MovieLens

Quyen Di Sabino

2024-07-28

A. Introduction

As we all know, recommendation systems are more important than ever, with many of the most popular services we use daily such as YouTube, Spotify, Amazon,…. Companies use these systems to provide users with content that is likely to be relevant and interesting to user, enhance user experience and satisfaction, thereby increasing the number of users or visits to their services. To achieve this, businesses need to build their own systems, using user data to analyze and understand their behavior and preferences.

Our challenge is to build a system to predict movie ratings for users in a large dataset http://files.grouplens.org/datasets/movielens/ml-10m.zip. We train a linear model to generate predicted movie ratings and calculate the Root Mean Squared Error (RMSE)

RMSE measures the average magnitude of the errors between predicted and actual values in a regression model. It gives a direct indication of prediction accuracy. We use RMSE for evaluating and comparing regression models, helping to assess how well a model fits the data and predicts outcomes. We use the built in function rmse() to calculate our RMSEs.

Here we assume that the process of collecting data from users is complete. As data scientists, we will detail the processes by which this data is used through exploring, visualizing, analyzing to find underlying patterns of the data. From there, we develop machine learning models, until we achieve the set goal.

With final model found above, a movie recommendation system built. This system can predict exactly or nearly exactly the rating of a movie. It then makes movie recommendations for users based on their previous ratings and preferences. Therefore, users discover new and relevant content, making their entertainment experience more effective and enjoyable.

Our dataset include:

1. movies: Contains information about movies including movie movieId, title, genres

    - movieId: the movie ID
    - title: the movie title
    - genres: the movie genres

2. ratings: Contains userId, movieId, rating, timestamp

    - userId: the user ID, used to extract user behavior and preferences
    - rating: the rate that the user gave to a particular movie (movieId)
    - timestamp: the time when the user rate a particular movie (movieId)

These two datasets then being combined into 'movielens' set - contains userId, movieId, rating, timestamp, title, genres.

B. Data analysis

1. edx set overview

```
# Controls the number of digits to print when printing numeric values
options(digits = 6)

# edx_summary table. Give an over view of edx data set
edx_summary <- data.frame(n_rows = nrow(edx),
                          n_columns = ncol(edx),
                          n_users = n_distinct(edx$userId),
                          n_movies = n_distinct(edx$movieId),
                          average_rating = round(mean(edx$rating),2),
                          n_genres = n_distinct(edx$genres),
                          first_rating_date = date(as_datetime(min(edx$timestamp), origin = "1970-01-01
                          last_rating_date = date(as_datetime(max(edx$timestamp), origin = "1970-01-01")
                          )

# Print table of edx_summary
edx_summary
```

```
##    n_rows n_columns n_users n_movies average_rating n_genres first_rating_date
## 1 9000055         6   69878    10677           3.51      797        1995-01-09
##   last_rating_date
## 1       2009-01-05
```

```
# edx structure
str(edx)
```

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

```
# Print first six rows of edx set
head(edx)
```

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
## 7      1     355      5 838984474       Flintstones, The (1995)
```

```
##                               genres
## 1                     Comedy|Romance
## 2             Action|Crime|Thriller
## 4  Action|Drama|Sci-Fi|Thriller
## 5           Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
## 7         Children|Comedy|Fantasy
```

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

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

Zero missing value

2. Explore rating feature

```r
# Unique ratings list
unique_ratings <- sort(unique(edx$rating))
unique_ratings
```

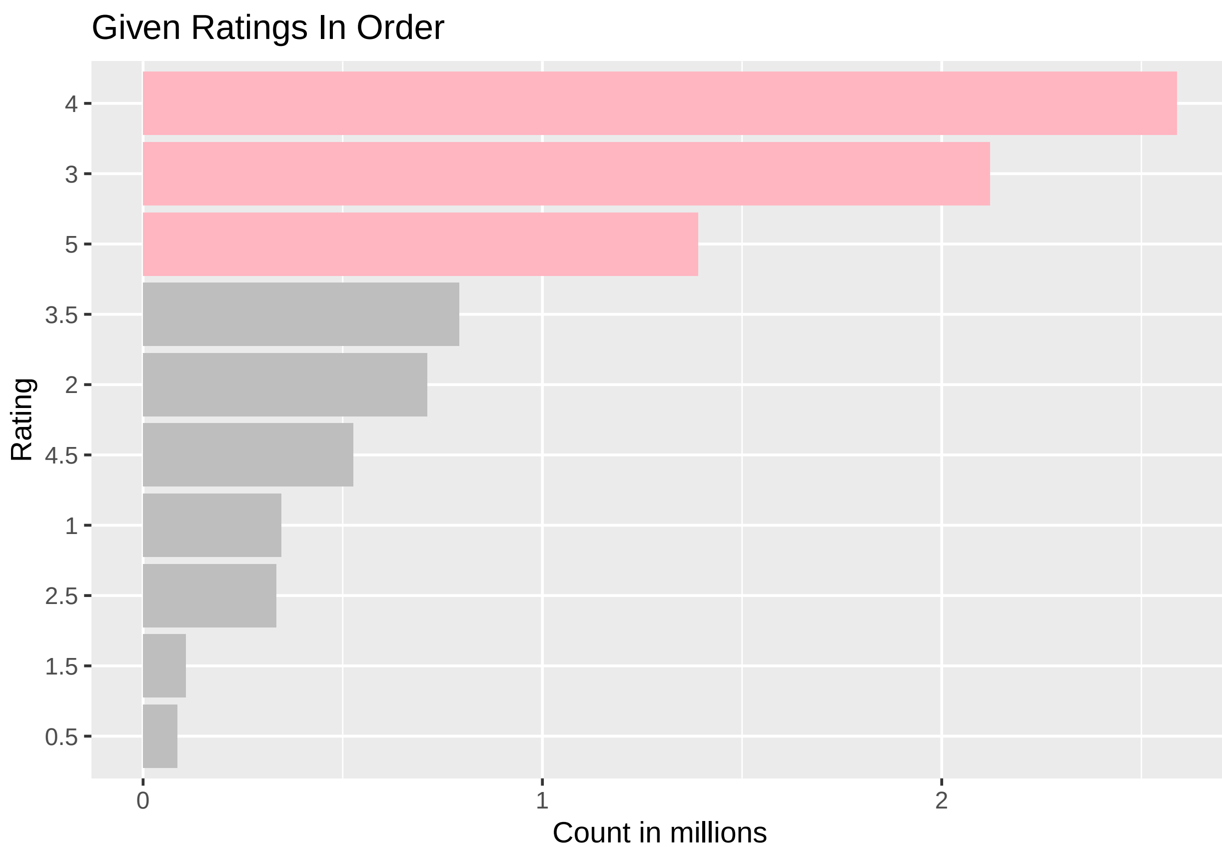
```
##  [1] 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

No zero ratings

```r
# Ratings distribution tibble
ratings_distribution <- edx %>%
  group_by(rating) %>%
  summarize(count = n()) %>%
  arrange(desc(count))
ratings_distribution
```

```
## # A tibble: 10 x 2
##    rating   count
##     <dbl>   <int>
##  1    4   2588430
##  2    3   2121240
##  3    5   1390114
##  4    3.5  791624
##  5    2    711422
##  6    4.5  526736
##  7    1    345679
##  8    2.5  333010
##  9    1.5  106426
## 10    0.5   85374
```

```r
# Most to lease given ratings plot
ratings_distribution %>%
  mutate(rating = factor(rating), rank = ifelse(rating %in% c(3,4,5),"high","low")) %>%
  ggplot(aes(x = reorder(rating, count), y = count/10^6, fill = rank)) +
  geom_bar(stat = "identity") +
  scale_fill_manual(values = c("lightpink","grey")) +
  theme(legend.position = "none") +
  ggtitle("Given Ratings In Order") +
  xlab("Rating") +
  ylab("Count in millions") +
  coord_flip()
```



4, 3, 5 have most given ratings.

3. Explore ratings per movie

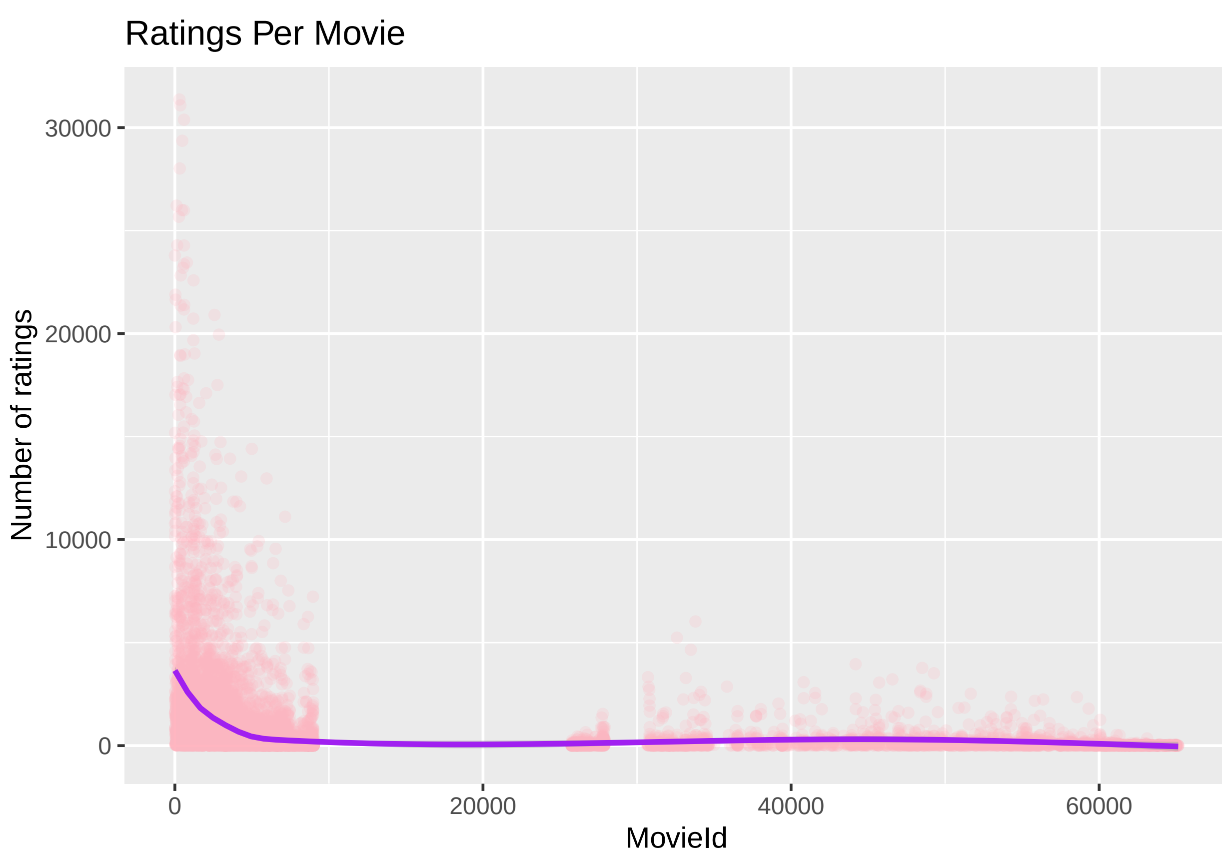
```r
# Ratings per movie tibble
ratings_per_movie <- edx %>%
  group_by(movieId) %>%
  summarize(n_ratings = n(),
            avg_rating = mean(rating)) %>%
  arrange(desc(n_ratings))
ratings_per_movie
```

```
## # A tibble: 10,677 x 3
```

```
##    movieId n_ratings avg_rating
##      <int>     <int>      <dbl>
##  1     296     31362       4.15
##  2     356     31079       4.01
##  3     593     30382       4.20
##  4     480     29360       3.66
##  5     318     28015       4.46
##  6     110     26212       4.08
##  7     457     25998       4.01
##  8     589     25984       3.93
##  9     260     25672       4.22
## 10     150     24284       3.89
## # i 10,667 more rows
```

```r
# Number of ratings per movie plot
ratings_per_movie %>%
  ggplot(aes(x = movieId, y = n_ratings)) +
  geom_point(alpha = 0.2, color = "lightpink") +
  geom_smooth(color = "purple") +
  ggtitle("Ratings Per Movie") +
  xlab("MovieId") +
  ylab("Number of ratings")
```

```
## 'geom_smooth()' using method = 'gam' and formula = 'y ~ s(x, bs = "cs")'
```

