HarvardX: PH125.9x MovieLens Project

Sohit Tandon

4/25/2020

Introduction/Overview/Executive Summary

The objective of this project is to create a movie recommendation system using the MovieLens dataset. The purpose of the recommendation system is to predict the movie rating based on other users' ratings. We will use the 10M version of the MovieLens dataset for this project. This project is motivated by the *Netflix challenge*, in which Netflix offered a challenge to the data science community in October 2006. Netflix uses a recommendation system to predict how many stars a user will give a specific movie on a scale of one to five. The challenge was to improve the recommendation algorithm by 10% for a million dollars reward.

The MovieLens 10M dataset has 10 million ratings and 1000,000 tag applications applied to 10,000 movies by 72,000 users. This data needs to be downloaded and the we need to use provided code to generate the dataset for the project.

```
################################
# Create edx set, validation set
################################

# Note: this process could take a couple of minutes

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

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

dl <- tempfile()
 download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
 colnames(movies) <- c("movieId", "title", "genres")
 movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                           title = as.character(title),
                                           genres = as.character(genres))

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

# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding")
# if using R 3.5 or earlier, use `set.seed(1)` instead
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
 edx <- movielens[-test_index,]
 temp <- movielens[test_index,]
```

```r
# Make sure userId and movieId in validation set are also in edx set
validation <- temp %>%
      semi_join(edx, by = "movieId") %>%
      semi_join(edx, by = "userId")

# Add rows removed from validation set back into edx set
removed <- anti_join(temp, validation)
 edx <- rbind(edx, removed)

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

The generated dataset is divided in two parts

- **edx** dataset which is the training set
- **validation** dataset which is the test set

The recommendation system has to be built using the edx data set. The developed system has to be tested against the validation data set. RMSE(Root Mean Square Error) will be used to evaluate the predictions.

Methods/Analysis

After downloading the data and dividing it into training (edx) and test (validation) data set, it was explored.

```r
glimpse(edx)
```

```
## Observations: 9,000,055
## Variables: 6
## $ userId    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...
## $ movieId   <dbl> 122, 185, 292, 316, 329, 355, 356, 362, 364, 370, 37...
## $ rating    <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5...
## $ timestamp <int> 838985046, 838983525, 838983421, 838983392, 83898339...
## $ title     <chr> "Boomerang (1992)", "Net, The (1995)", "Outbreak (19...
## $ genres    <chr> "Comedy|Romance", "Action|Crime|Thriller", "Action|D...
```

```r
glimpse(validation)
```

```
## Observations: 999,999
## Variables: 6
## $ userId    <int> 1, 1, 1, 2, 2, 2, 3, 3, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5...
## $ movieId   <dbl> 231, 480, 586, 151, 858, 1544, 590, 4995, 34, 432, 4...
## $ rating    <dbl> 5.0, 5.0, 5.0, 3.0, 2.0, 3.0, 3.5, 4.5, 5.0, 3.0, 3....
## $ timestamp <int> 838983392, 838983653, 838984068, 868246450, 86824564...
## $ title     <chr> "Dumb & Dumber (1994)", "Jurassic Park (1993)", "Hom...
## $ genres    <chr> "Comedy", "Action|Adventure|Sci-Fi|Thriller", "Child...
```

The data types of the edx and validation data set are

Column Name	Data Type
userId	integer
movieId	double class (numeric)
rating	double class (numeric)
timestamp	integer
title	character
genres	character

Now lets check for existence of any NAs in the dataset

```r
## Checking for NAs in edx dataset
sapply(edx, {function(x) any(is.na(x))}) %>% knitr::kable()
```

	x
userId	FALSE
movieId	FALSE
rating	FALSE
timestamp	FALSE
title	FALSE
genres	FALSE

```r
## Checking for NAs in validation dataset
sapply(validation, {function(x) any(is.na(x))}) %>% knitr::kable()
```

	x
userId	FALSE
movieId	FALSE
rating	FALSE
timestamp	FALSE
title	FALSE
genres	FALSE

Data wrangling

1. Converting the timestamp column to a human readable value. Two new columns are added to the dataset namely **year_rated** which denotes the year the movie was rated and **year_released** which denotes the year the movie was released.

```r
########################################
########## Data Wrangling ##############
########################################

## Converting timestamp column to a human readable value
## need to install lubridate package as need the function as_datetime
if(!require(lubridate)) install.packages("lubridate", repos = "http://cran.us.r-project.org")

## using mutate function to add a new column year_rated to the data set edx and
##storing the result in a new data frame edx_ts
edx_ts <- mutate(edx, year_rated = year(as_datetime(timestamp)))

## using mutate function to add a new column year_release to the data set edx and
##storing the result in the frame edx_ts
edx_ts <- mutate(edx_ts, year_release = as.numeric(str_sub(title,-5,-2)))
```

- Checking the new dataset **edx__ts**

```r
head (edx_ts)
```

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 3      1     292      5 838983421               Outbreak (1995)
## 4      1     316      5 838983392               Stargate (1994)
## 5      1     329      5 838983392 Star Trek: Generations (1994)
## 6      1     355      5 838984474       Flintstones, The (1994)
##                           genres year_rated year_release
## 1                 Comedy|Romance       1996         1992
## 2          Action|Crime|Thriller       1996         1995
## 3  Action|Drama|Sci-Fi|Thriller       1996         1995
## 4        Action|Adventure|Sci-Fi       1996         1994
## 5 Action|Adventure|Drama|Sci-Fi       1996         1994
## 6        Children|Comedy|Fantasy       1996         1994
```

2. Similarly two new columns **year__rated** and **year__released** are added to validation dataset as well to maintain consistency.

```r
## using mutate function to add a new column year_rated to the data set validate and
##storing the result in a new data frame validation_ts
validation_ts <- mutate(validation, year_rated = year(as_datetime(timestamp)))

## using mutate function to add a new column year_release to the data set validate and
##storing the result in the data frame validation_ts
validation_ts <- mutate(validation_ts, year_release = as.numeric(str_sub(title,-5,-2)))
```

