Report on Movielens Rating Predictions

Olivia Malkowski

10/18/2019

Contents

1. Introduction

1.1 Overview/Executive Summary

The following project is focused on the theme of recommendation systems, with the objective of predicting the ratings (out of five stars) that users will attribute to a particular movie by filtering information and accounting for factors such as genre, the user and their history, as well as the movie itself. The recommendation system will take into consideration the users' ratings to give specific suggestions for future movies. Those with the highest predicted ratings will then be recommended to the user.

The use of such predictive models is of practical relevance; for instance, in 2006, Netflix challenged the public to improve their recommendation algorithm by 10%, providing movie suggestions that were better tailored to the users' preferences and interests. Thus, these systems play a significant role in helping clients to find products and services that are right for them. The "Netflix Challenge" evaluated the quality of the proposed algorithms using the "typical error": the Root Mean Square Error (RMSE); as such, the same method will be used to determine the strength of a variety of prediction models using the 10M version of the MovieLens dataset, made available by the GroupLens Research Lab. The goal will be to train a machine learning algorithm using the inputs from one subset (named `edx`) of the database to predict movie ratings in the `validation` set.

1.2 Loading the Data

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")

### The validation 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
```

```
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 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)
```

1.3 Tidy Data

```
### Additional libraries to install:
library(lubridate)
library(stringr)
library(ggplot2)
```

```
### We can check that the data is in tidy format with the as_tibble function:
edx%>%as_tibble()
```

```
## # A tibble: 9,000,055 x 6
##    userId movieId rating timestamp title              genres
##     <int>   <dbl>  <dbl>     <int> <chr>              <chr>
##  1      1     122      5 838985046 Boomerang (1992)   Comedy|Romance
##  2      1     185      5 838983525 Net, The (1995)    Action|Crime|Thrill~
```

```
##  3      1     292      5 838983421 Outbreak (1995)     Action|Drama|Sci-Fi~
##  4      1     316      5 838983392 Stargate (1994)     Action|Adventure|Sc~
##  5      1     329      5 838983392 Star Trek: Generat~ Action|Adventure|Dr~
##  6      1     355      5 838984474 Flintstones, The (~ Children|Comedy|Fan~
##  7      1     356      5 838983653 Forrest Gump (1994) Comedy|Drama|Romanc~
##  8      1     362      5 838984885 Jungle Book, The (~ Adventure|Children|~
##  9      1     364      5 838983707 Lion King, The (19~ Adventure|Animation~
## 10      1     370      5 838984596 Naked Gun 33 1/3: ~ Action|Comedy
## # ... with 9,000,045 more rows
```

1.4 Data Summary

Before processing the data, it is important to familiarise ourselves with the `edx` dataset. The `edx` set contains 9000055 observations of 6 variables, whilst the `validation` set contains 999999 observations of 6 variables.

```
### We can determine the number of distinct users and movies with the following:
edx%>%summarize(n_users=n_distinct(userId),n_movies=n_distinct(movieId))
```

```
##   n_users n_movies
## 1   69878    10677
```

```
### We can look at the first six lines of the `edx` dataset with the "head" function:
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 (1994)
##                           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
```

```
### To compute summary statistics for the dataset, we will use:
summary(edx)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:9000055     Length:9000055
```

```
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

```
### We then check that there is no missing data:
any(is.na(edx))
```

```
## [1] FALSE
```

2. Methods and Analysis

2.1 Data Cleaning and Exploration

Before commencing the data visualisation process, it is useful to add new column variables to our `edx` and `validation` sets. Upon familiarisation with the dataset, we can observe that the date and time of each rating is in `timestamp` format. To display the data in a manner that is meaningful to a lay audience, it would be useful to convert these into a typical date format (e.g. yyyy/mm/dd); we will name this column `date`. In addition, to depict the evolution of movie ratings over time, it will be practical to transform the `date` column to a shorter format (e.g. year); we will name this column `year`.

Add column - converting `timestamp` to `date` and `year` format:

```
class(edx$timestamp)
```

```
## [1] "integer"
```

```
edx<-edx%>%mutate(date=as.Date(as.POSIXlt(timestamp,origin="1970-01-01",format="%Y-%m-%d"),format="%Y-%
edx<-edx%>%mutate(year=format(date,"%Y"))
validation<-validation%>%mutate(date=as.Date(as.POSIXlt(timestamp,origin="1970-01-01",format="%Y-%m-%d"
validation<-validation%>%mutate(year=format(date,"%Y"))
```

We can also observe that the `title` column of both sets includes both the movie title and the release year. In order to monitor rating differences between older and newly-released movies, as well as evolutions in genre over time in line with cultural changes, we can create a new column to separate the release year from the title using the `stringr` package. We will name this `release_year`.

The `str_sub` function enables us to extract a substring (in this case, the release year from the title column), with defined limits (i.e. "-5" and "-2" are chosen to prevent the inclusion of the brackets and punctuation on either side of the release year in the `title` column).

To add a column for `release_year`:

```
edx<-edx%>%mutate(release_year=as.numeric(str_sub(title,-5,-2)))
```

In order to build an accurate recommendation system, it is important to account for users' preferences for different genres. Upon contemplation of the `edx` and `validation` sets, the genre column includes all of the genres that the movie could be categorised into. To separate the rows by individual genres, we can split the dataset. We then add the columns `date`, `year`, and `release_year` to this new dataset.

Split dataset by genre

```
edx_genre<-edx%>%separate_rows(genres,sep = "\\|")
edx_genre<-edx_genre%>%mutate(date=as.Date(as.POSIXlt(timestamp,origin="1970-01-01",format="%Y-%m-%d"),i
edx_genre<-edx_genre%>%mutate(year=format(date,"%Y"))
edx_genre<-edx_genre%>%mutate(release_year=as.numeric(str_sub(title,-5,-2)))
validation_genre<-validation%>%separate_rows(genres,sep = "\\|")
validation_genre<-validation_genre%>%mutate(date=as.Date(as.POSIXlt(timestamp,origin="1970-01-01",forma
validation_genre<-validation_genre%>%mutate(year=format(date,"%Y"))
```

The `edx_genre` set has 23371423 observations whilst the `validation_genre` set contains 2595771 observations.

To compute the genres with the most ratings, we can use the following code:

```
ratings_genre <- edx_genre%>%
  group_by(genres) %>%
  summarize(count = n()) %>%
  arrange(desc(count))
ratings_genre
```

```
## # A tibble: 20 x 2
##    genres               count
##    <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
```

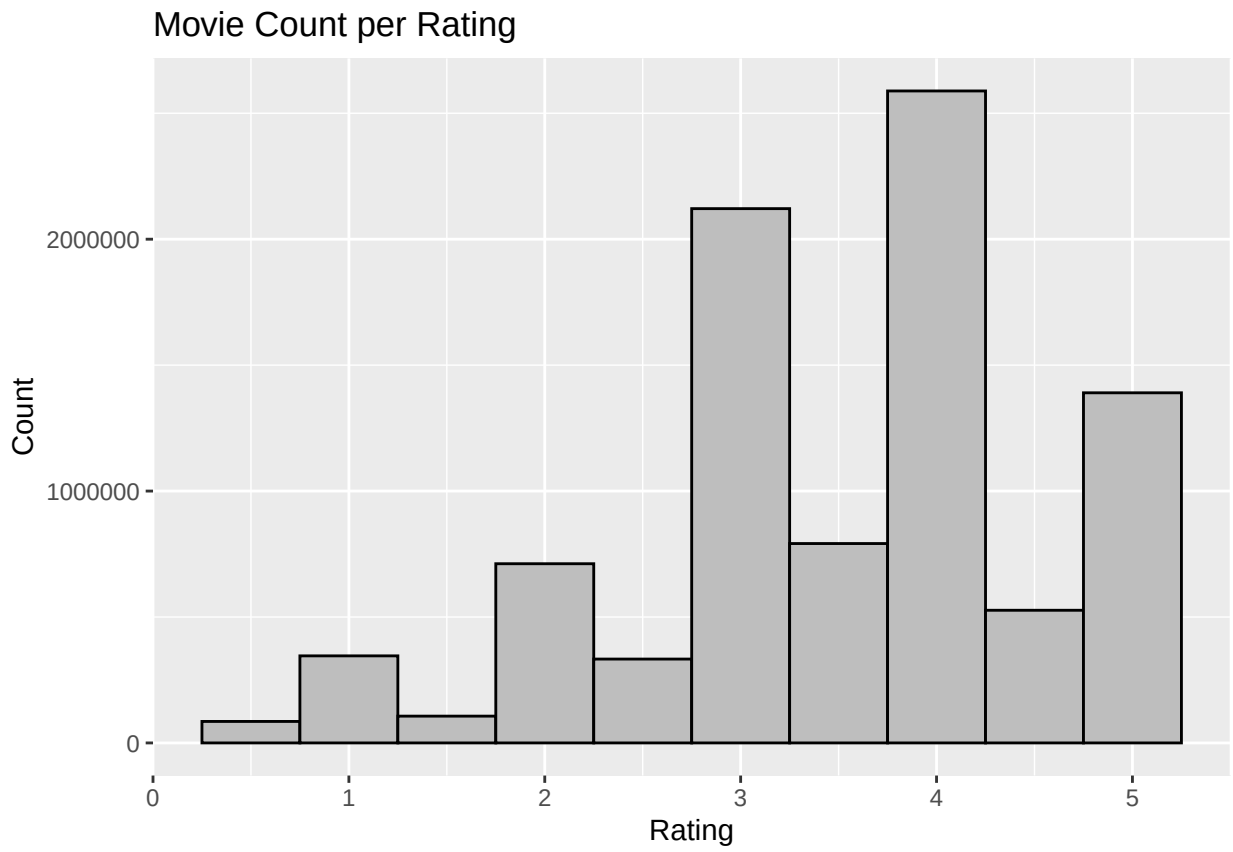
2.2 Data Visualisation

To gain a visual insight into the dataset's trends and patterns, we will use a range of data visualisation techniques.