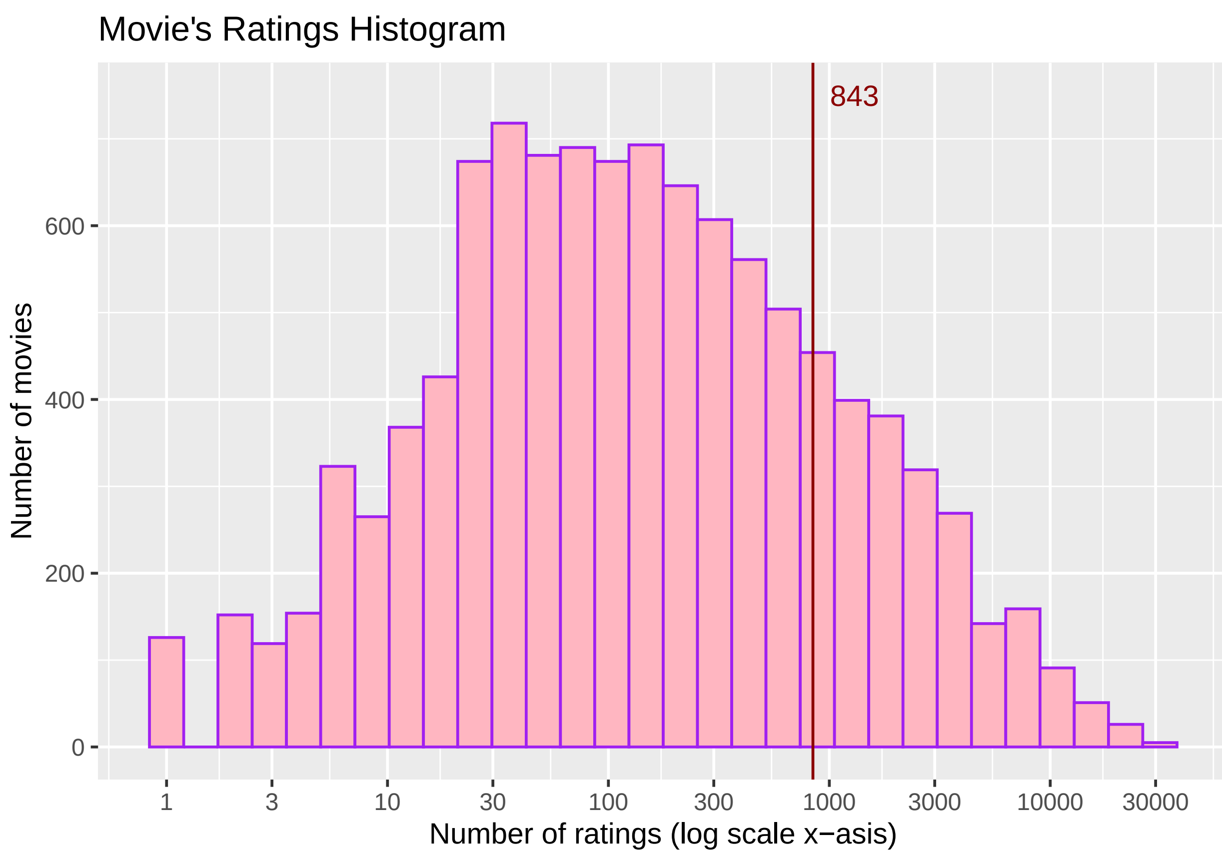


There is no movies with ID between 10000 and 25000. Some movies with smaller movieId have higher number of ratings

```r
# Movie's Ratings Histogram
ratings_per_movie %>%
  ggplot(aes(x = n_ratings)) +
  geom_histogram(fill = "lightpink", color = "purple") +
  # include average ratings per movie in plot.
  geom_vline(aes(xintercept = mean(n_ratings)), color = "darkred") +
  annotate("text", x = 1300, y = 750,
           label = print(round(mean(ratings_per_movie$n_ratings),0)),
           color = "darkred", size = 4) +
  ggtitle("Movie's Ratings Histogram") +
  xlab("Number of ratings (log scale x-asis)") +
  ylab("Number of movies") +
  # aipplies a logarithmic transformation to the x-axis with 10 breaks
  scale_x_log10(n.breaks = 10)
```

```
## [1] 843
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



It's a nearly symmetric plot, large ratings probably for blockbuster movies. There is about 843 ratings per movie in average.

```r
summary(ratings_per_movie$n_ratings)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       1      30     122     843     565   31362
```

Half the movies are rated between 30 and 565 times

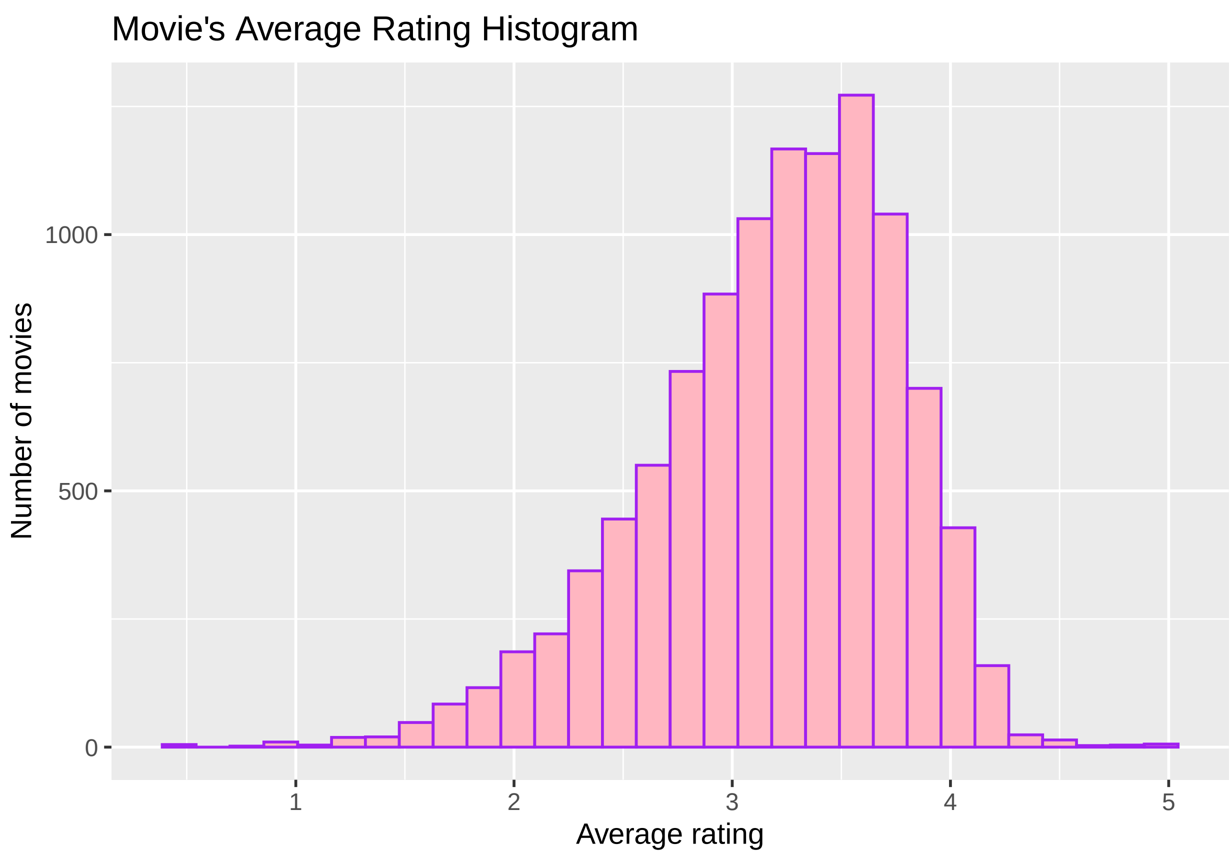
```
quantile(ratings_per_movie$n_ratings,
         probs = 0.75,
         na.rm = TRUE)
```

```
## 75%
## 565
```

There are 75% of the movies are rated less than or equal to 565 times

```
# Movie's Average Rating Histogram
ratings_per_movie %>%
  ggplot(aes(x = avg_rating)) +
  geom_histogram(fill = "lightpink", color = "purple") +
  ggtitle("Movie's Average Rating Histogram") +
  xlab("Average rating") +
  ylab("Number of movies")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Left skewness indicates that there are some movies with lower rating values. These movies are pulling overall avg rating to the left.