- Checking the new dataset **validation__ts**

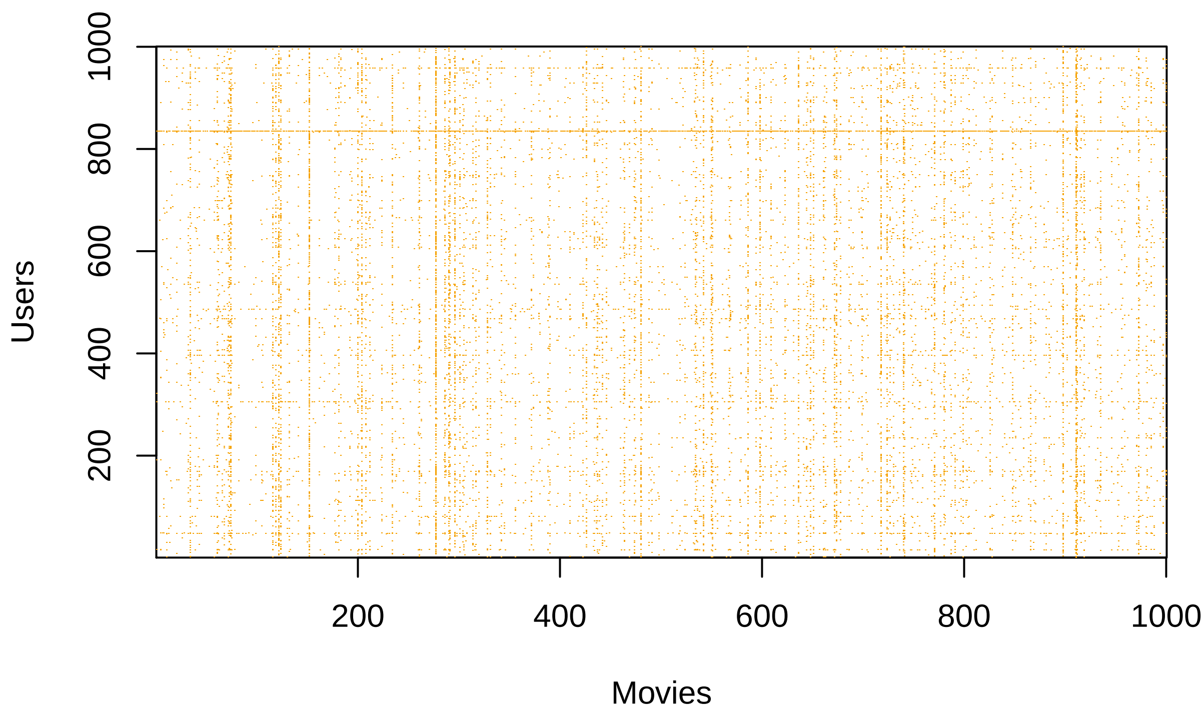
```r
head (validation_ts)
```

```
##   userId movieId rating timestamp
## 1      1     231      5 838983392
## 2      1     480      5 838983653
## 3      1     586      5 838984068
## 4      2     151      3 868246450
## 5      2     858      2 868245645
## 6      2    1544      3 868245920
##                                                  title
## 1                                   Dumb & Dumber (1994)
## 2                                   Jurassic Park (1993)
## 3                                      Home Alone (1990)
## 4                                         Rob Roy (1995)
## 5                                  Godfather, The (1972)
## 6 Lost World: Jurassic Park, The (Jurassic Park 2) (1997)
##                                   genres year_rated year_release
## 1                                 Comedy       1996         1994
## 2        Action|Adventure|Sci-Fi|Thriller       1996         1993
## 3                        Children|Comedy       1996         1990
## 4                Action|Drama|Romance|War       1997         1995
## 5                            Crime|Drama       1997         1972
## 6 Action|Adventure|Horror|Sci-Fi|Thriller       1997         1997
```

Visualizations

Let's visualize the matrix to understand how dense or sparse it is

```
###### Visualizing the matrix to see how  dense or sparse it is.
##Matrix of 1000 users and 1000 movies with yellow indicating a user/movie
##combination with rating
users <- sample(unique(edx$userId), 1000)

edx %>% filter(userId %in% users) %>%
  select(userId, movieId, rating) %>%
  mutate(rating = 1) %>%
  spread(movieId, rating) %>% select(sample(ncol(.), 1000)) %>%
  as.matrix() %>% t(.) %>%
  image(1:1000, 1:1000,. , xlab="Movies", ylab="Users")
```



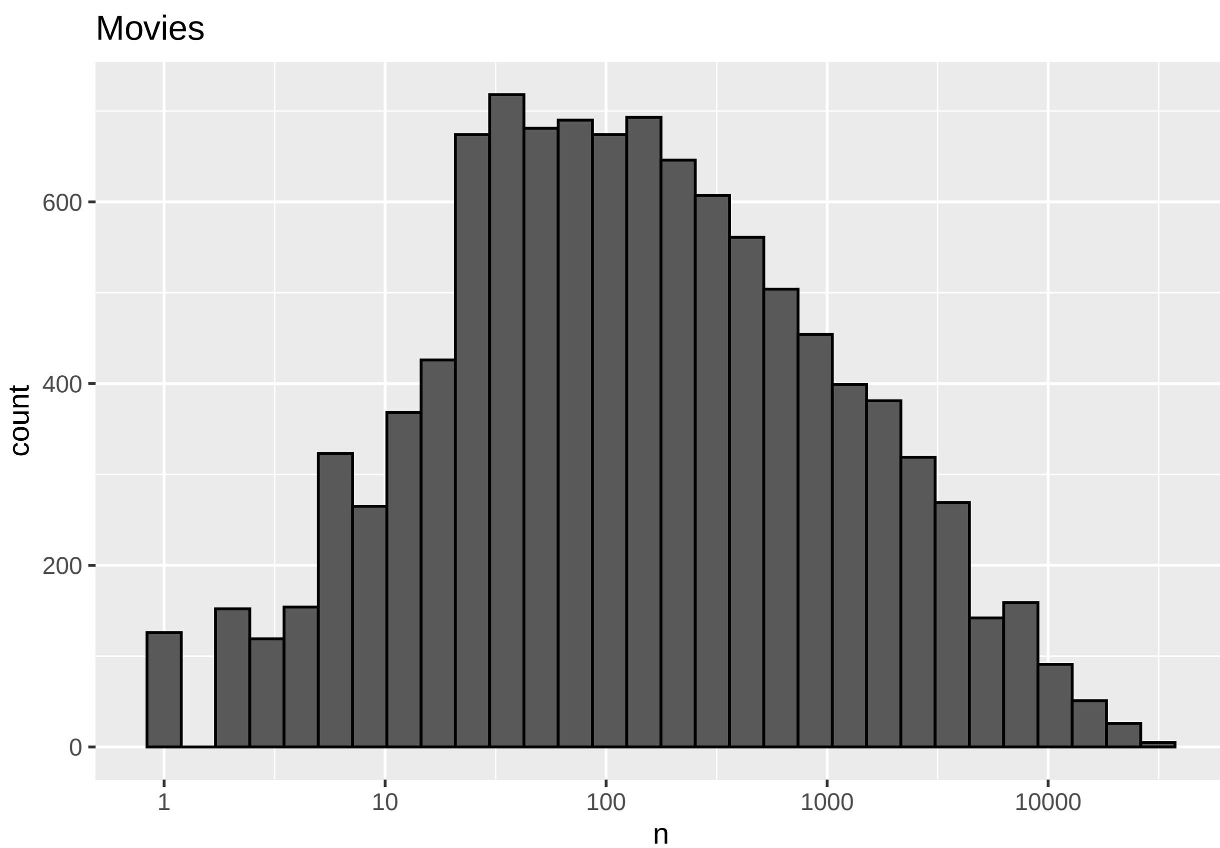
The matrix generated for 1000 users depicts how sparse the ratings are. The yellow spots are the user/movie combination for which we have a rating. The empty white spaces are the once were no ratings exits.