Rating distributions

The first step in our analysis is to determine the rating distribution, that is, the most common ratings awarded to movies, as demonstrated in the histogram below.

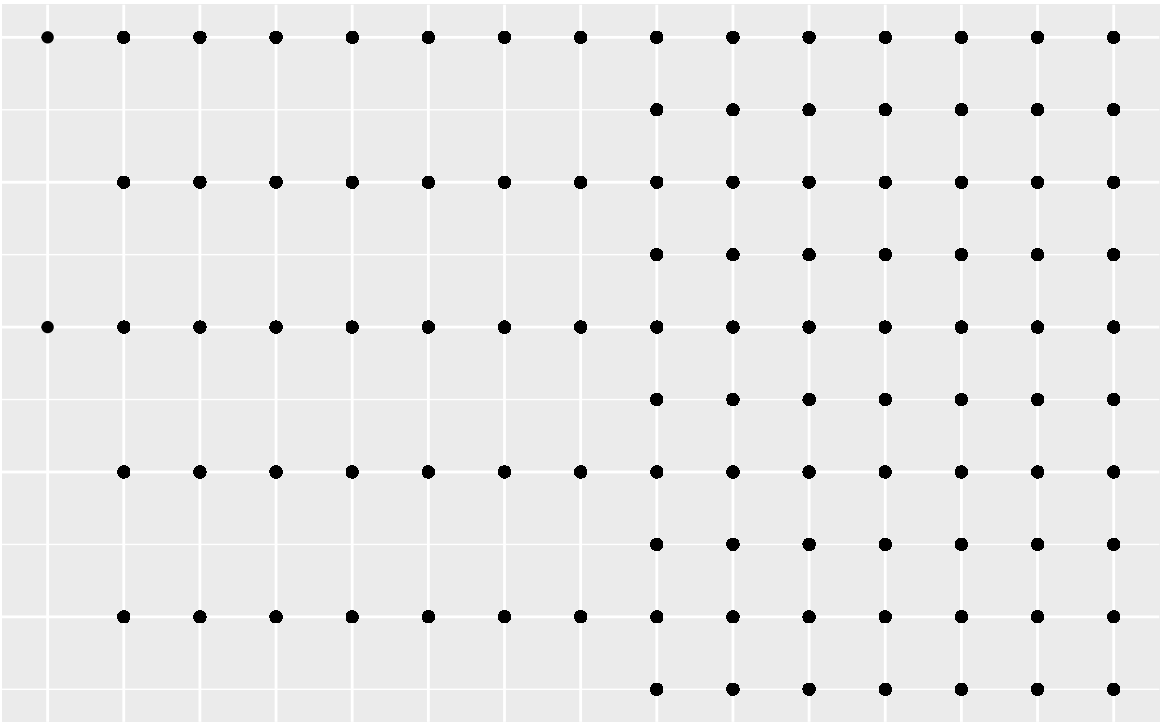
```
options(scipen=999)
edx%>%ggplot(aes(rating))+geom_histogram(bins=10,fill="grey",color="black")+ ggtitle("Movie Count per Ra
```



A main observation from studying this plot is that full-star ratings appear to be more common than half-star ratings. We use the `scipen` option to display the numerical units in fixed format.

Ratings awarded each year

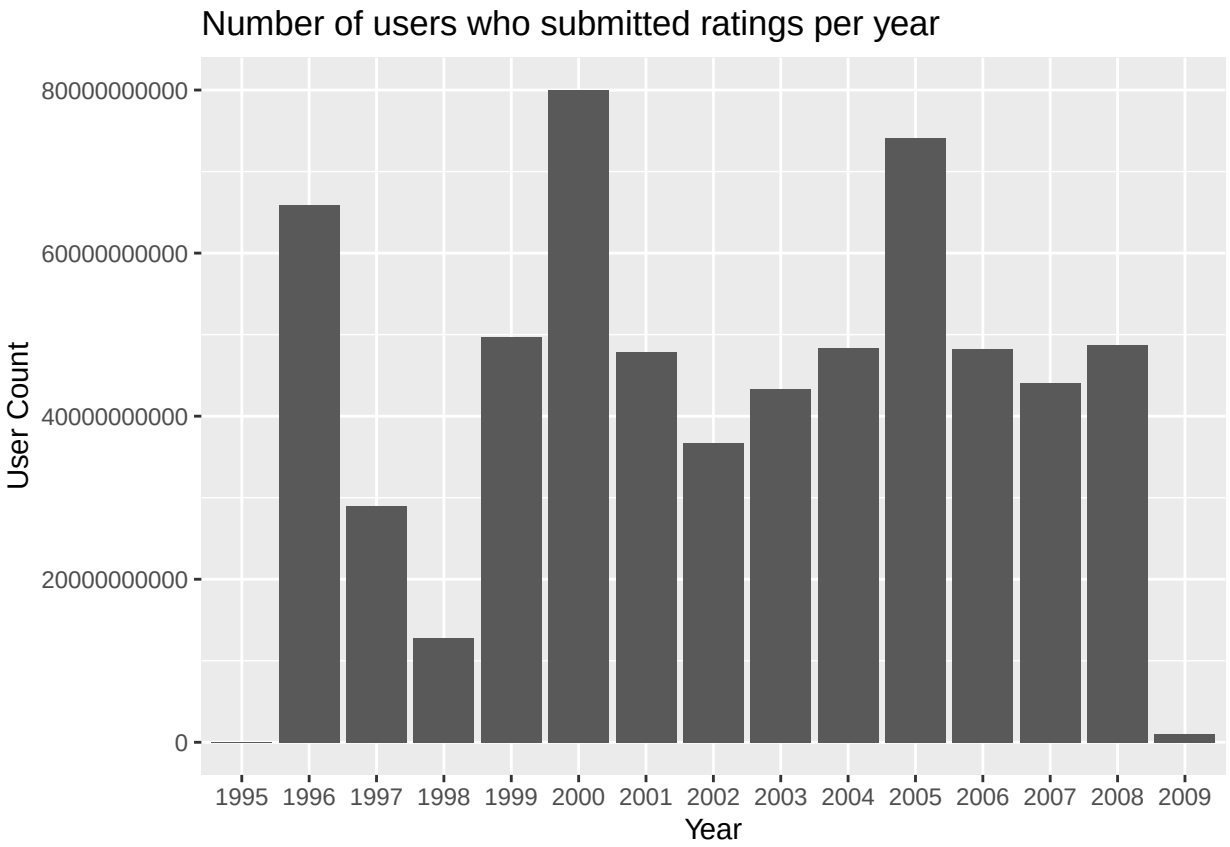
```
edx%>%ggplot(aes(year))+geom_point(aes(year,rating))+ ggtitle("Range of Ratings Awarded by Year")+xlab(
```



The above plot is informative as it demonstrates that half-star ratings were only awarded from 2003 onwards; this may explain the higher number of full-star relative to half-star ratings.

Number of users who submitted ratings for one or more movies per year

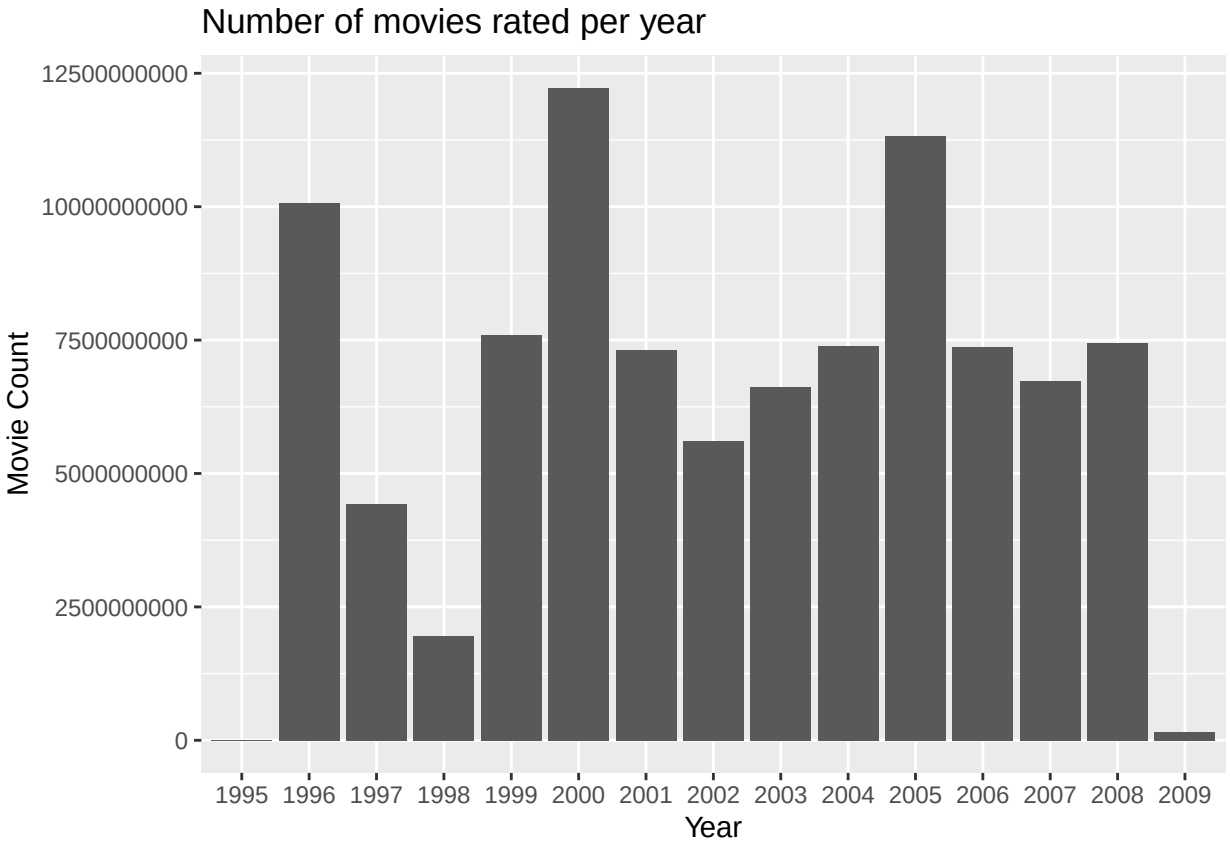
```
options(scipen=999)
edx%>%group_by(year)%>%ggplot(aes(year, n_distinct(userId)))+geom_bar(stat="identity")+ggtitle("Number
```



We can see that there is some variation in the number of active users per year.