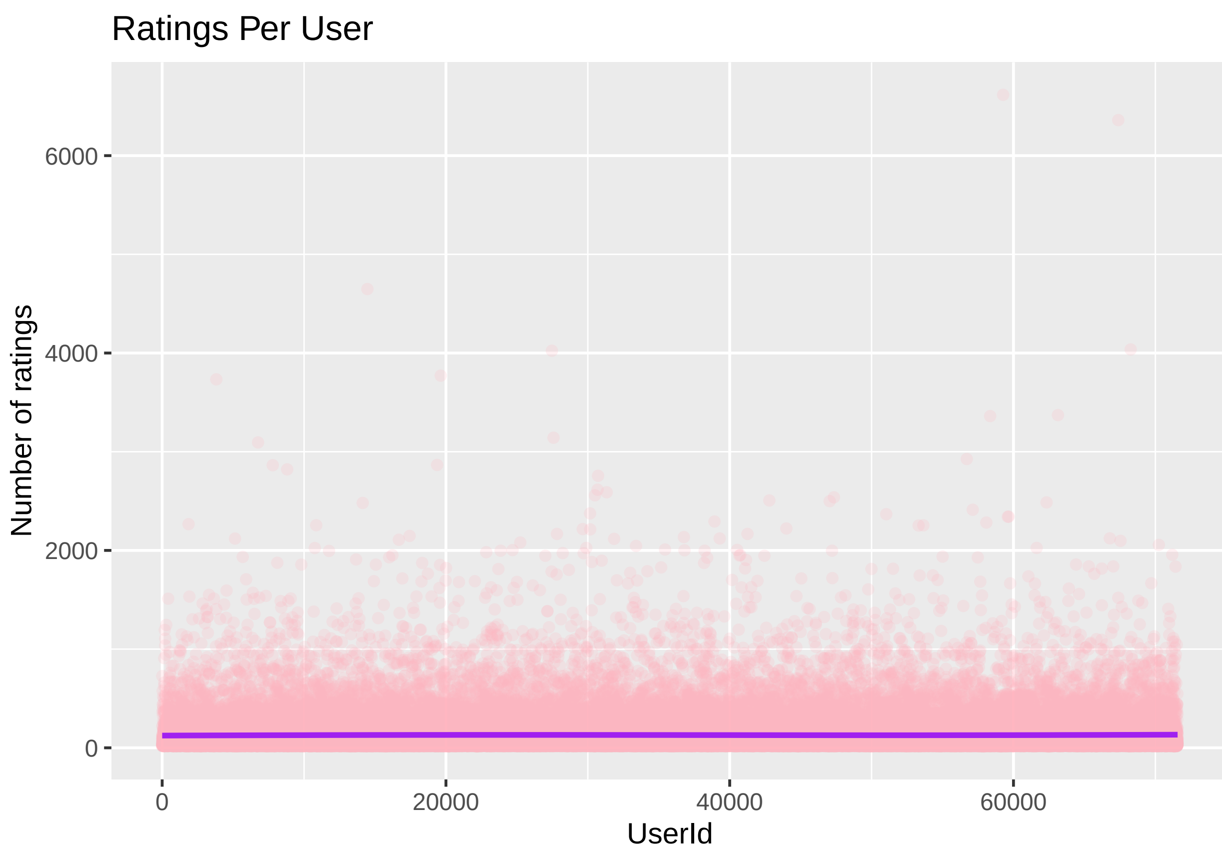
4. Explore ratings per user

```
# Ratings per user tibble
ratings_per_user <- edx %>%
  group_by(userId) %>%
  summarize(n_ratings = n(),
            avg_rating = mean(rating)) %>%
  arrange(desc(n_ratings))
ratings_per_user
```

```
## # A tibble: 69,878 x 3
##    userId n_ratings avg_rating
##     <int>     <int>      <dbl>
##  1  59269      6616       3.26
##  2  67385      6360       3.20
##  3  14463      4648       2.40
##  4  68259      4036       3.58
##  5  27468      4023       3.83
##  6  19635      3771       3.50
##  7   3817      3733       3.11
##  8  63134      3371       3.27
##  9  58357      3361       3.00
## 10  27584      3142       3.00
## # i 69,868 more rows
```

```
# Number of ratings per user plot
ratings_per_user %>%
  ggplot(aes(x = userId, y = n_ratings)) +
  geom_point(alpha = 0.2, color = "lightpink") +
  geom_smooth(color = "purple") +
  ggtitle("Ratings Per User") +
  xlab("UserId") +
  ylab("Number of ratings")
```

```
## `geom_smooth()` using method = 'gam' and formula = 'y ~ s(x, bs = "cs")'
```

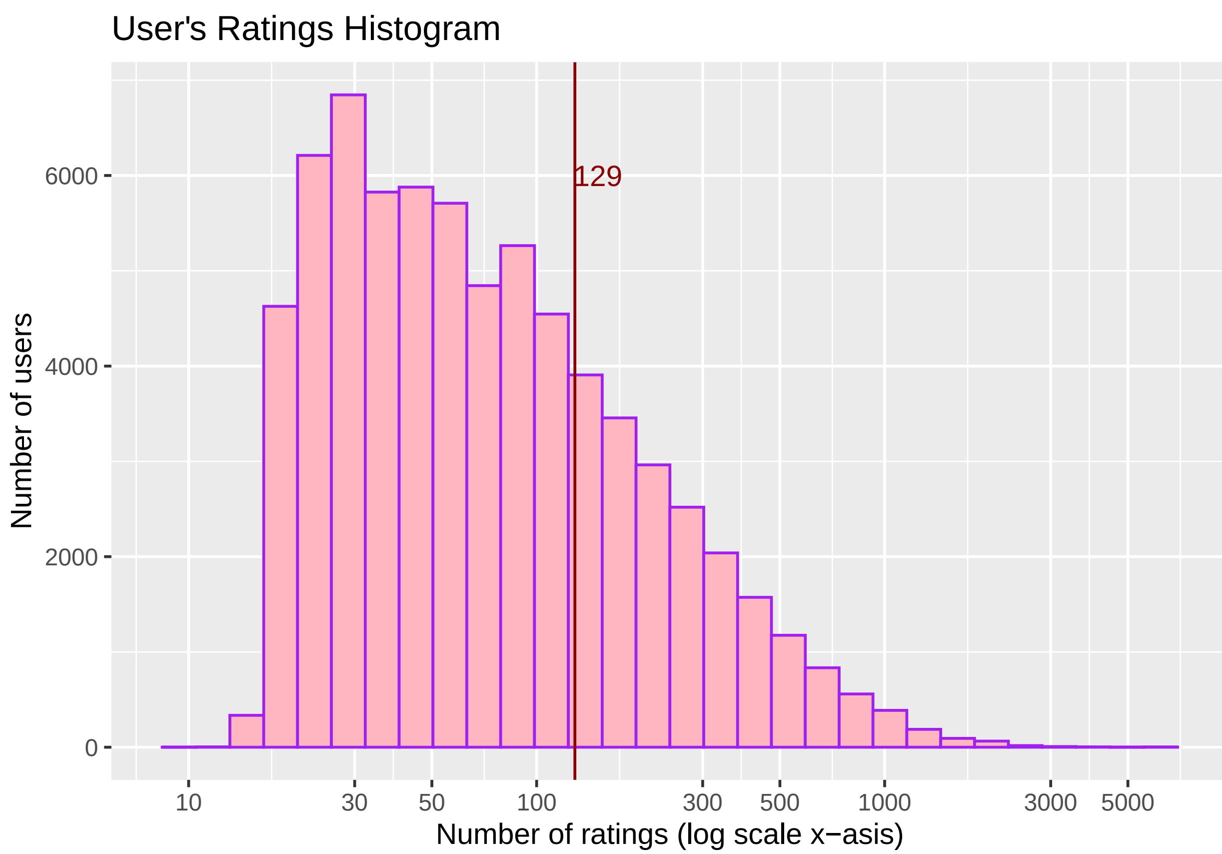


Majority of users have rated less than 1000 movies. There are some outliers

```
# User's Ratings Histogram
ratings_per_user %>%
  ggplot(aes(x = n_ratings)) +
  geom_histogram(fill = "lightpink", color = "purple") +
  # include average ratings per user in plot
  geom_vline(aes(xintercept = mean(n_ratings)), color = "darkred")+
  annotate("text", x = 150, y = 6000,
           label = print(round(mean(ratings_per_user$n_ratings),0)),
           color = "darkred", size = 4) +
  ggtitle("User's Ratings Histogram") +
  xlab("Number of ratings (log scale x-asis)") +
  ylab("Number of users") +
  scale_x_log10(n.breaks = 10)
```

```
## [1] 129
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Right skewness indicates that not many users rated large number of movies. Some users are more active than others at rating movies. There is about 129 ratings rated by a user in average.

```
summary(ratings_per_user$n_ratings)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      10      32      62     129     141    6616
```

Haft of the users rated between 32 and 141 movies. There are 6616 ratings by a user, that could be an outlier

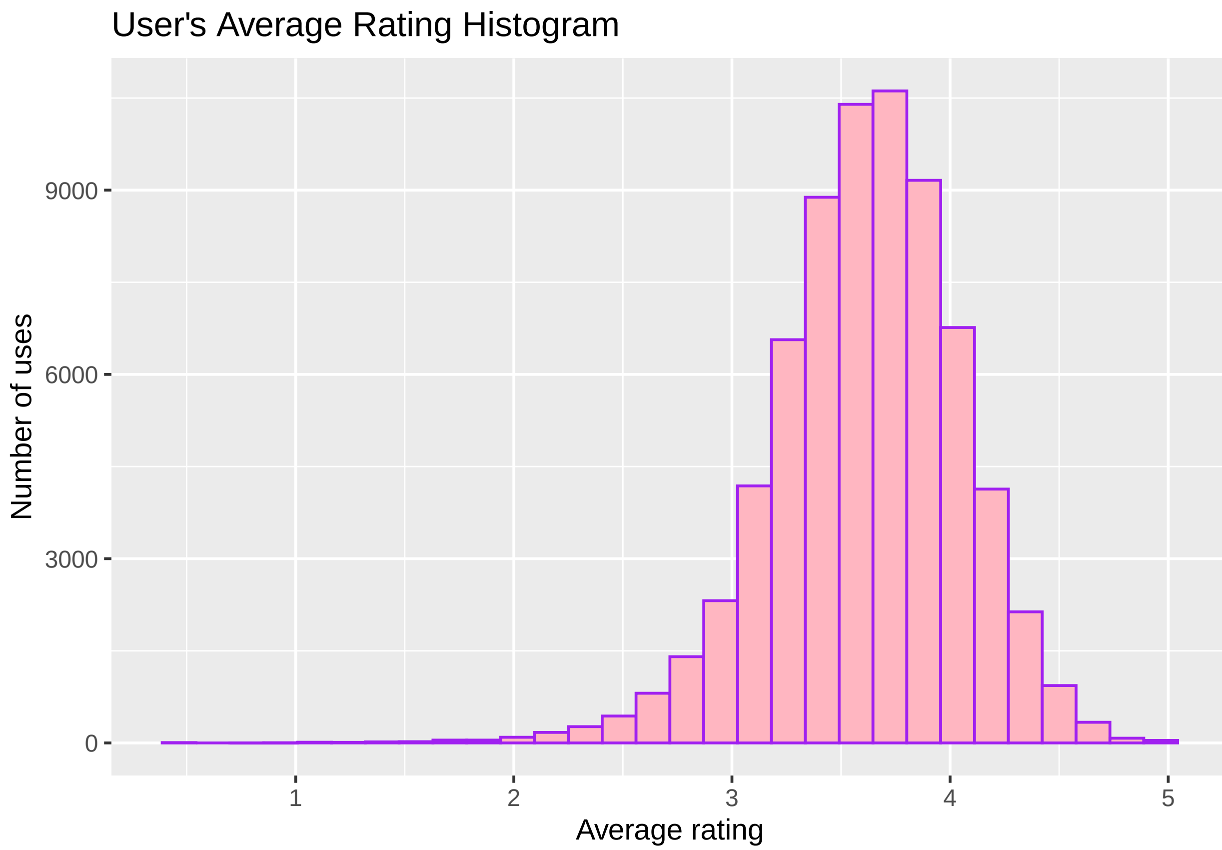
```
quantile(ratings_per_user$n_ratings,
         probs = 0.75,
         na.rm = TRUE)
```

```
## 75%
## 141
```

There are 75% of users rated less than or equal to 141 movies

```
# User's Average rating histogram
ratings_per_user %>%
  ggplot(aes(x = avg_rating, )) +
  geom_histogram(fill = "lightpink", color = "purple") +
  ggtitle("User's Average Rating Histogram") +
  xlab("Average rating") +
  ylab("Number of uses")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Symmetric histogram indicates user's avg_rating is nearly normal distribution.

