summary_by_userId <- edx %>%
  group_by(userId) %>%
  summarise(mean_rating = mean(rating), sd_rating = sd(rating))
```

The histogram of average rating per user can be plotted using the following code.

```
mean_mean_rating_userId <- mean(summary_by_userId$mean_rating)
summary_by_userId %>%
  ggplot(aes(x = mean_rating)) +
  geom_histogram(bins = 50, color = "black", fill = "skyblue") +
  labs( x = "Average Movie Ratings", y = "User Count") +
  geom_vline(xintercept=mean_mean_rating_userId, size=1.5, color="red") +
  geom_text(aes(x=mean_mean_rating_userId,
              label=paste0("Average Rating\n",
                           round(mean_mean_rating_userId,4)), y=7000)) +
  xlim(0, 5) +
  theme_bw()
```

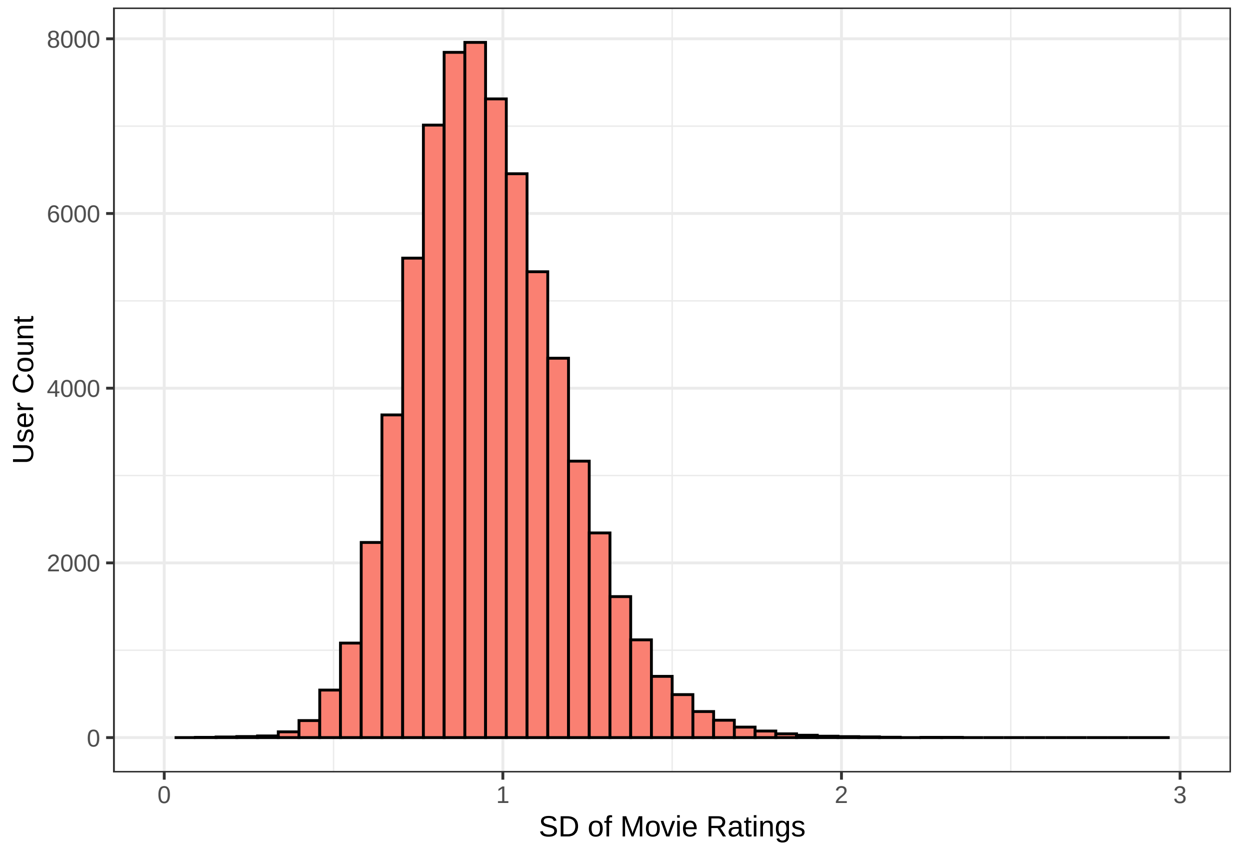
```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
## Warning: Removed 2 rows containing missing values (`geom_bar()`).
```



And the standard deviation of ratings per user can be plotted using the following code.

```
summary_by_userId %>%
  ggplot(aes(x = sd_rating)) +
  geom_histogram(bins = 50, color = "black", fill = "salmon") +
  labs( x = "SD of Movie Ratings", y = "User Count") +
  xlim(0, 3) +
  theme_bw()
```