Let's look at some of the general properties of the data to determine various predictors for movie ratings.

Number of ratings for each movie

Visualizing the histogram for number of ratings for each movie.

```
#### Visualizing number of ratings for each movie ##############################

edx %>% count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Movies")
```

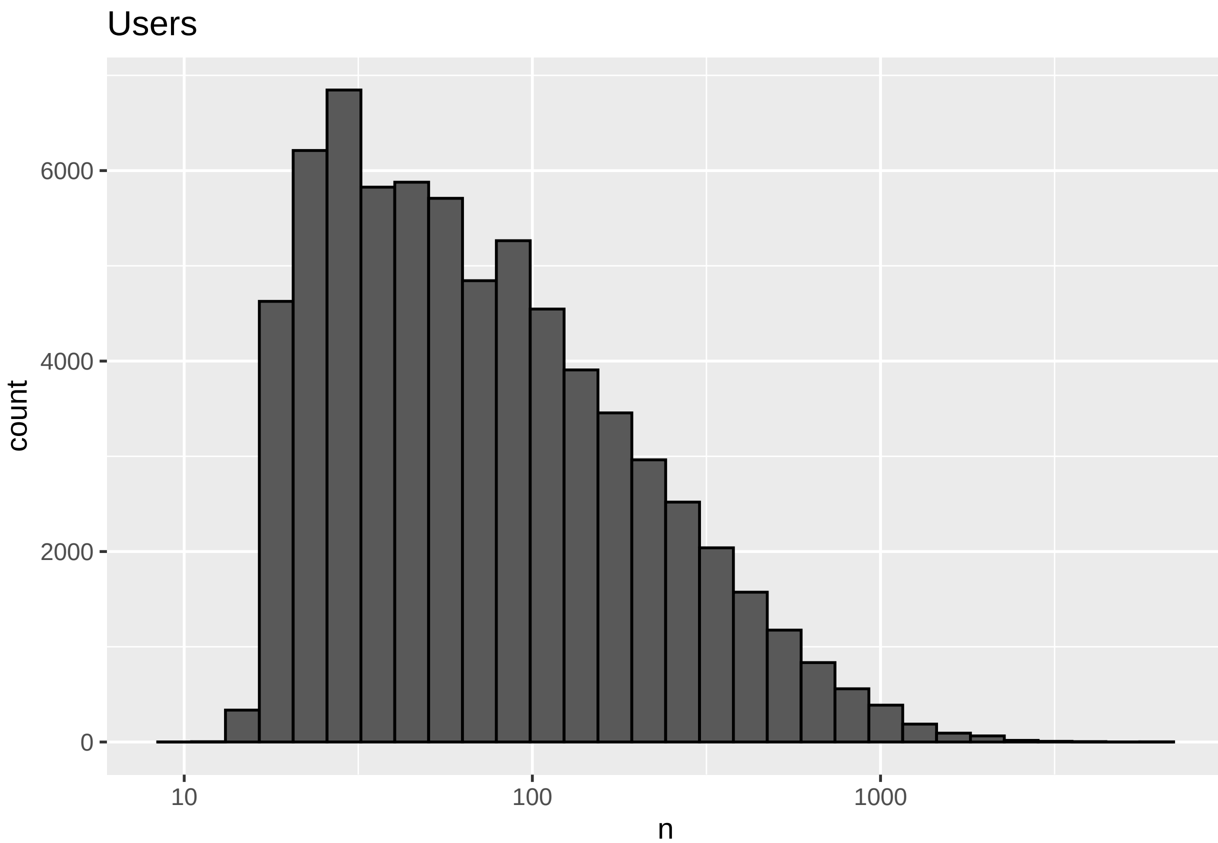


The above rating distribution by movieId depicts that some movies get rated more than others. This is expected as some are blockbusters watched by millions while some non popular movies are watched by a few only.

Number of ratings by userId

Let's generate the distribution of users and the number of ratings they give.

```
#### Visualizing number of ratings for each user ##############################

edx %>% count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Users")
```