Number of different movies rated per year

```
options(scipen=999)
edx%>%group_by(year)%>%ggplot(aes(year, n_distinct(movieId)))+geom_bar(stat="identity")+ggtitle("Number
```



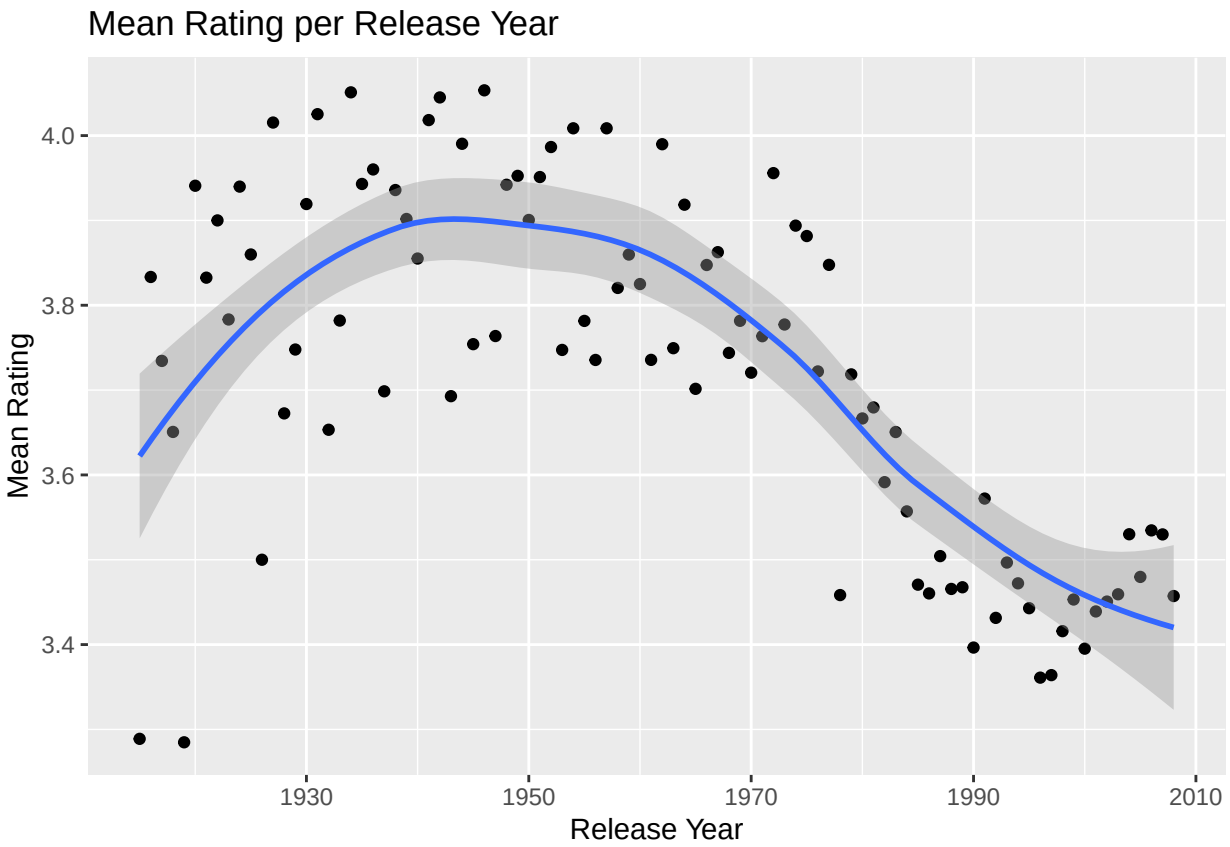
There is similar variation in the number of distinct movies rated each year. Note that years 1995 and 2009 were not full years, explaining the disparities seen in the figure.

Smooth plot of mean rating based on release year

We can create a plot of mean rating over release year (with 95% confidence intervals) to determine whether older or more recent movies tend to be rated more favourably.

```
edx %>% group_by(release_year)%>%summarize(rating=mean(rating)) %>%  ggplot(aes(release_year,rating))+g
```

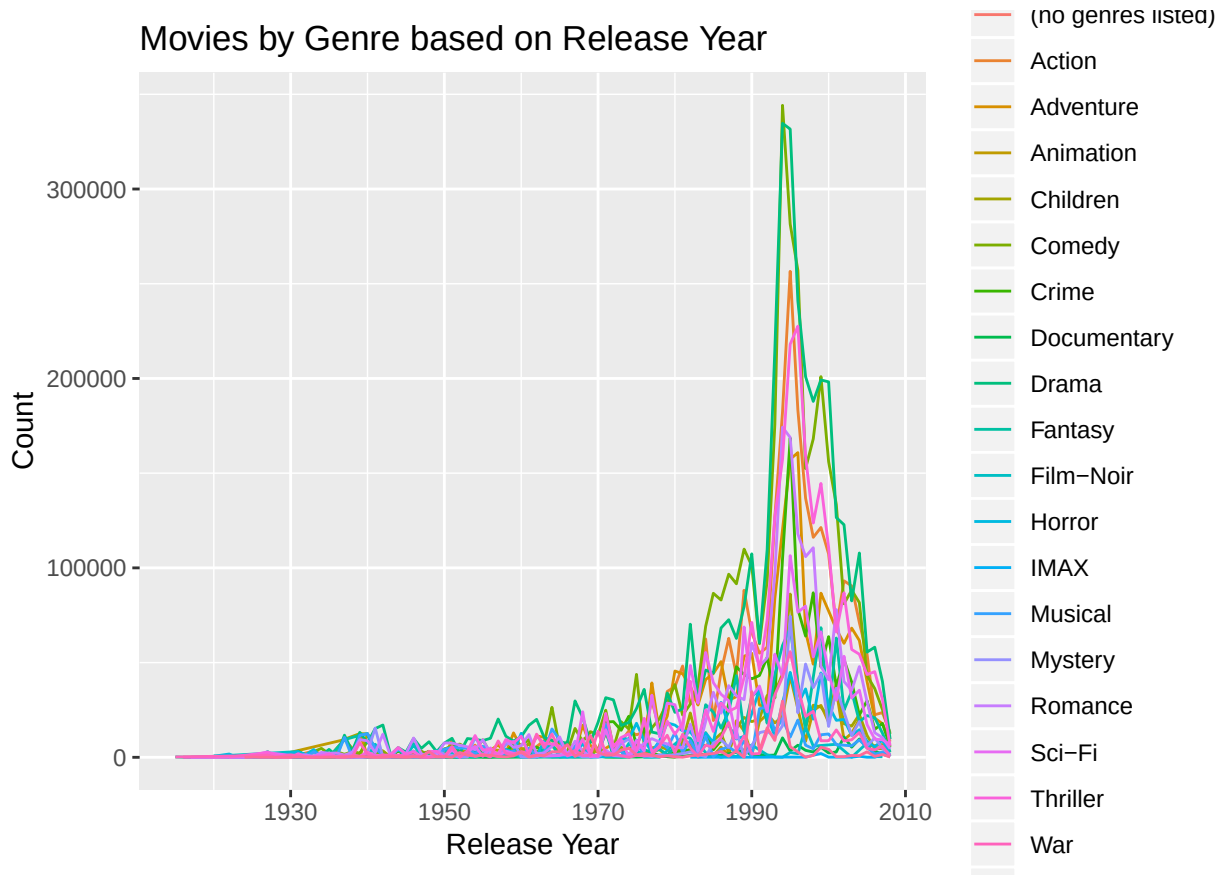
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Interestingly, users tend to rate modern movies lower than movies released in the 20th century.

Genres per release year

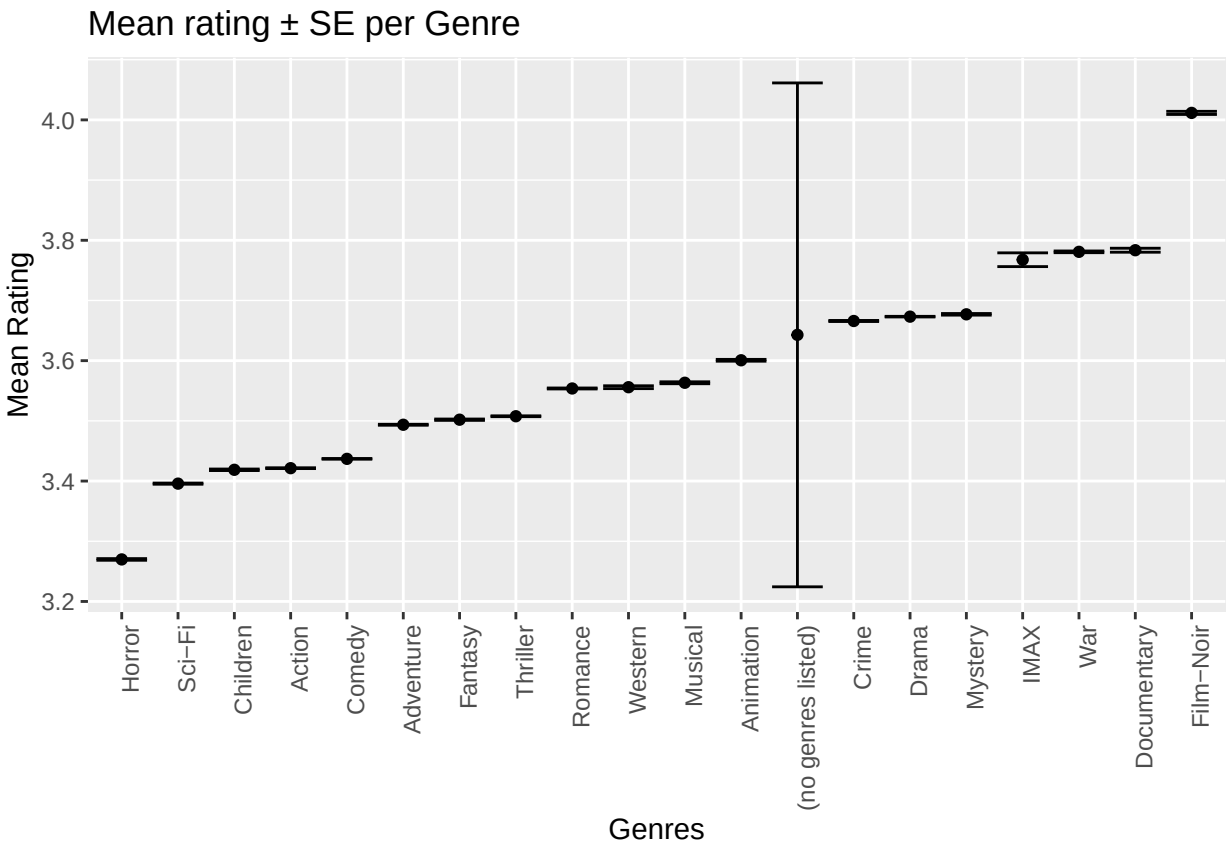
```
options(scipen=999)
edx_genre%>%mutate(genres=as.factor(genres))%>%group_by(release_year,genres)%>%summarize(n=n())%>%ggplo
```