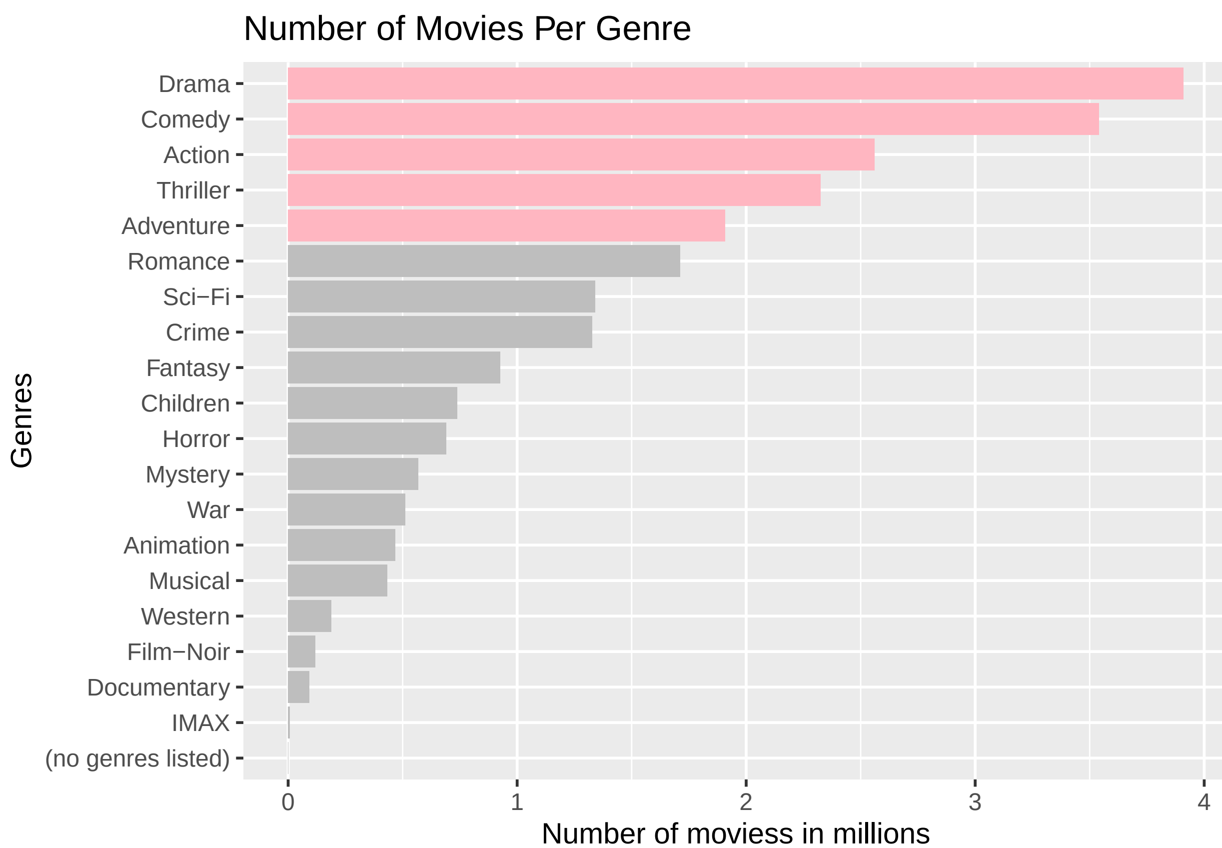
5. Explore genres feature and ratings per genre

```
# List of genres
# Codes gotten from the answer to Q5 of Quiz: MovieLens Dataset
genres <- edx %>%
  separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>%
  summarize(n_movies = n()) %>%
  arrange(desc(n_movies))

# Genres tibble
genres <- genres %>%
  mutate(avg_rating = sapply(genres, function(g) {
    ind <- which(str_detect(edx$genres, g))
    round(mean(edx$rating[ind]),2)
    }))
genres
```

```
## # A tibble: 20 x 3
##    genres             n_movies avg_rating
```

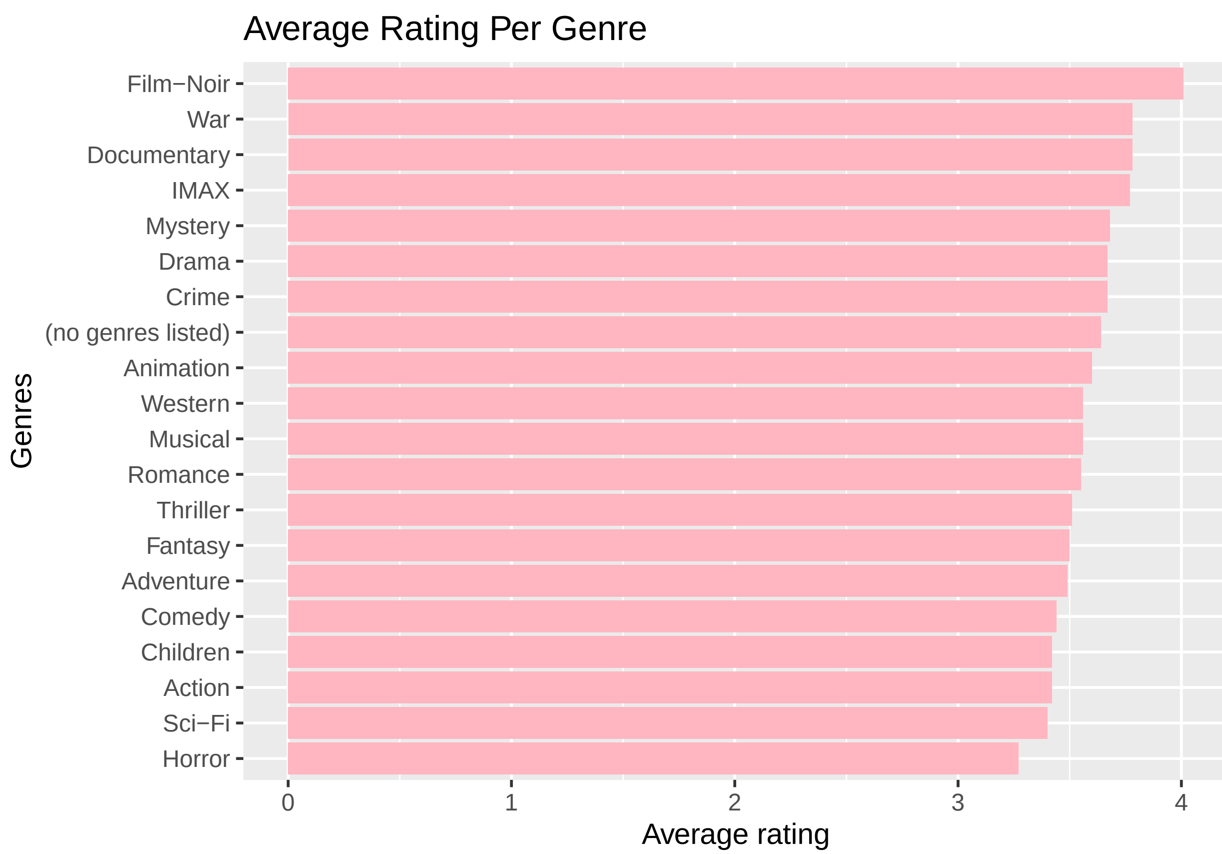
```
##    <chr>                   <int>      <dbl>
##  1 Drama                  3910127       3.67
##  2 Comedy                 3540930       3.44
##  3 Action                 2560545       3.42
##  4 Thriller               2325899       3.51
##  5 Adventure              1908892       3.49
##  6 Romance                1712100       3.55
##  7 Sci-Fi                 1341183       3.4
##  8 Crime                  1327715       3.67
##  9 Fantasy                 925637       3.5
## 10 Children                737994       3.42
## 11 Horror                  691485       3.27
## 12 Mystery                 568332       3.68
## 13 War                     511147       3.78
## 14 Animation               467168       3.6
## 15 Musical                 433080       3.56
## 16 Western                 189394       3.56
## 17 Film-Noir               118541       4.01
## 18 Documentary              93066       3.78
## 19 IMAX                      8181       3.77
## 20 (no genres listed)           7       3.64
```

```r
# Five genres with highest n_movies
top5_genres <- head(genres,5)$genres

# Number of movies per genres plot
genres %>%
  mutate(top5 = ifelse(genres %in% top5_genres, "top5","non")) %>%
  ggplot(aes(x = reorder(genres, n_movies), y = n_movies/10^6, fill = top5)) +
  geom_bar(stat = "identity") +
  scale_fill_manual(values = c("grey","lightpink")) +
  theme(legend.position = "none") +
  ggtitle("Number of Movies Per Genre") +
  xlab("Genres") +
  ylab("Number of moviess in millions") +
  coord_flip()
```



Drama, Comedy, Action, Thriller and Adventure are five genres with highest number of movies made

```
# Average rating per genre plot
genres %>%
  ggplot(aes(x = reorder(genres, avg_rating), avg_rating)) +
  geom_bar(stat = "identity", fill= "lightpink") +
  ggtitle("Average Rating Per Genre") +
  xlab("Genres") +
  ylab("Average rating") +
  coord_flip()
```



The plot shows that most genres were given a rating between 3 and 4. Horror has the worst rating