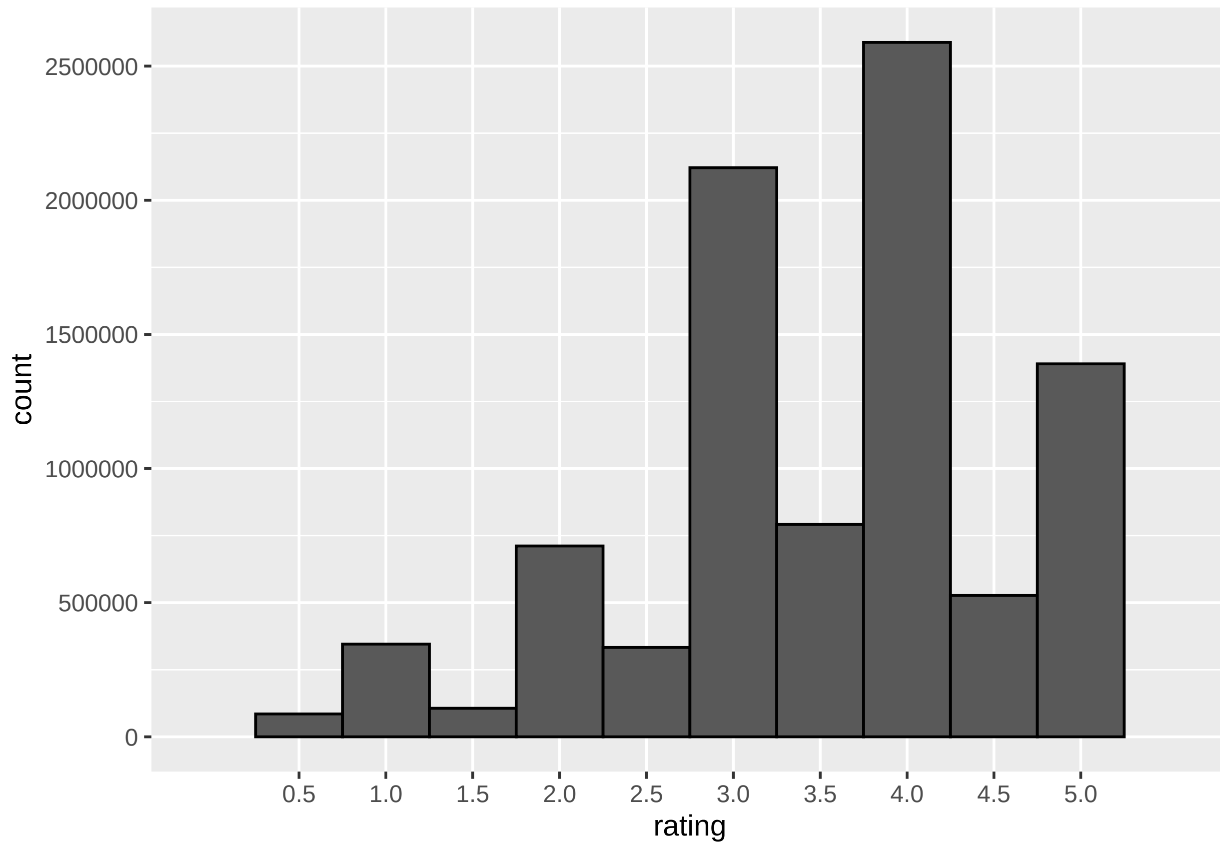


The above user distribution depicts that some users are more active than others at rating movies.

Count of each rating.

Lets look at the histogram of counts of each rating.

```
#### Visualizing number of count for each rating ##############################
edx %>% ggplot(aes(rating)) + geom_histogram(binwidth = 0.5, color = "black") +
  scale_x_discrete(limits = c(seq(0.5,5,0.5))) +
  scale_y_continuous(breaks = c(seq(0, 3000000, 500000)))
```

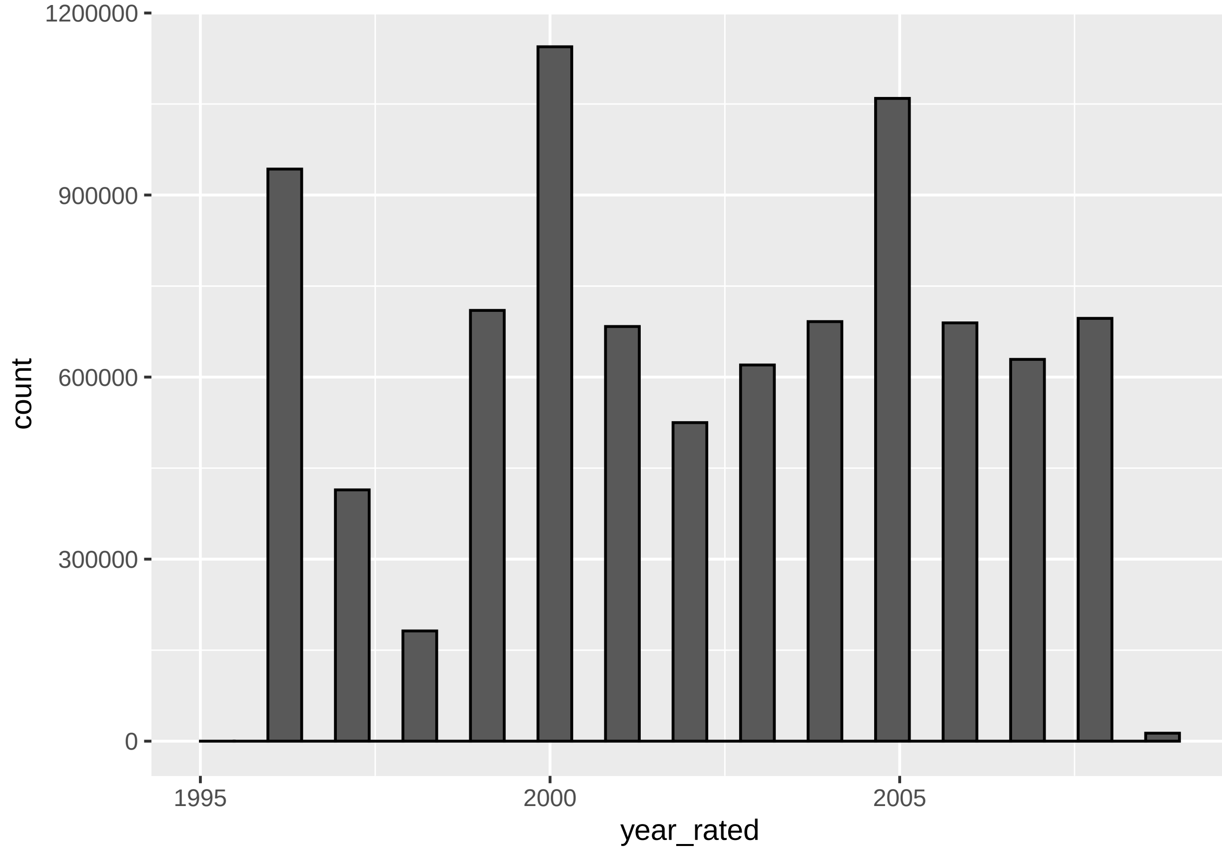


We see that some ratings are more popular with users than others. From the plot above we see that 4 is the most popular rating followed by 3 and 5 respectively.

Distribution of ratings by year of rating.