```
## Warning: Removed 2 rows containing missing values (`geom_bar()`).
```

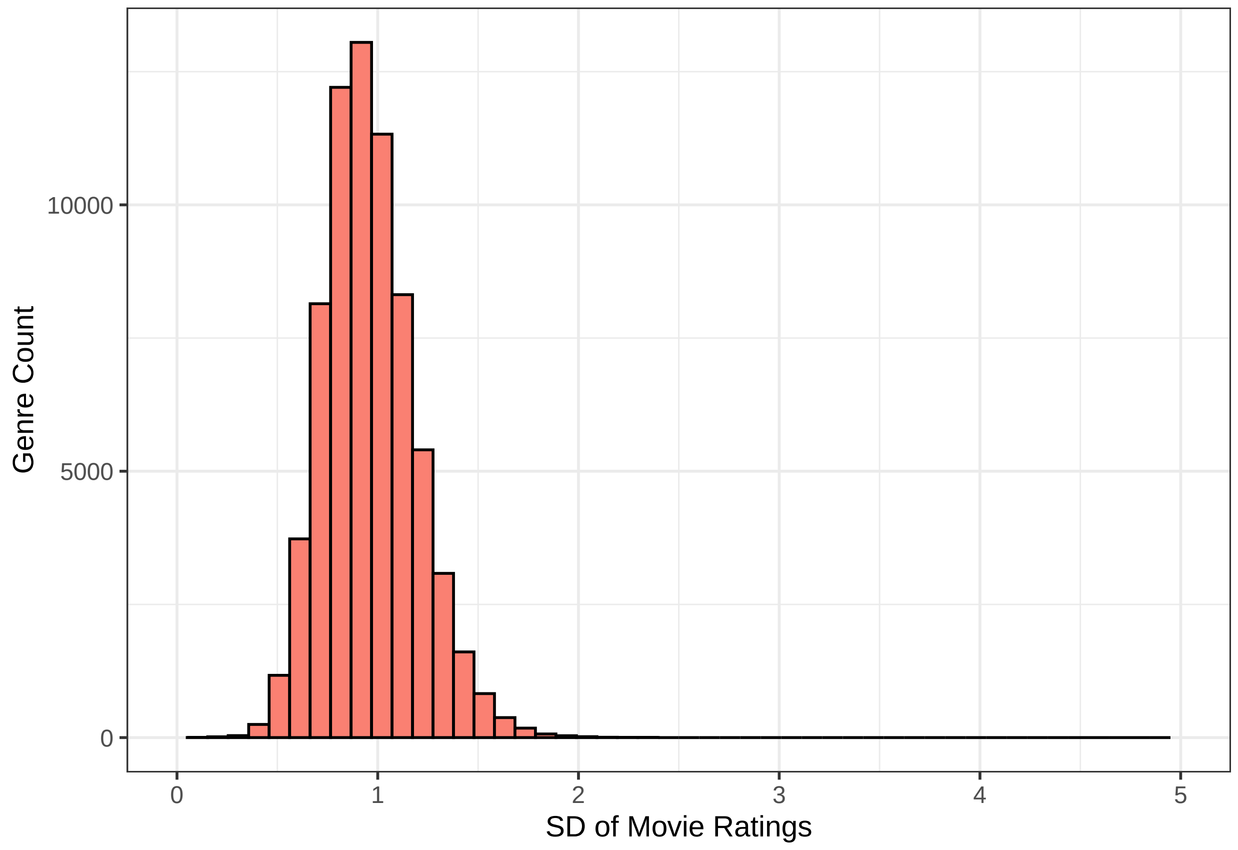


It can be observed that rating per user has considerable variations for rating per user. The other two variables "years_to_rate" and "genres" can also be analysed similarly to find out whether a single variable can be used for rating prediction.

Calculating summary statistics and plotting of other two variables can be achieved using the following codes.

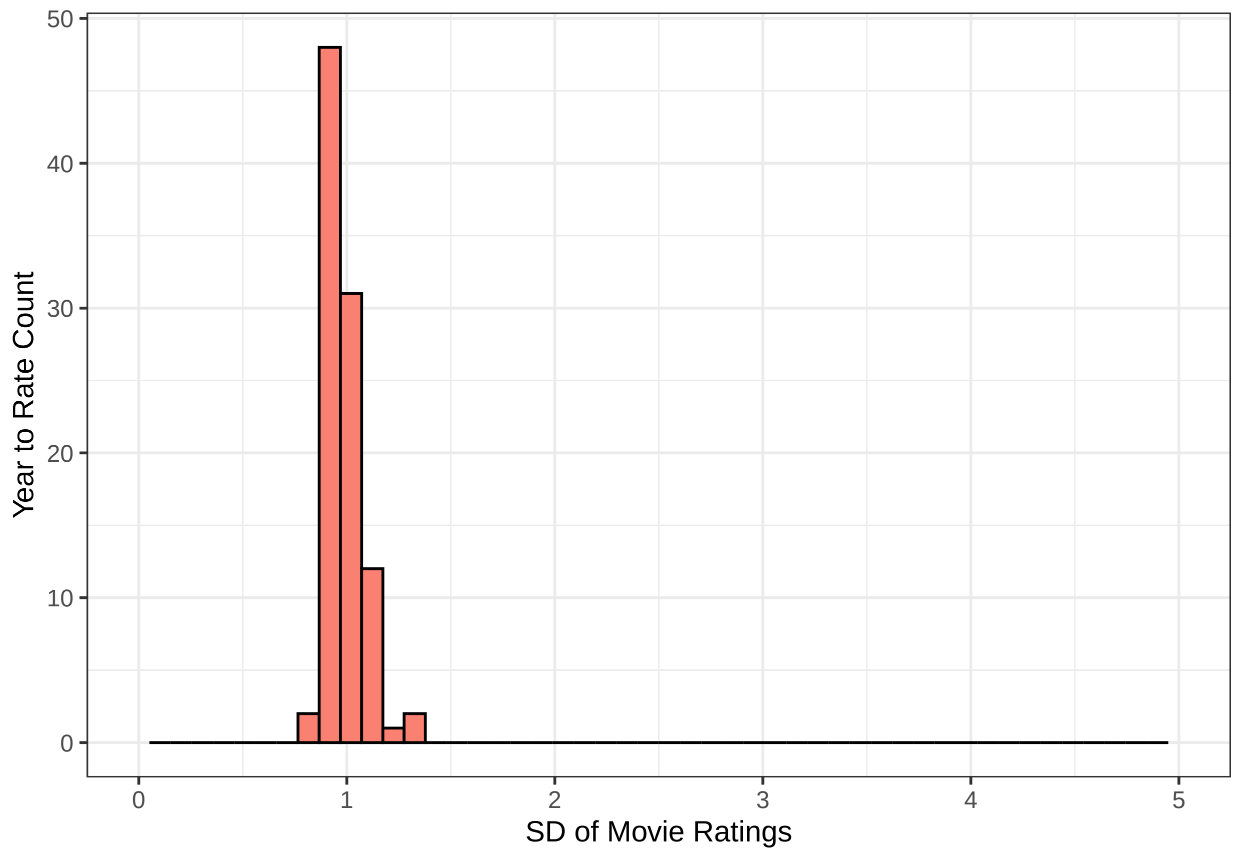
```r
summary_by_genres <- edx %>%
  group_by(genres) %>%
  summarise(mean_rating = mean(rating), sd_rating = sd(rating))