The figure above provides an insight into how genres have evolved in line with cultural and social changes.

Mean ± SE rating per genre

```
edx_genre%>%group_by(genres)%>%summarize(n=n(),avg=mean(rating),se=sd(rating/sqrt(n())))%>%mutate(genres
```



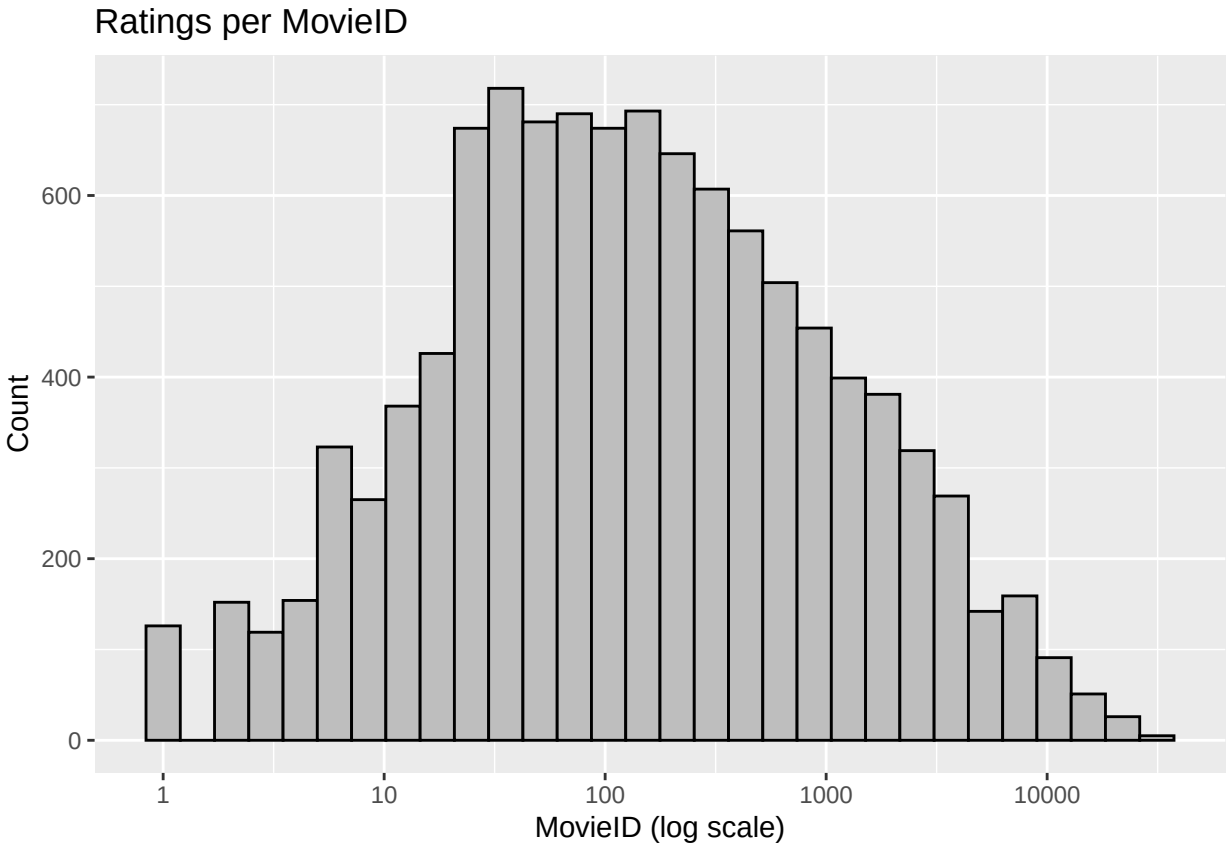
After separating the rows to display individual genres, we can also observe that certain genres have a higher mean rating than others. Note that the "no genres listed" refers to the movie "Pull My Daisy" released in 1958; this has a large standard error as only seven users rated the movie:

```
edx_genre%>%filter(genres=="(no genres listed)")
```

```
##   userId movieId rating  timestamp                title             genres
## 1   7701    8606    5.0 1190806786 Pull My Daisy (1958) (no genres listed)
## 2  10680    8606    4.5 1171170472 Pull My Daisy (1958) (no genres listed)
## 3  29097    8606    2.0 1089648625 Pull My Daisy (1958) (no genres listed)
## 4  46142    8606    3.5 1226518191 Pull My Daisy (1958) (no genres listed)
## 5  57696    8606    4.5 1230588636 Pull My Daisy (1958) (no genres listed)
## 6  64411    8606    3.5 1096732843 Pull My Daisy (1958) (no genres listed)
## 7  67385    8606    2.5 1188277325 Pull My Daisy (1958) (no genres listed)
##         date year release_year
## 1 2007-09-26 2007         1958
## 2 2007-02-11 2007         1958
## 3 2004-07-12 2004         1958
## 4 2008-11-12 2008         1958
## 5 2008-12-29 2008         1958
## 6 2004-10-02 2004         1958
## 7 2007-08-28 2007         1958
```

MovieID Count

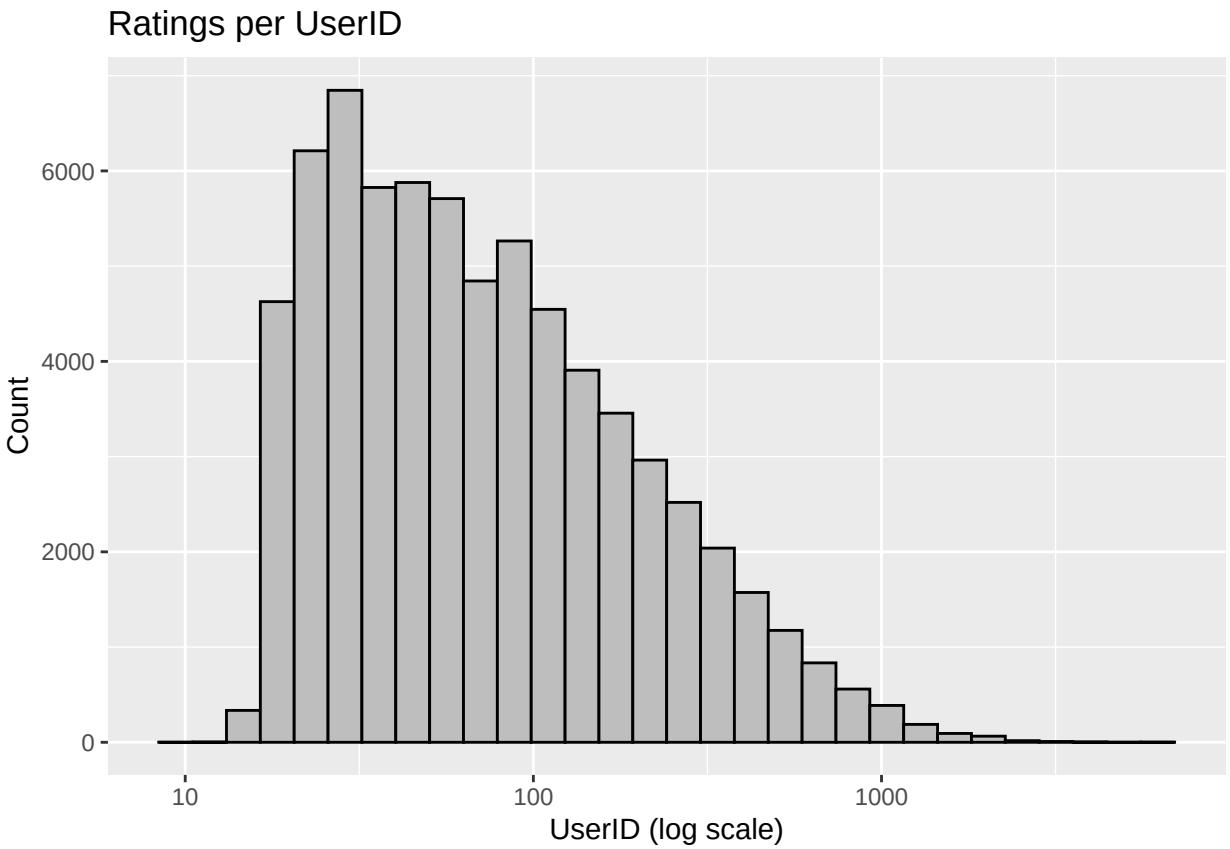
```
edx%>%count(movieId)%>%ggplot(aes(n))+geom_histogram(bins = 30, fill="grey", color="black") + scale_x_l
```



The above plot demonstrates that some movies have been rated more than others, as would be expected. It also provides initial insight into why the movies themselves may be an important factor when forming our prediction model.