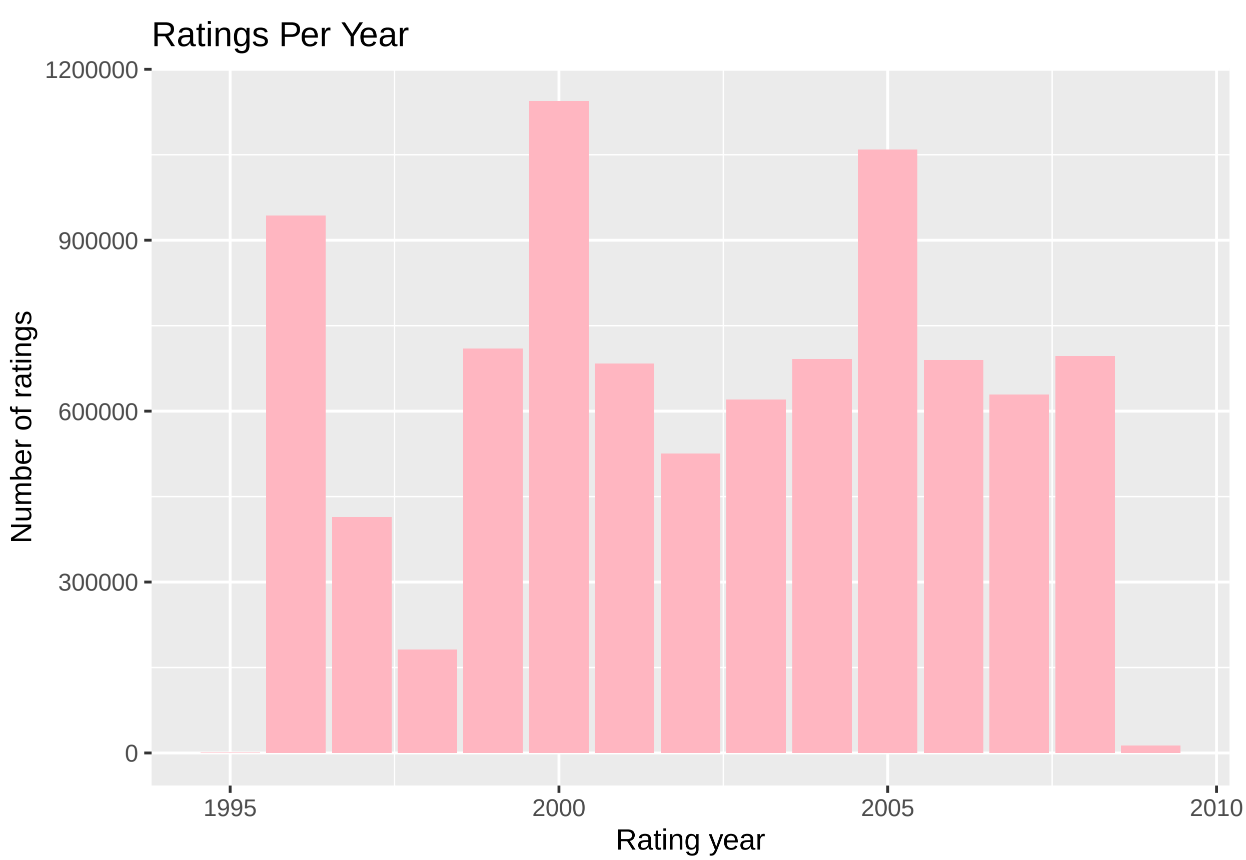
6. Explore timestamp feature

```
# Add "release_year" column, tells the year a movie was released, extracted from title
edx <- edx %>%
  # extract year from title feature.
  mutate(release_year = str_sub(title, start = -5, end = -2) %>% as.integer()) %>%
  # remove year from movie's title.
  mutate(title = str_sub(title, 1, -8))

# Add "rating_year" column, tells the year a movie was rated in, extracted from timestamp
edx <- edx %>%
  mutate(rating_year = year(as_datetime(timestamp, origin = "1970-01-01")) %>% as.integer())

# rating_year summary tibble
rating_year_sum <- edx %>%
  group_by(rating_year) %>%
  summarize(n_ratings = n(),
            avg_rating = mean(rating)) %>%
  select(rating_year, n_ratings, avg_rating)
rating_year_sum
```

```
## # A tibble: 15 x 3
##    rating_year n_ratings avg_rating
```

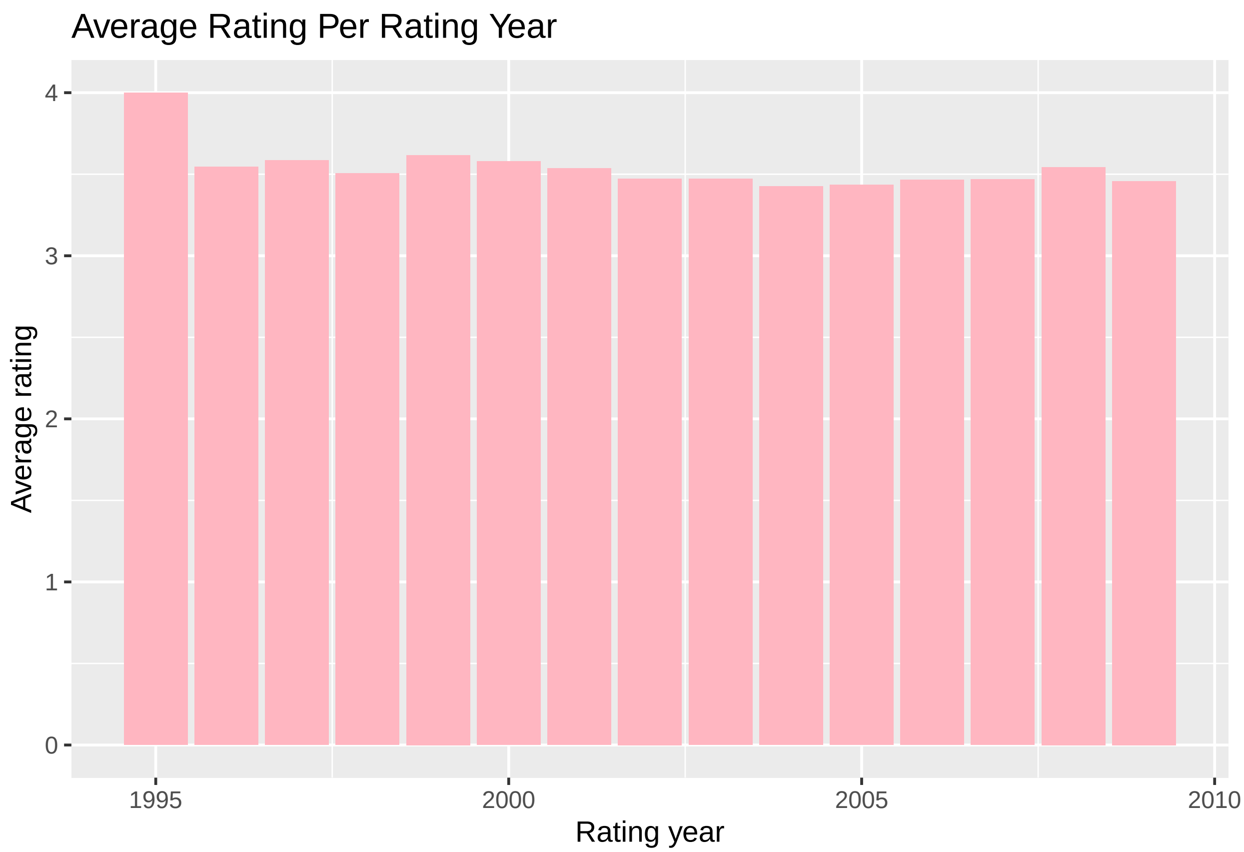
```
##           <int>     <int>      <dbl>
##   1       1995         2       4
##   2       1996    942772       3.55
##   3       1997    414101       3.59
##   4       1998    181634       3.51
##   5       1999    709893       3.62
##   6       2000   1144349       3.58
##   7       2001    683355       3.54
##   8       2002    524959       3.47
##   9       2003    619938       3.47
## 10       2004    691429       3.43
## 11       2005   1059277       3.44
## 12       2006    689315       3.47
## 13       2007    629168       3.47
## 14       2008    696740       3.54
## 15       2009     13123       3.46
```

```r
# Number of ratings per rating_year plot
rating_year_sum %>%
  ggplot(aes(x = rating_year, y = n_ratings)) +
  geom_bar(stat = "identity", fill = "lightpink") +
  ggtitle("Ratings Per Year") +
  xlab("Rating year") +
  ylab("Number of ratings")
```



The two years of 1997 and 1998 have low number of rating. Data for 2009 may be incomplete

```r
# Average rating per rating_year plot
rating_year_sum %>%
  ggplot(aes(x = rating_year, y = avg_rating)) +
  geom_bar(stat = "identity", fill = "lightpink") +
  ggtitle("Average Rating Per Rating Year") +
  xlab("Rating year") +
  ylab("Average rating")
```



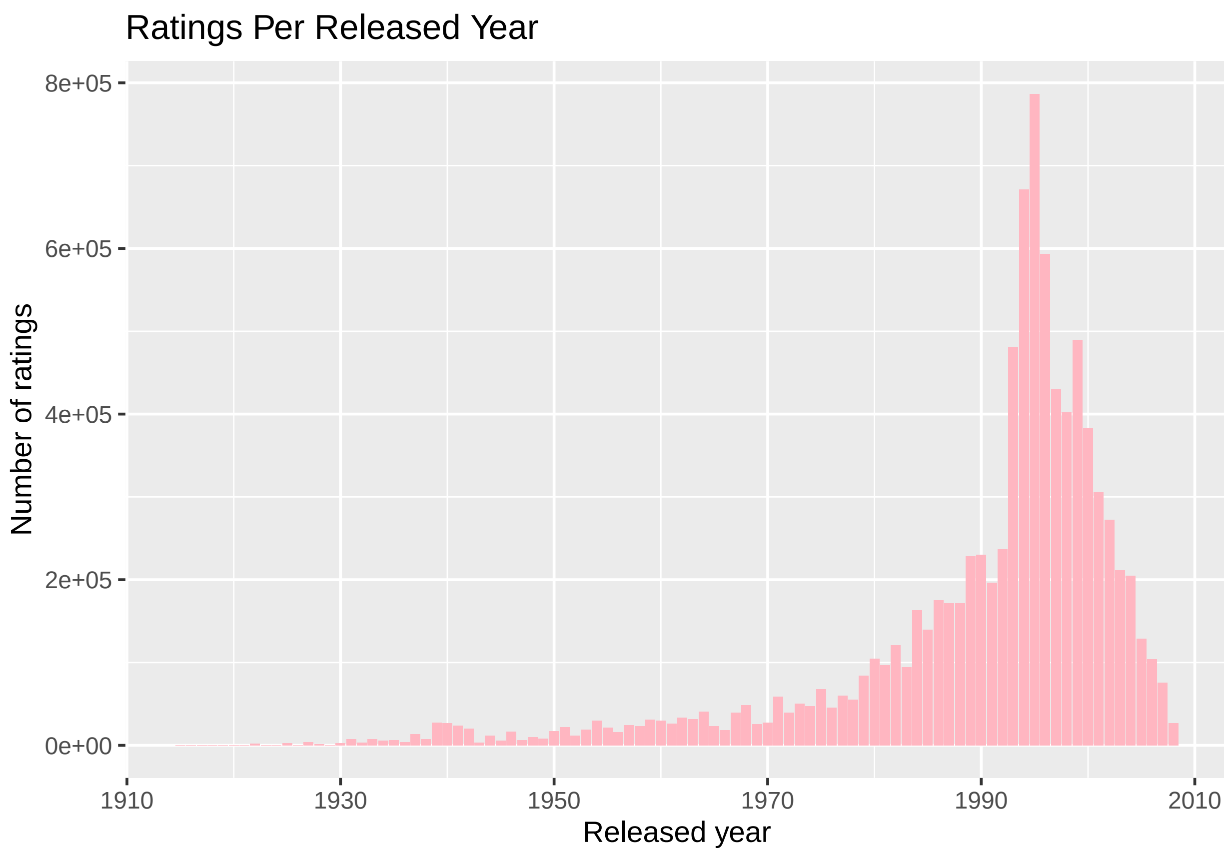
User ratings are not significantly affected by time.

```r
# release_year summary tibble
release_year_sum <- edx %>%
  group_by(release_year) %>%
  summarize(n_ratings = n(),
            avg_rating = mean(rating)) %>%
  select(release_year, n_ratings, avg_rating)
release_year_sum
```

```
## # A tibble: 94 x 3
##    release_year n_ratings avg_rating
##           <int>     <int>      <dbl>
##  1         1915       180       3.29
##  2         1916        84       3.83
##  3         1917        32       3.73
##  4         1918        73       3.65
##  5         1919       158       3.28
```

```
## 6        1920     575     3.94
## 7        1921     406     3.83
## 8        1922    1825     3.9
## 9        1923     316     3.78
## 10       1924     457     3.94
## # i 84 more rows
```