summary_by_userId %>%
  ggplot(aes(x = sd_rating)) +
  geom_histogram(bins = 50, color = "black", fill = "salmon") +
  labs( x = "SD of Movie Ratings", y = "Genre Count") +
  xlim(0, 5) +
  theme_bw()
```

```
## Warning: Removed 2 rows containing missing values (`geom_bar()`).
```



```
summary_by_YearsToRate <- edx %>%
  group_by(years_to_rate) %>%
  summarise(mean_rating = mean(rating), sd_rating = sd(rating), count_rating = n())

summary_by_YearsToRate %>%
  ggplot(aes(x = sd_rating)) +
  geom_histogram(bins = 50, color = "black", fill = "salmon") +
  labs( x = "SD of Movie Ratings", y = "Year to Rate Count") +
  xlim(0, 5) +
  theme_bw()
```

```
## Warning: Removed 2 rows containing missing values (`geom_bar()`).
```



From the above histograms of the standard deviations, it can be observed that standard deviation of rating per each four variables are substantially large, and will not be able to achieve the stated RMSE goals if used alone.

2.3 Development of the Prediction Model

There are four variables in total for the prediction of rating, which are movieId, userId, genres, years_to_rate. However, the given dataset is too large to deploy any Machine Learning techniques for prediction.

Therefore, the steps taken to accomplish the stated RMSE goals are as follows.

1. Use one variable to calculate average rating per variable and use the calculated average rating as the first prediction for the rating.
2. Calculate the error between the actual rating and first rating prediction.
3. Choose a second variable to compensate the error by the first prediction.
4. Calculate the average prediction error per second variable and use average error to compensate the error in prediction. The second prediction is the sum of the average error and first prediction.
5. Calculate the error between the actual rating and the second prediction.
6. Choose a third variable to compensate the error by the third prediction.
7. Calculate the average prediction error per third variable and use average error to compensate the error in prediction. The third prediction is the sum of the average error and the second prediction.
8. Calculate the error between the actual rating and the third prediction.

9. Calculate the average prediction error per forth variable and use average error to compensate the error in prediction. The final prediction is the sum of the average error and the third prediction.

The first variable selected for prediction is the movieId. This is because the number of movieId is considered to the sufficient and desirable.

2.3.1 Step 1 and 2

The following code can be used to calculate the average rating per movieId, append the edx dataset for each observation and caculate the error between the actual rating and the first prediction.

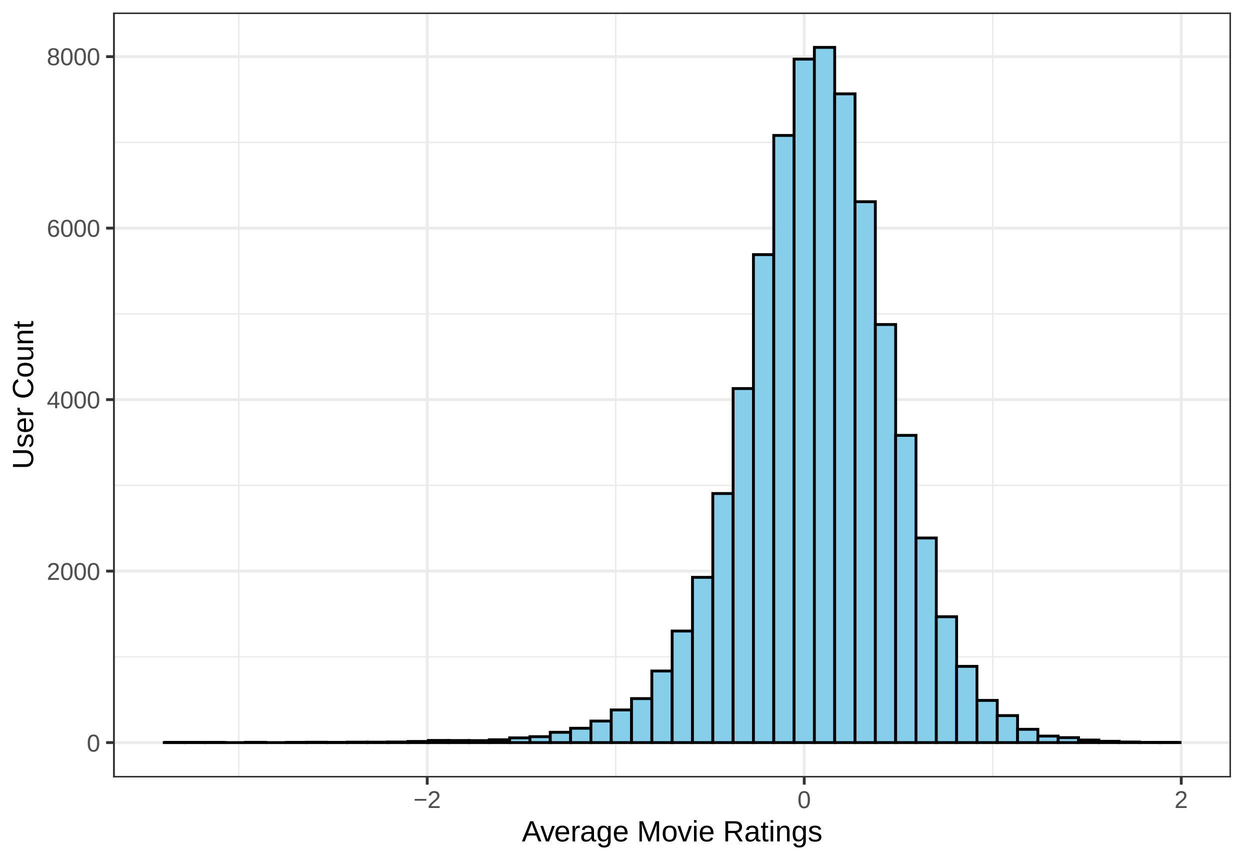
```
mean_rating_per_movieId_vec <- edx %>%
  group_by(movieId) %>%
  summarise(mean_rate_movieId = mean(rating))