UserID Count

```
edx%>%count(userId)%>%ggplot(aes(n))+geom_histogram(bins = 30, fill="grey", color="black") + scale_x_log
```



Similarly to the movie plot, we notice that there are sizeable contrasts in the number of ratings per user. Whereas some users rate movies very often, others may have only submitted their opinion for a fraction of the movies they watched.

2.3 Predictive Model - Methods

We will test a variety of predictive models and compute the results to choose the one that provides the lowest RMSE. The approach will be inspired by some of the methods used in the "Netflix Challenge". We will start with the simplest model, consisting of the average of all ratings (thus not taking into account movie or user effects). The second method will involve modeling movie effects, whilst the third will model movie and user effects. Finally, three regularization approaches will be tested using cross-validation to choose the penalty terms. The models with the lowest RMSE will be re-run with the `edx` and `validation` sets.

The function that computes the RMSE:

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

The above function will compute the RMSE for ratings vectors and their predictors.

3. Results and Discussion

3.1 Partitioning the edx dataset into train and test sets

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

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

```
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2, list = FALSE)
train_set <- edx[-test_index,] ### Create train set
temp <- edx[test_index,] ### Create test set

### Make sure `userId` and `movieId` in the `test_set` set are also in the `train_set`
test_set <- temp %>% semi_join(train_set,by="movieId") %>% semi_join(train_set,by="userId")
```

Separating the **edx** dataset into a **train_set** and a **test_set** is essential for training, tuning, and regularization processes, unless full cross-validation is used.

3.2 Model 1 - Naive Bayes

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

```
## [1] 3.512482
```

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

```
## [1] 1.059904
```

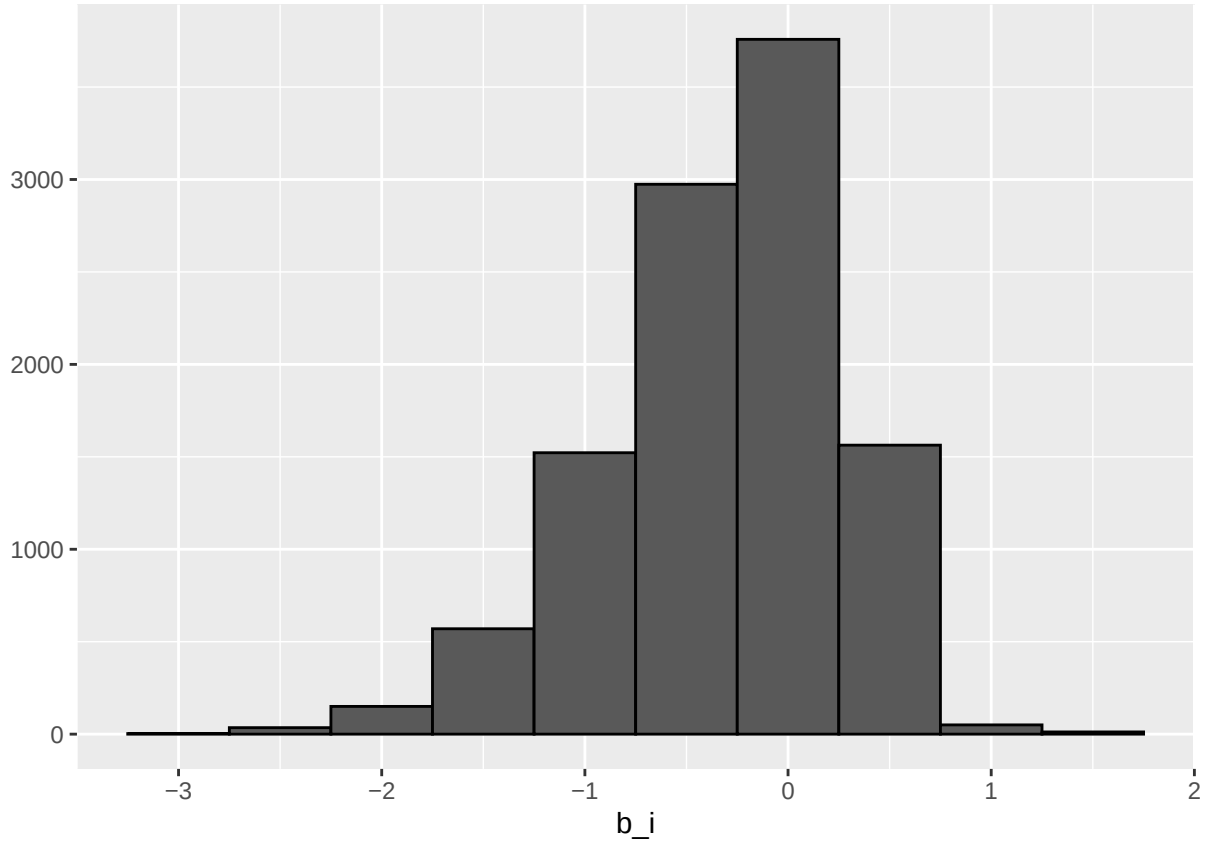
```
rmse_results <- tibble(method = "Just the average", RMSE = naive_rmse)
rmse_results
```

```
## # A tibble: 1 x 2
##   method            RMSE
##   <chr>            <dbl>
## 1 Just the average  1.06
```

The above model assumes the same ratings, regardless of the movie, user, or genre. Our RMSE is approximately 1.06, we can do better!

3.3 Model 2 - Movie Effect

```
library(caret)
library(dslabs)
library(tidyverse)
library(stringr)
mu <- mean(train_set$rating)
movie_avgs <- train_set %>% group_by(movieId) %>% summarize(b_i = mean(rating - mu))
movie_avgs %>% qplot(b_i, geom ="histogram", bins = 10, data = ., color = I("black"))
```



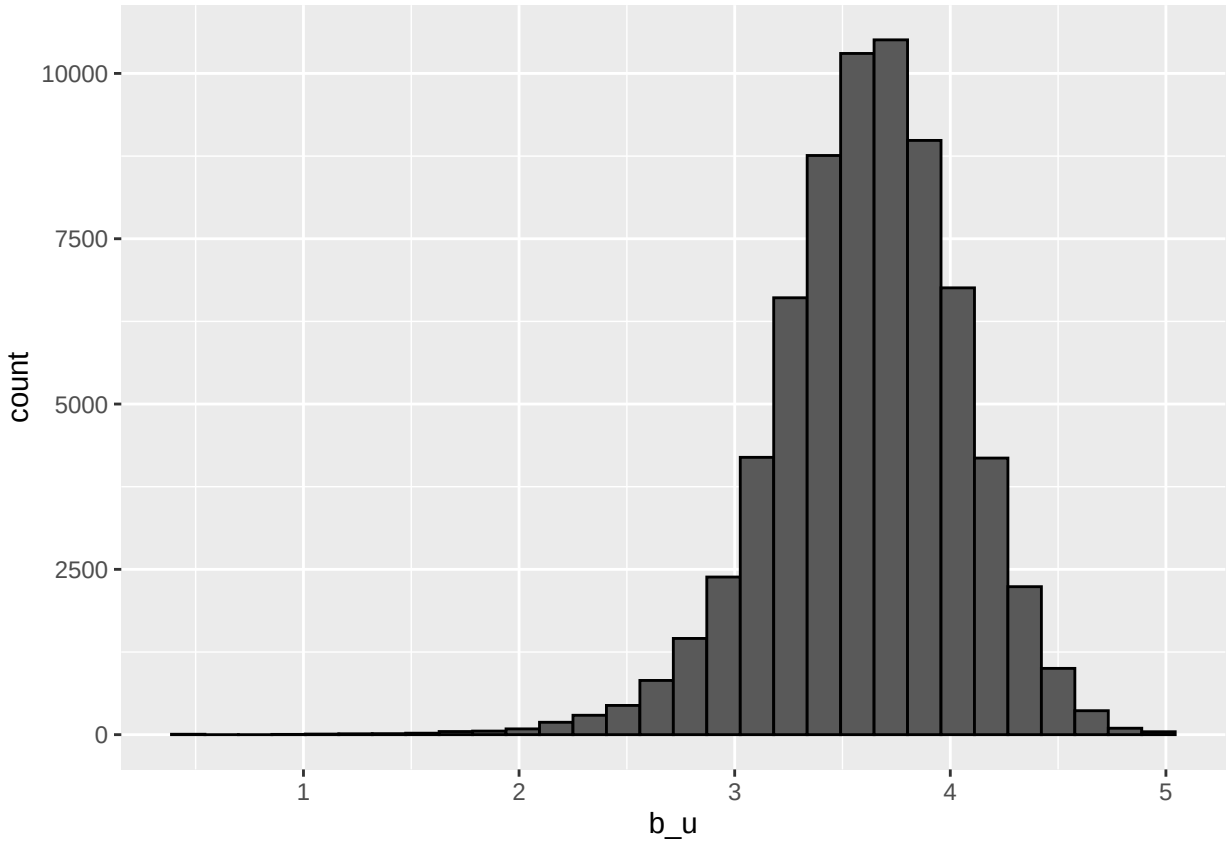

```
predicted_ratings <- mu + test_set %>% left_join(movie_avgs, by='movieId') %>% pull(b_i)
model_1_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,tibble(method="Movie Effect Model", RMSE = model_1_rmse))
rmse_results
```

```
## # A tibble: 2 x 2
##   method              RMSE
##   <chr>              <dbl>
## 1 Just the average   1.06
## 2 Movie Effect Model 0.944
```

We saw previously that movies are rated differently. To account for this observation, we can adapt our least squares estimate by grouping the datasets by the `movieId`. There is an improvement in the RMSE to approximately 0.94.