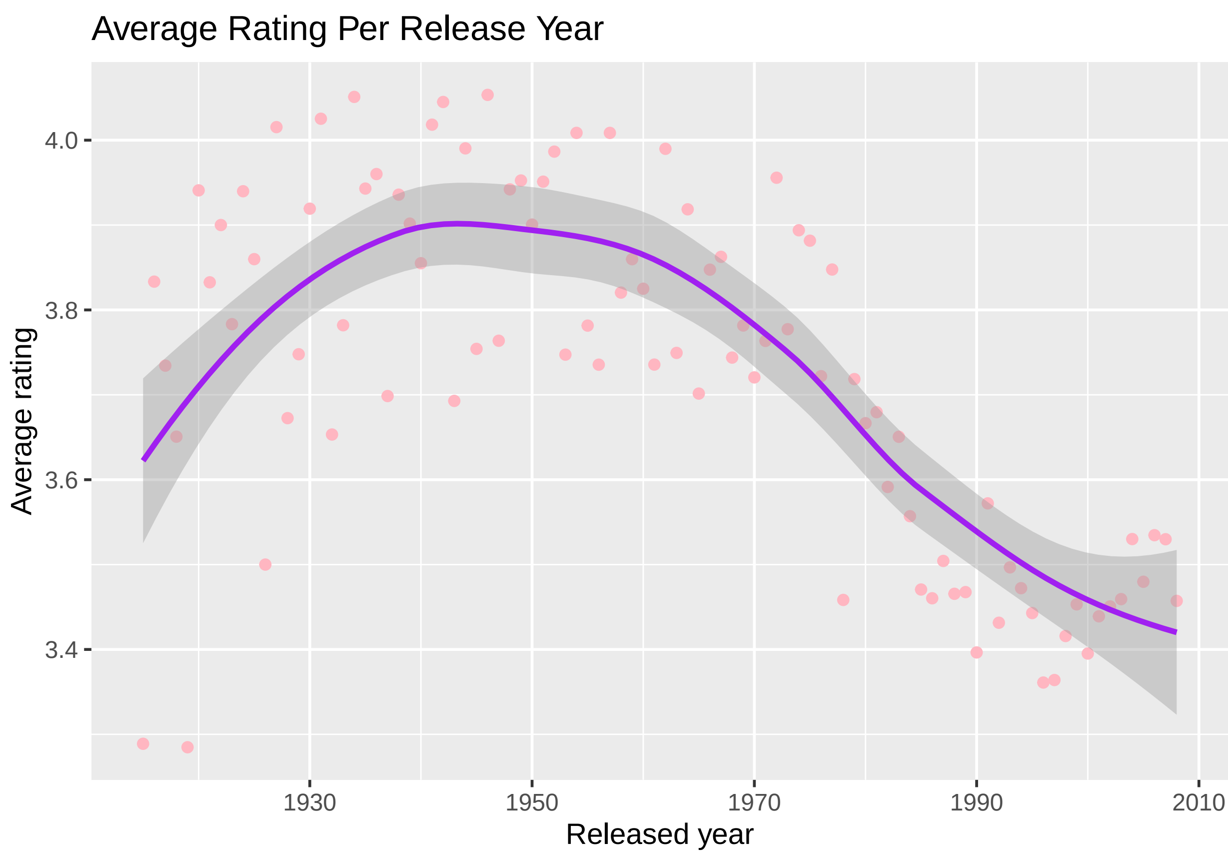
```
# Number of ratings per released_year plot
release_year_sum %>%
  ggplot(aes(x = release_year, y = n_ratings)) +
  geom_bar(stat = "identity", fill = "lightpink") +
  ggtitle("Ratings Per Released Year") +
  xlab("Released year") +
  ylab("Number of ratings")
```



Left skewness indicates movies released before 1980 have fewer number of rating

```
# Average rating per release_year plot
release_year_sum %>%
  ggplot(aes(x = release_year, y = avg_rating)) +
  geom_point(color = "lightpink") +
  geom_smooth(color = "purple") +
  ggtitle("Average Rating Per Release Year") +
  xlab("Released year") +
  ylab("Average rating")
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

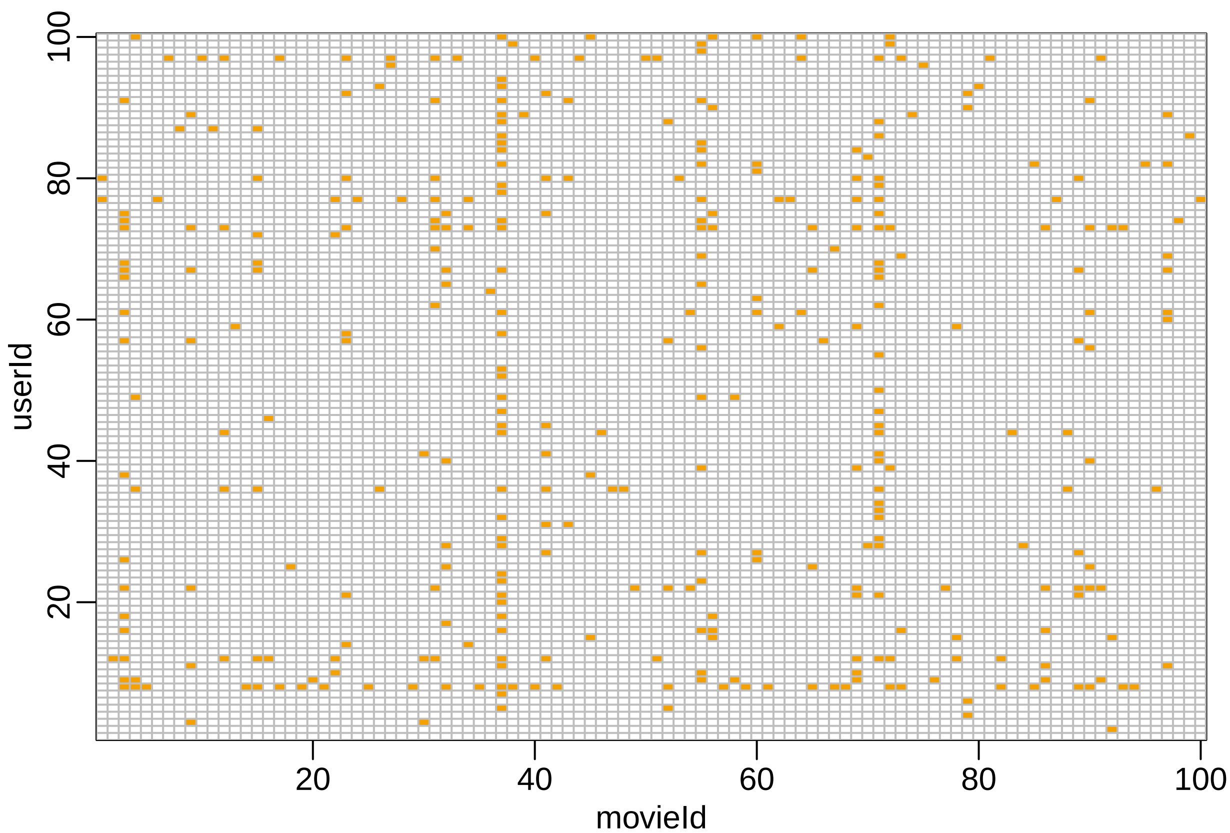


Movies released between 1920 to 1980 seem to have higher average rating than movies released later years

C. Methods

"This machine learning challenge is complicated because each outcome 'y' has a different set of predictors. Note that if we are predicting the rating for movie 'i' by user 'u', [. . .] we may be able to use information from other movies that we have determined are similar to movie 'i' or from users determined to be similar to user 'u'.[. . .]" Fore more details in code and text, please see **here**

```
#The matrix for a random sample of 100 movies and 100 users
users <- sample(unique(edx$userId), 100)
mypar() # optimizes graphical parameters
edx %>% filter(userId %in% users) %>%
  select(userId, movieId, rating) %>%
  mutate(rating = 1) %>%
  spread(movieId, rating) %>%
  select(sample(ncol(.), 100)) %>%
  as.matrix() %>%
  t(.) %>% # transpose the metrix
  image(1:100, 1:100, . , xlab="movieId", ylab="userId") +
  abline(h = 0:100+0.5, v = 0:100+0.5, col = "grey")
```




```
## integer(0)
```

Yellow indicates a user/movie combination for which we have a rating. In essence, the entire matrix can be used as predictors for each cell.

1. Target: RMSE < 0.86490

```
# Create a tibble to store RMSEs, rmse_results
rmse_results <- tibble(Model = "Target: less than",
                       RMSE = signif(0.86490, 5)
                       )

# Format RMSE column as numeric
rmse_results$RMSE <- rmse_results$RMSE %>% as.numeric()

# Print rmse_results tibble
rmse_results %>% kable()
```

Model	RMSE
Target: less than	0.8649

2. Mean baseline model y_hat = mu + e(u,i)

This model assumes the same rating mu for all movies regardless of users, with all the differences explained by random variation. y_hat = mu + e(u, i)
Here:
mu = overall average rating of edx set, represents the predicted rating for all movies regardless of users
e(u, i) represents independent errors sampled from the same distribution, centered at zero

```
# Calculate average rating across training data, mu
mu <- mean(edx_train$rating)

# Predict all ratings with mu
y_hat_mu <- rep(mu, nrow(edx_test))

# calculate the RMSE then add on to rmse_results tibble
rmse_results <- rbind(rmse_results,
                      tibble(Model = "Mean baseline",
                             RMSE = signif(rmse(edx_test$rating, y_hat_mu), 5)
                      )
)