Now let's see look at the histogram for year of rating.

```
#### Visualizing year of rating ###############################################
edx_ts %>% ggplot(aes(year_rated))+geom_histogram(bins = 30, color = "black")
```

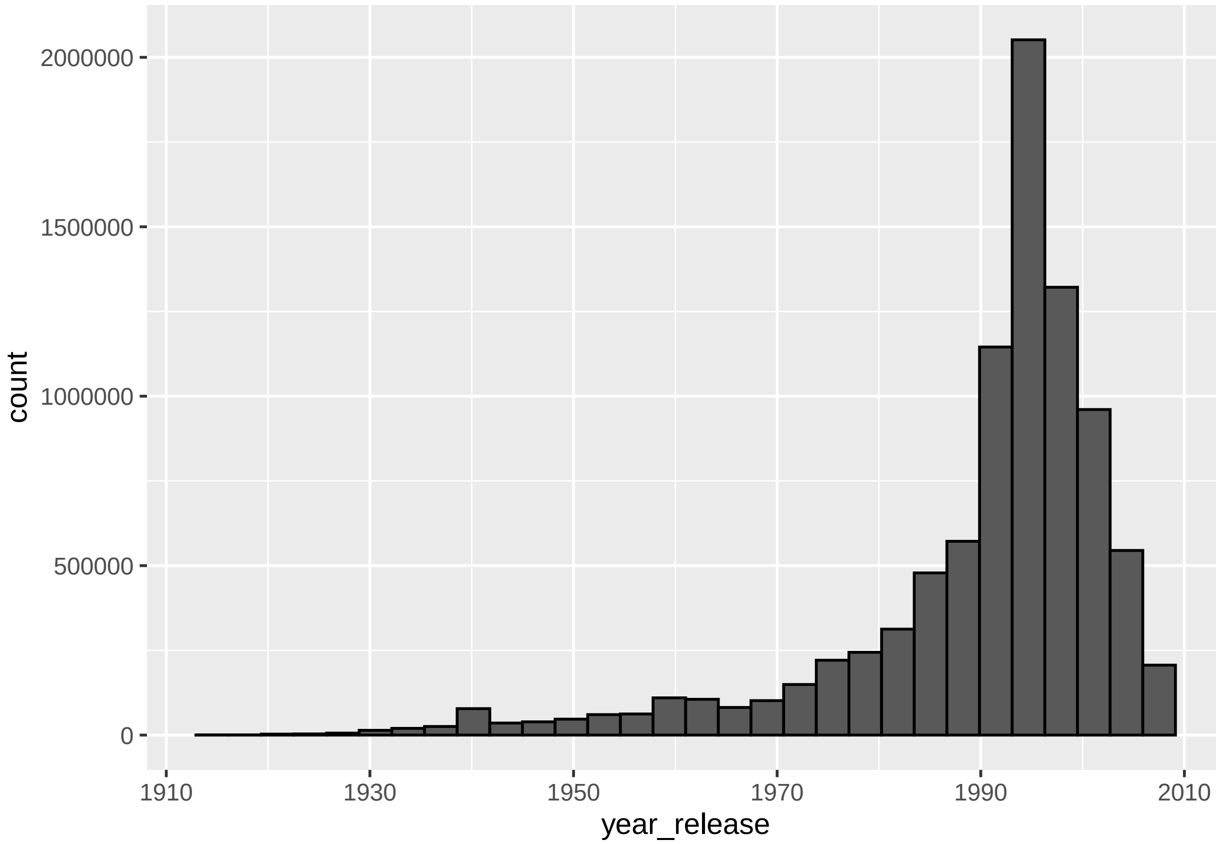


The distribution of year_rated seems to be irregular. Some years have more ratings than others.

Distribution of ratings by year of release

Visualizing the year of release histogram.

```
#### Visualizing year of release ################################################
edx_ts %>% ggplot(aes(year_release))+geom_histogram(bins = 30, color = "black")
```



This chart depicts that movies released between 1990 and 2006 were rated more than those released earlier or later.

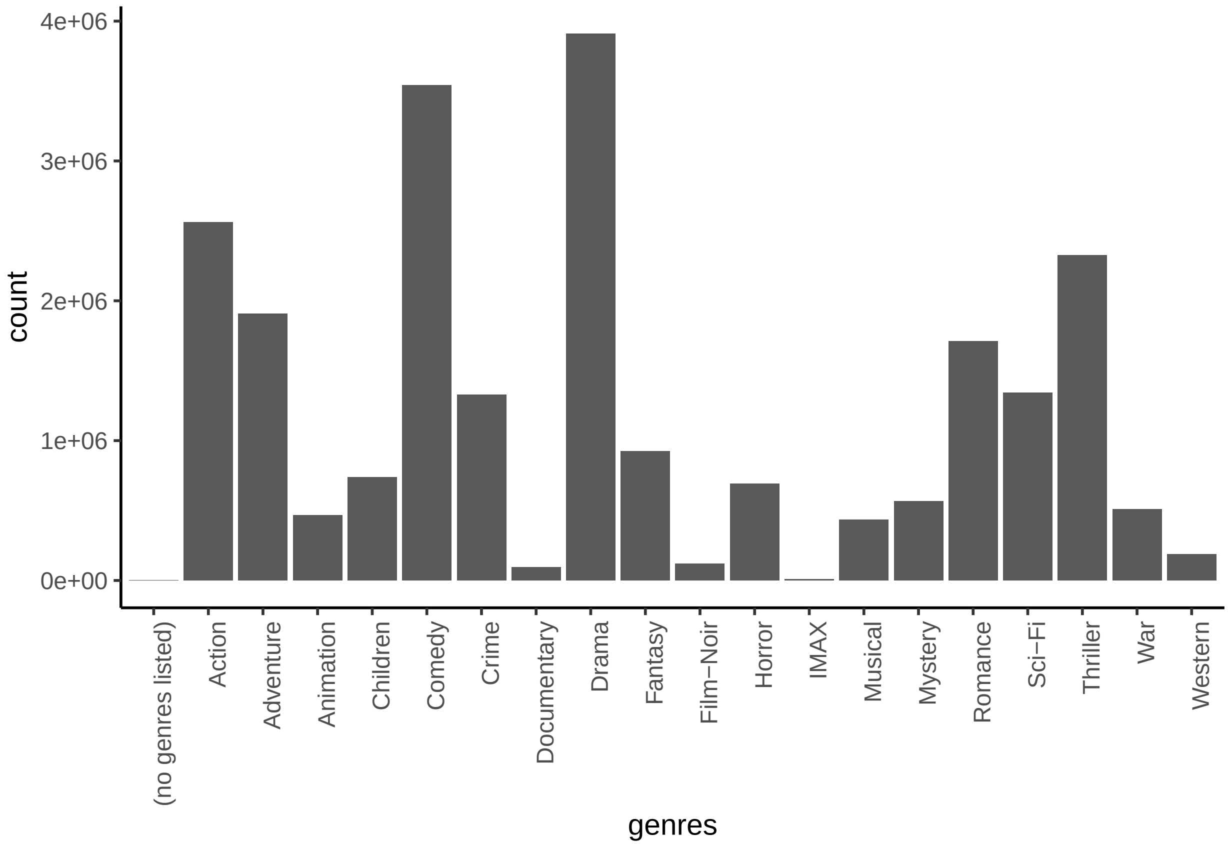
Top rated movie genres Let's look at the top rated genres

```
edx_ts %>% separate_rows(genres, sep = "\\|") %>%
group_by(genres) %>%
summarize(count = n()) %>%
arrange(desc(count)) %>%
knitr::kable()
```