3.4 Model 3 - Movie and User Effect

```
train_set %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating)) %>% filter(n()>=100) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = "black")
```




```
user_avgs <- train_set %>% left_join(movie_avgs, by='movieId') %>% group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
predicted_ratings <- test_set %>% left_join(movie_avgs, by='movieId') %>% left_join(user_avgs, by='user
model_2_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results, tibble(method="Movie + User Effects Model", RMSE = model_2_rmse
rmse_results
```

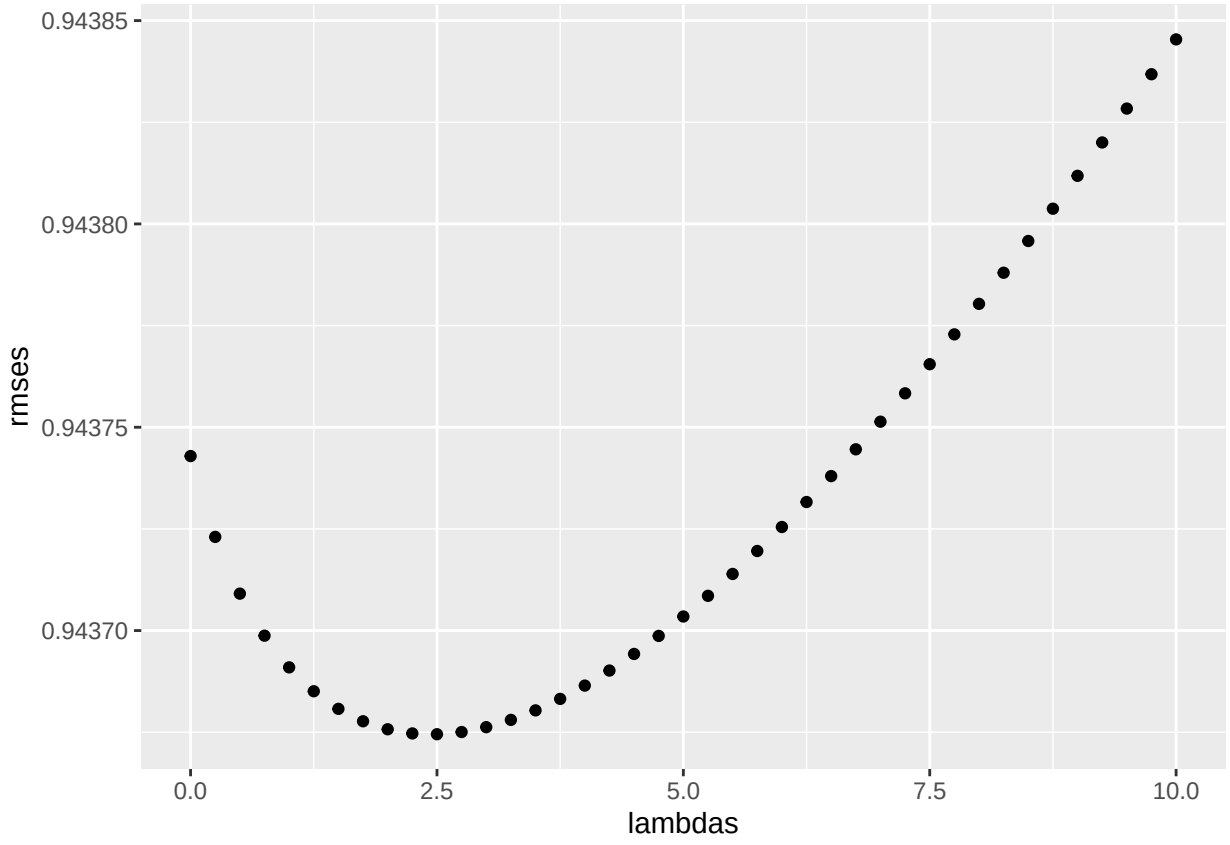
```
## # A tibble: 3 x 2
##   method                     RMSE
##   <chr>                     <dbl>
## 1 Just the average          1.06
## 2 Movie Effect Model        0.944
## 3 Movie + User Effects Model 0.866
```

We can now repeat the process, whilst also accounting for the fact that our dataset is not based on a single user. Our RMSE improves again to about 0.87.

3.5 Model 4 - Regularization Movie Effect

The next step is to use a process known as regularization. This uses penalized regression to control the variability of the effect sizes. To choose the penalty terms using the tuning parameter (lambda), we make use of cross-validation.

```
lambdas <- seq(0, 10, 0.25)
mu <- mean(train_set$rating)
just_the_sum <- train_set %>%
  group_by(movieId) %>%
  summarize(s = sum(rating - mu), n_i = n())
rmses <- sapply(lambdas, function(l){
  predicted_ratings <- test_set %>%
    left_join(just_the_sum, by='movieId') %>% mutate(b_i = s/(n_i+l)) %>%
    mutate(pred = mu + b_i) %>%
    pull(pred)
  return(RMSE(predicted_ratings, test_set$rating))
})
qplot(lambdas, rmses)
```




```
lambdas[which.min(rmses)]
```

```
## [1] 2.5
```

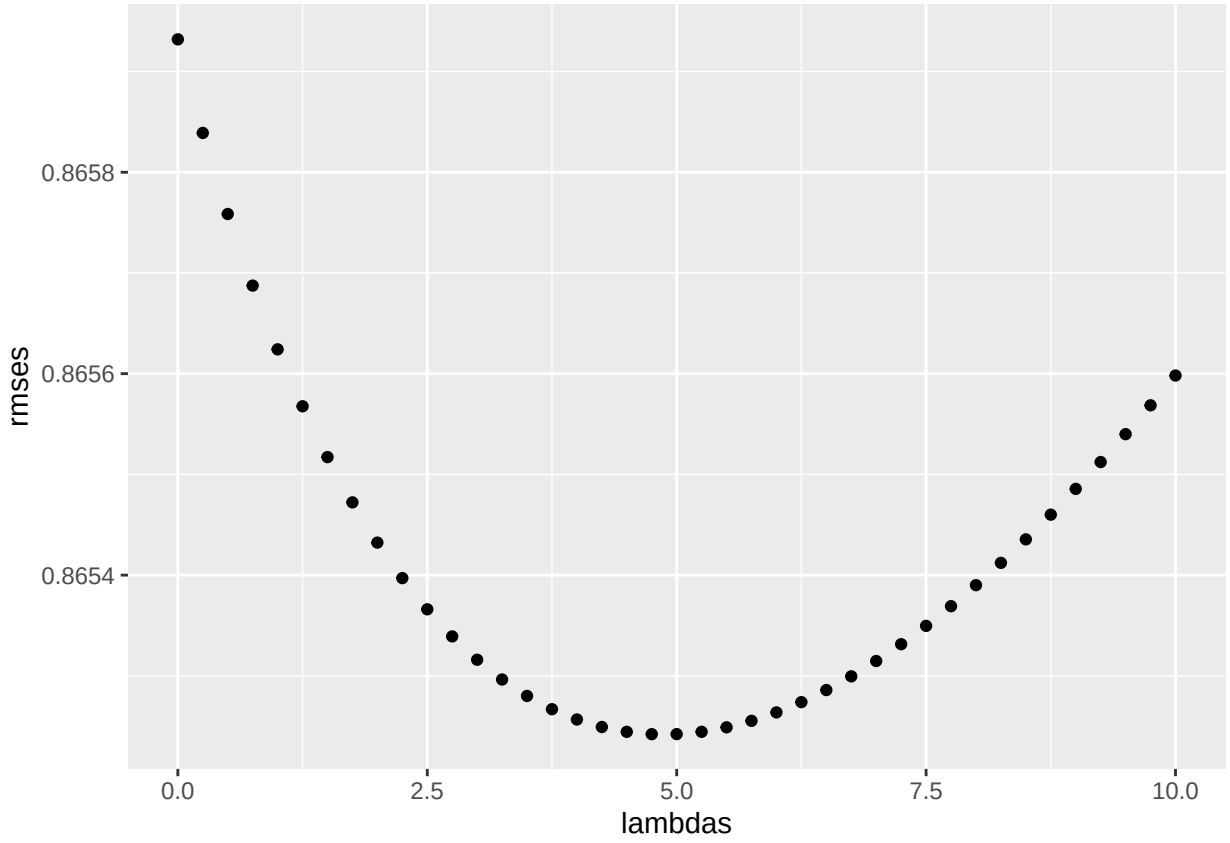
```
rmse_results <- bind_rows(rmse_results,tibble(method="Regularized Movie Effect Model",RMSE = min(rmses)))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0599043
Movie Effect Model	0.9437429
Movie + User Effects Model	0.8659320
Regularized Movie Effect Model	0.9436745

For this model, the labmda that minimises the RMSE is 2.5, giving a final RMSE of approximately 0.94.

3.6 Model 5 - Regularization Movie and User Effect