# Print rmse_results tibble
rmse_results %>% kable()
```

Model	RMSE
Target: less than	0.8649
Mean baseline	1.0601

3. Median baseline model y_hat = med + e(u,i)

This model assumes the same rating med for all movies regardless of users, with all the differences explained by random variation. y_hat = med + e(u, i)
Here:
med = median rating of edx set, represents the predicted rating for all movies regardless of users

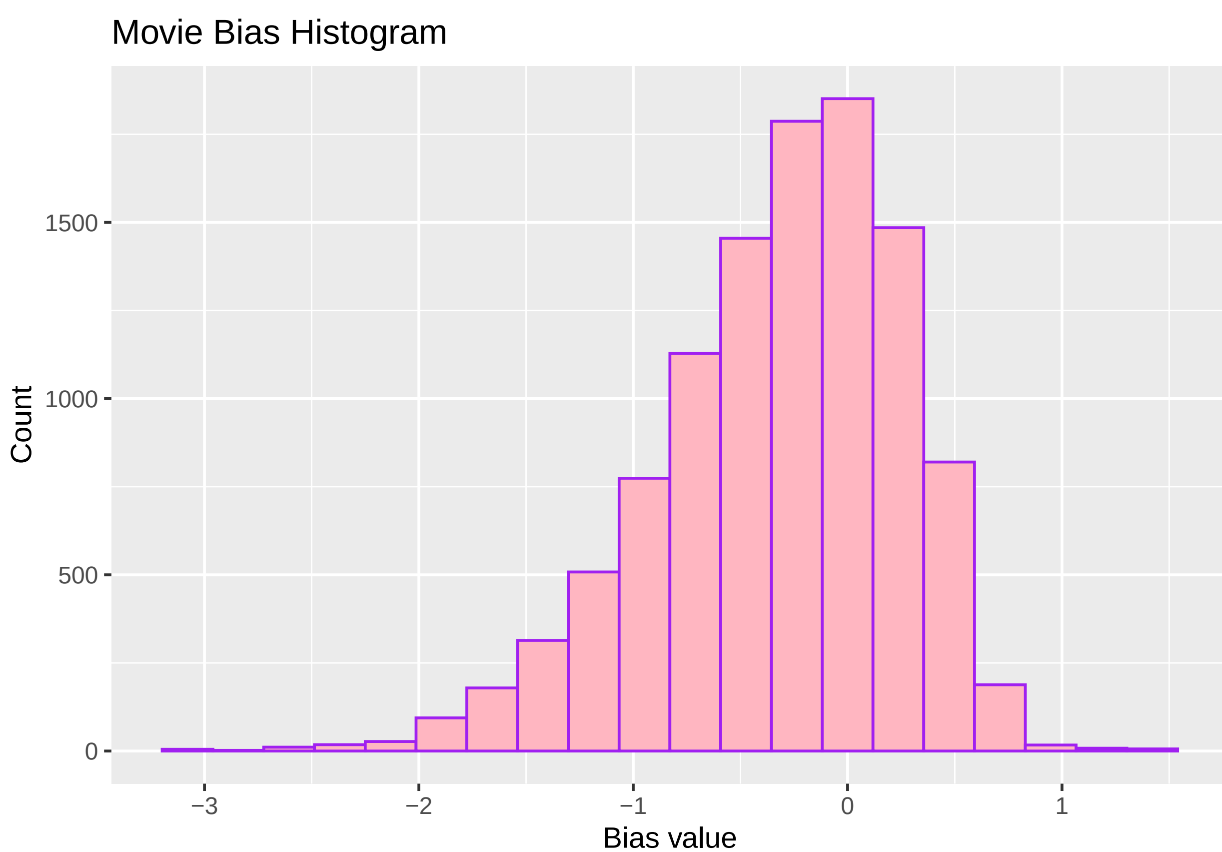
```
# Find median rating across training data, med
med <- median(edx_train$rating)

# Predict all ratings with median
y_hat_med <- rep(med, nrow(edx_test))

# Calculate the RMSE
rmse_results <- rbind(rmse_results,
                      tibble(Model = "Median baseline",
                             RMSE = signif(rmse(edx_test$rating, y_hat_med), 5)
                      )
)

# Print rmse_results tibble
rmse_results %>% kable()
```

Model	RMSE
Target: less than	0.8649
Mean baseline	1.0601
Median baseline	1.1668

4. Movie bias model y_hat = mu + e(u,i) + b_i

We can improve our model by adding movie bias b_i, represents movie-specific effect. y_hat = mu + e(u,i) + b_i

```
# Compute movie bias term, b_i
b_i <- edx_train %>%
  group_by(movieId) %>%
  summarise(b_i = mean(rating - mu))

# b_i plot
b_i %>%
  ggplot(aes(x = b_i)) +
  geom_histogram(bins = 20, fill = "lightpink", color = "purple") +
  ggtitle("Movie Bias Histogram") +
  xlab("Bias value") +
  ylab("Count")
```



These estimates vary substantially

```r
# Predict all ratings with mu + b_i
y_hat_bi <- mu + edx_test %>%
  left_join(b_i, by = "movieId") %>%
  .$b_i

# Calculate RMSE with movie ranking effect
rmse_results <- rbind(rmse_results,
                      tibble(Model = "Mean + Movie bias",
                             RMSE = signif(rmse(edx_test$rating, y_hat_bi), 5)
                      )
)

# Print rmse_results tibble
rmse_results %>% kable()
```

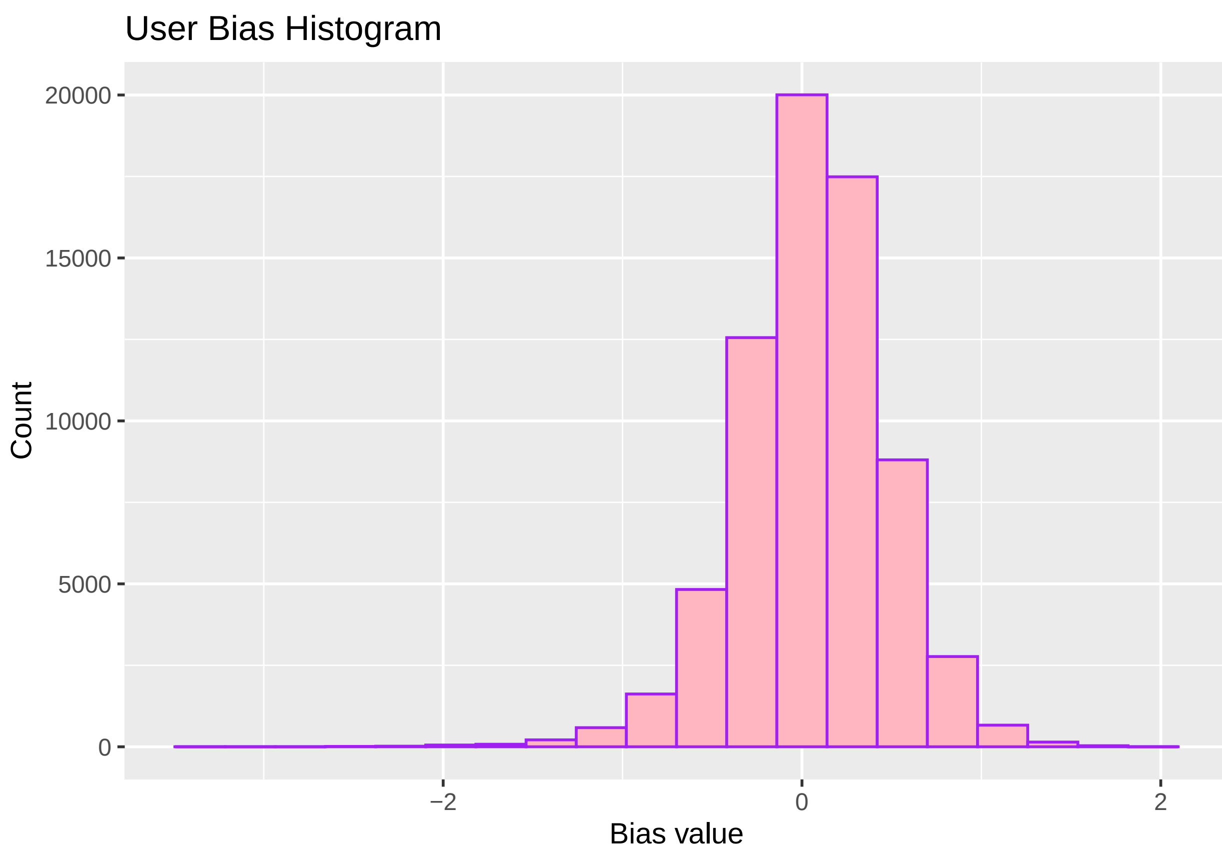
Model	RMSE
Target: less than	0.86490
Mean baseline	1.06010
Median baseline	1.16680
Mean + Movie bias	0.94296

Notice the improvement of RMSE

5. Movie and user bias model y_hat = mu + e(u,i) + b_i + b_u

We can further improve our model by adding user bias b_u, represents user-specific effect. y_hat = mu + e(u,i) + b_i + b_u

```r
# Compute user bias term, b_u
b_u <- edx_train %>%
  left_join(b_i, by = 'movieId') %>%
  group_by(userId) %>%
  summarise(b_u = mean(rating - mu - b_i))

# b_u plot
b_u %>%
  ggplot(aes(x = b_u)) +
  geom_histogram(bins = 20, fill = "lightpink", color = "purple") +
  ggtitle("User Bias Histogram") +
  xlab("Bias value") +
  ylab("Count")
```



There is substantial variability across users as well

```
# Predict all ratings with mu + b_i + b_u
y_hat_bu <- edx_test %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(y_hat_bu = mu + b_i + b_u) %>%
  .$y_hat_bu

# Calculate the RMSE with movie and user ranking effect
rmse_results <- rbind(rmse_results,
                      tibble(Model = "Mean + Movie bias + User bias",
                             RMSE = signif(rmse(edx_test$rating, y_hat_bu), 5)
                      )
)

# Print rmse_results tibble
rmse_results %>% kable()
```

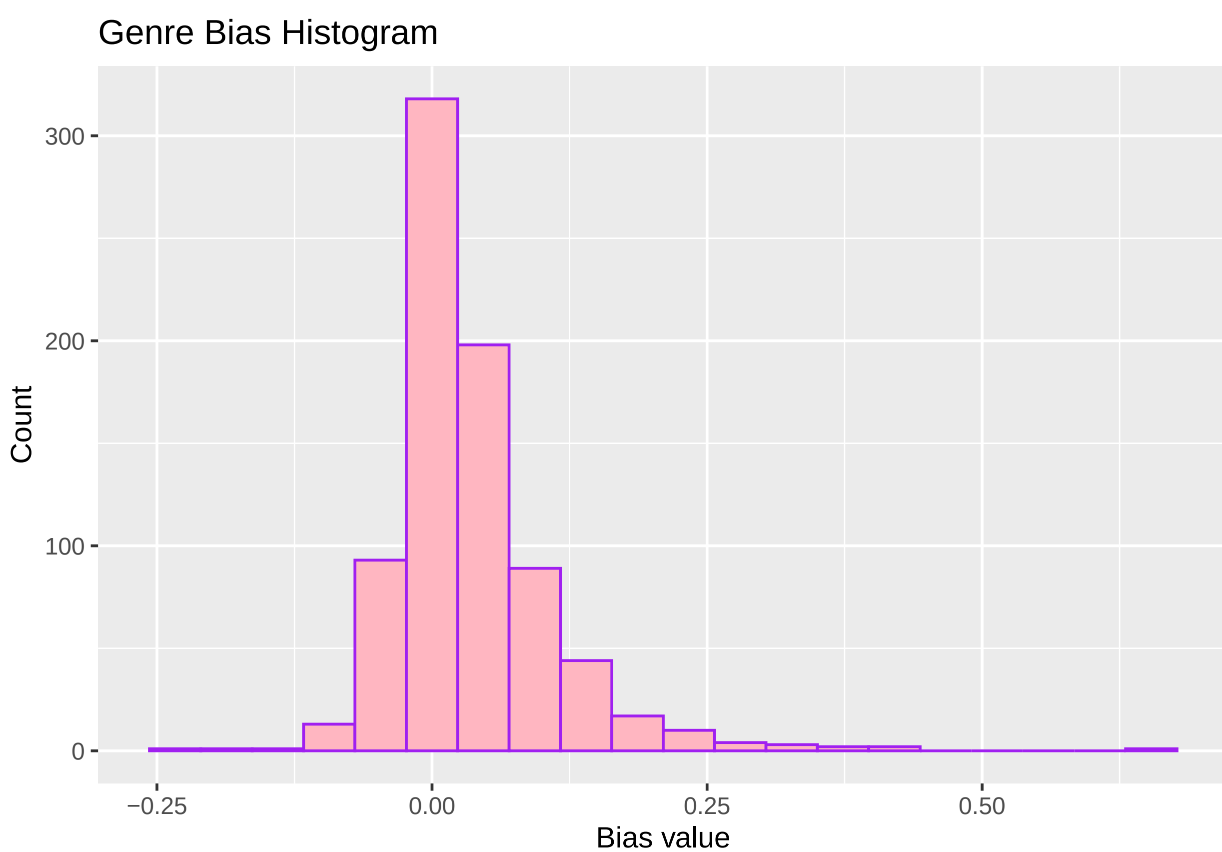
Model	RMSE
Target: less than	0.86490
Mean baseline	1.06010
Median baseline	1.16680
Mean + Movie bias	0.94296
Mean + Movie bias + User bias	0.86468

Notice further improvement of the RMSE

6. Movie, user and genre bias model y_hat = mu + e(u,i) + b_i + b_u + b_g

We can continue improving our model by adding genre bias b_g, represents the genre-specific effect. y_hat = mu + e(u,i) + b_i + b_u + b_g