genres	count
Drama	3910127
Comedy	3540930
Action	2560545
Thriller	2325899
Adventure	1908892
Romance	1712100
Sci-Fi	1341183
Crime	1327715
Fantasy	925637
Children	737994
Horror	691485
Mystery	568332
War	511147
Animation	467168
Musical	433080
Western	189394
Film-Noir	118541
Documentary	93066
IMAX	8181
(no genres listed)	7

Visualizing the ratings by different genres

```
edx_ts %>% separate_rows(genres, sep = "\\|") %>% group_by(genres) %>%
  summarize(count = n()) %>%  ggplot(aes(x = genres, y = count)) +
  theme_classic()+
  geom_col()+
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```

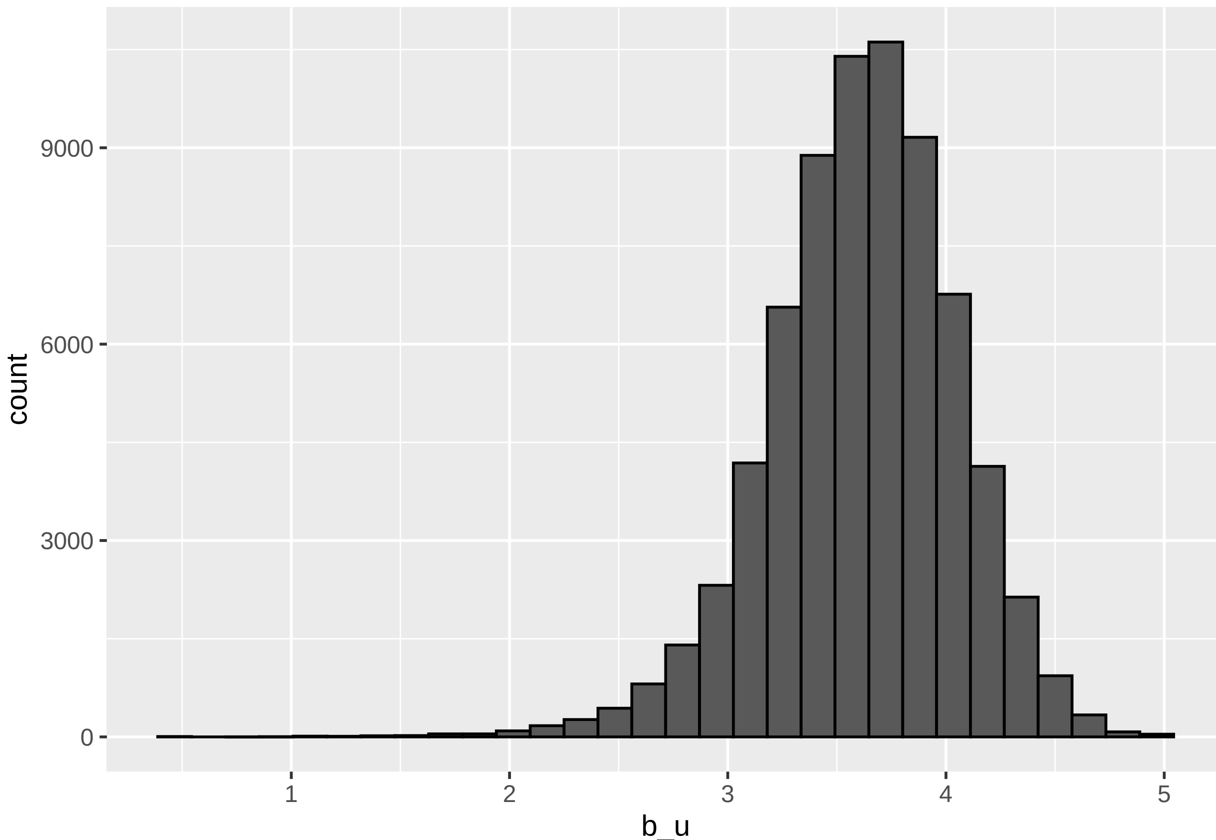


The most rated genre is Drama followed by Comedy and Action.

Average ratings of users who have rated over 100 movies

Visualizing the average rating of users who have rated over 100 movies.

```
#######Visualizing the average rating for user u who have rated over 100 movies ########
edx %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating)) %>%
  filter(n()>=100) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = "black")
```



Loss function

The Netflix challenge used the residual mean squared error (RMSE) on a test set to decide the winner. We define $y_{u,i}$ as the rating for movie i by user u and denote our prediction with $\hat{y}_{u,i}$.

$$RMSE = \sqrt{\frac{1}{N}\sum_{u,i}(\hat{y}_{u,i} - y_{u,i})^2}$$

where N is the number of users/movie combinations, and the sum incorporating the total combinations.

We will define the RMSE function that would compute the RMSE for ratings and the corresponding predictors

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

Recommendation Models

The Naive model

We build the first model in the recommendation system by predicting the same rating for all the movies regardless of the user. The model that assumes the same rating for all movies and users with all the differences explained by random variations would be

$$Y_{u,i} = \mu + \epsilon_{u,i}$$

where $Y_{u,i}$ is the prediction
$\epsilon_{u,i}$ is the independent errors
μ is the expected "true" rating for all movies.

Predicting the mean of all the ratings