edx <- edx %>%
  left_join(mean_rating_per_movieId_vec, by = "movieId")

edx <- edx %>%
  mutate(rate_error_movieId = rating - mean_rate_movieId)
```

The calculated vector with name "mean_rating_per_movieId_vec" is kept to append the final test set. This will be prediction based on movieId.

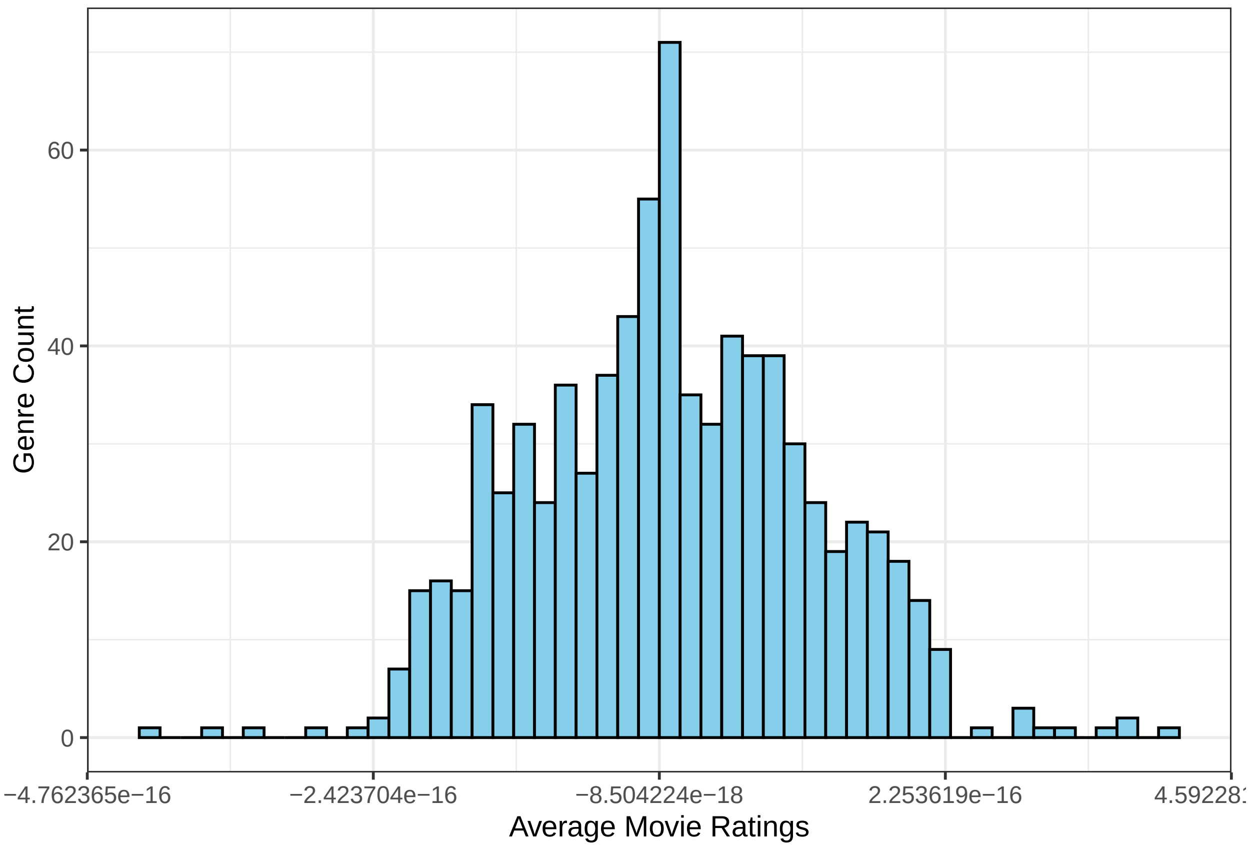
2.3.2 Step 3

The following code can be used to calculate the error in prediction per the remaining three variables and then plot histograms of the average error.

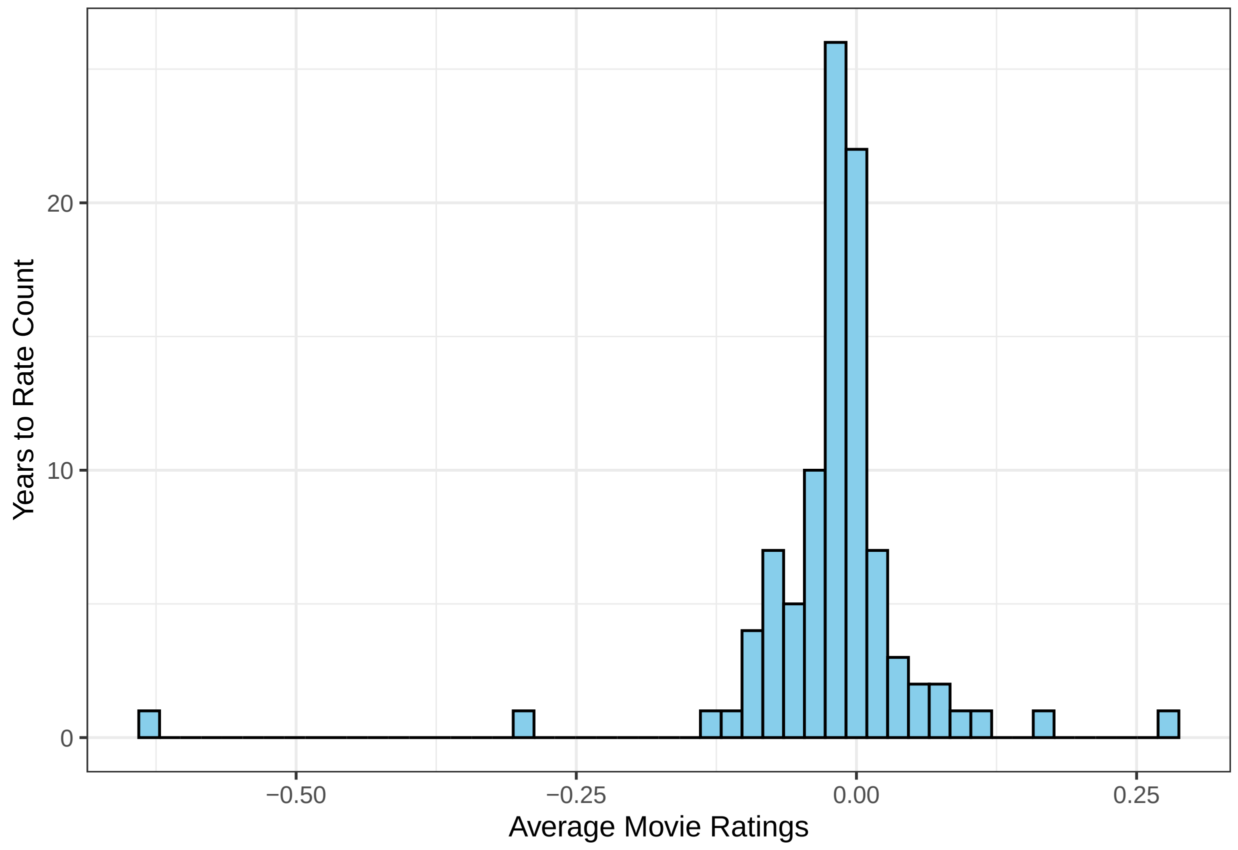
```
mean_error_per_userId_vec <- edx %>%
  group_by(userId) %>%
  summarise(mean_RateError_userId = mean(rate_error_movieId))

mean_error_per_genre_vec <- edx %>%
  group_by(genres) %>%
  summarise(mean_RateError_genres = mean(rate_error_movieId))

mean_error_per_YearsToRate_vec <- edx %>%
  group_by(years_to_rate) %>%
  summarise(mean_RateError_YearsToRate = mean(rate_error_movieId))

mean_error_per_userId_vec %>%
  ggplot(aes(x = mean_RateError_userId)) +
  geom_histogram(bins = 50, color = "black", fill = "skyblue") +
  labs( x = "Average Movie Ratings", y = "User Count") +
  theme_bw()
```