```r
# Compute genre bias term, b_g
b_g <- edx_train %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  group_by(genres) %>%
  summarise(b_g = mean(rating - mu - b_i - b_u))

# b_g plot
b_g %>%
  ggplot(aes(x = b_g)) +
  geom_histogram(bins = 20, fill = "lightpink", color = "purple") +
  ggtitle("Genre Bias Histogram") +
  xlab("Bias value") +
  ylab("Count")
```



There are variability across genres as well

```r
# Predict all ratings with mu + b_i + b_u + b_g
y_hat_bg <- edx_test %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  mutate(y_hat_bg = mu + b_i + b_u + b_g) %>%
  .$y_hat_bg

# Calculate the RMSE with movie and user ranking effect
rmse_results <- rbind(rmse_results,
                      tibble(Model = "Mean + Movie bias + User bias + Genre bias",
                             RMSE = signif(rmse(edx_test$rating, y_hat_bg), 5)
                      )
)

# Print rmse_results tibble
rmse_results %>% kable()
```

Model	RMSE
Target: less than	0.86490
Mean baseline	1.06010
Median baseline	1.16680
Mean + Movie bias	0.94296
Mean + Movie bias + User bias	0.86468
Mean + Movie bias + User bias + Genre bias	0.86432

Notice more improvement of the RMSE

7. Regularization

RMSE is particularly sensitive to large errors. During system development, models learn not only the underlying patterns in the training data but also the noise, which leads to poor performance on unknown data. This is called over_fitting. Regularization is a technique used in machine learning to prevent over_fitting, enhance generalization, and improve the robustness of the model. Regularization constrains the total variability of the effect sizes by penalizing large estimates that come from small sample sizes. b's is now regularized as b_i_reg = sum(rating - mu)/(n_i + lambda).
Here:
n_i is number of ratings made for movie i
lambda is a penalty term. The larger lambda, the more we shrink.

We can also use regularization to estimate the user effect, genre effect. For more detail, please see **here**

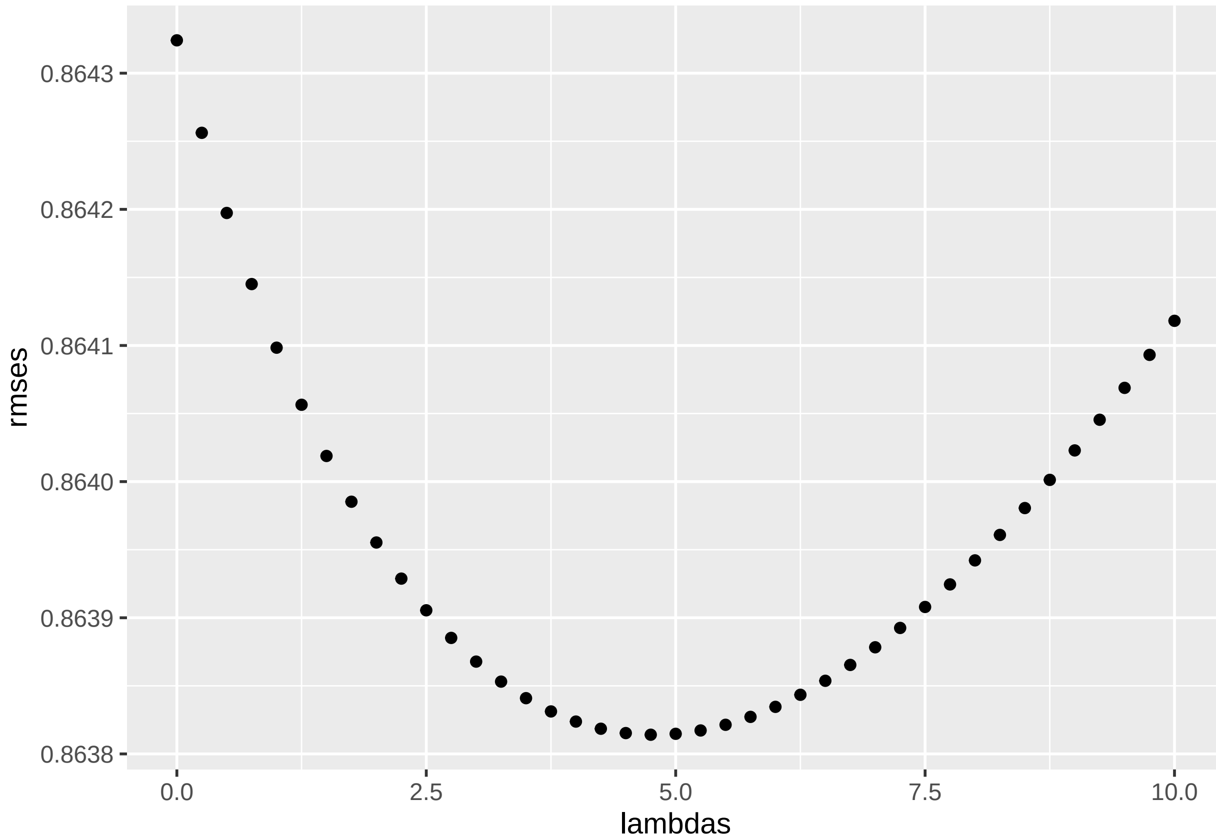
```r
# Regularization function
regularization <- function(lambda, train, test){

  # Calculate average rating across training data, mu
  mu <- mean(train$rating)

  # Movie bias regularization term
  b_i_reg <- train %>%
    group_by(movieId) %>%
```

```r
    summarise(b_i_reg = sum(rating - mu) / (n() + lambda))

  # User bias regularization term
  b_u_reg <- train %>%
    left_join(b_i_reg, by = "movieId") %>%
    filter(!is.na(b_i_reg)) %>%
    group_by(userId) %>%
    summarise(b_u_reg = sum(rating - mu - b_i_reg) / (n() + lambda))

  # Genre bias regularization term
  b_g_reg <- train %>%
    left_join(b_i_reg, by = "movieId") %>%
    left_join(b_u_reg, by = "userId") %>%
    filter(!is.na(b_i_reg), !is.na(b_u_reg)) %>%
    group_by(genres) %>%
    summarise(b_g_reg = sum(rating - mu - b_i_reg - b_u_reg) / (n() + lambda))

  # Predict all ratings using regularization terms
  y_hat <- test %>%
    left_join(b_i_reg, by = "movieId") %>%
    left_join(b_u_reg, by = "userId") %>%
    left_join(b_g_reg, by = "genres") %>%
    filter(!is.na(b_i_reg), !is.na(b_u_reg), !is.na(b_g_reg)) %>%
    mutate(y_hat = mu + b_i_reg + b_u_reg + b_g_reg) %>%
    .$y_hat

  # Calculate and return rmses
  return(rmse(test$rating, y_hat))
}
```

```r
# Cross validation to find best lambda
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, regularization, edx_train, edx_test)

# Find best lambda
lambda <- lambdas[which.min(rmses)]
lambda
```

```
## [1] 4.75
```

```r
# lambdas vs. rsmes plot
qplot(x = lambdas, y = rmses)
```




```r
# Define rmse with the best lambda
rmse <- min(rmses)

# Add rmse to rmse_results tibble
rmse_results <- rbind(rmse_results,
                      tibble(Model = "Mean + Regularization of Movie bias, User bias and Genre bias",
                             RMSE = signif(rmse, 5)
                             )
                      )

# Print rmse_results tibble
rmse_results %>% kable()
```

Model	RMSE
Target: less than	0.86490
Mean baseline	1.06010
Median baseline	1.16680
Mean + Movie bias	0.94296
Mean + Movie bias + User bias	0.86468
Mean + Movie bias + User bias + Genre bias	0.86432
Mean + Regularization of Movie bias, User bias and Genre bias	0.86381

Notably improvement of RMSE

D. Regularization linear final model validation using final_holdout_test

```r
# Calculate RMSE of final regularization linear model
RMSE <- regularization(lambda, edx_train, final_holdout_test)

rmse_results <- rbind(rmse_results,
                      tibble(Model = "Final model validation with final_holdout_test",
                             RMSE = signif(RMSE, 5)
                             )
                     )

# Print rmse_results tibble
rmse_results %>% kable()
```

Model	RMSE
Target: less than	0.86490
Mean baseline	1.06010
Median baseline	1.16680
Mean + Movie bias	0.94296
Mean + Movie bias + User bias	0.86468
Mean + Movie bias + User bias + Genre bias	0.86432
Mean + Regularization of Movie bias, User bias and Genre bias	0.86381
Final model validation with final_holdout_test	0.86485

The evaluation metrics indicate that the system performs well in terms of accuracy and relevance.

E. Conclusion

The system effectively generates recommendations that enhance user satisfaction through:
1. The combination of user preferences and movie attributes
2. Assess the accuracy and effectiveness of the recommendations using various metrics

Since our data set is quite large, if we use linear algorithms built in R, the program will take a very long time to run and may crash our computer due to the large calculations. The linear models built above with their simplicity can help predict movie ratings without severely affecting the computer resources.

Future work:
1. Integrating additional features, such as movie age, user feedback could benefit to our system
2. Apply advanced machine learning techniques, such as matrix factorization, could lead to even more precise and diverse recommendations

F. References:

https://rafalab.dfci.harvard.edu/dsbook/large-datasets.html#recommendation-systems
https://rafalab.dfci.harvard.edu/dsbook/
https://www.datacamp.com/tutorial/category/r-programming
https://translate.google.com/?sl=auto&tl=en&op=translate to translate some of my text from my native language