```
########## First Model ###########################################
######## Step 1: Calculate mean rating of the training set ########
mu_hat <- mean(edx_ts$rating)
mu_hat
```

```
## [1] 3.512465
```

Calculating the RMSE for the naive model

```
########Step 2: Calculate the RMSE for Model1 on the validation data set####
naive_rmse <- RMSE(validation_ts$rating, mu_hat)
naive_rmse
```

```
## [1] 1.061202
```

We will create a data frame and store the result of the RMSE

```
########Creating a results table to store results of different models ####
rmse_results <- data_frame(Method ="Just the average", RMSE = naive_rmse)
rmse_results %>% knitr::kable()
```

Method	RMSE
Just the average	1.061202

The Movie Effect Model

We have seen during visualization that some movies are just generally rated higher than others. We will augment our first model by adding the term b_i to represent average ranking for movie i

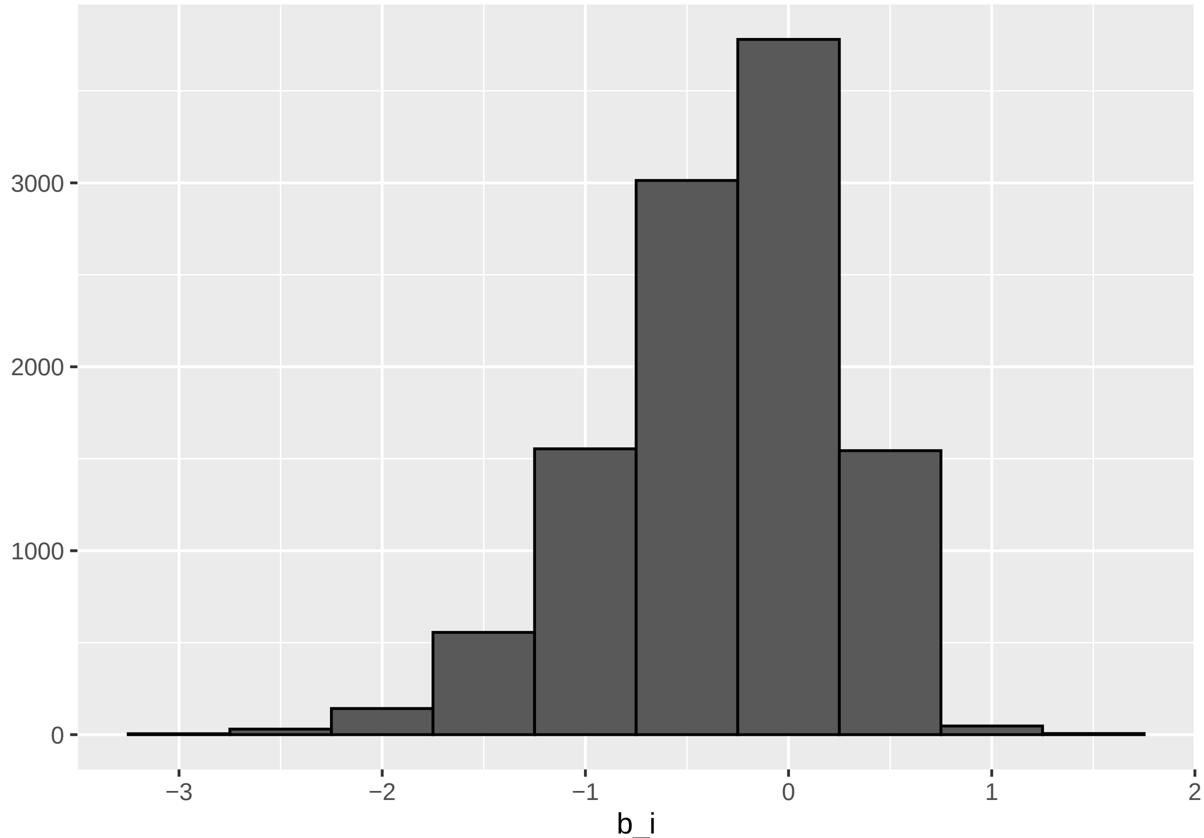
$$Y_{u,i} = \mu + b_i + \epsilon_{u,i}$$

Calculating the bias b_i for the movie effect model

```
### Second Model ########################################
######## Step1: Calculate 'bias' b which refers to some movies which
####### are generally rated higher than others ########
movie_avgs <- edx_ts %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu_hat))
```

Next we will plot the b_i to see the distribution of the estimates

```r
##############Step2: Plotting b_i to see the distribution of the estimates#############
movie_avgs %>% qplot(b_i, geom ="histogram", bins = 10, data = ., color = I("black"))
```



Next we will add b_i to our naive model to arrive at the new predictions.

```r
##############Step3: add the b_i to our Model1 predicted ratings to arrive
#############        at the new predicted#####
predicted_ratings <- mu_hat + validation_ts %>% left_join(movie_avgs,by='movieId') %>%
                     pull(b_i)
```

Calculating the RMSE for the movie effect model.

```r
######## Step4: Calculate the RMSE for Model2 on the validation data set###############
model2_rmse <- RMSE(predicted_ratings,validation_ts$rating)
model2_rmse
```

```
## [1] 0.9439087
```

Storing the results of the RMSE in the results data frame

```r
########## Store the result set of Model2 to rmse_results data frame #################
rmse_results <- bind_rows(rmse_results,data_frame(Method = "Movie Effect Model",
                                                  RMSE = model2_rmse))
rmse_results %>% knitr::kable()
```

Method	RMSE
Just the average	1.0612018
Movie Effect Model	0.9439087

The RSME is lower for the movie effect model as compared to the naive model.

The Movie and User Effect Model

We have noticed during visualization that some users are very cranky while others love every movie. Thus we will incorporate the user effect b_u in our earlier movie model. This improved model will be

$$Y_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$$

b_u is the bias for each user u.
Let's calculate the user bias b_u for this model

```
##############Third Model################################
#########Step1: Calculate the affect b_u to counter the cranky user who rates
#########        a great movie 3 rather than 5
user_avgs <- edx_ts %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu_hat - b_i))
```

Next we will calculate the modified predicted ratings for this model