```
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){
  mu <- mean(train_set$rating)
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  b_u <- train_set %>%
    left_join(b_i, by="movieId") %>% group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  predicted_ratings <-
    test_set %>%
    left_join(b_i, by = "movieId") %>% left_join(b_u, by = "userId") %>% mutate(pred = mu + b_i + b_u) %
  return(RMSE(predicted_ratings, test_set$rating))
})
qplot(lambdas, rmses)
```




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

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

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

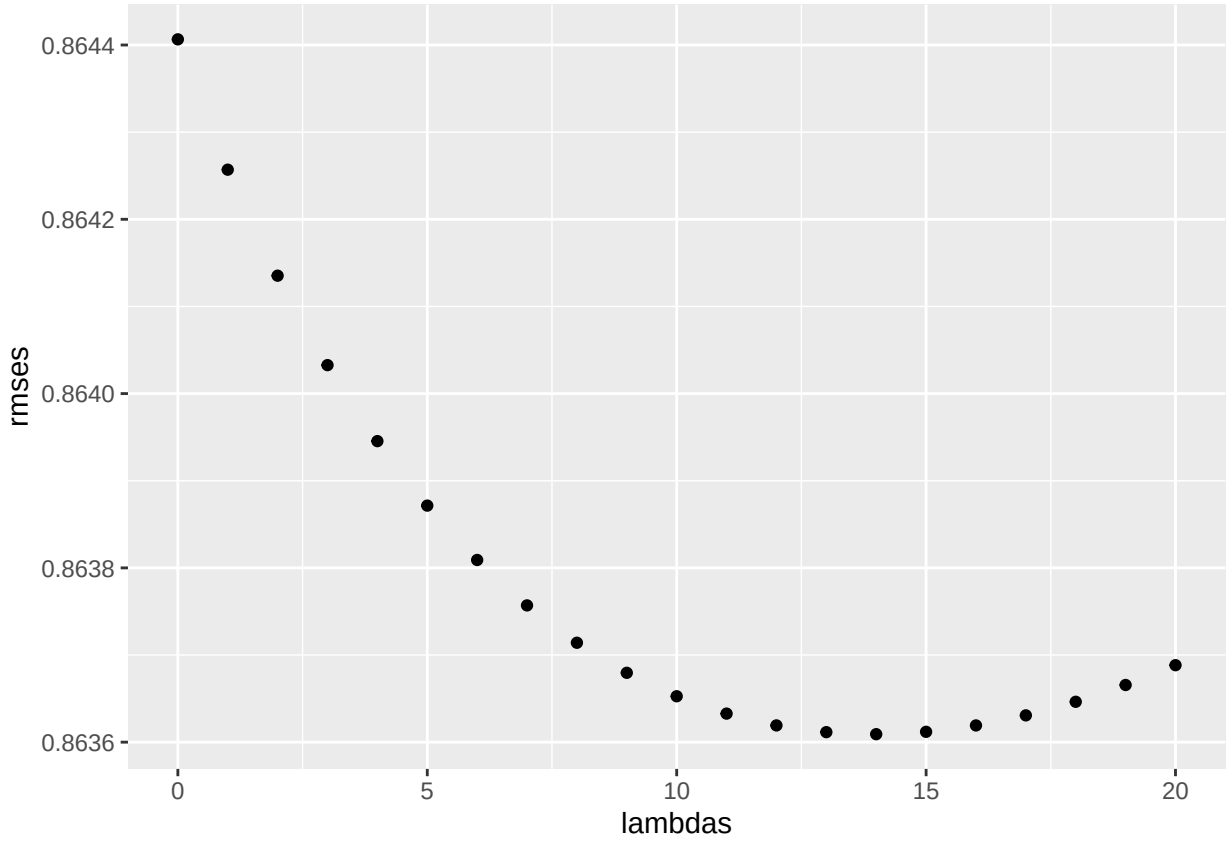
method	RMSE
Just the average	1.0599043
Movie Effect Model	0.9437429
Movie + User Effects Model	0.8659320
Regularized Movie Effect Model	0.9436745
Regularized Movie + User Effect Model	0.8652421

As in Model 3, we want to account for the movie effect AND the user effect. To do this, we can re-use regularization by including both of these parameters in the model. The lambda that minimises the RMSE is 4.75. Our total RMSE has been reduced to 0.87.

3.7 Model 6 - Regularization Movie, User, and Genre Effect

So far, we have looked at the effect of movies and users on the RMSE. Earlier, we noted that ratings also vary considerably with genre. To add this element into the prediction model, we must first split the train and test sets by genre (as previously performed on the `edx` and `validation` sets). Note this process could take a long time.

```
train_genre<-train_set%>%separate_rows(genres,sep = "\\|")
train_genre<-train_genre%>%mutate(date=as.Date(as.POSIXlt(timestamp,origin="1970-01-01",format="%Y-%m-%
train_genre<-train_genre%>%mutate(year=format(date,"%Y"))
train_genre<-train_genre%>%mutate(release_year=as.numeric(str_sub(title,-5,-2)))
test_genre<-test_set%>%separate_rows(genres,sep = "\\|")
test_genre<-test_genre%>%mutate(date=as.Date(as.POSIXlt(timestamp,origin="1970-01-01",format="%Y-%m-%d"
test_genre<-test_genre%>%mutate(year=format(date,"%Y"))
lambdas <- seq(0, 20, 1)
rmses <- sapply(lambdas, function(l){
  mu <- mean(train_set$rating)
  b_i <- train_genre %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  b_u <- train_genre %>%
    left_join(b_i, by="movieId") %>% group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  b_y <- train_genre %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    group_by(year) %>%
    summarize(b_y = sum(rating - b_u - b_i - mu)/(n()+l), n_y = n())
  b_g <- train_genre %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    left_join(b_y, by = "year") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - b_y - b_u - b_i - mu)/(n()+l), n_g = n())
  predicted_ratings <-
    test_genre %>%
    left_join(b_i, by = "movieId") %>% left_join(b_u, by = "userId") %>% left_join(b_y, by = "year") %>%
  return(RMSE(predicted_ratings, test_genre$rating))
})
qplot(lambdas, rmses)
```




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

```
## [1] 14
```

```
rmse_results <- bind_rows(rmse_results,tibble(method="Regularized Movie + User + Genres Effect Model",RM
rmse_results %>% knitr::kable()
```

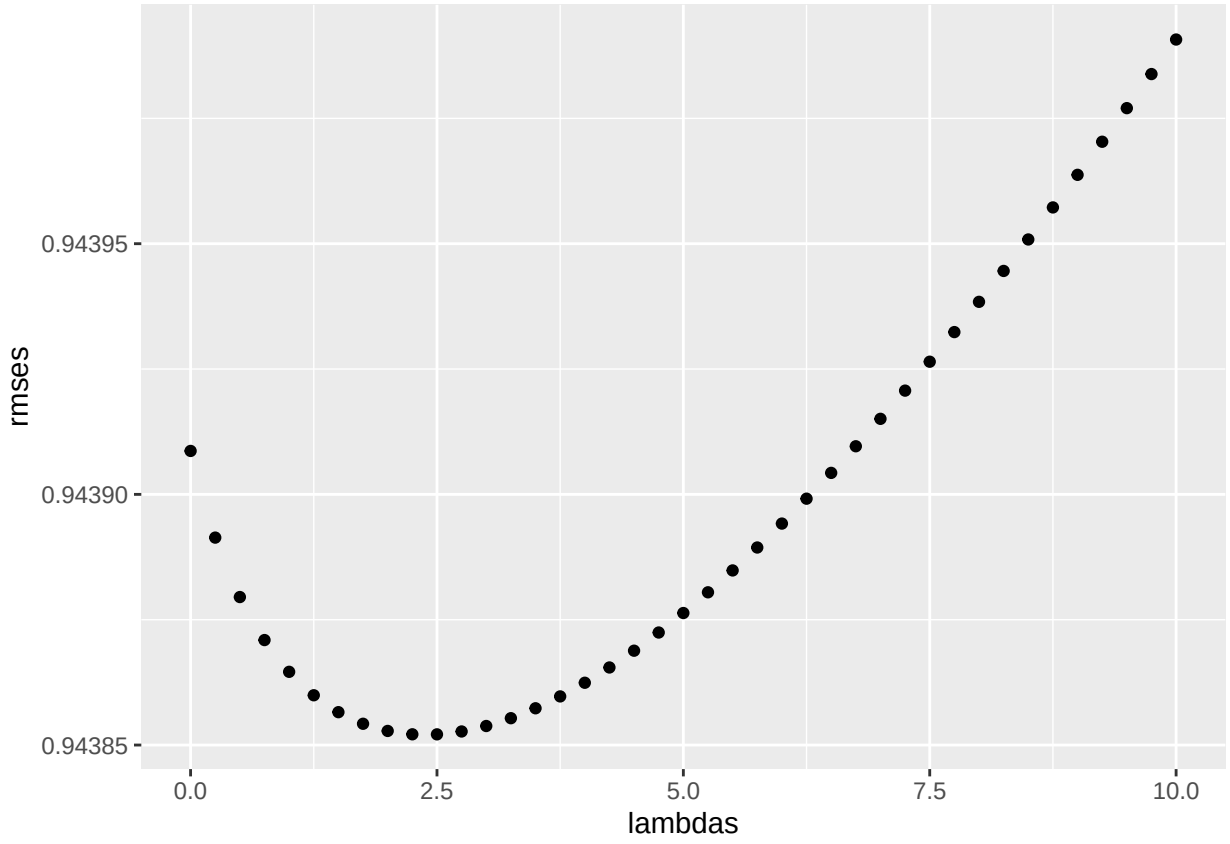
method	RMSE
Just the average	1.0599043
Movie Effect Model	0.9437429
Movie + User Effects Model	0.8659320
Regularized Movie Effect Model	0.9436745
Regularized Movie + User Effect Model	0.8652421
Regularized Movie + User + Genres Effect Model	0.8636092

The optimal lambda is 14; this results in a substantial improvement in our RMSE, down to 0.8636.

3.8 Model 7 - Regularization Movie Effect (Validation)

We are now ready to substitute our train and test sets for the `edx` and `validation` sets.