```
mean_error_per_genre_vec %>%
  ggplot(aes(x = mean_RateError_genres)) +
  geom_histogram(bins = 50, color = "black", fill = "skyblue") +
  labs( x = "Average Movie Ratings", y = "Genre Count") +
  theme_bw()
```



```
mean_error_per_YearsToRate_vec %>%
  ggplot(aes(x = mean_RateError_YearsToRate)) +
  geom_histogram(bins = 50, color = "black", fill = "skyblue") +
  labs( x = "Average Movie Ratings", y = "Years to Rate Count") +
  theme_bw()
```



It can be observed from the above histograms that the average error is highest per userId. Therefore, compensating prediction error by first prediction using userId will yield the best performance.

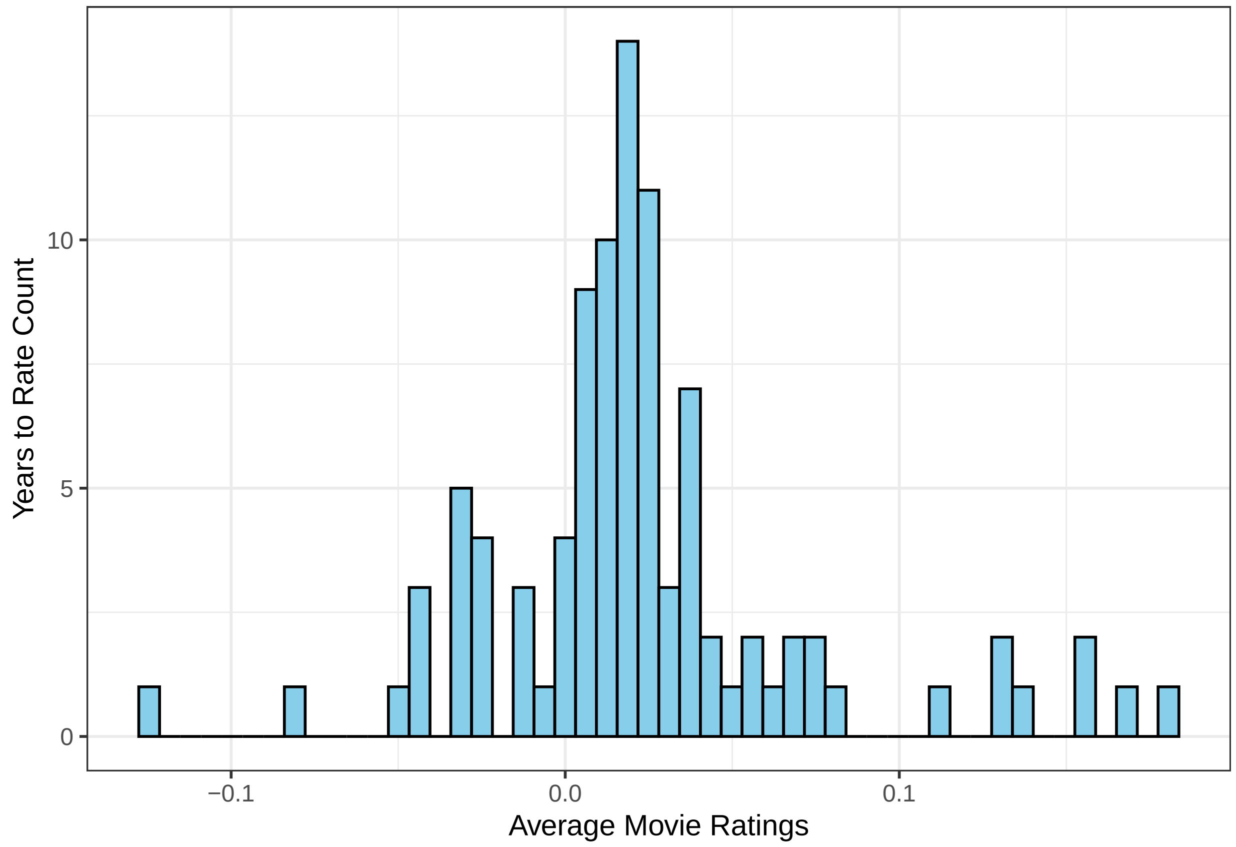
2.3.3 Step 4 and 5 The average prediction error per userId has to be calculated to compensate the error in first prediction. After that edx dataset should be appended for each observation, the average error. The second rating prediction is the sum of first rating prediction and the appended average error. The average prediction error between the actual rating and the second rating prediction should then be calculated for each observation. The following code can be used to accomplish the all of this.