```
###########Step2: calculating the new predicted ratings ##############################
predicted_ratings <- validation_ts %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu_hat + b_i + b_u) %>%
  pull(pred)
```

Calculating the RMSE for movie and user effects model

```
###############Step3: Calculate the RMSE for Model3 on the validation data set#######
model3_rmse <- RMSE(predicted_ratings,validation_ts$rating)
model3_rmse
```

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

Storing the results of the RMSE in the results data frame

```
######### Store the result set of Model3 to rmse_results data frame ################
rmse_results <- bind_rows(rmse_results,data_frame(Method = "Movie + User Effect Model",
                                                  RMSE = model3_rmse))
rmse_results %>%knitr::kable()
```

Method	RMSE
Just the average	1.0612018
Movie Effect Model	0.9439087
Movie + User Effect Model	0.8653488

The RSME is even lower for the movie + user effect model as compared to the movie model.

Regularization

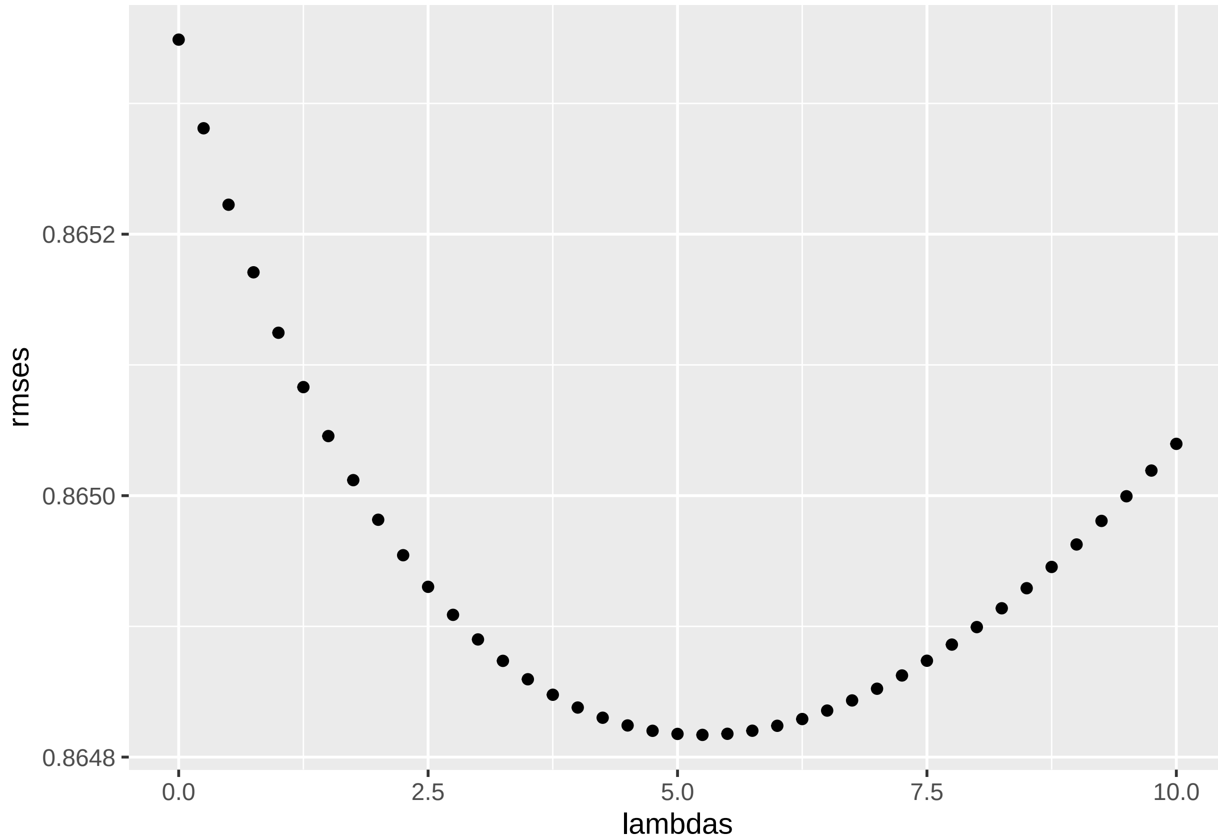
Regularization enables us to penalize large estimates that are formed using small sample sizes. This is caused when best and worst movies are rated by very few users say just one. These are noises which can be removed

by using regularization.
The regularization method uses a tuning parameter, λ, to minimize the RMSE. This will help to modify the movie bias b_i and user bias b_u for movies with few ratings.

```
##########Regularization#############################################
########## Here we use cross-validation to pick a lambda #################
lambdas <- seq(0, 10, 0.25)

rmses <- sapply(lambdas, function(l){
  mu_hat <- mean(edx_ts$rating)

    b_i <- edx_ts %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu_hat)/(n()+l))

    b_u <- edx_ts %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu_hat)/(n()+l))

    predicted_ratings <-
    validation_ts %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu_hat + b_i + b_u) %>%
    pull(pred)
  return(RMSE(predicted_ratings, validation_ts$rating))
})
```

Plotting the RMSE against lambdas to find the optimal lambda value.

```
##########Plotting the RMSE against lambdas ####
qplot(lambdas, rmses)
```



Finding the optimal lambda.

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

```
## [1] 5.25
```

Storing the regularized RMSE in the results data frame.

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method="Regularized Movie + User Effect Model",
                                     RMSE = min(rmses)))
rmse_results %>%knitr::kable()
```

Method	RMSE
Just the average	1.0612018
Movie Effect Model	0.9439087
Movie + User Effect Model	0.8653488
Regularized Movie + User Effect Model	0.8648170

The regularized RMSE for movie and user effect is even lower than the earlier models.

Results

Here is a summary of the prediction models for movie ratings.

```
rmse_results %>%knitr::kable()
```

Method	RMSE
Just the average	1.0612018
Movie Effect Model	0.9439087
Movie + User Effect Model	0.8653488
Regularized Movie + User Effect Model	0.8648170

As depicted in the results, we have made progress in lowering the RMSE with each successive model. The most optimal is the regularized movie and user effects model with an RMSE of 0.8648170 . This value of RMSE is lower than the target of < 0.86490 for this project.

Conclusion

In this project we developed a machine learning model to predict the movie ratings on the Movielens 10M dataset.The lowest RMSE was achieved by using regularization with cross validation on the movie and user effects model. The model was developed on the training set and finally was cross validated on the test set at the end.