```
lambdas <- seq(0, 10, 0.25)
mu <- mean(edx$rating)
just_the_sum <- edx %>%
  group_by(movieId) %>%
  summarize(s = sum(rating - mu), n_i = n())
rmses <- sapply(lambdas, function(l){
  predicted_ratings <- validation %>%
    left_join(just_the_sum, by='movieId') %>% mutate(b_i = s/(n_i+l)) %>%
    mutate(pred = mu + b_i) %>%
    pull(pred)
  return(RMSE(predicted_ratings, validation$rating))
})
qplot(lambdas, rmses)
```




```
lambdas[which.min(rmses)]
```

```
## [1] 2.5
```

```
rmse_results <- bind_rows(rmse_results,tibble(method="Regularized Movie Effect Model",RMSE = min(rmses)))
rmse_results %>% knitr::kable()
```

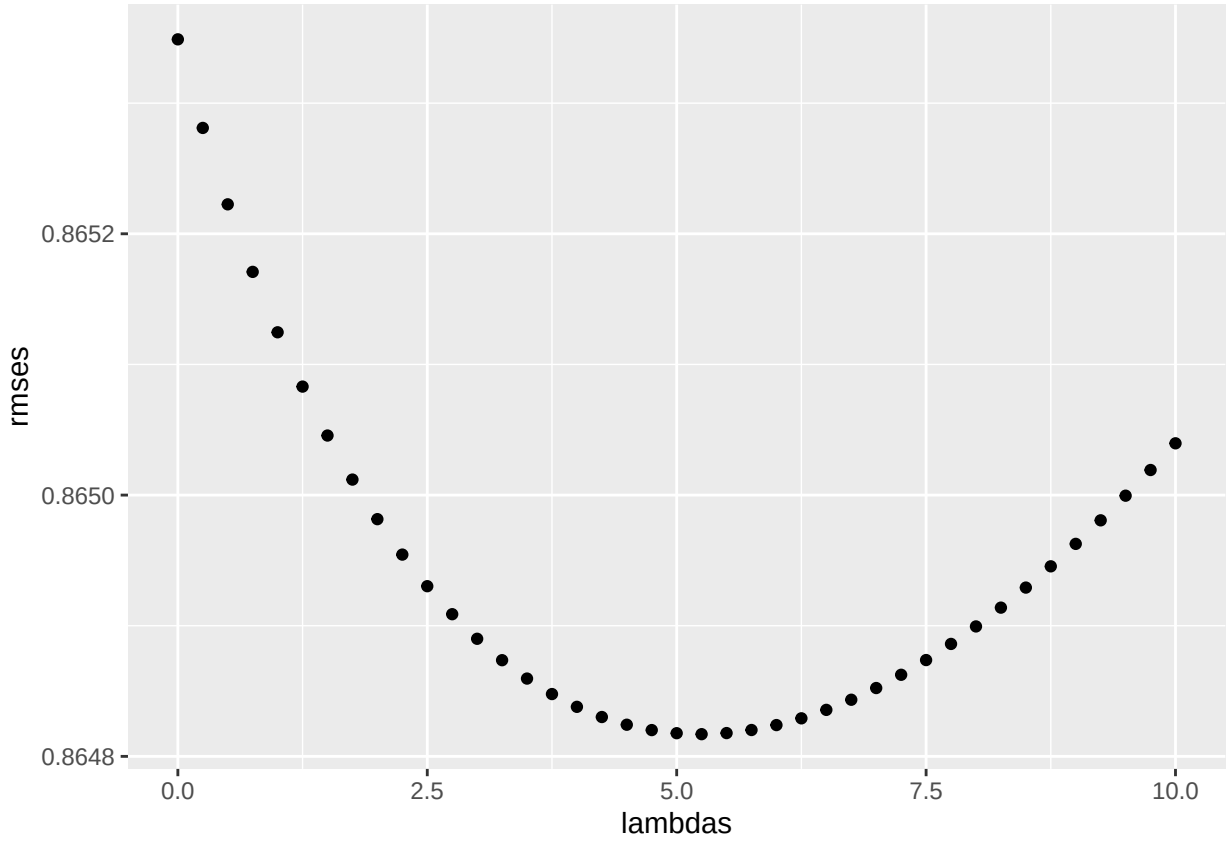
method	RMSE
Just the average	1.0599043

method	RMSE
Movie Effect Model	0.9437429
Movie + User Effects Model	0.8659320
Regularized Movie Effect Model	0.9436745
Regularized Movie + User Effect Model	0.8652421
Regularized Movie + User + Genres Effect Model	0.8636092
Regularized Movie Effect Model	0.9438521

We start by looking at movie effects, it looks like a lambda value of 2.5 gives us the lowest RMSE: about 0.94.

3.9 Model 8 - Regularization Movie and User Effect (Validation)

```
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){
  mu <- mean(edx$rating)
  b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  b_u <- edx %>%
    left_join(b_i, by="movieId") %>% group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  predicted_ratings <-
    validation %>%
    left_join(b_i, by = "movieId") %>% left_join(b_u, by = "userId") %>% mutate(pred = mu + b_i + b_u) %
  return(RMSE(predicted_ratings, validation$rating))
})
qplot(lambdas, rmses)
```




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

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

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

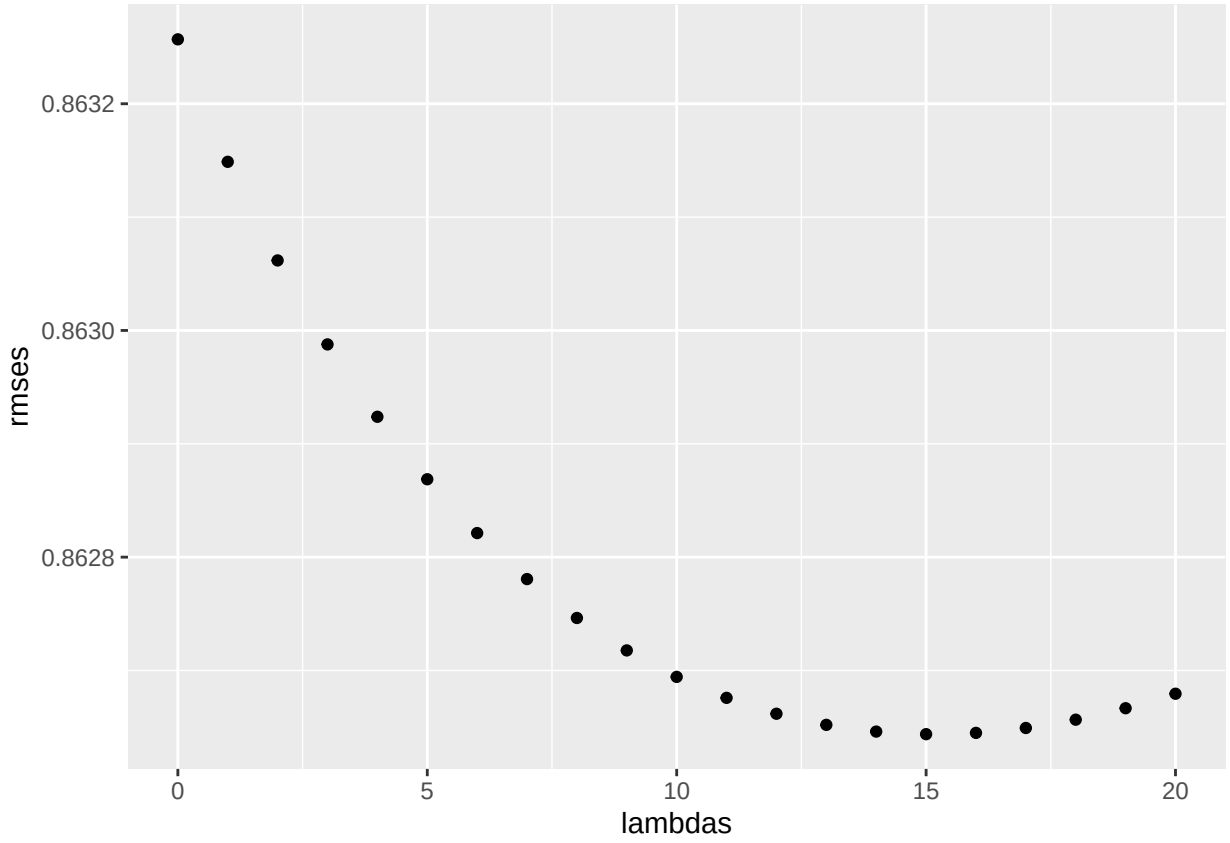
method	RMSE
Just the average	1.0599043
Movie Effect Model	0.9437429
Movie + User Effects Model	0.8659320
Regularized Movie Effect Model	0.9436745
Regularized Movie + User Effect Model	0.8652421
Regularized Movie + User + Genres Effect Model	0.8636092
Regularized Movie Effect Model	0.9438521
Regularized Movie + User Effect Model	0.8648170

The above model makes use of movie and user effects. The lambda that minimises the RMSE is 5.25, giving an RMSE value of 0.8648.

3.10 Model 9 - Regularization Movie, User, and Genre Effect (Validation)

The model that returned the lowest RMSE using the train and test sets included movie, user, and genre effects. Hence, we will re-run this model below with the `validation` set. Please note this process may take some time.

```
lambdas <- seq(0, 20, 1)
rmses <- sapply(lambdas, function(l){
  mu <- mean(edx$rating)
  b_i <- edx_genre %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  b_u <- edx_genre %>%
    left_join(b_i, by="movieId") %>% group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  b_y <- edx_genre %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    group_by(year) %>%
    summarize(b_y = sum(rating - b_u - b_i - mu)/(n()+l), n_y = n())
  b_g <- edx_genre %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    left_join(b_y, by = "year") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - b_y - b_u - b_i - mu)/(n()+l), n_g = n())
  predicted_ratings <-
    validation_genre %>%
    left_join(b_i, by = "movieId") %>% left_join(b_u, by = "userId") %>% left_join(b_y, by = "year") %>%
  return(RMSE(predicted_ratings, validation_genre$rating))
})
qplot(lambdas, rmses)
```




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

```
## [1] 15
```

```
rmse_results <- bind_rows(rmse_results,tibble(method="Regularized Movie + User + Genres Effect Model",RM
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0599043
Movie Effect Model	0.9437429
Movie + User Effects Model	0.8659320
Regularized Movie Effect Model	0.9436745
Regularized Movie + User Effect Model	0.8652421
Regularized Movie + User + Genres Effect Model	0.8636092
Regularized Movie Effect Model	0.9438521
Regularized Movie + User Effect Model	0.8648170
Regularized Movie + User + Genres Effect Model	0.8626438

This final model has a lambda value of 15 and lowers our RMSE all the way to 0.8626!

4. Conclusion

4.1 Summary

Throughout this investigation, we have trialled a number of models to create a recommendation system that can accurately suggest movies for different users, whilst accounting for their rating history and preferences. It was also important to consider factors such as genre and use data visualisation techniques to map ratings over time so as to reflect social and cultural changes which may impact users' viewing choices. The regularization approach provided the most accurate results, when accounting for movie ID, user ID, and genre, as seen in the resulting RMSE.

4.2 Limitations and future work

We could improve our work by trialling new models, for instance using matrix factorization and experimenting with advanced packages. Although there were some constraints on the models that could be used as a result of technical capacity, there remains scope to delve into other machine learning techniques, for instance k-nearest neighbors.