```
mean_error_per_userId_vec <- edx %>%
  group_by(userId) %>%
  summarise(mean_RateError_userId = mean(rate_error_movieId))

edx <- edx %>%
  left_join(mean_error_per_userId_vec, by = "userId")

edx <- edx %>%
  mutate(rate_predict_movieId_userId =
           mean_rate_movieId + mean_RateError_userId)%>%
  mutate(rate_error_movieId_userID =
           rating - rate_predict_movieId_userId)
```

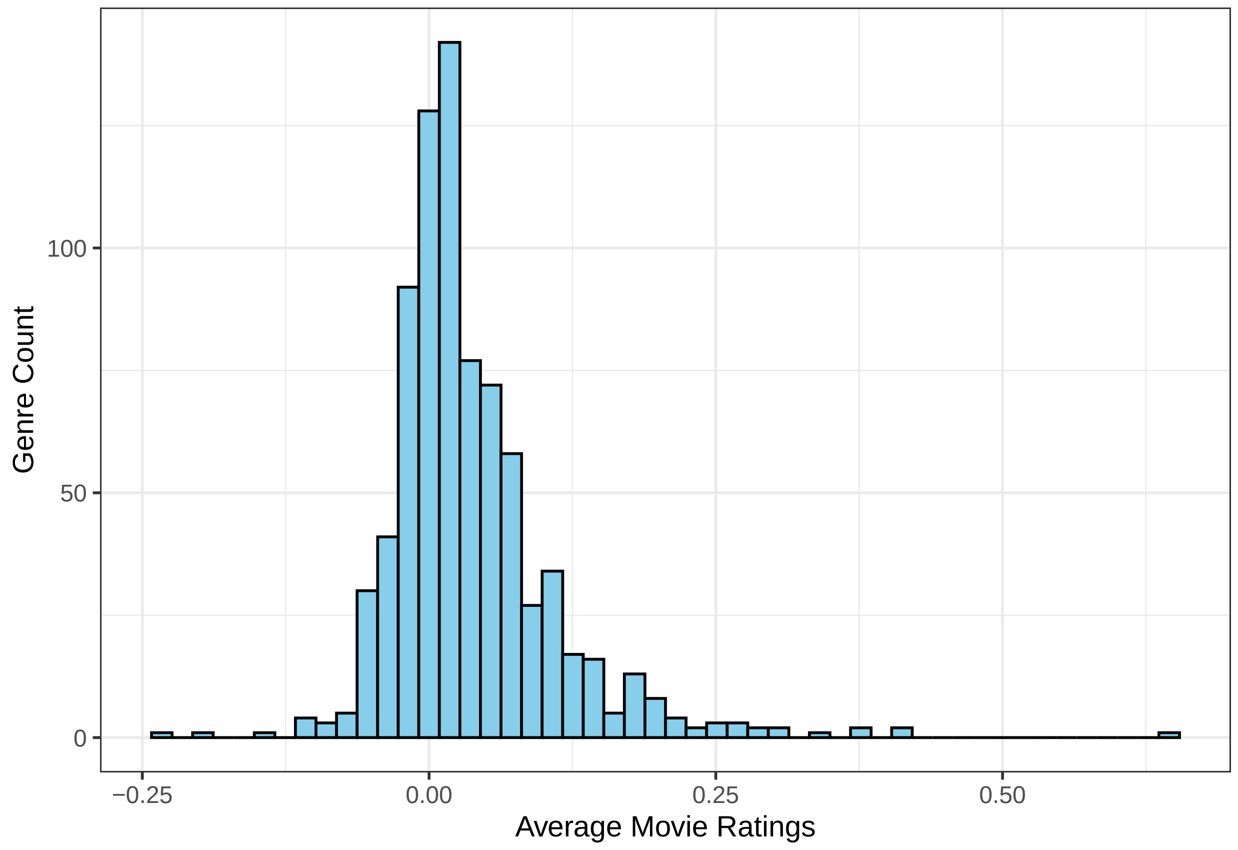
The calculated vector with name "mean__error__per__userId__vec is kept to append the final test set. This will be used for prediction based on movieId and userId.

2.3.4 Step 6 The following code can be used to calculate the error in prediction per the remaining two variables and then plot histograms of the average error.

```
mean_error_per_userId_YearsToRate_vec <- edx %>%
  group_by(years_to_rate) %>%
  summarise(mean_RateError_userId_YearsToRate =
              mean(rate_error_movieId_userID))

mean_error_per_userId_genre_vec <- edx %>%
  group_by(genres) %>%
  summarise(mean_RateError_userId_genres =
              mean(rate_error_movieId_userID))

mean_error_per_userId_YearsToRate_vec %>%
  ggplot(aes(x = mean_RateError_userId_YearsToRate)) +
  geom_histogram(bins = 50, color = "black", fill = "skyblue") +
  labs( x = "Average Movie Ratings", y = "Years to Rate Count") +
  theme_bw()
```



```
mean_error_per_userId_genre_vec %>%
  ggplot(aes(x = mean_RateError_userId_genres)) +
  geom_histogram(bins = 50, color = "black", fill = "skyblue") +
  labs( x = "Average Movie Ratings", y = "Genre Count") +
  theme_bw()
```



It can be observed from the above histograms that, there is no considerable difference between selecting years_to_rate and the genres. Therefore, the variable genres is selected as the third variable for prediction error compensation.

2.3.5 Step 7 and 8 The average prediction error per genre has to be calculated to compensate the error in second prediction. After that edx dataset should be appended for each observation, the average error. The third rating prediction is the sum of second rating prediction and the appended average error. The average prediction error between the actual rating and the third rating prediction should then be calculated for each observation. The following code can be used to accomplish the all of this.

```
mean_error_per_userId_genre_vec <- edx %>%
  group_by(genres) %>%
  summarise(mean_RateError_userId_genres
            = mean(rate_error_movieId_userID))

edx <- edx %>%
  left_join(mean_error_per_userId_genre_vec, by = "genres")

edx <- edx %>%
  mutate(rate_predict_movieId_userId_genres =
           rate_predict_movieId_userId + mean_RateError_userId_genres) %>%
  mutate(rate_error_movieId_userID_genres =
           rating - rate_predict_movieId_userId_genres)
```

The calculated vector with name "mean_error_per_userId_genre_vec" is kept to append the final test set. This will be used for prediction based on movieId, userId and genre.

2.3.6 Step 9 The average prediction error per year_to_come has to be calculated to compensate the error in third prediction. After that edx dataset should be appended for each observation, the average error. The forth rating prediction is the sum of thrid rating prediction and the appended average error. The average prediction error between the actual rating and the third rating prediction should then be calculated for each observation. The following code can be used to accomplish the all of this.

```r
mean_error_per_userId_genre_YearsToRate_vec <- edx %>%
  group_by(years_to_rate) %>%
  summarise(mean_error_per_userId_genre_YearsToRate =
              mean(rate_error_movieId_userID_genres))

edx <- edx %>%
  left_join(mean_error_per_userId_genre_YearsToRate_vec,
            by = "years_to_rate")

edx <- edx %>%
  mutate(rate_predict_movieId_userId_genre_YearsToRate =
           rate_predict_movieId_userId_genres +
           mean_error_per_userId_genre_YearsToRate) %>%
  mutate(rate_error_movieId_userID_genre_YearsToRate =
           rating - rate_predict_movieId_userId_genre_YearsToRate)
```

The calculated vector with name "mean_error_per_userId_genre_YearsToRate_vec" is kept to append the final test set. This will be used for the final prediction based on movieId, userId, genre and years_to_rate.

2.3 Calculation of Training RMSE

The following code can be used to calculate the RMSEs for first prediction, second prediction, third prediction and the final prediction.

```r
RMSE_Train_movieId <- RMSE(edx$rating,
                           edx$mean_rate_movieId)

RMSE_Train_movieId_userID <- RMSE(edx$rating,
                                  edx$rate_predict_movieId_userId)

RMSE_Train_movieId_userID_genre <- RMSE(edx$rating,
                                        edx$rate_predict_movieId_userId_genres)

RMSE_Train_movieId_userID_genre_YearsToRate <- RMSE(edx$rating,edx$rate_predict_movieId_userId_genre_Yea
```

Based on the results obtained the following table highlights the training RMSE obtained for each prediction.

Prediction Level	Training RMSE
Prediction based on movieId	0.9423475 |
Prediction based on movieId and userId	0.8567039 |
Prediction based on movieId, userId and genre	0.8563595 |
Prediction based on movieId, userId, genre and years_to_rate	0.8559438 |

3. Results

This sections contains the calculation of RMSE Results for the test set. Before generating the predictions for the test set, the calculated average values should be appended to the final test set. Then predictions can be calculated for each observation of the test set. This can be accomplished using the following code.

```
final_holdout_test <- final_holdout_test %>%
  left_join(mean_rating_per_movieId_vec, by = "movieId")
final_holdout_test <- final_holdout_test %>%
  mutate(rate_predict_movieID = mean_rate_movieId)

final_holdout_test <- final_holdout_test %>%
  left_join(mean_error_per_userId_vec, by = "userId")
final_holdout_test <- final_holdout_test %>%
  mutate(rate_predict_movieID_userId = mean_rate_movieId +
           mean_RateError_userId)

final_holdout_test <- final_holdout_test %>%
  left_join(mean_error_per_userId_genre_vec, by = "genres")
final_holdout_test <- final_holdout_test %>%
  mutate(rate_predict_movieID_userId_genre = mean_rate_movieId +
           mean_RateError_userId + mean_RateError_userId_genres)

final_holdout_test <- final_holdout_test %>%
  left_join(mean_error_per_userId_genre_YearsToRate_vec, by = "years_to_rate")
final_holdout_test <- final_holdout_test %>%
  mutate(rate_predict_movieID_userId_genre_YearsToRate =
           mean_rate_movieId + mean_RateError_userId +
           mean_RateError_userId_genres +
           mean_error_per_userId_genre_YearsToRate)
```

The test RMSE can be calculated using the following code.

```
RMSE_Test_movieID <- RMSE(final_holdout_test$rating,
                          final_holdout_test$rate_predict_movieID)

RMSE_Test_movieID_userId <- RMSE(final_holdout_test$rating,
                                 final_holdout_test$rate_predict_movieID_userId)
RMSE_Test_movieID_userId_genre <- RMSE(final_holdout_test$rating,
                                  final_holdout_test$rate_predict_movieID_userId_genre)

RMSE_Test_movieID_userId_genre_YearsToRate <- RMSE(final_holdout_test$rating,
```

Based on the results obtained the following table highlights the testing RMSE obtained for each prediction.

Prediction Level	Testing RMSE
Prediction based on movieId	0.9439087 |
Prediction based on movieId and userId	0.8653488 |
Prediction based on movieId, userId and genre	0.8649469 |
Prediction based on movieId, userId, genre and years_to_rate	0.8645393 |
Final Prediction	

The final test prediction RMSE is

```
0.8645393
```

4. Conclusion

The final testing RMSE of the developed model is below the goal of RMSE < 0.86490, which satisfy the objective of the project.